Vantalect

AWS COST · SAMPLE DELIVERABLE

# Rightsizing and Commitments

Week 4 of 6: Rightsizing and commitments · prepared for StrataForge Robotics, Inc.

**Prepared by Charles Sieg · charles.sieg@vantalect.com · vantalect.com**

September 18, 2026 · AWS Cost Analysis & Optimization: The Optimization (six weeks)

**SAMPLE DELIVERABLE**

This is a sample deliverable. StrataForge Robotics, Inc. is a fictional company: every account, resource, finding, figure, and person in this report is simulated to show the form and depth of a Vantalect engagement. AWS prices are list prices from the AWS Price List API on October 13, 2026.

# Contents

**Summary** ..... **3**

**What went in** ..... **5**

Production this week, and what waits ..... 7

**The largest changes** ..... **7**

**How each change was made, and how it is undone** ..... **10**

**What was declined or deferred** ..... **12**

**The commitment decision memo** ..... **15**

Recommendation ..... 15

What it commits you to ..... 17

What if the estate shrinks, or the plans are too small ..... 17

The risk, plainly ..... 18

Approval and purchase ..... 19

The commitment calendar ..... 21

Utilization of what StrataForge holds ..... 22

**The exact mechanism** ..... **23**

Rightsizing and Graviton in the CDK ..... 23

Multi-architecture images ..... 24

Karpenter NodePool ..... 25

Graviton in production: one node group first ..... 27

Aurora and RDS, reader first ..... 28

Aurora I/O-Optimized ..... 30

ElastiCache: Redis OSS to Valkey, then Graviton ..... 30

Kinesis shards sized to peak ..... 31

OpenSearch, MSK, Lambda, DynamoDB, and io2 ..... 32

**What comes next** ..... **33**

# Summary

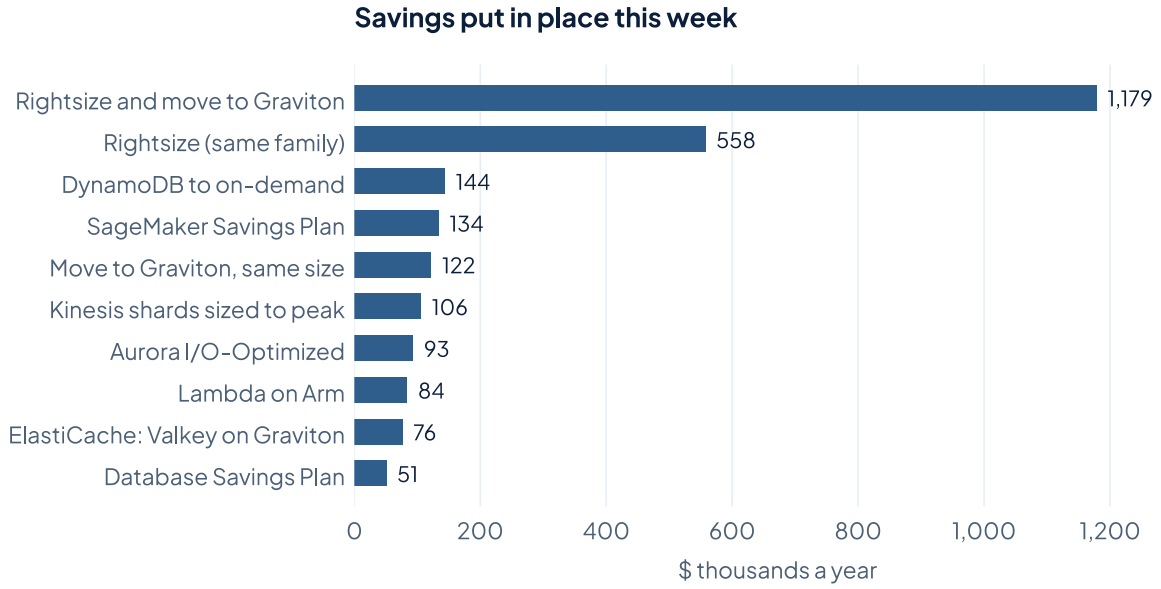
In week 4 StrataForge's engineers put **$2.6 million a year** of rightsizing, Graviton, database, and commitment savings in place, in changes Vantalect drafted and reviewed: 1,232 changes to 1,232 resources and two commitments, both approved by Ravi Menon, Vice President, Finance, and purchased by Elena Vasquez in the management account on September 18, 2026: a one-year Database Savings Plan at **$25 an hour** and a one-year SageMaker Savings Plan at **$61 an hour**. A third decision is queued rather than made: three-year Compute Savings Plans totaling **$740 an hour**, the first tranche starting November 1, 2026, the day after the one-year plan ends. It is worth $2.1 million a year and is the largest single item in the roadmap, but it starts after the measurement window, so it is not in the guarantee's count. It also commits StrataForge to more than it holds today, and The commitment decision memo says plainly how much and what happens if the estate shrinks.

Production moves wait for their windows. The only production instance or node-group move to Graviton in this week's count is a pilot in `fleet-ingest-prod`, on one node group of three, so 33% of that move is in place and the other node groups follow after two weeks of the first running clean. It went in on September 15, 2026. Rightsizing also went in on the shared accounts, which hold build and network infrastructure and are neither production nor non-production: 53 instances in `sf-cicd`, 3 instances in `sf-network-hub`, and 2 instances in `sf-shared-services`. Everything else in production that needs a restart, a failover, or a rebuild is deferred with a date (What was declined or deferred): rightsizing, the rest of Graviton, databases, caches, OpenSearch, and MSK.

As in week 3, every figure is the Analysis's estimate, annualized. Week 6 replaces each with the change on your Cost and Usage Report.

Week 4 in numbers. Savings are estimates until the week-6 measurement.

| Week 4 at a glance | Result |
|---|---|
| **Changes put in place** | 1,232 |
| **Resources changed** | 1,232 |
| **Annualized savings, estimated** | $2,578,344 |
| **Of which the two Savings Plans** | $184,887 |
| **Changes by pull request** | 1,165 |
| **Changes by scripted runbook under a change ticket** | 65 (across 8 tickets) |
| **Week-4 savings not put in place** | $3,776,522 a year |
| **Of which queued, not yet counted: Compute Savings Plans from November 1, 2026** | $2,100,001 a year |




The ten largest savings put in place in week 4, estimated and annualized.

**Decision for the sponsor**

Priya Raman: the Compute Savings Plan purchase must be queued before October 31, 2026, when the existing one-year plan ends, or StrataForge pays on-demand rates on the usage it covers. The decision memo in The commitment decision memo asks Ravi Menon to approve the queue, and it states the obligation up front: the recommended plans commit $19.4 million over three years, against $8.6 million at a time for the one-year alternative, which costs $176,549 a month more. Nothing is bought until the approval, and a queued purchase can be deleted until it starts.

# What went in

Deliverable 9, rightsizing applied, is every row but two in the table below. Deliverable 10, commitments in place, is the Database Savings Plan and SageMaker Savings Plan rows, and the decision memo that follows covers both and the queued purchase. The Database Savings Plan row is partial: the plan is sized to what is in place after this week, 24% of the full-roadmap size, and we count only that share until the rest of the Graviton work lands. Every figure in the table is what is in place now, not what the roadmap will reach.

**Every kind of change put in place in week 4, ranked by estimated annualized savings. Figures are net of the private pricing discount.**

| Kind of change | Changes | Resources | Monthly | Annualized |
|---|---|---|---|---|
| **Rightsize and move to Graviton** | 318 | 318 | $98,223 | $1,178,678 |
| **Rightsize (same family)** | 305 | 305 | $46,469 | $557,629 |
| **DynamoDB to on-demand** | 38 | 38 | $11,981 | $143,768 |
| **SageMaker Savings Plan** | 1 | 1 | $11,164 | $133,969 |
| **Move to Graviton, same size** | 274 | 274 | $10,181 | $122,177 |
| **Kinesis shards sized to peak** | 32 | 32 | $8,821 | $105,858 |
| **Aurora I/O-Optimized** | 8 | 8 | $7,783 | $93,397 |
| **Lambda on Arm** | 192 | 192 | $7,000 | $83,997 |
| **ElastiCache: Valkey on Graviton** | 6 | 6 | $6,366 | $76,388 |
| **Database Savings Plan** | 1 | 1 | $4,243 | $50,918 |

| Kind of change | Changes | Resources | Monthly | Annualized |
|---|---|---|---|---|
| **Databases: rightsize and Graviton** | 46 | 46 | $1,889 | $22,673 |
| **MSK brokers on Graviton** | 2 | 2 | $489 | $5,866 |
| **Move to the current generation** | 9 | 9 | $252 | $3,027 |
| **Total** | **1,232** | **1,232** | **$214,862** | **$2,578,344** |

Graviton moves are the largest block: 58% of the week's estimated value comes from moves to Arm-based Graviton instances. They are also the ones that depend most on your own work: every one needs a container image, an AMI, a Lambda package, or a database engine version that runs on Arm, which is why the deferrals in What was declined or deferred cluster around multi-architecture builds and native dependencies.

### Production this week, and what waits

Production changes went in only where the change is online, in place, and reversible without a restart or a failover: 8 Aurora clusters moved to I/O-Optimized, 38 DynamoDB tables to on-demand capacity, 32 Kinesis streams sized to their peak, 192 Lambda functions on Arm behind a weighted alias, and the one Graviton pilot above. Everything that restarts a server, fails over a database, rolls a cache or a search domain, or rebuilds an image waits for the owner's window: production rightsizing, the rest of Graviton, databases, caches, OpenSearch, and MSK. What was declined or deferred gives each wait and its date.

## The largest changes

**The largest week-4 changes with what each moved from and to. The annualized figure is what is in place this week: the Database Savings Plan row and the pilot rows show only their share.**

| ID | Change | Account | From | To | Owner | Channel | Applied | Annualized |
|---|---|---|---|---|---|---|---|---|
| **C-0001** | Database Savings Plan | sf-management | on-demand Aurora, RDS, ElastiCach… | Database Savings Plan, 1 year, No Upfro… | Elena Vasquez | purchase approved by the VP of Finance | Sep 18 | $50,918 |
| **C-0003** | SageMaker Savings Plan | sf-management | on-demand real-time inference end… | SageMaker Savings Plan, 1 year, No Upfr… | Elena Vasquez | purchase approved by the VP of Finance | Sep 18 | $133,969 |
| **R-1886** | ElastiCache: Valkey on Graviton | fleet-staging | redis cache.r5.xlarge x 6 | valkey cache.r7g.xlarge x 6 | Sam Okafor | fleet-infra PR #8908 | Sep 14 | $20,833 |
| **A-0179** | Aurora I/O-Optimized | fleet-api-prod | Aurora Standard, 12,394 million I… | Aurora I/O-Optimized (no per-I/O charge… | Sam Okafor | fleet-infra PR #9064 | Sep 15 | $20,259 |
| **R-1890** | ElastiCache: Valkey on Graviton | fleet-staging | redis cache.r5.xlarge x 5 | valkey cache.r7g.xlarge x 5 | Sam Okafor | fleet-infra PR #8908 | Sep 16 | $17,361 |
| **A-0175** | Aurora I/O-Optimized | fleet-api-prod | Aurora Standard, 17,764 million I… | Aurora I/O-Optimized (no per-I/O charge… | Sam Okafor | fleet-infra PR #9064 | Sep 18 | $14,553 |
| **A-0176** | Aurora I/O-Optimized | fleet-api-prod | Aurora Standard, 12,625 million I… | Aurora I/O-Optimized (no per-I/O charge… | Sam Okafor | fleet-infra PR #9064 | Sep 14 | $14,284 |

| ID | Change | Account | From | To | Owner | Channel | Applied | Annualized |
|---|---|---|---|---|---|---|---|---|
| **R-1889** | ElastiCache: Valkey on Graviton | fleet-staging | redis cache.r5.xlarge x 4 | valkey cache.r7g.xlarge x 4 | Sam Okafor | fleet-infra PR #8908 | Sep 15 | $13,889 |
| **A-0178** | Aurora I/O-Optimized | fleet-api-prod | Aurora Standard, 16,449 million I… | Aurora I/O-Optimized (no per-I/O charge… | Sam Okafor | fleet-infra PR #9064 | Sep 16 | $11,845 |
| **A-0177** | Aurora I/O-Optimized | fleet-api-prod | Aurora Standard, 13,004 million I… | Aurora I/O-Optimized (no per-I/O charge… | Sam Okafor | fleet-infra PR #9064 | Sep 17 | $11,447 |
| **R-1891** | ElastiCache: Valkey on Graviton | fleet-staging | redis cache.r5.xlarge x 3 | valkey cache.r7g.xlarge x 3 | Sam Okafor | fleet-infra PR #8908 | Sep 17 | $10,416 |
| **R-0333** | Rightsize and move to Graviton | fleet-ingest-prod | c5.9xlarge (Linux) | c7g.4xlarge | Sam Okafor | fleet-infra PR #8836 | Sep 15 | $3,254 |
| **R-0011** | Rightsize (same family) | sf-cicd | c5.9xlarge (Linux) | c5.2xlarge | Elena Vasquez | platform-landing-zone PR #1413 | Sep 15 | $9,695 |
| **R-0301** | Rightsize and move to Graviton | fleet-ingest-prod | c5.9xlarge (Linux) | c7g.4xlarge | Sam Okafor | fleet-infra PR #8833 | Sep 16 | $2,950 |
| **R-0316** | Rightsize and move to Graviton | fleet-ingest-prod | c5.9xlarge (Linux) | c7g.4xlarge | Sam Okafor | fleet-infra PR #8834 | Sep 18 | $2,950 |
| **A-0180** | Aurora I/O-Optimized | fleet-api-prod | Aurora Standard, 14,846 million I… | Aurora I/O-Optimized (no per-I/O charge… | Sam Okafor | fleet-infra PR #9064 | Sep 17 | $8,623 |
| **T-3417** | DynamoDB to on-demand | fleet-api-prod | provisioned 1,496 WCU / 4,397 RCU | on-demand | Sam Okafor | fleet-infra PR #9060 | Sep 16 | $7,968 |
| **A-0181** | Aurora I/O-Optimized | fleet-eu-prod | Aurora Standard, 5,316 million I/… | Aurora I/O-Optimized (no per-I/O charge… | Sam Okafor | fleet-infra PR #9064 | Sep 14 | $7,793 |
| **R-0201** | Rightsize and move to Graviton | fleet-ingest-prod | c5.9xlarge (Linux) | c7g.4xlarge | Sam Okafor | fleet-infra PR #8819 | Sep 18 | $2,580 |
| **R-0219** | Rightsize and move to Graviton | fleet-ingest-prod | c5.9xlarge (Linux) | c7g.4xlarge | Sam Okafor | fleet-infra PR #8821 | Sep 15 | $2,580 |

| ID | Change | Account | From | To | Owner | Channel | Applied | Annualized |
|---|---|---|---|---|---|---|---|---|
| **R-0227** | Rightsize and move to Graviton | fleet-ingest-prod | c5.9xlarge (Linux) | c7g.4xlarge | Sam Okafor | fleet-infra PR #8822 | Sep 17 | $2,580 |
| **R-0233** | Rightsize and move to Graviton | fleet-ingest-prod | c5.9xlarge (Linux) | c7g.4xlarge | Sam Okafor | fleet-infra PR #8823 | Sep 14 | $2,580 |
| **T-3426** | DynamoDB to on-demand | fleet-api-prod | provisioned 1,484 WCU / 4,988 RCU | on-demand | Sam Okafor | fleet-infra PR #9060 | Sep 17 | $7,673 |
| **R-2021** | Lambda on Arm | fleet-api-prod | nodejs20.x, x86_64, 3,008 MB | arm64 | Sam Okafor | fleet-infra PR #8917 | Sep 18 | $7,422 |
| **R-1926** | Kinesis shards sized to peak | fleet-ingest-prod | 60 shards, 8% of write capacity u… | 11 shards (UpdateShardCount, uniform scaling) | Sam Okafor | fleet-infra PR #8911 | Sep 18 | $7,345 |

# How each change was made, and how it is undone

1,165 of the 1,232 changes went through pull requests in the four repositories, and 65 through scripted runbooks under 8 change tickets for the resources no repository defines. The purchase was approved by finance and made in the console, with no code at all.

**Where the week-4 changes were made.**

| Repository or ticket group | Changes |
|---|---|
| `fleet-infra` | 1,079 |
| `platform-landing-zone` | 58 |
| `crucible-infra` | 28 |
| **Scripted runbook, 8 change tickets** | 65 |

Non-production went first. Every production change to a database, cache, or search domain waited for its non-production twin to run clean for at least one full business day. Every change has a rollback, and the rollbacks are not equally cheap, so we say which are not.

**How each kind of week-4 change is reversed and what reversing it costs.**

| Kind of change | Rollback | What it costs you |
|---|---|---|
| **Instance rightsizing within an architecture** | Revert the pull request; CloudFormation stops the instance, changes the type, and starts it | A restart, about the same disruption as the change itself |
| **Move to Graviton** | Revert the type and the image, which replaces a standalone instance again; an Auto Scaling group rolls instead. Container images are multi-architecture, so no rebuild | A replacement: a new root volume and private address for a standalone instance, a rolling update for a group. The AMI must exist for both architectures |
| **Aurora and RDS resizing** | Run the same reader-first script with the old class, which the ledger holds | Two failovers of about 30 seconds each on a cluster with a reader, and a temporary reader for a cluster without one |
| **ElastiCache to Valkey** | Restore the pre-change snapshot to a new Redis OSS cluster. Only Valkey 7.2 to Redis OSS 7.1 rolls back in place | Re-pointing clients to the new endpoint |
| **OpenSearch Graviton and gp3** | `update-domain-config` back to the old type; a blue/green deployment runs again | Hours of blue/green, no downtime |
| **MSK broker type** | `update-broker-type` back to the old type | A rolling restart of the brokers |

| Kind of change | Rollback | What it costs you |
|---|---|---|
| **Lambda on Arm** | Shift the alias back to the x86 version | None; it is a weighted alias |
| **DynamoDB on-demand** | `update-table` back to provisioned, which AWS allows at any time | Recreating the auto scaling targets |
| **io2 IOPS** | Raise `--iops` again | Six hours between modifications of one volume |
| **Aurora I/O-Optimized** | Switch the cluster back to Aurora Standard | AWS limits how often a cluster can change storage configuration and has changed that rule, so read the current one in the Aurora User Guide before planning a reversal |
| **Kinesis shards** | `update-shard-count` back to the old count | Scaling is uniform and bounded (at most double or half in one call), so a large reversal takes several calls |
| **Database Savings Plan** | A purchase cannot be canceled | The commitment runs its year, about $219,000. It is sized to 24% of the full-roadmap level so the hours it covers are hours already running |
| **SageMaker Savings Plan** | A purchase cannot be canceled | The commitment runs its year, about $534,360. It covers the always-on endpoints, which run every hour |

# What was declined or deferred

$3.8 million a year of week-4 items did not go in; $1.7 million of it is not the queued Compute Savings Plan. The 20 largest are below, with each owner's reason as they gave it.

**The largest week-4 items not put in place, with each owner's reason.**

| ID | Change | Owner | Status | Annualized | Reason, as the owner gave it |
|---|---|---|---|---|---|
| **C-0002** | Commitments resized at renewal: Compute Savings Plans | Elena Vasquez | deferred | $2,100,001 | queued purchase; starts November 1, after the measurement window |
| **R-1718** | Rightsize (same family): i-0f267afe0c90c5d4f | Mei Chen | deferred | $32,392 | production resizes go through the owning team's October change windows, one service at a time |
| **R-1880** | ElastiCache: Valkey on Graviton: fleet-api-cache-116fa | Sam Okafor | deferred | $31,748 | production caches upgrade in the October maintenance window |
| **R-1881** | ElastiCache: Valkey on Graviton: fleet-api-cache-a25cb | Sam Okafor | deferred | $27,803 | production caches upgrade in the October maintenance window |
| **R-1879** | ElastiCache: Valkey on Graviton: fleet-api-cache-f2086 | Sam Okafor | deferred | $20,853 | production caches upgrade in the October maintenance window |
| **R-1866** | Databases: rightsize and Graviton: db-perception-api-636ad2 | Jonah Reyes | deferred | $15,699 | production databases resize in the October maintenance window, readers first |
| **R-1894** | OpenSearch: Graviton and gp3: fleet-search-search-ee19 | Lucas Ferreira | deferred | $13,099 | production clusters change by blue/green deployment in the October maintenance window |
| **R-1895** | OpenSearch: Graviton and gp3: fleet-search-search-5994 | Lucas Ferreira | deferred | $11,103 | production clusters change by blue/green deployment in the October maintenance window |
| **R-1892** | OpenSearch: Graviton and gp3: fleet-search-search-8e98 | Lucas Ferreira | deferred | $10,034 | production clusters change by blue/green deployment in the October maintenance window |
| **R-2115** | Lambda on Arm: fleet-api-9b6394 | Sam Okafor | deferred | $8,311 | native dependencies need an arm64 build |

| ID | Change | Owner | Status | Annualized | Reason, as the owner gave it |
|---|---|---|---|---|---|
| **R-1870** | ElastiCache: Valkey on Graviton: fleet-api-cache-516b9 | Sam Okafor | deferred | $7,997 | production caches upgrade in the October maintenance window |
| **R-1877** | ElastiCache: Valkey on Graviton: fleet-api-cache-90035 | Sam Okafor | deferred | $7,997 | production caches upgrade in the October maintenance window |
| **R-1893** | OpenSearch: Graviton and gp3: fleet-search-search-8420 | Lucas Ferreira | deferred | $7,969 | production clusters change by blue/green deployment in the October maintenance window |
| **R-1806** | Databases: rightsize and Graviton: db-fleet-api-8ebcdb | Sam Okafor | deferred | $7,902 | production databases resize in the October maintenance window, readers first |
| **R-1807** | Databases: rightsize and Graviton: db-fleet-api-876197 | Sam Okafor | deferred | $7,902 | production databases resize in the October maintenance window, readers first |
| **R-1876** | ElastiCache: Valkey on Graviton: fleet-api-cache-bc196 | Sam Okafor | deferred | $6,664 | production caches upgrade in the October maintenance window |
| **R-1878** | ElastiCache: Valkey on Graviton: fleet-api-cache-e16bb | Sam Okafor | deferred | $6,664 | production caches upgrade in the October maintenance window |
| **R-1706** | Rightsize (same family): i-09ffafd850f4c7cb7 | Mei Chen | deferred | $6,126 | production resizes go through the owning team's October change windows, one service at a time |
| **R-1707** | Rightsize (same family): i-0965f41a18627ec0f | Mei Chen | deferred | $6,126 | production resizes go through the owning team's October change windows, one service at a time |
| **R-1708** | Rightsize (same family): i-05584531b5cca49c4 | Mei Chen | deferred | $6,126 | production resizes go through the owning team's October change windows, one service at a time |

Nothing here was refused. Every deferral waits on something with a date, and the date is on the register:

**What week 4's deferrals wait on, from the handover register. A wait can include lines from later weeks that share its reason, so the counts here can run above this week's own.**

| Waits on | Target | Changes | Annualized |
|---|---|---|---|
| **the pilot node group's two weeks in production, then the other node groups** | Oct 16 | 284 | $299,064 |
| **the October maintenance window** | Oct 17 | 16 | $137,549 |
| **the October maintenance window, readers first** | Oct 17 | 28 | $66,014 |
| **the owning team's October change window** | Oct 23 | 125 | $216,040 |
| **a blue/green deployment in the October maintenance window** | Oct 24 | 8 | $57,382 |
| **the queued purchase's start date** | Nov 1 | 1 | $2,100,001 |
| **an arm64 build of the function's native dependencies** | Nov 13 | 80 | $42,746 |
| **multi-arch (x86 and Arm) container builds for the service** | Dec 18 | 614 | $1,172,213 |
| **the Aurora reservations ending January 14, 2027, and the Graviton database moves** | Jan 15 | 1 | $164,973 |

- **The Compute Savings Plan is queued on purpose.** Its first tranche starts November 1, 2026, after the measurement window, and the purchase waits on finance's approval of the queue (The commitment decision memo).
- **Production writers, caches, and search domains wait for the October maintenance window.** Readers and non-production databases were resized this week. A cluster's writer needs a failover, a cache node upgrade needs a failover, and an OpenSearch domain needs a blue/green deployment, and the owners chose to take each inside their own window. Resizing goes by cluster, readers first, and only where the target class leaves CPU and memory headroom (Aurora and RDS, reader first).
- **Production Graviton waits on the pilot and on images.** The pilot node group has to run two weeks in production before the other node groups move. Beyond that, the largest group of lines waits on multi-architecture (x86 and Arm) container builds, and some Lambda functions use native dependencies with no Arm build in their pipeline. Both are the owner's work, scheduled in the handover roadmap.

- **Owners' own change windows.** Some owning teams asked for their own October change window for rightsizing. The saving is real, and the window is theirs to choose.

## The commitment decision memo

**The decision**

**To:** Ravi Menon, Vice President, Finance. **Copy:** Priya Raman, Tom Becker, Elena Vasquez.

**Done:** two one-year Savings Plans, No Upfront, approved and bought September 18, 2026: a Database Savings Plan at $25 an hour (a commitment of $219,000 over the year) and a SageMaker Savings Plan at $61 an hour ($534,360).

**Asked:** approve two queued three-year Compute Savings Plans, No Upfront: **$530 an hour** starting November 1, 2026, and **$210 an hour** starting the day after the 2024 plan ends on February 28, 2027, together $740 an hour. That is a commitment of **$19,447,200** over three years ($13,928,400 and $5,518,800), and $150 an hour more than the $590 an hour of Compute Savings Plans StrataForge holds today. A plan cannot be changed once bought, so as the Graviton moves finish the Database Savings Plan grows by a second purchase of $81 an hour, taking the two together to $106.

### Recommendation

Replace the two Compute Savings Plans StrataForge holds, $590 an hour today, with three-year plans totaling $740 an hour, sized to the Linux compute that still runs every hour after the roadmap, and do not renew the one-year plan as it is. Today the two Compute Savings Plans cover about a third of the on-demand Linux pool, and three-year rates are deeper than one-year rates: the roadmap removes waste, not the always-on floor, so a larger three-year commitment on a smaller bill covers about 85% of what remains. The plans cover 33.9% of the on-demand Linux pool today and 84.9% of what is left after the roadmap. Relative to renewing each commitment as it stands, this lowers the post-roadmap bill by $175,000 a month. Relative to letting them lapse, it lowers it by $396,797 a month.

Four ways through the November 1, 2026 renewal, compared on the post-roadmap estate, hour by hour. The last column is the dollars committed whether or not the usage exists.

| Option | Monthly bill after the roadmap, net, before Support | Against the recommendation | Committed over the term |
|---|---|---|---|
| **Recommended: three-year Compute Savings Plans at $740 an hour in two tranches, plus the Database and SageMaker Savings Plans** | $2,513,805 | | $19,447,200 |
| **One-year Compute Savings Plans at the cost-minimizing $985 an hour, renewed each year** | $2,690,354 | +$176,549 | $8,628,600 a year |
| **Renew every existing commitment as it is today** | $2,688,805 | +$175,000 | |
| **Let the Compute Savings Plans lapse** | $2,910,603 | +$396,797 | none |

The comparison is a simulation of every hour of the post-roadmap estate against each option, not a spreadsheet of averages. It is an estimate, and it moves with the roadmap: if less of the roadmap lands, the floor it protects is higher and the case for the commitment is stronger; if more lands, the floor falls and the case weakens. That is why the purchase is in two tranches and the second is dated to the end of the 2024 plan.

The one-year option is the honest alternative, not a straw man. It saves less, $176,549 a month less, or $2.1 million a year, and it commits $8.6 million at a time, renewed each year, instead of $19.4 million over three years. If finance would rather hold less commitment than capture that saving, the one-year plan is a defensible choice. We recommend the three-year plans because the floor they cover is the steady part of the estate, and because the sizing below leaves a margin.

## What it commits you to

**What each commitment costs per year and over its term, at the hourly rate, whether or not StrataForge uses it. A year is 8,760 hours.**

| Commitment | Hourly | A year | Term total | Starts | Covers |
|---|---|---|---|---|---|
| **Database Savings Plan, 1 year, No Upfront (bought)** | $25 | $219,000 | $219,000 | Sep 18 | Aurora, RDS, ElastiCache for Valkey, OpenSearch Service, and DynamoDB on generation 7 and newer |
| **SageMaker Savings Plan, 1 year, No Upfront (bought)** | $61 | $534,360 | $534,360 | Sep 18 | SageMaker real-time inference endpoints, the always-on floor |
| **Compute Savings Plan, 3 years, No Upfront, tranche 1 (queued)** | $530 | $4,642,800 | $13,928,400 | Nov 1 | EC2, Fargate, and Lambda in any Region, family, or size |
| **Compute Savings Plan, 3 years, No Upfront, tranche 2 (queued)** | $210 | $1,839,600 | $5,518,800 | The day after Feb 28 | The same, replacing the 2024 plan when it ends |

The Database Savings Plan applies to the latest instance generations (generation 7 and newer), to Valkey for ElastiCache, and not alongside a reserved instance or reserved DynamoDB capacity on the same usage, which is why it is sized to the Graviton databases the roadmap creates and not to what runs today. AWS lists discounts of up to 35% for serverless and up to 20% for provisioned instances. The SageMaker Savings Plan is sized to the always-on inference endpoints after the idle ones are deleted, which run every hour of the month. The Compute Savings Plans cover EC2, Fargate, and Lambda, and not the databases.

Two break-even figures to hold in mind. A one-year No Upfront Compute Savings Plan costs about 74% of the on-demand rate on the instances this estate runs, and a three-year plan about 51%. A commitment pays for itself whenever the usage it covers runs at least that share of the hours, which is why it must be sized to the floor, the usage that runs every hour, and not to the average.

## What if the estate shrinks, or the plans are too small

Both directions cost money, and the memo should say how much of each. The model ran the estate with the plans at 80% and 90% of their recommended size, and with the on-demand Linux hours falling 20%.

How the three-year Compute Savings Plans behave when they are smaller than recommended, and when usage falls beneath them.

| Case | Compute Savings Plan hourly | Monthly bill after the roadmap, net | Against the recommendation | Utilization of the plans |
|---|---|---|---|---|
| **Recommended size** | $740 | $2,513,805 | | 100.0% |
| **90% of the recommended size** | $665 | $2,533,323 | +$19,518 | 100.0% |
| **80% of the recommended size, which is today's $590 an hour** | $590 | $2,555,061 | +$41,256 | 100.0% |
| **Recommended size, on-demand Linux hours down 20%** | $740 | | | 83.4% |

Too small costs little: at 80% of the size the bill is $41,256 a month higher, which is $495,076 a year. Too large is where the risk is. If the on-demand Linux hours fall 20% below the post-roadmap estate, the three-year plans are used 83.4% of the time, which leaves about $123 an hour paid for and unused, or $1,073,485 a year. The other commitments stay fully used in that case, because the Database, SageMaker, and SAP HANA plans cover usage that does not shrink with the Linux pool. A 20% fall in Linux hours is a stress case, not a forecast: the roadmap removes waste, not the always-on floor the plans are sized to.

Sized on Linux EC2, Aurora, RDS, ElastiCache for Valkey, OpenSearch, and SageMaker endpoints only. Windows, RHEL, and SUSE EC2, Lambda, and Fargate (Compute Savings Plans) and DynamoDB (Database Savings Plans) are eligible too and were left out, so the plans can only be smaller than the knee, never larger: extend them at the March 2027 tranche.

## The risk, plainly

- **Usage below the commitment.** A three-year commitment is paid every hour whether or not the usage exists. If StrataForge moved a large part of its compute to a service the plan does not cover, such as SageMaker, or shut a product down, it would pay for hours it does not use. The stress case above puts a number on a 20% fall. The tranches are sized to the lowest sustained level the post-roadmap model produces, not the average.
- **Graviton and the Compute plan.** Compute Savings Plans apply to Graviton and x86 alike, so moving more workloads to Arm lowers the bill without stranding the commitment. This is the reason to prefer them to instance-family plans here.

- **The Database plan is narrower.** It covers only generation 7 and newer. Databases still on generation 6 when it starts earn nothing from it, which is why it was bought at 24% of the full size. A second plan, bought as the moves finish, brings it to the full size.
- **The SageMaker plan covers one service.** If the always-on endpoints moved to another platform, the plan would be paid for hours it cannot cover. It is a one-year term for that reason.
- **The private pricing commitment.** Savings Plan and reservation fees count toward the spend that StrataForge's private pricing commitment measures (spend net of the private pricing discount, Savings Plans and reservation fees included), so these purchases neither cause nor cure its year-3 shortfall: at today's usage, contract year 3 spend is $31.9 million against a $42.0 million commitment, a true-up of $10.1 million. We recommend not buying usage to meet that commitment, and renegotiating it instead; the handover report carries the options.

## Approval and purchase

Nothing is bought without Ravi Menon's approval, and Vantalect buys nothing. The purchase is made by Elena Vasquez's team in the management account (`sf-management`), where a Savings Plan applies across every account in the organization by default. The steps are the same for all three: read the recommendation, find the offering, purchase at the approved hourly rate, and confirm.

```bash
# 1. Read AWS's own recommendation to compare with ours (Database plan shown; use SAGEMAKER_SP
or COMPUTE_SP for the others).
aws ce get-savings-plans-purchase-recommendation \
  --savings-plans-type DATABASE_SP --term-in-years ONE_YEAR \
  --payment-option NO_UPFRONT --lookback-period-in-days SIXTY_DAYS

# 2. Find the offering for the plan, term, and payment option.
aws savingsplans describe-savings-plans-offerings \
  --plan-types Database --durations 31536000 --payment-options "No Upfront" --currencies USD

# 3. Buy the Database Savings Plan at the approved hourly commitment.
aws savingsplans create-savings-plan \
  --savings-plan-offering-id <database-offering-id> \
  --commitment 25 --client-token <uuid>
```

The SageMaker Savings Plan is the same three steps with its own plan type and hourly commitment.

```bash
aws savingsplans describe-savings-plans-offerings \
  --plan-types SageMaker --durations 31536000 --payment-options "No Upfront" --currencies USD

aws savingsplans create-savings-plan \
  --savings-plan-offering-id <sagemaker-offering-id> \
  --commitment 61 --client-token <uuid>
```

The Compute Savings Plans are queued with a purchase time, so they start the minute the old plan ends and not before. A queued purchase is not a commitment until it starts, and it can be deleted until then.

```bash
aws savingsplans describe-savings-plans-offerings \
  --plan-types Compute --durations 94608000 --payment-options "No Upfront" --currencies USD

aws savingsplans create-savings-plan \
  --savings-plan-offering-id <compute-offering-id> \
  --commitment 530 --purchase-time 2026-11-01T00:00:00Z --client-token <uuid>

aws savingsplans describe-savings-plans --states queued active
aws savingsplans delete-queued-savings-plan --savings-plan-id <queued-plan-id>
```

Record every purchase, with the approver and the date, in the commitment register that the monthly cost review keeps (see the handover report).

## The commitment calendar

**Every commitment date from now to the last reservation. Each is an action, not only a reminder.**

| Date | Event | Why |
|---|---|---|
| **September 18, 2026** | Buy a one-year Database Savings Plan at $25 an hour | covers the Graviton databases, Valkey caches, and OpenSearch domains in place after week four |
| **September 18, 2026** | Buy a one-year SageMaker Savings Plan at $61 an hour | the always-on inference endpoints, which run every hour |
| **October 31, 2026** | One-year Compute Savings Plan ($380 an hour) ends | do not renew it as it is |
| **November 1, 2026** | Queued three-year Compute Savings Plan starts at $530 an hour | sized to the post-roadmap floor, net of the 2024 plan |
| **November 26, 2026** | EKS upgrades any Kubernetes 1.31 control plane still running | extended support for 1.31 ends; upgrade on your own schedule before then |
| **November 30, 2026** | Twelve r5.8xlarge reservations end | the EMR fleet moves to Graviton in December |
| **January 14, 2027** | The Aurora reservations end | move those instances to Graviton generation 7, then buy a second one-year Database Savings Plan of $81 an hour (a plan cannot be changed once bought), for $106 an hour in all |
| **February 28, 2027** | Three-year Compute Savings Plan from 2024 ($210 an hour) ends | replace with a three-year plan at $210 an hour, queued for March 1 |
| **May 31, 2027** | SAP HANA reservations (2 x u-6tb1.112xlarge) end | renew for three years; the HANA hosts are steady |

## Utilization of what StrataForge holds

Every existing commitment is fully used against the estate in place at the end of this week, which is what makes renewing them as they are the wrong default: the post-roadmap estate is smaller in the places they cover.

**The five commitments in place before the engagement, measured against the estate as it stands after week 4.**

| Commitment | Type | Term and payment | Ends | Commitment | Utilization | Unused | Covers, on-demand | Savings a month | Fee a month |
|---|---|---|---|---|---|---|---|---|---|
| `sp-3yr-2024` | Compute Savings Plan | 3yr, No Upfront | February 28, 2027 | $210 an hour | 100% | $0 | $314,144 | $160,844 | $153,300 |
| `sp-1yr-2025` | Compute Savings Plan | 1yr, No Upfront | October 31, 2026 | $380 an hour | 100% | $0 | $377,154 | $99,754 | $277,400 |
| `ri-hana` | EC2 reservation | 3yr, Partial Upfront | May 31, 2027 | 2 x u-6tb1.112xlarge | 100% | $0 | $79,898 | $56,104 | $23,794 |
| `ri-r5-8xl` | EC2 reservation | 1yr, No Upfront | November 30, 2026 | 12 x r5.8xlarge | 100% | $0 | $17,660 | $6,535 | $11,125 |
| `ri-aurora` | RDS reservation | 1yr, No Upfront | January 14, 2027 | 6 x db.r5.4xlarge | 100% | $0 | $10,162 | $3,548 | $6,614 |

The commitments StrataForge will hold once the recommendation is in place are used just as fully against the post-roadmap estate, and the stress case shows which of them would slip:

**Utilization of the commitments after the recommendation, in the post-roadmap model and in the stress case.**

| Commitment after the recommendation | Utilization, post-roadmap | Utilization, Linux hours down 20% |
|---|---|---|
| **SAP HANA reservations, renewed** | 100.0% | 100.0% |
| **Three-year Compute Savings Plans** | 100.0% | 83.4% |
| **Database Savings Plan** | 100.0% | 100.0% |
| **SageMaker Savings Plan** | 100.0% | 100.0% |

# The exact mechanism

The code is what we used, reduced to what you need to run it again. Every infrastructure change below is made in code in the repository that owns the resource, and only resources in no repository get a scripted runbook.

## Rightsizing and Graviton in the CDK

An instance defined in `fleet-infra`, `crucible-infra`, or `enterprise-infra` changes with one property. A type change inside one architecture (a smaller size, or a newer x86 family) is an in-place update: CloudFormation stops the instance, changes the type, and starts it, so the instance keeps its volumes and its private address. A move to Graviton is not. It needs an Arm image, the `ImageId` property changes, and CloudFormation replaces an instance when `ImageId` changes, with a new root volume and a new private address. So the pull request for an x86-to-Arm move carries a `cdk diff` that shows the instance replaced, the owner approves it as a replacement, and anything the instance holds lives on separate volumes or is rebuilt from code. Resolve the image once with `cachedInContext: true`, or the next Amazon Linux release replaces the instance on an unrelated deploy.

```typescript
// enterprise-infra/lib/mes-app-stack.ts
const app = new ec2.Instance(this, 'MesApp1', {
  vpc,
  instanceType: ec2.InstanceType.of(ec2.InstanceClass.M7G, ec2.InstanceSize.XLARGE2), // was
R5, XLARGE4
  machineImage: ec2.MachineImage.latestAmazonLinux2023({ cpuType:
ec2.AmazonLinuxCpuType.ARM_64, cachedInContext: true }), // x86 to Arm: a replacement
  blockDevices: [{ deviceName: '/dev/xvda', volume: ec2.BlockDeviceVolume.ebs(100,
{ volumeType: ec2.EbsDeviceVolumeType.GP3 }) }],
});
```

For an Auto Scaling group, the same change goes in the launch template, with a rolling update that keeps the full desired count in service, so a replacement arrives before an old instance leaves. The maximum size must sit above the desired count to leave room for the extra instance.

```typescript
new autoscaling.AutoScalingGroup(this, 'Workers', {
  vpc,
  instanceType: ec2.InstanceType.of(ec2.InstanceClass.M7G, ec2.InstanceSize.XLARGE2),
  machineImage: ec2.MachineImage.latestAmazonLinux2023({ cpuType:
ec2.AmazonLinuxCpuType.ARM_64 }),
  minCapacity: 3,
  maxCapacity: 6,
  updatePolicy: autoscaling.UpdatePolicy.rollingUpdate({ minInstancesInService: 3,
maxBatchSize: 1 }),
});
```

Windows and SQL Server instances (a large part of the enterprise estate) stay on x86, so they resize within their family and are not Graviton candidates. Look first at `Compute Optimizer`'s finding for the instance: it names the Arm option with its migration effort and the platform differences.

## Multi-architecture images

An Arm node cannot run an x86 image. The container work comes first and is why the largest deferrals wait on it. Build each architecture on a native runner (no emulation, so no slow builds), then publish one manifest that points at both. The same tag then runs on whichever node Karpenter picks. Pin the actions to commit SHAs in your repository.

```yaml
# .github/workflows/image.yml
name: image
on: { push: { branches: [main] } }
permissions: { id-token: write, contents: read }
env:
  REPOSITORY: <account-id>.dkr.ecr.<region>.amazonaws.com/<repository>
jobs:
  build:
```

```
    strategy:
      matrix:
        include:
          - { arch: amd64, runner: X64 }
          - { arch: arm64, runner: ARM64 }
    runs-on: [self-hosted, linux, "${{ matrix.runner }}"]
    steps:
      - uses: actions/checkout@v4
      - uses: aws-actions/configure-aws-credentials@v4
        with: { role-to-assume: "<ci-role-arn>", aws-region: "<region>" }
      - uses: aws-actions/amazon-ecr-login@v2
      - uses: docker/setup-buildx-action@v3
      - uses: docker/build-push-action@v6
        with:
          context: .
          push: true
          provenance: false
          platforms: linux/${{ matrix.arch }}
          tags: ${{ env.REPOSITORY }}:sha-${{ github.sha }}-${{ matrix.arch }}
  manifest:
    needs: build
    runs-on: [self-hosted, linux, X64]
    steps:
      - uses: aws-actions/configure-aws-credentials@v4
        with: { role-to-assume: "<ci-role-arn>", aws-region: "<region>" }
      - uses: aws-actions/amazon-ecr-login@v2
      - run: |
          docker buildx imagetools create -t "$REPOSITORY:sha-${{ github.sha }}" \
            "$REPOSITORY:sha-${{ github.sha }}-amd64" "$REPOSITORY:sha-${{ github.sha }}-arm64"
```

Verify that the tag carries both architectures before any Arm node pulls it, and keep the ECR lifecycle policy from week 3 in mind. All three tags (`sha-<git-sha>` on the manifest, and the `-amd64` and `-arm64` ones) match its `sha-*` rule and age out after 30 days. ECR does not expire an image that a manifest list still references, so the per-architecture images go only after the manifest tag does. A release tag starting with `v` goes on the manifest and keeps it out of that rule.

```bash
docker buildx imagetools inspect <account-id>.dkr.ecr.<region>.amazonaws.com/<repository>:<tag>
\
  | grep -E "Platform:"
```

## Karpenter NodePool

On EKS the nodes follow the images, but Karpenter cannot see which architectures an image supports. A single NodePool that allows both would put an x86-only pod on an Arm node, where it crashes on start. So the Arm capacity is its own NodePool, tainted so that nothing lands on it by accident. Only a workload whose image is verified multi-architecture carries the toleration, and Karpenter then chooses the cheaper node for it, which is Graviton. The existing `general` pool stays x86 only. Consolidation removes the nodes the rightsized pods no longer need.

YAML

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: general
spec:
  template:
    spec:
      nodeClassRef: { group: karpenter.k8s.aws, kind: EC2NodeClass, name: default }
      requirements:
        - { key: kubernetes.io/arch, operator: In, values: ["amd64"] }
        - { key: karpenter.sh/capacity-type, operator: In, values: ["on-demand"] }
        - { key: karpenter.k8s.aws/instance-category, operator: In, values: ["c", "m", "r"] }
        - { key: karpenter.k8s.aws/instance-generation, operator: Gt, values: ["5"] }
      expireAfter: 720h
  limits:
    cpu: "4000"
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 5m
    budgets:
      - nodes: "10%"
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: general-arm
spec:
  template:
    spec:
      nodeClassRef: { group: karpenter.k8s.aws, kind: EC2NodeClass, name: default }
      taints:
        - { key: arch, value: arm64, effect: NoSchedule }   # only verified workloads tolerate
this
      requirements:
        - { key: kubernetes.io/arch, operator: In, values: ["arm64"] }
        - { key: karpenter.sh/capacity-type, operator: In, values: ["on-demand"] }
        - { key: karpenter.k8s.aws/instance-category, operator: In, values: ["c", "m", "r"] }
        - { key: karpenter.k8s.aws/instance-generation, operator: Gt, values: ["5"] }
      expireAfter: 720h
  limits:
    cpu: "2000"
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 5m
    budgets:
      - nodes: "10%"
---
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: default
spec:
  role: <node-role-name>
  amiSelectorTerms:
    - alias: al2023@<release-version>   # pinned: a new AMI release is a change you make, not
```

```
one that arrives on its own
  subnetSelectorTerms:
    - tags: { karpenter.sh/discovery: <cluster-name> }
  securityGroupSelectorTerms:
    - tags: { karpenter.sh/discovery: <cluster-name> }
  blockDeviceMappings:
    - deviceName: /dev/xvda
      ebs: { volumeSize: 100Gi, volumeType: gp3, encrypted: true }
```

A workload opts in to Arm with a toleration in its pod template. Without a `nodeSelector` it can run on either pool and Karpenter picks the cheaper node.

```yaml
tolerations:
  - { key: arch, operator: Equal, value: arm64, effect: NoSchedule }
```

Roll it out with the toleration and a `nodeSelector` of `kubernetes.io/arch: arm64` on one workload in a non-production cluster first, then remove the selector. Rightsize the pods' requests in the same pull request: Karpenter sizes nodes to requests, so oversized requests keep oversized nodes. To roll back, remove the toleration; Karpenter replaces the Arm nodes with x86 ones as pods reschedule, and the Arm pool can then be deleted.

## Graviton in production: one node group first

In production the move starts with one node group, not the whole service. The pilot is `fleet-ingest-prod`: one of three node groups changes from c5.9xlarge (Linux) to c7g.4xlarge, and the other node groups stay as they are for two weeks. The service runs on both architectures during that time, which is the point: a regression shows up in 33% of the traffic, and the rollback is changing one node group back.

```typescript
// fleet-infra/lib/ingest-nodegroups.ts: the pilot group moves; the others do not
cluster.addNodegroupCapacity('IngestA', {
  amiType: eks.NodegroupAmiType.AL2023_ARM_64_STANDARD,
  instanceTypes: [new ec2.InstanceType('<arm-instance-type>')],   // the pilot
  minSize: 2, maxSize: 12,
});
cluster.addNodegroupCapacity('IngestB', {
  amiType: eks.NodegroupAmiType.AL2023_X86_64_STANDARD,
  instanceTypes: [new ec2.InstanceType('<x86-instance-type>')],   // unchanged
  minSize: 2, maxSize: 12,
});
```

Check the placement and the service's own error rate and latency on both architectures for the two weeks, then move the other node groups in a second pull request. The register dates that second step October 16, 2026.

```bash
kubectl get nodes -L kubernetes.io/arch,eks.amazonaws.com/nodegroup
kubectl top pods -n <namespace> --containers
```

## Aurora and RDS, reader first

A database resizes only when the target class leaves headroom, and it resizes by cluster, never one instance at a time in isolation. The evidence for a cluster is its 95th-percentile CPU projected onto the target class's vCPUs and its lowest freeable memory, over 14 days: for `db-perception-api-636ad2` the Analysis read p95 CPU 21%; working set 24% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 131 connections on average over 14 days, so mysql db.r5.4×large Multi-AZ became db.r7g.2×large. A cluster that does not leave room on both measures is not resized.

```bash
aws cloudwatch get-metric-statistics --namespace AWS/RDS --metric-name CPUUtilization \
  --dimensions Name=DBInstanceIdentifier,Value=<instance> --period 3600 --extended-statistics
p95 \
  --start-time <14-days-ago> --end-time <now>
aws cloudwatch get-metric-statistics --namespace AWS/RDS --metric-name FreeableMemory \
  --dimensions Name=DBInstanceIdentifier,Value=<instance> --period 3600 --statistics Minimum \
  --start-time <14-days-ago> --end-time <now>
```

A writer resize is a failover; a reader resize is not. So the order is always readers first, then one failover to a resized reader, then the old writer. Clusters with only a writer get a temporary reader of the target class first. Check that the target class is orderable for the engine version before you start.

```bash
aws rds describe-orderable-db-instance-options --engine aurora-postgresql --engine-version
<version> \
  --db-instance-class db.r7g.4xlarge --query 'OrderableDBInstanceOptions[].DBInstanceClass'
```

```bash
#!/usr/bin/env bash
# platform-landing-zone/finops/runbooks/aurora-resize.sh <cluster-id> <target-class> [--
execute]
set -euo pipefail
cluster="$1"; target="$2"; MODE="dry-run"; [[ "${3:-}" == "--execute" ]] && MODE="execute"
run() { if [[ "$MODE" == "execute" ]]; then "$@"; else echo "DRY-RUN: $*" >&2; fi; }

wait_until() {                       # wait_until <what> <test command...>: poll up to 3 hours,
then stop the run
  [[ "$MODE" == "execute" ]] || return 0
  local what="$1" n; shift
  for n in $(seq 1 360); do "$@" && return 0; sleep 30; done
```

```
  echo "FAIL: timed out waiting for $what; nothing further was changed" >&2; exit 1
}
resized() {                          # the instance has the target class and is available
  [[ "$(aws rds describe-db-instances --db-instance-identifier "$1" \
        --query 'DBInstances[0].[DBInstanceClass,DBInstanceStatus]' --output text)" ==
"$target"$'\t'available ]]
}
is_writer() {                        # the instance is the cluster's writer
  [[ "$(aws rds describe-db-clusters --db-cluster-identifier "$cluster" \
        --query "DBClusters[0].DBClusterMembers[?DBInstanceIdentifier=='$1'].IsClusterWriter |
[0]" --output text)" == "True" ]]
}

members=$(aws rds describe-db-clusters --db-cluster-identifier "$cluster" \
  --query 'DBClusters[0].DBClusterMembers[].[DBInstanceIdentifier,IsClusterWriter]' --output
text)
writer=$(awk '$2 == "True" {print $1}' <<<"$members")
readers=$(awk '$2 == "False" {print $1}' <<<"$members")
engine=$(aws rds describe-db-clusters --db-cluster-identifier "$cluster" --query
'DBClusters[0].Engine' --output text)
echo "$members" | while read -r id _; do                # the ledger: every instance and its
class before the change
  aws rds describe-db-instances --db-instance-identifier "$id" \
    --query 'DBInstances[0].[DBInstanceIdentifier,DBInstanceClass]' --output text >> aurora-
ledger.tsv
done

temp=""
if [[ -z "$readers" ]]; then                       # writer only: add a temporary reader of the
target class
  temp="${cluster}-resize"; readers="$temp"
  run aws rds create-db-instance --db-cluster-identifier "$cluster" \
      --db-instance-identifier "$temp" --db-instance-class "$target" --engine "$engine"
  wait_until "$temp" resized "$temp"
else
  for r in $readers; do                            # one reader at a time; the others keep
serving
    run aws rds modify-db-instance --db-instance-identifier "$r" --db-instance-class "$target"
--apply-immediately
    wait_until "$r" resized "$r"
  done
fi

first=$(head -n1 <<<"$readers")                    # one reader takes over, never every reader
run aws rds failover-db-cluster --db-cluster-identifier "$cluster" --target-db-instance-
identifier "$first"
wait_until "$first to become the writer" is_writer "$first"
run aws rds modify-db-instance --db-instance-identifier "$writer" --db-instance-class "$target"
--apply-immediately
wait_until "$writer" resized "$writer"

if [[ -n "$temp" ]]; then                          # hand the cluster back to the original
instance, then remove the temporary one
  run aws rds failover-db-cluster --db-cluster-identifier "$cluster" --target-db-instance-
identifier "$writer"
  wait_until "$writer to become the writer again" is_writer "$writer"
```

```
  run aws rds delete-db-instance --db-instance-identifier "$temp"
fi
```

The script checks every step by the instance's own class and status, not by a waiter that can return before the change starts, and it stops with nothing further changed if a step does not finish in three hours. The ledger file holds the old class of every instance; the rollback is the same script with that class. Verify with `aws rds describe-db-clusters` and the `CPUUtilization` and `DatabaseConnections` metrics for a day. For Multi-AZ RDS (not Aurora) instances, `modify-db-instance --db-instance-class` resizes the standby first and fails over for you.

## Aurora I/O-Optimized

Aurora Standard bills every I/O; I/O-Optimized bills none, and charges more for instances and storage. It pays off when I/O is a large share of a cluster's bill: for `cl-a9ac96ece1` the evidence was "I/O is 58% of the cluster's cost; I/O-Optimized pays off above about 25%". The change is made in place, with no failover. AWS allows the move to I/O-Optimized once every 30 days and the move back to Standard at any time; instances on classes with local NVMe storage restart during the change, and others change with no downtime. It went into production this week on 8 clusters.

```bash
aws rds modify-db-cluster --db-cluster-identifier <cluster> --storage-type aurora-iopt1 --
apply-immediately
aws rds describe-db-clusters --db-cluster-identifier <cluster> --query
'DBClusters[0].StorageType'
```

In `fleet-infra` the same change is `storageType: rds.DBClusterStorageType.AURORA_IOPT1` on the cluster. Roll back with `--storage-type aurora`, within the current limit on switches.

## ElastiCache: Redis OSS to Valkey, then Graviton

Valkey nodes cost 20% less than Redis OSS on the same size, and Redis OSS 4 and 5 are paying an Extended Support surcharge of 80% in years 1 and 2 and 160% in year 3 since February 2026. The engine upgrade is in place, and endpoints stay the same; node IP addresses change, and a failover of about 30 to 60 seconds is expected. Take a snapshot first, because the only in-place rollback is Valkey 7.2 to Redis OSS 7.1.

```bash
aws elasticache create-snapshot --replication-group-id <id> --snapshot-name pre-valkey-<ticket>
until [ "$(aws elasticache describe-snapshots --snapshot-name pre-valkey-<ticket> \
    --query 'Snapshots[0].SnapshotStatus' --output text)" = available ]; do sleep 30; done
aws elasticache modify-replication-group --replication-group-id <id> \
  --engine valkey --engine-version 9.0 --apply-immediately
aws elasticache wait replication-group-available --replication-group-id <id>
aws elasticache modify-replication-group --replication-group-id <id> \
  --cache-node-type cache.r7g.2xlarge --apply-immediately
aws elasticache wait replication-group-available --replication-group-id <id>
aws elasticache describe-replication-groups --replication-group-id <id> \
  --query 'ReplicationGroups[0].[Status,CacheNodeType,Engine]'
```

If a custom parameter group is attached, add `--cache-parameter-group-name` with a Valkey group that keeps the same static parameter values. A single-node Redis OSS cluster is first added to a replication group. Requires AWS CLI v2 2.18.2 or later. In `fleet-infra`, the same change is `engine: 'valkey'` and `cacheNodeType` on the `CfnReplicationGroup`.

## Kinesis shards sized to peak

A provisioned stream bills per shard-hour whether or not data flows. Size each stream to its peak, not its average: take the 99th percentile of `IncomingBytes` and `IncomingRecords` per second over 14 days, divide by a shard's write limit (1 MiB and 1,000 records a second), and add headroom. One-minute data lasts 15 days and a `get-metric-statistics` call returns at most 1,440 points, so the query uses `get-metric-data` on one-minute sums and divides by 60 before taking the percentile; repeat it with `IncomingRecords`. For `robot-alerts-51281` the stream had 60 shards, 8% of write capacity used at the 99th percentile, and the evidence was IncomingBytes and IncomingRecords at the 99th percentile over 14 days, so it became 11 shards (UpdateShardCount, uniform scaling).

```bash
aws cloudwatch get-metric-data --start-time <14-days-ago> --end-time <now> --metric-data-
queries '[
  {"Id": "bytes", "MetricStat": {"Metric": {"Namespace": "AWS/Kinesis", "MetricName":
"IncomingBytes",
    "Dimensions": [{"Name": "StreamName", "Value": "<stream>"}]}, "Period": 60, "Stat": "Sum"},
"ReturnData": false},
  {"Id": "bytes_per_second", "Expression": "bytes / 60", "ReturnData": true}]' \
  --query 'MetricDataResults[0].Values' --output json | jq 'sort | .[(length * 0.99 | floor)]'
aws kinesis update-shard-count --stream-name <stream> --target-shard-count <peak-sized-count>
--scaling-type UNIFORM_SCALING
```

A single call can halve a stream, not more, so a large reduction takes several calls; rolling back is the same call in the other direction. Producers that retry on `ProvisionedThroughputExceededException` absorb a brief squeeze, so make the change in a quiet hour and watch that metric for a day.

## OpenSearch, MSK, Lambda, DynamoDB, and io2

**OpenSearch Service** domains change instance type and volume type in one call; a blue/green deployment moves the data without downtime and takes hours, not minutes.

```bash
aws opensearch update-domain-config --domain-name <domain> \
  --cluster-config InstanceType=r7g.4xlarge.search,InstanceCount=<count> \
  --ebs-options EBSEnabled=true,VolumeType=gp3,VolumeSize=<gb>,Iops=3000,Throughput=125
aws opensearch describe-domain-change-progress --domain-name <domain>
```

**MSK** brokers move to Graviton with one call, which restarts the brokers one at a time. It is safe only with a replication factor of 3 and `min.insync.replicas` of 2 across three Availability Zones; check both first.

```bash
aws kafka describe-cluster-v2 --cluster-arn <cluster-arn> --query 'ClusterInfo.CurrentVersion'
--output text
aws kafka update-broker-type --cluster-arn <cluster-arn> --current-version <current-version> \
  --target-instance-type kafka.m7g.xlarge
```

**Lambda** functions move to Arm in the CDK with one property, and by CLI with a package built for arm64. Roll out through a weighted alias: send 10% to the Arm version, watch errors and duration for a day, then shift fully. Native dependencies are the usual blocker; build them for the target platform.

```typescript
new lambda.Function(this, 'Normalize', {
  runtime: lambda.Runtime.PYTHON_3_13,
  architecture: lambda.Architecture.ARM_64,
  handler: 'app.handler',
  code: lambda.Code.fromAsset('dist/normalize-arm64.zip'),
});
```

```bash
pip install --platform manylinux2014_aarch64 --python-version 3.13 --only-binary=:all: --target
package -r requirements.txt
aws lambda update-function-code --function-name <function> --architectures arm64 --zip-file
fileb://function-arm64.zip --publish
aws lambda update-alias --function-name <function> --name live \
  --function-version <old-version> --routing-config 'AdditionalVersionWeights={"<new-
version>"=0.1}'
```

**DynamoDB** tables with spiky or low traffic move to on-demand, at $0.625 per million write request units and $0.125 per million read request units since the November 2024 cut. Compare the table's consumed capacity with what it provisions over 30 days before switching, and remove the old auto scaling targets after. AWS allows four switches to on-demand in any rolling 24 hours and a switch back to provisioned at any time, so a one-table trial is cheap. On-demand throughput is also covered by the Database Savings Plan, at up to 18%.

```bash
aws dynamodb update-table --table-name <table> --billing-mode PAY_PER_REQUEST
for dim in WriteCapacityUnits ReadCapacityUnits; do
  aws application-autoscaling deregister-scalable-target --service-namespace dynamodb \
    --resource-id table/<table> --scalable-dimension dynamodb:table:$dim
  aws application-autoscaling deregister-scalable-target --service-namespace dynamodb \
    --resource-id table/<table>/index/<index> --scalable-dimension dynamodb:index:$dim   # one
pair per global secondary index
done
```

**io2** volumes are billed for the IOPS provisioned, not used: $0.065 per IOPS-month up to 32,000, then $0.0455. Provision to the measured peak plus headroom. The change is online.

```bash
aws ec2 modify-volume --volume-id <volume-id> --iops <measured-peak-plus-headroom>
```

## What comes next

Week 5 takes on architecture: VPC endpoints and NAT charges, Kafka consumers that read inside their own Availability Zone, telemetry through Firehose, metrics, CloudTrail and Config scope, and Spot for simulation and CI (Deliverable 11), then the guardrails that keep the bill down (Deliverable 13). The Compute Savings Plan queue needs Ravi Menon's answer before October 31, 2026.

© 2026 Vantalect, LLC. Sample deliverable; StrataForge Robotics is fictional.

AWS, Amazon Web Services, AWS Graviton, and the AWS service names are trademarks of Amazon.com, Inc. or its affiliates. Kubernetes is a registered trademark of The Linux Foundation. GitHub is a trademark of GitHub, Inc. NVIDIA and Isaac Sim are trademarks of NVIDIA Corporation. SAP and SAP S/4HANA are trademarks of SAP SE. Vantalect is not affiliated with or endorsed by any of them. StrataForge Robotics is fictional.