Vantalect

AWS COST · SAMPLE DELIVERABLE

Cost Analysis and Savings Roadmap

Week 2 of 6: Find and rank · prepared for StrataForge Robotics, Inc.

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

September 4, 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

Executive summary **4**
Why a capable team missed it 5
Tiers, and the range we plan on 5
What is in the number, by category and by wave 8
What we need from you 9
Method **9**
The ordered stack 10
Tiers and the planning range 10
What counts toward the private pricing floor 11
Prices 11
Thresholds 11
Effort, risk, and confidence 12
Where each change is made 13
Deliverable 2: the waste register **13**
The 25 largest findings 14
How every removal is done 17
Runbooks, one per kind of waste 17
Deliverable 3: the rightsizing plan **28**
The rules 29
What is in the plan 29
The 25 largest compute changes 31
Databases 33
Graviton readiness 37
Examples: where the change goes 39
How a rightsizing change ships, and how it rolls back 43
Deliverable 4: the commitment plan **43**
What StrataForge holds today 44
How the sizes were found 44
What each plan commits StrataForge to 46
The recommendation, and its alternatives 46
The calendar 48

How to queue a purchase ... 49
The private pricing floor ... 50
Deliverable 5: the architecture review ... 53
The 30 largest changes ... 55
The network path: NAT gateways and VPC endpoints ... 60
Kafka: let consumers read in their own Availability Zone ... 65
Service calls: keep traffic in its zone ... 68
Robot telemetry: write to S3 through Firehose, not to CloudWatch Logs ... 69
Per-robot metrics: embedded metric format, not one metric per robot ... 70
Audit and logging scope: CloudTrail data events and AWS Config ... 72
Logs and metrics ... 75
S3: lifecycle rules and Intelligent-Tiering by data class ... 76
Over-the-air updates: serve them through CloudFront ... 84
Spot capacity: simulation and CI ... 86
Services priced by storage, throughput, or I/O ... 88
Version upgrades that stop an Extended Support charge ... 91
Smaller changes, with their commands ... 94
Deliverable 6: the savings roadmap ... 98
The waves ... 98
By kind ... 99
The 30 largest opportunities ... 102
Effort, risk, and owners ... 105
One-time costs and payback ... 105
The order that matters ... 106
The guarantee ... 107
Appendix A: the registers ... 107
Appendix B: assumptions ... 108
Appendix C: price provenance ... 109
Appendix D: AWS facts this report relies on ... 110

Executive summary

We found **$23.5 million a year** of savings across 13,020 opportunities on 73,592 resources, and we would plan on **$17.2 million to $21.6 million** of it. Carried out in full, the opportunities take the monthly bill from **$4.6 million to $2.6 million**, a reduction of 43.5%, and the annual bill by $24.2 million once the knock-on effect on Support is counted. These are estimates from your own usage and AWS list prices, net of the 7% private pricing discount. They become measured savings only in week 6, change by change, on your Cost and Usage Report.

Identified is not the same as realizable, and the gap is a contract. StrataForge's private pricing agreement has a spend floor. At today's usage the roadmap takes spend in contract year 3 to $31.9 million, which is $10.1 million under the $42.0 million commitment, and we assume AWS bills the difference as a true-up (we have not read the contract). So at today's usage **$13.7 million** of the identified savings reaches StrataForge in that year. Renegotiating the commitment to $31.0 million closes the gap and makes it $23.8 million (The private pricing floor).

The identified savings beside what StrataForge can realize. The third row depends on usage staying flat, and the last row depends on the private pricing renegotiation.

Measure	Per year	What it is
Identified	$23,520,211	The sum of all 13,020 opportunities, each priced net of the private pricing discount
Planning range	$17,239,885 to $21,617,305	The identified figure by tier, at the realization rate we assume for each (Tiers, and the range we plan on)
Realizable in contract year 3, usage flat	$13,715,103	The reduction in year-3 spend, $23,791,047, less the $10,075,944 true-up under the commitment
Realizable once the commitment is renegotiated	$23,791,047	The same year, with the commitment reset to $31,000,000 and the discount held at 7%; it is above the identified figure for the reasons below the table

Why the last row is above the identified figure. It measures the reduction in contract-year-3 spend, not the sum of the opportunities, and the two differ by three things: $705,742 for the fall in Enterprise Support, which is a percentage of the bill and shrinks with it; $4,519 for the coverage effect; and $439,425 less, because some lines start after contract year 3 begins. The bridge is in The private pricing floor. This report plans the roadmap as scheduled, so none of that is a decline; later reports replace the plan with the owners' decisions.

Three things shape how to read the number:

1. **Nothing is counted twice.** The savings are an ordered stack. Waste is removed first, then rightsizing, then schedules, then storage and architecture changes, and only then are commitments sized to what is left (Method). A resource can appear in more than one line, such as an instance that is rightsized and then scheduled, and each line is priced on the state the earlier ones leave. The waterfall below adds up to the after-roadmap bill exactly.
2. **Not all of it is the same kind of work.** $10.0 million (42%) is hygiene: 10,071 scripted, low-risk changes that need only an owner's sign-off. $9.5 million needs build, test, and a change window, and $1.6 million waits on a decision, a firmware release, or a vendor (Tiers, and the range we plan on).
3. **The private pricing floor is the number that surprises.** The roadmap is still the right call, and the time to renegotiate is now, before the savings reach the bill (The private pricing floor).

Why a capable team missed it

Three things outran a capable FinOps team: the Kestrel acquisition added five accounts with never-expire defaults in November 2025, GPU training spend tripled in a year, and platform defaults (no log retention, no lifecycle rules, gp2 launch templates) created waste faster than quarterly reviews could find it.

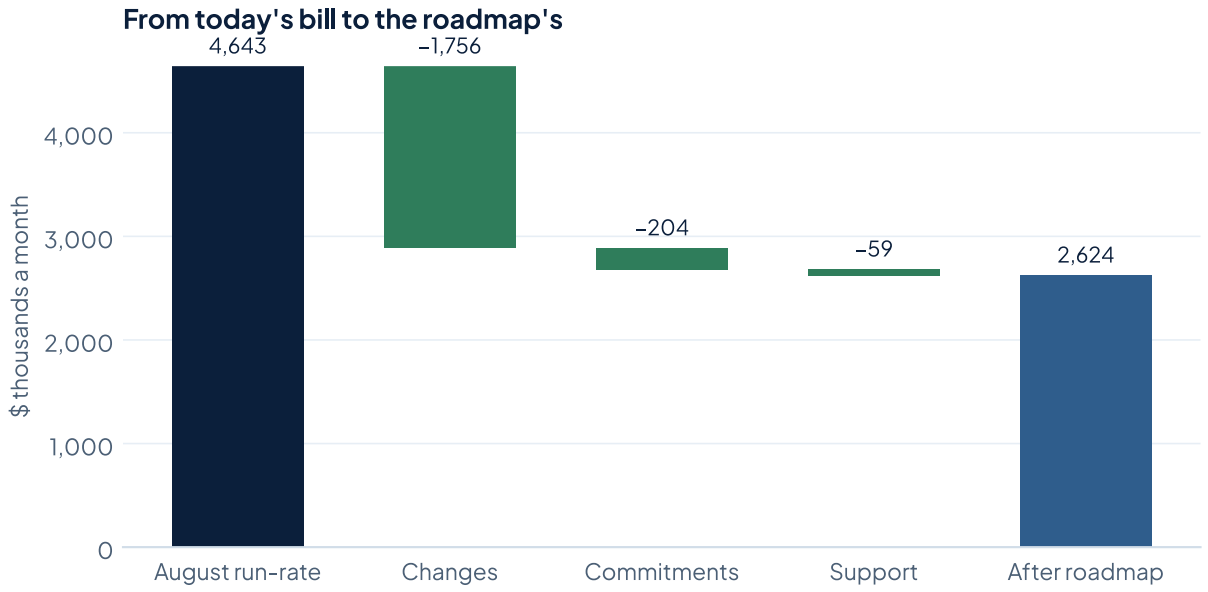
Tiers, and the range we plan on

Opportunities differ in how likely they are to land in full. We sort them into four tiers and apply a realization rate to each: the share of the identified savings we expect to see on the bill. **The rates are planning assumptions, not measurements.** Week 6 replaces them with a measured figure for every change.

The four tiers, the realization rate assumed for each, and the planning range they give.

Tier	Opportunities	Identified per year	Realization assumed	Planning range per year
Hygiene: scripted, low risk, owner sign-off only	10,071	$9,957,050	90% to 100%	$8,961,345 to $9,957,050
Engineering: needs build, test, and a change window	2,879	$9,540,560	60% to 85%	$5,724,336 to $8,109,476
Decision or dependency: needs an owner decision, firmware, or a vendor	67	$1,572,740	30% to 70%	$471,822 to $1,100,918
Commitments: finance approval, sized to what remains	3	$2,449,861	85% to 100%	$2,082,382 to $2,449,861
Total	**13,020**	**$23,520,211**		**$17,239,885 to $21,617,305**

The hygiene tier is nearly certain because the changes are scripted and reversible. The decision tier is the least certain: it holds the changes that need robot firmware 7.4, a data-governance review, or a sponsor's approval, and some of those will not be approved as written. The commitments tier is the firmest after hygiene, because it is a purchase finance approves and not a change engineers build.



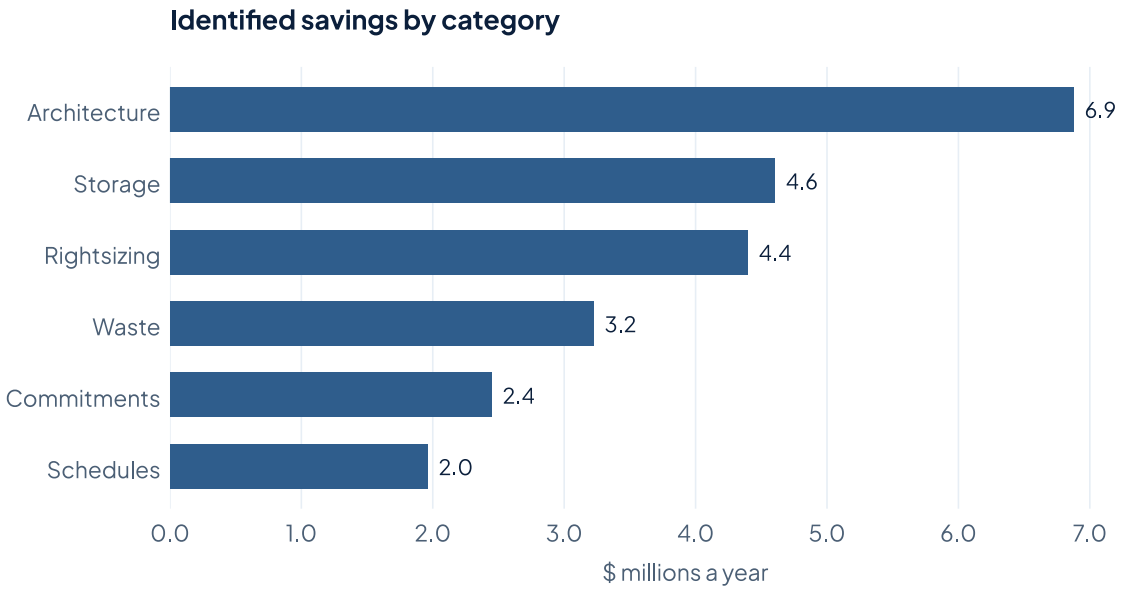
The reconciliation: from today's bill to the roadmap's, in thousands of dollars a month.

The reconciliation, line by line. The last row is the sum of the four lines above it.

Step	Change to the monthly bill	Running total
August 2026 run-rate, net		$4,642,925
Changes: waste, schedules, rightsizing, storage, and architecture	−$1,755,863	$2,887,063
Commitments: Database, SageMaker, and Compute Savings Plans, resized as they renew	−$204,155	$2,682,908
Support: Enterprise Support is a percentage of the bill	−$58,812	$2,624,096
Coverage effect: re-pricing the whole estate once, not line by line	−$377	$2,623,719
After the roadmap	**−$2,019,206**	**$2,623,719**

The "coverage effect" is the measured overlap between the lines: where two changes touch the same resource, adding up each opportunity's estimate counts a little too much or too little, and re-pricing the whole estate hour by hour after every change shows how much. It is $377 a month here, and we show it so that the table reconciles to the cent. It is measured, never a balancing figure. Each commitment line is a measured step too: the estate is priced with today's commitments, then with the Database Savings Plan added, then with the SageMaker Savings Plan, then with the commitments resized, and the difference at each step is that line's saving (Deliverable 4: the commitment plan).

What is in the number, by category and by wave



Identified savings by category, millions of dollars a year.

Identified savings by category. One-time costs are lifecycle transition requests and archive snapshots.

Category	Opportunities	Resources	Per month	Per year	One-time cost
Architecture	820	820	$573,258	$6,879,096	$0
Storage	3,447	3,447	$383,690	$4,604,276	$115,614
Rightsizing	2,205	2,205	$366,507	$4,398,082	$0
Waste	6,060	66,309	$268,835	$3,226,015	$13,198
Commitments	3	3	$204,155	$2,449,861	$0
Schedules	485	808	$163,573	$1,962,882	$0
Total	**13,020**	**73,592**	**$1,960,018**	**$23,520,211**	**$128,812**

Savings by wave. The backlog waits on a date or a dependency that falls outside the six weeks.

Wave	When	Opportunities	Per month	Per year
Week 3	Week of September 7, 2026	9,955	$806,971	$9,683,651
Week 4	Week of September 14, 2026	2,094	$568,242	$6,818,903
Week 5	Week of September 21, 2026	473	$368,037	$4,416,439
Backlog	After October 2, 2026	498	$216,768	$2,601,218

What we need from you

Five decisions come out of this report. The sponsor readout (deliverable 7, delivered with this report) puts them in front of Priya Raman and Ravi Menon.

1. **Approve the roadmap and its four waves,** with every change still approved by the owner of its workload.
2. **Approve the commitment plan:** a one-year Database Savings Plan at $25 an hour (and a second one for $81 an hour when the Aurora reservations end) and a one-year SageMaker Savings Plan at $61 an hour in week four, and a queued three-year Compute Savings Plan of $530 an hour starting November 1, 2026, with a second tranche of $210 an hour in March 2027. The three-year tranches commit $19.4 million over their term (Deliverable 4: the commitment plan).
3. **Approve Deep Archive** as the storage class for the disaster-recovery copy of raw telemetry. Any object comes back within 48 hours, and a restore of the whole copy has a price and a duration, in Replicating straight into Deep Archive.
4. **Send the data-lake retention changes to the data-governance council** before anyone is asked to approve them, and SAP backup retention to the SOX controls owner (S3: lifecycle rules and Intelligent-Tiering by data class).
5. **Start the private pricing renegotiation now,** before the savings reach the bill: at today's usage the roadmap leaves contract year 3 $10.1 million under the commitment (The private pricing floor).

Method

The numbers in this report come from one model of your estate, rebuilt from the resource inventory and the Cost and Usage Report, and priced hour by hour. This section says how it is built, so that a finance reviewer or your own engineers can check it.

The ordered stack

Each opportunity is a change to one resource or a group of resources. A resource can appear in more than one opportunity, such as an instance that is rightsized and then scheduled, or a database that is rightsized, upgraded, and scheduled. Each step is priced on the estate as the steps before it leave it, so the later line is worth what remains after the earlier one, and the lines add up without counting the same dollar twice:

1. **Waste is removed.** Idle, orphaned, and unused resources go to zero.
2. **Compute and databases are rightsized,** and moved to Graviton where the workload allows it, using the capacity that is left.
3. **Schedules apply.** Non-production instances and databases run only outside the hours nobody uses them, priced at the size step 2 leaves.
4. **Storage and architecture change.** Lifecycle policies, endpoints, logging scope, and data-path changes are priced on the volumes that remain.
5. **Commitments are sized to what is left, one measured step at a time.** The estate after steps 1 to 4 is priced with today's commitments held as they are. Then the Database Savings Plan is added and the saving is measured, then the SageMaker Savings Plan, then the Compute Savings Plans are resized as the old ones end. Each commitment line in the roadmap is the difference between two of those prices, never a remainder.
6. **Support follows the bill.** AWS Enterprise Support is the greater of $5,000 or 10% of monthly charges up to $150,000, 7% from $150,000 to $500,000, 5% from $500,000 to $1 million, and 3% above $1 million, so it falls as the bill does.
7. **The coverage effect** is the measured difference between the sum of the parts and the re-priced whole. It is small, and it is measured, not plugged.

A snapshot deleted in step 1 is therefore never also tiered in step 4. An instance rightsized in step 2 and scheduled in step 3 is scheduled at its new size, and a database rightsized in step 2 is covered at its new size in step 5, never at its old one.

Tiers and the planning range

Every opportunity belongs to one of four tiers by who has to act and what could stop it: **hygiene** (scripted, low risk, owner sign-off only), **engineering** (build, test, and a change window), **decision** (an owner decision, a firmware release, or a vendor), and **commitments** (finance approval). Each tier carries a realization rate, the share of its identified savings we expect to reach the bill. The rates are planning assumptions, and the report says so wherever it uses the range (Tiers, and the range we plan on). They are not a forecast, and week 6 replaces them with measurements.

What counts toward the private pricing floor

The private pricing agreement commits StrataForge to a minimum annual spend. We assume the commitment counts spend net of the private pricing discount, Savings Plans and reservation fees included, and that 96.4% of the bill is eligible for the discount. Contract-year spend is projected two ways, at today's usage and at the pace of the last six months, and the realizable savings in this report are the reduction in spend net of any true-up under the floor (The private pricing floor). We have not read the contract, so each of those assumptions is a question for Ravi Menon's team.

Prices

Every price is a list price from the AWS Price List API (`get-products`, us-east-1 endpoint). For this sample, all of them were retrieved on October 13, 2026; a live report carries the date of its own analysis. Savings Plans rates come from `describe-savings-plans-offering-rates` and a 30-day Spot price history from `describe-spot-price-history`. Savings are then shown **net of the 7% private pricing discount**: an opportunity worth $40,179 a month at list is worth $37,367 a month to StrataForge. Where we quote an AWS price in a recommendation, it is the us-east-1 price on the retrieval date. The appendix lists every price source (Appendix C: price provenance).

Thresholds

The rules that turn measurements into findings.

Finding	Rule
Idle instance	95th-percentile CPU under 4%, average CPU under 2%, and under 1 GB of network a month over 30 days; GPU utilization under 2% where there is a GPU
Long-stopped instance	Stopped for more than 30 days, with volumes still billed
Unattached volume	Detached for more than 30 days
Orphaned snapshot or AMI	Source volume deleted, no AMI or backup policy references it, and the newest snapshot is more than 90 days old
Idle load balancer	No healthy targets and no requests for 30 days
Unassociated Elastic IP	Unassociated for 30 days or more
Idle non-production database	Zero connections and 95th-percentile CPU under 3.5% for 14 days; deletion keeps a final snapshot for 90 days
Idle notebook or Studio app	No kernel activity for 21 days or more

Finding	Rule
Rightsizing, production	One size down when p95 CPU is at or under 30% and p95 memory at or under 45% over 14 days
Rightsizing, non-production	One size down when p95 CPU is at or under 40% and p95 memory at or under 50%; two sizes when p95 CPU at or under 20% and p95 memory at or under 35%
Databases	Resized only with CPU and memory headroom measured, one cluster at a time, readers before the writer
Rightsizing without memory data	One step only when p95 cpu is at or under 15% (production) or 25% (non-production), and labeled medium confidence
Schedules	Non-production capacity with no use outside 7 a.m. to 7 p.m. local time on weekdays runs 260 of 730 hours a month; a performance-test node group scales to zero outside its test windows and runs 100
Excluded from rightsizing	GPU hosts, SAP HANA hosts, Spot instances, and instances inside a capacity reservation
S3 lifecycle	Object age from S3 Inventory (share of bytes older than 90 and 365 days) and reads from server access logs; retrieval charges on objects still read after they move are counted against the saving

Effort, risk, and confidence

- **Effort S:** one owner, one change, a day or less. **M:** a few days with testing and a pull request in one repository. **L:** needs another team's release, such as robot firmware, or a database migration.
- **Risk Low:** the change can be reversed in minutes and affects nothing customers use. **Medium:** it touches production or is hard to reverse, and needs a rollback rehearsed first. Every commitment other than the three-year Compute plan is Medium, because a Savings Plan cannot be returned once it starts, and so is every S3 line that expires objects or noncurrent versions, because an expired version is gone. **High:** a commitment that cannot be returned and runs for years. The only opportunity rated High is the three-year Compute Savings Plan (C-0002).
- **Confidence** applies to rightsizing: high means CPU and memory were measured, medium means CPU only.

Where each change is made

If an opportunity's repository column names one of your four repositories, the change is a pull request there: AWS CDK (TypeScript) in `fleet-infra`, `crucible-infra`, and `enterprise-infra`, and CloudFormation StackSets in `platform-landing-zone`. If it says "(not in code)," the resource was built by hand, and the change is a scripted runbook run by the owner, with the command in the runbook below. Neither path changes anything until the owner signs off.

Deliverable 2: the waste register

$3.2 million a year of StrataForge's bill pays for resources nobody uses: 6,060 findings covering 66,309 resources. Removing them needs no new design, only an owner's yes and a safe way back. The full register is a CSV with one row per finding: `docs/samples/strataforge/data/cost/waste-register.csv`.

The waste register by kind, largest first.

Kind of waste	Findings	Resources	Per month	Per year	One-time cost
Idle instances	146	146	$57,296	$687,554	$1,868
Unattached volumes	1,428	1,428	$46,511	$558,133	$10,472
Orphaned snapshot chains	1,433	43,961	$41,979	$503,752	$0
Idle SageMaker notebooks and Studio apps	49	49	$41,237	$494,839	$0
Unused p5 capacity reservation	1	1	$37,367	$448,400	$0
Unused AMIs and their snapshots	2,727	20,448	$21,483	$257,790	$0
Unused Redshift cluster	1	1	$13,035	$156,419	$0
Idle non-production databases	14	14	$6,192	$74,307	$624
Kinesis streams with no traffic	3	3	$1,528	$18,330	$0
Long-stopped instances	48	48	$955	$11,456	$234
Idle load balancers	44	44	$689	$8,273	$0
Unassociated Elastic IPs	166	166	$563	$6,762	$0
Total	**6,060**	**66,309**	**$268,835**	**$3,226,015**	**$13,198**

The 25 largest findings

The 25 waste findings with the largest annual savings, each with its evidence.

ID	Account	Resource	Finding	Evidence	Owner	Per month	Per year
W-6059	crucible-training-prod	`cr-0c6c09c9a870c0fa9`	Reduce the On-Demand Capacity Reservation to what training uses: 5 x p5.48xlarge reserved, 2.9 in use on average. Proposed: 4 x p5.48xlarge; burst with EC2 Capacity Blocks for ML	30-day average of instances running in the reservation	Hiro Tanaka	$37,367	$448,400
W-6060	corp-data-warehouse	`bi-legacy`	Snapshot and delete the unused dc2 Redshift cluster: 4 x dc2.8xlarge. Proposed: Final snapshot, then delete	no queries since March 2025	Lucas Ferreira	$13,035	$156,419
W-0157	sandbox-ml	`i-0d8c93123ba6b74bd`	Terminate idle instance: g5.12xlarge (Linux), 1 volumes, 88 GB. Proposed: Archive snapshot, then terminate	p95 CPU 1.1%, average 0.4%, network 0.53 GB in 30 days	Hiro Tanaka	$3,859	$46,307
W-6029	crucible-research	`studio-studio-app-83776`	Stop idle notebook and turn on idle shutdown: ml.g5.12xlarge, running 653 hours a month. Proposed: Idle shutdown after 60 minutes (lifecycle configuration)	no kernel activity for 29 days	Hiro Tanaka	$3,517	$42,205
W-6030	crucible-research	`studio-studio-app-03803`	Stop idle notebook and turn on idle shutdown: ml.g5.12xlarge, running 624 hours a month. Proposed: Idle shutdown after 60 minutes (lifecycle configuration)	no kernel activity for 26 days	Hiro Tanaka	$3,321	$39,855
W-6042	crucible-research	`studio-studio-app-f0c4c`	Stop idle notebook and turn on idle shutdown: ml.g5.12xlarge, running 603 hours a month. Proposed: Idle shutdown after 60 minutes (lifecycle configuration)	no kernel activity for 35 days	Hiro Tanaka	$3,185	$38,225
W-6033	crucible-research	`studio-studio-app-cb9b2`	Stop idle notebook and turn on idle shutdown: ml.g5.12xlarge, running 583 hours a month. Proposed: Idle shutdown after 60 minutes (lifecycle configuration)	no kernel activity for 24 days	Hiro Tanaka	$3,054	$36,650
W-6027	crucible-research	`studio-studio-app-8e587`	Stop idle notebook and turn on idle shutdown: ml.g5.12xlarge, running 531 hours a month. Proposed: Idle shutdown after 60 minutes (lifecycle configuration)	no kernel activity for 36 days	Hiro Tanaka	$2,709	$32,512

ID	Account	Resource	Finding	Evidence	Owner	Per month	Per year
W-6041	crucible-research	`studio-studio-app-77573`	Stop idle notebook and turn on idle shutdown: ml.g5.12xlarge, running 391 hours a month. Proposed: Idle shutdown after 60 minutes (lifecycle configuration)	no kernel activity for 30 days	Hiro Tanaka	$1,787	$21,443
W-0075	crucible-research	`i-0d3d374dd66ea66cc`	Terminate idle instance: g5.4xlarge (Linux), 1 volumes, 115 GB. Proposed: Archive snapshot, then terminate	p95 CPU 2.3%, average 0.8%, network 0.46 GB in 30 days	Hiro Tanaka	$1,113	$13,359
W-0071	crucible-research	`i-0134859cef95a12a8`	Terminate idle instance: g5.4xlarge (Linux), 1 volumes, 142 GB. Proposed: Archive snapshot, then terminate	p95 CPU 1.6%, average 0.6%, network 0.11 GB in 30 days	Hiro Tanaka	$1,113	$13,357
W-0074	crucible-research	`i-06eef61249b6709f7`	Terminate idle instance: g5.4xlarge (Linux), 1 volumes, 115 GB. Proposed: Archive snapshot, then terminate	p95 CPU 0.5%, average 0.3%, network 0.59 GB in 30 days	Hiro Tanaka	$1,111	$13,333
W-0070	crucible-research	`i-0ad36e94d736d661a`	Terminate idle instance: g5.4xlarge (Linux), 1 volumes, 94 GB. Proposed: Archive snapshot, then terminate	p95 CPU 3.0%, average 1.6%, network 0.38 GB in 30 days	Hiro Tanaka	$1,110	$13,314
W-0072	crucible-research	`i-0f9a1bef769e75e70`	Terminate idle instance: g5.4xlarge (Linux), 1 volumes, 94 GB. Proposed: Archive snapshot, then terminate	p95 CPU 2.5%, average 1.3%, network 0.56 GB in 30 days	Hiro Tanaka	$1,110	$13,314
W-0076	crucible-research	`i-04c53b2c78bb33d05`	Terminate idle instance: g5.4xlarge (Linux), 1 volumes, 65 GB. Proposed: Archive snapshot, then terminate	p95 CPU 3.1%, average 1.3%, network 0.06 GB in 30 days	Hiro Tanaka	$1,109	$13,303
W-0073	crucible-research	`i-0a5579401ccbc8069`	Terminate idle instance: g5.4xlarge (Linux), 1 volumes, 72 GB. Proposed: Archive snapshot, then terminate	p95 CPU 2.6%, average 0.8%, network 0.17 GB in 30 days	Hiro Tanaka	$1,108	$13,295
W-0077	crucible-research	`i-0b8deb1d821dcca6a`	Terminate idle instance: g5.4xlarge (Linux), 1 volumes, 55 GB. Proposed: Archive snapshot, then terminate	p95 CPU 1.7%, average 0.9%, network 0.16 GB in 30 days	Hiro Tanaka	$1,107	$13,280

ID	Account	Resource	Finding	Evidence	Owner	Per month	Per year
W-0078	crucible-research	`i-0ffcd8a7e38639077`	Terminate idle instance: g5.4xlarge (Linux), 1 volumes, 55 GB. Proposed: Archive snapshot, then terminate	p95 CPU 1.2%, average 0.6%, network 0.11 GB in 30 days	Hiro Tanaka	$1,107	$13,280
W-0194	suspended-legacy-ci	`i-0c2e039619a76e318`	Terminate idle instance: c4.4xlarge (Linux), 2 volumes, 4,703 GB. Proposed: Archive snapshot, then terminate	p95 CPU 1.9%, average 0.9%, network 0.46 GB in 30 days	Elena Vasquez	$978	$11,733
W-0190	suspended-legacy-ci	`i-0bb0a54763063fa2f`	Terminate idle instance: c4.4xlarge (Linux), 2 volumes, 4,544 GB. Proposed: Archive snapshot, then terminate	p95 CPU 3.1%, average 1.1%, network 0.33 GB in 30 days	Elena Vasquez	$963	$11,556
W-0183	suspended-legacy-ci	`i-076ce561bd48be463`	Terminate idle instance: c4.4xlarge (Linux), 2 volumes, 4,450 GB. Proposed: Archive snapshot, then terminate	p95 CPU 1.9%, average 0.9%, network 0.20 GB in 30 days	Elena Vasquez	$954	$11,451
W-0186	suspended-legacy-ci	`i-0ee3f4f06f9f263ac`	Terminate idle instance: c4.4xlarge (Linux), 2 volumes, 4,438 GB. Proposed: Archive snapshot, then terminate	p95 CPU 1.9%, average 0.7%, network 0.11 GB in 30 days	Elena Vasquez	$953	$11,438
W-0192	suspended-legacy-ci	`i-08e592f836ccf9ab5`	Terminate idle instance: c4.4xlarge (Linux), 2 volumes, 4,408 GB. Proposed: Archive snapshot, then terminate	p95 CPU 2.6%, average 1.1%, network 0.03 GB in 30 days	Elena Vasquez	$950	$11,404
W-0184	suspended-legacy-ci	`i-0caeafb95b0c70587`	Terminate idle instance: c4.4xlarge (Linux), 2 volumes, 4,049 GB. Proposed: Archive snapshot, then terminate	p95 CPU 0.4%, average 0.2%, network 0.05 GB in 30 days	Elena Vasquez	$917	$11,004
W-0185	suspended-legacy-ci	`i-0f2d0271f1f1e6db9`	Terminate idle instance: c4.4xlarge (Linux), 2 volumes, 3,974 GB. Proposed: Archive snapshot, then terminate	p95 CPU 2.3%, average 1.4%, network 0.20 GB in 30 days	Elena Vasquez	$910	$10,920

Two patterns stand out. The idle GPU instances and Studio apps in the Crucible research and sandbox accounts are a habit, not an accident: a researcher starts a large instance, finishes, and the instance keeps running. The fix is an idle shutdown that stops them without anyone remembering to (Idle notebooks and Studio apps: stop them, then make idle shutdown the default). And the largest single line is a capacity reservation for p5 instances that keeps two of its five slots idle on average (The p5 capacity reservation: shrink it, and burst with Capacity Blocks).

How every removal is done

The same five rules apply to every runbook below.

1. **The owner signs off first.** The register names the owner. Nothing is deleted on our word alone.
2. **Tag before you act.** Tag the resource `sf:pending-removal=<date>` and wait one business day for an objection.
3. **Keep a way back,** at the cost the runbook states: a snapshot in the archive tier, a final snapshot, or a retention rule in Recycle Bin.
4. **Prove it in the bill.** Each change is measured on the Cost and Usage Report a week later; this is what week 6 counts.
5. **If the resource is in code,** delete it from the repository in a pull request, so the next deploy does not recreate it. If the register says "(not in code)," use the commands.

Runbooks, one per kind of waste

Idle and long-stopped instances: archive a snapshot, then terminate

Scope: 146 idle instances ($687,554 a year) and 48 instances stopped for more than 30 days ($11,456 a year). **Rollback:** restore the snapshots from the archive tier, register the root snapshot as an image, and launch a new instance from the settings the script saves first. The archive tier costs $0.0125 per GB-month against $0.05 for a standard snapshot, with a 90-day minimum and a restore time of up to 72 hours. Keep the archived snapshots for the full 90 days, then delete them; the one-time cost in the register is those three months of archive storage.

The script stops at the first failed call. It terminates only after every snapshot reads `completed`, so a failed snapshot leaves the instance stopped and intact.

```bash
#!/bin/bash
# Archive every volume of one instance, then terminate it. Run as the remediation role, once
per instance.
set -euo pipefail
ID=<instance-id>; REGION=<region>; REG=<register-id>; DELETE_AFTER=<yyyy-mm-dd>

# 1. Stop the instance (an idle instance can wait a week stopped; it is a cheap, reversible
first step).
```

```
aws ec2 stop-instances --instance-ids "$ID" --region "$REGION"
aws ec2 wait instance-stopped --instance-ids "$ID" --region "$REGION"

# 2. Save what a rollback needs: the launch settings (subnet, security groups, profile, key,
private IP, device names)
#    and which volumes outlive the instance.
aws ec2 describe-instances --instance-ids "$ID" --region "$REGION" --query
'Reservations[0].Instances[0]' > "$ID-instance.json"
aws ec2 describe-volumes --region "$REGION" --filters "Name=attachment.instance-id,Values=$ID"
\
  --query 'Volumes[].[VolumeId,Attachments[0].DeleteOnTermination]' --output text > "$ID-
volumes.txt"

# 3. Snapshot every attached volume together, with tags that say why. Stop if no snapshot ID
comes back.
SNAPS=$(aws ec2 create-snapshots \
  --instance-specification "InstanceId=$ID,ExcludeBootVolume=false" \
  --description "sf-cost-removal $REG" \
  --tag-specifications "ResourceType=snapshot,Tags=[{Key=sf:register-id,Value=$REG},
{Key=sf:delete-after,Value=$DELETE_AFTER}]" \
  --region "$REGION" --query 'Snapshots[].SnapshotId' --output text)
case "$SNAPS" in snap-*) ;; *) echo "no snapshot was created; the instance stays" >&2; exit
1 ;; esac

# 4. Wait until each snapshot is completed (every 30 seconds, up to 6 hours; an error state
stops the script).
wait_snapshot() {
  local i state
  for i in $(seq 720); do
    state=$(aws ec2 describe-snapshots --snapshot-ids "$1" --region "$REGION" --query
'Snapshots[0].State' --output text) || return 1
    [ "$state" = completed ] && return 0
    [ "$state" = error ] && return 1
    sleep 30
  done
  return 1
}
for s in $SNAPS; do
  wait_snapshot "$s" || { echo "snapshot $s did not complete; the instance stays" >&2; exit
1; }
done

# 5. Move the completed snapshots to the archive tier (the 90-day minimum applies; sf:delete-
after is that date).
for s in $SNAPS; do
  aws ec2 modify-snapshot-tier --snapshot-id "$s" --storage-tier archive --region "$REGION"
done

# 6. Turn off termination protection, terminate, and wait for it.
aws ec2 modify-instance-attribute --instance-id "$ID" --no-disable-api-termination --region
"$REGION"
aws ec2 terminate-instances --instance-ids "$ID" --region "$REGION"
aws ec2 wait instance-terminated --instance-ids "$ID" --region "$REGION"

# 7. Volumes marked DeleteOnTermination=False are now unattached and still billing. Their data
is in the completed
#    snapshots from step 4, so delete them now.
```

```
awk '$2 == "False" {print $1}' "$ID-volumes.txt" | while read -r vol; do
  aws ec2 delete-volume --volume-id "$vol" --region "$REGION"
done
```

Verify with `aws ec2 describe-instances --instance-ids <instance-id> --query 'Reservations[].Instances[].State.Name'` (it returns `terminated`), then check the instance's line in the next day's Cost and Usage Report. An Elastic IP that was attached is now unassociated; the Elastic IP runbook below handles it. To roll back, read the settings from `<instance-id>-instance.json` and run:

```bash
# 1. Restore every snapshot of the register ID for 7 days, then wait (up to 72 hours) for
temporary-restore-completed.
aws ec2 restore-snapshot-tier --snapshot-id <snapshot-id> --temporary-restore-days 7 --region
<region>
aws ec2 describe-snapshot-tier-status --filters Name=snapshot-id,Values=<snapshot-id> --region
<region> \
  --query 'SnapshotTierStatuses[0].Status' --output text

# 2. Register the root snapshot as an image, with the saved root device name and architecture
#    (add --boot-mode uefi if the saved file shows BootMode uefi).
aws ec2 register-image --name sf-restore-<register-id> --architecture <architecture> --
virtualization-type hvm --ena-support \
  --root-device-name <root-device-name> \
  --block-device-mappings DeviceName=<root-device-name>,Ebs={SnapshotId=<root-snapshot-
id>,VolumeType=gp3} \
  --region <region>

# 3. Launch from the saved settings. Add one more --block-device-mappings entry for each data
volume.
aws ec2 run-instances --image-id <new-image-id> --instance-type <old-instance-type> --subnet-id
<subnet-id> \
  --security-group-ids <security-group-ids> --iam-instance-profile Arn=<instance-profile-arn>
--key-name <key-name> \
  --private-ip-address <private-ip-address> \
  --block-device-mappings DeviceName=<data-device-name>,Ebs={SnapshotId=<data-snapshot-
id>,VolumeType=gp3} \
  --region <region>
```

Unattached volumes: archive a snapshot, then delete

Scope: 1,428 volumes ($558,133 a year; each archived snapshot is kept the 90-day minimum, then deleted). The script below does the whole register in one pass: it reads the CSV, skips every row whose owner has not signed off, snapshots, archives, and only then deletes. It makes no change unless run with `--apply`.

```python
#!/usr/bin/env python3
"""Archive a snapshot of each approved unattached volume, then delete the volume.

usage: remove_volumes.py waste-register.csv accounts.json [--apply]
```

```
accounts.json maps an account name to {"profile": "<AWS CLI profile>"}. Each profile assumes
the remediation role with
role_arn and source_profile in ~/.aws/config, so boto3 refreshes the credentials before they
expire.
The register needs an "approved" column (yes) added by the owner before a row is touched.
Two passes: snapshot every approved volume, then archive and delete each volume whose snapshot
reads "completed".
"""
import csv
import datetime
import json
import sys
import time
import boto3
from botocore.exceptions import ClientError

path, accounts_path = sys.argv[1], sys.argv[2]
apply = "--apply" in sys.argv
ARCHIVE_DAYS = 90  # the archive tier's minimum storage duration
WAIT_SECONDS = 6 * 3600  # give large snapshots six hours in all
delete_after = (datetime.date.today() + datetime.timedelta(days=ARCHIVE_DAYS)).isoformat()
accounts = json.load(open(accounts_path))
clients = {}


def ec2_for(account, region):
    if (account, region) not in clients:
        session = boto3.Session(profile_name=accounts[account]["profile"], region_name=region)
        clients[(account, region)] = session.client("ec2")
    return clients[(account, region)]


pending = []  # (ec2, volume ID, snapshot ID, register ID)
for row in csv.DictReader(open(path)):
    if row["kind"] != "unattached-volume" or row.get("approved", "").lower() != "yes":
        continue
    ec2 = ec2_for(row["account"], row["region"])
    vol = row["resource"]
    try:
        state = ec2.describe_volumes(VolumeIds=[vol])["Volumes"][0]
        if state["State"] != "available":
            print(f"skip {vol}: state is {state['State']}")
            continue
        print(f"{'SNAPSHOT' if apply else 'would snapshot'} {vol} ({state['Size']} GiB),
archive it, then delete the volume")
        if apply:
            snap = ec2.create_snapshot(VolumeId=vol, Description=f"sf-cost-removal
{row['id']}", TagSpecifications=[
                {"ResourceType": "snapshot", "Tags": [{"Key": "sf:register-id", "Value":
row["id"]},
                                                      {"Key": "sf:delete-after", "Value":
delete_after}]}])["SnapshotId"]
            pending.append((ec2, vol, snap, row["id"]))
    except ClientError as err:
        print(f"skip {vol}: {err.response['Error']['Code']}")  # one bad row never stops the
rest
```

```
deadline = time.time() + WAIT_SECONDS
while pending and time.time() < deadline:
    still = []
    for ec2, vol, snap, reg in pending:
        try:
            snap_state = ec2.describe_snapshots(SnapshotIds=[snap])["Snapshots"][0]["State"]
            if snap_state == "pending":
                still.append((ec2, vol, snap, reg))
            elif snap_state == "completed":
                ec2.modify_snapshot_tier(SnapshotId=snap, StorageTier="archive")
                ec2.delete_volume(VolumeId=vol)  # reached only after the snapshot completed
and the archive call succeeded
                print(f"  {reg}: snapshot {snap} archived; volume {vol} deleted")
            else:
                print(f"  {reg}: snapshot {snap} is {snap_state}; volume {vol} kept")
        except ClientError as err:
            print(f"  {reg}: {err.response['Error']['Code']} on {vol}; volume kept")
    pending = still
    if pending:
        time.sleep(30)
for _, vol, snap, reg in pending:
    print(f"  {reg}: snapshot {snap} not complete after six hours; volume {vol} kept")
```

Roll back with `restore-snapshot-tier`, wait for `temporary-restore-completed` as in the instance rollback, and then `create-volume` in the original Availability Zone. A volume is deleted only after its snapshot reads `completed` and the archive call succeeds, so there is no moment when the data exists in neither place, and any snapshot that fails or stalls leaves its volume where it is. The script tags each snapshot `sf:delete-after` with the date 90 days out; delete it then with `aws ec2 delete-snapshot`; an archived snapshot deleted earlier is still billed for the days it had left.

Orphaned snapshot chains: delete through Recycle Bin

Scope: 1,433 chains ($503,752 a year). The source volume is gone, no AMI uses the snapshot, no backup policy owns it, and the newest snapshot is more than 90 days old. EBS snapshots are incremental, so deleting one frees only the blocks no other snapshot needs; deletion order within a chain does not matter. Put a Recycle Bin rule in front of the deletion so that a mistake is recoverable for 14 days:

```bash
# Once per Region. The tag limits the rule to snapshots this work deletes; without it the rule
holds every deleted
# snapshot in the Region (DLM rotations included) for 14 more days at the full snapshot price.
aws rbin create-rule \
  --resource-type EBS_SNAPSHOT \
  --resource-tags ResourceTagKey=sf:register-id \
  --retention-period RetentionPeriodValue=14,RetentionPeriodUnit=DAYS \
  --description "sf-cost-removal: recover a deleted snapshot within 14 days" \
  --region <region>

aws ec2 describe-snapshots --owner-ids self --region <region> \
  --filters Name=tag:sf:register-id,Values=<register-id> --query 'Snapshots[].SnapshotId' --
output text
aws ec2 delete-snapshot --snapshot-id <snapshot-id> --region <region>
```

To roll back inside the window: `aws ec2 restore-snapshot-from-recycle-bin --snapshot-id <snapshot-id> --region <region>`. Snapshots held in Recycle Bin are billed at the normal snapshot rate, so the saving starts when the 14 days end. The register's per-month figure is the steady state after that.

Unused AMIs and their snapshots: deregister, then delete the snapshots

Scope: 2,727 AMIs ($257,790 a year). Check that nothing launched from the image recently, deregister it, and delete the snapshots behind it. `--delete-associated-snapshots` deletes the snapshots too, and Recycle Bin keeps an image and its snapshots under separate rules, so the way back needs one rule of type `EC2_IMAGE` and one of type `EBS_SNAPSHOT`, both limited by tag, each held for 14 days.

```bash
# 1. Checks. Any output from the permission queries, or any launch template that names the
image, stops the removal.
aws ec2 describe-image-attribute --image-id <ami-id> --attribute lastLaunchedTime --region
<region>
aws ec2 describe-image-attribute --image-id <ami-id> --attribute launchPermission --region
<region> \
  --query 'LaunchPermissions'
aws ec2 describe-snapshot-attribute --snapshot-id <snapshot-id> --attribute
createVolumePermission --region <region> \
  --query 'CreateVolumePermissions'
aws ec2 describe-launch-template-versions --versions '$Latest,$Default' --region <region> \
  --filters Name=image-id,Values=<ami-id> --query 'LaunchTemplateVersions[].LaunchTemplateId'

# 2. Once per Region: one rule for the image and one for its snapshots, each limited by tag.
aws rbin create-rule --resource-type EC2_IMAGE --resource-tags ResourceTagKey=sf:register-id \
  --retention-period RetentionPeriodValue=14,RetentionPeriodUnit=DAYS \
  --description "sf-cost-removal: recover a deregistered AMI" --region <region>
aws rbin create-rule --resource-type EBS_SNAPSHOT --resource-tags ResourceTagKey=sf:register-id
\
  --retention-period RetentionPeriodValue=14,RetentionPeriodUnit=DAYS \
  --description "sf-cost-removal: recover the snapshots of a deregistered AMI" --region
<region>

# 3. Tag the image and each snapshot behind it, then deregister.
aws ec2 create-tags --resources <ami-id> <snapshot-id> --tags Key=sf:register-
id,Value=<register-id> --region <region>
aws ec2 deregister-image --image-id <ami-id> --delete-associated-snapshots --region <region>
```

Roll back inside the 14 days by restoring the snapshots first and the image second: `aws ec2 restore-snapshot-from-recycle-bin --snapshot-id <snapshot-id> --region <region>` for each snapshot, then `aws ec2 restore-image-from-recycle-bin --image-id <ami-id> --region <region>`. The launch-permission and volume-permission queries show whether the image or its snapshots are shared with another account, and `lastLaunchedTime` shows when the image last started an instance. It does not show a launch template or an Auto Scaling group that names the image, which is what the launch-template query is for. Instances that launched from the image do not depend on it once they are running.

Unassociated Elastic IPs: release

Scope: 166 addresses ($6,762 a year at $0.005 an hour each, used or not). Check the list against the addresses your customers have allow-listed before releasing any of them; those stay.

```bash
aws ec2 describe-addresses --region <region> \
  --query 'Addresses[?AssociationId==`null`].[AllocationId,PublicIp,Tags]' --output table
aws ec2 release-address --allocation-id <eipalloc-id> --region <region>
```

If you release an address in error, `aws ec2 allocate-address --address <ip-address> --region <region>` recovers it as long as nobody else has been allocated it in the meantime.

The p5 capacity reservation: shrink it, and burst with Capacity Blocks

Scope: one On-Demand Capacity Reservation in `crucible-training-prod` ($448,400 a year). It reserves 5 `p5.48xlarge` instances and averages 2.9 in use over 30 days. We recommend 4, and Capacity Blocks for ML for the peaks. **This reservation is rated Medium risk** (so are the long-stopped instances in `sf-forensics`, which may be held as evidence), and the reason is honest: capacity you release may not be available when you ask for it back. Do not shrink it during a training run the team cannot interrupt. The change is a pull request in `crucible-infra`:

```typescript
import * as ec2 from "aws-cdk-lib/aws-ec2";

new ec2.CfnCapacityReservation(this, "TrainingP5Reservation", {
  instanceType: "p5.48xlarge",
  instancePlatform: "Linux/UNIX",
  availabilityZone: "<availability-zone>",
  instanceCount: 4, // was 5; average use is 2.9
  instanceMatchCriteria: "targeted",
  endDateType: "unlimited",
});
```

Or, for a reservation not in code, directly:

```bash
aws ec2 describe-capacity-reservations --capacity-reservation-ids <cr-id> --region us-east-1
aws ec2 modify-capacity-reservation --capacity-reservation-id <cr-id> --instance-count 4 --
region us-east-1
```

For a peak, buy a block for the days you need it:

```bash
aws ec2 describe-capacity-block-offerings --instance-type p5.48xlarge --instance-count 1 \
  --capacity-duration-hours 72 --start-date-range <start> --end-date-range <end> --region us-
east-1
aws ec2 purchase-capacity-block --capacity-block-offering-id <offering-id> --instance-platform
Linux/UNIX --region us-east-1
```

To roll back, raise `--instance-count` to 5 again. If the capacity is gone, the call fails; that is the risk, and the reason Hiro Tanaka confirms the training calendar first.

Idle notebooks and Studio apps: stop them, then make idle shutdown the default

Scope: 49 notebooks and Studio apps ($494,839 a year), most of them large GPU types with no kernel activity for weeks. Stop what is running, then turn on idle shutdown for every Studio domain so that the habit stops costing money. The files stay in the user's home directory when an app stops.

```bash
# JupyterLab apps live in spaces, so find the space name first.
aws sagemaker list-apps --domain-id-equals <domain-id> --region us-east-1 \
  --query "Apps[?Status=='InService' && AppType=='JupyterLab'].[SpaceName,AppName]" --output
text
aws sagemaker delete-app --domain-id <domain-id> --space-name <space-name> \
  --app-type JupyterLab --app-name <app-name> --region us-east-1

# Idle shutdown for new JupyterLab apps, in the domain's user and space defaults.
aws sagemaker update-domain --domain-id <domain-id> --region us-east-1 \
  --default-user-settings '{"JupyterLabAppSettings": {"AppLifecycleManagement":
{"IdleSettings": {
    "LifecycleManagement": "ENABLED", "IdleTimeoutInMinutes": 60,
    "MinIdleTimeoutInMinutes": 60, "MaxIdleTimeoutInMinutes": 240}}}}' \
  --default-space-settings '{"JupyterLabAppSettings": {"AppLifecycleManagement":
{"IdleSettings": {
    "LifecycleManagement": "ENABLED", "IdleTimeoutInMinutes": 60,
    "MinIdleTimeoutInMinutes": 60, "MaxIdleTimeoutInMinutes": 240}}}}'
```

Notebook instances have no such setting; they take a lifecycle configuration whose `on-start` script installs an auto-stop check. This is the condensed form of AWS's published auto-stop sample:

```bash
cat > on-start.sh <<'SCRIPT'
#!/bin/bash
set -e
IDLE_SECONDS=3600
cat > /home/ec2-user/autostop.py <<'PY'
import json, ssl, sys, time, urllib.request
import boto3

idle_limit = int(sys.argv[1])
ctx = ssl._create_unverified_context()  # the notebook server's local certificate is self-
signed
sessions = json.load(urllib.request.urlopen("https://localhost:8443/api/sessions",
context=ctx))
now = time.time()
idle = True
if not sessions:
    # No kernel has run yet, so count idle time from when the instance last started (AWS's own
sample does the same).
    name = json.load(open("/opt/ml/metadata/resource-metadata.json"))["ResourceName"]
    started = boto3.client("sagemaker").describe_notebook_instance(NotebookInstanceName=name)
```

```
["LastModifiedTime"].timestamp()
    idle = now - started >= idle_limit
for s in sessions:
    kernel = s["kernel"]
    if kernel["execution_state"] != "idle":
        idle = False
        break
    last = time.mktime(time.strptime(kernel["last_activity"][:19], "%Y-%m-%dT%H:%M:%S")) -
time.timezone
    if now - last < idle_limit:
        idle = False
        break
if idle:
    name = json.load(open("/opt/ml/metadata/resource-metadata.json"))["ResourceName"]
    boto3.client("sagemaker").stop_notebook_instance(NotebookInstanceName=name)
PY
# The system Python has no boto3; the notebook server's own environment does. Append to the
crontab without replacing it.
PYTHON=/home/ec2-user/anaconda3/envs/JupyterSystemEnv/bin/python
CRON=$(mktemp)
crontab -l > "$CRON" 2>/dev/null || true
grep -q autostop.py "$CRON" || echo "*/5 * * * * $PYTHON /home/ec2-user/autostop.py
$IDLE_SECONDS" >> "$CRON"
crontab "$CRON"
SCRIPT

aws sagemaker create-notebook-instance-lifecycle-config \
  --notebook-instance-lifecycle-config-name sf-idle-shutdown-60 \
  --on-start Content="$(base64 < on-start.sh)" --region us-east-1
aws sagemaker stop-notebook-instance --notebook-instance-name <name> --region us-east-1
aws sagemaker wait notebook-instance-stopped --notebook-instance-name <name> --region us-east-1
aws sagemaker update-notebook-instance --notebook-instance-name <name> \
  --lifecycle-config-name sf-idle-shutdown-60 --region us-east-1
aws sagemaker wait notebook-instance-stopped --notebook-instance-name <name> --region us-east-1
aws sagemaker start-notebook-instance --notebook-instance-name <name> --region us-east-1
```

The notebook instance's role needs `sagemaker:StopNotebookInstance` and `sagemaker:DescribeNotebookInstance` on itself. A new notebook instance gets a full idle limit before its first stop, because the script counts from the last start when no kernel has run. Roll back by stopping the instance, removing the lifecycle configuration (`aws sagemaker update-notebook-instance --notebook-instance-name <name> --disassociate-lifecycle-config`), and starting it again.

The unused Redshift cluster: final snapshot, then delete

Scope: one cluster of 4 x dc2.8xlarge, `bi-legacy`, in `corp-data-warehouse` ($156,419 a year; no queries since March 2025). Take the snapshot first, keep it for a year, then delete.

```bash
aws redshift delete-cluster --cluster-identifier bi-legacy \
  --final-cluster-snapshot-identifier bi-legacy-final-<yyyymmdd> \
  --final-cluster-snapshot-retention-period 365 --region us-east-1
```

To roll back: `aws redshift restore-from-cluster-snapshot --cluster-identifier bi-legacy --snapshot-identifier bi-legacy-final-<yyyymmdd> --node-type ra3.xlplus --number-of-nodes 2`. The `dc2` node type is an older generation; if the cluster is ever needed again it should come back on RA3.

Idle non-production databases, dead streams, and idle load balancers

Three small kinds share one pattern: confirm idleness, save the configuration, delete with a final copy.

- **Idle non-production databases** (14, $74,307 a year): the final snapshot is kept 90 days, then deleted, and the one-time cost in the register is that storage. Turn deletion protection off first (`aws rds modify-db-instance --db-instance-identifier <id> --no-deletion-protection --apply-immediately`, or `aws rds modify-db-cluster --db-cluster-identifier <id> --no-deletion-protection --apply-immediately` for Aurora), because a protected database refuses the delete. Then `aws rds delete-db-instance --db-instance-identifier <id> --final-db-snapshot-identifier <id>-final --delete-automated-backups` for an RDS instance, or `aws rds delete-db-cluster --db-cluster-identifier <id> --final-db-snapshot-identifier <id>-final` after deleting its instances for Aurora. Then tag the final snapshot (`aws rds add-tags-to-resource --resource-name <snapshot-arn> --tags Key=sf:delete-after,Value=<yyyy-mm-dd>`) with the date 90 days out, and delete it then. Roll back inside the window with `aws rds restore-db-instance-from-db-snapshot` or `restore-db-cluster-from-snapshot`.
- **Kinesis streams with no traffic** (3, $18,330 a year): confirm `IncomingRecords` was zero for 30 days, check there is no consumer, then `aws kinesis delete-stream --stream-name <name> --region <region>`. A stream with no traffic holds nothing worth restoring; roll back by recreating it from `fleet-infra`.
- **Idle load balancers** (44, $8,273 a year): save the whole configuration first, then delete. A recreated load balancer gets a new DNS name, so look for the old one in Route 53 before you delete. Deleting a load balancer leaves its target groups behind, so the saved files carry them and you decide separately whether to delete them.

```bash
# Save the load balancer, its attributes and tags, every listener and its rules, and its target groups with their attributes and targets.
aws elbv2 describe-load-balancers --load-balancer-arns <lb-arn> --region <region> > <lb-name>-lb.json
aws elbv2 describe-load-balancer-attributes --load-balancer-arn <lb-arn> --region <region> > <lb-name>-attributes.json
aws elbv2 describe-tags --resource-arns <lb-arn> --region <region> > <lb-name>-tags.json
aws elbv2 describe-listeners --load-balancer-arn <lb-arn> --region <region> > <lb-name>-listeners.json
for l in $(aws elbv2 describe-listeners --load-balancer-arn <lb-arn> --region <region> --query 'Listeners[].ListenerArn' --output text); do
  aws elbv2 describe-rules --listener-arn "$l" --region <region> > "<lb-name>-rules-$(basename "$l").json"
done
aws elbv2 describe-target-groups --load-balancer-arn <lb-arn> --region <region> > <lb-name>-target-groups.json
for tg in $(aws elbv2 describe-target-groups --load-balancer-arn <lb-arn> --region <region> --query 'TargetGroups[].TargetGroupArn' --output text); do
  aws elbv2 describe-target-group-attributes --target-group-arn "$tg" --region <region> > "<lb-name>-tg-attributes-$(basename "$tg").json"
  aws elbv2 describe-target-health --target-group-arn "$tg" --region <region> > "<lb-name>-tg-health-$(basename "$tg").json"
done

# Confirm the files are not empty, turn off deletion protection if it is on, then delete.
test -s <lb-name>-lb.json && test -s <lb-name>-listeners.json
aws elbv2 modify-load-balancer-attributes --load-balancer-arn <lb-arn> --attributes Key=deletion_protection.enabled,Value=false --region <region>
aws elbv2 delete-load-balancer --load-balancer-arn <lb-arn> --region <region>
```

To roll back, recreate the load balancer with `aws elbv2 create-load-balancer` from `<lb-name>-lb.json`, then the listeners and their rules from the saved files, pointing them at the target groups that remain (recreate any you deleted from `<lb-name>-target-groups.json`).

For a Classic Load Balancer use `aws elb describe-load-balancers` and `aws elb delete-load-balancer --load-balancer-name <name>`; the Classic type is also one to migrate off, which is a separate change.

Deliverable 3: the rightsizing plan

Compute, databases, caches, search, and brokers sized to what they use are worth **$4.4 million a year** (2,205 changes on 2,205 resources). Of that, **$3.5 million** comes from moving to AWS Graviton, which prices each vCPU lower than the equivalent x86 instance and is the better default for most of what StrataForge runs. The rest comes from sizes that fit. The full plan is `docs/samples/strataforge/data/cost/rightsizing-plan.csv`.

The rules

We measured the 95th percentile of CPU, memory, network, and connections over 14 days. We size to the 95th percentile, not the average, and we leave production more headroom than non-production.

The thresholds behind every rightsizing recommendation.

Environment	One size down	Two sizes down
Production	p95 CPU at or under 30% and p95 memory at or under 45%	Never
Non-production	p95 CPU at or under 40% and p95 memory at or under 50%	p95 CPU at or under 20% and p95 memory at or under 35%
No memory data (agent missing)	one step only when p95 CPU is at or under 15% (production) or 25% (non-production)	Never

An instance below `large`, a GPU host, an SAP HANA host, a Spot instance, and an instance in a capacity reservation are never resized by these rules. An instance with both CPU and memory measured is marked "high confidence." One with CPU only is marked "medium," and we resize it no more than one step.

Databases follow stricter rules than instances. A database is resized only when both CPU and memory have headroom on the smaller class. Memory is judged by the working set, not by free memory, because a database fills its buffer pool on purpose and free memory says little about what a smaller class would need. So each database row's evidence names its 95th-percentile CPU, the working set as a share of memory, and the buffer cache hit ratio, not CPU alone. A database changes by cluster, never by single instance: the readers move first, the writer follows through a planned failover, and nothing changes in a cluster whose smaller class would leave a reader without headroom.

What is in the plan

The rightsizing plan by kind of change.

Kind of change	Changes	Resources	Per month	Per year
Rightsize and move to Graviton	579	579	$183,124	$2,197,489
Move to Graviton, same size	867	867	$74,874	$898,490
Rightsize (same family)	340	340	$58,384	$700,605
ElastiCache: Valkey on Graviton	22	22	$17,828	$213,937

Kind of change	Changes	Resources	Per month	Per year
Lambda on Arm	272	272	$10,562	$126,742
Kinesis shards sized to peak	32	32	$8,821	$105,858
Databases: rightsize and Graviton	74	74	$7,391	$88,686
OpenSearch: Graviton and gp3	4	4	$3,517	$42,204
MSK brokers on Graviton	6	6	$1,754	$21,043
Move to the current generation	9	9	$252	$3,027
Total	**2,205**	**2,205**	**$366,507**	**$4,398,082**

Graviton targets

1,446 instances move to Graviton: 533 to c7g, 695 to m7g, 208 to r7g, and 10 to t4g. The `m7g` and `c7g` targets are for general-purpose and compute-heavy hosts, `r7g` for memory-heavy ones such as the EMR fleet, and `t4g` for the small burstable hosts. We chose generation 7 because it is also the oldest generation the Database Savings Plan covers.

The 25 largest compute changes

The 25 compute changes with the largest annual savings.

ID	Account	Resource	Change	Evidence	Owner and repository	Wave	Risk	Per month	Per year
R-1718	mfg-mes-prod	`i-0f267afe0c90c5d4f`	r5.4×large (Windows+SQLEnt) to r5.2×large	p95 CPU 17%, p95 memory 44% over 14 days; high confidence	Mei Chen, `enterprise-infra`	Week 4	Medium	$2,699	$32,392
R-0333	fleet-ingest-prod	`i-08782f899c09c727c`	c5.9×large (Linux) to c7g.4×large	p95 CPU 29%, p95 memory 36% over 14 days; high confidence	Sam Okafor, `fleet-infra`	Week 4	Medium	$814	$9,762
R-0011	sf-cicd	`i-08997ed81347972dd`	c5.9×large (Linux) to c5.2×large	p95 CPU 20%, p95 memory 24% over 14 days; high confidence	Elena Vasquez, `platform-landing-zone`	Week 4	Low	$808	$9,695
R-0679	fleet-data-prod	`i-0fdc10bcc96385d5e`	r5.8×large (Linux) to r7g.4×large	p95 CPU 29%, p95 memory 33% over 14 days; high confidence	Lucas Ferreira, `fleet-infra`	Backlog (from December 1, 2026)	Medium	$787	$9,444
R-0694	fleet-data-prod	`i-097c996730ac1af29`	r5.8×large (Linux) to r7g.4×large	p95 CPU 24%, p95 memory 39% over 14 days; high confidence	Lucas Ferreira, `fleet-infra`	Backlog (from December 1, 2026)	Medium	$787	$9,444
R-0705	fleet-data-prod	`i-069ef295d5caf7fa2`	r5.8×large (Linux) to r7g.4×large	p95 CPU 20%, p95 memory 36% over 14 days; high confidence	Lucas Ferreira, `fleet-infra`	Backlog (from December 1, 2026)	Medium	$787	$9,444
R-0712	fleet-data-prod	`i-042e01aa31afa729d`	r5.8×large (Linux) to r7g.4×large	p95 CPU 14%, p95 memory 41% over 14 days; high confidence	Lucas Ferreira, `fleet-infra`	Backlog (from December 1, 2026)	Medium	$787	$9,444
R-0722	fleet-data-prod	`i-039645f8ed90d8360`	r5.8×large (Linux) to r7g.4×large	p95 CPU 23%, p95 memory 43% over 14 days; high confidence	Lucas Ferreira, `fleet-infra`	Backlog (from December 1, 2026)	Medium	$787	$9,444

ID	Account	Resource	Change	Evidence	Owner and repository	Wave	Risk	Per month	Per year
R-0753	fleet-data-prod	`i-0b5ad4d4612cc1e2c`	r5.8xlarge (Linux) to r7g.4xlarge	p95 CPU 28%, p95 memory 31% over 14 days; high confidence	Lucas Ferreira, `fleet-infra`	Backlog (from December 1, 2026)	Medium	$787	$9,444
R-0301	fleet-ingest-prod	`i-009a361b9c06658b8`	c5.9xlarge (Linux) to c7g.4xlarge	p95 CPU 23%, p95 memory 44% over 14 days; high confidence	Sam Okafor, `fleet-infra`	Week 4	Medium	$738	$8,851
R-0316	fleet-ingest-prod	`i-090b1ee9fbdb14a17`	c5.9xlarge (Linux) to c7g.4xlarge	p95 CPU 25%, p95 memory 43% over 14 days; high confidence	Sam Okafor, `fleet-infra`	Week 4	Medium	$738	$8,851
R-0201	fleet-ingest-prod	`i-0836dfc39a7030dcd`	c5.9xlarge (Linux) to c7g.4xlarge	p95 CPU 25%, p95 memory 44% over 14 days; high confidence	Sam Okafor, `fleet-infra`	Week 4	Medium	$645	$7,739
R-0219	fleet-ingest-prod	`i-00625e98bb75fff43`	c5.9xlarge (Linux) to c7g.4xlarge	p95 CPU 16%, p95 memory 43% over 14 days; high confidence	Sam Okafor, `fleet-infra`	Week 4	Medium	$645	$7,739
R-0227	fleet-ingest-prod	`i-0b5d81c67b91d3ffa`	c5.9xlarge (Linux) to c7g.4xlarge	p95 CPU 26%, p95 memory 44% over 14 days; high confidence	Sam Okafor, `fleet-infra`	Week 4	Medium	$645	$7,739
R-0233	fleet-ingest-prod	`i-034346c3ab3fc685c`	c5.9xlarge (Linux) to c7g.4xlarge	p95 CPU 21%, p95 memory 45% over 14 days; high confidence	Sam Okafor, `fleet-infra`	Week 4	Medium	$645	$7,739
R-0013	sf-cicd	`i-0bc70eb0e8dfc9a00`	c5.9xlarge (Linux) to c5.4xlarge	p95 CPU 23%, p95 memory 30% over 14 days; high confidence	Elena Vasquez, `platform-landing-zone`	Week 4	Low	$577	$6,925
R-0019	sf-cicd	`i-0e9b3d9093da769a3`	c5.9xlarge (Linux) to c5.4xlarge	p95 CPU 23%, p95 memory 47% over 14 days; high confidence	Elena Vasquez, `platform-landing-zone`	Week 4	Low	$577	$6,925
R-0021	sf-cicd	`i-09a8ee80722bb3cbe`	c5.9xlarge (Linux) to c5.4xlarge	p95 CPU 18% (no memory metrics: CloudWatch agent missing) over 14 days; medium confidence	Elena Vasquez, `platform-landing-zone`	Week 4	Low	$577	$6,925

ID	Account	Resource	Change	Evidence	Owner and repository	Wave	Risk	Per month	Per year
R-0024	sf-cicd	`i-090b5d4340b3513d8`	c5.9xlarge (Linux) to c5.4xlarge	p95 CPU 24%, p95 memory 48% over 14 days; high confidence	Elena Vasquez, `platform-landing-zone`	Week 4	Low	$577	$6,925
R-0027	sf-cicd	`i-0c2666506fa16933c`	c5.9xlarge (Linux) to c5.4xlarge	p95 CPU 16%, p95 memory 45% over 14 days; high confidence	Elena Vasquez, `platform-landing-zone`	Week 4	Low	$577	$6,925
R-0030	sf-cicd	`i-06b877796b07fcbf8`	c5.9xlarge (Linux) to c5.4xlarge	p95 CPU 34%, p95 memory 38% over 14 days; high confidence	Elena Vasquez, `platform-landing-zone`	Week 4	Low	$577	$6,925
R-1706	plm-prod	`i-09ffafd850f4c7cb7`	m5.4xlarge (Windows) to m5.2xlarge	p95 CPU 21%, p95 memory 27% over 14 days; high confidence	Mei Chen, `enterprise-infra`	Week 4	Medium	$511	$6,126
R-1707	plm-prod	`i-0965f41a18627ec0f`	m5.4xlarge (Windows) to m5.2xlarge	p95 CPU 16%, p95 memory 39% over 14 days; high confidence	Mei Chen, `enterprise-infra`	Week 4	Medium	$511	$6,126
R-1708	plm-prod	`i-05584531b5cca49c4`	m5.4xlarge (Windows) to m5.2xlarge	p95 CPU 15%, p95 memory 37% over 14 days; high confidence	Mei Chen, `enterprise-infra`	Week 4	Medium	$511	$6,126
R-1709	plm-prod	`i-0f6764aa45a226024`	m5.4xlarge (Windows) to m5.2xlarge	p95 CPU 27%, p95 memory 33% over 14 days; high confidence	Mei Chen, `enterprise-infra`	Week 4	Medium	$511	$6,126

Databases

74 database instances are larger than their load ($88,686 a year), and 22 ElastiCache clusters, 4 OpenSearch domains, and 6 MSK clusters move to Graviton. Databases are sized to CPU and memory headroom together, with connections as a check: a database with a handful of connections, single-digit CPU, and a working set that fits in a smaller class with the buffer cache hit ratio still near 100% does not need the larger class, and one with low CPU but a working set too large for the smaller class stays where it is.

The 12 largest database rightsizing changes.

ID	Account	Database	Change	Evidence	Owner	Wave	Per month	Per year
R-1866	kestrel-prod	`db-perception-api-636ad2`	mysql db.r5.4xlarge Multi-AZ to db.r7g.2xlarge	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	Jonah Reyes	Week 4	$1,308	$15,699
R-1806	fleet-api-prod	`db-fleet-api-8ebcdb`	aurora-postgresql db.r6g.4xlarge to db.r7g.2xlarge	p95 CPU 4%; working set 35% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 117 connections on average over 14 days	Sam Okafor	Week 4	$659	$7,902
R-1807	fleet-api-prod	`db-fleet-api-876197`	aurora-postgresql db.r6g.4xlarge to db.r7g.2xlarge	p95 CPU 5%; working set 35% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 117 connections on average over 14 days	Sam Okafor	Week 4	$659	$7,902
R-1863	corp-apps	`db-atlassian-7ac263`	postgres db.m5.2xlarge Multi-AZ to db.m7g.xlarge	p95 CPU 22%; working set 37% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 270 connections on average over 14 days	Mei Chen	Week 4	$509	$6,110
R-1796	fleet-api-prod	`db-fleet-api-2f0b88`	aurora-postgresql db.r6i.8xlarge to db.r7g.8xlarge	p95 CPU 27%; working set 66% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 341 connections on average over 14 days	Sam Okafor	Week 4	$148	$1,776
R-1797	fleet-api-prod	`db-fleet-api-469e4d`	aurora-postgresql db.r6i.8xlarge to db.r7g.8xlarge	p95 CPU 39%; working set 32% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 341 connections on average over 14 days	Sam Okafor	Week 4	$148	$1,776
R-1798	fleet-api-prod	`db-fleet-api-6fac55`	aurora-postgresql db.r6i.8xlarge to db.r7g.8xlarge	p95 CPU 34%; working set 57% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 296 connections on average over 14 days	Sam Okafor	Week 4	$148	$1,776

ID	Account	Database	Change	Evidence	Owner	Wave	Per month	Per year
R-1799	fleet-api-prod	`db-fleet-api-083a8f`	aurora-postgresql db.r6i.8xlarge to db.r7g.8xlarge	p95 CPU 34%; working set 58% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 296 connections on average over 14 days	Sam Okafor	Week 4	$148	$1,776
R-1800	fleet-api-prod	`db-fleet-api-3e018a`	aurora-postgresql db.r6i.8xlarge to db.r7g.8xlarge	p95 CPU 49%; working set 29% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 296 connections on average over 14 days	Sam Okafor	Week 4	$148	$1,776
R-1801	fleet-api-prod	`db-fleet-api-29521e`	aurora-postgresql db.r6i.8xlarge to db.r7g.8xlarge	p95 CPU 11%; working set 73% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 142 connections on average over 14 days	Sam Okafor	Week 4	$148	$1,776
R-1802	fleet-api-prod	`db-fleet-api-8d661e`	aurora-postgresql db.r6i.8xlarge to db.r7g.8xlarge	p95 CPU 11%; working set 68% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 142 connections on average over 14 days	Sam Okafor	Week 4	$148	$1,776
R-1803	fleet-api-prod	`db-fleet-api-df4f38`	aurora-postgresql db.r6i.8xlarge to db.r7g.8xlarge	p95 CPU 16%; working set 50% of memory with the buffer cache hit ratio above 99.9% (Performance Insights and Enhanced Monitoring); 142 connections on average over 14 days	Sam Okafor	Week 4	$148	$1,776

In code, a database change is one line in the CDK construct. This is the shape in `fleet-infra`; the writer and a reader both move to Graviton (`db.r6i.8xlarge` to `db.r7g.8xlarge`, the size the register's evidence supports), each in its own change set, and the old class stays one commit away:

```typescript
import * as ec2 from "aws-cdk-lib/aws-ec2";
import * as rds from "aws-cdk-lib/aws-rds";

const cluster = new rds.DatabaseCluster(this, "PerfDb", {
  engine: rds.DatabaseClusterEngine.auroraPostgres({ version:
rds.AuroraPostgresEngineVersion.VER_16_4 }),
  vpc,
  writer: rds.ClusterInstance.provisioned("writer", {
    instanceType: ec2.InstanceType.of(ec2.InstanceClass.R7G, ec2.InstanceSize.XLARGE8), //
change set 3; was r6i.8xlarge
  }),
  readers: [
    rds.ClusterInstance.provisioned("reader1", {
      instanceType: ec2.InstanceType.of(ec2.InstanceClass.R7G, ec2.InstanceSize.XLARGE8), //
change set 1; was r6i.8xlarge
      promotionTier: 1,
    }),
  ],
  storageEncrypted: true,
});
```

Changing the instance class of an Aurora writer interrupts it for about a minute, so the order is fixed. Resize one instance per change set: (1) deploy the reader change and let it soak; (2) fail the writer over to that reader, which is already the new class, with `aws rds failover-db-cluster --db-cluster-identifier <cluster> --target-db-instance-identifier <reader1>`; (3) deploy the writer change, which now resizes a reader. A database that is resized by cluster still has one owner, one window, and one rollback, and each step can be undone by reverting its own commit. For a Multi-AZ RDS instance, the standby is changed first and the instance fails over once. After each step, check the buffer cache hit ratio against the evidence column (`BufferCacheHitRatio` on Aurora, the Performance Insights cache counters on RDS). It should stay near the figure in the register over the next 14 days, and a rise in `ReadIOPS` or any `SwapUsage` is the signal to go back a size.

Graviton readiness

Graviton instances are Arm (`arm64`), so every image, AMI, and binary must exist in an `arm64` build. Readiness is an inventory question before it is a performance one, and answering it takes three checks per workload.

1. **Compute Optimizer's migration effort.** For each instance, Compute Optimizer reports an Arm option with a `migrationEffort` and the `platformDifferences` it found. By default it recommends only the current CPU vendor and architecture, so the call asks for Arm options with `cpuVendorArchitectures=AWS_ARM64`; the filter keeps every option that carries a migration effort, whatever the Graviton family:

```bash
aws compute-optimizer get-ec2-instance-recommendations --region us-east-1 \
  --recommendation-preferences cpuVendorArchitectures=AWS_ARM64 \
  --query 'instanceRecommendations[].
{id:instanceArn,current:currentInstanceType,options:recommendationOptions[?migrationEffort].
{type:instanceType,effort:migrationEffort,diff:platformDifferences}}'
```

2. **Container images.** An image is Graviton-ready if its manifest list includes `arm64`. This lists the platforms of every image in a repository:

```bash
for tag in $(aws ecr list-images --repository-name <repository> --query 'imageIds[].imageTag'
--output text); do
  platforms=$(aws ecr batch-get-image --repository-name <repository> --image-ids
imageTag="$tag" \
    --query 'images[0].imageManifest' --output text | python3 -c "import json,sys;
m=json.load(sys.stdin); print(',.join(sorted({p['platform']['architecture'] for p in
m.get('manifests', [])})) or 'single-arch')")
  echo "$tag $platforms"
done
```

3. **AMIs and native dependencies.** `aws ec2 describe-images --image-ids <ami-id> --query 'Images[].Architecture'` returns `x86_64` for an image that needs rebuilding. Packages with compiled extensions (some Python wheels, Node native modules, JNI libraries) need `arm64` builds; the build log of a trial image shows it.

The result is a short list per repository of what blocks the move. An instance where one of these checks turns up a blocker stays on x86, and its saving drops out of the Graviton total before the change is scheduled.

Build every image for both architectures

Build natively on each architecture and publish one manifest list, so a node of either kind pulls the right image. This GitHub Actions workflow builds on an x86 and an Arm runner and merges the results; the Arm runner can be a Graviton instance on the CI fleet:

```yaml
name: build-image
on:
  push:
    branches: [main]
permissions:
  id-token: write   # lets the job exchange its GitHub token for the AWS role
  contents: read
jobs:
  build:
    strategy:
      matrix:
        include:
          - arch: amd64
            runner: [self-hosted, linux, x64]
          - arch: arm64
            runner: [self-hosted, linux, arm64]
    runs-on: ${{ matrix.runner }}
    steps:
      - uses: actions/checkout@v4
      - uses: docker/setup-buildx-action@v3
      - uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::<account-id>:role/<ecr-push-role>
          aws-region: us-east-1
      - uses: aws-actions/amazon-ecr-login@v2
      - uses: docker/build-push-action@v6
        with:
          context: .
          platforms: linux/${{ matrix.arch }}
          push: true
          tags: <account-id>.dkr.ecr.us-east-1.amazonaws.com/<repository>:${{ github.sha }}-$
{{ matrix.arch }}
  manifest:
    needs: build
    runs-on: [self-hosted, linux, x64]
    steps:
      - uses: docker/setup-buildx-action@v3
      # A login does not carry over from the build job, so the manifest job signs in again.
      - uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::<account-id>:role/<ecr-push-role>
          aws-region: us-east-1
      - uses: aws-actions/amazon-ecr-login@v2
      - run: |
          docker buildx imagetools create \
            --tag <account-id>.dkr.ecr.us-east-1.amazonaws.com/<repository>:${{ github.sha }} \
```

```
            <account-id>.dkr.ecr.us-east-1.amazonaws.com/<repository>:${{ github.sha }}-amd64 \
            <account-id>.dkr.ecr.us-east-1.amazonaws.com/<repository>:${{ github.sha }}-arm64
```

Pin each action to the commit SHA your supply-chain policy requires. The `<ecr-push-role>` is the OIDC role your pipelines already assume to push images, and it needs `ecr:GetAuthorizationToken` plus the layer and image push actions on the repository. Emulating Arm on an x86 runner with QEMU also works for small images, and it is several times slower for large ones.

Examples: where the change goes

EKS nodes: change the NodePool, not the node

Nodes in an EKS cluster are cattle. The note on those rows reads "change the Karpenter NodePool or node group, not the node," because a node you resize by hand returns to its old size at the next replacement. In `fleet-infra`, a Graviton NodePool and its node class look like this (the Karpenter v1 API):

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: general-arm64
spec:
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      requirements:
        - key: kubernetes.io/arch
          operator: In
          values: ["arm64"]
        - key: karpenter.k8s.aws/instance-family
          operator: In
          values: ["m7g", "c7g", "r7g"]
        - key: karpenter.k8s.aws/instance-size
          operator: In
          values: ["xlarge", "2xlarge", "4xlarge"]
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["on-demand"]
      taints:
        - key: arch
          value: arm64
          effect: NoSchedule
      expireAfter: 720h
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 5m
  limits:
```

```
    cpu: "2000"
---
apiVersion: karpenter.k8s.aws/v1
kind: EC2NodeClass
metadata:
  name: default
spec:
  role: <karpenter-node-role-name>
  amiSelectorTerms:
    - alias: al2023@latest
  subnetSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
  securityGroupSelectorTerms:
    - tags:
        karpenter.sh/discovery: <cluster-name>
```

For a cluster that uses managed node groups, the same change is two lines of CDK:

```typescript
import * as ec2 from "aws-cdk-lib/aws-ec2";
import * as eks from "aws-cdk-lib/aws-eks";

cluster.addNodegroupCapacity("GeneralArm64", {
  instanceTypes: [new ec2.InstanceType("m7g.2xlarge")],
  amiType: eks.NodegroupAmiType.AL2023_ARM_64_STANDARD,
  taints: [{ key: "arch", value: "arm64", effect: eks.TaintEffect.NO_SCHEDULE }],
  minSize: 3,
  maxSize: 30,
});
```

Run the Arm pool beside the old pool. The `arch=arm64:NoSchedule` taint on the Arm pool keeps every workload off it until the workload opts in, because once an Arm pool exists the scheduler, and Karpenter, which prefers the cheaper Graviton capacity, will otherwise put an amd64-only pod on an Arm node, where it fails with `exec format error`. Before anything moves, pin each workload whose image is not multi-arch to the old pool with `nodeSelector: {kubernetes.io/arch: amd64}`. Then give a workload that passed the image check in the previous section a toleration for the taint (`key: arch, operator: Equal, value: arm64, effect: NoSchedule`) and a `kubernetes.io/arch: arm64` selector, and move it. Drain the old pool only when no workload is left on it.

Valkey in place of Redis OSS

22 ElastiCache clusters move to Valkey on Graviton ($213,937 a year). Node-based Valkey is 20% cheaper than Redis OSS (for example, `cache.r7g.large` is $0.1752 an hour against $0.219), and the Redis OSS 5 clusters leave the paid Extended Support premium of 80% (160% in year 3). Existing Redis OSS reserved nodes apply to Valkey nodes of the same family and Region. The upgrade is in place, keeps the endpoint, and fails over in 30 to 60 seconds:

```bash
aws elasticache create-snapshot --replication-group-id <replication-group-id> --snapshot-name
<id>-pre-valkey --region <region>
aws elasticache modify-replication-group --replication-group-id <replication-group-id> \
  --engine valkey --engine-version 9.0 --apply-immediately --region <region>
aws elasticache describe-replication-groups --replication-group-id <replication-group-id> \
  --query 'ReplicationGroups[0].[Status,Engine,EngineVersion]' --region <region>
```

If a custom parameter group is attached, add `--cache-parameter-group-name <valkey-parameter-group>` with a Valkey group that keeps the same static parameter values. A single-node cluster (cluster mode disabled) must first be added to a replication group. The AWS CLI must be v2.18.2 or later.

Rollback: AWS supports a rollback only from Valkey 7.2 to Redis OSS 7.1, so the real rollback is the snapshot above: `aws elasticache create-replication-group --replication-group-id <new-id> --snapshot-name <id>-pre-valkey ...` restores the data into a Redis OSS cluster of the old version, at the cost of a new endpoint. Rehearse it once in non-production.

MSK brokers and OpenSearch to Graviton

```bash
aws kafka update-broker-type --cluster-arn <cluster-arn> --current-version <current-version> \
  --target-instance-type kafka.m7g.xlarge --region <region>

aws opensearch update-domain-config --domain-name <domain> --region <region> \
  --cluster-config InstanceType=r7g.large.search,InstanceCount=<count> \
  --ebs-options EBSEnabled=true,VolumeType=gp3,VolumeSize=<gib>
```

MSK replaces one broker at a time, with no data loss. An OpenSearch instance-type change is different: it runs as a blue/green deployment, so AWS briefly stands up a second set of data nodes beside the first, which needs free IP addresses in the domain's subnets and shifts shards across to the new nodes. Do both in a quiet period, and watch under-replicated partitions (MSK) and cluster health and the shard relocation count (OpenSearch) until each update finishes.

Kinesis shards: size them to the peak

32 provisioned Kinesis streams carry more shards than their peak needs ($105,858 a year). A shard accepts 1 MB or 1,000 records a second on write and serves 2 MB a second on read, and every shard bills by the hour whether it is busy or not. Size a stream to its **peak** minute, not its average, plus the headroom its owner chooses for growth:

```bash
# One-minute data is kept for 15 days, so pull 14 days of it. GetMetricStatistics returns at most 1,440 points a call;
# GetMetricData returns up to 100,800, which covers 14 days of one-minute points (20,160) in one call.
aws cloudwatch get-metric-data --region <region> --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"}},
    {"Id": "records", "MetricStat": {"Metric": {"Namespace": "AWS/Kinesis", "MetricName": "IncomingRecords",
      "Dimensions": [{"Name": "StreamName", "Value": "<stream>"}]}, "Period": 60, "Stat": "Sum"}}]' \
  --query 'MetricDataResults[].[Id, max(Values)]' --output text
# bytes: divide by 60 for bytes a second, then by 1,048,576 for MB a second. records: divide by 60 for records a second.
aws kinesis update-shard-count --stream-name <stream> --target-shard-count <shards> \
  --scaling-type UNIFORM_SCALING --region <region>
```

The target is the larger of peak MB a second and peak thousands of records a second, rounded up, plus headroom. Fourteen days of one-minute data is what CloudWatch keeps; to cover a month, repeat the pull two weeks later and keep the larger peak. One call can halve a stream or double it and no more, so a large reduction takes several calls, each after the last finishes. **Verify** that `WriteProvisionedThroughputExceeded` stays at zero for a week. **Roll back** by raising the shard count with the same call. In `fleet-infra` set the stream's `shardCount` to the same number in the same pull request, or the next deploy puts the old count back.

Lambda on Arm

272 functions ($126,742 a year) move to `arm64` at 20% lower duration price: $0.0000133334 per GB-second against $0.0000166667. Only functions with no x86-only native dependency qualify, and the dependency check is the one in the previous section. A function with **provisioned concurrency** is priced at the allocation rate for the concurrency it holds, which is a separate per-GB-second rate from the duration price above and is billed whether or not the function is invoked. Arm lowers that rate too, and the register prices those functions on it, not on the duration rate.

```typescript
import * as lambda from "aws-cdk-lib/aws-lambda";

new lambda.Function(this, "TelemetryEnrich", {
  runtime: lambda.Runtime.PYTHON_3_13,
  architecture: lambda.Architecture.ARM_64, // was X86_64
  handler: "app.handler",
  code: lambda.Code.fromAsset("lambda/telemetry-enrich"),
});
```

How a rightsizing change ships, and how it rolls back

1. **Non-production first,** then production through your change process, with a 14-day soak in between so the same 95th percentile is measured again on the smaller size.
2. **One canary, then the rest.** Resize or replace one instance of a group, compare p95 CPU, memory, and latency for a day, and continue only if they hold.
3. **Every change records the previous type.** The register's `current` column is the rollback target. For an instance resize: `aws ec2 stop-instances`, `aws ec2 modify-instance-attribute --instance-id <id> --instance-type "{\"Value\": \"<previous-type>\"}"`, `aws ec2 start-instances`. A move between x86 and Arm cannot be done in place, so a launch template or node pool carries both versions and the rollback is to the earlier one.
4. **Databases and caches** are rolled back by the failover and snapshot paths above, and are changed in a maintenance window.
5. **Stop on a regression.** If p95 latency or error rate on the workload moves outside its own baseline, the change is reverted first and discussed after.

Deliverable 4: the commitment plan

StrataForge holds five commitments today, and the biggest decision in the engagement is what replaces the ones that end first. We recommend two new one-year plans (the Database plan in two purchases) and a resize of the Compute plans as they renew, **$2.4 million a year** in all:

- **A one-year Database Savings Plan** (C-0001, $215,890 a year): $25 an hour in week four, and a second one-year plan of $81 an hour when the Aurora reservations end, $106 an hour in all. A Savings Plan cannot be changed once it is bought, so the increase is a second purchase.
- **A one-year SageMaker Savings Plan** (C-0003, $133,969 a year): $61 an hour for the always-on inference endpoints.
- **Resized Compute Savings Plans** (C-0002, $2,100,001 a year): two three-year plans, $530 an hour starting November 1, 2026 and $210 an hour starting March 1, 2027, for $740 an hour in total, against the $590 an hour StrataForge holds today. The r5 and Aurora reservations are not renewed, and the SAP HANA reservations are.

Each line is a measured step, not a remainder: the post-roadmap estate is priced with today's commitments, then with each purchase added in turn, and the difference is that line's saving (How the sizes were found). Compared with renewing every existing commitment as it is, the

recommended set saves **$175,000 a month** ($2.1 million a year) in the steady state after the roadmap. Vantalect does not purchase commitments. Ravi Menon approves each one and Elena Vasquez's team buys it in the management account.

What StrataForge holds today

The existing commitments, with their utilization once the roadmap's week 3 to 5 changes are in place.

Commitment	Type and term	Ends	Size	Monthly fee	Utilization at week six
`sp-1yr-2025`	Compute Savings Plan, one year, No Upfront	October 31, 2026	$380 an hour	$277,400	100%
`ri-r5-8xl`	EC2 Reserved Instances, one year, No Upfront	November 30, 2026	12 x r5.8xlarge	$11,125	100%
`ri-aurora`	RDS Reserved Instances, one year, No Upfront	January 14, 2027	6 x db.r5.4xlarge	$6,614	100%
`sp-3yr-2024`	Compute Savings Plan, three years, No Upfront	February 28, 2027	$210 an hour	$153,300	100%
`ri-hana`	EC2 Reserved Instances, three years, Partial Upfront	May 31, 2027	2 x u-6tb1.112xlarge	$23,794	100%

The existing plans stay fully used after the roadmap's changes, so there is no wasted commitment to unwind. The decision is about renewal. Today's Compute Savings Plans total $590 an hour ($380 ending October 31, 2026 and $210 ending February 28, 2027) and cover 34% of the On-Demand Linux EC2 pool. The post-roadmap estate wants $740 an hour. 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.

How the sizes were found

We simulated the estate hour by hour after the roadmap, with the plans' own rules: a Compute Savings Plan applies first to the eligible usage with the highest discount, an hour's unused commitment is lost, and a reservation applies before a plan. Each commitment is the size that minimizes the net monthly bill, searched across sizes, with the other commitments held in place. The steps are measured one after another on the same estate, so no line is what is left over:

1. The post-roadmap estate with **today's commitments held as they are** is the starting price, $2,717,960 a month before Support.

2. Adding the **Database Savings Plan** gives C-0001.
3. Adding the **SageMaker Savings Plan** gives C-0003.
4. **Resizing the Compute Savings Plans and the reservations as they renew** gives C-0002, and ends at the recommended set's $2,513,805 a month.

Five decisions fall out:

- **Compute Savings Plans, not EC2 Instance Savings Plans.** A Compute Savings Plan applies across instance families, sizes, Regions, operating systems, Fargate, and Lambda, so the move to Graviton (a different family) does not strand it. EC2 Instance plans are deeper discounts per family, and they would lock StrataForge into the families the roadmap is moving it out of.
- **Three years, No Upfront, for Compute.** On the post-roadmap estate, a three-year plan costs 51.4% of On-Demand on average for the instances it covers, against 74.3% for a one-year plan. **The break-even utilization is the same figure:** a three-year plan beats paying On-Demand if at least 51% of its committed hours are used, and a one-year plan if at least 74% are. The simulation sizes the plan to a floor StrataForge never drops below, so the three-year plan has wide margin. No Upfront keeps cash in StrataForge's hands, and the discount for paying upfront is small next to the risk of prepaying.
- **A Database Savings Plan, one year.** Database Savings Plans, launched December 2, 2025, are a dollars-an-hour commitment for one year with no upfront payment, applied across engines, instance families, sizes, and Regions. They cover Aurora, RDS, DynamoDB, ElastiCache (Valkey only), DocumentDB, Neptune, Keyspaces, Timestream (InfluxDB), DMS, and OpenSearch Service, and only the latest instance generations. That is why the roadmap's order matters: the plan is sized to the generation 7 Graviton databases that exist after week four, not to the instances running today. Reserved instances and Database Savings Plans cannot stack on the same usage, so the Aurora reservations that end January 14, 2027 are not renewed.
- **A SageMaker Savings Plan, one year.** The real-time inference endpoints run every hour of the month, so a plan sized to the endpoints that remain after the idle ones are deleted is never left with an unused hour. It is bought at its full size, $61 an hour, on the first day.
- **The pools left out make every plan smaller, not larger.** The simulation was 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 come out smaller than the full estate supports, never larger. Add those pools at the March 2027 tranche.

What each plan commits StrataForge to

A Savings Plan is a promise to spend a dollar amount an hour for its whole term. The table is the total, not the hourly rate, because the total is what finance approves.

The total each plan commits StrataForge to, which is the amount finance approves. The second Database row is the total of both Database plans once the Aurora reservations end (the second plan is $81 an hour); it includes the first row's commitment.

Plan	Term	Per hour	Total committed
Database Savings Plan, bought in week four	One year	$25	$219,000
Both Database Savings Plans together, at full roadmap size	One year	$106	$928,560
SageMaker Savings Plan	One year	$61	$534,360
Compute Savings Plan, starting November 1, 2026	Three years	$530	$13,928,400
Compute Savings Plan, starting March 1, 2027	Three years	$210	$5,518,800
Compute Savings Plans together	Three years	**$740**	**$19,447,200**

The recommendation, and its alternatives

Five ways to handle the renewals, priced on the post-roadmap estate at net prices.

Choice	Steady-state monthly bill, before Support	Compared with the recommendation, per month	Per year
Recommended: $106 an hour of Database, $61 of SageMaker, and $740 of Compute Savings Plans	$2,513,805		
Like for like: renew every commitment as it is today, and buy the two new plans	$2,688,805	$175,000	$2,100,001
One-year Compute plan of $985 an hour instead of the three-year plans	$2,690,354	$176,549	$2,118,586
Lapse: let every plan end except the SAP reservations, and buy the two new plans	$2,910,603	$396,797	$4,761,570
Hold: keep today's commitments and buy nothing new	$2,717,960	$204,155	$2,449,861

Three readings of this table matter. First, **renewing as it is costs more than the recommendation**, because the renewals would be sized for the estate StrataForge has today. The roadmap's commitment lines ($2.4 million a year) are measured against holding what exists, one step at a time. Second, **letting everything lapse is worse still:** without commitments the post-roadmap estate pays On-Demand rates for usage that never goes away. Third, **the one-year Compute plan is the cautious alternative, and its price is plain.** It commits $8.6 million instead of $19.4 million, and costs $176,549 a month more, because one-year rates are shallower. Avoiding the extra $10.8 million of commitment therefore costs $2.1 million a year.

If usage turns out smaller than the simulation

A three-year plan is only wrong if StrataForge uses fewer hours than it committed. Two tests show how far wrong.

A smaller three-year plan is used in full, and costs more each month than the recommended size.

Three-year Compute plan sized at	Per hour	Steady-state monthly bill	Extra cost per month against the recommended size	Utilization
80% of the recommended size	$590	$2,555,061	$41,256	100%
90% of the recommended size	$665	$2,533,323	$19,518	100%
The recommended size	$740	$2,513,805		100%

The first test is the cost of buying less: sizing the plan at 90% leaves it fully used and costs $19,518 a month more, which is the price of insurance against a smaller estate. The second is a stress test: if On-Demand Linux hours fall 20% below the roadmap estate, the three-year plan is used 83.4% of its hours, still above the 51% at which it beats On-Demand, and the other plans stay at 100%. Because the March 2027 tranche is bought last and the pools left out can be added to it, the plan can be extended later and cannot be shrunk, so we size it to the floor and not to the average.

The Database Savings Plan is bought in two plans. Its full-roadmap size is $106 an hour. By week four only part of the Graviton database work is done, so we recommend buying $25 an hour then, and a second plan of $81 an hour as the 74 database changes and 22 cache moves land and the Aurora reservations end. A plan cannot be modified after it is bought, so the second one is a new purchase, queued like the first.

The calendar

The commitment calendar. Each row is an action with a date, and an owner is named in the roadmap.

Date	What happens	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

Two dates are close together and easy to get wrong. The one-year plan ends at the end of October 31, 2026 (UTC), and the first queued plan starts at the beginning of November 1, 2026 (UTC), so there is no hour of uncovered usage. In U.S. Central time, that switch is at 7:00 p.m. on October 31.

How to queue a purchase

A Savings Plan purchase can be queued for a future date, and a queued purchase can be deleted until it starts. That is how we recommend buying: queue it after finance approves, with every approval on record, so that nobody needs to be at a keyboard at midnight.

```bash
# 1. Find the offering for the type, term, and payment option. Three years is 94608000 seconds.
aws savingsplans describe-savings-plans-offerings --region us-east-1 \
  --plan-types Compute --payment-options "No Upfront" --durations 94608000 --currencies USD \
  --query 'searchResults[].
{id:offeringId,type:planType,term:durationSeconds,payment:paymentOption}'

# 2. Queue the purchase for the start of November 1, UTC, in the management account.
aws savingsplans create-savings-plan --region us-east-1 \
  --savings-plan-offering-id <offering-id> \
  --commitment 530 \
  --purchase-time 2026-11-01T00:00:00Z \
  --client-token sf-compute-sp-nov-2026 \
  --tags '{"sf:approved-by": "<approver>", "sf:change-ticket": "<ticket-id>"}'

# 3. Confirm it is queued, and delete it if the decision changes before it starts.
aws savingsplans describe-savings-plans --region us-east-1 --states queued
aws savingsplans delete-queued-savings-plan --region us-east-1 --savings-plan-id <savings-plan-
id>
```

The Database Savings Plan uses the same calls with `--plan-types Database`, `--durations 31536000`, and `--commitment 25`. The SageMaker Savings Plan uses `--plan-types SageMaker`, `--durations 31536000`, and `--commitment 61`. Before each purchase, check the commitment against AWS's own recommendation, which is made from your recent usage and so ignores the roadmap; it should come out lower than ours, and the difference is the roadmap:

```bash
aws ce get-savings-plans-purchase-recommendation --region us-east-1 \
  --savings-plans-type COMPUTE_SP --term-in-years THREE_YEARS --payment-option NO_UPFRONT \
  --lookback-period-in-days SIXTY_DAYS
```

Use `SAGEMAKER_SP` and `ONE_YEAR` for the SageMaker plan. AWS lets you return a plan only within narrow limits, so treat every purchase as final. The approval record for each purchase names the roadmap wave that has to be complete before it is queued; if that wave slips, the queued purchase is deleted and re-queued.

The private pricing floor

This is the part of the report that CFOs ask about first, so it is here and in the readout. **StrataForge's private pricing agreement has an annual spend floor, and the roadmap takes spend under it in contract year 3 at today's usage.** We assume the commitment counts spend net of the private pricing discount, Savings Plans and reservation fees included, so Savings Plans and reservations count toward it. Buying more No Upfront plans does not close the gap at any sensible size: a plan that is used in full lowers the bill, and one bigger than the usage it covers closes the gap only by paying for hours nobody uses. Prepaying is different: an All Upfront or Partial Upfront payment is spend in the month AWS bills it, and whether it counts toward a contract year depends on the agreement. We do not recommend prepaying to meet a floor, for the reason at the end of this section.

The private pricing agreement by contract year. Year 3 begins February 1, 2027 and is projected at today's usage after the roadmap.

Contract year	Period	Commitment	Projected spend, net	Result	How we know
1	February 2025 to January 2026	$36,000,000	$36,449,975	Above the commitment by $449,975	estimated: the CUR export starts in August 2025
2	February 2026 to January 2027	$42,000,000	$44,976,749	Above the commitment by $2,976,749	six months actual, six projected with the roadmap
3	February 2027 to January 2028	$42,000,000	$31,924,056	**$10,075,944 below**	projected at today's usage

The floor is a promise to spend. When spend falls short, we assume AWS bills the $10.1 million difference as a true-up (many agreements do, and this one is unread), so for that year the roadmap reduces the bill by less than it identifies. The next table shows both ways the year could go.

Contract year 3 under two usage paths. "Growing" is the pace of the last six months carried through the year, and it is a scenario, not a forecast.

Contract year 3	Usage flat at today's level	Growing 3.6% a month
Spend without the roadmap	$55,715,103	$84,307,253
Spend with the roadmap	$31,924,056	$48,195,008
Reduction in spend	$23,791,047	$36,112,245
True-up billed under the $42.0 million commitment	$10,075,944	$0
Realizable	**$13,715,103**	**$36,112,245**

Four points put this in proportion.

The roadmap still wins. Even at today's usage, with the true-up paid in full, the roadmap leaves StrataForge $13.7 million better off in contract year 3. A shortfall is a cost of having saved money, not a reason to stop.

A flat bill is not the likely case. If usage keeps growing at the last six months' pace, year-3 spend clears the commitment with no true-up and the whole reduction ($36.1 million, measured against a bill that would also have grown) is realizable. We do not rely on that pace holding: it is shown so the two ends are visible.

The time to talk is now, while StrataForge is growing and spending $55.7 million a year. A customer who asks for a different shape before the shortfall exists is in a stronger position than one who is explaining it afterward. Resetting the commitment to the flat projection, $31.0 million, removes the true-up and makes the whole $23.8 million realizable, but a lower commitment may cost discount points:

Realizable savings in contract year 3 with the commitment reset to the flat projection, at three discount levels. A lower discount raises the bill, so realizable savings fall.

Discount after renegotiation	Commitment	Contract-year-3 spend	Realizable
7%	$31,000,000	$31,924,056	$23,791,047
6%	$31,000,000	$32,254,851	$23,460,252
5%	$31,000,000	$32,585,645	$23,129,457

Each point of discount traded for the lower commitment costs about $330,795 a year, and even at 5% the realizable figure ($23.1 million) beats the $13.7 million that the current commitment leaves. About 96.4% of the bill is eligible for the discount.

Why the reduction is not the identified figure. The reduction in contract-year-3 spend ($23,791,047) differs from the identified savings ($23,520,211) because they measure different things. The reduction is the change in what AWS bills in that one year, with the roadmap taken as planned; the identified figure is the sum of the opportunities at the August run-rate.

From the identified savings to the year-3 reduction at today's usage. Nothing here is a decline yet: the projection takes every line on the first day of its wave.

Step	Per year
Identified savings, the sum of the opportunities	$23,520,211
Enterprise Support falls with the bill	$705,742
Coverage effect between the lines	$4,519
Lines not in effect for all of contract year 3: the backlog is assumed in place by the end of March 2027, and a line that is declined later drops out the same way	−$439,425
Reduction in contract-year-3 spend	**$23,791,047**

When the owners' decisions are in (week 6), the same bridge is rebuilt with the declines and real dates.

We have not read the contract. We worked from the summary StrataForge gave us. Whether the commitment is measured before or after the discount, what spend counts toward it, how a shortfall is billed, and whether an overage in one year carries into the next are questions for Ravi Menon's team and AWS counsel. The answers move the number.

What to ask AWS for, in the order we would ask:

- **Reset contract year 3 toward $31.0 million:** a lower annual figure, or a ramp (a lower commitment in the first six months). Ask for it as an amendment, and expect that AWS may not grant one: a customer who wants a lower floor is usually offered a longer term or a new agreement in exchange.
- **Early renewal with a longer term.** If AWS will not amend year 3, the usual trade is to renew early into a new agreement with a longer term and a floor sized to the bill after the roadmap. The cost is plain: StrataForge is committed for longer, at whatever discount the new agreement carries, so the discount table above is the number to hold the negotiation to.

- **Route existing third-party software through AWS Marketplace.** Software StrataForge already buys from other vendors can be bought through AWS Marketplace, and Marketplace spend counts toward the commitment up to a cap the agreement sets. It adds no new spend, only moves spend that exists, so it is the one lever that closes part of a gap without paying for anything nobody uses. How much it can move depends on the cap and on the vendors' contracts, which we have not seen.
- **Confirm what counts toward the commitment:** spend net of the private pricing discount, Savings Plans and reservation fees included, plus Support, Marketplace purchases, and spend by the Kestrel Vision accounts that joined in November 2025.
- **Ask about carry-forward.** If contract year 2 spend above the commitment ($3.0 million) can count toward year 3, the timing of the savings becomes worth money, and we would sequence the roadmap differently.

If AWS declines. The roadmap still pays: at today's usage the realizable figure is $13.7 million in contract year 3, with the $10.1 million true-up paid in full. Plan the budget on that column, keep the roadmap on schedule, and ask again at the next renewal with a year of measured spend to show. Every dollar of growth above the flat path reduces the true-up, and the growth scenario shows the true-up gone.

What we do not recommend: buying more commitments than the estate uses to fill the gap, prepaying only to meet the floor, or slowing the roadmap. Each spends real money to avoid a payment that is no larger than the saving it protects.

Deliverable 5: the architecture review

Architecture changes, storage tiers, schedules, and logging scope are worth **$13.4 million a year**: $6.9 million from data paths and services, $4.6 million from storage, and $2.0 million from schedules. These are the changes where the price is set by how the system is built, not by how big its machines are. Each one below has its evidence, the change as code or a command, how to verify it, and how to roll it back.

Architecture, storage, and schedule changes by kind.

Change	Count	Per month	Per year	One-time cost
S3 lifecycle and storage classes	145	$190,055	$2,280,657	$87,182
Instances off nights and weekends	415	$130,628	$1,567,541	$0
Simulation on Spot through AWS Batch	334	$118,544	$1,422,523	$0
S3 Intelligent-Tiering	161	$111,837	$1,342,048	$28,433

Change	Count	Per month	Per year	One-time cost
Robot telemetry to S3 through Firehose	96	$68,167	$818,009	$0
Per-robot metrics to site metrics	4	$67,895	$814,735	$0
Ephemeral CI runners on Spot	143	$55,162	$661,942	$0
VPC endpoints instead of NAT	53	$52,872	$634,466	$0
CloudTrail data events scoped	1	$49,952	$599,428	$0
IoT Basic Ingest for telemetry	4	$41,153	$493,835	$0
ECR lifecycle policies	2	$34,009	$408,108	$0
Databases off nights and weekends	70	$32,945	$395,341	$0
FSx for Lustre throughput tier	1	$25,998	$311,980	$0
Database engines out of Extended Support	49	$18,678	$224,137	$0
Kafka consumers read in their own AZ	6	$16,762	$201,148	$0
SQL Server Enterprise to Standard	6	$15,235	$182,814	$0
CloudWatch log retention	692	$12,395	$148,740	$0
DynamoDB to on-demand	38	$11,981	$143,768	$0
EBS gp2 and io1 to gp3	2,362	$11,541	$138,492	$0
Device shadows on change	4	$9,833	$117,992	$0
AWS Config recording scoped	2	$8,788	$105,462	$0
Aurora I/O-Optimized	8	$7,783	$93,397	$0
AWS Backup cold storage	2	$7,082	$84,988	$0
All update downloads through CloudFront	4	$5,308	$63,697	$0
Kubernetes out of extended support	15	$5,092	$61,101	$0
Debug logs to Infrequent Access	45	$4,789	$57,474	$0
Service calls stay in their AZ	1	$2,883	$34,596	$0
WorkSpaces billed by the hour	1	$2,854	$34,247	$0
Public IPv4 addresses removed	88	$299	$3,585	$0

The 30 largest changes

The 30 architecture, storage, and schedule changes with the largest annual savings.

ID	Account	Change	What the evidence says	Owner and repository	Wave	Risk	Per month	Per year
A-0161	sf-management	Scope CloudTrail data events with advanced event selectors	94% of data events are reads of the telemetry and sensor lakes by the platform's own roles. Agreed with the CISO: the security engagement's detection coverage is unchanged	Elena Vasquez, `platform-landing-zone`	Week 5	Low	$49,952	$599,428
A-0157	fleet-ingest-prod	Stop publishing one CloudWatch metric per robot	dashboards and alarms use site-level aggregates; per-robot series are queried in the lake	Sam Okafor, `fleet-infra`	Week 5	Medium	$34,474	$413,688
T-3444	sf-cicd	Add ECR lifecycle policies	72% of image bytes are untagged builds older than 30 days	Elena Vasquez, `platform-landing-zone`	Week 3	Low	$28,250	$339,005
A-0170	crucible-training-prod	Lower the FSx for Lustre throughput tier to what training reads	peak read throughput 96 MB/s per TiB over 30 days, including checkpoint restores. Persistent 2 file systems change throughput in place; schedule it between training runs	Hiro Tanaka, `crucible-infra`	Week 5	Medium	$25,998	$311,980
A-0162	fleet-ingest-prod	Publish telemetry through IoT Basic Ingest	telemetry topics have no subscribers other than the rule. Needs robot firmware 7.4, which ships through OTA in November; the 3,412 ForgeArm gen-2 robots on firmware 3.x keep standard topics until their own update	Sam Okafor, `fleet-infra`	Backlog	Medium	$22,760	$273,124
A-0158	fleet-ingest-prod	Stop publishing one CloudWatch metric per robot	dashboards and alarms use site-level aggregates; per-robot series are queried in the lake	Sam Okafor, `fleet-infra`	Week 5	Medium	$18,052	$216,619

ID	Account	Change	What the evidence says	Owner and repository	Wave	Risk	Per month	Per year
A-0164	fleet-ingest-prod	Publish telemetry through IoT Basic Ingest	telemetry topics have no subscribers other than the rule. Needs robot firmware 7.4, which ships through OTA in November; the 3,412 ForgeArm gen-2 robots on firmware 3.x keep standard topics until their own update	Sam Okafor, `fleet-infra`	Backlog	Medium	$10,187	$122,249
A-0054	fleet-ingest-prod	Let consumers read from the replica in their own Availability Zone	60% of cross-AZ bytes are consumer fetches	Sam Okafor, `fleet-infra`	Week 5	Low	$10,033	$120,394
A-0159	fleet-ingest-prod	Stop publishing one CloudWatch metric per robot	dashboards and alarms use site-level aggregates; per-robot series are queried in the lake	Sam Okafor, `fleet-infra`	Week 5	Medium	$9,054	$108,644
T-2565	crucible-data	Lifecycle policy (`sf-sensor-log-lake-sensor-logs-7065-us-east-1-043`)	S3 Inventory: 77% of bytes older than 90 days, 59% older than 365; access logs: 73% not read in 90 days. One-time transition requests repaid within the first week; includes $952 a month of retrieval on objects still read after they move (server access logs)	Hiro Tanaka, `crucible-infra`	Week 3	Low	$8,649	$103,784
A-0075	fleet-ingest-prod	Write robot telemetry to S3 through Firehose, not to CloudWatch Logs	the telemetry is analyzed in the lake; CloudWatch copies are read only during incidents, by Logs Insights queries	Sam Okafor, `fleet-infra`	Week 5	Medium	$8,328	$99,938
T-2576	crucible-data	Move to S3 Intelligent-Tiering	S3 Inventory: 96% of bytes older than 90 days, 88% older than 365; access logs: 98% not read in 90 days. One-time transition requests repaid within the first week	Hiro Tanaka, `crucible-infra`	Week 3	Medium	$7,899	$94,792
T-2638	kestrel-data	Lifecycle policy (`sf-labeled-datasets-image-datasets-9098-eu-west-1-003`)	S3 Inventory: 93% of bytes older than 90 days, 65% older than 365; access logs: 95% not read in 90 days. One-time transition requests repaid in 2.6 months; includes $280 a month of retrieval on objects still read after they move (server access logs)	Jonah Reyes, `(not in code)`	Week 3	Low	$7,088	$85,053

ID	Account	Change	What the evidence says	Owner and repository	Wave	Risk	Per month	Per year
T-2520	crucible-training-prod	Lifecycle policy (`sf-checkpoints-model-checkpoints-7042-us-east-1-004`)	S3 Inventory: 90% of bytes older than 90 days, 61% older than 365; access logs: 88% not read in 90 days. One-time transition requests repaid within the first week; includes $144 a month of retrieval on objects still read after they move (server access logs)	Hiro Tanaka, `crucible-infra`	Week 3	Medium	$6,940	$83,277
T-2515	crucible-sim-prod	Lifecycle policy (`sf-episode-results-sim-results-7019-us-east-1-009`)	S3 Inventory: 84% of bytes older than 90 days, 73% older than 365; access logs: 86% not read in 90 days	Hiro Tanaka, `crucible-infra`	Week 3	Medium	$6,807	$81,685
A-0160	fleet-ingest-prod	Stop publishing one CloudWatch metric per robot	dashboards and alarms use site-level aggregates; per-robot series are queried in the lake	Sam Okafor, `fleet-infra`	Week 5	Medium	$6,315	$75,784
A-0081	fleet-ingest-prod	Write robot telemetry to S3 through Firehose, not to CloudWatch Logs	the telemetry is analyzed in the lake; CloudWatch copies are read only during incidents, by Logs Insights queries	Sam Okafor, `fleet-infra`	Week 5	Medium	$6,182	$74,188
S-0410	fleet-perf	Scale the node group to zero outside performance-test windows	no pods scheduled outside 7 a.m. to 7 p.m. local time on weekdays	Sam Okafor, `fleet-infra`	Week 3	Low	$6,120	$73,435
T-2617	erp-sap-prod	Lifecycle policy (`sf-hana-backups-backups-8016-us-east-1-001`)	S3 Inventory: 94% of bytes older than 90 days, 52% older than 365; access logs: 96% not read in 90 days. One-time transition requests repaid within the first week	Ana Kowalski, `enterprise-infra`	Week 3	Medium	$6,119	$73,430
T-2369	sf-log-archive	Lifecycle policy (`sf-log-archive-vpc-flow-logs-2291-us-east-1-002`)	S3 Inventory: 97% of bytes older than 90 days, 89% older than 365; access logs: 97% not read in 90 days. One-time transition requests repaid in 1.0 months; includes $17 a month of retrieval on objects still read after they move (server access logs)	Nadia Petrova, `platform-landing-zone`	Week 3	Medium	$6,046	$72,549

ID	Account	Change	What the evidence says	Owner and repository	Wave	Risk	Per month	Per year
T-3447	crucible-models	Add ECR lifecycle policies	72% of image bytes are untagged builds older than 30 days	Hiro Tanaka, `crucible-infra`	Week 3	Low	$5,759	$69,103
A-0094	fleet-ingest-prod	Write robot telemetry to S3 through Firehose, not to CloudWatch Logs	the telemetry is analyzed in the lake; CloudWatch copies are read only during incidents, by Logs Insights queries	Sam Okafor, `fleet-infra`	Backlog	Medium	$5,667	$68,005
A-0163	fleet-ingest-prod	Update device shadows on change, not every minute	86% of shadow updates carry no change. Needs robot firmware 7.4, which ships through OTA in November	Sam Okafor, `fleet-infra`	Backlog	Low	$5,666	$67,988
T-2469	fleet-eu-prod	Lifecycle policy (`sf-telemetry-lake-eu-telemetry-replica-5160-eu-west-1-009`)	S3 Inventory: 93% of bytes older than 90 days, 42% older than 365; access logs: 98% not read in 90 days. One-time transition requests repaid in 0.5 months; includes $25 a month of retrieval on objects still read after they move (server access logs)	Sam Okafor, `fleet-infra`	Week 3	Medium	$5,609	$67,303
S-0408	fleet-perf	Scale the node group to zero outside performance-test windows	no pods scheduled outside 7 a.m. to 7 p.m. local time on weekdays	Sam Okafor, `fleet-infra`	Week 3	Low	$5,546	$66,551
A-0072	fleet-ingest-prod	Write robot telemetry to S3 through Firehose, not to CloudWatch Logs	the telemetry is analyzed in the lake; CloudWatch copies are read only during incidents, by Logs Insights queries	Sam Okafor, `fleet-infra`	Week 5	Medium	$5,467	$65,605
T-2461	fleet-eu-prod	Lifecycle policy (`sf-telemetry-lake-eu-telemetry-replica-5160-eu-west-1-001`)	S3 Inventory: 94% of bytes older than 90 days, 68% older than 365; access logs: 98% not read in 90 days. One-time transition requests repaid in 0.6 months; includes $48 a month of retrieval on objects still read after they move (server access logs)	Sam Okafor, `fleet-infra`	Week 3	Medium	$5,433	$65,199

ID	Account	Change	What the evidence says	Owner and repository	Wave	Risk	Per month	Per year
T-3445	sf-backup	Archive monthly EBS recovery points and move EFS backups to cold storage	45% of EBS and 62% of EFS recovery points are monthly copies older than 90 days	Elena Vasquez, `platform-landing-zone`	Week 3	Low	$5,424	$65,090
T-2458	fleet-eu-prod	Move to S3 Intelligent-Tiering	S3 Inventory: 88% of bytes older than 90 days, 54% older than 365; access logs: 90% not read in 90 days. One-time transition requests repaid in 0.3 months	Sam Okafor, `fleet-infra`	Week 3	Low	$5,223	$62,676
A-0172	sf-cicd	Stop recording ephemeral network interfaces in AWS Config	AWS Config configuration-item counts by resource type, July and August. The SOC 2 evidence the security engagement mapped needs daily state, which periodic recording keeps	Elena Vasquez, `platform-landing-zone`	Week 5	Low	$5,078	$60,934

The network path: NAT gateways and VPC endpoints

Traffic from the spoke VPCs to AWS services leaves through Transit Gateway, Network Firewall, and the NAT gateways in `sf-network-hub`. The hub and the spoke VPCs are priced differently, and the fix is different in each. 53 findings cover NAT traffic that never needed to leave the AWS network ($634,466 a year): 15 on the hub's NAT gateways ($183,630 a year, all of it Transit Gateway and firewall processing) and 38 on NAT gateways in 9 spoke accounts ($450,836 a year).

- **In the hub, the NAT gateways cost nothing to keep.** Network Firewall waives the hours and the per-gigabyte charge of the NAT gateways it inspects. Moving bytes off those gateways saves nothing on the gateways. The cost is on the other two legs: Transit Gateway ($0.02 per GB) and Network Firewall processing ($0.065 per GB) charge for every byte of S3 and DynamoDB traffic that crosses the hub to reach AWS. A gateway endpoint cannot be reached across a Transit Gateway, so the S3 and DynamoDB gateway endpoints go in **each spoke VPC** that sends the traffic, and the hub's NAT gateways stay as they are. The saving is the Transit Gateway and firewall processing those bytes no longer incur. This is rated **Medium** risk and not Low, because the firewall used to inspect this traffic: the endpoint policies below keep that control.
- **In the spokes, the NAT gateway is the cost,** and the endpoint design removes it: gateway endpoints for S3 and DynamoDB, and interface endpoints for the services with steady traffic. Rated Low.

NAT gateway traffic by destination, from VPC flow logs.

Destination	GB a month through NAT, hub and spokes	Share	Where it can go instead
Amazon S3	728,854	38%	S3 gateway endpoint in each spoke VPC (no charge)
Internet	557,378	29%	Stays on NAT: it is real internet traffic
Amazon ECR	225,958	12%	ECR interface endpoints; image layers come from S3 through the gateway endpoint, so its policy allows the ECR layer bucket
CloudWatch Logs	130,833	7%	CloudWatch Logs interface endpoint
Kinesis Data Streams	85,918	5%	Kinesis Data Streams interface endpoint
Other	80,464	4%	Reviewed per destination
Amazon DynamoDB	75,929	4%	DynamoDB gateway endpoint in each spoke VPC (no charge)
AWS STS	15,678	1%	STS interface endpoint

66% of the bytes through NAT are headed for an AWS service. A **gateway endpoint** for S3 and DynamoDB is free and removes 42% of the traffic by itself. **Interface endpoints** cost $0.01 an hour per Availability Zone plus $0.01 per GB, against $0.045 per GB through NAT (and $0.045 an hour per gateway), so they pay for themselves on any service with steady traffic; the design prices each one and adds it only where it does. All of them take the traffic off the Transit Gateway and Network Firewall path. The endpoints go in the spoke VPCs, where the traffic starts:

```typescript
import * as ec2 from "aws-cdk-lib/aws-ec2";
import * as iam from "aws-cdk-lib/aws-iam";
import { Stack } from "aws-cdk-lib";
import { Construct } from "constructs";

export interface ServiceEndpointsProps {
  vpc: ec2.IVpc;
  organizationId: string;
  interfaceServices?: ec2.InterfaceVpcEndpointAwsService[];
}

/** AWS-owned buckets that the S3 endpoint must keep reachable (ECR layers, Amazon Linux,
Systems Manager, EMR). */
const awsOwnedBucketArns = (region: string): string[] => [
  `arn:aws:s3:::prod-${region}-starport-layer-bucket/*`,
  `arn:aws:s3:::al2023-repos-${region}-*/*`,
```

```
  `arn:aws:s3:::amazonlinux-2-repos-${region}/*`,
  `arn:aws:s3:::aws-ssm-${region}/*`,
  `arn:aws:s3:::amazon-ssm-${region}/*`,
  `arn:aws:s3:::amazon-ssm-packages-${region}/*`,
  `arn:aws:s3:::${region}-birdwatcher-prod/*`,
  `arn:aws:s3:::patch-baseline-snapshot-${region}/*`,
  `arn:aws:s3:::repo.${region}.emr.amazonaws.com/*`,
  `arn:aws:s3:::${region}.elasticmapreduce/*`,
];

/** Gateway endpoints for S3 and DynamoDB in every VPC; interface endpoints for the heavy
services. */
export class ServiceEndpoints extends Construct {
  constructor(scope: Construct, id: string, props: ServiceEndpointsProps) {
    super(scope, id);
    const { vpc } = props;

    const s3 = vpc.addGatewayEndpoint("S3", { service: ec2.GatewayVpcEndpointAwsService.S3 });
    const dynamoDb = vpc.addGatewayEndpoint("DynamoDb", { service:
ec2.GatewayVpcEndpointAwsService.DYNAMODB });

    // Endpoint policies (the next section explains them): organization resources only.
    // Adding any statement replaces the default allow-all policy.
    for (const endpoint of [s3, dynamoDb]) {
      endpoint.addToPolicy(new iam.PolicyStatement({
        sid: "AllowOrganizationResources",
        principals: [new iam.AnyPrincipal()],
        actions: ["*"],
        resources: ["*"],
        conditions: { StringEquals: { "aws:ResourceOrgID": props.organizationId } },
      }));
    }
    // The S3 endpoint also needs the AWS-owned buckets that ECR, Amazon Linux, Systems
Manager, and EMR read.
    s3.addToPolicy(new iam.PolicyStatement({
      sid: "AllowAwsOwnedBuckets",
      principals: [new iam.AnyPrincipal()],
      actions: ["s3:GetObject"],
      resources: awsOwnedBucketArns(Stack.of(this).region),
    }));
    // CloudFormation wait conditions (cfn-signal) write to an AWS-owned bucket.
    s3.addToPolicy(new iam.PolicyStatement({
      sid: "AllowCloudFormationSignals",
      principals: [new iam.AnyPrincipal()],
      actions: ["s3:PutObject"],
      resources: [`arn:aws:s3:::cloudformation-waitcondition-${Stack.of(this).region}/*`],
    }));

    const services = props.interfaceServices ?? [
      ec2.InterfaceVpcEndpointAwsService.ECR,
      ec2.InterfaceVpcEndpointAwsService.ECR_DOCKER,
      ec2.InterfaceVpcEndpointAwsService.CLOUDWATCH_LOGS,
      ec2.InterfaceVpcEndpointAwsService.KINESIS_STREAMS,
      ec2.InterfaceVpcEndpointAwsService.STS,
    ];
    for (const service of services) {
      vpc.addInterfaceEndpoint(`Endpoint${service.shortName.replace(/[^A-Za-z0-9]/g, "")}`, {
```

```
        service,
        privateDnsEnabled: true,
        subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
      });
    }
  }
}
```

In `platform-landing-zone` the same endpoints are CloudFormation resources in the account baseline, so every new VPC gets them:

```yaml
S3GatewayEndpoint:
  Type: AWS::EC2::VPCEndpoint
  Properties:
    VpcId: !Ref Vpc
    ServiceName: !Sub "com.amazonaws.${AWS::Region}.s3"
    VpcEndpointType: Gateway
    RouteTableIds: !Ref PrivateRouteTableIds
    PolicyDocument:  # the three statements of the endpoint policy below; without it the
endpoint allows everything
      Version: "2012-10-17"
      Statement:
        - Sid: AllowOrganizationResources
          Effect: Allow
          Principal: "*"
          Action: "*"
          Resource: "*"
          Condition:
            StringEquals: { "aws:ResourceOrgID": !Ref OrganizationId }
        - Sid: AllowAwsOwnedBuckets
          Effect: Allow
          Principal: "*"
          Action: "s3:GetObject"
          Resource:
            - !Sub "arn:aws:s3:::prod-${AWS::Region}-starport-layer-bucket/*"
            - !Sub "arn:aws:s3:::al2023-repos-${AWS::Region}-*/*"
            - !Sub "arn:aws:s3:::amazonlinux-2-repos-${AWS::Region}/*"
            - !Sub "arn:aws:s3:::aws-ssm-${AWS::Region}/*"
            - !Sub "arn:aws:s3:::amazon-ssm-${AWS::Region}/*"
            - !Sub "arn:aws:s3:::amazon-ssm-packages-${AWS::Region}/*"
            - !Sub "arn:aws:s3:::${AWS::Region}-birdwatcher-prod/*"
            - !Sub "arn:aws:s3:::patch-baseline-snapshot-${AWS::Region}/*"
            - !Sub "arn:aws:s3:::repo.${AWS::Region}.emr.amazonaws.com/*"
            - !Sub "arn:aws:s3:::${AWS::Region}.elasticmapreduce/*"
        - Sid: AllowCloudFormationSignals
          Effect: Allow
          Principal: "*"
          Action: "s3:PutObject"
          Resource: !Sub "arn:aws:s3:::cloudformation-waitcondition-${AWS::Region}/*"
LogsInterfaceEndpoint:
  Type: AWS::EC2::VPCEndpoint
  Properties:
    VpcId: !Ref Vpc
    ServiceName: !Sub "com.amazonaws.${AWS::Region}.logs"
```

```
    VpcEndpointType: Interface
    PrivateDnsEnabled: true
    SubnetIds: !Ref PrivateSubnetIds
    SecurityGroupIds: [!Ref EndpointSecurityGroup]
```

Verify with the same flow-log query that found the traffic: bytes to the S3 and DynamoDB prefixes should fall to near zero within a day, through the NAT gateway in a spoke and across the Transit Gateway attachments in the hub. The `NAT Gateway` lines (spokes), and the `Transit Gateway` and `Network Firewall` lines (hub) in the next day's Cost and Usage Report should drop by the amount in the roadmap. **Roll back** by deleting the endpoint; traffic returns to the NAT path on its own, because the NAT route is still in the route table (and, in a spoke behind the hub, to the Transit Gateway path through the hub). Order the work by gateway endpoints first (free, no risk), then the interface endpoints in the busiest VPCs.

Endpoint policies keep the control the firewall gave

The firewall in the hub used to see every byte to S3 and DynamoDB, and could stop a copy to a bucket in someone else's account. An endpoint with the default policy allows any principal to reach any bucket, so a gateway endpoint without a policy would trade a control for a saving. Each endpoint carries a policy that allows only resources owned by StrataForge's AWS organization, using the `aws:ResourceOrgID` condition. Bytes to a bucket or table outside the organization are denied at the endpoint, so the control the firewall gave this traffic stays in place.

```JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowOrganizationResources",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "*",
      "Resource": "*",
      "Condition": { "StringEquals": { "aws:ResourceOrgID": "<organization-id>" } }
    },
    {
      "Sid": "AllowAwsOwnedBuckets",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": [
        "arn:aws:s3:::prod-<region>-starport-layer-bucket/*",
        "arn:aws:s3:::al2023-repos-<region>-*/*",
        "arn:aws:s3:::amazonlinux-2-repos-<region>/*",
        "arn:aws:s3:::aws-ssm-<region>/*",
        "arn:aws:s3:::amazon-ssm-<region>/*",
        "arn:aws:s3:::amazon-ssm-packages-<region>/*",
        "arn:aws:s3:::<region>-birdwatcher-prod/*",
        "arn:aws:s3:::patch-baseline-snapshot-<region>/*",
        "arn:aws:s3:::repo.<region>.emr.amazonaws.com/*",
```

```
        "arn:aws:s3:::<region>.elasticmapreduce/*"
      ]
    },
    {
      "Sid": "AllowCloudFormationSignals",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::cloudformation-waitcondition-<region>/*"
    }
  ]
}
```

The second and third statements are the ones people miss. ECR serves image layers from an AWS-owned S3 bucket, and the Amazon Linux package repositories, the Systems Manager agent and Patch Manager (`aws-ssm-<region>`, `amazon-ssm-<region>`, `amazon-ssm-packages-<region>`, `<region>-birdwatcher-prod`, `patch-baseline-snapshot-<region>`), the EMR repositories, and CloudFormation wait conditions all use AWS-owned buckets too. An organization-only policy stops image pulls, package installs, patching, EMR cluster launches, and `cfn-signal` until each bucket is allowed by name. Set the same first statement on the DynamoDB gateway endpoint without the others. The CDK construct above applies the policy with `addToPolicy` on each gateway endpoint (a gateway endpoint created without a policy allows everything, so nothing is applied until the construct adds one), and the CloudFormation baseline sets it as the endpoint's `PolicyDocument`. The bucket list is the one AWS documents for these services today; confirm it against the current documentation for each service StrataForge runs in a spoke before the policy goes on. The interface endpoints' security groups need the same review as any other access path: scope each to the VPC's CIDR on port 443.

Verify with a pull of an image from ECR, a package install, a Systems Manager patch scan (`AWS-RunPatchBaseline` with `Operation=Scan`), and an EMR cluster launch from a spoke, then a `GetObject` on a bucket in the organization (allowed) and the same call on a bucket in another account (denied by the endpoint, and visible as an `AccessDenied` in the CloudTrail event). Do the hub's traffic last: start with the spoke VPC that sends the most S3 bytes, watch the flow logs on the Transit Gateway attachment fall, and move to the next spoke only when the pulls and installs have run clean for a day.

Kafka: let consumers read in their own Availability Zone

Each partition has a leader and followers in three Availability Zones. A consumer that always reads from the leader crosses a zone boundary two times in three, and every crossing bills $0.01 per GB each way. 6 MSK clusters send consumer fetches across zones ($201,148 a year). Since Kafka 2.4, a consumer can fetch from the follower in its own zone. It takes one setting on the cluster and one on every consumer.

The setting goes into the cluster's **existing** configuration, as a new revision. A new configuration with one property would replace the current one and reset everything else the brokers carry (retention, `min.insync.replicas`, and any other custom property) to the defaults. So read the current revision, append the line, and publish it as the next revision of the same configuration:

```bash
CONFIG_ARN=$(aws kafka describe-cluster --cluster-arn <cluster-arn> --region <region> \
  --query 'ClusterInfo.CurrentBrokerSoftwareInfo.ConfigurationArn' --output text) || exit 1
REVISION=$(aws kafka describe-cluster --cluster-arn <cluster-arn> --region <region> \
  --query 'ClusterInfo.CurrentBrokerSoftwareInfo.ConfigurationRevision' --output text) || exit
1

# Save the current properties: this file is the rollback.
aws kafka describe-configuration-revision --arn "$CONFIG_ARN" --revision "$REVISION" --region
<region> \
  --query 'ServerProperties' --output text | base64 --decode > msk-config-before.properties ||
exit 1

cp msk-config-before.properties msk-config-after.properties
echo "replica.selector.class=org.apache.kafka.common.replica.RackAwareReplicaSelector" >> msk-
config-after.properties

aws kafka update-configuration --arn "$CONFIG_ARN" --server-properties fileb://msk-config-
after.properties \
  --description "rack-aware fetch" --region <region> || exit 1
NEW_REVISION=$(aws kafka describe-configuration --arn "$CONFIG_ARN" --region <region> \
  --query 'LatestRevision.Revision' --output text)
aws kafka update-cluster-configuration --cluster-arn <cluster-arn> --current-version <current-
version> \
  --configuration-info "Arn=$CONFIG_ARN,Revision=$NEW_REVISION" --region <region>
```

If the cluster still runs on the default configuration, `msk-config-before.properties` is empty or absent: create a configuration that holds the line and the defaults you want to keep, and name it in the change record.

Every consumer sets `client.rack` to the AZ **ID** of the zone it runs in (MSK uses AZ IDs, such as `use1-az1`, because zone names map differently in each account). A Kafka `.properties` file does not expand `${...}`, so the value has to be written into the file, or into the client's settings, before the client starts. The node already carries its AZ ID as the label `topology.k8s.aws/zone-id`, and an init container can read it without touching the instance metadata service:

```yaml
initContainers:
  - name: rack
    image: <account-id>.dkr.ecr.us-east-1.amazonaws.com/kubectl:<tag>
    env:
      - name: NODE_NAME
        valueFrom: { fieldRef: { fieldPath: spec.nodeName } }
    command: ["sh", "-c", "kubectl get node \"$NODE_NAME\" -o
jsonpath='{.metadata.labels.topology\\.k8s\\.aws/zone-id}' > /rack/zone-id"]
    volumeMounts: [{ name: rack, mountPath: /rack }]
volumes:
  - name: rack
    emptyDir: {}
```

```bash
#!/bin/sh
# Container entrypoint: write client.rack into the consumer's properties file, then start the
consumer.
export KAFKA_CLIENT_RACK="$(cat /rack/zone-id)"
envsubst < /etc/kafka/consumer.properties.template > /tmp/consumer.properties
exec "$@"
```

```properties
# /etc/kafka/consumer.properties.template
client.rack=${KAFKA_CLIENT_RACK}
```

Kafka clients from 3.5 can skip the template and read the environment themselves, with `config.providers=env`, `config.providers.env.class=org.apache.kafka.common.config.provider.EnvVarConfigProvider`, and `client.rack=${env:KAFKA_CLIENT_RACK}`. The init container's service account needs one rule, `get` on `nodes`, and nothing else.

The simpler route has a price. Reading the zone from the instance metadata service needs the node's metadata hop limit raised to 2 (`HttpPutResponseHopLimit` in the launch template) so a container can reach it. With hop limit 2, every pod on that node can also reach the node role's credentials. Take that route only on a node group that runs nothing but these consumers, with a node role that carries no more than they need.

The broker change is a rolling restart; it takes time but not downtime. **Verify** with the `BytesOutPerSec` metric by broker and the inter-AZ lines of the Cost and Usage Report. **Roll back** by removing `client.rack` from the consumers, which sends them back to the leader at once, and then applying `msk-config-before.properties` as a new revision with `update-configuration` and `update-cluster-configuration`.

Service calls: keep traffic in its zone

Services on EKS calling each other across zones pay the same $0.01 per GB each way. Kubernetes can prefer an endpoint in the caller's zone, and falls back to any endpoint when a zone has too few. It needs three things: the Service annotation, replicas spread across zones, and enough replicas per zone for the hint to hold.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: mission-planner
  namespace: fleet
  annotations:
    service.kubernetes.io/topology-mode: Auto
spec:
  selector:
    app: mission-planner
  ports:
    - port: 80
      targetPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mission-planner
  namespace: fleet
spec:
  replicas: 6
  selector:
    matchLabels:
      app: mission-planner
  template:
    metadata:
      labels:
        app: mission-planner
    spec:
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: topology.kubernetes.io/zone
          whenUnsatisfiable: DoNotSchedule
          labelSelector:
            matchLabels:
              app: mission-planner
      containers:
        - name: app
          image: <account-id>.dkr.ecr.us-east-1.amazonaws.com/mission-planner:<tag>
          ports:
            - containerPort: 8080
```

On clusters at 1.33 or later, `spec.trafficDistribution: PreferClose` on the Service does the same job without an annotation. The change is rated effort M and risk Medium, and the reason is real: if a zone loses most of its replicas, the hint is dropped and traffic spreads again, which is the right behavior and shows up as a latency step. Verify with the inter-AZ lines of the Cost and Usage Report and with zone-labeled request metrics. Roll back by removing the annotation.

Robot telemetry: write to S3 through Firehose, not to CloudWatch Logs

CloudWatch Logs charges $0.50 per GB to ingest and $0.03 per GB-month to store. The robot telemetry log groups take in 146.5 TB a month across 96 groups ($87,481 a month), and 74 of those groups never expire. The telemetry is analyzed in the data lake, and the CloudWatch copies are read only during incidents. Amazon Data Firehose delivers to S3 as Parquet, and Athena queries it where it lies. The saving is the CloudWatch bill less everything the new path costs, and the new path has four parts: Firehose bills ingestion at $0.029 per GB with **each record rounded up to 5 KB** (a log record under 5 KB is billed as 5 KB, so small records cost more than their size); **Parquet conversion** is billed per GB converted; the Parquet then sits in S3 until the lake's lifecycle moves it; and **Athena** bills per terabyte scanned when responders query it. 96 log groups move ($818,009 a year, net of all four), rated Medium risk because incident responders lose a familiar view; keep one week of the most important groups in CloudWatch while the Athena views are built. The groups in Regions other than us-east-1 are in the backlog, behind the first Region's move. Where the firmware allows, batch several readings into one record: it brings the record nearer the 5 KB boundary and lowers the Firehose line.

In `fleet-infra`, replace the IoT rule's CloudWatch Logs action with a Firehose action, and add the delivery stream:

```typescript
import * as firehose from "aws-cdk-lib/aws-kinesisfirehose";
import * as iam from "aws-cdk-lib/aws-iam";
import * as iot from "aws-cdk-lib/aws-iot";
import * as s3 from "aws-cdk-lib/aws-s3";

const telemetryBucket = s3.Bucket.fromBucketName(this, "TelemetryBucket", "<telemetry-lake-
bucket>");

const deliveryRole = new iam.Role(this, "TelemetryDeliveryRole", {
  assumedBy: new iam.ServicePrincipal("firehose.amazonaws.com"),
});
telemetryBucket.grantReadWrite(deliveryRole);
deliveryRole.addToPolicy(new iam.PolicyStatement({
  actions: ["glue:GetTable", "glue:GetTableVersion", "glue:GetTableVersions"],
  resources: ["*"],
}));

const stream = new firehose.CfnDeliveryStream(this, "TelemetryStream", {
  deliveryStreamName: "robot-telemetry",
  deliveryStreamType: "DirectPut",
```

```
  extendedS3DestinationConfiguration: {
    bucketArn: telemetryBucket.bucketArn,
    roleArn: deliveryRole.roleArn,
    prefix: "raw/dt=!{timestamp:yyyy-MM-dd}/hr=!{timestamp:HH}/",
    errorOutputPrefix: "errors/!{firehose:error-output-type}/dt=!{timestamp:yyyy-MM-dd}/",
    bufferingHints: { intervalInSeconds: 300, sizeInMBs: 128 },
    compressionFormat: "UNCOMPRESSED",
    dataFormatConversionConfiguration: {
      enabled: true,
      inputFormatConfiguration: { deserializer: { openXJsonSerDe: {} } },
      outputFormatConfiguration: { serializer: { parquetSerDe: { compression: "SNAPPY" } } },
      schemaConfiguration: {
        databaseName: "telemetry",
        tableName: "robot_telemetry_raw",
        roleArn: deliveryRole.roleArn,
        region: this.region,
      },
    },
  },
});

const ruleRole = new iam.Role(this, "TelemetryRuleRole", {
  assumedBy: new iam.ServicePrincipal("iot.amazonaws.com"),
});
ruleRole.addToPolicy(new iam.PolicyStatement({
  actions: ["firehose:PutRecord", "firehose:PutRecordBatch"],
  resources: [stream.attrArn],
}));

new iot.CfnTopicRule(this, "TelemetryToFirehose", {
  ruleName: "robot_telemetry_to_firehose",
  topicRulePayload: {
    sql: "SELECT * FROM 'fleet/+/telemetry'",
    awsIotSqlVersion: "2016-03-23",
    actions: [{ firehose: { deliveryStreamName: stream.ref, roleArn: ruleRole.roleArn,
separator: "\n" } }],
  },
});
```

The Glue table `telemetry.robot_telemetry_raw` defines the Parquet schema; Firehose records that do not match it go to the error prefix instead of being lost. **Verify** that objects appear under the prefix within five minutes, and that an Athena `SELECT count(*)` on a partition matches the rule's `Success` metric (namespace `AWS/IoT`, dimensions `RuleName` and `ActionType=Firehose`). **Roll back** by pointing the rule back at its CloudWatch Logs action, which stays in the stack as a commented block for the first two weeks. Then set retention on the groups that remain (Logs and metrics).

Per-robot metrics: embedded metric format, not one metric per robot

CloudWatch charges $0.30 per custom metric-month for the first 10,000, $0.10 for the next 240,000, $0.05 for the next 750,000, and $0.02 beyond that. StrataForge publishes 848,000 per-robot metrics in four Regions, which cost $73,908 a month. A `RobotId` dimension turns every

robot into its own metric, four of them each. Dashboards and alarms use the site-level aggregates, and the per-robot series are queried in the data lake, so the per-robot metrics are paid for and never read.

Custom metrics by Region.

Region	Custom metrics today	Monthly cost
us-east-1	524,000	$38,064
eu-central-1	188,000	$19,421
ap-northeast-1	84,000	$9,706
ap-southeast-1	52,000	$6,717

The change is to publish through the **embedded metric format** (EMF): a structured log event that CloudWatch turns into metrics. Only the dimensions you name become metrics; everything else is a property that stays queryable in the log event. `SiteId` and `RobotModel` become the dimensions, and `RobotId` becomes a property. In the services that publish robot telemetry:

```python
from aws_embedded_metrics import metric_scope


@metric_scope
def publish_robot_metrics(metrics, site_id, robot_model, robot_id, battery_pct, motor_temp_c,
localization, latency_ms):
    metrics.set_namespace("StrataForge/Robot")
    # One dimension set, and no default dimensions (LogGroup, ServiceName, ServiceType): each
metric is one series per site and model.
    metrics.set_dimensions({"SiteId": site_id, "RobotModel": robot_model}, use_default=False)
    metrics.put_metric("BatteryPct", battery_pct, "Percent")
    metrics.put_metric("MotorTemperature", motor_temp_c, "None")
    metrics.put_metric("LocalizationConfidence", localization, "None")
    metrics.put_metric("NetworkLatency", latency_ms, "Milliseconds")
    metrics.set_property("RobotId", robot_id)  # a property: searchable in the log event, not
billed as a metric
```

CloudWatch aggregates the robots of one model at a site into a single series per site and model (min, max, average, and p95 are available), which is what the dashboards and alarms already read. **The per-robot values do not become free.** An EMF event is a log line, and log lines cost $0.50 per GB to ingest, so keeping every robot's values queryable in Logs Insights costs ingestion on one small event per robot per minute. The saving in the register is net of that ingestion: it is the per-robot metric charge less the EMF log charge, and the EMF line is the reason the saving is smaller than the metric bill above. The change is rated Medium risk and not Low: it removes a view of every robot and moves its alarms to new metrics, so the alarms are moved only after the new series exist. If the services emit more than one event per robot per minute, measure the ingest first and

aggregate at the site gateway before publishing. **Verify** with `aws cloudwatch list-metrics --namespace StrataForge/Robot` (the count falls from 848,000 to the site and model combinations) and with an alarm that fires on a test value, and move each alarm from a per-robot metric to its site series only after `list-metrics` shows the series with data. **Roll back** by restoring the old publisher; no history is lost, because CloudWatch keeps metric data for 15 months.

Audit and logging scope: CloudTrail data events and AWS Config

Logging is a security control first and a cost second. These two changes keep every control the security engagement put in place and remove only what nobody reads. Marcus Oyelaran agreed the CloudTrail scope with us, and the Config change keeps the daily state the SOC 2 evidence needs.

CloudTrail data events

The organization trail records read and write data events on every bucket and function. Scoping them saves $599,428 a year, because 94% of data events are reads of the telemetry and sensor lakes by the platform's own roles. Advanced event selectors keep every write event on every bucket, and keep read events on the buckets that hold customer data, firmware, and models. GuardDuty S3 Protection reads S3 data events directly from AWS, not from your trail, so its detection coverage does not change.

```JSON
[
  {
    "Name": "Management events",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Management"] }
    ]
  },
  {
    "Name": "S3 write events on every bucket",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::S3::Object"] },
      { "Field": "readOnly", "Equals": ["false"] }
    ]
  },
  {
    "Name": "S3 read events on buckets that hold customer data, firmware, and models",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::S3::Object"] },
      { "Field": "readOnly", "Equals": ["true"] },
      { "Field": "resources.ARN", "StartsWith": [
        "arn:aws:s3:::<customer-data-bucket-prefix>",
        "arn:aws:s3:::<firmware-bucket-prefix>",
        "arn:aws:s3:::<model-bucket-prefix>"
      ] }
```

```
    ]
  },
  {
    "Name": "Lambda invocations of functions that touch customer data",
    "FieldSelectors": [
      { "Field": "eventCategory", "Equals": ["Data"] },
      { "Field": "resources.type", "Equals": ["AWS::Lambda::Function"] },
      { "Field": "resources.ARN", "StartsWith": ["arn:aws:lambda:<region>:<account-
id>:function:<customer-data-function-prefix>"] }
    ]
  }
]
```

```bash
aws cloudtrail get-event-selectors --trail-name <org-trail-name> --region us-east-1 > event-
selectors-full.json || exit 1
jq '.AdvancedEventSelectors' event-selectors-full.json > event-selectors-before.json
aws cloudtrail put-event-selectors --trail-name <org-trail-name> \
  --advanced-event-selectors file://selectors.json --region us-east-1
```

Save the `before` file first: it is the rollback (`put-event-selectors --advanced-event-selectors file://event-selectors-before.json` restores it). The command's output is an object, so the jq step keeps only the array that `put-event-selectors` accepts. If the trail uses basic event selectors, the output has `EventSelectors` and not `AdvancedEventSelectors`; save that array instead and restore it with `--event-selectors`. Run both commands from the account and Region the organization trail lives in. The trail's configuration lives in `platform-landing-zone`, so the same JSON goes into the trail's `AdvancedEventSelectors` property in the same pull request. **Verify** that the `readOnly=true` data-event volume for the lake buckets drops in the next day's CloudTrail line of the Cost and Usage Report, and that a test write to a scoped bucket is still logged.

AWS Config recording

AWS Config charges $0.003 per configuration item recorded, and $0.012 for periodic (daily) recording, so periodic is cheaper only for a resource type that would otherwise produce more than four items a day. In `sf-cicd`, the runner and simulation fleets create network interfaces that live less than an hour, and 2 recorders spend money recording them ($105,462 a year). The fix is an exclusion, plus daily recording for instances:

```JSON
{
  "name": "default",
  "roleARN": "arn:aws:iam::<account-id>:role/<config-recorder-role>",
  "recordingGroup": {
    "allSupported": false,
    "includeGlobalResourceTypes": false,
    "exclusionByResourceTypes": {
      "resourceTypes": ["AWS::EC2::NetworkInterface"]
    },
    "recordingStrategy": { "useOnly": "EXCLUSION_BY_RESOURCE_TYPES" }
  },
  "recordingMode": {
    "recordingFrequency": "CONTINUOUS",
    "recordingModeOverrides": [
      {
        "description": "Runner instances change state many times a day; one daily item each is
cheaper and keeps the SOC 2 evidence",
        "resourceTypes": ["AWS::EC2::Instance"],
        "recordingFrequency": "DAILY"
      }
    ]
  }
}
```

```BASH
aws configservice describe-configuration-recorders --region <region> > config-recorder-
full.json || exit 1
jq '.ConfigurationRecorders[0]' config-recorder-full.json > config-recorder-before.json
aws configservice put-configuration-recorder --configuration-recorder file://recorder.json --
region <region>
```

Daily recording costs four times as much per item ($0.012 against $0.003), so it pays only for a resource type that would otherwise produce more than four items a day. A runner instance does: it starts, passes through several states, and stops within the hour, so one item a day is cheaper. A long-lived instance that changes once a week would cost more. Because the SOC 2 evidence the security engagement mapped needs daily state, not every change, daily recording for instances keeps it.

Take `name` and `roleARN` for `recorder.json` from `config-recorder-before.json`, and change only the recording group and mode: an account can hold one recorder per Region, and a second recorder with a new name is refused. If the existing recorder came from Control Tower (the name and role in `config-recorder-before.json` will say so), Control Tower can reset it when the landing zone updates. In that case StrataForge's platform team makes this change through the customization path Control Tower documents for the landing zone version it runs, and compares the live recorder with `recorder.json` after each landing zone update. **Roll back** by putting `config-recorder-before.json` back with `put-configuration-recorder`. Where a recorder is defined in

CloudFormation (`platform-landing-zone`), the same settings are the `RecordingGroup` and `RecordingMode` properties of that stack's `AWS::Config::ConfigurationRecorder`; do not add a second recorder resource to an account that already has one.

Logs and metrics

CloudWatch Logs by source.

Log source	Groups	Ingest per month	Cost per month	Groups that never expire
robot-telemetry	96	146.5 TB	$87,481	74
application	3,120	101.8 TB	$57,101	2,179
debug	45	20.1 TB	$10,374	34

692 log groups have no retention and grow without limit ($148,740 a year), and 45 debug groups can move to the Infrequent Access log class, which ingests at $0.25 per GB against $0.50 ($57,474 a year). The register names the groups to change; the other never-expire groups in the table keep their settings until their owners decide. Leave longer retention where a group was queried in the last 90 days, and leave out every group that carries audit evidence, because the security engagement's retention schedule governs those.

Setting retention deletes log events older than the new period, and nothing brings them back. So the script below changes only the groups on an approved list, prints what it will do first, and skips any group that already has a retention setting. Build `approved-log-groups.txt` from the register's rows for the Region (one line per group: the group name, a space, and the retention in days the register gives it; CloudWatch accepts only its fixed values, such as 30, 90, and 365), remove the audit-evidence groups, and have each group's owner confirm its line in the pull request. Run it once with the default, which only prints the plan, and again with `APPLY=yes` after the plan has been read.

```bash
APPLY=${APPLY:-no}
while read -r group days; do
  [ -z "$group" ] && continue
  row=$(aws logs describe-log-groups --log-group-name-prefix "$group" --region <region> \
    --query "logGroups[?logGroupName=='$group'].[logGroupName,to_string(retentionInDays)]" --
output text </dev/null) \
    || { echo "SKIP $group: lookup failed"; continue; }
  set -- $row
  [ "$1" = "$group" ] || { echo "SKIP $group: not found"; continue; }
  [ "$2" = "null" ] || { echo "SKIP $group: already keeps $2 days"; continue; }
  echo "PLAN $group: never expire -> $days days"
  if [ "$APPLY" = yes ]; then
    aws logs put-retention-policy --log-group-name "$group" --retention-in-days "$days" --
region <region> </dev/null \
      && echo "SET  $group" || echo "FAIL $group"
  fi
done < approved-log-groups.txt | tee retention-run.log
```

The log class of an existing group cannot be changed, so a debug group moves by creating its replacement and pointing the application at it:

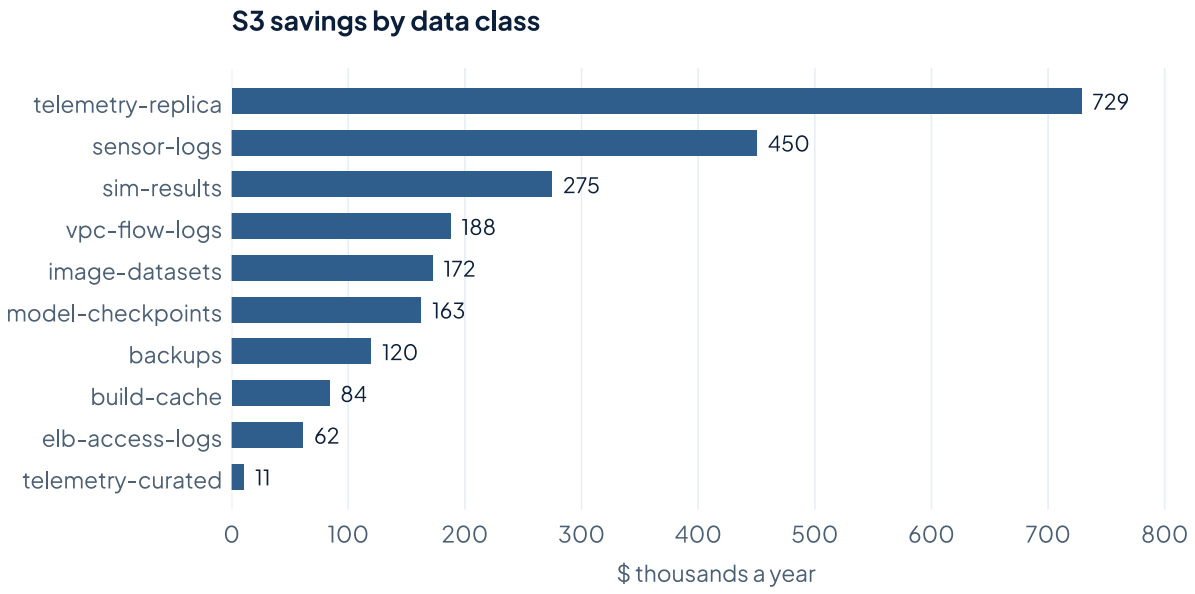
```bash
aws logs create-log-group --log-group-name <name>-ia --log-group-class INFREQUENT_ACCESS --
region <region>
aws logs put-retention-policy --log-group-name <name>-ia --retention-in-days 14 --region
<region>
```

Infrequent Access groups do not support metric filters or subscriptions, which is why the evidence for each debug group says it has neither. To stop the problem coming back, week 5 adds an EventBridge rule on `CreateLogGroup` that invokes a small Lambda function, and the function sets 30-day retention on any new group that has none and is not on the audit-evidence list (EventBridge cannot call `PutRetentionPolicy` itself). **Roll back** retention by setting the old value, which for every group this change touches was "never expire": `aws logs delete-retention-policy --log-group-name <name>` restores that setting for events that are still stored, and the `SET` lines in `retention-run.log` list the groups to run it on. Events that have already expired are gone, so keep the first change to groups nobody has queried.

S3: lifecycle rules and Intelligent-Tiering by data class

StrataForge stores 29.9 PB in 56.5 billion objects in 317 buckets, and most of it has not been read in months. 145 buckets get a lifecycle policy ($2,280,657 a year, with a one-time cost of $87,182 for the transition requests, repaid in 0.5 months of savings), and 161 move to S3 Intelligent-Tiering ($1,342,048 a year, one-time cost $28,433). Policies are set by **data class**, because a class has one answer to "how long do we need this and how fast do we need it back."

How a bucket is judged. A lifecycle rule acts on **object age**, so the age of the bytes comes from S3 Inventory: the share older than 90 days and older than 365 days. Age alone does not say a move pays. An object that moves and is then read pays a retrieval charge, so each bucket's server access logs say what share of its bytes was not read in 90 days, and the retrieval charge on objects still read after they move is counted against the saving. The register's `evidence` column carries both measurements and its `notes` column carries the retrieval figure, for example "includes $952 a month of retrieval on objects still read after they move (server access logs)" on `sf-sensor-log-lake-sensor-logs-7065-us-east-1-043`. A bucket where tiering by access is cheaper than tiering by age gets **S3 Intelligent-Tiering** instead (kind `s3-intelligent-tiering`), which moves each object by its own reads and charges no retrieval. It saves nothing until objects have gone 30 days unread in the new class, so its saving arrives a month after the rule is set, and its monitoring fee is in the figure. Intelligent-Tiering is offered only for buckets whose objects average 128 KB or more, and never for simulation results, backups, build caches, or replicas written to Deep Archive; those take the age-based rules or expiry below.



S3 savings by data class, the ten largest, thousands of dollars a year.

S3 savings by data class, for the 17 classes with a positive saving.

Data class	Buckets	Size	Monthly cost today	Per year	One-time cost
telemetry-replica	40	5,030 TB	$90,940	$728,672	$33,271
sensor-logs	48	7,200 TB	$112,274	$450,401	$199
sim-results	10	1,450 TB	$29,172	$274,912	$0
vpc-flow-logs	4	960 TB	$19,404	$187,797	$13,485

Data class	Buckets	Size	Monthly cost today	Per year	One-time cost
image-datasets	13	1,710 TB	$34,600	$172,402	$33,066
model-checkpoints	6	1,240 TB	$25,066	$162,642	$2
backups	2	520 TB	$10,462	$119,741	$474
build-cache	6	340 TB	$7,483	$84,013	$0
elb-access-logs	3	380 TB	$7,628	$61,507	$6,499
telemetry-curated	12	1,180 TB	$25,528	$10,770	$0
sim-scenarios	8	640 TB	$13,711	$9,657	$0
point-clouds	19	1,100 TB	$26,649	$9,326	$0
telemetry-raw	34	4,170 TB	$88,984	$5,012	$0
analytics	14	280 TB	$6,160	$2,547	$0
experiments	54	620 TB	$13,136	$1,147	$188
cloudtrail	3	530 TB	$10,753	$84	$0
findings	2	18 TB	$384	$26	$0

The rules, per class

All rules abort incomplete multipart uploads after 7 days. The classes with the largest savings, and what each is allowed to do:

- **Sensor logs** (`sensor-logs`): noncurrent versions expire after 30 days, Glacier Instant Retrieval after 90 days, Deep Archive after 365.
- **Telemetry replicas** (`telemetry-replica`): replicate **straight into Deep Archive** instead of replicating into Standard and transitioning later, and expire noncurrent versions after 180 days (Replicating straight into Deep Archive).
- **Raw telemetry** (`telemetry-raw`): Glacier Instant Retrieval after 90 days and Deep Archive after 730, or S3 Intelligent-Tiering where the access logs say it is cheaper than aging the objects out.
- **Sensor replicas** (`sensor-replica`): replicate straight into Glacier Instant Retrieval, or S3 Intelligent-Tiering where it is cheaper.
- **Image datasets** (`image-datasets`): Glacier Instant Retrieval after 90 days.
- **Simulation results** (`sim-results`): expire after 180 days.

- **Model checkpoints** (`model-checkpoints`): working checkpoints expire after 30 days, release checkpoints move to Glacier Instant Retrieval at 90.
- **VPC flow logs** (`vpc-flow-logs`): Standard-IA after 30, Glacier Flexible Retrieval after 90, expire after 400 days. The log-archive bucket is **versioned**, and on a versioned bucket `Expiration` only adds a delete marker and turns the current version into a noncurrent one that keeps billing. The rule therefore pairs the 400-day expiration with a noncurrent-version rule (30 days). The register's saving for this bucket holds only with that pair. The log archive otherwise keeps its object versions as tamper evidence, so the CISO confirms the 30-day window for flow-log objects, which are written once and never overwritten, before the rule is applied.
- **Cost and Usage Reports** (`cur`): stay in Standard for 400 days, then Glacier Instant Retrieval. They are read often while recent, and the table above shows no saving for them.
- **SAP backups** (`backups`): Glacier after 30 days, expire after 120 days. The SOX controls owner signs off on that retention before the rule is applied, and the register rates the line Medium for the same reason: an expired backup is gone.

```JSON
{
  "Rules": [
    {
      "ID": "abort-incomplete-uploads",
      "Status": "Enabled",
      "Filter": { "Prefix": "" },
      "AbortIncompleteMultipartUpload": { "DaysAfterInitiation": 7 }
    },
    {
      "ID": "expire-old-versions",
      "Status": "Enabled",
      "Filter": { "Prefix": "" },
      "NoncurrentVersionExpiration": { "NoncurrentDays": 30 }
    },
    {
      "ID": "tier-large-objects",
      "Status": "Enabled",
      "Filter": { "ObjectSizeGreaterThan": 131072 },
      "Transitions": [
        { "Days": 90, "StorageClass": "GLACIER_IR" },
        { "Days": 365, "StorageClass": "DEEP_ARCHIVE" }
      ]
    }
  ]
}
```

That is `sensor-logs`: age-based, with a noncurrent-version rule and a 365-day Deep Archive step. The same file shape, with different transitions, covers the other age-based classes. A bucket that Intelligent-Tiering suits takes a rule that moves every object in on day zero and lets S3 do the rest:

```JSON
{
  "Rules": [
    {
      "ID": "abort-incomplete-uploads",
      "Status": "Enabled",
      "Filter": { "Prefix": "" },
      "AbortIncompleteMultipartUpload": { "DaysAfterInitiation": 7 }
    },
    {
      "ID": "expire-old-versions",
      "Status": "Enabled",
      "Filter": { "Prefix": "" },
      "NoncurrentVersionExpiration": { "NoncurrentDays": 30 }
    },
    {
      "ID": "intelligent-tiering",
      "Status": "Enabled",
      "Filter": { "ObjectSizeGreaterThan": 131072 },
      "Transitions": [
        { "Days": 0, "StorageClass": "INTELLIGENT_TIERING" }
      ]
    }
  ]
}
```

Intelligent-Tiering moves an object to the Infrequent Access tier after 30 days unread and to Archive Instant Access after 90, with no retrieval charge, and bills a small monitoring fee per object. Objects under 128 KB are never tiered and pay no monitoring fee, so the size filter above leaves them where they are. New objects go into the class on day zero and so pay the transition request; the register's one-time cost covers the objects already in the bucket, and its monthly figure allows for the new ones.

Model checkpoints need the training job to tag what it writes, because lifecycle rules can match a tag but cannot exclude one. Working checkpoints carry `checkpoint-class=working`, and a promoted release checkpoint carries `checkpoint-class=release`:

```JSON
{
  "Rules": [
    {
      "ID": "abort-incomplete-uploads",
      "Status": "Enabled",
      "Filter": { "Prefix": "" },
      "AbortIncompleteMultipartUpload": { "DaysAfterInitiation": 7 }
    },
    {
      "ID": "expire-working-checkpoints",
      "Status": "Enabled",
      "Filter": { "Tag": { "Key": "checkpoint-class", "Value": "working" } },
      "Expiration": { "Days": 30 },
      "NoncurrentVersionExpiration": { "NoncurrentDays": 30 }
    },
    {
      "ID": "tier-release-checkpoints",
      "Status": "Enabled",
      "Filter": { "And": { "Tags": [{ "Key": "checkpoint-class", "Value": "release" }],
"ObjectSizeGreaterThan": 131072 } },
      "Transitions": [{ "Days": 90, "StorageClass": "GLACIER_IR" }]
    }
  ]
}
```

Flow logs step down through three classes before they expire. `Expiration` sits in its own rule with a whole-bucket filter, because under the 128 KB size filter it would never apply to smaller objects. The bucket is versioned, so the same rule expires noncurrent versions and a last rule removes the delete markers left behind:

```JSON
{
  "Rules": [
    {
      "ID": "abort-incomplete-uploads",
      "Status": "Enabled",
      "Filter": { "Prefix": "" },
      "AbortIncompleteMultipartUpload": { "DaysAfterInitiation": 7 }
    },
    {
      "ID": "step-down",
      "Status": "Enabled",
      "Filter": { "ObjectSizeGreaterThan": 131072 },
      "Transitions": [
        { "Days": 30, "StorageClass": "STANDARD_IA" },
        { "Days": 90, "StorageClass": "GLACIER" }
      ]
    },
    {
      "ID": "expire-and-remove-versions",
      "Status": "Enabled",
      "Filter": { "Prefix": "" },
```

```
      "Expiration": { "Days": 400 },
      "NoncurrentVersionExpiration": { "NoncurrentDays": 30 }
    },
    {
      "ID": "remove-leftover-delete-markers",
      "Status": "Enabled",
      "Filter": { "Prefix": "" },
      "Expiration": { "ExpiredObjectDeleteMarker": true }
    }
  ]
}
```

Rules that expire objects or versions are rated **Medium** risk, because a deleted object cannot be undone, and rules that only move objects between classes are Low unless they feed a replica. Apply each with `aws s3api put-bucket-lifecycle-configuration --bucket <bucket> --lifecycle-configuration file://lifecycle.json`, in the repository that owns the bucket (the register's `repo` column), as a `lifecycleRules` entry on the CDK `s3.Bucket` or by the script for buckets outside code. The call replaces the whole configuration, so read the old one first and keep it in the shape the put call accepts: `aws s3api get-bucket-lifecycle-configuration --bucket <bucket> | jq '{Rules}' > lifecycle-before.json`. The output also carries `TransitionDefaultMinimumObjectSize`, which the put call takes as a separate option, so the jq step drops it; if the command reports `NoSuchLifecycleConfiguration`, the bucket had no rules and the rollback is `aws s3api delete-bucket-lifecycle --bucket <bucket>`. Otherwise that file is the rollback. **Verify** the first lifecycle saving in the Cost and Usage Report 30 days after the rule is set, since the first transitions run overnight and Intelligent-Tiering waits 30 days for its first move, and check `S3 Inventory` for the storage-class mix. A rise in `GET` requests on a moved bucket, or in the retrieval line, is the sign that objects are still read after they move: set that bucket's transition back a tier and re-measure.

Replicating straight into Deep Archive

For the telemetry replicas, the disaster-recovery copy is written to the destination bucket directly in Deep Archive. A replica that lands in Standard and transitions later pays the Standard storage and a $0.05 per 1,000 transition request on top of the replication request. Landing it in Deep Archive pays **Deep Archive's PUT request price**, $0.05 per 1,000 objects, and nothing for Standard storage in between. That price is ten times the Standard PUT price, so it is the one-time cost to watch on a bucket of many small objects: the register's one-time cost for these buckets is those requests. Replication needs versioning on both the source and the replica bucket. Two settings go with the rule. Noncurrent versions in the destination expire after 180 days, Deep Archive's minimum storage duration, because an object deleted earlier is billed for the days it had left. And delete markers are not replicated, so a delete in the source does not remove the copy.

```JSON
{
  "Role": "arn:aws:iam::<account-id>:role/<replication-role>",
  "Rules": [
    {
      "ID": "replicate-to-deep-archive",
      "Status": "Enabled",
      "Priority": 1,
      "Filter": { "Prefix": "" },
      "DeleteMarkerReplication": { "Status": "Disabled" },
      "Destination": {
        "Bucket": "arn:aws:s3:::<replica-bucket>",
        "StorageClass": "DEEP_ARCHIVE"
      }
    }
  ]
}
```

```BASH
aws s3api put-bucket-replication --bucket <source-bucket> --replication-configuration file://
replication.json
```

This is a decision for the sponsor. Deep Archive restores are slow: a standard retrieval is within 12 hours and a bulk retrieval within 48 hours ($0.0025 per GB, against $0.02 for standard). The decision needs the price of the worst case, which is restoring the whole copy: 625,495,726 objects, 3.12 PB, in eu-west-1 and us-west-2. Bulk retrieval of all of it costs about $24,271 (against $131,701 at Standard) and takes about 6 days from the first request to the last object, submitted with S3 Batch Operations at up to 1,000 restore requests a second in each Region; each object is ready within 48 hours of its request (Bulk), or 12 hours (Standard). That is the retrieval alone. A restore also leaves a temporary Standard copy of every object for the 7 days in the request, about $15,937, and bringing the data back adds about $65,335 of transfer, so the worst case costs about $105,543 in all. For a disaster-recovery copy of raw telemetry, that is the right trade, and the readout asks Priya Raman to approve it. A restore looks like this:

```BASH
aws s3api restore-object --bucket <replica-bucket> --key <object-key> \
  --restore-request '{"Days": 7, "GlacierJobParameters": {"Tier": "Bulk"}}'
```

For a large restore, use S3 Batch Operations with the same request in a manifest. Objects already in the source bucket are not replicated by a new rule; S3 Batch Replication copies them.

The object-size rules

Two size rules decide whether a transition pays. **First, 128 KB.** Standard-IA, One Zone-IA, and Glacier Instant Retrieval bill at least 128 KB per object, and since September 2024 a lifecycle rule does not transition an object smaller than 128 KB unless the rule sets its own size filter. We set `ObjectSizeGreaterThan: 131072` explicitly in every transition rule above (inside an `And` block where the rule also filters on a tag) so that the behavior is the same on every bucket and is written down. **Second, 40 KB.** Glacier Flexible Retrieval and Deep Archive add 40 KB to every object: 8 KB billed at the Standard rate and 32 KB at the archive rate.

Both rules come from arithmetic worth doing for your own objects. For a 128 KB object, moving from Standard ($0.023 per GB-month) to Glacier Instant Retrieval ($0.004) saves $0.0000023 a month, and the transition request costs $0.00002, so it repays in 8.6 months. For a 1 MB object moving to Deep Archive ($0.00099 per GB-month plus the 40 KB overhead), the saving is $0.0000213 a month against a $0.00005 request, so it repays in 2.3 months. The same move for a 128 KB object takes 20 months, which is why small sensor files are compacted into larger Parquet or archive files before they are archived. Deep Archive also has a 180-day minimum storage duration, and Glacier Instant Retrieval has 90 days; an object that expires earlier pays for the remaining days.

Over-the-air updates: serve them through CloudFront

Signed firmware updates for every robot are served from S3. "Data Transfer: S3 to internet" costs $18,340 a month across the estate. S3 charges $0.09 per GB for the first 10 TB of internet transfer; CloudFront charges $0.085 for the first 10 TB in North America and Europe and steps down to $0.02 per GB, and transfer from S3 to CloudFront is free. An update goes to many robots at a site on the same day, so a CloudFront edge cache also turns thousands of S3 requests into a few.

Serving every update download through CloudFront is worth $63,697 a year across the 4 update buckets, on the transfer price alone. CloudFront's tiers apply to the organization's whole CloudFront volume, so we priced the added downloads at the margin of that volume, which is below the S3 internet rate StrataForge pays today; transfer from S3 to CloudFront is free. We did not count the edge cache's saving on S3 requests and on repeated downloads at a site, because the cache hit ratio is not known until a release ships through the distribution; the first release will show it on the bill. The update service issues the download URLs, so the change is made in the service, not in robot firmware. It is also a security improvement: the bucket becomes reachable only through the distribution. In `fleet-infra`:

```typescript
import * as cloudfront from "aws-cdk-lib/aws-cloudfront";
import * as origins from "aws-cdk-lib/aws-cloudfront-origins";
import * as s3 from "aws-cdk-lib/aws-s3";

// Imported bucket: CDK cannot add the CloudFront read grant to it, so the stack that owns
// the bucket carries the bucket policy statement shown after this code.
const otaBucket = s3.Bucket.fromBucketName(this, "OtaBucket", "<ota-bucket>");

const keyGroup = new cloudfront.KeyGroup(this, "OtaKeyGroup", {
  items: [new cloudfront.PublicKey(this, "OtaPublicKey", { encodedKey: "<public-key-pem>" })],
});

const distribution = new cloudfront.Distribution(this, "OtaDistribution", {
  defaultBehavior: {
    origin: origins.S3BucketOrigin.withOriginAccessControl(otaBucket),
    viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.HTTPS_ONLY,
    allowedMethods: cloudfront.AllowedMethods.ALLOW_GET_HEAD,
    cachePolicy: cloudfront.CachePolicy.CACHING_OPTIMIZED,
    trustedKeyGroups: [keyGroup],
  },
  priceClass: cloudfront.PriceClass.PRICE_CLASS_ALL,
  httpVersion: cloudfront.HttpVersion.HTTP2_AND_3,
});
```

`withOriginAccessControl` adds the bucket policy that lets CloudFront read the bucket only when the bucket is a construct in the same app. When the bucket is imported with `fromBucketName` (it belongs to another stack or was created by hand), CDK warns and adds nothing, and CloudFront answers 403 for every object. In that case the stack that owns the bucket carries the grant, with the distribution's ARN in the condition, and a KMS statement as well if the bucket uses SSE-KMS:

```json
{
  "Sid": "AllowCloudFrontServicePrincipalRead",
  "Effect": "Allow",
  "Principal": { "Service": "cloudfront.amazonaws.com" },
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::<ota-bucket>/*",
  "Condition": { "StringEquals": { "AWS:SourceArn": "arn:aws:cloudfront::<account-
id>:distribution/<distribution-id>" } }
}
```

For an SSE-KMS bucket, the key policy also allows `cloudfront.amazonaws.com` to call `kms:Decrypt` and `kms:GenerateDataKey*` with the same `aws:SourceArn` condition.

The update service starts issuing CloudFront signed URLs in place of S3 presigned URLs, which keeps the signed-update pipeline's signature check as the second control. A presigned URL is authorized by the IAM role that signed it, not by a bucket policy statement, so keep the S3

presigned-URL path working for one release cycle, then remove `s3:GetObject` on the bucket from the update service's role. After that, the bucket answers only to the distribution. **Roll back** by switching the service back to S3 presigned URLs; the distribution costs nothing while idle.

Spot capacity: simulation and CI

Two workloads are interruption-tolerant by design and run on On-Demand capacity: simulation batches in `crucible-sim-prod` (334 instances, $1,422,523 a year) and the ephemeral CI runners in `sf-cicd` (143 instances, $661,942 a year). Spot capacity costs a fraction of On-Demand and can be reclaimed with two minutes' notice, which a retried job does not notice. For simulation, AWS's guidance after the AWS RoboMaker shutdown on September 10, 2025 was AWS Batch, with multi-container jobs for containerized simulations; the same Batch queue is the right home for the training-adjacent batch work.

```typescript
import * as batch from "aws-cdk-lib/aws-batch";
import * as ec2 from "aws-cdk-lib/aws-ec2";
import * as ecs from "aws-cdk-lib/aws-ecs";
import { Size } from "aws-cdk-lib";

const spotEnv = new batch.ManagedEc2EcsComputeEnvironment(this, "SimSpot", {
  vpc,
  spot: true,
  allocationStrategy: batch.AllocationStrategy.SPOT_PRICE_CAPACITY_OPTIMIZED,
  instanceClasses: [ec2.InstanceClass.G5, ec2.InstanceClass.G6, ec2.InstanceClass.G6E],
  minvCpus: 0,
  maxvCpus: 4096,
});

// An On-Demand environment at order 2: jobs fall back to it when Spot has no capacity, and the
rollback below uses it.
const onDemandEnv = new batch.ManagedEc2EcsComputeEnvironment(this, "SimOnDemand", {
  vpc,
  spot: false,
  instanceClasses: [ec2.InstanceClass.G5, ec2.InstanceClass.G6, ec2.InstanceClass.G6E],
  minvCpus: 0,
  maxvCpus: 1024,
});

const queue = new batch.JobQueue(this, "SimQueue", { priority: 10 });
queue.addComputeEnvironment(spotEnv, 1);
queue.addComputeEnvironment(onDemandEnv, 2);

new batch.EcsJobDefinition(this, "SimEpisode", {
  retryAttempts: 5,
  retryStrategies: [
    batch.RetryStrategy.of(batch.Action.RETRY, batch.Reason.SPOT_INSTANCE_RECLAIMED),
    batch.RetryStrategy.of(batch.Action.EXIT, batch.Reason.CANNOT_PULL_CONTAINER),
    // Last: anything else is an application failure. Without this entry Batch retries it up to
retryAttempts times on GPUs.
    batch.RetryStrategy.of(batch.Action.EXIT, batch.Reason.custom({ onReason: "*" })),
```

```
  ],
  container: new batch.EcsEc2ContainerDefinition(this, "SimContainer", {
    image: ecs.ContainerImage.fromRegistry("<account-id>.dkr.ecr.us-east-1.amazonaws.com/sim-
runner:<tag>"),
    cpu: 8,
    memory: Size.gibibytes(32),
    gpu: 1,
  }),
});
```

The retry list is read in order, and a failure that matches no entry is retried like a reclaim, so the last entry ends every failure that is not a Spot reclaim. Episodes must write their state to S3 at checkpoints so a retry resumes and does not restart; that is the one code change, and Hiro Tanaka's team owns it. For the CI runners, a Karpenter NodePool on Spot and the GitHub Actions runner scale set give ephemeral runners that exist for one job:

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: ci-runners-spot
spec:
  template:
    spec:
      nodeClassRef:
        group: karpenter.k8s.aws
        kind: EC2NodeClass
        name: default
      taints:
        - key: sf-ci
          value: "true"
          effect: NoSchedule
      requirements:
        - key: karpenter.sh/capacity-type
          operator: In
          values: ["spot"]
        - key: karpenter.k8s.aws/instance-family
          operator: In
          values: ["c7g", "m7g", "c7i", "m7i"]
        - key: kubernetes.io/arch
          operator: In
          values: ["arm64", "amd64"]
      expireAfter: 8h
  disruption:
    consolidationPolicy: WhenEmpty
    consolidateAfter: 1m
  limits:
    cpu: "1500"
```

One scale set must not span both architectures, because a job lands on whichever runner is free and a build for one architecture can run on the other. So there are two scale sets, `sf-ci-arm64` and `sf-ci-amd64`, and each workflow chooses with `runs-on:`. Both use the NodePool above; each pins its pods to one architecture:

```yaml
# values.yaml for the gha-runner-scale-set Helm chart (the amd64 set is the same with amd64 in
both places)
githubConfigUrl: https://github.com/<organization>
githubConfigSecret: github-runner-app
runnerScaleSetName: sf-ci-arm64
minRunners: 0
maxRunners: 400
template:
  spec:
    nodeSelector:
      kubernetes.io/arch: arm64
    tolerations:
      - key: sf-ci
        operator: Equal
        value: "true"
        effect: NoSchedule
    containers:
      - name: runner
        image: ghcr.io/actions/actions-runner:latest
        command: ["/home/runner/run.sh"]
```

A reclaimed runner fails its job, and GitHub Actions can retry the failed job (`gh run rerun --failed`), so a reclaimed node is a retry, not an outage. Keep a small On-Demand pool for jobs that cannot be interrupted, such as release signing. **Roll back** by sending the queue to On-Demand capacity: put the `SimOnDemand` environment at order 1 (or remove the Spot environment from the queue) and deploy. A Spot allocation strategy cannot be combined with `spot: false`, which is why the On-Demand environment is a second environment and not a switch on the first. For the runners, remove the `capacity-type: spot` requirement from the NodePool, or add `on-demand` to its values.

Services priced by storage, throughput, or I/O

Three services bill for how they are provisioned more than for how much they hold, and each has a cheaper setting that the usage already supports.

AWS Backup: split the vault by what each resource type can do

The backup vault keeps every recovery point warm (2 findings, $84,988 a year). Today: 371 TB EBS, 186 TB RDS, 61 TB EFS, all warm. A vault holds several resource types and they cannot all go cold, so the vault is split by what each type allows, and 45% of EBS and 62% of EFS recovery points are monthly copies older than 90 days:

- **EBS recovery points** older than 90 days that are monthly copies move to the **snapshot archive tier**. The archive tier bills the full size of the snapshot, not the increment, and has a 90-day minimum. That is why only monthly copies move: a daily incremental is small while warm and large in the archive.
- **EFS recovery points** move to **cold storage** after 90 days, and are kept in it for at least 90 days.
- **RDS and Aurora recovery points stay warm.** They cannot move to cold storage, and the plan leaves them alone.

167 TB of monthly EBS points to the snapshot archive tier (billed at full size, 90-day minimum); 38 TB of EFS backups to cold storage; RDS and Aurora stay warm (they cannot move). A backup plan's rules cannot be limited to one resource type: every rule applies to every resource the plan selects. So the change is **one edit to the plan's existing monthly rule**, with every other rule left exactly as it is, and not a second monthly rule (that would make a second set of monthly copies and a new bill). `update-backup-plan` replaces the whole rule set, which is why the plan is fetched first and only the monthly rule's lifecycle is changed in the saved copy. With the opt-in, EBS recovery points move to the snapshot archive tier and EFS recovery points move to cold storage. Backup ignores the cold-storage setting for RDS and Aurora recovery points, so they stay warm. In `platform-landing-zone`, the `Lifecycle` object of the monthly rule becomes:

```JSON
{
  "MoveToColdStorageAfterDays": 90,
  "DeleteAfterDays": <retention-days>,
  "OptInToArchiveForSupportedResources": true
}
```

```BASH
aws backup get-backup-plan --backup-plan-id <plan-id> > backup-plan-full.json || exit 1
# The saved copy in the shape update-backup-plan accepts: rule IDs are output only.
jq '.BackupPlan | del(.Rules[].RuleId)' backup-plan-full.json > backup-plan-before.json
# Change only the monthly rule's lifecycle; every other rule is copied unchanged.
jq '(.Rules[] | select(.RuleName == "<monthly-rule-name>") | .Lifecycle) +=
{MoveToColdStorageAfterDays: 90, OptInToArchiveForSupportedResources: true}' \
  backup-plan-before.json > backup-plan-after.json
aws backup update-backup-plan --backup-plan-id <plan-id> --backup-plan file://backup-plan-
after.json
```

A lifecycle change on a rule applies to the recovery points made after it. The monthly copies that already exist move one at a time, from the vault listing, with the same lifecycle:

```bash
aws backup list-recovery-points-by-backup-vault --backup-vault-name <vault-name> \
  --query 'RecoveryPoints[?ResourceType==`EBS` || ResourceType==`EFS`].
[RecoveryPointArn,ResourceType,CreationDate,Lifecycle]' --output table
aws backup update-recovery-point-lifecycle --backup-vault-name <vault-name> --recovery-point-
arn <recovery-point-arn> \
  --lifecycle MoveToColdStorageAfterDays=90,DeleteAfterDays=<retention-
days>,OptInToArchiveForSupportedResources=true
```

`DeleteAfterDays` must be at least 90 days after `MoveToColdStorageAfterDays`; the retention is the one the plan already has, which this change does not shorten. Run the second command only for the monthly copies older than 90 days that the register lists. **Verify** in the vault listing: points older than 90 days show the cold or archive lifecycle, and the storage lines of the next month's Cost and Usage Report move from warm backup storage to the cold lines. **Roll back** for future points by applying the saved plan: `update-backup-plan --backup-plan file://backup-plan-before.json`. A recovery point that has already moved to cold storage can still be restored, with a longer restore time and a retrieval charge, so restore one of each type before the change to learn how long it takes.

Aurora storage: I/O-Optimized where I/O is a quarter of the bill

8 Aurora clusters pay more for I/O requests than the I/O-Optimized configuration would cost ($93,397 a year). Aurora Standard bills each I/O request on top of storage and instance hours. I/O-Optimized bills no I/O charges, at a higher price for instances and storage. AWS puts the break-even at I/O charges of about 25% of a cluster's Aurora spend, so pick clusters from the Cost and Usage Report (the I/O usage lines against the cluster's total) and not from the instance size. The change is in place:

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

In `fleet-infra` the same setting is one property on the cluster construct, `storageType: rds.DBClusterStorageType.AURORA_IOPT1`, so the next deploy does not undo it. Make the change in a maintenance window. **Verify** that the I/O lines of the Cost and Usage Report fall to zero for the cluster and the instance and storage lines rise by less than the I/O lines fell. **Roll back** with `--storage-type aurora`, which AWS allows at any time; it is the move to I/O-Optimized that is limited,

to once every 30 days, so check the cluster's I/O share for a full month before switching a large cluster. On instance classes with local NVMe storage the change restarts the engine, so schedule those in a maintenance window; on other classes it causes no downtime.

FSx for Lustre: lower the throughput tier to what training reads

One file system, `crucible-training-prod`, is provisioned for far more read throughput than training uses ($311,980 a year, rated Medium risk). It is 420 TB at 250 MB/s per TiB; peak read throughput 96 MB/s per TiB over 30 days, including checkpoint restores. The change is 125 MB/s per TiB (update the file system in place). Persistent 2 file systems change throughput in place; schedule it between training runs, and the rating reflects it: a training job that reads faster than the new tier slows down, and does not fail.

```bash
aws fsx update-file-system --file-system-id <fs-id> \
  --lustre-configuration PerUnitStorageThroughput=125 --region <region>
aws fsx describe-file-systems --file-system-ids <fs-id> --region <region> \
  --query 'FileSystems[0].[Lifecycle,LustreConfiguration.PerUnitStorageThroughput]'
```

In `crucible-infra` it is the construct's `perUnitStorageThroughput`, set in the same pull request. Do the change between training runs, and not during a checkpoint restore. **Verify** with the file system's `DataReadBytes` metric over the next two training runs: sustained throughput per TiB below the new tier means the change fits. **Roll back** by setting the tier back with the same command; AWS limits how often a file system can be updated, so confirm the interval before the change and keep the first window long.

Version upgrades that stop an Extended Support charge

Three kinds of resource are billed extra for running old versions, and each has a date after which AWS acts for you.

Amazon EKS

15 clusters run Kubernetes versions in extended support ($61,101 a year, $0.60 per cluster-hour against $0.10 in standard support). **Versions 1.31, 1.32, and 1.33 are in extended support.** Version 1.31 reaches the end of extended support on November 26, 2026; version 1.32 on March 23, 2027; version 1.33 on July 29, 2027. At the end, EKS upgrades the control plane to the oldest supported version on its own schedule, with no advance notice, and an auto-upgraded cluster cannot be rolled back. Node groups, add-ons, and self-managed nodes are not upgraded for you, so each step has two halves: the control plane, then the nodes and add-ons. Land on **1.36**, which is in standard support until August 2, 2027, one minor version at a time:

```bash
aws eks update-cluster-version --name <cluster-name> --kubernetes-version <next-version> --
region <region>
aws eks describe-update --name <cluster-name> --update-id <update-id> --region <region>
aws eks update-addon --cluster-name <cluster-name> --addon-name <addon> --resolve-conflicts
PRESERVE --region <region>
aws eks update-nodegroup-version --cluster-name <cluster-name> --nodegroup-name <nodegroup> --
region <region>
aws eks update-cluster-config --name <cluster-name> --upgrade-policy supportType=STANDARD --
region <region>
```

The last command sets the cluster's upgrade policy to `STANDARD`, so that after this upgrade it is auto-upgraded at the end of standard support instead of entering paid extended support. For the clusters on 1.31, which have the earliest deadline, do them first and do non-production before production. A cluster on 1.31 needs five upgrades (1.32, 1.33, 1.34, 1.35, then 1.36), so start it early enough that all five finish before the deadline. Before each step, run `kubectl get --raw /metrics` for deprecated-API usage (`apiserver_requested_deprecated_apis`) and fix what it shows. An in-place upgrade can be rolled back within 7 days; after that, rollback is a new cluster. In `fleet-infra`, `crucible-infra`, and `enterprise-infra`, the version is one line of CDK, and EKS moves one minor version at a time, so a cluster on 1.33 gets three pull requests (1.34, then 1.35, then 1.36), each merged only after the previous upgrade and its node and add-on steps have finished:

```typescript
import * as eks from "aws-cdk-lib/aws-eks";

// In the existing eks.Cluster construct, change only the version; every other property stays
as it is.
const cluster = new eks.Cluster(this, "Cluster", {
  version: eks.KubernetesVersion.of("1.34"), // was 1.33; the next pull requests set 1.35, then
1.36
  kubectlLayer: props.kubectlLayer,
  defaultCapacity: 0,
});
```

Amazon RDS and Aurora

49 database instances run engine versions in paid Extended Support ($224,137 a year), billed per vCPU-hour on every vCPU including Multi-AZ standbys: $0.10 in years 1 and 2 and $0.20 in year 3 for RDS for MySQL and PostgreSQL and Aurora PostgreSQL; Aurora MySQL version 2 is in year 2 and its year 3 price starts December 1, 2026. The dates that bind:

RDS and Aurora Extended Support dates. PostgreSQL 14 enters Extended Support on March 1, 2027, so it is not a destination.

Engine	Extended Support year 3 pricing	End of Extended Support
RDS for MySQL 5.7	In year 3 since March 1, 2026	June 30, 2029
RDS for PostgreSQL 11, Aurora PostgreSQL 11	In year 3 since April 1, 2026	March 31, 2027
RDS for PostgreSQL 12, Aurora PostgreSQL 12	March 1, 2027	February 29, 2028
RDS for PostgreSQL 13, Aurora PostgreSQL 13	March 1, 2028	February 28, 2029
Aurora MySQL version 2	December 1, 2026	June 30, 2029
RDS for MySQL 8.0	August 1, 2028	July 31, 2029

Upgrade with a blue/green deployment so that the old environment stays intact until you switch, and switchover takes under a minute in most cases. For an RDS instance:

```bash
aws rds create-blue-green-deployment --blue-green-deployment-name <name> \
  --source <source-db-arn> --target-engine-version <target-engine-version> \
  --target-db-parameter-group-name <target-parameter-group> --region <region>
aws rds describe-blue-green-deployments --blue-green-deployment-identifier <deployment-id> --
region <region>
aws rds switchover-blue-green-deployment --blue-green-deployment-identifier <deployment-id> \
  --switchover-timeout 300 --region <region>
```

For an Aurora cluster, the source is the cluster ARN and a major version also needs the cluster parameter group, `--target-db-cluster-parameter-group-name <target-cluster-parameter-group>`, next to the instance parameter group. List the versions a database can move to with `aws rds describe-db-engine-versions --engine <engine> --engine-version <current-version> --query 'DBEngineVersions[].ValidUpgradeTarget[].EngineVersion'`, and make each major version its own blue/green deployment.

Choose a target that is current: a major version of PostgreSQL later than 14 and Aurora MySQL version 3 (MySQL 8.0 compatible, in standard support until April 30, 2028). RDS for MySQL 8.0 entered Extended Support on August 1, 2026, so a MySQL 5.7 database does not stop at 8.0: it goes to 8.0 first and then on to 8.4 as a second deployment, which follows the first once the application has run on 8.0 clean. Confirm both steps with the valid-target command above before the work is scheduled.

There is no switch back after switchover. Before switchover, the blue environment is untouched and the way back is to delete the green one (`aws rds delete-blue-green-deployment --blue-green-deployment-identifier <deployment-id> --delete-target`). Once the switchover completes, the old instances are renamed with an `-old1` suffix and stop receiving writes, while the application writes to the upgraded instances. Going back then means repointing the application at the `-old1` instances, which loses every write made since switchover, or restoring the pre-upgrade snapshot. So take a manual snapshot first, keep the `-old1` instances for the observation window, and delete them only after it. You can also opt a database out of Extended Support at creation with `--engine-lifecycle-support open-source-rds-extended-support-disabled`, but on a database already past the end of standard support, disabling it triggers an automatic major version upgrade, so it is a decision to make deliberately.

ElastiCache for Redis OSS 5

Redis OSS 4 and 5 left standard support on January 31, 2026, and AWS enrolled the clusters in paid Extended Support the next day, at an 80% premium in years 1 and 2 and 160% in year 3 (for `cache.r7g.large`, $0.175 an hour more in years 1 and 2 and $0.35 in year 3). The upgrade to Valkey, with its commands and its rollback, is in the rightsizing plan (Valkey in place of Redis OSS); it removes the premium and the Redis OSS price.

Smaller changes, with their commands

Public IPv4 addresses

Every public IPv4 address costs $0.005 an hour whether it is attached or not, including the addresses AWS creates for load balancers, NAT gateways, and EKS nodes. 88 addresses can be removed ($3,585 a year) by putting instances behind a load balancer or NAT, by using IPv6 where the client allows it, and by turning off auto-assigned public IPs on private subnets. To find the addresses:

```bash
aws ec2 describe-network-interfaces --region <region> \
  --query 'NetworkInterfaces[?Association.PublicIp!=`null`].
[NetworkInterfaceId,Description,Association.PublicIp]' --output table
```

SQL Server: Enterprise to Standard

6 RDS for SQL Server instances run Enterprise edition ($182,814 a year if they move to Standard). RDS cannot change edition in place, so the move is a new instance and a native backup and restore. Before any of it, check that the databases use no Enterprise-only feature, and that they fit Standard's limits of 24 cores and 128 GB for the buffer pool:

```sql
SELECT feature_name FROM sys.dm_db_persisted_sku_features;
```

An empty result means no persisted Enterprise feature. Then create the Standard instance, add the `SQLSERVER_BACKUP_RESTORE` option to the option groups of both the Enterprise instance (to take the backup) and the Standard instance (to restore it), and move each database:

```sql
exec msdb.dbo.rds_backup_database @source_db_name='<database>',
@s3_arn_to_backup_to='arn:aws:s3:::<backup-bucket>/<database>.bak',
@overwrite_S3_backup_file=1;
exec msdb.dbo.rds_restore_database @restore_db_name='<database>',
@s3_arn_to_restore_from='arn:aws:s3:::<backup-bucket>/<database>.bak';
```

Cut over by changing the application's connection string. **Roll back** by pointing it back at the Enterprise instance, which stays until the Standard one has run for a month. The effort is L and the risk Medium for a reason: the licensing change is only as good as the feature check.

EBS gp2 to gp3, ECR lifecycle, DynamoDB, and instance schedules

- **EBS gp2 and io1 to gp3** (2,362 volumes, $138,492 a year). gp3 is $0.08 per GB-month against $0.10 for gp2, includes 3,000 IOPS and 125 MiB/s at any size, and can be changed in place on a running volume: `aws ec2 modify-volume --volume-id <volume-id> --volume-type gp3 --iops <iops> --throughput <mibps>`. Set IOPS to at least the volume's gp2 baseline (3 IOPS per GiB, up to 16,000) and throughput to 250 MiB/s on a volume over 334 GiB (a gp2 volume that size delivers up to 250 MiB/s, and a gp3 volume defaults to 125), on any volume whose peak was above 3,000 IOPS, and on every converted io1 volume, so nothing gets slower; the register's proposed setting for each volume carries the numbers, and the saving is priced with them. Write down the old IOPS, size, and type for each volume before the call, because the call overwrites them. A volume can be modified once every six hours; roll back by modifying it back to gp2 after the wait. The `io2` volume that holds the SAP HANA disaster-recovery copy keeps its provisioned IOPS: the failover time is the reason for them, so no change to it is in the register.
- **ECR lifecycle policies** (2 registries, $408,108 a year): 72% of image bytes are untagged builds older than 30 days. Preview before you apply (`aws ecr start-lifecycle-policy-preview --repository-name <repository> --lifecycle-policy-text file://policy.json`), then `aws ecr put-lifecycle-policy`:

```JSON
{
  "rules": [
    {
      "rulePriority": 1,
      "description": "Release images are never expired by the rules below",
      "selection": { "tagStatus": "tagged", "tagPatternList": ["v*"], "countType":
"imageCountMoreThan", "countNumber": 9999 },
      "action": { "type": "expire" }
    },
    {
      "rulePriority": 2,
      "description": "Keep the 30 most recent untagged images",
      "selection": { "tagStatus": "untagged", "countType": "imageCountMoreThan", "countNumber":
30 },
      "action": { "type": "expire" }
    },
    {
      "rulePriority": 3,
      "description": "Expire pull-request images after 30 days",
      "selection": { "tagStatus": "tagged", "tagPatternList": ["pr-*"], "countType":
"sinceImagePushed", "countUnit": "days", "countNumber": 30 },
      "action": { "type": "expire" }
    },
    {
      "rulePriority": 4,
      "description": "Expire build images after 30 days",
      "selection": { "tagStatus": "tagged", "tagPatternList": ["build-*"], "countType":
"sinceImagePushed", "countUnit": "days", "countNumber": 30 },
      "action": { "type": "expire" }
    }
  ]
}
```

ECR has no "keep" action, so release images are protected by rule order. Priority 1 matches every image tagged `v*` and only expires one when a repository holds more than 9,999 of them, which in practice never happens, and an image that a higher-priority rule has matched is not expired by a lower one. Use StrataForge's real release-tag pattern in place of `v*`. A rule's `tagPatternList` selects an image only if the image matches **all** of its patterns, so each pattern has its own rule; one rule listing both `pr-*` and `build-*` would match nothing, and the CI images would never expire. An

expiry removes the image and every tag on it, which is why release images get the first rule. Build the CI image tags as `build-<git-sha>` (and `pr-<number>`), so the patterns match what the workflow pushes.

- **DynamoDB to on-demand** (38 tables, $143,768 a year), for tables using under 20% of their provisioned capacity: `aws dynamodb update-table --table-name <table> --billing-mode PAY_PER_REQUEST`. On-demand costs $0.625 per million writes and $0.125 per million reads; AWS allows four switches from provisioned to on-demand in any rolling 24 hours, and a switch back to provisioned at any time. Read and global secondary index autoscaling targets for a switched table are deregistered at the same time, so they stop adjusting capacity that no longer exists.
- **Schedules** (415 instance and node-group schedules and 70 databases, $1,962,882 a year): non-production capacity with no use outside 7 a.m. to 7 p.m. local time on weekdays runs 260 hours a month and not 730. Standalone instances and databases take a tag (`Schedule=office-hours`) and the AWS Instance Scheduler solution starts and stops them. **Instances in an EKS node group are scheduled by the node group, not by the node:** the group replaces any node you stop, so a schedule sets the group's size to zero and back instead. The performance-test node groups in `fleet-perf` scale to zero outside their test windows and run 100 hours a month. A stopped instance keeps its volumes, so a schedule is the safest change in the report: starting an instance restores it exactly. To take a resource off its schedule, start it first and remove the `Schedule` tag afterward: a stopped instance whose tag is removed is no longer started by the schedule and stays stopped.

```bash
aws autoscaling put-scheduled-update-group-action --auto-scaling-group-name <node-group-asg> \
  --scheduled-action-name sf-scale-down --recurrence "0 19 * * 1-5" --time-zone "America/
Chicago" \
  --min-size 0 --desired-capacity 0 --region <region>
aws autoscaling put-scheduled-update-group-action --auto-scaling-group-name <node-group-asg> \
  --scheduled-action-name sf-scale-up --recurrence "0 7 * * 1-5" --time-zone "America/Chicago"
\
  --min-size <min-size> --desired-capacity <desired-size> --region <region>
```

In `fleet-infra` the same two actions go on the node group's Auto Scaling group in CDK, so a redeploy does not remove them. Roll back by deleting both scheduled actions and setting the node group's size back with `aws eks update-nodegroup-config`.

- **IoT Core** (Basic Ingest and device shadows, $611,827 a year together, priced Region by Region: 4 Basic Ingest findings and 4 shadow findings, one per Region that carries the traffic): telemetry published to `$aws/rules/<rule-name>/<topic>` skips the messaging charge (the rule and its actions are still billed), and shadows that update on change with a 15-minute heartbeat avoid the updates that carry no change (86% of shadow updates carry no change). Both need robot firmware 7.4, which ships through the OTA channel in November, so both sit in the backlog and not in a wave. The 3,412 ForgeArm gen-2 robots on firmware 3.x keep standard topics until their own update, so Basic Ingest changes nothing for those robots, and the saving in the register already leaves them out.

Deliverable 6: the savings roadmap

The roadmap is every opportunity in this report, ranked by annual savings, with its effort, risk, owner, and wave. **13,020 opportunities are worth $23.5 million a year** (a planning range of $17.2 million to $21.6 million, and $13.7 million realizable in contract year 3 until the private pricing commitment is renegotiated), and one-time costs total $128,812. It is a CSV with one row per opportunity (`docs/samples/strataforge/data/cost/savings-roadmap.csv`) that StrataForge can sort, filter, and work through without us. This section summarizes it.

The waves

Changes are grouped into waves so that each lands in the right order and under the right approval. The Analysis ends at the readout; the waves are what the Optimization's weeks 3 to 5 carry out, and the backlog is what StrataForge's own team does afterward.

The four waves and what each carries.

Wave	Dates	What lands	Opportunities	Per year
Week 3	Week of September 7, 2026	Waste removed, schedules, S3 lifecycle and Intelligent-Tiering, ECR and log retention, backup tiers	9,955	$9,683,651
Week 4	Week of September 14, 2026	Rightsizing and Graviton, databases and caches, Kinesis shards, the Database and SageMaker Savings Plans, and the queued Compute plan	2,094	$6,818,903
Week 5	Week of September 21, 2026	Endpoints, Kafka and service routing, telemetry and metrics, logging scope, the FSx tier, upgrades	473	$4,416,439
Backlog	After October 2, 2026	Changes that wait on a date or a release	498	$2,601,218

The backlog is $2.6 million that cannot land inside six weeks, mainly for two reasons that are in the register. EMR hosts on `r5.8xlarge` move to Graviton after the reservation on them ends on November 30, because moving earlier would strand a reservation that is still paid for; and the IoT Core changes need robot firmware 7.4, which ships through the OTA channel in November. The Basic Ingest line is worth less than the whole fleet: the 3,412 ForgeArm gen-2 robots on firmware 3.x keep standard topics until their own update, and those robots reach 4.2 only in each customer's own maintenance window. A backlog item has a start date in the register and an owner, and it is still counted in the roadmap.

By kind

Every kind of opportunity, ranked by annual savings.

Kind	Opportunities	Per month	Per year	One-time cost
S3 lifecycle and storage classes	145	$190,055	$2,280,657	$87,182
Rightsize and move to Graviton	579	$183,124	$2,197,489	$0
Commitments resized at renewal	1	$175,000	$2,100,001	$0
Instances off nights and weekends	415	$130,628	$1,567,541	$0
Simulation on Spot through AWS Batch	334	$118,544	$1,422,523	$0
S3 Intelligent-Tiering	161	$111,837	$1,342,048	$28,433

Kind	Opportunities	Per month	Per year	One-time cost
Move to Graviton, same size	867	$74,874	$898,490	$0
Robot telemetry to S3 through Firehose	96	$68,167	$818,009	$0
Per-robot metrics to site metrics	4	$67,895	$814,735	$0
Rightsize (same family)	340	$58,384	$700,605	$0
Idle instances	146	$57,296	$687,554	$1,868
Ephemeral CI runners on Spot	143	$55,162	$661,942	$0
VPC endpoints instead of NAT	53	$52,872	$634,466	$0
CloudTrail data events scoped	1	$49,952	$599,428	$0
Unattached volumes	1,428	$46,511	$558,133	$10,472
Orphaned snapshot chains	1,433	$41,979	$503,752	$0
Idle SageMaker notebooks and Studio apps	49	$41,237	$494,839	$0
IoT Basic Ingest for telemetry	4	$41,153	$493,835	$0
Unused p5 capacity reservation	1	$37,367	$448,400	$0
ECR lifecycle policies	2	$34,009	$408,108	$0
Databases off nights and weekends	70	$32,945	$395,341	$0
FSx for Lustre throughput tier	1	$25,998	$311,980	$0
Unused AMIs and their snapshots	2,727	$21,483	$257,790	$0
Database engines out of Extended Support	49	$18,678	$224,137	$0
Database Savings Plan	1	$17,991	$215,890	$0
ElastiCache: Valkey on Graviton	22	$17,828	$213,937	$0
Kafka consumers read in their own AZ	6	$16,762	$201,148	$0
SQL Server Enterprise to Standard	6	$15,235	$182,814	$0
Unused Redshift cluster	1	$13,035	$156,419	$0
CloudWatch log retention	692	$12,395	$148,740	$0

Kind	Opportunities	Per month	Per year	One-time cost
DynamoDB to on-demand	38	$11,981	$143,768	$0
EBS gp2 and io1 to gp3	2,362	$11,541	$138,492	$0
SageMaker Savings Plan	1	$11,164	$133,969	$0
Lambda on Arm	272	$10,562	$126,742	$0
Device shadows on change	4	$9,833	$117,992	$0
Kinesis shards sized to peak	32	$8,821	$105,858	$0
AWS Config recording scoped	2	$8,788	$105,462	$0
Aurora I/O-Optimized	8	$7,783	$93,397	$0
Databases: rightsize and Graviton	74	$7,391	$88,686	$0
AWS Backup cold storage	2	$7,082	$84,988	$0
Idle non-production databases	14	$6,192	$74,307	$624
All update downloads through CloudFront	4	$5,308	$63,697	$0
Kubernetes out of extended support	15	$5,092	$61,101	$0
Debug logs to Infrequent Access	45	$4,789	$57,474	$0
OpenSearch: Graviton and gp3	4	$3,517	$42,204	$0
Service calls stay in their AZ	1	$2,883	$34,596	$0
WorkSpaces billed by the hour	1	$2,854	$34,247	$0
MSK brokers on Graviton	6	$1,754	$21,043	$0
Kinesis streams with no traffic	3	$1,528	$18,330	$0
Long-stopped instances	48	$955	$11,456	$234
Idle load balancers	44	$689	$8,273	$0
Unassociated Elastic IPs	166	$563	$6,762	$0
Public IPv4 addresses removed	88	$299	$3,585	$0
Move to the current generation	9	$252	$3,027	$0
Total	**13,020**	**$1,960,018**	**$23,520,211**	**$128,812**

The 30 largest opportunities

The 30 opportunities with the largest annual savings. Effort S is a day or less and M is a few days; risk is Low, Medium, or High.

RankID	Opportunity	Where	Owner	Repository	Wave	Effort	Risk	One-time	Per year
1C-0002	Resize the commitments as they renew	sf-management	Elena Vasquez	`platform-landing-zone`	Week 4	S	High	$0	$2,100,001
2A-0161	Scope CloudTrail data events with advanced event selectors	sf-management	Elena Vasquez	`platform-landing-zone`	Week 5	S	Low	$0	$599,428
3W-6059	Reduce the On-Demand Capacity Reservation to what training uses	crucible-training-prod	Hiro Tanaka	`crucible-infra`	Week 3	S	Medium	$0	$448,400
4A-0157	Stop publishing one CloudWatch metric per robot	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Week 5	M	Medium	$0	$413,688
5T-3444	Add ECR lifecycle policies	sf-cicd	Elena Vasquez	`platform-landing-zone`	Week 3	S	Low	$0	$339,005
6A-0170	Lower the FSx for Lustre throughput tier to what training reads	crucible-training-prod	Hiro Tanaka	`crucible-infra`	Week 5	S	Medium	$0	$311,980
7A-0162	Publish telemetry through IoT Basic Ingest	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Backlog	L	Medium	$0	$273,124
8A-0158	Stop publishing one CloudWatch metric per robot	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Week 5	M	Medium	$0	$216,619
9C-0001	Buy a Database Savings Plan after the Graviton moves	sf-management	Elena Vasquez	`platform-landing-zone`	Week 4	S	Medium	$0	$215,890
10W-6060	Snapshot and delete the unused dc2 Redshift cluster	corp-data-warehouse	Lucas Ferreira	`enterprise-infra`	Week 3	S	Low	$0	$156,419
11C-0003	Buy a SageMaker Savings Plan for the always-on endpoints	sf-management	Elena Vasquez	`platform-landing-zone`	Week 4	S	Medium	$0	$133,969

RankID	Opportunity	Where	Owner	Repository	Wave	Effort	Risk	One-time	Per year
12A-0164	Publish telemetry through IoT Basic Ingest	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Backlog	L	Medium	$0	$122,249
13A-0054	Let consumers read from the replica in their own Availability Zone	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Week 5	S	Low	$0	$120,394
14A-0159	Stop publishing one CloudWatch metric per robot	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Week 5	M	Medium	$0	$108,644
15T-2565	Lifecycle policy (`sf-sensor-log-lake-sensor-logs-7065-us-east-1-043`)	crucible-data	Hiro Tanaka	`crucible-infra`	Week 3	S	Low	$35	$103,784
16A-0075	Write robot telemetry to S3 through Firehose, not to CloudWatch Logs	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Week 5	M	Medium	$0	$99,938
17T-2576	Move to S3 Intelligent-Tiering	crucible-data	Hiro Tanaka	`crucible-infra`	Week 3	S	Medium	$13	$94,792
18T-2638	Lifecycle policy (`sf-labeled-datasets-image-datasets-9098-eu-west-1-003`)	kestrel-data	Jonah Reyes	`(not in code)`	Week 3	S	Low	$19,975	$85,053
19T-2520	Lifecycle policy (`sf-checkpoints-model-checkpoints-7042-us-east-1-004`)	crucible-training-prod	Hiro Tanaka	`crucible-infra`	Week 3	S	Medium	$1	$83,277
20T-2515	Lifecycle policy (`sf-episode-results-sim-results-7019-us-east-1-009`)	crucible-sim-prod	Hiro Tanaka	`crucible-infra`	Week 3	S	Medium	$0	$81,685
21A-0160	Stop publishing one CloudWatch metric per robot	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Week 5	M	Medium	$0	$75,784
22A-0081	Write robot telemetry to S3 through Firehose, not to CloudWatch Logs	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Week 5	M	Medium	$0	$74,188
23S-0410	Scale the node group to zero outside performance-test windows	fleet-perf	Sam Okafor	`fleet-infra`	Week 3	S	Low	$0	$73,435
24T-2617	Lifecycle policy (`sf-hana-backups-backups-8016-us-east-1-001`)	erp-sap-prod	Ana Kowalski	`enterprise-infra`	Week 3	S	Medium	$300	$73,430

RankID	Opportunity	Where	Owner	Repository	Wave	Effort	Risk	One-time	Per year
25T-2369	Lifecycle policy (`sf-log-archive-vpc-flow-logs-2291-us-east-1-002`)	sf-log-archive	Nadia Petrova	`platform-landing-zone`	Week 3	S	Medium	$6,234	$72,549
26T-3447	Add ECR lifecycle policies	crucible-models	Hiro Tanaka	`crucible-infra`	Week 3	S	Low	$0	$69,103
27A-0094	Write robot telemetry to S3 through Firehose, not to CloudWatch Logs	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Backlog	M	Medium	$0	$68,005
28A-0163	Update device shadows on change, not every minute	fleet-ingest-prod	Sam Okafor	`fleet-infra`	Backlog	M	Low	$0	$67,988
29T-2469	Lifecycle policy (`sf-telemetry-lake-eu-telemetry-replica-5160-eu-west-1-009`)	fleet-eu-prod	Sam Okafor	`fleet-infra`	Week 3	S	Medium	$3,205	$67,303
30S-0408	Scale the node group to zero outside performance-test windows	fleet-perf	Sam Okafor	`fleet-infra`	Week 3	S	Low	$0	$66,551

These 30 are 29% of the total. The roadmap is a long tail on purpose: 13,020 opportunities, because most of the savings come from doing the same simple thing across thousands of resources.

Effort, risk, and owners

Every opportunity has a named owner, and the owner decides. The register's owner column is the person who signs off; the repository column is where the pull request goes. By owner, in the top 30:

Owners of the largest opportunities.

Owner	Opportunities in the top 30	Annual savings in the top 30
Elena Vasquez	5	$3,388,294
Sam Okafor	14	$1,847,910
Hiro Tanaka	7	$1,193,021
Lucas Ferreira	1	$156,419
Jonah Reyes	1	$85,053
Ana Kowalski	1	$73,430
Nadia Petrova	1	$72,549

Most opportunities are rated Low risk and effort S or M. The Medium ones are mostly those that touch production data paths or are hard to reverse: the p5 capacity reservation, the hub's S3 and DynamoDB endpoints (the firewall used to inspect that traffic), the telemetry move to Firehose, the removal of per-robot metrics, the FSx throughput tier, simulation on Spot, the Kubernetes and database version upgrades, SQL Server editions, service routing, the IoT changes, S3 Intelligent-Tiering and the replication into Deep Archive, S3 lifecycle lines that expire objects or noncurrent versions, the Database and SageMaker Savings Plans, and rightsizing in production. One is rated High: the three-year Compute Savings Plan (C-0002), a $19.4 million commitment that AWS does not take back. Each Medium that can be reversed has a rollback in its runbook that is rehearsed before the change. The ones that cannot be reversed, the commitments and the expirations, get a second approval and a dated record instead; a queued Savings Plan can be deleted until it starts, and not after.

One-time costs and payback

Most changes cost nothing but an engineer's time. One-time costs are charges AWS makes to do the change: lifecycle transition requests, and archive snapshots taken before a deletion.

One-time costs and the months of savings that repay them.

Kind	One-time cost	Monthly savings	Payback
S3 lifecycle and storage classes	$87,182	$190,055	0.5 months
S3 Intelligent-Tiering	$28,433	$111,837	0.3 months
Idle instances	$1,868	$57,296	0.0 months
Unattached volumes	$10,472	$46,511	0.2 months
Idle non-production databases	$624	$6,192	0.1 months
Long-stopped instances	$234	$955	0.2 months
All opportunities	**$128,812**	**$1,960,018**	**0.07 months**

The longest payback is in the S3 lifecycle line, and it varies by bucket: a bucket of many small objects can take most of a year to repay its transition requests, which is the arithmetic in The object-size rules. Each lifecycle row in the register carries its own payback, and our rule is to skip any bucket whose payback is longer than twelve months until its objects are compacted.

The order that matters

Four dependencies set the sequence, and each is a reason the roadmap cannot be run in any order:

1. **Waste first.** It needs no design and it shrinks the base that everything after is sized on.
2. **Rightsizing and schedules before commitments.** The Compute Savings Plan is sized to the estate after the roadmap, so the changes that move it (rightsizing, Graviton moves, schedules, Spot) must be at least decided before it is queued.
3. **The Database Savings Plan after the Graviton databases, and the SageMaker plan after the idle endpoints go.** The Database plan covers generation 7 usage only, so buying it before the databases move buys coverage for instances that are about to go. The first plan is $25 an hour, and a second plan of $81 an hour brings the total to $106 when the Aurora reservations end, because a Savings Plan cannot be changed once it is bought. The SageMaker plan is sized to the endpoints that remain.
4. **Private pricing renegotiation before the savings land,** because it is easier to ask for a different shape while the spend is still high, and because until it is done $10.1 million of the year-3 reduction comes back as a true-up in contract year 3.

The guarantee

Where this stands against the guarantee

The Optimization's second invoice is issued only if the savings in place at the end of week six, measured on StrataForge's Cost and Usage Report and annualized, are at least three times the $35,000 fee: **$105,000 a year**. This roadmap identifies $23.5 million, and even the low end of the planning range, $17.2 million, is 164 times the gate, so the week-two exit does not apply: the engagement continues into week 3.

That multiple is an estimate and not a measurement. What counts at week six is what is in place and visible on the bill, measured on the Cost and Usage Report. A change that is declined, or that is approved and later declined, is not in place and counts for nothing.

Appendix A: the registers

Three CSV files carry the full detail behind this report. They sit under `docs/samples/strataforge/data/cost/`, and the repository is the system of record.

The three registers and their row counts.

File	Rows	Contents
`waste-register.csv`	6,060	Every waste finding (Deliverable 2)
`rightsizing-plan.csv`	2,205	Every compute, database, cache, search, broker, and Lambda change (Deliverable 3)
`savings-roadmap.csv`	13,020	Every opportunity in every category, ranked (Deliverable 6)

The register columns. From week 3, the same files gain status columns as each change lands.

Column	Meaning
`id`	Stable identifier: W for waste, R for rightsizing, A for architecture, T for storage, S for schedules, C for commitments
`category, kind`	The category and the kind of change
`account, account_id, region`	Where the resource is; account IDs are masked
`resource, team`	The resource ID and its Team tag, or "(untagged)"

Column	Meaning
`title`, `current`, `proposed`	The change, what exists now, and what it becomes
`evidence`	The measurement behind the finding
`monthly_list`, `monthly_net`	Monthly savings at list prices and net of the 7% private pricing discount
`annual_net`, `one_time`	Annual net savings and the one-time cost
`effort`, `risk`, `wave`	S, M, or L; Low, Medium, or High; and the wave
`owner`, `repo`	The person who signs off and the repository that makes the change, or "(not in code)"
`count`, `notes`, `starts`	Resources covered, caveats, and a start date for backlog items

Appendix B: assumptions

- **Estimates, not measurements.** Every saving here is an estimate from usage and list prices. Week 6 replaces it with a measurement, change by change.
- **The planning range is an assumption.** The realization rates by tier (90% to 100% for hygiene, 60% to 85% for engineering, 30% to 70% for decisions and dependencies, 85% to 100% for commitments) are our planning assumptions, not measurements, and they give the range of $17.2 million to $21.6 million.
- **The private pricing floor is read from a summary.** We have not read the contract. Realizable savings assume the commitment counts spend net of the private pricing discount, Savings Plans and reservation fees included, that a shortfall is billed as a true-up, and that contract year 3 runs on the projected bill ($31.9 million at today's usage); each assumption is open until Ravi Menon's team confirms it.
- **The commitment simulation covers part of the estate.** 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.
- **Steady state.** Savings are monthly figures at the August run-rate. They neither add the baseline's growth nor assume it stops.

- **Net of the private pricing discount.** The 7% discount applies to eligible usage after Savings Plans and reservations; we applied it to every opportunity uniformly. Usage that turns out not to be eligible would make a saving slightly smaller than shown, and the week 6 measurement corrects it.
- **Support follows the bill.** Enterprise Support falls as the bill falls, by $58,812 a month at full roadmap. It is not counted as a saving of any opportunity, but is shown in the reconciliation.
- **The coverage effect** is the measured difference between the sum of the parts and the re-priced whole ($377 a month); it is not a balancing figure.
- **No new commitments counted before they are bought.** The commitment lines assume the recommendations are approved and purchased on the dates in the calendar.
- **Out of scope:** the separate AWS GovCloud (US) organization, and any change that needs a robot firmware release other than the two named in the backlog.

Appendix C: price provenance

Every price used by the model and quoted in this report comes from an AWS source, and each is dated.

The price sources behind the model.

Price	Source	Retrieved
On-Demand prices for EC2, EBS, S3, RDS, ElastiCache, OpenSearch, MSK, Kinesis, Firehose, DynamoDB, Lambda, EKS, load balancers, networking, CloudFront, CloudWatch, SageMaker, FSx for Lustre, IoT Core, Redshift, and security services	AWS Price List API, `get-products`, us-east-1 endpoint, On-Demand terms	October 13, 2026
Savings Plans effective rates	AWS Savings Plans API, `describe-savings-plans-offering-rates` (Compute and EC2 Instance, Linux, shared tenancy, 1 and 3 years)	October 13, 2026
Reserved Instance effective rates	AWS Price List API, Reserved terms (standard class, all payment options)	October 13, 2026
Spot prices	`ec2 describe-spot-price-history`, 30-day time-weighted mean across Availability Zones	October 13, 2026
Database Savings Plans rates	AWS Price List API and the launch documentation	October 13, 2026
Extended Support, Support tiers, EKS versions, and Lambda, RDS, and ElastiCache lifecycle dates	AWS documentation and pricing pages, read on the retrieval date	October 13, 2026

Prices are for us-east-1 unless a line names another Region, and the model prices each resource in its own Region. Where AWS's sources disagree (for example, on whether CloudWatch vended-log tiers pool across services), we model each service's cost in its own tier bucket and say so.

Appendix D: AWS facts this report relies on

These are the dated AWS facts behind the recommendations, so that a reviewer can check them against AWS's pages.

- **EKS:** versions 1.34 through 1.36 are in standard support at the retrieval date and 1.31 through 1.33 are in extended support, at $0.10 and $0.60 per cluster-hour. Extended support for 1.31 ends November 26, 2026, for 1.32 March 23, 2027, and for 1.33 July 29, 2027. Auto-upgrade at the end of extended support has no advance notice and no rollback.

- **RDS and Aurora Extended Support:** $0.10 per vCPU-hour in years 1 and 2 and $0.20 in year 3, per the dates in the table above. RDS for MySQL 5.7 and Aurora MySQL version 2 end Extended Support on June 30, 2029.
- **ElastiCache:** Valkey is 20% below Redis OSS for node-based clusters; Redis OSS 4 and 5 are in Extended Support at an 80% premium in years 1 and 2 and 160% in year 3.
- **Savings Plans:** Compute Savings Plans up to 66% off On-Demand, EC2 Instance up to 72%, and Database Savings Plans up to 35%, one year, No Upfront, announced December 2, 2025. SageMaker Savings Plans cover the always-on endpoints in dollars an hour for one or three years.
- **Network Firewall and NAT:** Network Firewall waives the hours and per-gigabyte charges of the NAT gateways it inspects, so the hub saving is Transit Gateway ($0.02 per GB) and firewall processing ($0.065 per GB). Gateway endpoints for S3 and DynamoDB cannot be reached across a Transit Gateway.
- **Firehose:** Direct PUT is billed at $0.029 per GB with each record rounded up to 5 KB, and Parquet conversion is billed per GB converted.
- **S3 Intelligent-Tiering:** objects move to Infrequent Access after 30 days without access and to Archive Instant Access after 90, with no retrieval charge and a monitoring fee per object of 128 KB or more. Deep Archive PUT requests are $0.05 per 1,000, and its minimum storage duration is 180 days.
- **Aurora I/O-Optimized:** no I/O charges, at a higher price for instances and storage; AWS puts the break-even at I/O charges of about 25% of cluster spend, and the move to I/O-Optimized is allowed once every 30 days, while the move back to Standard is allowed at any time.
- **AWS Backup:** EBS recovery points can move to the snapshot archive tier (billed at full size, 90–day minimum) and EFS recovery points to cold storage; RDS and Aurora recovery points cannot move.
- **Public IPv4:** $0.005 per address-hour, attached or not.
- **CloudWatch:** $0.50 per GB of log ingestion in the Standard class and $0.25 in Infrequent Access; $0.03 per GB-month of storage.
- **AWS Config:** $0.003 per configuration item continuously, $0.012 periodically.
- **S3:** the lifecycle transition prices, the 30, 90, and 180–day minimum storage durations, and the 128 KB and 40 KB object-size rules in The object-size rules.
- **EBS:** gp3 at $0.08 per GB-month with 3,000 IOPS and 125 MiB/s included; gp2 at $0.10.
- **Support plans:** AWS Enterprise Support is the greater of $5,000 a month or 10% of monthly charges up to $150,000, 7% to $500,000, 5% to $1 million, and 3% above.

- **IoT Core:** $1.00 per million messages for the first billion a month; AWS IoT Analytics and AWS IoT Events are shut down.
- **AWS RoboMaker:** shut down on September 10, 2025; AWS pointed simulation customers to AWS Batch.

© 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.