Vantalect

AWS COST · SAMPLE DELIVERABLE

Quick Wins Applied

Week 3 of 6: Quick wins · prepared for StrataForge Robotics, Inc.

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

September 11, 2026 · AWS Cost Analysis & Optimization: The Optimization (six weeks)

SAMPLE DELIVERABLE

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

Contents

Summary 2
What went in 4
The largest changes 6
How each change was made 9
Safety 10
What was declined or deferred 12
The exact mechanism 14
The waste removal runbook 14
Non-production schedules 21
S3 lifecycle rules and Intelligent-Tiering 25
CloudWatch Logs retention 29
AWS Backup tiers 30
ECR lifecycle policies 32
Volume types and WorkSpaces 33
Verifying a change on the bill 34
What comes next 35

Summary

In week 3 StrataForge's engineers put **$7.8 million a year** of savings in place, in changes Vantalect drafted and reviewed: 9,302 changes to 65,190 resources across 22 kinds of change, every one approved by the owner of the workload it touches. That is 33% of the $23.5 million the roadmap identified, from the quick wins alone. $1.9 million a year of week-3 items did not go in, each for a reason its owner gave, listed in What was declined or deferred.

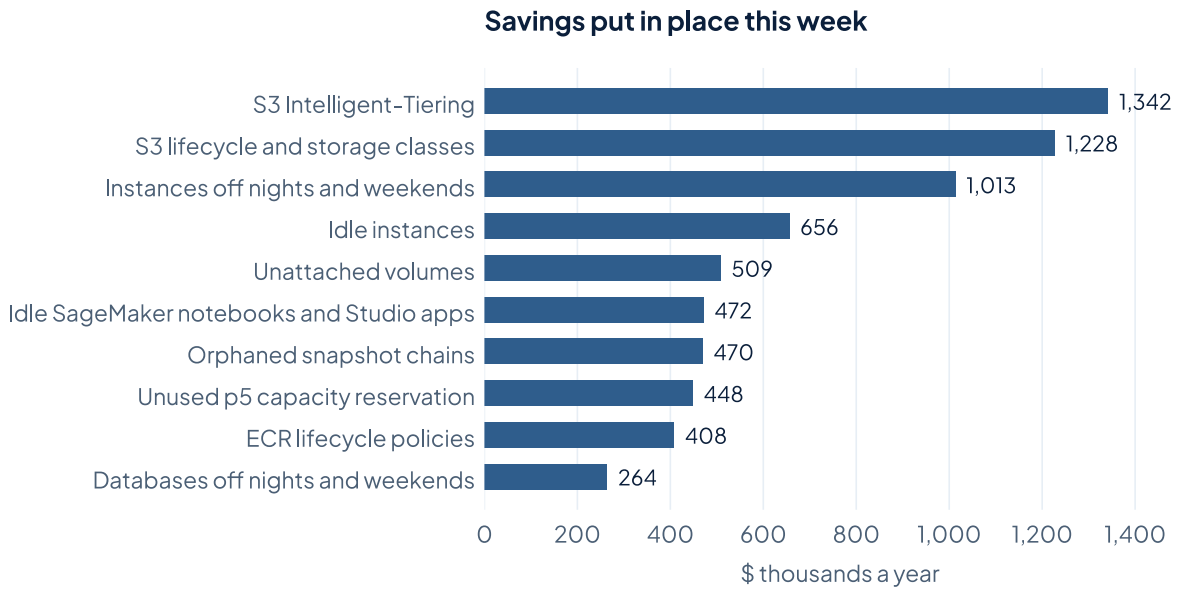
The figure is the estimate the Analysis made for each change, annualized. It is not yet a measurement. Week 6 replaces every estimate with the change in what the resource cost on your Cost and Usage Report, which decides the second invoice.

Two parts of the figure will not show on the bill at once. 161 buckets moved to S3 Intelligent-Tiering ($1.3 million a year, the largest single kind), and that class saves nothing until an object has gone 30 days unread in it, so the first saving appears about October 10, 2026 and the full saving by

December 9, 2026 (S3 lifecycle rules and Intelligent-Tiering). And the rules that move or expire data wait for the owners who hold it: $1.9 million a year did not go in this week, most of it waiting on a review or a trial, not a refusal.

Week 3 in numbers. Savings are estimates until the week-6 measurement.

Week 3 at a glance	Result
Changes put in place	9,302
Resources changed	65,190
Annualized savings, estimated	$7,772,626
Monthly savings, estimated	$647,719
One-time cost of making the changes	$97,097
Changes made by pull request to your repositories	1,300
Changes made by a scripted runbook under a change ticket	8,002 (across 652 tickets)
Week-3 savings not put in place (declined, deferred)	$1,911,025 a year



The ten largest savings put in place in week 3, estimated and annualized.

Decision for the sponsor

Priya Raman: no decision is needed to keep this week's savings. One needs a date: the data-governance council's review on October 14 decides the 51 data-lake retention changes held back in What was declined or deferred, worth $658,358 a year, the largest share of what is waiting.

What went in

Three of the engagement's deliverables landed this week. Deliverable 8, waste removed, is the largest group by count. Deliverable 11, storage fixed, is the largest by value. Deliverable 12 puts non-production on a schedule.

Week 3 by deliverable. Data-transfer and endpoint work, the rest of deliverable 11, lands in week 5.

Deliverable	Changes	Annualized, estimated	Share of week 3
8. Waste removed	5,594	$3,052,991	39%
11. Storage fixed (lifecycle, tiers, retention, volume types)	3,346	$3,407,496	44%
12. Non-production on a schedule	362	$1,312,138	17%

Every kind of change put in place in week 3, ranked by annualized savings. Monthly and annualized figures are net of the private pricing discount.

Kind of change	Changes	Resources	Monthly	Annualized	One-time
S3 Intelligent-Tiering	161	161	$111,837	$1,342,048	$28,433
S3 lifecycle and storage classes	82	82	$102,304	$1,227,646	$56,431
Instances off nights and weekends	307	497	$84,458	$1,013,492	$0
Idle instances	135	135	$54,688	$656,256	$1,727
Unattached volumes	1,325	1,325	$42,376	$508,516	$9,714
Idle SageMaker notebooks and Studio apps	46	46	$39,304	$471,651	$0
Orphaned snapshot chains	1,333	40,590	$39,128	$469,530	$0

Kind of change	Changes	Resources	Monthly	Annualized	One-time
Unused p5 capacity reservation	1	1	$37,367	$448,400	$0
ECR lifecycle policies	2	2	$34,009	$408,108	$0
Databases off nights and weekends	54	54	$22,033	$264,400	$0
Unused AMIs and their snapshots	2,502	18,943	$19,868	$238,410	$0
Unused Redshift cluster	1	1	$13,035	$156,419	$0
CloudWatch log retention	692	692	$12,395	$148,740	$0
EBS gp2 and io1 to gp3	2,362	2,362	$11,541	$138,492	$0
AWS Backup cold storage	2	2	$7,082	$84,988	$0
Idle non-production databases	13	13	$5,547	$66,569	$597
Debug logs to Infrequent Access	45	45	$4,789	$57,474	$0
WorkSpaces billed by the hour	1	1	$2,854	$34,247	$0
Kinesis streams with no traffic	2	2	$1,173	$14,078	$0
Long-stopped instances	41	41	$803	$9,632	$196
Idle load balancers	38	38	$595	$7,137	$0
Unassociated Elastic IPs	157	157	$533	$6,395	$0
Total	**9,302**	**65,190**	**$647,719**	**$7,772,626**	**$97,097**

The S3 work carries most of the one-time cost: $56,431 of transition-request charges on the lifecycle rules, repaid by that group's own savings in about 0.6 months, and $28,433 of transition charges into Intelligent-Tiering. One-time costs are reported here and again in week 6, where the guarantee is checked with them deducted as well. They are never netted into the annual savings.

The largest changes

These are the 25 largest changes by annualized savings. The channel column names where each change was made: a pull request in one of your four repositories, or a scripted runbook run by your operators under a change ticket, for resources that are not in code (the Kestrel Vision accounts, the sandboxes, and resources built by hand).

The largest week-3 changes, each with its owner and the channel it went through. Annualized savings are estimates.

ID	Change	Account	Resource	Owner	Channel	Applied	Annualized
W-6059	Unused p5 capacity reservation	crucible-training-prod	`cr-0c6c09c9a870c0fa9`	Hiro Tanaka	crucible-infra PR #3321	Sep 8	$448,400
T-3444	ECR lifecycle policies	sf-cicd	`sf-cicd:ecr`	Elena Vasquez	platform-landing-zone PR #1430	Sep 10	$339,005
W-6060	Unused Redshift cluster	corp-data-warehouse	`bi-legacy`	Lucas Ferreira	enterprise-infra PR #978	Sep 11	$156,419
T-2576	S3 Intelligent-Tiering	crucible-data	`sf-sensor-log-lake-sensor-rep…`	Hiro Tanaka	crucible-infra PR #3337	Sep 11	$94,792
S-0410	Instances off nights and weekends	fleet-perf	`fleet-perf-us-east-1-aac/ng-f…`	Sam Okafor	fleet-infra PR #8960	Sep 8	$73,435
T-2369	S3 lifecycle and storage classes	sf-log-archive	`sf-log-archive-vpc-flow-logs-…`	Nadia Petrova	platform-landing-zone PR #1427	Sep 10	$72,549
T-3447	ECR lifecycle policies	crucible-models	`crucible-models:ecr`	Hiro Tanaka	crucible-infra PR #3354	Sep 10	$69,103
T-2469	S3 lifecycle and storage classes	fleet-eu-prod	`sf-telemetry-lake-eu-telemetr…`	Sam Okafor	fleet-infra PR #8975	Sep 10	$67,303
T-2461	S3 lifecycle and storage classes	fleet-eu-prod	`sf-telemetry-lake-eu-telemetr…`	Sam Okafor	fleet-infra PR #8974	Sep 10	$65,199
T-3445	AWS Backup cold storage	sf-backup	`sf-backup:us-east-1:backup-vault`	Elena Vasquez	platform-landing-zone PR #1430	Sep 10	$65,090

ID	Change	Account	Resource	Owner	Channel	Applied	Annualized
T-2458	S3 Intelligent-Tiering	fleet-eu-prod	`sf-telemetry-lake-eu-telemetr…`	Sam Okafor	fleet-infra PR #8974	Sep 11	$62,676
S-0407	Instances off nights and weekends	fleet-perf	`fleet-perf-us-east-1-aac/ng-f…`	Sam Okafor	fleet-infra PR #8960	Sep 9	$55,051
T-2438	S3 lifecycle and storage classes	fleet-data-prod	`sf-telemetry-lake-telemetry-r…`	Lucas Ferreira	fleet-infra PR #8973	Sep 11	$54,722
T-2414	S3 lifecycle and storage classes	fleet-data-prod	`sf-telemetry-lake-telemetry-r…`	Lucas Ferreira	fleet-infra PR #8968	Sep 8	$53,974
T-2581	S3 Intelligent-Tiering	crucible-data	`sf-sensor-log-lake-sensor-rep…`	Hiro Tanaka	crucible-infra PR #3337	Sep 8	$52,359
T-2480	S3 Intelligent-Tiering	digital-twin-prod	`sf-site-maps-point-clouds-523…`	Sam Okafor	fleet-infra PR #8976	Sep 8	$49,962
T-2556	S3 Intelligent-Tiering	crucible-data	`sf-sensor-log-lake-sensor-log…`	Hiro Tanaka	crucible-infra PR #3333	Sep 11	$49,525
T-2409	S3 Intelligent-Tiering	fleet-data-prod	`sf-telemetry-lake-telemetry-c…`	Lucas Ferreira	fleet-infra PR #8967	Sep 10	$49,500
T-2418	S3 lifecycle and storage classes	fleet-data-prod	`sf-telemetry-lake-telemetry-r…`	Lucas Ferreira	fleet-infra PR #8968	Sep 10	$49,323
T-2394	S3 Intelligent-Tiering	fleet-data-prod	`sf-telemetry-lake-telemetry-r…`	Lucas Ferreira	fleet-infra PR #8965	Sep 8	$49,034
S-0409	Instances off nights and weekends	fleet-perf	`fleet-perf-us-east-1-aac/ng-f…`	Sam Okafor	fleet-infra PR #8960	Sep 11	$48,934
T-2518	S3 lifecycle and storage classes	crucible-training-prod	`sf-checkpoints-model-checkpoi…`	Hiro Tanaka	crucible-infra PR #3332	Sep 8	$47,010
W-0157	Idle instances	sandbox-ml	`i-0d8c93123ba6b74bd`	Hiro Tanaka	change CHG0041875 (scripted runbook)	Sep 8	$46,307

ID	Change	Account	Resource	Owner	Channel	Applied	Annualized
T-2367	S3 lifecycle and storage classes	sf-log-archive	`sf-log-archive-vpc-flow-logs-…`	Nadia Petrova	platform-landing-zone PR #1426	Sep 11	$45,657
W-6029	Idle SageMaker notebooks and Studio apps	crucible-research	`studio-studio-app-83776`	Hiro Tanaka	crucible-infra PR #3318	Sep 8	$42,205

How each change was made

1,300 of the 9,302 changes went through pull requests, which is the route we prefer: the owner reviews the diff, the pipeline applies it, and the repository history is the rollback record. The other 8,002 are resources no repository defines, so we wrote a runbook script and your operators ran it under 652 change tickets, as your change process requires. We never held standing write access to production: Vantalect writes the pull request or the script and the dry-run output, and a StrataForge engineer merges or runs it.

Where the week-3 changes were made. A pull request changes resources a repository defines; a ticket covers resources it does not.

Repository or ticket group	Changes	Share
`fleet-infra`	1,022	11.0%
`crucible-infra`	215	2.3%
`enterprise-infra`	35	0.4%
`platform-landing-zone`	28	0.3%
Scripted runbook, 652 change tickets	8,002	86.0%

Every change followed the same five steps.

1. The roadmap row (for example `W-6059`) names the resource, the change, the evidence, and the owner.
2. The owner answers within two business days: approve, decline, or defer. Their words are kept as written in What was declined or deferred.
3. For resources in code, the pull request carries the `cdk diff` or the CloudFormation change set. For the rest, the ticket carries the dry-run output of the runbook (The waste removal runbook).
4. Non-production runs first wherever the workload has a counterpart there: the sandbox, development, or staging account of the same team. The production change waits for a clean day.
5. The ledger records what changed and the exact command that reverses it.

Safety

Everything deleted this week left a copy, a name on it, and a way back, with three exceptions the owners signed off in writing: log events past a new retention period, S3 objects past an approved expiry rule, and container images past an approved expiry rule. The rules were the same for every kind.

- **Archive snapshots first.** A volume, an instance, a database, or a cluster is snapshotted (or imaged) before it is removed, tagged with the change ticket and a `Retain-Until` date 90 days out, and moved to the snapshot archive tier where AWS offers it. Archived snapshots cost $0.0125 per GB-month against $0.05 for a standard snapshot.
- **The Recycle Bin behind the deletes.** Orphaned snapshots and unused AMIs are removed under Recycle Bin retention rules (The waste removal runbook), so a wrong deletion is recoverable for seven days with a single command. A Region-level rule holds every deleted snapshot and image in the Region for the week, including ones other jobs delete, and the storage is billed for that week.
- **Owner sign-off, in writing.** The ticket or pull request holds the owner's approval. Resources an owner declined are untouched, and nothing was removed from an account tagged for legal hold.
- **Non-production first**, then production on a clean day, and never two production changes to the same workload on the same day.
- **Dry run by default.** The runbook prints what it would do. It runs for real only with `--execute`, and only for rows the owner approved.

What is kept and how each kind of change is reversed.

Kind of change	What is kept	How to undo it
Unattached volumes	Archive-tier snapshot, 90 days	`restore-snapshot-tier`, then `create-volume` from the snapshot
Idle and long-stopped instances	AMI of the stopped instance (its snapshots tagged), the full `describe-instances` output saved	Launch the AMI with the instance type, subnet, security groups, instance profile, and tags in the saved file
Orphaned snapshots and unused AMIs	Recycle Bin, 7 days	`restore-snapshot-from-recycle-bin` for each snapshot first, then `restore-image-from-recycle-bin` for the image

Kind of change	What is kept	How to undo it
Idle databases	Final snapshot, kept 90 days and tagged with its `Retain-Until` date	`restore-db-instance-from-db-snapshot`
The idle Redshift cluster	Final snapshot, kept until the owner deletes it	`restore-from-cluster-snapshot`
Idle load balancers and Kinesis streams	A load balancer's description, attributes, tags, listeners, rules, certificates, and target groups; a stream's summary and tags	Recreate the load balancer from the saved files, then its listeners and rules (the target groups were not deleted, and the DNS name changes); recreate the stream with the saved shard count
Elastic IPs	The address and its allocation in a saved file	`allocate-address --address <ip>` takes it back if nobody has claimed it since; once someone has, it is gone. We checked DNS, allow lists, and the NAT and VPN configuration before each release
Capacity reservation	The previous instance count in the ledger	`modify-capacity-reservation` back to the old count, subject to capacity being available
Instance and database schedules	The `Schedule` tag on each resource	Start the resource by hand (or let a start run reach it while it is still tagged), then remove the tag. A start run looks only for tagged resources, so removing the tag from a stopped one leaves it stopped
EKS node groups scaled to zero	The node group's previous minimum, desired, and maximum size in the ledger	Run the start action by hand, or set the saved sizes with `aws eks update-nodegroup-config`
S3 lifecycle, ECR, and log retention	The previous configuration saved with `get-*` before the change, or a note that none existed	Reapply the saved file; where none existed, delete the rule set (`delete-bucket-lifecycle`, `delete-lifecycle-policy`), and for log groups `delete-retention-policy`. Expired objects, images, and log events are not restored, so expiry ran only where the owner approved it
S3 Intelligent-Tiering	The previous lifecycle configuration	Reapply it. Objects stay in the tier they reached; only a copy returns them to Standard
Backup tiers	The previous plan version	Reapply the previous plan. Points already moved stay moved; an archived EBS point takes one to three days to restore
Volume type (gp2, io1)	The previous volume type in the ledger	`modify-volume` back, once the 6-hour cool-down passes

What was declined or deferred

$1.9 million a year of week-3 savings did not go in. We list the 20 largest below with each owner's reason in their words, kept factual. Of these, 12 are deferred, 4 declined, and 4 approved at the readout and declined afterward.

The largest week-3 items that were not put in place, with the reason each owner gave.

ID	Resource	Owner	Status	Annualized	Reason, as the owner gave it
T-2565	`sf-sensor-log-lake-sensor-logs-7065-us-…`	Hiro Tanaka	deferred	$103,784	the data owner wants the data-governance council to review the retention change first (October 14)
T-2638	`sf-labeled-datasets-image-datasets-9098…`	Jonah Reyes	deferred	$85,053	the data owner wants the data-governance council to review the retention change first (October 14)
T-2520	`sf-checkpoints-model-checkpoints-7042-u…`	Hiro Tanaka	deferred	$83,277	the data owner wants the data-governance council to review the retention change first (October 14)
T-2515	`sf-episode-results-sim-results-7019-us-…`	Hiro Tanaka	approved-declined	$81,685	approved at the readout; the Crucible team later chose to keep a year of results for a regression study
T-2617	`sf-hana-backups-backups-8016-us-east-1-001`	Ana Kowalski	deferred	$73,430	the SOX controls owner signs off on backup retention first
S-0408	`fleet-perf-us-east-1-aac/ng-fleet-perf-…`	Sam Okafor	deferred	$66,551	the owning team asked for a two-week trial in one account first
T-2507	`sf-episode-results-sim-results-7019-us-…`	Hiro Tanaka	approved-declined	$59,685	approved at the readout; the Crucible team later chose to keep a year of results for a regression study
T-2629	`sf-labeled-datasets-image-datasets-9098…`	Jonah Reyes	deferred	$52,810	the data owner wants the data-governance council to review the retention change first (October 14)
T-2513	`sf-episode-results-sim-results-7019-us-…`	Hiro Tanaka	approved-declined	$46,691	approved at the readout; the Crucible team later chose to keep a year of results for a regression study

ID	Resource	Owner	Status	AnnualizedReason, as the owner gave it
T-2616	`sf-hana-backups-backups-8016-us-east-1-000`	Ana Kowalski	deferred	$46,311the SOX controls owner signs off on backup retention first
S-0395	`fleet-staging-us-east-1-d6b/ng-fleet-st…`	Sam Okafor	deferred	$44,513the owning team asked for a two-week trial in one account first
T-2557	`sf-sensor-log-lake-sensor-logs-7065-us-…`	Hiro Tanaka	deferred	$38,530the data owner wants the data-governance council to review the retention change first (October 14)
T-2506	`sf-episode-results-sim-results-7019-us-…`	Hiro Tanaka	approved-declined	$37,711approved at the readout; the Crucible team later chose to keep a year of results for a regression study
S-0315	`i-0616f232ffb61c560`	Ana Kowalski	declined	$35,636SAP quality systems run month-end close tests overnight
S-0316	`i-05409f656079d9781`	Ana Kowalski	declined	$35,636SAP quality systems run month-end close tests overnight
S-0317	`i-0de8671e7757cb9d2`	Ana Kowalski	declined	$35,636SAP quality systems run month-end close tests overnight
S-0318	`i-0c22733034e5f93ce`	Ana Kowalski	declined	$35,636SAP quality systems run month-end close tests overnight
T-2528	`sf-sensor-log-lake-sensor-logs-7065-us-…`	Hiro Tanaka	deferred	$33,452the data owner wants the data-governance council to review the retention change first (October 14)
T-2567	`sf-sensor-log-lake-sensor-logs-7065-us-…`	Hiro Tanaka	deferred	$28,264the data owner wants the data-governance council to review the retention change first (October 14)
S-0394	`fleet-staging-us-east-1-d6b/ng-fleet-st…`	Sam Okafor	deferred	$26,537the owning team asked for a two-week trial in one account first

Four patterns account for most of it.

- **Data-lake retention goes to the data-governance council.** The owners of the large telemetry, sensor, image, and checkpoint buckets want the council to review any change to how long data is kept before we move or expire it. 51 changes, worth $658,358 a year, wait on the data-governance council's review on October 14. The rules are written and tested, and they go in by October 30, 2026 if the council agrees.
- **Schedules start with a trial.** The owners of 108 instance, node group, and database schedules, worth $502,856 a year, asked for a two-week trial in one account. The trial is the production-safe way to find the workload that did not look idle, and the rest follows by October 9, 2026.
- **Backup retention waits for SOX sign-off.** The SAP backup retention is a SOX control, so 2 lifecycle rules, worth $119,741 a year, wait on the SOX controls owner's sign-off on backup retention. Nothing shortens that retention until the controls owner signs.
- **Approved at the readout, declined afterward.** The Crucible team approved the lifecycle rules for the simulation results buckets, then chose to keep a year of results for a regression study. That is the owner's call and we did not push. Under the engagement's rule, an item approved at the readout and later declined counts toward the guarantee at its estimate, and week 6 reports it separately (its own section in the week-6 report).

Owners also declined smaller items for reasons that need no argument: SAP quality systems run month-end close tests overnight, so scheduling them off would break a business process, and some resources are held for a legal hold, a research reproducibility check, or a customer demo. Each has an owner's name on it.

The exact mechanism

Everything below is the code we used, reduced to what you need to run it again. Nothing here changes a resource until it runs against an approved list. The repositories named are the four in scope: `platform-landing-zone`, `fleet-infra`, `crucible-infra`, and `enterprise-infra`. Runbook scripts live in `platform-landing-zone/finops/runbooks/`, and your operators run them from the shared services account with a role assumed in the target account.

The waste removal runbook

The input is the approved rows of the waste register (Deliverable 2), exported as a CSV with these columns: `id,kind,profile,region,resource,ticket`. The `profile` is the AWS CLI profile that assumes the remediation role in that account. The script writes one ledger line per action; keep the ledger with the change ticket.

First, put the Recycle Bin rules in place in every account and Region that has snapshots or images. Deletions of unreferenced snapshots and AMIs are then recoverable for seven days.

```bash
aws rbin create-rule --resource-type EBS_SNAPSHOT \
  --retention-period RetentionPeriodValue=7,RetentionPeriodUnit=DAYS \
  --description "Seven-day safety net for cost cleanup" \
  --region <region> --profile <profile>
aws rbin create-rule --resource-type EC2_IMAGE \
  --retention-period RetentionPeriodValue=7,RetentionPeriodUnit=DAYS \
  --description "Seven-day safety net for cost cleanup" \
  --region <region> --profile <profile>
```

The rule for images keeps the deregistered AMI and the rule for snapshots keeps its snapshots, so the runbook creates both. An image restored from the bin needs its snapshots restored first.

Then the runbook. It handles each kind of waste removed this week; each branch checks the resource is still in the state the register recorded, so a resource someone started using since the Analysis is skipped, not removed. Every call that matters is checked: a snapshot that fails or has not completed keeps its volume, an image that is not available keeps its instance, and a failed delete is logged as FAIL and the run goes on to the next row. An unexpected CLI error stops the run with the ledger up to that row, and a rerun of the same file skips the rows already done, because their resources are gone.

```bash
#!/usr/bin/env bash
# platform-landing-zone/finops/runbooks/waste-remove.sh
# Usage: waste-remove.sh <approved.csv> [--execute]
# Needs bash 4.4 or later, GNU date, and jq. The Linux runner in the shared services account
has all three.
set -euo pipefail

PLAN="$1"
MODE="dry-run"; [[ "${2:-}" == "--execute" ]] && MODE="execute"
LEDGER="ledger-$(date -u +%Y%m%dT%H%M%SZ).tsv"
RETAIN_UNTIL="$(date -u -d '+90 days' +%F)"

run()  { if [[ "$MODE" == "execute" ]]; then "$@"; else echo "DRY-RUN: $*" >&2; fi; }
note() { local act="$2"; [[ "$MODE" == "execute" ]] || act="DRY-RUN $2"
         printf '%s\t%s\t%s\t%s\n' "$(date -u +%FT%TZ)" "$1" "$act" "$3" | tee -a "$LEDGER"
>&2; }
tags() { echo "ResourceType=$1,Tags=[{Key=Archive,Value=$ticket},{Key=Retain-
Until,Value=$RETAIN_UNTIL},{Key=Team,Value=finops}]"; }

do_it() {                          # do_it <id> <verb> <detail> <command...>: record the verb
on success, FAIL on failure
  local id="$1" verb="$2" detail="$3"; shift 3
  if run "$@" >/dev/null; then note "$id" "$verb" "$detail"; else note "$id" FAIL "$detail ($1
$2 $3 failed)"; return 1; fi
}
state() {                          # state <volume|instance|image|snapshot> <id>; prints "gone"
```

```
when it cannot be read
  case "$1" in
    volume)   aws ec2 describe-volumes --volume-ids "$2" --query 'Volumes[0].State' --output
text ;;
    instance) aws ec2 describe-instances --instance-ids "$2" --query
'Reservations[0].Instances[0].State.Name' --output text ;;
    image)    aws ec2 describe-images --image-ids "$2" --query 'Images[0].State' --output
text ;;
    snapshot) aws ec2 describe-snapshots --snapshot-ids "$2" --query 'Snapshots[0].State' --
output text ;;
  esac 2>/dev/null || echo gone
}
wait_for() {                        # the CLI waiters give up after 10 to 40 minutes; this polls
for up to 6 hours
  local n
  for n in $(seq 1 720); do
    case "$(state "$1" "$2")" in "$3") return 0 ;; error|failed|gone) return 1 ;; esac
    sleep 30
  done
  return 1
}
quiet() {                           # quiet <namespace> <metric> <dimension> <value> <Sum|
Maximum>: nothing in the last 7 days
  local v
  v=$(aws cloudwatch get-metric-statistics --namespace "$1" --metric-name "$2" --dimensions
"Name=$3,Value=$4" \
        --statistics "$5" --period 604800 --start-time "$(date -u -d '-7 days' +%FT%TZ)" --end-
time "$(date -u +%FT%TZ)" \
        --query "max(Datapoints[].$5)" --output text) || return 1
  [[ "$v" =~ ^(0|0\.0|None)$ ]]
}
archive_volume() {                  # snapshot, wait until completed, move to the archive tier;
prints the snapshot ID
  local snap
  snap=$(aws ec2 create-snapshot --volume-id "$1" --description "archive $ticket" \
          --tag-specifications "$(tags snapshot)" --query SnapshotId --output text) || return 1
  [[ "$snap" == snap-* ]] || return 1
  wait_for snapshot "$snap" completed || return 1
  aws ec2 modify-snapshot-tier --snapshot-id "$snap" --storage-tier archive >/dev/null \
    || note "$id" WARN "$snap stays in the standard tier; the copy is good, the tier move
failed" >&2
  echo "$snap"
}
save_lb() {                         # everything needed to recreate a load balancer
  local arn="$1" base l
  base="describe-$(basename "$1")"
  aws elbv2 describe-load-balancers --load-balancer-arns "$arn" > "$base-lb.json"
  aws elbv2 describe-load-balancer-attributes --load-balancer-arn "$arn" > "$base-
attributes.json"
  aws elbv2 describe-tags --resource-arns "$arn" > "$base-tags.json"
  aws elbv2 describe-target-groups --load-balancer-arn "$arn" > "$base-target-groups.json"
  aws elbv2 describe-listeners --load-balancer-arn "$arn" > "$base-listeners.json"
  for l in $(jq -r '.Listeners[].ListenerArn' "$base-listeners.json"); do
    aws elbv2 describe-rules --listener-arn "$l"
    aws elbv2 describe-listener-certificates --listener-arn "$l"
  done | jq -s . > "$base-rules-and-certificates.json"
}
```

```
while IFS=, read -r id kind profile region resource ticket; do
  [[ "$id" == "id" ]] && continue
  export AWS_PROFILE="$profile" AWS_DEFAULT_REGION="$region" AWS_PAGER=""
  case "$kind" in
    unattached-volume)
      [[ "$(state volume "$resource")" == "available" ]] || { note "$id" SKIP "$resource is
attached or gone"; continue; }
      if [[ "$MODE" == "execute" ]]; then
        snap=$(archive_volume "$resource") && [[ "$snap" == snap-* ]] \
          || { note "$id" FAIL "no completed snapshot of $resource; volume kept"; continue; }
        note "$id" ARCHIVED "$resource -> $snap"
        [[ "$(state volume "$resource")" == "available" ]] || { note "$id" SKIP "$resource was
attached during the snapshot"; continue; }
      else
        note "$id" ARCHIVED "$resource -> a new archive-tier snapshot"
      fi
      do_it "$id" DELETED "$resource" aws ec2 delete-volume --volume-id "$resource" ||
continue ;;
    idle-instance|stopped-instance)
      st=$(state instance "$resource")
      if [[ "$st" != "stopped" && ( "$kind" == "stopped-instance" || "$st" != "running" ) ]];
then
        note "$id" SKIP "$resource is $st now; the register recorded $kind"; continue; fi
      [[ "$(aws ec2 describe-instance-attribute --instance-id "$resource" --attribute
disableApiTermination \
            --query DisableApiTermination.Value --output text)" == "False" ]] \
        || { note "$id" SKIP "$resource has termination protection; the owner decides";
continue; }
      aws ec2 describe-instances --instance-ids "$resource" > "describe-$resource.json"
      do_it "$id" STOPPED "$resource" aws ec2 stop-instances --instance-ids "$resource" ||
continue
      if [[ "$MODE" == "execute" ]]; then
        wait_for instance "$resource" stopped || { note "$id" FAIL "$resource did not stop;
kept"; continue; }
        ami=$(aws ec2 create-image --instance-id "$resource" --name "archive-$ticket-$resource"
\
                --tag-specifications "$(tags image)" "$(tags snapshot)" --query ImageId --
output text) \
          && [[ "$ami" == ami-* ]] && wait_for image "$ami" available \
          || { note "$id" FAIL "no available image of $resource; instance kept, stopped";
continue; }
        note "$id" ARCHIVED "$resource -> $ami"
        [[ "$(state instance "$resource")" == "stopped" ]] || { note "$id" SKIP "$resource was
started again; $ami kept"; continue; }
      else
        note "$id" ARCHIVED "$resource -> a new image with tagged snapshots"
      fi
      do_it "$id" TERMINATED "$resource (settings in describe-$resource.json)" \
          aws ec2 terminate-instances --instance-ids "$resource" || continue ;;
    orphan-snapshot)
      [[ "$(state snapshot "$resource")" == "completed" ]] || { note "$id" SKIP "$resource is
gone or not completed"; continue; }
      [[ "$(aws ec2 describe-images --owners self --filters "Name=block-device-
mapping.snapshot-id,Values=$resource" \
            --query 'length(Images)' --output text)" == "0" ]] \
        || { note "$id" SKIP "$resource backs an image now"; continue; }
```

```
      do_it "$id" DELETED "$resource" aws ec2 delete-snapshot --snapshot-id "$resource" ||
continue ;;
    orphan-ami)
      [[ "$(state image "$resource")" == "available" ]] || { note "$id" SKIP "$resource is gone
or not available"; continue; }
      [[ "$(aws ec2 describe-instances --filters "Name=image-id,Values=$resource" \
            --query 'length(Reservations)' --output text)" == "0" ]] \
        || { note "$id" SKIP "an instance runs from $resource now"; continue; }
      mapfile -t snaps < <(aws ec2 describe-images --image-ids "$resource" \
        --query 'Images[0].BlockDeviceMappings[?Ebs].Ebs.SnapshotId' --output text | tr '\t'
'\n')
      do_it "$id" DEREGISTERED "$resource" aws ec2 deregister-image --image-id "$resource" ||
continue
      for s in "${snaps[@]}"; do do_it "$id" DELETED "$s" aws ec2 delete-snapshot --snapshot-id
"$s" || true; done ;;
    idle-eip)
      aws ec2 describe-addresses --allocation-ids "$resource" > "describe-$resource.json" \
        || { note "$id" SKIP "$resource is gone"; continue; }
      [[ "$(jq -r '.Addresses[0].AssociationId // "none"' "describe-$resource.json")" ==
"none" ]] \
        || { note "$id" SKIP "$resource is associated now"; continue; }
      do_it "$id" RELEASED "$resource ($(jq -r '.Addresses[0].PublicIp' "describe-
$resource.json"))" \
          aws ec2 release-address --allocation-id "$resource" || continue ;;
    idle-load-balancer)
      [[ "$(aws elbv2 describe-load-balancer-attributes --load-balancer-arn "$resource" \
            --query "Attributes[?Key=='deletion_protection.enabled'].Value | [0]" --output text
2>/dev/null || echo gone)" == "false" ]] \
        || { note "$id" SKIP "$resource has deletion protection, or is gone; the owner
decides"; continue; }
      case "$resource" in
        *:loadbalancer/app/*) quiet AWS/ApplicationELB RequestCount LoadBalancer "$
{resource#*:loadbalancer/}" Sum ;;
        *:loadbalancer/net/*) quiet AWS/NetworkELB NewFlowCount LoadBalancer "$
{resource#*:loadbalancer/}" Sum ;;
        *) false ;;
      esac || { note "$id" SKIP "$resource shows traffic, or is a type this script does not
check"; continue; }
      save_lb "$resource"
      do_it "$id" DELETED "$resource (description, listeners, rules, and target groups saved)"
\
          aws elbv2 delete-load-balancer --load-balancer-arn "$resource" || continue ;;
    idle-database)
      [[ "$(aws rds describe-db-instances --db-instance-identifier "$resource" \
            --query 'DBInstances[0].DeletionProtection' --output text 2>/dev/null || echo
gone)" == "False" ]] \
        || { note "$id" SKIP "$resource has deletion protection, or is gone; the owner
decides"; continue; }
      quiet AWS/RDS DatabaseConnections DBInstanceIdentifier "$resource" Maximum \
        || { note "$id" SKIP "$resource has connections now"; continue; }
      do_it "$id" DELETED "$resource (final snapshot final-$resource-$ticket, keep until
$RETAIN_UNTIL)" \
          aws rds delete-db-instance --db-instance-identifier "$resource" \
          --final-db-snapshot-identifier "final-$resource-$ticket" --no-delete-automated-
backups || continue ;;
    idle-redshift)
      quiet AWS/Redshift DatabaseConnections ClusterIdentifier "$resource" Maximum \
```

```
        || { note "$id" SKIP "$resource has connections now"; continue; }
      do_it "$id" DELETED "$resource (final snapshot final-$resource-$ticket)" \
          aws redshift delete-cluster --cluster-identifier "$resource" \
          --final-cluster-snapshot-identifier "final-$resource-$ticket" || continue ;;
    idle-notebook)
      [[ "$(aws sagemaker describe-notebook-instance --notebook-instance-name "$resource" \
            --query NotebookInstanceStatus --output text 2>/dev/null || echo gone)" ==
"InService" ]] \
        || { note "$id" SKIP "$resource is not running"; continue; }
      do_it "$id" STOPPED "$resource" aws sagemaker stop-notebook-instance --notebook-instance-
name "$resource" || continue ;;
    dead-stream)
      quiet AWS/Kinesis IncomingRecords StreamName "$resource" Sum \
        || { note "$id" SKIP "$resource received records this week"; continue; }
      aws kinesis describe-stream-summary --stream-name "$resource" > "describe-$resource.json"
      aws kinesis list-tags-for-stream --stream-name "$resource" > "tags-$resource.json"
      do_it "$id" DELETED "$resource" aws kinesis delete-stream --stream-name "$resource" ||
continue ;;
    *) note "$id" SKIP "no handler for kind $kind" ;;
  esac
done < "$PLAN"
echo "Ledger: $LEDGER (mode: $MODE)" >&2
```

A deleted database keeps its final snapshot for 90 days. Once the snapshot is available, tag it with the date it may go, so the cleanup that follows has something to read.

```bash
aws rds add-tags-to-resource --resource-name <final-snapshot-arn> \
  --tags Key=Archive,Value=<ticket> Key=Retain-Until,Value=<date-90-days-out>
```

Verify with `describe-*` on a sample of resources and, after one full day, with the Cost and Usage Report query in Verifying a change on the bill. Roll back with the right-hand column of What is kept and how each kind of change is reversed.

The idle notebooks are stopped by the runbook, then kept from drifting back with a lifecycle configuration that stops a notebook after 60 minutes without kernel activity. Create it once per account, attach it to each notebook (a notebook must be stopped to change its lifecycle configuration), and give the notebook role `sagemaker:StopNotebookInstance`. Jupyter on a notebook instance serves HTTPS on port 8443 with its own certificate, and the system Python has no boto3, so the script calls the local port without certificate checks and runs under the notebook's `JupyterSystemEnv` Python. A browser tab left open does not keep a notebook running; only kernel activity does. With no notebook open there is no kernel to measure, so the clock runs from when the instance last changed state.

```bash
cat > on-start.sh <<'SCRIPT'
#!/bin/bash
set -e
cat > /home/ec2-user/autostop.py <<'PY'
```

```
import json, ssl, time, urllib.request
from datetime import datetime, timezone
import boto3

IDLE_SECONDS = 3600
name = json.load(open("/opt/ml/metadata/resource-metadata.json"))["ResourceName"]
local = ssl.create_default_context()
local.check_hostname = False
local.verify_mode = ssl.CERT_NONE                      # the instance's own self-signed
certificate
sessions = json.load(urllib.request.urlopen("https://localhost:8443/api/sessions",
context=local))


def age(stamp):                                        # seconds since an ISO 8601 UTC time
    then = datetime.strptime(stamp[:19], "%Y-%m-%dT%H:%M:%S").replace(tzinfo=timezone.utc)
    return time.time() - then.timestamp()


sm = boto3.client("sagemaker")
if sessions:
    idle = all(s["kernel"]["execution_state"] == "idle" and age(s["kernel"]["last_activity"]) >
IDLE_SECONDS
               for s in sessions)
else:
    changed = sm.describe_notebook_instance(NotebookInstanceName=name)["LastModifiedTime"]
    idle = time.time() - changed.timestamp() > IDLE_SECONDS
if idle:
    sm.stop_notebook_instance(NotebookInstanceName=name)
PY
PYTHON=/home/ec2-user/anaconda3/envs/JupyterSystemEnv/bin/python
(crontab -l 2>/dev/null || true; echo "*/5 * * * * $PYTHON /home/ec2-user/autostop.py") |
crontab -
SCRIPT
aws sagemaker create-notebook-instance-lifecycle-config \
  --notebook-instance-lifecycle-config-name idle-shutdown-60 \
  --on-start Content="$(base64 < on-start.sh | tr -d '\n')"
aws sagemaker update-notebook-instance --notebook-instance-name <notebook-name> \
  --lifecycle-config-name idle-shutdown-60
```

The unused p5 capacity reservation (one of the largest waste items) is a one-line change in the repository that defines it. `crucible-infra` sets the reservation's instance count. The reservation was 5 x p5.48xlarge reserved, 2.9 in use on average, based on 30-day average of instances running in the reservation. The pull request changes it to 4 x p5.48xlarge; burst with EC2 Capacity Blocks for ML: training bursts above the reserved count on EC2 Capacity Blocks for ML, booked by the day when a run needs them.

```bash
aws ec2 modify-capacity-reservation --capacity-reservation-id <capacity-reservation-id> --
instance-count 4
aws ec2 describe-capacity-block-offerings --instance-type p5.48xlarge --instance-count 1 \
  --capacity-duration-hours 24 --start-date-range <start-iso8601> --end-date-range <end-
iso8601>
aws ec2 purchase-capacity-block --capacity-block-offering-id <offering-id> --instance-platform
Linux/UNIX
```

Reserved capacity is the one waste item where rollback depends on AWS having capacity to give back, which is why the owner's team keeps what training actually uses and why we rated this change Medium risk and made it with Hiro Tanaka's team on the call.

Non-production schedules

Deliverable 12 stops non-production instances and databases outside 7 a.m. to 7 p.m. on weekdays, local time, which is where the usage data showed no activity, and scales non-production EKS node groups to zero as whole groups. A tag opts a resource in: `Schedule=weekdays-7-19`. One Amazon EventBridge Scheduler pair (stop at 19:00, start at 07:00, Monday to Friday, `America/New_York`) calls one Lambda function that finds every tagged resource in its account and Region. The function skips anything in an Auto Scaling group or an EKS or EMR cluster, because those replace what you stop. If one instance in a batch cannot be stopped, the function retries the batch one instance at a time, and it fails the invocation at the end so an alarm fires.

```python
# fleet-infra/finops/schedules/handler.py
import logging
import boto3
from botocore.exceptions import ClientError

log = logging.getLogger()
log.setLevel(logging.INFO)
tagging = boto3.client("resourcegroupstaggingapi")
ec2 = boto3.client("ec2")
rds = boto3.client("rds")

SKIP = {"aws:autoscaling:groupName", "eks:cluster-name", "karpenter.sh/nodepool",
        "aws:elasticmapreduce:job-flow-id"}


def tagged(schedule, types):
    for page in tagging.get_paginator("get_resources").paginate(
            TagFilters=[{"Key": "Schedule", "Values": [schedule]}], ResourceTypeFilters=types):
        for item in page["ResourceTagMappingList"]:
            if {t["Key"] for t in item["Tags"]} & SKIP:
                log.info("skip managed resource %s", item["ResourceARN"])
                continue
```

```
            yield item["ResourceARN"]


def change(action, ids):
    """Stop or start instances; returns the IDs that failed. One bad ID fails a whole call, so
retry singly."""
    call = ec2.stop_instances if action == "stop" else ec2.start_instances
    try:
        call(InstanceIds=ids)
        return []
    except ClientError as err:
        if len(ids) > 1:
            return [i for one in ids for i in change(action, [one])]
        log.error("ec2 %s failed for %s: %s", action, ids[0], err.response["Error"]["Code"])
        return ids


def main(event, _context):
    action, schedule = event["action"], event["schedule"]
    ids = [a.split("/")[-1] for a in tagged(schedule, ["ec2:instance"])]
    failed = []
    for i in range(0, len(ids), 100):
        failed += change(action, ids[i:i + 100])
    for arn in tagged(schedule, ["rds:db", "rds:cluster"]):
        kind, name = arn.split(":")[-2:]
        try:
            if kind == "cluster":
                (rds.stop_db_cluster if action == "stop" else rds.start_db_cluster)
(DBClusterIdentifier=name)
            else:
                (rds.stop_db_instance if action == "stop" else rds.start_db_instance)
(DBInstanceIdentifier=name)
        except ClientError as err:
            log.error("rds %s %s failed: %s", action, name, err.response["Error"]["Code"])
            failed.append(name)
    log.info("%s %s: %d instances, %d failures", action, schedule, len(ids), len(failed))
    if failed:                                    # fail the invocation so the Errors alarm
below fires
        raise RuntimeError(f"{action} failed for {len(failed)} resources: {failed[:10]}")
```

The stack that deploys it. It is one construct in `fleet-infra` (and the same file in `crucible-infra` and `enterprise-infra` for their non-production accounts), so a new account gets the schedule by adding the stack.

```typescript
// fleet-infra/lib/nonprod-schedule-stack.ts
import { Duration, Stack, StackProps } from 'aws-cdk-lib';
import * as cloudwatch from 'aws-cdk-lib/aws-cloudwatch';
import * as iam from 'aws-cdk-lib/aws-iam';
import * as lambda from 'aws-cdk-lib/aws-lambda';
import * as scheduler from 'aws-cdk-lib/aws-scheduler';
import { Construct } from 'constructs';

const SCHEDULE = 'weekdays-7-19';
```

```
const EVENTS = [
  { action: 'stop', expression: 'cron(0 19 ? * MON-FRI *)' },
  { action: 'start', expression: 'cron(0 7 ? * MON-FRI *)' },
];

export class NonProdScheduleStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);

    const fn = new lambda.Function(this, 'ScheduleFn', {
      runtime: lambda.Runtime.PYTHON_3_13,
      architecture: lambda.Architecture.ARM_64,
      handler: 'handler.main',
      code: lambda.Code.fromAsset('finops/schedules'),
      timeout: Duration.minutes(5),
    });
    fn.addToRolePolicy(new iam.PolicyStatement({
      actions: ['tag:GetResources', 'ec2:StopInstances', 'ec2:StartInstances',
                'rds:StopDBInstance', 'rds:StartDBInstance', 'rds:StopDBCluster',
'rds:StartDBCluster'],
      resources: ['*'],
    }));

    const role = new iam.Role(this, 'SchedulerRole', {
      assumedBy: new iam.ServicePrincipal('scheduler.amazonaws.com'),
    });
    fn.grantInvoke(role);

    // The handler raises when any stop or start fails; add an alarm action that pages the
owning team.
    fn.metricErrors({ period: Duration.minutes(5) }).createAlarm(this, 'ScheduleFailed', {
      threshold: 1,
      evaluationPeriods: 1,
      treatMissingData: cloudwatch.TreatMissingData.NOT_BREACHING,
    });

    for (const e of EVENTS) {
      new scheduler.CfnSchedule(this, `Weekday-${e.action}`, {
        name: `${SCHEDULE}-${e.action}`,
        scheduleExpression: e.expression,
        scheduleExpressionTimezone: 'America/New_York',
        flexibleTimeWindow: { mode: 'OFF' },
        target: {
          arn: fn.functionArn,
          roleArn: role.roleArn,
          input: JSON.stringify({ action: e.action, schedule: SCHEDULE }),
        },
      });
    }
  }
}
```

EKS node groups are scheduled as a unit, never node by node. Stopping one node's instance only makes the node group replace it, and a node group at zero nodes costs nothing, so each node group in a test or non-production cluster gets its own scale-to-zero pair, which calls the EKS API

directly. This is why the function above skips anything an Auto Scaling group or Karpenter manages. The windows come from the node group's own use: the performance-test groups in `fleet-infra` run only when pods are scheduled on them, and the saving is every hour outside that.

```typescript
// fleet-infra/lib/nodegroup-schedule.ts
import * as iam from 'aws-cdk-lib/aws-iam';
import * as scheduler from 'aws-cdk-lib/aws-scheduler';
import { Construct } from 'constructs';

export interface NodeGroupWindow {
  cluster: string;
  nodegroup: string;
  minSize: number;      // the size the group has today; the start action restores it
  desiredSize: number;
  start: string;        // for example cron(0 7 ? * MON-FRI *)
  stop: string;         // for example cron(0 19 ? * MON-FRI *)
}

export function scaleToZero(scope: Construct, role: iam.IRole, w: NodeGroupWindow, timezone:
string): void {
  const action = (id: string, expression: string, scaling: { MinSize: number; DesiredSize:
number }) =>
    new scheduler.CfnSchedule(scope, `${w.nodegroup}-${id}`, {
      name: `${w.nodegroup}-${id}`.slice(0, 64),
      scheduleExpression: expression,
      scheduleExpressionTimezone: timezone,
      flexibleTimeWindow: { mode: 'OFF' },
      target: {
        arn: 'arn:aws:scheduler:::aws-sdk:eks:updateNodegroupConfig',
        roleArn: role.roleArn,
        retryPolicy: { maximumRetryAttempts: 3 },
        input: JSON.stringify({ ClusterName: w.cluster, NodegroupName: w.nodegroup,
ScalingConfig: scaling }),
      },
    });
  action('stop', w.stop, { MinSize: 0, DesiredSize: 0 });
  action('start', w.start, { MinSize: w.minSize, DesiredSize: w.desiredSize });
}
```

The scheduler's role needs `eks:UpdateNodegroupConfig` on those node groups and nothing else. The maximum size is left alone, so the start action restores exactly what the group had. Verify each morning and evening, and record the sizes in the ledger before the first stop.

```bash
aws eks describe-nodegroup --cluster-name <cluster-name> --nodegroup-name <node-group> \
  --query 'nodegroup.scalingConfig'
```

Opt an instance or database in with the tag, in code or by CLI, and verify it the next morning.

```bash
aws ec2 create-tags --resources <instance-id> --tags Key=Schedule,Value=weekdays-7-19
aws rds add-tags-to-resource --resource-name <db-arn> --tags Key=Schedule,Value=weekdays-7-19
aws ec2 describe-instances --filters Name=tag:Schedule,Values=weekdays-7-19 \
  --query 'Reservations[].Instances[].[InstanceId,State.Name]' --output text
```

Two limits to know. RDS and Aurora restart a stopped database on their own after seven days, which this schedule never reaches (the longest stop is the 60 hours from Friday evening to Monday morning). And stopping a database does not stop its storage charge: the saving is the instance hours, which is why the estimate for databases is smaller than for instances. To roll back, start the stopped resources first (or let a start run reach them while they are still tagged), and only then remove the `Schedule` tag. Removing the tag from a stopped resource leaves it stopped, because the start run looks only for tagged resources.

S3 lifecycle rules and Intelligent-Tiering

$1,227,646 a year of the week's estimate is lifecycle rules on 82 buckets, and $1,342,048 a year is S3 Intelligent-Tiering on 161 more. Each bucket was sized from two measurements, not from its name. S3 Inventory says how many bytes are older than 90 days and older than 365 days, and the server access logs say how many bytes are read at all. For one of the largest, `sf-log-archive-vpc-flow-logs-2291-us-ea…`, the evidence was: S3 Inventory: 97% of bytes older than 90 days, 89% older than 365; access logs: 97% not read in 90 days. A rule keyed to object age moves only what is old, and the access logs price what the rule would get wrong.

That second part is a cost the estimate carries. An object that is read after it moves to a colder class is billed retrieval, so each lifecycle line nets that retrieval out of its saving. The line's notes say how much; the example above reads: "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)". Where access is too unpredictable for an age rule, the bucket gets S3 Intelligent-Tiering instead, which tiers by access, not by age, and charges no retrieval.

Intelligent-Tiering saves nothing on the first day

An object moves to the infrequent-access tier only after it has gone 30 days unread in Intelligent-Tiering, and to the archive instant-access tier after 90. The class also charges a monitoring fee per object, and objects under 128 KB are never moved, so a bucket of small files saves little. The week-3 changes put 161 buckets in the class, and the first saving shows on the bill about October 10, 2026, with the full saving by December 9, 2026. The week-6 measurement cannot count it, and does not: it reports the class separately until the saving is visible.

The rules by data class:

The lifecycle rule shapes by data class. Every rule also aborts incomplete multipart uploads after 7 days.

Data class	What the rule does	Why
Telemetry replicas	Replicate straight into Deep Archive; expire noncurrent versions after 180 days	Deep Archive bills a 180-day minimum; the replica pays its PUT price once per object
Raw telemetry, sensor replicas	Intelligent-Tiering for every object	Access is unpredictable
Sensor logs, image datasets, checkpoints	Glacier Instant Retrieval after 90 days; Deep Archive after 365 days where most bytes are older than a year; checkpoints expire after 30 days except release checkpoints	Age-based, from S3 Inventory. Waits for the data-governance council
Simulation results	Expire after 180 days	Approved at the readout, then declined by the owner
Build cache	Expire after 30 days, current and noncurrent	Rebuilt on demand
Flow and load balancer logs in `sf-log-archive`	Standard-IA after 30 days, Glacier after 90 days (flow logs), expire after 400 days; versions kept	The account keeps its versions, and the security engagement set its retention
SAP HANA backups	Glacier after 30 days, expire after 120 days	Waits for the SOX controls owner's sign-off
Cost and Usage Report	No tiering rule: it stays in Standard for its first 400 days	The showback and the week-6 measurement read it daily

Before applying, save the current configuration; it is your rollback.

```bash
b=<bucket>
if out=$(aws s3api get-bucket-lifecycle-configuration --bucket "$b" 2>&1); then
  jq '{Rules}' <<<"$out" > "before-$b.json"                          # reapply this file to
roll back
  jq -r '.TransitionDefaultMinimumObjectSize' <<<"$out" > "before-$b.minsize"
elif grep -q NoSuchLifecycleConfiguration <<<"$out"; then
  echo "no rules" > "before-$b.none"                                  # roll back with delete-
bucket-lifecycle
else
  echo "$out" >&2; exit 1                                             # AccessDenied or worse:
change nothing
fi
aws s3api get-bucket-versioning --bucket "$b" --query Status --output text
```

The saved file keeps only `Rules`, because `put-bucket-lifecycle-configuration` rejects the other keys the `get` returns. If the bucket had a non-default `TransitionDefaultMinimumObjectSize`, the `.minsize` file holds it, and the rollback passes it back with `--transition-default-minimum-object-size`.

The age-based shape, for a bucket whose bytes are mostly old and rarely read. The size filter matters: S3 bills a minimum of 128 KB per object in Glacier Instant Retrieval, and since September 2024 a rule does not transition smaller objects unless it says so, so a lake full of small files would pay more, not less, if tiered.

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

The Intelligent-Tiering shape moves every object into the class on day zero, and the class does the rest.

```JSON
{
  "Rules": [
    {"ID": "abort-incomplete-multipart", "Status": "Enabled", "Filter": {"Prefix": ""},
     "AbortIncompleteMultipartUpload": {"DaysAfterInitiation": 7}},
    {"ID": "intelligent-tiering", "Status": "Enabled", "Filter": {"Prefix": ""},
     "Transitions": [{"Days": 0, "StorageClass": "INTELLIGENT_TIERING"}]}
  ]
}
```

A replica bucket takes its Deep Archive class from the replication rule on the source bucket, not from a lifecycle rule. Putting a replication configuration replaces the whole of it, so read the existing one, change only the destination's storage class in that copy, and put the edited copy back. Add the 180-day noncurrent expiry to the replica bucket separately.

```bash
aws s3api get-bucket-replication --bucket <source-bucket> > replication-before.json
jq '.ReplicationConfiguration | (.Rules[] | select(.ID == "<rule-id>")
| .Destination.StorageClass) = "DEEP_ARCHIVE"' \
  replication-before.json > replication-after.json
aws s3api put-bucket-replication --bucket <source-bucket> --replication-configuration file://
replication-after.json
```

The rule changes only objects replicated from now on. Replicas already in the bucket keep their class until a lifecycle transition or an S3 Batch Operations copy moves them, each at the $0.05 per 1,000 objects that a Deep Archive request costs. Rolling back is `put-bucket-replication` with the `.ReplicationConfiguration` of `replication-before.json`.

```bash
aws s3api put-bucket-lifecycle-configuration --bucket <bucket> --lifecycle-configuration
file://lifecycle.json
```

That call replaces the whole configuration, so `lifecycle.json` is the saved `Rules` with the new rules merged in, never the new rules alone.

For a bucket that `fleet-infra`, `crucible-infra`, or `enterprise-infra` defines, make the same change on the construct so the next deploy does not undo it.

```typescript
new s3.Bucket(this, 'SensorLogLake', {
  versioned: true,
  lifecycleRules: [
    { abortIncompleteMultipartUploadAfter: Duration.days(7) },
    { noncurrentVersionExpiration: Duration.days(30) },
    {
      objectSizeGreaterThan: 131072,
      transitions: [
        { storageClass: s3.StorageClass.GLACIER_INSTANT_RETRIEVAL, transitionAfter:
Duration.days(90) },
        { storageClass: s3.StorageClass.DEEP_ARCHIVE, transitionAfter: Duration.days(365) },
      ],
    },
  ],
});

new s3.Bucket(this, 'TelemetryRaw', {
  versioned: false,
  lifecycleRules: [
    { abortIncompleteMultipartUploadAfter: Duration.days(7) },
    { transitions: [{ storageClass: s3.StorageClass.INTELLIGENT_TIERING, transitionAfter:
Duration.days(0) }] },
  ],
});
```

Transitions cost money once: $0.02 per 1,000 objects into Glacier Instant Retrieval, $0.05 per 1,000 into Deep Archive (and for a PUT into it), and $0.01 per 1,000 into Intelligent-Tiering, and Deep Archive has a 180-day minimum. That is the $84,863 of one-time cost on the S3 lines in Every kind of change put in place in week 3, ranked by annualized savings. Monthly and annualized figures are net of the private pricing discount. Expiration of current objects is a different class of change: it ran only where the owner approved it in writing, and it is the part the data-governance council is reviewing for the lake buckets. The shape for those buckets adds one rule, and only after the council answers.

```JSON
{"ID": "expire-tagged-after-30-days", "Status": "Enabled",
 "Filter": {"Tag": {"Key": "Expire-After", "Value": "30d"}},
 "Expiration": {"Days": 30}}
```

A lifecycle rule has no deny, so an exception is made by the filter: the rule touches only objects the owner has tagged `Expire-After=30d`, and release artifacts never carry the tag. `list-objects-v2` returns keys without tags, so it cannot test the filter. Take a sample of keys under the prefix and read each one's tags with `get-object-tagging`: every key a release artifact owns must come back without `Expire-After`. On a versioned bucket, `Expiration` only adds a delete marker; the noncurrent-version rule is what frees the space 30 days later. Rolling back a lifecycle change restores the old rules, not the objects: that is why the first rule shape contains no expiry of current objects.

```BASH
aws s3api list-objects-v2 --bucket <bucket> --prefix <prefix> --max-items 25 --query
'Contents[].Key' --output text \
  | tr '\t' '\n' | while IFS= read -r key; do
    echo "$key: $(aws s3api get-object-tagging --bucket <bucket> --key "$key" --query 'TagSet[?
Key==`Expire-After`].Value' --output text)"
  done
```

CloudWatch Logs retention

A log group with no retention keeps every byte forever at $0.03 per GB-month. The Analysis found that 74 robot-telemetry groups, 2,179 application groups, and 34 debug groups never expire. We set retention by class: 14 days for robot telemetry copies (read only during incidents), 30 days for application logs, and 7 days for debug logs. Audit logs in `sf-log-archive` (CloudTrail, AWS Config, VPC flow logs) are excluded: the security engagement set their retention, and nothing here shortens it.

```bash
set -euo pipefail
for region in <regions-in-use>; do
  for entry in "/robot/telemetry:14" "/aws/lambda:30" "/app:30" "/debug:7"; do
    prefix="${entry%%:*}"; days="${entry##*:}"
    aws logs describe-log-groups --region "$region" --log-group-name-prefix "$prefix" \
      --query 'logGroups[?retentionInDays==`null`].logGroupName' --output text | tr '\t' '\n' |
\
    while read -r group; do
      [[ -z "$group" ]] && continue
      echo "$region $group $days" >> retention-ledger.tsv
      aws logs put-retention-policy --region "$region" --log-group-name "$group" --retention-
in-days "$days"
    done
  done
done
```

The ledger holds every group touched and its new retention; rollback is `aws logs delete-retention-policy --log-group-name <group>`, but logs already past the new retention are gone, which is the point and the one change here that cannot be reversed. The ledger lists every group, so the rollback is a loop over it. We confirmed the retention classes with each owner first.

Debug logs go one step further, to the Infrequent Access log class: $0.25 per GB ingested against $0.50 in the Standard class. A log group's class cannot be changed after it is created, so the change is made in code: the construct creates the group as Infrequent Access, the application points at it, and the old group expires on its 7-day retention.

```typescript
new logs.LogGroup(this, 'DebugLogs', {
  logGroupName: '/debug/<service>',
  logGroupClass: logs.LogGroupClass.INFREQUENT_ACCESS,
  retention: logs.RetentionDays.ONE_WEEK,
});
```

Infrequent Access gives up some features (metric filters and subscription filters among them), so it is for logs people search with Logs Insights, not logs that drive alarms. The default-retention guardrail that keeps new groups from drifting back is in the week-5 report.

AWS Backup tiers

The central vaults in `sf-backup` held 371 TB EBS, 186 TB RDS, 61 TB EFS, all warm, and 45% of EBS and 62% of EFS recovery points are monthly copies older than 90 days. The vault is split by what each type of recovery point can do. Monthly EBS recovery points older than 90 days move to the snapshot archive tier, and EFS backups move to cold storage. RDS and Aurora backups stay

warm, because AWS Backup cannot move them, so the change in `sf-backup:us-east-1:backup-vault` is 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).

Two properties decide which points move. The snapshot archive tier bills an EBS point at its full size, not the incremental size it bills in the warm tier, so only points old enough and rarely enough restored to repay that are worth moving: the monthly copies, not the daily ones. And both tiers have a 90-day minimum, so a rule needs `DeleteAfterDays` at least 90 days past `MoveToColdStorageAfterDays`. Restoring from the archive tier takes one to three days, not minutes.

```yaml
# platform-landing-zone/stacksets/backup-plans.yaml (deployed to each member account by a
StackSet)
Parameters:
  RetentionDays:
    Type: Number
    MinValue: 180            # 90 days warm, then at least 90 in the archive tier
    Description: Keep the retention you have today; this only moves the tier.
  CentralVaultArn:
    Type: String
    Description: The vault in sf-backup that receives the monthly copies.
Resources:
  LocalVault:
    Type: AWS::Backup::BackupVault
    Properties:
      BackupVaultName: monthly-local
  BackupRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - { Effect: Allow, Principal: { Service: backup.amazonaws.com }, Action:
sts:AssumeRole }
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/service-role/AWSBackupServiceRolePolicyForBackup
        - arn:aws:iam::aws:policy/service-role/AWSBackupServiceRolePolicyForRestores
  MonthlyTiered:
    Type: AWS::Backup::BackupPlan
    Properties:
      BackupPlan:
        BackupPlanName: monthly-tier-after-90
        BackupPlanRule:
          - RuleName: monthly
            TargetBackupVault: !Ref LocalVault        # a plan's vault must be in its own
account
            ScheduleExpression: cron(0 5 1 * ? *)
            Lifecycle:
              DeleteAfterDays: 35                     # the local copy is only the staging
point
            CopyActions:
              - DestinationBackupVaultArn: !Ref CentralVaultArn
                Lifecycle:                            # tiering happens on the copy in sf-
```

```
backup
                  MoveToColdStorageAfterDays: 90
                  DeleteAfterDays: !Ref RetentionDays
                  OptInToArchiveForSupportedResources: true   # EBS to the snapshot archive
tier
  MonthlyTieredSelection:
    Type: AWS::Backup::BackupSelection
    Properties:
      BackupPlanId: !Ref MonthlyTiered
      BackupSelection:
        SelectionName: ebs-and-efs
        IamRoleArn: !GetAtt BackupRole.Arn
        Resources:
          - arn:aws:ec2:*:*:volume/*
          - arn:aws:elasticfilesystem:*:*:file-system/*
```

Deploy the StackSet with `--capabilities CAPABILITY_IAM`, and give the central vault an access policy that lets the member accounts copy into it (`backup:CopyIntoBackupVault`). This plan replaces the account's existing monthly rule: remove that rule in the same change, or the account takes two monthly copies.

RDS and Aurora stay in the plan they are in today, with no tiering. Recovery points that already exist change with one call each:

```bash
aws backup update-recovery-point-lifecycle --backup-vault-name <vault> \
  --recovery-point-arn <recovery-point-arn> \
  --lifecycle MoveToColdStorageAfterDays=90,DeleteAfterDays=<retention-
days>,OptInToArchiveForSupportedResources=true
```

Verify with `aws backup describe-recovery-point` (the `StorageClass` field reads `COLD` once a point has moved), and on the bill a day later.

ECR lifecycle policies

The registries held 412 TB of images, and 72% of image bytes are untagged builds older than 30 days. The policy keeps every release-tagged image and the last 30 untagged images per repository, and expires CI-tagged images after 30 days. Two properties of ECR decide its shape. A rule's `tagPatternList` selects an image only when the image matches every pattern in the list, so each CI pattern gets its own rule. And an image is judged by the first rule that matches it, so a first rule that matches release tags (here, tags that start with `v`) and never expires them (`imageCountMoreThan` 9,999) keeps every release image, and every tag on it, out of reach of the rules below. Adjust the release pattern to your tag scheme.

```JSON
{
  "rules": [
    {"rulePriority": 1, "description": "Never expire release images",
     "selection": {"tagStatus": "tagged", "tagPatternList": ["v*"],
                   "countType": "imageCountMoreThan", "countNumber": 9999},
     "action": {"type": "expire"}},
    {"rulePriority": 2, "description": "Expire sha- images after 30 days",
     "selection": {"tagStatus": "tagged", "tagPatternList": ["sha-*"],
                   "countType": "sinceImagePushed", "countUnit": "days", "countNumber": 30},
     "action": {"type": "expire"}},
    {"rulePriority": 3, "description": "Expire pr- images after 30 days",
     "selection": {"tagStatus": "tagged", "tagPatternList": ["pr-*"],
                   "countType": "sinceImagePushed", "countUnit": "days", "countNumber": 30},
     "action": {"type": "expire"}},
    {"rulePriority": 4, "description": "Keep the last 30 untagged images",
     "selection": {"tagStatus": "untagged", "countType": "imageCountMoreThan", "countNumber":
30},
     "action": {"type": "expire"}}
  ]
}
```

Save the policy the repository has today, preview the new one and wait for the preview to finish, and only then apply it. The preview lists what the policy would expire without expiring anything.

```BASH
aws ecr get-lifecycle-policy --repository-name <repository> --query lifecyclePolicyText --
output text > ecr-before.json \
  || echo "no policy (or no access): read the error before going on" >&2
aws ecr start-lifecycle-policy-preview --repository-name <repository> --lifecycle-policy-text
file://ecr-policy.json
aws ecr wait lifecycle-policy-preview-complete --repository-name <repository>
aws ecr get-lifecycle-policy-preview --repository-name <repository> --query 'summary'
aws ecr put-lifecycle-policy --repository-name <repository> --lifecycle-policy-text file://ecr-
policy.json
```

Expired images cannot be restored, only rebuilt from source, which is why release tags sit behind the first rule. Roll back by putting `ecr-before.json` back, or by `aws ecr delete-lifecycle-policy` where there was none. A deploy that pins an image by a `sha-` tag older than 30 days will fail to pull; the preview output, filtered for tags your clusters still reference, is the check. The multi-architecture workflow in the week-4 report tags every build `sha-<git-sha>`, with `-amd64` and `-arm64` on the per-architecture images, so all three tags age out together.

Volume types and WorkSpaces

gp3 costs $0.08 per GB-month against $0.10 for gp2, includes 3,000 IOPS and 125 MiB/s at any size, and (since September 2025) scales to 80,000 IOPS and 2,000 MiB/s, so io1 volumes under that ceiling move too. The conversion is online. Keep the performance the volume has today, in

both IOPS and throughput. A gp2 volume delivers 3 IOPS per GiB, so set IOPS to match when that is above 3,000. It delivers up to 250 MiB/s above 170 GiB (as burst between 170 and 334 GiB), and gp3 defaults to 125, so those volumes, and every io1 volume, are converted with `--throughput 250`, which the Analysis priced at $0.04 per MiB/s-month for the 125 above the baseline. The ledger line is written before the change, so the old values are never lost.

```bash
vol=<volume-id>
read -r size type iops < <(aws ec2 describe-volumes --volume-ids "$vol" --query 'Volumes[0].
[Size,VolumeType,Iops]' --output text)
tput=125
if [[ "$type" == "gp2" ]]; then
  iops=$(( size * 3 > 3000 ? (size * 3 > 16000 ? 16000 : size * 3) : 3000 ))
  (( size > 170 )) && tput=250
else
  tput=250
fi
echo "$vol $type $size $iops ($tput MiB/s)" >> volume-ledger.tsv
aws ec2 modify-volume --volume-id "$vol" --volume-type gp3 --iops "$iops" --throughput "$tput"
```

A volume can be modified once every six hours, so a rollback to the old type waits for that. In code, the guardrail in week 5 makes gp3 the default in all four repositories. For the WorkSpaces that bill by the month and run a few hours a day, switch the running mode to hourly with an automatic stop.

```bash
aws workspaces modify-workspace-properties --workspace-id <workspace-id> \
  --workspace-properties RunningMode=AUTO_STOP,RunningModeAutoStopTimeoutInMinutes=60
```

Verifying a change on the bill

A change is verified the next day against the Cost and Usage Report, by resource ID, before it is trusted. The same query, on a seven-day window either side, is the week-6 measurement, which uses CUR 2.0 net amortized cost with the usage and fee line types that are not a change's own left out. It reads the CUR 2.0 data export in the `cur2` Athena table that the showback in the week-5 report builds on.

```sql
SELECT date(line_item_usage_start_date) AS usage_day,
       sum(CASE line_item_line_item_type
              WHEN 'SavingsPlanCoveredUsage'
                THEN savings_plan_net_savings_plan_effective_cost
              WHEN 'DiscountedUsage'
                THEN reservation_net_effective_cost
              ELSE line_item_net_unblended_cost
            END) AS cost
FROM cur2.strataforge_cur
WHERE line_item_resource_id = '<resource-id>'
  AND line_item_line_item_type NOT IN (
    'SavingsPlanRecurringFee',
    'SavingsPlanNegation',
    'RIFee',
    'Fee',
    'Tax',
    'Credit',
    'Refund'
  )
  AND line_item_usage_start_date >= date_add('day', -7, date '<change-date>')
  AND line_item_usage_start_date <  date_add('day',  8, date '<change-date>')
GROUP BY 1
ORDER BY 1;
```

The window is the seven days before the change and the seven after, and the day of the change is left out when the two sides are averaged. A resource that was deleted has no rows after the change, so read its missing days as zero cost, not as missing data. The exports finalize about 48 hours late, so the next-day check reads the export's latest finalized day, not yesterday. The week-6 report explains the method in full.

What comes next

Week 4 sizes compute, databases, caches, and search to measured use and moves what fits to Graviton (Deliverable 9), and puts commitments in place after finance approves them (Deliverable 10). The week-4 report covers both, including the commitment decision memo for Ravi Menon.

© 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.