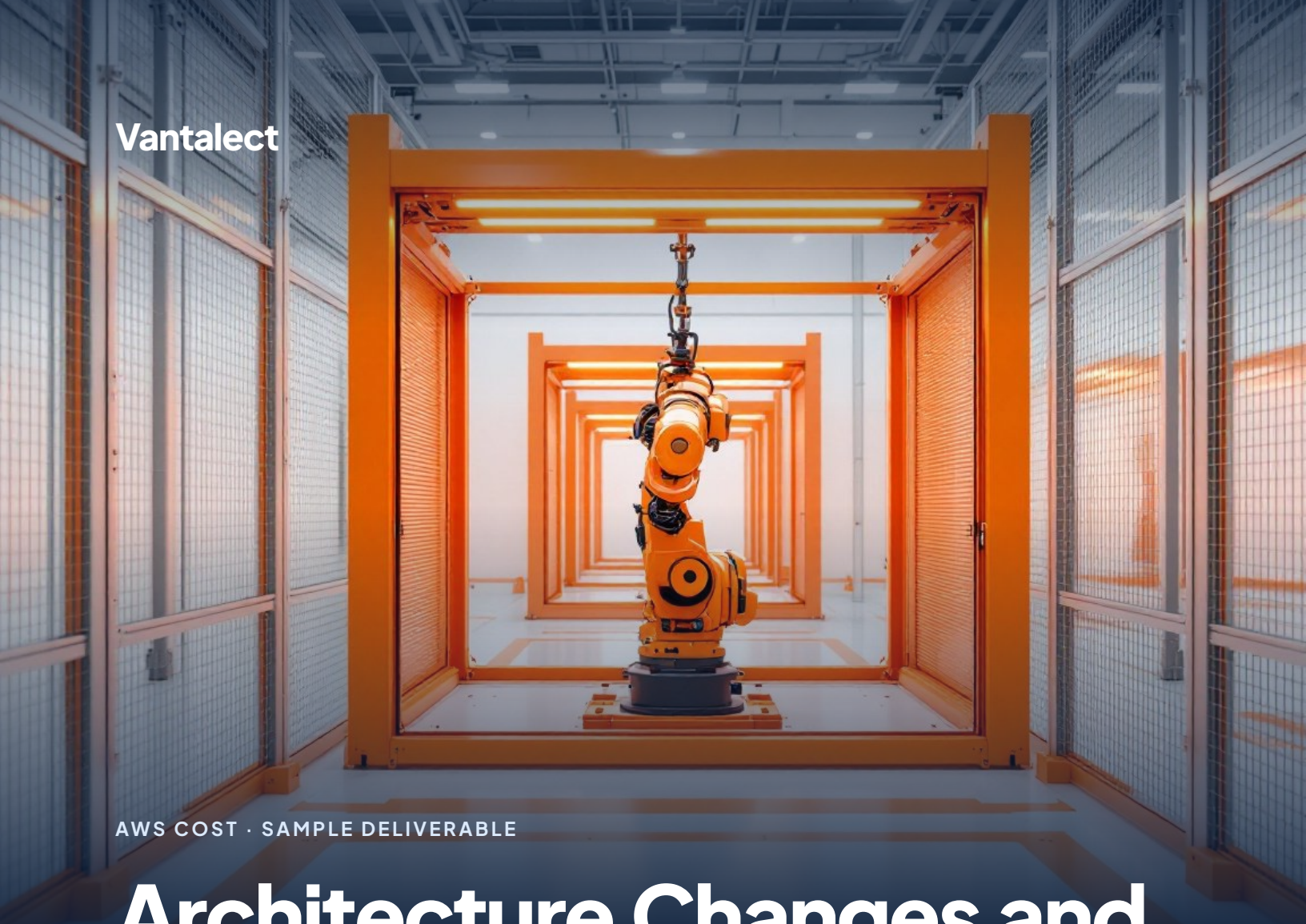


Vantalect

AWS COST · SAMPLE DELIVERABLE

Architecture Changes and Guardrails

Week 5 of 6: Architecture and guardrails · prepared for StrataForge Robotics, Inc.

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

September 25, 2026 · AWS Cost Analysis & Optimization: The Optimization (six weeks)

SAMPLE DELIVERABLE

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

Contents

Summary ... 3

What went in ... 5

The evidence for the architecture changes ... 5

The largest changes ... 8

Safety and rollback ... 10

What was declined or deferred ... 12

Update delivery through CloudFront ... 14

Guardrails ... 15

Guardrail ownership ... 16

Why tags come first ... 16

The exact mechanism ... 16

VPC endpoints in place of NAT ... 17

Kafka consumers read in their own zone ... 19

Robot telemetry to S3 through Firehose ... 20

Per-robot metrics to site metrics ... 22

CloudTrail data events, scoped ... 23

AWS Config recording scope ... 24

Spot for simulation and for CI ... 25

Service calls stay in their zone ... 27

Non-production upgrades: EKS and RDS ... 28

Releasing public IPv4 addresses ... 29

Delivering updates through CloudFront ... 30

Budgets ... 30

Cost Anomaly Detection ... 31

The tag policy ... 32

Showback from the CUR 2.0 data export ... 34

Default retention and volume-type defaults ... 36

What comes next ... 39

Summary

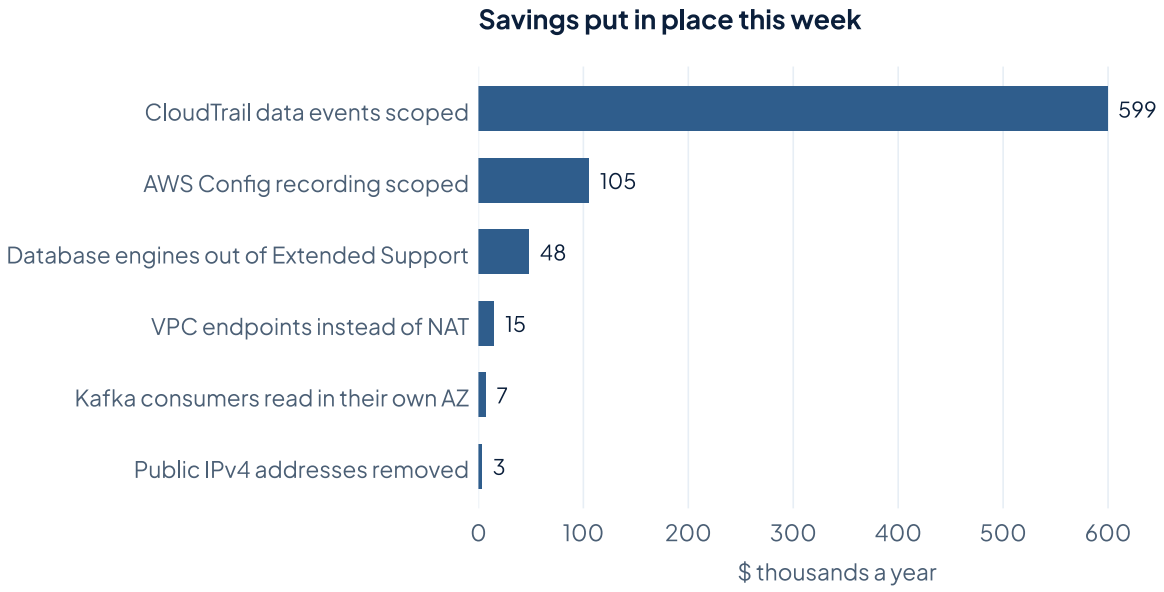
In week 5 StrataForge's engineers put **$778,520 a year** of architecture savings in place: 131 changes to 131 resources across 6 kinds of change, covering the data-event, configuration-recording, non-production upgrade, NAT, and public-address work in Deliverable 11. They also put 6 guardrails (Deliverable 13) in place, so the savings of weeks 3 through 5 stay saved. Vantalect drafted each change and reviewed each pull request and change ticket; every change is owned by a named StrataForge engineer, who applied it. Week 5 changes ran only on September 21, 2026 and September 22, 2026, so each has the full week of data the week-6 measurement needs.

The larger re-architectures are not in that figure. Robot telemetry through Firehose, per-robot metrics, topology-aware routing, and Spot for simulation and CI are merged and rolling out in October, one Region or service at a time. They are worth $2.4 million a year across 276 changes, target October 30, 2026, and none of it is counted yet. Production changes that need a route-table window, a consumer release, or a gap in training wait for those dates (What was declined or deferred). The Kubernetes upgrades are not in the figure either: the readout expected the 1.31 clusters to be upgraded in weeks 3 to 5, and none ran (What was declined or deferred).

The largest single change is also the one that needed the most care. Scoping CloudTrail data events saves $599,428 a year and was agreed with Marcus Oyelaran, Chief Information Security Officer, because it touches the detection coverage the security engagement built. The roadmap's note reads: "agreed with the CISO: the security engagement's detection coverage is unchanged."

Week 5 in numbers. Savings are estimates until the week-6 measurement.

Week 5 at a glance	Result
Changes put in place	131
Resources changed	131
Annualized savings, estimated	$778,520
Monthly savings, estimated	$64,877
One-time cost	$0
Changes by pull request	106
Changes by scripted runbook under a change ticket	25 (across 4 tickets)
Week-5 savings not yet in place (merged and rolling out, or waiting on a window)	$3,637,919 a year
Guardrails in place	6



The ten largest savings put in place in week 5, estimated and annualized.

Decision for the sponsor

Priya Raman: no decision is needed to keep this week's savings. The guardrails need owners to stay in force, and Guardrail ownership names one for each. The one date that changes the number is the change advisory board's route-table window for the production endpoint changes, October 8, 2026, which carries $619,307 a year (What was declined or deferred).

What went in

Every kind of change put in place in week 5, ranked by estimated annualized savings.

Kind of change	Changes	Resources	Monthly	Annualized
CloudTrail data events scoped	1	1	$49,952	$599,428
AWS Config recording scoped	2	2	$8,788	$105,462
Database engines out of Extended Support	38	38	$4,014	$48,167
VPC endpoints instead of NAT	6	6	$1,263	$15,159
Kafka consumers read in their own AZ	2	2	$580	$6,964
Public IPv4 addresses removed	82	82	$278	$3,340
Total	**131**	**131**	**$64,877**	**$778,520**

Data that leaves a network, a Region, or an Availability Zone is billed per gigabyte at every hop it crosses. The NAT and endpoint changes make fewer hops, and the evidence for where is in The evidence for the architecture changes. Only the non-production accounts are in this week's count: production endpoints wait for the change advisory board's route-table window.

The evidence for the architecture changes

Two pieces of the Analysis's architecture review (Deliverable 5) drove most of the week.

Where NAT gateway traffic goes. The two paths cost differently, and the saving depends on which one a workload is on. In a spoke VPC that has its own NAT gateway, a gigabyte of AWS-service traffic pays the NAT gateway's hours and per-gigabyte charge, even though the

destination, an AWS service in the same Region, never needed to leave AWS's network. In the network hub, the NAT gateways sit behind AWS Network Firewall, which waives their hours and gigabytes, so the cost of the same traffic there is the Transit Gateway and the Network Firewall's processing. A gateway endpoint in each spoke VPC takes the S3 and DynamoDB share off both paths: it saves the spoke's NAT charges, and for hub-routed traffic it saves the Transit Gateway and firewall processing, with the hub's NAT gateways staying as they are.

NAT gateway bytes by destination, from VPC flow logs matched to AWS's published address ranges.

NAT destination	GB a month	Share	Where it goes instead
s3	728,854	38%	S3 gateway endpoint (no hourly or per-GB charge)
internet	557,378	29%	Stays on NAT: it really is internet traffic
ecr	225,958	12%	ECR interface endpoints
logs	130,833	7%	CloudWatch Logs interface endpoint
kinesis	85,918	5%	Kinesis Data Streams interface endpoint
other	80,464	4%	Reviewed per destination; stays on NAT for now
dynamodb	75,929	4%	DynamoDB gateway endpoint (no hourly or per-GB charge)
sts	15,678	1%	STS interface endpoint

What the transfer lines cost. The transfer and network lines of the bill, as the August run-rate, are below. An endpoint in a spoke removes traffic from the spoke's NAT gateway; for hub-routed traffic it removes the Transit Gateway and Network Firewall charges, because the hub's NAT charges are already waived.

Network and data-transfer lines on the August bill, from the Cost and Usage Report.

Usage line	Monthly, net
AWS Network Firewall	$73,540
NAT Gateway	$50,992
AWS Transit Gateway	$43,812
Data Transfer: client traffic across AZs	$27,937
Data Transfer: internet	$23,043
Data Transfer: S3 to internet	$18,340
Amazon CloudFront	$18,244
Data Transfer: inter-AZ	$14,434
Data Transfer: replication to us-west-2	$5,395
Data Transfer: replication to eu-west-1	$438

The largest changes

The largest week-5 changes, each with its owner and the channel it went through. Annualized savings are estimates.

ID	Change	Account	Owner	Channel	Applied	Annualized
A-0161	Scope CloudTrail data events with advanced event selectors	sf-management	Elena Vasquez	platform-landing-zone PR #1430	Sep 22	$599,428
A-0172	Stop recording ephemeral network interfaces in AWS Config	sf-cicd	Elena Vasquez	platform-landing-zone PR #1431	Sep 22	$60,934
A-0173	Stop recording ephemeral network interfaces in AWS Config	crucible-sim-prod	Hiro Tanaka	crucible-infra PR #3354	Sep 22	$44,528
A-0725	Upgrade the database engine out of Extended Support	kestrel-staging	Jonah Reyes	change CHG0042528 (scripted runbook)	Sep 22	$9,285
A-0726	Upgrade the database engine out of Extended Support	kestrel-staging	Jonah Reyes	change CHG0042529 (scripted runbook)	Sep 21	$9,285
A-0058	Let consumers read from the replica in their own Availability Zone	fleet-staging	Sam Okafor	fleet-infra PR #9064	Sep 22	$5,357
A-0045	Send AWS-service traffic through VPC endpoints instead of the NAT gateway	fleet-staging	Sam Okafor	fleet-infra PR #9064	Sep 21	$5,258
A-0044	Send AWS-service traffic through VPC endpoints instead of the NAT gateway	fleet-staging	Sam Okafor	fleet-infra PR #9064	Sep 21	$3,022
A-0043	Send AWS-service traffic through VPC endpoints instead of the NAT gateway	fleet-staging	Sam Okafor	fleet-infra PR #9064	Sep 21	$2,397
A-0711	Upgrade the database engine out of Extended Support	fleet-dev	Sam Okafor	fleet-infra PR #9072	Sep 21	$2,321
A-0047	Send AWS-service traffic through VPC endpoints instead of the NAT gateway	fleet-dev	Sam Okafor	fleet-infra PR #9064	Sep 22	$1,810
A-0059	Let consumers read from the replica in their own Availability Zone	fleet-perf	Sam Okafor	fleet-infra PR #9064	Sep 22	$1,607

ID	Change	Account	Owner	Channel	Applied	Annualized
A-0046	Send AWS-service traffic through VPC endpoints instead of the NAT gateway	fleet-dev	Sam Okafor	fleet-infra PR #9064	Sep 22	$1,435
A-0048	Send AWS-service traffic through VPC endpoints instead of the NAT gateway	fleet-dev	Sam Okafor	fleet-infra PR #9064	Sep 22	$1,238
A-0687	Upgrade the database engine out of Extended Support	fleet-dev	Sam Okafor	fleet-infra PR #9066	Sep 22	$1,161
A-0691	Upgrade the database engine out of Extended Support	fleet-dev	Sam Okafor	fleet-infra PR #9066	Sep 21	$1,161
A-0696	Upgrade the database engine out of Extended Support	fleet-dev	Sam Okafor	fleet-infra PR #9067	Sep 22	$1,161
A-0700	Upgrade the database engine out of Extended Support	fleet-dev	Sam Okafor	fleet-infra PR #9069	Sep 21	$1,161
A-0703	Upgrade the database engine out of Extended Support	fleet-dev	Sam Okafor	fleet-infra PR #9070	Sep 22	$1,161
A-0704	Upgrade the database engine out of Extended Support	fleet-dev	Sam Okafor	fleet-infra PR #9071	Sep 21	$1,161
A-0708	Upgrade the database engine out of Extended Support	fleet-dev	Sam Okafor	fleet-infra PR #9071	Sep 21	$1,161
A-0710	Upgrade the database engine out of Extended Support	fleet-dev	Sam Okafor	fleet-infra PR #9072	Sep 22	$1,161
A-0714	Upgrade the database engine out of Extended Support	customer-portal-dev	Sam Okafor	fleet-infra PR #9072	Sep 22	$1,161
A-0715	Upgrade the database engine out of Extended Support	customer-portal-dev	Sam Okafor	fleet-infra PR #9072	Sep 21	$1,161
A-0716	Upgrade the database engine out of Extended Support	customer-portal-dev	Sam Okafor	fleet-infra PR #9072	Sep 21	$1,161

Safety and rollback

Architecture changes are the ones most likely to surprise a service, so the rule this week was stricter: non-production first, one production Region or service at a time, production only inside a window its owner or the change advisory board has set, and every change reversible by reverting the pull request unless this table says otherwise.

What undoes each week-5 change.

Kind of change	How it is undone	Notes
VPC endpoints instead of NAT	Delete the endpoint; traffic returns to the NAT route	Gateway endpoints add a route-table entry, so they take effect without a restart, and production route-table changes wait for the change advisory board. Interface endpoints with private DNS switch resolution within minutes. An endpoint policy that is too narrow fails the request, so it is tested in a spoke first
CloudTrail data-event selectors	Reapply the saved selectors from `selectors-before.json`	Events between the change and a rollback are not recorded retroactively. The CISO accepted this
AWS Config recording scope	Reapply the saved recorder settings from `recorder-before.json`	Items not recorded in the gap are not recovered
Kafka consumers in their own zone	Remove `client.rack` from the consumer, or point the cluster back at the previous configuration revision	Consumers fall back to leader reads at once. `client.rack` ships with the consumers' own release train
Telemetry to S3 through Firehose	Re-add the CloudWatch Logs action to the IoT rule	The two run side by side during the rollout, so there is no data gap
Per-robot metrics to site metrics	Revert the publisher; the old alarms stay in place until the site metrics have shown a full week of data	Per-robot series for the gap are in the lake, not in CloudWatch
FSx for Lustre throughput tier	Raise the throughput per TiB again with `update-file-system`	In place, scheduled for a gap between training runs; a file system can change throughput only so often, so a reversal can wait

Kind of change	How it is undone	Notes
Spot for simulation and CI	Set the compute environment or NodePool to on-demand	Jobs reclaimed by Spot are retried; a failed job is a rerun, never lost data
Service calls stay in their zone	Remove `trafficDistribution` from the Service	Instant
Engine and Kubernetes upgrades	Engine upgrades in non-production only this week; no Kubernetes upgrade ran. EKS allows an in-place upgrade to be rolled back within seven days. RDS blue/green keeps the old environment until you delete it, but after the switchover nothing updates it, so there is no switching back without losing the writes made since	Production follows in the handover roadmap
Public IPv4 addresses	Allocate the same address again with `aws ec2 allocate-address --address <ip>`, which works only if no one else has taken it since	Released only after checking DNS, allow lists, and the NAT and VPN configuration; in production, after the route tables are checked in the owner's window

What was declined or deferred

$3.6 million a year of week-5 items are not in place. The 20 largest are below, each with its owner's reason as they gave it.

The largest week-5 items not put in place, with each owner's reason.

ID	Change	Owner	Status	AnnualizedReason, as the owner gave it
A-0157	Per-robot metrics to site metrics: fleet-ingest-prod:us-east-1:Strat…	Sam Okafor	deferred	$413,688merged and rolling out in October, one Region or service at a time
A-0170	FSx for Lustre throughput tier: fsx:training-data	Hiro Tanaka	deferred	$311,980the Crucible team scheduled it for the gap between training runs
A-0158	Per-robot metrics to site metrics: fleet-ingest-prod:eu-central-1:St…	Sam Okafor	deferred	$216,619merged and rolling out in October, one Region or service at a time
A-0054	Kafka consumers read in their own AZ: telemetry-msk-f535	Sam Okafor	deferred	$120,394client.rack ships with the consumer redeploys on the October release train
A-0159	Per-robot metrics to site metrics: fleet-ingest-prod:ap-northeast-1:…	Sam Okafor	deferred	$108,644merged and rolling out in October, one Region or service at a time
A-0075	Robot telemetry to S3 through Firehose: /sf/telemetry/robot-telemetry/us-…	Sam Okafor	deferred	$99,938merged and rolling out in October, one Region or service at a time
A-0160	Per-robot metrics to site metrics: fleet-ingest-prod:ap-southeast-1:…	Sam Okafor	deferred	$75,784merged and rolling out in October, one Region or service at a time
A-0081	Robot telemetry to S3 through Firehose: /sf/telemetry/robot-telemetry/us-…	Sam Okafor	deferred	$74,188merged and rolling out in October, one Region or service at a time
A-0072	Robot telemetry to S3 through Firehose: /sf/telemetry/robot-telemetry/us-…	Sam Okafor	deferred	$65,605merged and rolling out in October, one Region or service at a time
A-0070	Robot telemetry to S3 through Firehose: /sf/telemetry/robot-telemetry/us-…	Sam Okafor	deferred	$60,598merged and rolling out in October, one Region or service at a time

ID	Change	Owner	Status	AnnualizedReason, as the owner gave it
A-0055	Kafka consumers read in their own AZ: telemetry-msk-190f	Sam Okafor	deferred	$42,720client.rack ships with the consumer redeploys on the October release train
A-0069	Robot telemetry to S3 through Firehose: /sf/telemetry/robot-telemetry/us-…	Sam Okafor	deferred	$37,618merged and rolling out in October, one Region or service at a time
A-0016	VPC endpoints instead of NAT: nat-07f9de82d6c40dd99	Sam Okafor	deferred	$37,488the change advisory board scheduled the route-table change for October
A-0060	Service calls stay in their AZ: fleet-api-prod:us-east-1:inter-az	Sam Okafor	deferred	$34,596merged and rolling out in October, one Region or service at a time
A-0018	VPC endpoints instead of NAT: nat-00c6c674b9435a8fc	Sam Okafor	deferred	$33,557the change advisory board scheduled the route-table change for October
A-0036	VPC endpoints instead of NAT: nat-0bdc0987facba1fd5	Lucas Ferreira	deferred	$30,861the change advisory board scheduled the route-table change for October
A-0035	VPC endpoints instead of NAT: nat-0b77a32fc2b89c875	Lucas Ferreira	deferred	$29,302the change advisory board scheduled the route-table change for October
A-0050	VPC endpoints instead of NAT: nat-0e369b4b952fa4ff1	Hiro Tanaka	deferred	$25,868the change advisory board scheduled the route-table change for October
A-0017	VPC endpoints instead of NAT: nat-0743daaf6ae328ab6	Sam Okafor	deferred	$25,254the change advisory board scheduled the route-table change for October
A-0003	VPC endpoints instead of NAT: nat-075429f1156cc5156	Elena Vasquez	deferred	$23,863the change advisory board scheduled the route-table change for October

Nothing here was refused. Every wait has a date:

What week 5's deferrals wait on, from the handover register. A wait can include lines from other weeks that share its reason.

Waits on	Target	Changes	Annualized
the gap between training runs on October 6	October 6, 2026	1	$311,980
the change advisory board's October route-table window	October 8, 2026	47	$619,307
the consumers' October release train	October 15, 2026	4	$194,184
the October OTA release, which moves the download host	October 20, 2026	4	$63,697
the rollout of the merged change, one Region or service at a time	October 30, 2026	276	$2,433,328

- **The re-architectures are merged and rolling out in October.** Telemetry through Firehose, per-robot metrics, topology-aware routing, and Spot for simulation and CI are pull requests that are already merged. They roll out one Region or service at a time, because each changes how a production service runs, and none of their savings is counted until it is in place and measured. The register targets October 30, 2026.
- **Route tables go through the change advisory board.** Endpoint routes change how production traffic flows, so the board scheduled them for its October route-table window. The pull requests are merged and wait for the date.
- **The Kubernetes upgrades did not run.** The readout planned the 1.31 clusters for weeks 3 to 5. None ran in weeks 3, 4, or 5, and week 5 is the last week of changes. 15 clusters are still to upgrade, worth $61,101 a year in avoided extended-support charges. 11 of them wait for their production upgrade windows, targeted November 20, 2026, ahead of the date AWS upgrades any 1.31 cluster itself, November 26, 2026, with no notice and no rollback; the handover roadmap carries the dates for the rest. That date is the one deadline in this register that cannot move.
- **Consumers, the update host, and the file system wait on their owners.** `client.rack` ships with the consumers' own October release train. The update host moves with the October robot update release. And the FSx throughput tier changes in a gap between training runs, which the Crucible team scheduled.

Update delivery through CloudFront

The August bill carried $18,340 a month of S3 data transfer to the internet beside $18,244 of CloudFront, and robot software updates from `fleet-ota-prod` were the main reason for both: most robots download their updates straight from S3, and 60% of update bytes bypass the

existing distribution. The change is ready: the update service issues CloudFront signed URLs in place of S3 presigned URLs, so every download leaves through the distribution, and no robot firmware changes. It is deferred, not declined, because the robots' update URL moves to the distribution and the host moves with the October update release, targeted October 20, 2026. It is worth $63,697 a year across the 4 update buckets on the transfer price alone: transfer from S3 to CloudFront is free, and CloudFront's tiers apply to the organization's whole CloudFront volume, so the added downloads are priced at the margin of that volume, below the S3 internet rate. We counted nothing for the edge cache, which also saves S3 requests and repeated downloads at a site, because its hit ratio is not known until a release ships through it; the first release will show it on the bill. The pattern and the way to measure it are in Delivering updates through CloudFront.

Guardrails

Deliverable 13 is what keeps the bill down after we leave. Waste returns the way weeds do: a developer launches a test cluster and forgets it, a new log group has no retention, a volume is created as gp2 from a ten-year-old template. A guardrail is a small piece of automation, with an owner, that catches the thing in the first day instead of the first quarterly review.

The guardrails, what each does, and who owns it.

Guardrail	What it does	Owner
AWS Budgets per account and per team	48 account budgets and 24 team budgets on the Team tag, alerting owners at 80% and 100% of actual spend and at 100% of forecast	Tom Becker
Cost Anomaly Detection	monitors per service and per Team tag, daily summaries to each owner, immediate alerts above $5,000 of impact	Tom Becker
Tag policy enforced	Organizations tag policy for Team, Application, Environment, and CostCenter with allowed values, enforced for EC2, EBS, RDS, S3, and Lambda	Elena Vasquez
Showback report	monthly per-team report from the CUR 2.0 data export and Athena, with unit costs (per robot, per simulation hour, per training GPU-hour)	Tom Becker
Default retention	an event rule sets 30–day retention on any new CloudWatch log group that has none	Elena Vasquez
gp3 by default	launch templates and the CDK constructs in all four repositories create gp3 volumes; gp2 and io1 creation alerts the owner	Elena Vasquez

Guardrail ownership

A guardrail with no owner decays. Each has one, named above, and the same people attend the monthly cost review in the handover report. The owners' jobs are narrow: read the alert, decide whether the cause is expected, and either fix it or change the threshold with a reason in the pull request. Tom Becker owns the budgets, anomaly detection, and showback, because they are financial signals; Elena Vasquez owns the tag policy, the retention check, and the volume-type default, because they are platform controls in `platform-landing-zone`.

Why tags come first

75.7% of the bill is attributable to a team today. A budget, an anomaly monitor, and a showback report all need a tag to send an alert to a person, and the rest of the bill goes to nobody. The accounts with the largest untagged spend are below, which is where the tag policy and the retrofit start.

The eight accounts with the most spend that no team tag attributes.

Account	Untagged monthly cost, net
sf-network-hub	$144,692
crucible-training-prod	$108,964
fleet-ingest-prod	$92,083
fleet-data-prod	$88,660
crucible-sim-prod	$64,271
sf-management	$60,071
sf-cicd	$56,606
sf-backup	$49,130

The exact mechanism

Every snippet is complete for the case shown. Placeholders are in angle brackets. Infrastructure is defined in the AWS CDK or CloudFormation in the repository that owns it.

VPC endpoints in place of NAT

Gateway endpoints for S3 and DynamoDB cost nothing, so every spoke VPC that sends that traffic gets them. Interface endpoints cost per Availability Zone per hour and per gigabyte, so they go only where a VPC sends enough traffic to the service to pay for them: ECR image pulls, CloudWatch Logs, Kinesis, and STS were the four where the flow logs showed it. Put the endpoints in the workload VPC, where the traffic starts, so it never reaches the hub. Spoke NAT gateways keep this design: the endpoint takes the traffic off the NAT gateway, and its charges go with it.

The hub is different, and the saving there is smaller than it looks. Network Firewall waives the hub NAT gateways' hours and gigabytes, so the hub's NAT gateways stay and their charges stay waived. What the endpoints save for hub-routed traffic is the Transit Gateway and Network Firewall processing that S3 and DynamoDB traffic paid on its way to the hub. The largest case in the register is 25,156 GB a month of S3 and DynamoDB traffic crossing the Transit Gateway and Network Firewall to this hub, on `nat-075429f1156cc5156`. This is the one place the change carries Medium risk, because the firewall was also the control that kept data from leaving for a bucket outside the organization. The endpoint policy replaces that control: it allows only buckets and tables that belong to StrataForge's organization, by `aws:ResourceOrgID`. The CISO agreed on exactly that condition ("the endpoint policies keep the exfiltration control the firewall gave this traffic; agreed with the CISO").

```typescript
// fleet-infra/lib/endpoints.ts
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import * as iam from 'aws-cdk-lib/aws-iam';

export function addServiceEndpoints(vpc: ec2.Vpc): void {
  vpc.addGatewayEndpoint('S3', { service: ec2.GatewayVpcEndpointAwsService.S3 });
  vpc.addGatewayEndpoint('DynamoDb', { service: ec2.GatewayVpcEndpointAwsService.DYNAMODB });

  const interfaceServices: Record<string, ec2.InterfaceVpcEndpointAwsService> = {
    EcrApi: ec2.InterfaceVpcEndpointAwsService.ECR,
    EcrDocker: ec2.InterfaceVpcEndpointAwsService.ECR_DOCKER,
    Logs: ec2.InterfaceVpcEndpointAwsService.CLOUDWATCH_LOGS,
    Kinesis: ec2.InterfaceVpcEndpointAwsService.KINESIS_STREAMS,
    Sts: ec2.InterfaceVpcEndpointAwsService.STS,
  };
  for (const [id, service] of Object.entries(interfaceServices)) {
    vpc.addInterfaceEndpoint(id, {
      service,
      privateDnsEnabled: true,
      subnets: { subnetType: ec2.SubnetType.PRIVATE_WITH_EGRESS },
    });
  }
}
```

The S3 endpoint policy is where the control lives. In the same file, replace the first line of `addServiceEndpoints` with the endpoint below: its policy allows the organization, and adds the AWS-owned buckets your nodes read by ARN, because they do not belong to your organization and the condition would otherwise block them. Those are the buckets that hold ECR image layers, the Amazon Linux package repositories, and the files that Systems Manager and its patching read, plus the Amazon EMR bucket if you run EMR; each is named in the service's documentation for the Region. List every AWS-owned bucket your workloads use before the policy goes on, because a missing one fails quietly as a download error. The DynamoDB endpoint gets the same organization condition, on `aws:ResourceOrgID` for its tables.

```typescript
// fleet-infra/lib/endpoints.ts: the S3 gateway endpoint, limited to the organization
const s3 = vpc.addGatewayEndpoint('S3', { service: ec2.GatewayVpcEndpointAwsService.S3 });
s3.addToPolicy(new iam.PolicyStatement({
  principals: [new iam.AnyPrincipal()],
  actions: ['s3:*'],
  resources: ['*'],
  conditions: { StringEquals: { 'aws:ResourceOrgID': '<organization-id>' } },
}));
s3.addToPolicy(new iam.PolicyStatement({     // AWS-owned buckets the nodes read
  principals: [new iam.AnyPrincipal()],
  actions: ['s3:GetObject'],
  resources: [
    'arn:aws:s3:::prod-<region>-starport-layer-bucket/*',      // ECR image layers
    'arn:aws:s3:::al2023-repos-<region>-de612dc2/*',           // Amazon Linux 2023 package
repositories
    'arn:aws:s3:::aws-ssm-<region>/*',                         // Systems Manager documents and
scripts
    'arn:aws:s3:::patch-baseline-snapshot-<region>/*',         // Patch Manager baseline
snapshots
  ],
}));
```

Test the policy in a spoke first with the two calls that matter: a bucket in the organization must work, and a bucket outside it must be refused. The second test proves something only if that bucket is readable by the same caller without the endpoint, because any bucket the caller cannot read returns AccessDenied anyway. So use a bucket outside the organization that your test role is allowed to read, and run the same call from a host with no endpoint first, to show that it succeeds there.

```bash
# from an instance in the spoke VPC, after the endpoint policy is on
aws s3api list-objects-v2 --bucket <bucket-in-the-organization> --max-items 1       # expect a
listing
aws s3api list-objects-v2 --bucket <readable-bucket-outside-the-organization> --max-items 1   #
expect AccessDenied

# the control: the same second call from a host with no S3 endpoint must succeed
aws s3api list-objects-v2 --bucket <readable-bucket-outside-the-organization> --max-items 1
```

ECR image layers come from S3, so the S3 gateway endpoint matters for pulls too. Verify the effect over the next day: a spoke NAT gateway's processed bytes should fall by the share the table above predicts, and for the hub, the Transit Gateway and Network Firewall processed bytes for the spoke should fall by the S3 and DynamoDB share.

```bash
aws cloudwatch get-metric-statistics --namespace AWS/NATGateway --metric-name
BytesOutToDestination \
  --dimensions Name=NatGatewayId,Value=<nat-gateway-id> \
  --start-time <yesterday-iso8601> --end-time <today-iso8601> --period 86400 --statistics Sum
```

Kafka consumers read in their own zone

The Analysis measured 40,000 GB a month across AZs, and 60% of cross-AZ bytes are consumer fetches. Consumers read from the partition leader by default, which is in another Availability Zone two times out of three. Since Kafka 2.4, a consumer can read from a follower in its own zone. Two settings turn it on: the brokers allow it, and each consumer says which zone it is in. MSK sets `broker.rack` to the zone ID for you.

```properties
# rack-aware-fetch.properties
replica.selector.class=org.apache.kafka.common.replica.RackAwareReplicaSelector
```

A new configuration holding one property would reset every other broker setting to its default, retention included, so start from the configuration the cluster uses now and add a revision to it. If the cluster runs MSK's default configuration, create a configuration holding just this property instead.

```bash
aws kafka describe-cluster --cluster-arn <cluster-arn> \
  --query 'ClusterInfo.
[CurrentVersion,CurrentBrokerSoftwareInfo.ConfigurationArn,CurrentBrokerSoftwareInfo.Configurat
ionRevision]'
aws kafka describe-configuration-revision --arn <configuration-arn> --revision <current-
revision> \
  --query ServerProperties --output text | base64 --decode > server.properties
grep -q '^replica.selector.class=' server.properties || cat rack-aware-fetch.properties >>
server.properties
aws kafka update-configuration --arn <configuration-arn> --server-properties fileb://
server.properties
aws kafka update-cluster-configuration --cluster-arn <cluster-arn> --current-version <cluster-
version> \
  --configuration-info '{"Arn":"<configuration-arn>","Revision":<new-revision>}'
```

`update-configuration` prints the new revision number. The rollback is the same last command with the previous revision.

On each consumer, set `client.rack` to the zone ID of the node it runs on, read at start-up from the instance metadata service (containers need a metadata hop limit of 2).

```bash
TOKEN=$(curl -sX PUT http://169.254.169.254/latest/api/token -H 'X-aws-ec2-metadata-token-ttl-
seconds: 60')
AZ_ID=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/placement/availability-zone-id)
export KAFKA_CLIENT_RACK="$AZ_ID"      # the consumer reads it into client.rack
```

The check is the `BytesOutPerSec` split by zone, or simply the cross-AZ transfer line on the next day's Cost and Usage Report. Followers lag the leader slightly, so a consumer that needs the very latest record in a latency-sensitive path should keep leader reads; we did not move any of those.

Robot telemetry to S3 through Firehose

The telemetry is analyzed in the lake and the CloudWatch copy is read only during incidents, so writing it to CloudWatch Logs at $0.50 per GB was paying twice: 20,970 GB a month into CloudWatch Logs. The change replaces the IoT rule's CloudWatch Logs action with an Amazon Data Firehose action that writes Parquet to S3, where Athena queries it. Run the two actions side by side for a week and compare record counts before removing the old one.

The saving is net of what the new path costs, and three charges replace the one. Firehose bills each record rounded up to the next 5 KB, so a robot message under 5 KB pays for 5 KB; Parquet conversion is billed per gigabyte converted; and Athena bills the data each query scans, which

Parquet and the date partitions keep small. The estimate priced all three. Check the median record size before the rollout reaches a Region: if most records are small, batch them before they reach the stream.

```typescript
// fleet-infra/lib/telemetry-stack.ts
const stream = new firehose.CfnDeliveryStream(this, 'TelemetryToLake', {
  deliveryStreamType: 'DirectPut',
  extendedS3DestinationConfiguration: {
    bucketArn: lake.bucketArn,
    roleArn: deliveryRole.roleArn,
    prefix: 'telemetry/dt=!{timestamp:yyyy-MM-dd}/hr=!{timestamp:HH}/',
    errorOutputPrefix: 'telemetry-errors/!{firehose:error-output-type}/dt=!{timestamp:yyyy-MM-
dd}/',
    bufferingHints: { intervalInSeconds: 300, sizeInMBs: 128 },
    dataFormatConversionConfiguration: {
      enabled: true,
      inputFormatConfiguration: { deserializer: { openXJsonSerDe: {} } },
      outputFormatConfiguration: { serializer: { parquetSerDe: { compression: 'SNAPPY' } } },
      schemaConfiguration: {
        databaseName: 'telemetry', tableName: 'robot_events', roleArn: deliveryRole.roleArn,
region: this.region,
      },
    },
  },
});

new iot.CfnTopicRule(this, 'TelemetryRule', {
  topicRulePayload: {
    sql: "SELECT * FROM 'sf/+/telemetry/#'",
    awsIotSqlVersion: '2016-03-23',
    ruleDisabled: false,
    actions: [
      { firehose: { deliveryStreamName: stream.ref, roleArn: iotRole.roleArn, separator:
'\n' } },
      // the existing action stays for the parallel week; remove it after one week of matching
counts
      { cloudwatchLogs: { logGroupName: '<existing-telemetry-log-group>', roleArn: '<existing-
iot-logs-role-arn>' } },
    ],
  },
});
```

```sql
-- Parity check: events per hour in the lake, to compare with Logs Insights for the same hours.
SELECT dt, hr, count(*) AS events FROM telemetry.robot_events
WHERE dt = '<yyyy-mm-dd>' GROUP BY dt, hr ORDER BY hr;
```

Per-robot metrics to site metrics

A custom CloudWatch metric costs $0.30 per 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. A dimension of `RobotId` multiplies every metric by the fleet size: the Analysis found 524,000 custom metrics (RobotId x 4 metrics (battery, motor temperature, localization confidence, network latency)). The change publishes 10,197 site and fleet metrics; RobotId moves into the embedded metric format as a property. The ingest service aggregates the robots of a site once a minute and publishes one statistic set per site and metric (minimum, maximum, sum, and sample count) with `PutMetricData`; the per-robot series stay in the lake, where they were already queried. The embedded metric format would put the same data through CloudWatch Logs at $0.50 per GB and accepts only numbers or arrays of up to 100 numbers per metric, not a statistic set, so a direct call is the better fit. The estimate was priced on the log-line route, with log ingestion counted against the saving. The direct call has no log ingestion; it bills API requests instead, and one request carries up to 1,000 metric data items, so a minute of the whole fleet is a small number of requests. The register keeps the estimate until the week-6 bill shows the real cost.

```typescript
// fleet-infra/services/ingest/site-metrics.ts
import { CloudWatchClient, PutMetricDataCommand, type MetricDatum } from '@aws-sdk/client-cloudwatch';

const cloudwatch = new CloudWatchClient({});

export interface SiteMetric {
  siteId: string;
  name: string;
  min: number;
  max: number;
  sum: number;
  count: number;
}

// One call carries up to 1,000 metric data items, so the whole fleet's minute goes out in a few calls,
// not one call per site.
export async function publishSiteMetrics(minute: Date, metrics: SiteMetric[]): Promise<void> {
  const data: MetricDatum[] = metrics.map((m) => ({
    MetricName: m.name,
    Dimensions: [{ Name: 'SiteId', Value: m.siteId }],
    Timestamp: minute,
    Unit: 'None',
    StatisticValues: { SampleCount: m.count, Sum: m.sum, Minimum: m.min, Maximum: m.max },
  }));
  for (let i = 0; i < data.length; i += 1000) {
    await cloudwatch.send(new PutMetricDataCommand({
      Namespace: 'StrataForge/Robot',
      MetricData: data.slice(i, i + 1000),
    }));
```

```
  }
}
```

Alarms and dashboards that used one robot's series move to the site aggregate or to an Athena query on the lake. Audit them first, including alarms built on metric math, which keep their dimensions inside `Metrics` instead of at the top level:

```bash
aws cloudwatch describe-alarms --alarm-types MetricAlarm \
  --query "MetricAlarms[?Dimensions[?Name=='RobotId'] || Metrics[?
MetricStat.Metric.Dimensions[?Name=='RobotId']]].AlarmName"
```

An alarm on a metric that does not exist goes to `INSUFFICIENT_DATA` and watches nothing, so confirm the new series first: the metrics must be listed and must have data points for a full day. Only then create the site-level alarms, wait for them to report `OK`, keep the old alarms for a week, and delete them last.

```bash
aws cloudwatch list-metrics --namespace StrataForge/Robot --metric-name <metric-name> --
dimensions Name=SiteId \
  --query 'length(Metrics)'
aws cloudwatch get-metric-statistics --namespace StrataForge/Robot --metric-name <metric-name>
\
  --dimensions Name=SiteId,Value=<site-id> --start-time <yesterday-iso8601> --end-time <today-
iso8601> \
  --period 3600 --statistics SampleCount Sum
```

CloudTrail data events, scoped

The organization trail recorded every read and write on every bucket and function. Writes are what detection needs; reads of the platform's own telemetry and sensor lakes by its own roles were most of the volume. Advanced event selectors keep all writes and keep reads only on the buckets that hold customer data, firmware, and models. Save the current advanced selectors first, as the bare list that `put-event-selectors` accepts: they are the rollback.

```bash
aws cloudtrail get-event-selectors --trail-name <org-trail-arn> --query AdvancedEventSelectors
> selectors-before.json
```

If that prints `null`, the trail uses basic event selectors, and the rollback is the `EventSelectors` from the same call.

```JSON
[
  {"Name": "Write events on every bucket",
   "FieldSelectors": [
     {"Field": "eventCategory", "Equals": ["Data"]},
     {"Field": "resources.type", "Equals": ["AWS::S3::Object"]},
     {"Field": "readOnly", "Equals": ["false"]}]},
  {"Name": "Read events on buckets holding 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>/", "arn:aws:s3:::<firmware-bucket>/",
"arn:aws:s3:::<models-bucket>/"]}]},
  {"Name": "Lambda invocations, unchanged",
   "FieldSelectors": [
     {"Field": "eventCategory", "Equals": ["Data"]},
     {"Field": "resources.type", "Equals": ["AWS::Lambda::Function"]}]}
]
```

Save the list above as `selectors.json`, then apply it. The rollback is the same command with `selectors-before.json`.

```BASH
aws cloudtrail put-event-selectors --trail-name <org-trail-arn> --advanced-event-selectors
file://selectors.json
aws cloudtrail put-event-selectors --trail-name <org-trail-arn> --advanced-event-selectors
file://selectors-before.json   # rollback
```

The trail is defined in `platform-landing-zone`, so the pull request changes the trail's `AdvancedEventSelectors` there; the commands above are the same change by hand, and the rollback. Management events are untouched, and so is detection of any change to a bucket, a policy, or a key.

AWS Config recording scope

AWS Config charges per configuration item recorded: $0.003 continuous and $0.012 periodic. Network interfaces of build runners and simulation nodes that live under an hour generated most of the items in two accounts (2,600,000 configuration items a month, 71% of them network interfaces of runners and simulation nodes that live under an hour). The recorder excludes that resource type and records instances daily, which keeps the daily state the SOC 2 evidence needs.

AWS Config allows one configuration recorder per Region in an account, and Control Tower already runs one in every governed account and Region (the Hardening engagement enabled it in all 17 Regions). A second recorder, in a StackSet or anywhere else, fails. So the change edits the recorder that is there, under its own name and role. Save its settings first; they are the rollback.

```bash
aws configservice describe-configuration-recorders \
  | jq '.ConfigurationRecorders[0] | {name, roleARN, recordingGroup, recordingMode}' >
recorder-before.json
jq '.recordingGroup = {allSupported: false,
      recordingStrategy: {useOnly: "EXCLUSION_BY_RESOURCE_TYPES"},
      exclusionByResourceTypes: {resourceTypes: ["AWS::EC2::NetworkInterface"]}}
    | .recordingMode = {recordingFrequency: "CONTINUOUS",
      recordingModeOverrides: [{resourceTypes: ["AWS::EC2::Instance"], recordingFrequency:
"DAILY"}]}' \
  recorder-before.json > recorder-after.json
aws configservice put-configuration-recorder --configuration-recorder file://recorder-
after.json
aws configservice put-configuration-recorder --configuration-recorder file://recorder-
before.json   # rollback
```

Control Tower reapplies its own baseline when it updates or re-enrolls an account, which would undo the edit. The documented way to keep a customization is a Lambda function in the management account that runs on Control Tower's lifecycle events (`CreateManagedAccount` and `UpdateManagedAccount`) and runs the same `put-configuration-recorder` call in the account. The function and its event rule live in `platform-landing-zone`. After the next Control Tower update, confirm with `describe-configuration-recorders` that the exclusion held.

Periodic recording costs four times the per-item price of continuous, so it saves money only for a resource type that would otherwise record more than four items a day. Instances qualify in these accounts because of their churn; check a resource type's item count in the Config dashboard before you move it.

Spot for simulation and for CI

Simulation jobs are the textbook Spot workload: many short, restartable jobs that write results to S3. The compute environment asks for Spot and picks the pools with the best price and the lowest chance of interruption, and the job definition retries a job that Spot reclaims and nothing else. Batch retries a failed attempt whenever no rule matches it, so the last rule is a catch-all that ends the job; without it, an application failure would run three times on GPUs.

```typescript
// crucible-infra/lib/sim-batch-stack.ts
const env = new batch.ManagedEc2EcsComputeEnvironment(this, 'SimSpot', {
  vpc,
  spot: true,
```

```
  allocationStrategy: batch.AllocationStrategy.SPOT_PRICE_CAPACITY_OPTIMIZED,
  instanceTypes: [
    ec2.InstanceType.of(ec2.InstanceClass.G5, ec2.InstanceSize.XLARGE4),
    ec2.InstanceType.of(ec2.InstanceClass.G6, ec2.InstanceSize.XLARGE4),
    ec2.InstanceType.of(ec2.InstanceClass.G6E, ec2.InstanceSize.XLARGE4),
  ],
  minvCpus: 0,
  maxvCpus: 4096,
});
const queue = new batch.JobQueue(this, 'SimQueue', {
  priority: 10,
  computeEnvironments: [{ computeEnvironment: env, order: 1 }],
});
new batch.EcsJobDefinition(this, 'SimJob', {
  container: new batch.EcsEc2ContainerDefinition(this, 'SimContainer', {
    image: ecs.ContainerImage.fromEcrRepository(repo, '<tag>'),
    cpu: 8, memory: Size.gibibytes(32), gpu: 1,
  }),
  retryAttempts: 3,
  retryStrategies: [
    batch.RetryStrategy.of(batch.Action.RETRY, batch.Reason.SPOT_INSTANCE_RECLAIMED),
    batch.RetryStrategy.of(batch.Action.EXIT, batch.Reason.CANNOT_PULL_CONTAINER),
    batch.RetryStrategy.of(batch.Action.EXIT, batch.Reason.custom({ onReason: '*' })),   //
everything else: do not retry
  ],
  timeout: Duration.hours(6),
});
```

Offer several instance types: the more pools, the fewer interruptions. A job must be safe to rerun, so each writes its results under a key derived from the episode ID and overwrites. Training stays on On-Demand and Capacity Blocks for ML: a multi-day run that loses a node is a worse trade than the saving.

Ephemeral CI runners use the same idea on Kubernetes. A separate NodePool taints its nodes so only runner pods land there, requests Spot, and is allowed to consolidate within a minute of a runner finishing. Karpenter chooses a node from what the pod asks for and cannot see which architectures a container image supports, so a runner whose image is built for x86 only can be placed on an Arm node and crash. Arm therefore gets its own NodePool with a second taint, and only runner sets whose images are verified as multi-architecture tolerate it.

```yaml
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: ci-spot
spec:
  template:
    spec:
      nodeClassRef: { group: karpenter.k8s.aws, kind: EC2NodeClass, name: default }
      taints:
        - { key: ci, value: "true", effect: NoSchedule }
      requirements:
```

```
        - { key: karpenter.sh/capacity-type, operator: In, values: ["spot"] }
        - { key: kubernetes.io/arch, operator: In, values: ["amd64"] }
        - { key: karpenter.k8s.aws/instance-category, operator: In, values: ["c", "m"] }
  limits:
    cpu: "2000"
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 1m
---
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: ci-spot-arm64
spec:
  template:
    spec:
      nodeClassRef: { group: karpenter.k8s.aws, kind: EC2NodeClass, name: default }
      taints:
        - { key: ci, value: "true", effect: NoSchedule }
        - { key: arch-verified, value: arm64, effect: NoSchedule }
      requirements:
        - { key: karpenter.sh/capacity-type, operator: In, values: ["spot"] }
        - { key: kubernetes.io/arch, operator: In, values: ["arm64"] }
        - { key: karpenter.k8s.aws/instance-category, operator: In, values: ["c", "m"] }
  limits:
    cpu: "1000"
  disruption:
    consolidationPolicy: WhenEmptyOrUnderutilized
    consolidateAfter: 1m
```

The runner scale set's pod template adds a toleration for `ci=true`, and for `arch-verified=arm64` only on a set whose image is verified. It also carries the annotation `karpenter.sh/do-not-disrupt: "true"`, because consolidation would otherwise move a runner in the middle of a job, and the job would fail. The annotation stops only voluntary disruption, so a node is still reclaimed after a Spot interruption notice. Set Karpenter's `settings.interruptionQueue` so it drains a node when AWS posts the two-minute warning. A reclaimed runner fails its job, and the workflow reruns it; CI should be idempotent anyway.

Service calls stay in their zone

Pods call each other across Availability Zones unless told not to, and every crossing is billed in both directions. Kubernetes can prefer a same-zone endpoint, with a spill to others when a zone has none. It is a field on the Service and is available by default from Kubernetes 1.31.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: <service>
spec:
  trafficDistribution: PreferClose
  selector: { app: <service> }
  ports:
    - { port: 80, targetPort: 8080 }
```

Do it for the busiest services first, and make sure each has replicas in every zone: with one replica in one zone, every call from the other zones still crosses. A service mesh with sidecar proxies may ignore the field and route by its own rules, so check the mesh's locality settings for any service that runs behind one. The Analysis measured 310,000 GB a month across AZs (service-mesh telemetry: 70% of cross-AZ bytes are between replicas of the same services). The change is Kubernetes topology-aware routing on the 40 busiest services. It is merged and rolls out at the owner's pace, so none of it is in Every kind of change put in place in week 5, ranked by estimated annualized savings.

Non-production upgrades: EKS and RDS

Extended support is billed whether or not anyone is using it. An EKS cluster on a version past standard support costs $0.60 an hour against $0.10 in standard, about $4,380 a year more per cluster at list price. RDS and Aurora charge Extended Support per vCPU-hour on every vCPU, standbys included: $0.100 in the first two years and $0.200 in the third. Upgrade non-production first, learn what breaks, and carry that to production. The version dates that set the order:

- EKS 1.31 ends extended support on November 26, 2026, after which AWS upgrades the control plane for you with no notice and no rollback. 1.34 enters extended support on December 2, 2026. Target 1.36, the newest version available as of this report, for a cluster you are touching now.
- RDS for PostgreSQL 11 and Aurora PostgreSQL 11 end extended support on March 31, 2027, and Aurora MySQL version 2 enters its third, most expensive year on December 1, 2026.

EKS upgrades one minor version at a time. Upgrade the control plane, then add-ons, then node groups (or let Karpenter replace the nodes), and record each step in the pull request.

```bash
aws eks update-cluster-version --name <cluster> --kubernetes-version <next-minor>
aws eks wait cluster-active --name <cluster>
aws eks update-addon --cluster-name <cluster> --addon-name vpc-cni --addon-version <compatible-
version> --resolve-conflicts PRESERVE
aws eks update-nodegroup-version --cluster-name <cluster> --nodegroup-name <nodegroup>
```

To stop paying extended-support fees automatically at the end of standard support, set the upgrade policy to `STANDARD`. The cluster is then upgraded for you at the end of standard support instead of entering extended support, so use it only on clusters you are content to have upgraded.

```bash
aws eks update-cluster-config --name <cluster> --upgrade-policy supportType=STANDARD
```

For databases, use a blue/green deployment: AWS builds a copy on the new version, keeps it in sync, and switches over in about a minute, leaving the old environment in place until you delete it. After the switchover the old environment is no longer updated, so it is a place to look, not a way back. For RDS for PostgreSQL the source needs `rds.logical_replication` set to 1 in its parameter group, which takes a reboot, before the deployment is created; for Aurora the source is the cluster ARN, not an instance. Check the valid targets first. Do not turn Extended Support off on a database already past end of standard support: that triggers an automatic major version upgrade.

```bash
aws rds describe-db-engine-versions --engine postgres --engine-version <current> \
  --query 'DBEngineVersions[].ValidUpgradeTarget[].EngineVersion'
aws rds create-blue-green-deployment --blue-green-deployment-name <name> \
  --source <db-instance-arn-or-aurora-cluster-arn> --target-engine-version <target-version> \
  --target-db-parameter-group-name <target-parameter-group>
aws rds switchover-blue-green-deployment --blue-green-deployment-identifier <bgd-id> --
switchover-timeout 300
```

Releasing public IPv4 addresses

AWS charges $0.005 per public IPv4 address per hour, in use or idle: about $3.65 a month each. Find every address, decide which ones need to be public, and remove the rest. Addresses that AWS services create (load balancers, NAT gateways) cannot be removed individually.

```bash
aws ec2 describe-network-interfaces \
  --query 'NetworkInterfaces[?Association.PublicIp!=`null`].
[NetworkInterfaceId,Association.PublicIp,Description,Attachment.InstanceId]' \
  --output text
aws ec2 describe-addresses --query 'Addresses[?AssociationId==`null`].[AllocationId,PublicIp]'
--output text
```

For an instance that only needs outbound access, launch it in a private subnet without a public address, and set the subnet not to assign one. Then release the address. A released address is often recoverable with `aws ec2 allocate-address --address <ip>` if no one else has taken it, but

plan on it being gone. Outbound traffic then needs the subnet's NAT gateway or the VPC endpoints, so check the route table first. The non-production addresses went this week; in production the route tables are checked first, in the owning team's change window.

```bash
aws ec2 modify-subnet-attribute --subnet-id <subnet-id> --no-map-public-ip-on-launch
aws ec2 disassociate-address --association-id <association-id>
aws ec2 release-address --allocation-id <allocation-id>
```

Delivering updates through CloudFront

Put the update bucket behind CloudFront with an origin access control, so the bucket is never public, and serve robot downloads through signed URLs. `otaBucket` must be a bucket that this CDK app creates, so CDK can add the bucket policy that lets the distribution read it. If the bucket was imported with `Bucket.fromBucketName`, CDK cannot attach that policy; add an `s3:GetObject` statement for the principal `cloudfront.amazonaws.com` to the bucket's own policy by hand, with the condition `AWS:SourceArn` equal to the distribution's ARN.

```typescript
// fleet-infra/lib/ota-distribution.ts
const distribution = new cloudfront.Distribution(this, 'UpdateDistribution', {
  defaultBehavior: {
    origin: origins.S3BucketOrigin.withOriginAccessControl(otaBucket),
    viewerProtocolPolicy: cloudfront.ViewerProtocolPolicy.HTTPS_ONLY,
    cachePolicy: cloudfront.CachePolicy.CACHING_OPTIMIZED,
    trustedKeyGroups: [keyGroup],
  },
  priceClass: cloudfront.PriceClass.PRICE_CLASS_ALL,
});
```

Measure it on the first release: compare the S3-to-internet and CloudFront lines in the Cost and Usage Report for the release week against the previous release. A high cache hit ratio is the signal, since a site's robots fetch the same files.

Budgets

One budget per account and one per team. Each alerts its owner at 80% and 100% of actual spend, and at 100% of forecast, which warns before the month ends without paying for a forecast that wobbles early in the month. The budget definitions come from a CSV of account IDs and owner addresses, so adding an account is adding a row, and the script skips any budget that already exists, so it can run again.

```bash
#!/usr/bin/env bash
# platform-landing-zone/finops/budgets/create-budgets.sh <accounts.csv>
(id,name,monthly_usd,owner_email)
set -euo pipefail
MGMT=<management-account-id>
while IFS=, read -r id name amount email; do
  [[ "$id" == "id" ]] && continue
  if aws budgets describe-budget --account-id "$MGMT" --budget-name "acct-$name" >/dev/null
2>&1; then
    echo "skip acct-$name: it exists"
    continue
  fi
  cat > budget.json <<JSON
{"BudgetName": "acct-$name", "BudgetType": "COST", "TimeUnit": "MONTHLY",
 "BudgetLimit": {"Amount": "$amount", "Unit": "USD"},
 "CostFilters": {"LinkedAccount": ["$id"]}}
JSON
  cat > notifications.json <<JSON
[{"Notification": {"NotificationType": "ACTUAL", "ComparisonOperator": "GREATER_THAN",
"Threshold": 80, "ThresholdType": "PERCENTAGE"},
  "Subscribers": [{"SubscriptionType": "EMAIL", "Address": "$email"}]},
 {"Notification": {"NotificationType": "ACTUAL", "ComparisonOperator": "GREATER_THAN",
"Threshold": 100, "ThresholdType": "PERCENTAGE"},
  "Subscribers": [{"SubscriptionType": "EMAIL", "Address": "$email"}]},
 {"Notification": {"NotificationType": "FORECASTED", "ComparisonOperator": "GREATER_THAN",
"Threshold": 100, "ThresholdType": "PERCENTAGE"},
  "Subscribers": [{"SubscriptionType": "EMAIL", "Address": "$email"}]}]
JSON
  aws budgets create-budget --account-id "$MGMT" --budget file://budget.json \
    --notifications-with-subscribers file://notifications.json
done < "$1"
```

A team budget is the same file with `"CostFilters": {"TagKeyValue": ["user:Team$<team-tag-value>"]}` in place of the account filter, and `team-$name` as its name. Set each amount from the account's trailing three-month average plus ten percent, and revisit it at the monthly review. The `Team` tag must be activated as a cost allocation tag before a tag-filtered budget sees anything (see the tag policy below).

Cost Anomaly Detection

One monitor for the whole bill by service, and one monitor per `Team` tag value, because a tag monitor follows one value. FinOps gets a daily summary and an immediate alert on the service monitor when one anomaly has an absolute impact of $5,000 or more; each team's owner gets a daily summary of that team's monitor. The teams come from a CSV of team tag values and owner addresses, like the budgets.

```bash
#!/usr/bin/env bash
# platform-landing-zone/finops/anomaly/create-monitors.sh <teams.csv>   (team,owner_email)
set -euo pipefail
daily() {   # daily <monitor-arn> <address> <name>: a daily email summary for anomalies of $100
or more
  aws ce create-anomaly-subscription --anomaly-subscription "$(jq -n --arg m "$1" --arg a "$2"
--arg n "$3" \
    '{SubscriptionName: $n, MonitorArnList: [$m], Subscribers: [{Address: $a, Type: "EMAIL"}],
Frequency: "DAILY",
      ThresholdExpression: {Dimensions: {Key: "ANOMALY_TOTAL_IMPACT_ABSOLUTE", MatchOptions:
["GREATER_THAN_OR_EQUAL"], Values: ["100"]}}}')"
}

SERVICE_MONITOR=$(aws ce create-anomaly-monitor --query MonitorArn --output text \
  --anomaly-monitor '{"MonitorName":"by-
service","MonitorType":"DIMENSIONAL","MonitorDimension":"SERVICE"}')
daily "$SERVICE_MONITOR" <finops-owner-email> daily-finops-summary
aws ce create-anomaly-subscription --anomaly-subscription "$(jq -n --arg m "$SERVICE_MONITOR"
--arg t "<sns-topic-arn>" \
  '{SubscriptionName: "immediate-over-5000", MonitorArnList: [$m], Subscribers: [{Address: $t,
Type: "SNS"}], Frequency: "IMMEDIATE",
    ThresholdExpression: {Dimensions: {Key: "ANOMALY_TOTAL_IMPACT_ABSOLUTE", MatchOptions:
["GREATER_THAN_OR_EQUAL"], Values: ["5000"]}}}')"

while IFS=, read -r team email; do
  [[ "$team" == "team" ]] && continue
  monitor=$(aws ce create-anomaly-monitor --query MonitorArn --output text --anomaly-monitor "$
(jq -n --arg t "$team" \
    '{MonitorName: ("team-" + $t), MonitorType: "CUSTOM", MonitorSpecification: {Tags: {Key:
"Team", MatchOptions: ["EQUALS"], Values: [$t]}}}')")
  daily "$monitor" "$email" "daily-$team"
done < "$1"
```

The tag policy

An Organizations tag policy fixes the key spelling and the allowed values for the four cost tags and enforces them on the services that support it. It does not require a tag to exist: that is a separate service control policy (below). Create the policy at the organization root, then activate the keys as cost allocation tags so they appear on the bill.

```JSON
{
  "tags": {
    "Team": {
      "tag_key": {"@@assign": "Team"},
      "tag_value": {"@@assign": ["fleet-ingest", "fleet-platform", "data-platform", "crucible-
train", "crucible-sim", "crucible-data", "crucible-research", "sap-basis", "mes", "corp-bi",
"cicd", "network", "platform", "secops", "finops", "kestrel"]},
      "enforced_for": {"@@assign": ["ec2:instance", "ec2:volume", "rds:db", "s3:bucket",
"lambda:function"]}
    },
    "Environment": {
      "tag_key": {"@@assign": "Environment"},
      "tag_value": {"@@assign": ["prod", "nonprod", "sandbox", "shared"]},
      "enforced_for": {"@@assign": ["ec2:instance", "ec2:volume", "rds:db", "s3:bucket",
"lambda:function"]}
    },
    "Application": {
      "tag_key": {"@@assign": "Application"},
      "enforced_for": {"@@assign": ["ec2:instance", "ec2:volume", "rds:db", "s3:bucket",
"lambda:function"]}
    },
    "CostCenter": {
      "tag_key": {"@@assign": "CostCenter"},
      "tag_value": {"@@assign": ["CC-*"]},
      "enforced_for": {"@@assign": ["ec2:instance", "ec2:volume", "rds:db", "s3:bucket",
"lambda:function"]}
    }
  }
}
```

```BASH
aws organizations enable-policy-type --root-id <root-id> --policy-type TAG_POLICY
aws organizations create-policy --name cost-allocation-tags --type TAG_POLICY \
  --description "Team, Environment, Application, CostCenter" --content file://tag-policy.json
aws organizations attach-policy --policy-id <policy-id> --target-id <root-id>
aws ce update-cost-allocation-tags-status --cost-allocation-tags-status \
  TagKey=Team,Status=Active TagKey=Environment,Status=Active TagKey=Application,Status=Active
TagKey=CostCenter,Status=Active
aws resourcegroupstaggingapi start-report-creation --s3-bucket <report-bucket>
```

The compliance report from the last command lists every resource that violates the policy: that is the retrofit list. To require the `Team` tag when a resource is created, add a service control policy that denies the create call without it. Attach it to the Sandbox organizational unit first, then development, then everything, and exempt the roles that launch resources for others (Karpenter's controller and the CloudFormation execution role that CDK deploys through) by naming them in the condition, after confirming those tag what they launch. Auto Scaling needs no exemption: service control policies do not apply to service-linked roles.

```JSON
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "RequireTeamTagOnCreate",
    "Effect": "Deny",
    "Action": ["ec2:RunInstances"],
    "Resource": ["arn:aws:ec2:*:*:instance/*", "arn:aws:ec2:*:*:volume/*"],
    "Condition": {
      "Null": {"aws:RequestTag/Team": "true"},
      "ArnNotLike": {"aws:PrincipalArn": ["arn:aws:iam::*:role/<karpenter-controller-role>",
"arn:aws:iam::*:role/cdk-*-cfn-exec-role-*"]}
    }
  }]
}
```

Showback from the CUR 2.0 data export

The showback is one Athena query over the Cost and Usage Report, run monthly, that gives each team its bill, its untagged share, and its unit costs. The source is a CUR 2.0 data export written as Parquet to an S3 bucket in the management account (the Data Exports API is in `us-east-1`).

```JSON
{
  "Name": "strataforge-cur2",
  "DataQuery": {
    "QueryStatement": "SELECT * FROM COST_AND_USAGE_REPORT",
    "TableConfigurations": {"COST_AND_USAGE_REPORT": {
      "TIME_GRANULARITY": "DAILY", "INCLUDE_RESOURCES": "TRUE",
      "INCLUDE_MANUAL_DISCOUNT_COMPATIBILITY": "FALSE", "INCLUDE_SPLIT_COST_ALLOCATION_DATA":
"FALSE"}}},
  "DestinationConfigurations": {"S3Destination": {
    "S3Bucket": "<cur-bucket>", "S3Prefix": "cur2", "S3Region": "us-east-1",
    "S3OutputConfigurations": {"OutputType": "CUSTOM", "Format": "PARQUET", "Compression":
"PARQUET", "Overwrite": "OVERWRITE_REPORT"}}},
  "RefreshCadence": {"Frequency": "SYNCHRONOUS"}
}
```

The file holds the export itself, with no outer wrapper, because `--export` takes that structure directly.

```BASH
aws bcm-data-exports create-export --export file://export.json --region us-east-1
```

Define the table over the export with partition projection, so there is no crawler and no `MSCK REPAIR`.

```sql
CREATE EXTERNAL TABLE cur2.strataforge_cur (
  line_item_usage_account_id string,
  line_item_usage_start_date timestamp,
  line_item_resource_id string,
  line_item_line_item_type string,
  line_item_net_unblended_cost double,
  savings_plan_net_savings_plan_effective_cost double,
  reservation_net_effective_cost double,
  product_servicecode string,
  resource_tags map<string,string>
)
PARTITIONED BY (billing_period string)
STORED AS PARQUET
LOCATION 's3://<cur-bucket>/cur2/strataforge-cur2/data/'
TBLPROPERTIES (
  'projection.enabled' = 'true',
  'projection.billing_period.type' = 'date',
  'projection.billing_period.format' = 'yyyy-MM',
  'projection.billing_period.range' = '2025-08,NOW',
  'projection.billing_period.interval' = '1',
  'projection.billing_period.interval.unit' = 'MONTHS',
  'storage.location.template' = 's3://<cur-bucket>/cur2/strataforge-cur2/data/BILLING_PERIOD=$
{billing_period}'
);
```

A view adds the cost every showback query uses: net amortized cost, the same expression as the week-6 measurement. Unblended cost would show usage that a reservation covers at $0 and usage that a Savings Plan covers at the On-Demand rate, so a team that runs on commitments would look cheap and a team that runs On-Demand would look expensive.

```sql
CREATE VIEW cur2.strataforge_cost AS
SELECT *,
       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 net_cost
FROM cur2.strataforge_cur
WHERE line_item_line_item_type IN ('Usage', 'DiscountedUsage', 'SavingsPlanCoveredUsage');
```

Then the monthly team showback. The `Team` tag appears in the report as `user_team`.

```sql
SELECT coalesce(element_at(resource_tags, 'user_team'), '(untagged)') AS team,
       round(sum(net_cost), 0) AS cost,
       round(100.0 * sum(net_cost) / sum(sum(net_cost)) OVER (), 1) AS share_pct
FROM cur2.strataforge_cost
WHERE billing_period = '<yyyy-mm>'
GROUP BY 1
ORDER BY cost DESC;
```

Unit costs divide a team's cost by what it produces. The Fleet Cloud cost per connected robot is the cost of the Fleet Cloud teams divided by the robots connected that month, taken from the fleet database (the count was 212,000 at baseline).

```sql
SELECT round(sum(net_cost) / <connected-robots>, 2) AS cost_per_connected_robot
FROM cur2.strataforge_cost
WHERE billing_period = '<yyyy-mm>'
  AND element_at(resource_tags, 'user_team') IN ('fleet-ingest', 'fleet-platform', 'data-
platform');
```

The showback is net of the private pricing discount, with Savings Plans and reservations spread over the usage they cover. It leaves out the fees for commitment hours that nothing used, taxes, credits, and refunds, so the teams' costs sum to less than the invoice; FinOps reports those lines separately, and the report says so.

Default retention and volume-type defaults

A new log group has no retention, and a new volume can still be created as gp2. Two small checks catch both on the day. The first is an EventBridge rule on `CreateLogGroup` that calls a function to set 30 days when none is set.

Audit-evidence log groups keep whatever retention their owners set, so the function skips any group under the prefixes listed in `SKIP_PREFIXES`.

```typescript
// platform-landing-zone/lib/log-retention-guardrail.ts
const fn = new lambda.Function(this, 'DefaultRetention', {
  runtime: lambda.Runtime.PYTHON_3_13,
  architecture: lambda.Architecture.ARM_64,
  handler: 'index.handler',
  environment: { SKIP_PREFIXES: '<audit-log-group-prefix>,<second-audit-log-group-prefix>' },
  code: lambda.Code.fromInline(`
import os
import boto3
logs = boto3.client("logs")
SKIP = tuple(p for p in os.environ.get("SKIP_PREFIXES", "").split(",") if p)
def handler(event, _context):
    name = event["detail"]["requestParameters"]["logGroupName"]
    if SKIP and name.startswith(SKIP):
        return
    group = logs.describe_log_groups(logGroupNamePrefix=name, limit=1)["logGroups"]
    if group and group[0]["logGroupName"] == name and "retentionInDays" not in group[0]:
        logs.put_retention_policy(logGroupName=name, retentionInDays=30)
`),
});
fn.addToRolePolicy(new iam.PolicyStatement({ actions: ['logs:DescribeLogGroups',
'logs:PutRetentionPolicy'], resources: ['*'] }));
new events.Rule(this, 'OnCreateLogGroup', {
  eventPattern: { source: ['aws.logs'], detailType: ['AWS API Call via CloudTrail'], detail:
{ eventName: ['CreateLogGroup'] } },
  targets: [new targets.LambdaFunction(fn)],
});
```

The second makes gp3 the default in code and alerts when something else is created. Set the CDK feature flag `"@aws-cdk/aws-ec2:ebsDefaultGp3Volume": true` in `cdk.json`, so a volume or block device that names no type is gp3. A CDK aspect then fails synthesis for any gp2 or io1 volume in the four repositories, whether it is a standalone volume, a block device on an instance, or a block device in a launch template. A volume with no type set counts as gp2, because that is the default when the flag is off. An EventBridge rule tells the owner about volumes created outside code, from both `CreateVolume` and `RunInstances`, which creates volumes without a `CreateVolume` call.

```typescript
// platform-landing-zone/lib/gp3-aspect.ts
import { Annotations, IAspect, Tokenization } from 'aws-cdk-lib';
import * as ec2 from 'aws-cdk-lib/aws-ec2';
import { IConstruct } from 'constructs';

const OLD_TYPES = ['gp2', 'io1'];

// An unset type is gp2. A value that is still a token cannot be read at synthesis; the
EventBridge rule below catches it.
function isOldType(type: string | undefined): boolean {
  if (type !== undefined && Tokenization.isResolvable(type)) return false;
  return type === undefined || OLD_TYPES.includes(type);
```

```
}

function hasOldDevice(mappings: unknown): boolean {
  if (!Array.isArray(mappings)) return false;
  return mappings.some((m) => {
    if (Tokenization.isResolvable(m)) return false;
    const ebs = (m as { ebs?: unknown }).ebs;
    if (ebs === undefined || Tokenization.isResolvable(ebs)) return false;
    return isOldType((ebs as { volumeType?: string }).volumeType);
  });
}

export class RequireGp3 implements IAspect {
  visit(node: IConstruct): void {
    let bad = false;
    if (node instanceof ec2.CfnVolume) {
      bad = isOldType(node.volumeType);
    } else if (node instanceof ec2.CfnInstance) {
      bad = hasOldDevice(node.blockDeviceMappings);
    } else if (node instanceof ec2.CfnLaunchTemplate) {
      const data = node.launchTemplateData;
      bad = data !== undefined && !Tokenization.isResolvable(data) &&
hasOldDevice(data.blockDeviceMappings);
    }
    if (bad) {
      Annotations.of(node).addError('Use gp3 (or io2 where IOPS justify it). See the cost
playbook.');
    }
  }
}
// in each app: Aspects.of(app).add(new RequireGp3());
```

TYPESCRIPT

```
new events.Rule(this, 'OnOldVolumeType', {
  eventPattern: {
    source: ['aws.ec2'], detailType: ['AWS API Call via CloudTrail'],
    detail: {
      $or: [
        { eventName: ['CreateVolume'], requestParameters: { volumeType: ['gp2', 'io1',
{ exists: false }] } },   // no type means gp2
        { eventName: ['RunInstances'], requestParameters: { blockDeviceMapping: { items: { ebs:
{ volumeType: ['gp2', 'io1'] } } } } },
      ],
    },
  },
  targets: [new targets.SnsTopic(ownerTopic)],
});
```

What comes next

Week 6 measures every change from weeks 3, 4, and 5 on the Cost and Usage Report, using net amortized cost and only the changes with a full week of data behind them, annualizes it, and decides the second invoice (Deliverable 14), then hands the playbook over (Deliverable 15). The re-architectures that are merged and rolling out are not in the measurement, because they are not in place yet. The deferred items above, and the rest of the roadmap, go into the handover plan with an owner and a target for each.

© 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.