

Cloud Cost & Carbon Savings

Why it matters to you as an AWS user, and a tool I built to find it automatically

Agenda

- Part 1 — Why your account is probably wasting money right now
- Part 2 — The most impactful ways to cut cloud cost
- Part 3 — GreenOps: a free CLI that finds this automatically
- How it works, what it can (and can't) access
- Live example: CLI output + generated PDF report

Part 1 — The problem

You can't see your own waste, and that's normal

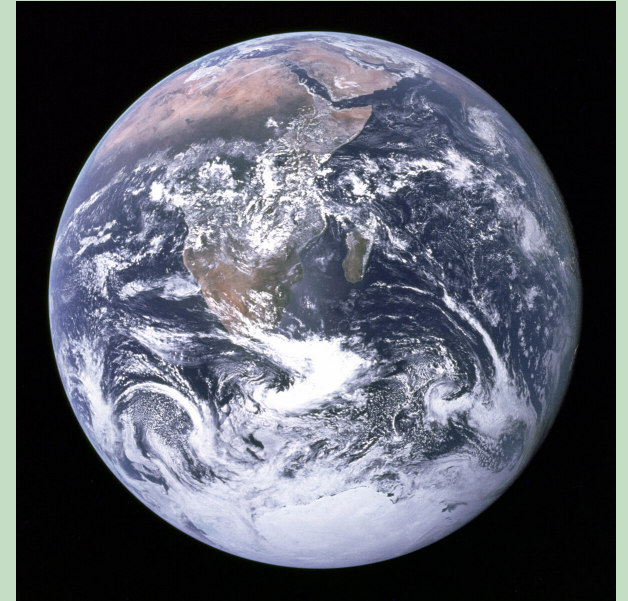
- You didn't do anything wrong — every AWS account accumulates this over time, just by being used
- An instance sized for a launch that ended months ago. A cache you resized for a spike that never came back. A bucket from a project someone else owns now.
- None of it looks wrong when you're looking at one resource. Nobody reviews a whole account unless something breaks
- Industry surveys put this at **27–32% of cloud spend**, account after account — this isn't specific to any one company

It keeps billing whether you think about it or not

- You can't go back and "un-spend" last month's compute hours — only stop the ones still running
- A \$500/month orphaned volume doesn't wait for your next sprint planning — it bills \$500 this month, next month, and the one after
- The fix is almost never the hard part — *finding* it is

Cost and carbon come from the same root cause

- An idle EC2 instance, an oversized Fargate task, an orphaned volume — reserved capacity doing nothing, costing money *and* burning real electricity for no reason
- You don't need to care about carbon to want this fixed — but if you do, fixing the dollar cost fixes the carbon cost too, same finding, same fix
- No framework required — this is stuff you can decide and act on yourself, resource by resource



Why nobody just goes and checks

- Checking one resource is easy. Checking all of them — across EC2, S3, RDS, ECS, Lambda, EBS, ElastiCache, and more — by hand, resource by resource, is hours of tedious, easy-to-get-wrong work
- Whoever provisioned something has usually moved on to the next ticket by the time it's worth revisiting
- By the time a spreadsheet audit finishes, the account has already changed — new resources, new owners, stale numbers

Part 2 — Where the money actually goes

The most impactful categories, ranked by how often they show up

#	Category	Typical finding
1	S3 without lifecycle policies	Data stuck in Standard tier forever; old versions pile up
2	Idle / oversized ElastiCache clusters	Redis/Memcached nodes reserved for peak, running at 2–4% CPU
3	Idle / oversized compute (EC2, Fargate, RDS)	Paid for reserved capacity that's rarely, if ever, reached
4	Orphaned EBS volumes & snapshots	Steady, predictable cost from storage nobody is using

#1 — S3 without lifecycle policies

- Standard → Standard-IA (after 30 days) is roughly half the storage price
- Glacier tiers are a fraction of that again
- Most buckets were set up once and never revisited — the rules either never existed or only covered one use case
- Consistently the single largest savings category across every real account scanned so far

#2 — Idle and oversized ElastiCache clusters

- Redis/Memcached billing is per-node-hour, regardless of how much of the cache is actually being hit
- A cluster sized for a traffic spike months ago keeps billing at that size forever, even if real usage settled at a fraction of it
- On real account scans: clusters sitting at **2–4% average CPU over 14 days** are common, and often the second-largest savings opportunity after S3

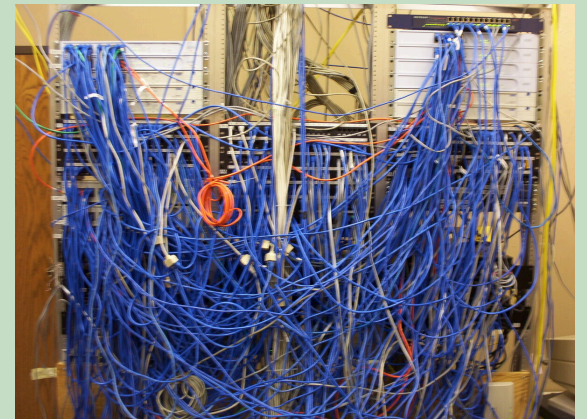
#3 — Idle and oversized compute

- EC2 / Fargate / RDS bill per-reservation, not per-utilization — same price whether it's used at 5% or 95%
- Nothing *fails* when oversized — it just quietly runs at 2–3x the cost it needs
- Rule of thumb: a right-sized instance reaches **40–70% peak utilization**
- Moving an over-provisioned instance to Graviton just copies the bloat — right-size *first*, then move to Graviton



#4 — Orphaned EBS volumes & snapshots

- Detached volumes and stale snapshots are doing almost nothing useful, but are still billed every day
- Deletion is lower-risk and faster than compute right-sizing — the cheapest "low-hanging fruit" review



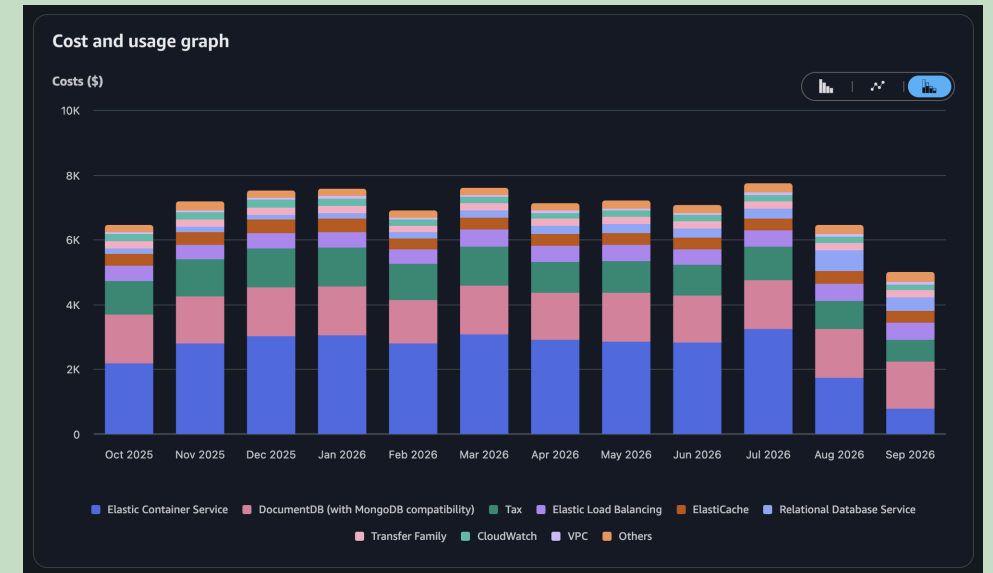
What this looks like on a real account: first scan

Same dev account, two real scans, 8 weeks apart:

- **Scan 1:** account billing **\$7,726.20/month**
- Scanner found **53 findings, \$1,176.83/month** (\$14,121.96/yr) in identifiable waste
- Largest line items: ECS (\$3,235.93/mo) and DocumentDB (\$1,506.78/mo)
- Team reviewed and acted on a subset of the findings

After: 8 weeks later

- **Scan 2:** fewer findings, lower monthly run-rate
- This account's own AWS Cost Explorer graph: billing climbed toward **~\$7,800/month**, then dropped to **~\$5,700** and **~\$4,600/month** in the two months after the cleanup
- This is the model: scan → review with the resource owners → fix what makes sense → re-scan to measure the delta



Part 3 — GreenOps

What is GreenOps

- A free, read-only CLI (`npx greenops-scan`) that scans an AWS account and reports cost + carbon savings opportunities
- Generates a terminal summary, a JSON report, and a branded PDF



How it works

1. You run `npx greenops-scan` (or the Docker image) against an AWS profile
2. The CLI calls **read-only AWS APIs** (describe/list/get-metric-statistics style calls) — in parallel, across up to 14 service modules
3. Findings are scored by estimated **dollar impact** and **carbon impact**
4. A report is generated **on your machine**: terminal output, JSON, and a PDF

What a scan looks like

```
$ npx greenops-scan --region us-east-1
```

```
Starting parallel scan in us-east-1...
```

✓ EC2 Instances & EBS Volumes	(11 findings)	[11/13]
✓ S3 Buckets & Storage	(2 findings)	[5/13]
✓ Lambda Functions	(3 findings)	[7/13]
✓ EBS Snapshots & AMIs	(1 finding)	[8/13]

```
...
```

```
Scan complete: 14 modules scanned, 17 findings
```

Safe by design — what it can't do

- Uses **read-only AWS APIs only** — it cannot modify, stop, resize, or delete any resource
- No agent is installed in the AWS account
- Safe to run against dev, staging, or even production accounts for a first, low-risk look

Safe by design — what happens to your data

- Runs **locally on your machine** by default; your AWS credentials never leave your machine
- The report stays local unless *you* choose to share it
- **Nothing is uploaded anywhere**

Don't just trust it — enforce it with IAM

- Create a dedicated, scan-only AWS CLI profile in under 5 minutes:

```
aws iam create-user --user-name greenops-scan-readonly
aws iam attach-user-policy --user-name greenops-scan-readonly \
  --policy-arn arn:aws:iam::aws:policy/ReadOnlyAccess
aws iam create-access-key --user-name greenops-scan-readonly
aws configure --profile greenops-readonly
```

- `ReadOnlyAccess` is AWS-managed — it grants `Describe*` / `Get*` / `List*` and nothing that creates, modifies, or deletes a resource
- The guarantee is enforced by IAM, not by trusting the scanner's code
- Full walkthrough: greenops.cloud/blog/blog-aws-profile-readonly

What it scans

Module	What it detects
EC2 / EBS	Idle instances, old-gen → Graviton upgrades, unattached volumes
S3	Missing lifecycle policies, version accumulation
ECS / EKS	Over-provisioned Fargate, outdated K8s versions, non-Graviton nodes
Lambda	Zombie functions, legacy runtimes, over-provisioned memory
RDS / ElastiCache	Idle databases/caches, Graviton upgrades, Extended Support surcharges
CloudFront / OpenSearch	Unused distributions, outdated TLS, legacy domain versions
VPC Endpoints	NAT Gateway data tax, cross-AZ transfer spend
EBS Snapshots	Old snapshots, unused AMIs

Example: live scan output

Scan Results – AWS / us-east-1
Account: 000000000000 | 2026-10-09

- severity: HIGH
module: EC2
resource: i-50fce4e5b61cd6a07
issue: Old-gen t3.xlarge – upgrade to t4g.xlarge (Graviton ARM)
costSavings: \$48.59/mo
co2Savings: 0.50 kg/mo
recommendation: Switch to t4g.xlarge for ~40% cost reduction
and ~60% lower carbon
- severity: MEDIUM
module: EBS-SNAPSHOTS
resource: 565 total snapshots
issue: 565 snapshots totaling 8475 GB (\$424/mo)
costSavings: \$127.13/mo
recommendation: Audit snapshot usage – consider Amazon Data
Lifecycle Manager for automated policies

Example: the summary block

Summary

totalFindings: 8
monthlyCostSavings: \$351.49
annualCostSavings: \$4,217.88
monthlyCo2Reduction: 3.94 kg
annualCo2Reduction: 47.29 kg

What This Carbon Reduction Means

- > Equivalent to driving a diesel car: 225 km/year
- > Trees needed to offset annually: 2 trees
- > Smartphone charges equivalent: 5,910 charges

(This exact example is from a demo AWS environment built specifically for this talk — not a real customer account.)

Example: generated PDF report (page 1)

GreenOps

Carbon & Cost Reduction Report

AWS | Region: us-east-1 | Account: default (000000000000)

Generated: 2026-10-09T22:17:01.291Z

N/A

Total Spend / mo

\$351

Est. Savings / mo

4 kg

Est. CO2 Reduction / mo

8

Findings

Example: generated PDF report (executive summary)

Executive Summary

This report presents the results of a GreenOps infrastructure scan conducted on 2026-10-09T22:17:01.291Z. The scan analyzed your AWS account (default (000000000000)) in the us-east-1 region to identify opportunities for cost optimization and carbon emission reduction.

Key Metrics

Total findings identified	8
Estimated monthly cost savings	\$351.49
Estimated annual cost savings	\$4217.88
Monthly carbon reduction potential	3.94 kg CO2
Annual carbon reduction potential	47.29 kg CO2

Try it yourself

```
npx greenops-scan
```

- No signup, no agent, no AWS console clicking required
- Takes about a minute end-to-end on a typical account
- Read-only — safe to point at a real account today
- More deep dives, examples, and write-ups: greenops.cloud/blog

Questions?

