

From Code to Carbon: GreenOps for OpenSearch: Building Cost-Efficient and Sustainable Tech

- Amrutha KH, SDE II



Key Terms & Concepts

GreenOps is our approach and tooling for greener operations: catching wasteful compute patterns in code on every pull request, before they reach production.

Green = less wasted work, less energy, lower emissions. **Ops** = where teams already work.

OpenSearch

The engine.

The search engine your app uses to find and analyze data.

GreenOps

The early warning on code.

The coach + gate that checks code before merge.

Code → Carbon

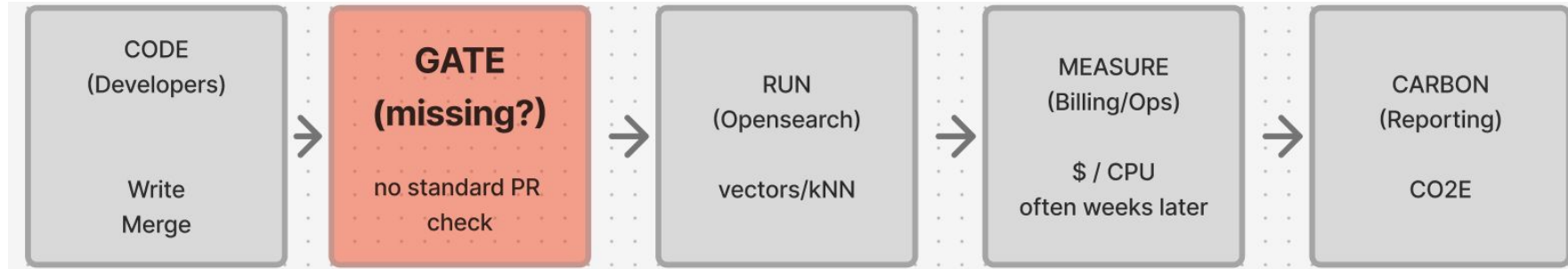
The story we're telling.

Connecting developer decisions to sustainability outcomes through PR checks and post-deploy truth.

"The software you merge causes work on servers; work uses electricity; electricity has a carbon footprint depending on the grid."



The Problem: Where the Chain Breaks



1. Design Cost

OpenSearch + vectors are costly by design

Embeddings, kNN, large indexes → RAM, CPU, ingest. The infrastructure track exists because this is hard.

2. Code Multiplication

Application code can multiply that cost

Serial API calls, huge retrieves, full reindexes. Passes tests, still burns the cluster.

3. Disconnected Impact

Carbon & cost are disconnected from the PR

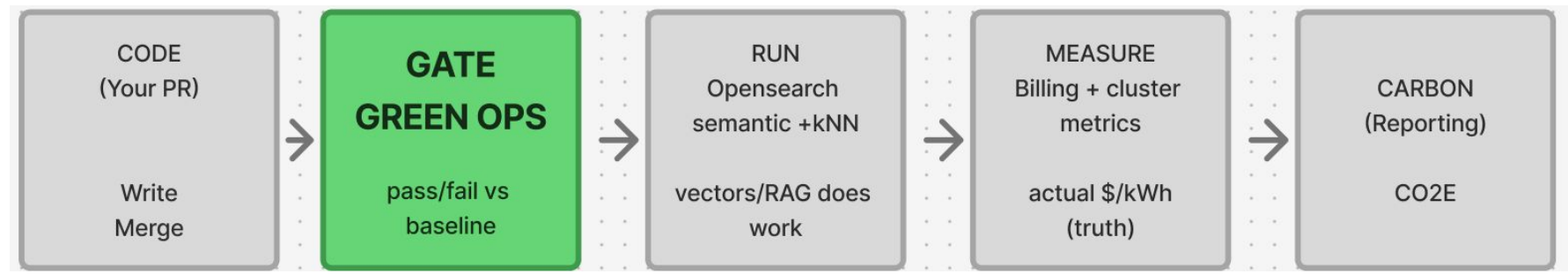
Devs see latency; finance sees kWh later. Rarely linked to the merge that caused it.

"We ship code, → OpenSearch does the work, → the planet and budget pay—but we don't gate 'getting worse' at step 2. "



GreenOps fills the gate - before OpenSearch and RAG pay the price

From merge decision to cluster truth, GreenOps guards the gate; OpenSearch and billing prove the impact.



What GreenOps does:

- **Reproducible:** Runs on every PR - same rules locally and in CI.
- **Static Analysis:** Scores efficiency risk based on code patterns (not real-time power).
- **Regression Fails:** Compared to a committed baseline on **main**.

PR (GreenOps) Production (OpenSearch)

"Should we merge?" "What actually happened?"

Indicative carbon impact score CPU, RAM, ingest, query latency, bill

Catches bad habits early Proves if fixes worked

"GreenOps: fail the PR, not the planet honestly."



How GreenOps runs in practice

Process: init once · Gate every PR · Measure after ship on OpenSearch:
init → **PR scan** → **ship** → measure

1. INIT (once)

```
npx greenops init  
greenops.config.json  
.greenops/baseline.json
```

Commit both → Baseline for future PRs

2. PR (every change)

```
greenops scan --diff → pass / fail
```

Only changed .ts/.js

AST → score 0–100

Gate vs baseline → Pass / Fail

INEFFICIENT EXAMPLE:

87 - FAIL (6 findings)
Serial index() in loop, KNN > 200

3. SHIP & MEASURE

App → OpenSearch (vectors, kNN, RAG)

Dashboard: `/metrics`

- Measure: \$, CPU, latency
- Carbon: kWh x grid
- Optional FinOps \$

EFFICIENT EXAMPLE:

0 - PASS (0 findings)
Bulk index(), filtered kNN

"GreenOps: fail the PR, not the planet. From code AST to cluster metrics truth."



Built for the OpenSearch ecosystem

Before merge	GreenOps gate on code	GreenOps gates application code that indexes, queries, and feeds RAG (v1: JS/TS; more languages next).
After merge	OpenSearch = truth	OpenSearch runs vector search & kNN; cluster metrics + billing show if the gate mattered.
Open direction	Others can extend <code>packages/cli/src/rules.ts</code>	CLI + rules others can extend (LF-style open tooling alongside OpenSearch, not a fork of it).

“GreenOps stops bad patterns before the merge. After merging, OpenSearch runs vectors and kNN - Dashboards, CPU, latency, and billing tell you if it worked. We don’t fake that number from AST; that’s the second half of code-to-carbon.”

“Our checks live in open TypeScript on GitHub: anyone can add rules for bulk ingest, kNN size, and filters. Teams still run the same `scan --diff` in CI. GreenOps doesn’t replace OpenSearch; it sits in front of it on the pull request.”

**Code smell
(demo)**

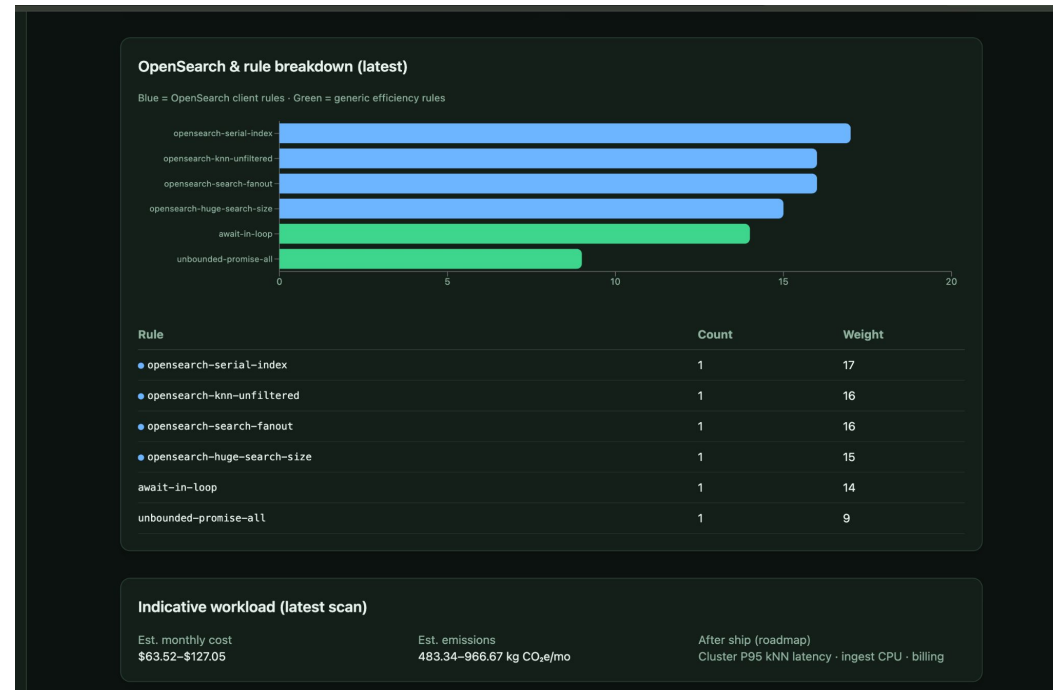
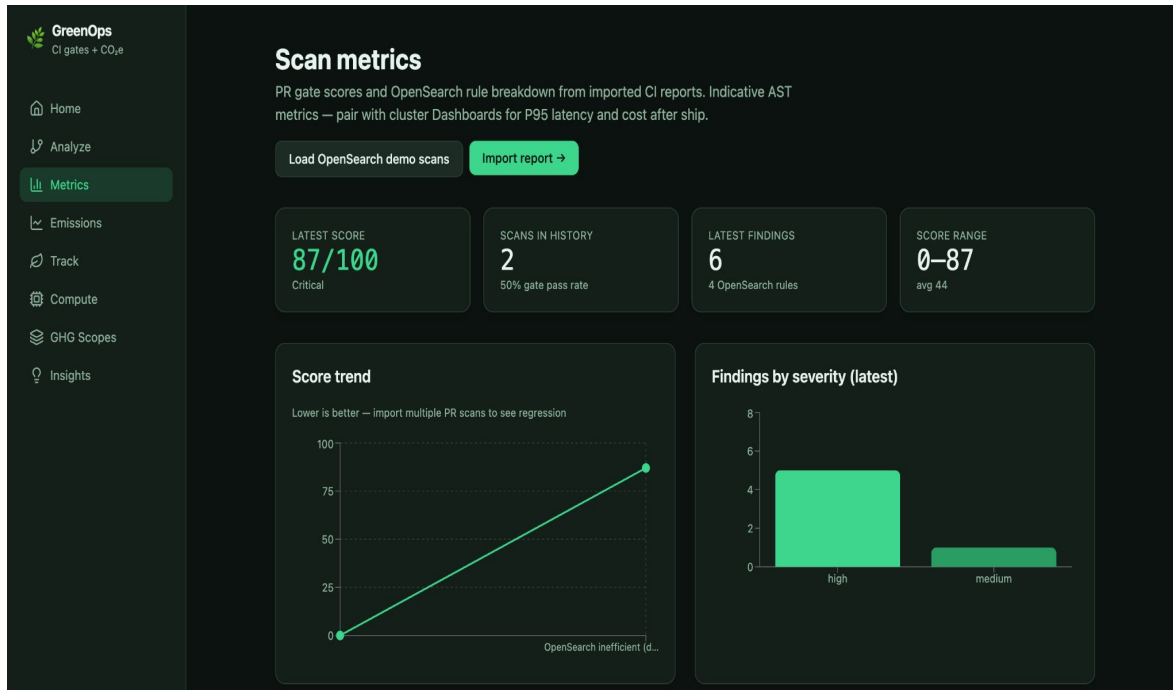
Promise.all + many calls OR Huge size / big kNN k

OpenSearch equivalent

Parallel search() fan out, Over-fetching hits & vectors for RAG

What's next and closing points

Now	Next
OpenSearch PR rules (bulk, kNN, fanout)	Auto-snapshot on every deploy
Metrics dashboard	AWS Cost Explorer auto-import
Proof timeline - before/after with git commit + score + cluster + \$	P95 kNN from Observability



DEMO

Analyze
View PR scan reports from CI. Gates use **baseline regression** (score delta + new high-severity findings), not fantasy AWS predictions.

Run locally (same as CI)

```
# First-time setup
npm run greenops:init

# PR mode - only changed files
npm run greenops:pr

# Full repo - update committed baseline (on main)
npm run greenops:baseline

# Dashboard JSON
npm run greenops:scan -- --format json --out greenops-report.json
```

Import report
Choose file No file chosen

History

Report	Score	Status	Date
OpenSearch inefficient (demo)	score 87	(failed)	12/06/2028
OpenSearch efficient (demo)	score 0	(passed)	12/06/2028

Compute & DevOps
Estimate emissions from local dev, cloud workloads, and CI — methodology aligned with [GHG Protocol](#) (power x time x grid intensity).

Grid region
Electricity grid region
Global average (0.475 kg/kWh)
Affects EV, electricity, and compute estimates (IEA / Electricity Maps-style averages).

Local workstation
Device profile: Laptop
Hours per day: 8
Work days per month: 22

Cloud & CI
Cloud kWh / month: 120
CI minutes / month: 600
From AWS Cost Explorer, GCP billing, or Azure carbon tools. GitHub Actions, GitLab CI, Jenkins, etc.

PR gate scores plus cluster snapshots tagged with git commits — compare before/after a fix to show the loop closing.

Load proof timeline | Load demo scans only | Import report →

RAG ingest fix — before/after proof
Illustrative timeline: wasteful OpenSearch client merged, then fixed. Cluster snapshots represent the same workload after bulk + filtered KNN.

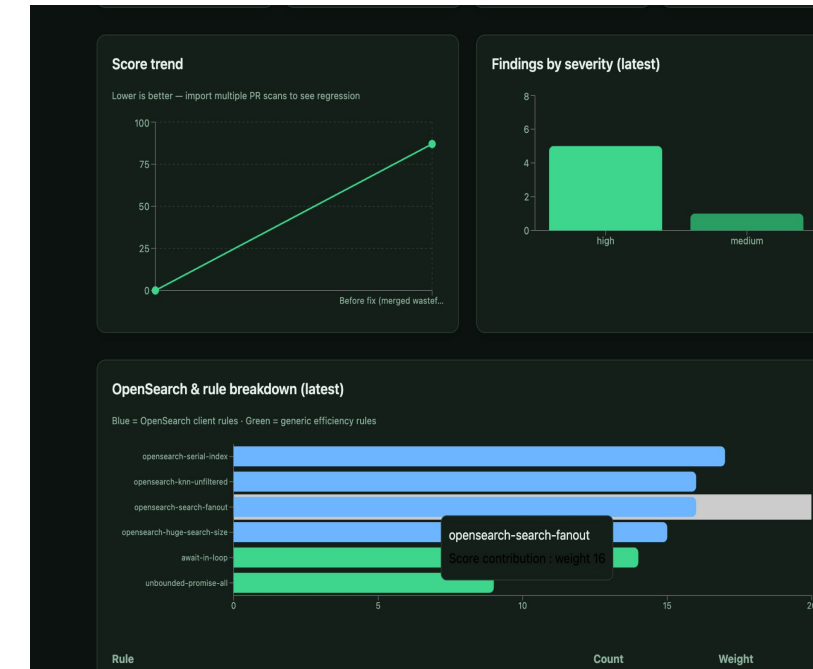
Metric	Before	After	Change
PR gate score	87/100	0/100	-87 pts (-100%)
Avg search latency	28 ms	12 ms	-16 ms (-57.1%)
Avg index latency	12 ms	4.8 ms	-7.2 ms (-60%)
Cluster avg CPU	79.3 %	41.7 %	-28.6 % (-40.7%)
Monthly infra \$	\$3,200	\$2,100	-\$1,100 (-34.4%)

Before vs after (same workload)

CLOSING THOUGHTS: How GreenOps make sense:

1. Gate in the PR - catch serial index, huge kNN, and search fanout before merge
2. Score with proof - 87 → 0 on the same OpenSearch ingest patterns you run in prod
3. Cluster truth after ship - CPU, latency, and \$ in one before/after timeline
4. Open alongside OpenSearch - AST in CI today; cluster metrics and FinOps close the loop next

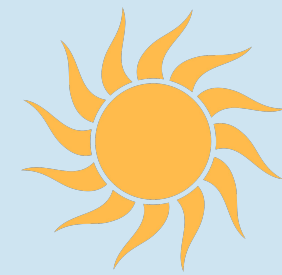
Catch waste in the PR. Measure truth in OpenSearch. Close the loop to carbon.



Demo repo: [\(REPO LINK\)](#)



 **OpenSearchCon**
INDIA



THANKYOU

