Cost-Aware and Sustainable Kubernetes Autoscaler - Project Proposal

Name: Vedant Sawal Hina Dawar
Creation Date: Oct 24, 2025
Due Date: Oct 30, 2025
Topic: Cost-Aware and Sustainable Kubernetes Autoscaler

Problem Statement & Proposed Solution

Problem Statement: Kubernetes usually scales apps based on CPU or a single traffic metric. It does not look at cloud price or how clean the power is in a region. As a result, clusters may run more pods than needed or pick the wrong kind of nodes. This wastes money, increases the carbon footprint, and can still miss latency or error targets during traffic spikes or Spot instance interruptions. There is no simple, closed loop that uses service health (like latency and queue depth) and live cost/carbon data together to auto-scale a cluster.

Proposed Solution: Build a cost and carbon-aware autoscaler. It reads service metrics (like request rate, p95 latency, and queue) from Prometheus, cost from OpenCost, and grid carbon intensity from a carbon API (e.g., ElectricityMaps/WattTime). Every few seconds, it chooses the cheapest and cleanest number of pods and node types that still meet the SLO. An example of this in action is that it tells HPA/KEDA how many pods to run and asks Karpenter to pick the best nodes (Spot instances when safe, On-Demand instances when needed). For batch/queue jobs, it can delay non-urgent work into low-carbon time windows if the SLA allows. Guardrails (like Kyverno/OPA, budgets, PDBs) keep it safe.

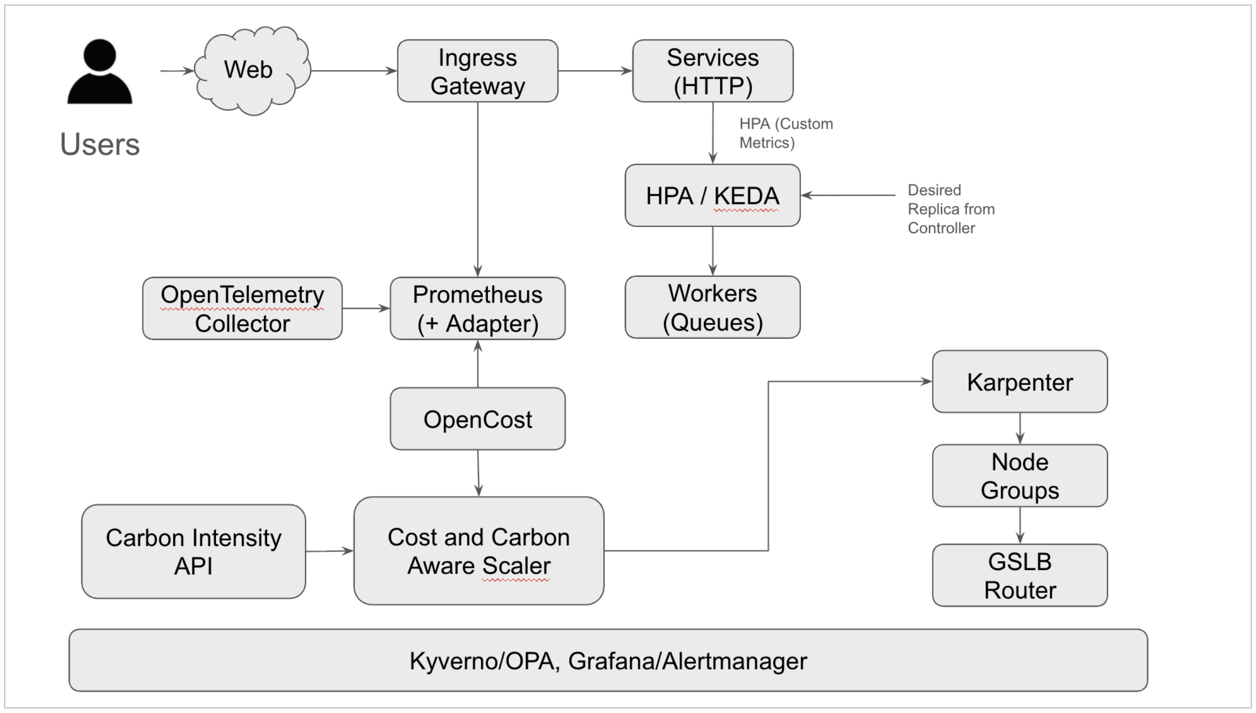
Target Users / Market Segment

- Platform/SRE & FinOps/GreenOps teams running cost-sensitive and sustainability-sensitive services on Kubernetes.
- Event/queue-heavy workloads that benefit from KEDA + price/carbon-aware scaling with optional time-shifting.
- Academic/research labs bursting on cloud that need guardrailed Spot usage and lower operational carbon.
- Biotech and Pharmaceutical Companies running large-scale genomics and oncology pipelines that need to balance speed, cost, and sustainability while staying compliant with regulations like HIPAA.

Selected Public Cloud Platform & Services

- Primary Cloud:
  - AWS
- Managed Services:
  - EKS (control plane)
  - AWS HealthOmics (omics workflows and data)
  - EC2 (nodes, including Spot)
  - IAM (least-privilege)
  - CloudWatch (logs/alarms)
  - ECR (for container images)
  - S3 (artifacts/config)
  - AWS Load Balancer Controller (ingress)
  - AWS KMS (encryption and key management)
- Open-Source/OSS Stack:
  - Prometheus + Alertmanager
  - Grafana
  - OpenTelemetry Collector
  - OpenCost
  - Prometheus Adapter
  - Karpenter
  - KEDA
  - Kyverno/OPA
- Carbon/Sustainability Signals:
  - External carbon intensity API (e.g., ElectricityMaps or WattTime) for regional grid mix forecast.

Preliminary Architecture Diagram



- Users
  - External clients who submit requests.
  - Traffic flows to the cluster via the ingress gateway.

- Ingress Gateway (Envoy/NGINX/Istio)
  - Terminates user request TLS, routes requests to services, generates rich request metrics.
  - Source for RPS, p95 latency, error rate used by autoscaling.

- Services (HTTP)
  - The stateless app Deployments behind the gateway.
  - Expose SLOs (latency/error) and resource usage, scaled by HPA.

- Workers (Queue)
  - Background/event consumers for ETL or jobs.
  - Scale based on backlog/lag via KEDA and can time-shift non-urgent work.

- OpenTelemetry Collector
  - Receives traces/metrics/logs from apps/gateway.
  - Forwards metrics to Prometheus and links exemplars from traces.

- Prometheus + Adapter (custom metrics)
  - Stores metrics (RPS, latency, queue depth, CPU/memory).

    - Adapter exposes custom metrics (e.g., requests_per_second) to HPA.

- OpenCost (real-time cost)
    - Calculates pod/node cost from cloud prices and usage.
    - Feeds $ per pod-hour / $ per 1k requests to the controller.

- Carbon Intensity API (ElectricityMaps/WattTime)
    - Provides current/forecast grid carbon for each region.
    - Enables “cleaner region” scoring and off-peak/low-carbon scheduling.

- Cost & Carbon Aware Controller (CRD + Operator)
    - Watches the ServiceAutoscalingPolicy (SLOs, min/max, budgets).
    - Pulls metrics + cost + carbon and computes cheapest/cleanest replica count that meets SLOs.
    - Sends desired replicas to HPA/KEDA and hints constraints to Karpenter.

- Horizontal Pod Autoscaler (HPA)
    - Scales HTTP services using custom metrics (RPS/pod, p95 guard).
    - Obeys stabilization to avoid thrash.

- KEDA (event/queue autoscaling)
    - Scales workers using triggers (SQS/Kafka lag, backlog size).
    - Cooldowns prevent rapid scale-in, and integrate with the controller’s targets.

- Karpenter (price-aware provisioning)
    - Provisions nodes on demand, consolidates to cut waste.
    - Chooses Spot when safe and falls back to On-Demand for SLO-critical pods.

- EC2 Node Group — Spot (preferred)
    - Lowest cost capacity and higher interruption risk.
    - Protected by PodDisruptionBudgets and eviction-safe workloads.

- EC2 Node Group — On-Demand (SLO fallback)
    - Stable capacity for critical or latency-sensitive pods.
    - Configured so that only SLO-critical workloads land here.

- Kyverno / OPA (policy guardrails)
    - Enforce resource requests/limits, namespace budgets, Spot exposure caps.
    - Prevent unsafe placements (e.g., critical pods on Spot without PDBs).

- PodDisruptionBudgets (PDBs)
    - Limit concurrent pod evictions during scale/consolidation/preemptions.
    - Maintain availability while using Spot and node consolidation.

- Grafana / Alertmanager
    - Dashboards: SLO burn, $ per 1k req, gCO_2e per 1k req, waste%, Spot exposure.
    - Alerts on latency/error spikes, spend anomalies, carbon policy breaches.

- GSLB (optional multi-cluster)
    - Latency + carbon-score routing across regions/clusters.
    - Supports “shift West during East-coast peak hours” and “prefer renewable-powered regions.”

Team Roles & Responsibilities

- Architect & IaC Owner: cluster bring-up, Terraform modules, GitOps, and networking/identity. - Hina Dawar
- Controller/CRD Developer: design ServiceAutoscalingPolicy CRD, implement Cost&CarbonAwareScaler, integrate Prometheus/OpenCost and carbon API, scale-in delays, time-shifting logic for queues. - Vedant Sawal
- Provisioning & Policy Engineer: Karpenter provisioners (Spot-preferred / On-Demand-SLO), PDBs, topology spread, Kyverno/OPA rules, carbon/cost ceilings. - Vedant Sawal
- FinOps & GreenOps Analyst: dashboards, alerts, experiments, compute $/1k requests, gCO_2e/1k requests, waste%, Pareto frontiers, budget caps, and spend guards. - Hina Dawar
- SRE & Testing: load/chaos tests (k6, Spot interruption drills), carbon-aware scenario tests, CI, canary rollouts, operational runbooks. - Vedant Sawal and Hina Dawar
- Documentation & Demo: proposal/report, architecture diagram, live demo script, reproducibility guide. - Vedant Sawal and Hina Dawar