CO_2 Monitoring for Energy-Efficient Workloads in Kubernetes: A Data Provider for CO_2-Aware Migration

Ivo Hrib
ivo.hrib@gmail.com
Jožef Stefan Institute
Ljubljana, Slovenia

Jan Šturm
jan.sturm@ijs.s
Jožef Stefan Institute
Ljubljana, Slovenia

Oleksandra Topal
oleksandra.topal@ijs.si
Jožef Stefan Institute
Ljubljana, Slovenia

Maja Škrjanc
maja.skrjanc@ijs.si
Jožef Stefan Institute
Ljubljana, Slovenia

Abstract

We present a CO_2 monitoring component developed within the FAME project's Energy Efficient Analytics Toolbox. The service continuously collects power usage for containerized workloads in Kubernetes via Kepler and fuses it with regional electricity-grid carbon intensity (e.g., ElectricityMaps) to compute per-workload CO_2 emission rates in $\mathrm{g\,s^{-1}}$. Its primary role is to *store* accurate, timestamped emission values and expose them through lightweight APIs and an optional time-series database (TimescaleDB). It acts as a data provider consumed by external orchestration services, enabling CO_2-aware migration strategies across clusters and regions.

Keywords

CO2 monitoring, Kubernetes, energy efficiency, carbon-aware computing, time-series storage, ElectricityMaps, Kepler

1 Introduction

Data centers are a significant contributor to global electricity demand. Beyond advances in hardware efficiency and renewable energy procurement, intelligent orchestration of workloads can reduce emissions by aligning computation with cleaner energy availability. A prerequisite for such carbon-aware orchestration is *reliable and accessible measurements* of workload-level emissions.

This paper introduces a CO_2 monitoring and storage service designed for Kubernetes environments. The service ingests pod/-container power data from Kepler [5], combines it with regional grid carbon intensity from ElectricityMaps [2], computes instantaneous emission rates, and *persists* the resulting time series. Unlike optimization or migration tools, this component deliberately restricts its scope: it provides measurements and exposes them via stable APIs for later consumption.

By decoupling measurement from decision-making, we ensure modularity and interoperability. External orchestrators such as the ATOS migration service in FAME D5.4 [3] can consume these metrics to implement CO_2-aware migration strategies without needing to handle the intricacies of measurement or data storage.

Our contributions are threefold: (i) a minimal but complete architecture for per-workload CO_2 measurement and storage in Kubernetes; (ii) a schema and REST API design that facilitates external consumption; and (iii) scenario-based evaluations demonstrating the potential of CO_2-aware workload migration.

Further testing will take place, utilizing real measurements and migrations from within the FAME framework, as to showcase the service's precise final capabilities, as opposed to benchmark tests.

1.1 Key-Idea

The key idea of our approach is to compute container-level CO_2 emissions by combining two complementary data sources: (i) instantaneous power consumption estimates from *Kepler*, and (ii) regional grid carbon intensity values from *ElectricityMaps*.

First, Kepler provides pod- and container-level telemetry in the form of estimated power usage $P(t)$, expressed in watts. This power signal is derived from eBPF-based kernel observations and model-based inference all provided by Keplers data source. Second, ElectricityMaps exposes a carbon intensity factor $I(t)$, expressed in $\mathrm{g\,kW^{-1}\,h}$, corresponding to the bidding zone of the node on which the container executes.

We align these two signals in time and compute instantaneous emission rates by:

$$E(t) = P(t) \cdot \frac{I(t)}{3600}$$

where $E(t)$ is the CO_2 emission rate in $\mathrm{g\,s^{-1}}$, $P(t)$ is container power in watts ($\mathrm{J\,s^{-1}}$), and the division by 3600 converts the intensity factor from per-kWh to per-second units.

These per-container emission rates are then aggregated into a time series, optionally persisted in TimescaleDB, and exposed via a REST API. This composition allows downstream orchestration services to reason about the carbon impact of workloads at fine temporal and spatial granularity, enabling CO_2-aware migration strategies.

2 Background and Related Work

Carbon-aware computing. Previous studies highlight the benefits of shifting workloads spatially or temporally to align with greener electricity supplies [4]. Such strategies require fine-grained, accurate emissions signals.

Power telemetry in Kubernetes. Kepler provides container-level power and energy estimates using eBPF-based telemetry and inference models [5, 1]. This service builds upon Kepler, avoiding duplication of low-level sensor logic.

Grid carbon intensity. ElectricityMaps aggregates near real-time and historical carbon intensity signals per bidding zone [2].

Information Society 2025, Ljubljana, Slovenia

These signals, expressed in gCO_2/kWh, provide essential context for translating power usage into CO_2 emissions.

Time-series storage. TimescaleDB extends PostgreSQL with hypertables and compression well-suited to telemetry workloads [6]. Our design can optionally persist emissions in TimescaleDB, but the service remains functional in buffer-only mode.

This paper positions our service as a foundational measurement substrate for carbon-aware orchestration in Kubernetes environments.

3 Design and Implementation

3.1 Architecture

The component runs as a Kubernetes deployment. Workers collect power metrics from Kepler, fetch grid intensity values, compute emissions, and either persist results in TimescaleDB or serve them from memory. A REST API provides read-only access to historical and recent emissions.

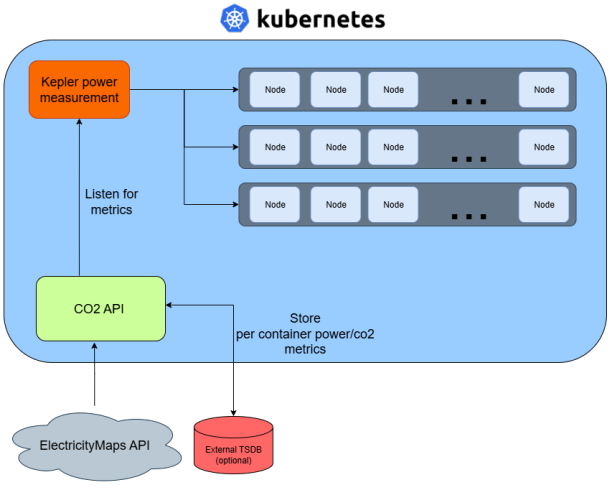

Figure 1: System architecture

3.2 Data Model

Each emission record is structured as a tuple that captures both workload identifiers and measurement values. This schema is designed to balance expressiveness with minimal storage overhead, while ensuring compatibility with external orchestration services.

- `ts` (timestamp, UTC): the precise moment when the measurement was taken, enabling time-series alignment across nodes and regions.
- `namespace`, `pod`, `container`: identifiers for locating the workload within the Kubernetes hierarchy, which is essential for container-level granularity and reproducibility.
- `node`, `region`, `country_iso2`: metadata that ties the container execution to its physical and geographical context. This supports carbon-aware decisions that depend on grid intensity differences across regions.
- `power_w`, `energy_j`: raw telemetry provided by Kepler, describing both instantaneous power and accumulated energy consumption.
- `intensity_g_per_kwh`: regional grid carbon intensity retrieved from ElectricityMaps, serving as the multiplier that translates energy into emissions.
- `co2_g_per_s`: the computed emission signal, representing the core value consumed by orchestrators.
- `source_version`: versioning tag for tracking provenance of measurements and external data dependencies.

This schema ensures that each record is self-contained, interpretable across clusters, and suitable for longitudinal analysis in time-series databases.

3.3 API Endpoints

The service exposes a lightweight REST API, designed to be easily consumed by external orchestrators or monitoring pipelines. The API emphasizes read-only access to maintain reliability and auditability.

- `GET /api/containers`: returns the set of containers currently monitored by the service, allowing orchestrators to discover available emission signals.
- `POST /api/emissions`: fetches recent emission values in bulk. This endpoint is optimized for dashboards or monitoring agents that need timely updates with low overhead.
- `POST /api/emissions/by-container`: queries the emission history of a specific container, useful for fine-grained migration or scheduling decisions.
- `GET /api/schema`: provides the data schema including units and field definitions. This enables clients to validate their assumptions and facilitates long-term interoperability across versions.

By standardizing access patterns, the API makes it possible for external services to reliably retrieve emissions information without depending on internal implementation details.

4 Evaluation

We now present evaluations based on benchmarks and scenario analyses conducted in the FAME project [3]. The goal was to assess whether exposing real-time CO_2 signals can enable meaningful emission reductions when coupled with migration strategies.

4.1 Benchmark Test

In a simple benchmark using `busybox`, a lightweight Linux container, the optimal CO_2 emissions achieved were significantly lower than the mean observed values. The key performance indicator (KPI) was defined as a 200% improvement, corresponding to a 66.6% reduction compared to baseline. Results show that this threshold can be achieved and often surpassed. The baseline is, for lack of a better, metric defined as the mean of emissions across all tracked countries with available resources for migration.

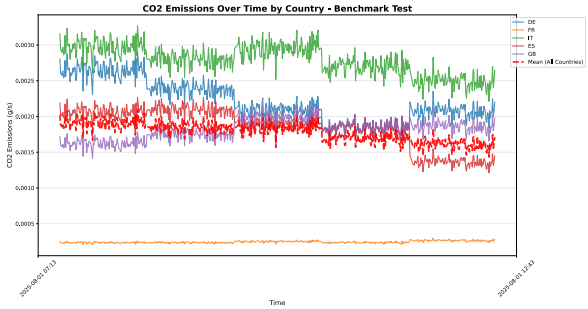

Figure 2: Small timeframe emissions of a benchmark Busybox for testing purposes

4.2 Scenario-Based Evaluation

Scenarios simulate workload migrations across subsets of European countries. Each scenario randomly selects 4–7 countries from a pool of 28, representing constrained deployment options. The system attempts to minimize emissions within these subsets. Results include:

- Scenario 1 (IS, CZ, BG, RO, AT, SE): 88.2% ± 2.1% reduction.
- Scenario 2 (DE, PL, GR, LV): 72.8% ± 5.6% reduction.
- Scenario 3 (GB, LT, SI, DE, AT, GR): 78.0% ± 1.7% reduction.
- Scenario 4 (ES, FR, GB, PL, HU, LT, SE): 89.6% ± 1.1% (best case).
- Scenario 5 (LV, ES, HU, LT): 32.4% ± 12.7% (worst case).
- All Countries: 87.7% ± 1.7% reduction (ideal case).

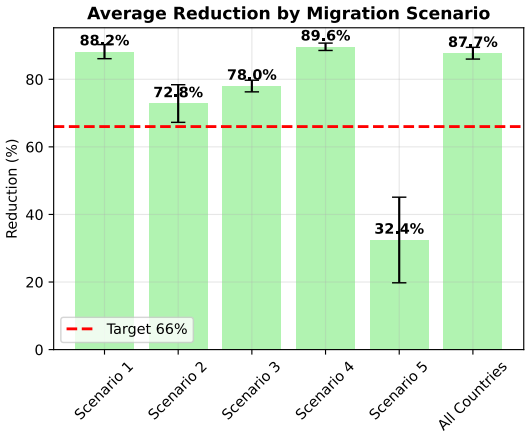

Figure 3: Plot of average reductions per scenario

Across all scenarios, at least one migration was executed per window, with an average emission reduction of 74.8%. These results confirm that even under limited availability, CO_2-aware migration strategies yield substantial benefits.

4.3 Insights

The best-performing scenario demonstrates that careful selection of even a limited number of regions can approach the effectiveness of full global availability. Conversely, the poorest-performing scenario illustrates the dependency on geographic flexibility. Overall, results validate that exposing reliable CO_2 signals through our service empowers orchestration layers to meet or exceed environmental KPIs.

5 Limitations and Future Work

The reported emissions are estimates subject to the accuracy of both Kepler's models and grid intensity data. As a result, the benchmark tests previously performed may not fully capture all possible scenarios, as grid dependency may sometimes force suboptimal migrations in the CO2-system as per resource availability. Resolution is limited by the update frequency of intensity sources, and storage requirements increase with sampling granularity.

Future work will focus on service options to adjust granularity and tackle scalability issues within the service.

6 Conclusion

We presented a Kubernetes-native CO_2 monitoring service that provides real-time emissions data through stable APIs. Evaluations demonstrate that when coupled with migration strategies, these metrics enable significant emission reductions, often surpassing KPI thresholds. Future work will include integration with more compute-intensive workloads, multi-source intensity aggregation, and cryptographic provenance for auditability.

Acknowledgements

This work was supported by the FAME project under the European Union's Horizon Europe programme. We thank the Kepler community and colleagues who contributed feedback during testing.

References

[1] eBPF Foundation. [n. d.] Ebpf. Accessed 2025-09-05. https://ebpf.io/.
[2] Electricity Maps. [n. d.] Electricity maps api documentation. Accessed 2025-09-05. https://www.electricitymaps.com/.
[3] FAME Consortium. 2025. Energy efficient analytics toolbox ii. Deliverable D5.4. (2025).
[4] Google. 2020. Carbon-intelligent computing at google. Whitepaper/blog overview; Accessed 2025-09-05. https://www.google.com/.
[5] Kepler Project. 2024. Kepler: kubernetes-based efficient power level exporter. Accessed 2025-09-05. https://github.com/sustainable-computing-io/kepler.
[6] Timescale Inc. [n. d.] Timescaledb: postgresql for time-series & events. Accessed 2025-09-05. https://www.timescale.com/.