Putting Green Software Principals into Practice.

James Uther

March 12, 2025

Putting green software principles into practice.

James Uther
james.uther@oliverwyman.com
Oliver Wyman
London, UK

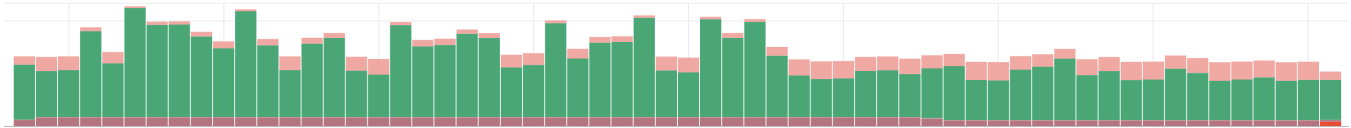Figure 1: CO$_2$ footprint of a system over time.

ABSTRACT

The need and theoretical methods for measuring and reducing CO$_2$ emitted by computing systems are well understood, but real-world examples are still limited. We describe a journey towards green software for a live product running on a public cloud. We discuss practical solutions found, in particular using the cost implications of serverless systems to drive efficiency. We end with some 'green software' principles that worked well in this project.

KEYWORDS

Cloud computing, Sustainability, Programming teams, IT Governance

1 INTRODUCTION

We recently built a mostly green-field product that involved substantial data processing, including developing and training various models and running production data pipelines using those models. Taking the advice "every job can be a climate job" [14] we sought from the outset to make the system as low-carbon as possible, given the business constraints.

We entered the project with a good understanding of green software principles, including the various sustainable architecture guidelines published by major hyperscalers [4, 18, 25], a "Green Software for Practitioners" [17] badge, and personal motivation to make a difference where we were.

We started with some rough components that were running on a well known public cloud, and had reasonably free access to the cloud products as long as we coöperated with our corporate IT organisation. There was a gap between the proof of concept code and a system that could be run confidentially in production with orders of magnitude more data, so we were free to rewrite as necessary, within schedule constraints.

We now have a system and supporting infrastructure that could be considered 'green'. In Section 2 we will describe how we structured the project and systems, Section 4 will discuss how the team found the process, and then we will offer some discovered principles in Section 5. Note that finding these principles and building the system was an iterative process, with much learning and rebuilding.

2 BUILDING THE INFRASTRUCTURE

We took a pragmatic definition of green software as "Does the job while producing minimal CO$_2$e". The 'does the job' part is important, as it introduces other architecture drivers such as corporate policies, security, reliability, schedule, budget, and team capabilities. These are driven by business or institutional goals, and any green software principles or actions that make these harder for the team are likely to be ignored or resisted.

We could not identify any incentives to adopt green principles beyond personal ethics. While there are efforts to introduce legislation, policies and standards to drive the implementation of green software [11, 13, 27], none applied to us at the time of writing, and in some cases are now unlikely to be finalised. This leaves green software in the odd space of being both desirable and at the same time all but valueless by the metrics of a company, and a distraction for most employees.

Thankfully, in the particular case of cloud computing the cost of the service can be used as a proxy for energy use, which can be related through grid intensity to an estimated CO$_2$ footprint. Cost is most definitely a good incentive for most businesses and thus employees. Low latency billing data from the cloud is watched carefully by cloud users, and proved invaluable in driving (and thus CO$_2$) reduction.

2.1 The platform

We needed to provide good *infrastructure* [20, 21] that satisfies the user, while operating in a way that fulfils broader goals. We are used to compute infrastructure that allows us to get work done while transparently looks after security, IT policies, budget etc. We needed to build infrastructure that supported these corporate goals while producing as little CO$_2$ as possible.

Our strategy was to use products that are charged by usage and can automatically scale according to load (ideally to and from 0). Such *serverless* products are becoming ubiquitous as the cost and operational benefits can be considerable, particularly for spiky workloads. If the usage/billing data is available quickly they give an excellent feedback signal: Increased usage implies increased CO$_2$ and that usage is quickly reflected in the billing console. Actions to decrease cost involve reducing compute usage, which reduces CO$_2$, linking good CO$_2$ management to good financial management.

Architectures for data intensive applications are well understood [22] and have been packaged into various platforms, frameworks and libraries [15, 26, 32]. While we had business requirements regarding throughput and reliability, we also had to consider institutional knowledge and procurement policies. Luckily this left us with a few good options, and we chose a managed Spark [32] platform. The platform offered the ability to auto-scale clusters, and later a fully serverless cluster feature. We were also able to use efficient and cost effective ARM CPUs in most cases.

Serverless products at a large scale allow the provider optionality to more efficiently schedule workloads and increase hardware utilisation. At a small scale or with a known planned workload, this benefit vanishes, so relying on serverless options to reduce CO_2 (and cost) does not work well for on-prem compute.

2.2 Software

The platform guided users to use pyspark [16], which transforms python code into faster JVM operations [19] that can be distributed on an auto-scaling cluster. Although Python was well known in the team, the need to transition from pandas [31] to pyspark was more of a challange, but necessary as the production data was large enough to need distribution across a cluster. We did need to optimise the logic of the application itself, but could be guided by latency and cost considerations, also reducing CO_2. While there is scope for further optimisation, the need to cater for diverse and unknown future maintainers limited our appetite for rewriting business logic in Rust.

We tried various ways of avoiding the *thundering herd problem* [33]. If everyone starts all batch jobs all at once (say 00:00 UTC, or on the hour), hyperscalers need to size capacity to those peaks leading to over-investment, higher embodied carbon, and the use of gas generators. Some cron implementations have wildcard values, allowing the data centre to schedule the job any time within given bounds. Mobile platforms overcome a related issue and save energy by providing a tolerance parameter within their timer APIs [5]. Our platform provided only simple cron scheduling, so we are simply staring jobs at a fixed arbitrary time that is not on the hour. We hope this off-beat predictable schedule may help cloud providers manage hardware provisioning [8]. We would welcome being able to schedule jobs based on grid intensity ('when the wind blows'), but this is a feature best provided by the platform.

2.3 Feedback

We built dashboards to display daily costs associated with particular clusters and jobs, enabling us to highlight the need to further optimise parts of the system. It also helped prevent inefficient algorithms, architectures and libraries from gaining a foothold, as the costs associated quickly became apparent.

Section 3 describes our attempt to measure actual CO_2, but it was not as effective to drive behaviour as cost. We also found our ability to measure increasingly limited as some workloads were moved to SAAS platforms and the compute hardware moved out of view.
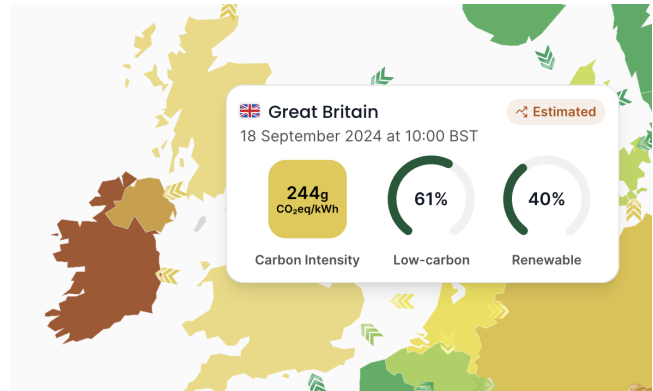
2.4 Power

---
[1]https://app.electricitymaps.com/map

Figure 2: Live CO₂ intensity map [1]

We were able to use a cloud zone that claimed to be carbon neutral. We recognise that all hyperscalers make use of opaque carbon credit schemes [6, 23] and a better assumption might be that the carbon intensity closely follows the regional grid intensity [2] as shown in Figure 2. Even this assumption is now doubtful as data centres are installing gas generators. Time-matched energy purchases (T-EACs) [1, 28] look like a way forward here and we look forward to seeing hyperscalers move in this direction.

3 MEASUREMENT

Measurement is a foundation of good engineering. While we assumed the cloud bill was a reasonable proxy, we did try to get more detailed and true measure of our CO_2 footprint. There are now a few tools available to measure the CO_2 output of a system [7, 24, 29] and cloud providers also provide customers with carbon footprint reports. We have noticed that it's still unusual for SAAS platforms to do this, and estimating the footprint of their control planes and serverless workloads becomes impossible.

Our cloud platform provides a carbon usage dashboard, although the data trails by 3 months as electricity market intricacies are settled making it unhelpful for optimisation. The tool was also on a

---
To use the customer carbon footprint tool

1. Sign in to the AWS Management Console and open the AWS Billing and Cost Management console at https://console.aws.amazon.com/billing/.
2. On the navigation pane, choose **Legacy Pages**, then **Cost and Usage Reports**.

Figure 3: The CO₂ dashboard is considered legacy.

page of deprecated dashboards, had a minimun usage threshold to activate, and did not support reporting for sub-accounts, which was our situation. We view this lack of reliable, regular and `actionable` information about our CO_2 footprint as a major gap in most serverless platform offerings, and encourage providers to provide an API for this purpose [9, 10].

As our workloads started to move to the managed serverless version of our spark platform we enquired whether that platform would support such an API. The provider admitted that CO_2 reporting was not on their roadmap.

We reverted to `Cloud Carbon Footprint` [29] which has a methodology [30] for converting billing data to estimated CO_2

footprint. Although usually used as an online system, we were able to instead generate a lookup table [3] that was imported into our platform, and used to augment the billing dashboard with our estimated footprint as in Figures 1 and 4. The values here are based
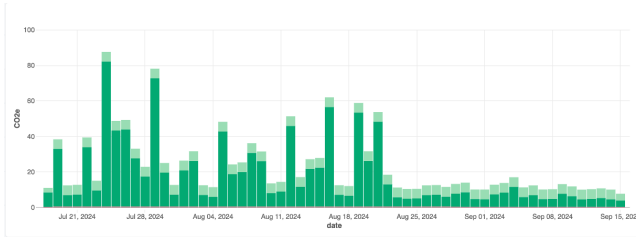

Figure 4: Our CO_2 footprint by day.

on estimates of the CO_2 footprint of the assumed hardware, given published figures and measurements of CPUs, memory, network etc, mixed with published PUE figures, and assumptions about management overhead. Kepler extends this with models trained on actual hardware [3], but is focussed on kubernetes and thus not applicable here.

4 PEOPLE

Having built foundations, we worked with others to build out the product. In a busy project there is little appetite for introducing risk and work for non-core goals. Thankfully the infrastructure of the project guided work to be low carbon while comfortably fulfilling the business needs. The data centre location (and grid intensity) was transparent. The billing data was vital for cost management. The platform scaled jobs to the lowest cost/CO_2 configuration automatically, or warned via billing when not being used effectively.

The project became a good learning opportunity for many. Often contributors were from a data science background which does not emphasise efficiency. Others were familiar with on-prem hardware where software efficiency has little impact on cost (which is carried in the initial purchase and fairly constant power bill). Some had not worked with data at this scale before. In all cases the move to a scalable serverless platform involved some learning around structuring solutions to enable distributed computing, and some lessons in efficiency and algorithm choice. It was not directly related to CO_2 footprint, but as the team learned the software improved and operating costs dropped. CO_2 dropped alongside.

5 SUMMARY

We would summarise the architecture principles and activities we used as:

- **Infrastructure** which is intrinsically low carbon.
  - **Renewable** electricity.
  - **#LightSwitchOps** - switch off unused resources. [12]
  - Select the **right size** of resources.
- **Feedback** by publishing usage (cost & CO_2) data.
- **Optimise** the software.

---

[3]https://www.cloudcarbonfootprint.org/docs/creating-a-lookup-table/

These guided us to serverless solutions which was a new approach for some, but worth embracing for many reasons beyond their green credentials.

In our case, further reduction in CO_2 footprint is now largely dependant on our platform providers and their choice of electricity supply, cron services, scheduling algorithms, CO_2 footprint dashboards, supported software, hardware, cooling, etc. We decided that a large, dedicated organisation can build a more efficient platform than us, but we recognise it has left near-term progress in the hands of those organisations. In particular we would like to see better low latency reporting of our actual CO_2 impact.

6 CITATIONS AND BIBLIOGRAPHIES
REFERENCES

[1] EnergyTag 2024. *Carbon accounting & tracking for* 24/7 *clean grids*. EnergyTag. https://energytag.org/
[2] Electricity Maps 2024. *Climate Impact by Area Ranked by carbon intensity of electricity consumed (gCO₂eq/kWh)*. Electricity Maps. https://app.electricitymaps.com/map
[3] Marcelo Amaral, Sunyanan Choochotkaew, Eun Kyung Lee, Huamin Chen, , and Tamar Eilam. 2023. *Exploring Kepler's potentials: unveiling cloud application power consumption*. https://www.cncf.io/blog/2023/10/11/exploring-keplers-potentials-unveiling-cloud-application-power-consumption/
[4] Amazon. 2024. *Sustainability Pillar - AWS Well-Architected Framework*. Amazon. https://docs.aws.amazon.com/wellarchitected/latest/sustainability-pillar/sustainability-pillar.html
[5] Apple. 2024. *Energy Efficiency Guide for iOS Apps - Minimize Timer Use*. https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/MinimizeTimerUse.html
[6] Anders Bjørn, Shannon M Lloyd, Matthew Brander, and H Damon Matthews. 2022. Renewable energy certificates threaten the integrity of corporate science-based targets. *Nature Climate Change* 12, 6 (2022), 539–546.
[7] Huamin Chen and Chen Wang. 2022. Sustainability the Container Native Way. In *Open Source Summit*.
[8] Adrian Cockcroft. 2023. *Don't follow the sun: Scheduling compute workloads to chase green energy can be counter-productive*. https://adrianco.medium.com/dont-follow-the-sun-scheduling-compute-workloads-to-chase-green-energy-can-be-counter-productive-b0cde6681763
[9] Adrian Cockcroft. 2024. Cloud Provider Sustainability, Current Status and Future Directions. In *QCon London*. https://github.com/adrianco/slides/blob/master/Cloud%20DevSusOps%20London.pdf
[10] Adrian Cockroft and et.al. 2023. *Github discussion - Extend Kepler to provide real time carbon metrics*. https://github.com/sustainable-computing-io/kepler/discussions/600
[11] The European Commission. 2020. *Implementing and delegated acts - Taxonomy Regulation*. https://finance.ec.europa.eu/regulation-and-supervision/financial-services-legislation/implementing-and-delegated-acts/taxonomy-regulation_en
[12] Holly Cummins. 2023. *Why Cloud Zombies Are Destroying the Planet and How You Can Stop Them*. https://hollycummins.com/cloud-zombies-qcon-london/
[13] Energy Department for Business and Industrial Strategy. 2024. *Environmental reporting guidelines including mandatory greenhouse gas emissions reporting guidance*. https://www.gov.uk/government/publications/environmental-reporting-guidelines-including-mandatory-greenhouse-gas-emissions-reporting-guidance
[14] Project Drawdown. 2024. *Climate Solutions at Work*. https://drawdown.org/publications/climate-solutions-at-work
[15] The Apache Foundation. 2024. *Apache Beam*. The Apache Foundataion. https://beam.apache.org/
[16] The Apache Foundation. 2024. *PySpark Overview*. https://spark.apache.org/docs/latest/api/python/index.html#
[17] The Green Software Foundation. 2024. *Green Software for Practitioners (LFC131)*. The Linux Foundation. https://training.linuxfoundation.org/training/green-software-for-practitioners-lfc131/
[18] Google. 2024. *Design for environmental sustainability*. Google. https://cloud.google.com/architecture/framework/sustainability
[19] Isaac Gouy. 2024. *The Computer Language Benchmarks Game*. https://benchmarksgame-team.pages.debian.net/benchmarksgame
[20] Saul Griffith. 2021. *Infrastructure and Climate Change*. https://longnow.org/seminars/02015/sep/21/infrastructure-and-climate-change/
[21] Saul Griffith. 2022. *Electrify: An optimist's playbook for our clean energy future*. MIT Press.

[22] M. Kleppmann. 2017. *Designing Data-intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly Media. https://books.google.co.uk/books?id=BM7woQEACAAJ

[23] Lissy Langer, Matthew Brander, Shannon M Lloyd, Dogan Keles, H Damon Matthews, and Anders Bjørn. 2023. Does the purchase of voluntary renewable energy certificates lead to emission reductions? A review of studies quantifying the impact. *SSRN* (2023). https://ssrn.com/abstract=4636218

[24] Loïc Lannelongue and Michael Inouye. 2024. *Green Algorithms - Towards environmentally sustainable computational science.* https://www.green-algorithms.org

[25] Microsoft. 2024. *Well-Architected for Microsoft Cloud for Sustainability.* Microsoft. https://learn.microsoft.com/en-us/industry/well-architected/sustainability/

[26] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. 2018. Ray: A distributed framework for emerging AI applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18).* 561−577.

[27] Securities and Exchange Commission. 2024. *The Enhancement and Standardization of Climate-Related Disclosures for Investors.* https://www.sec.gov/files/rules/final/2024/33-11275.pdf

[28] Maud Texier. 2021. *A timely new approach to certifying clean energy.* google. https://cloud.google.com/blog/topics/sustainability/t-eacs-offer-new-approach-to-certifying-clean-energy

[29] Thoughtworks. 2024. *Cloud Carbon Footprint.* https://github.com/cloud-carbon-footprint/

[30] Thoughtworks. 2024. *Cloud Carbon Footprint - Methodology.* https://www.cloudcarbonfootprint.org/docs/methodology/

[31] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 56 − 61. https://doi.org/10.25080/Majora-92bf1922-00a

[32] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing* (Boston, MA) *(HotCloud'10).* USENIX Association, USA, 10.

[33] Štěpán Davidovič. 2016. Distributed Periodic Scheduling with Cron. In *Site reliability engineering: How Google runs production systems.* O'Reilly Media, Inc.