Putting green software principals into practice.

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

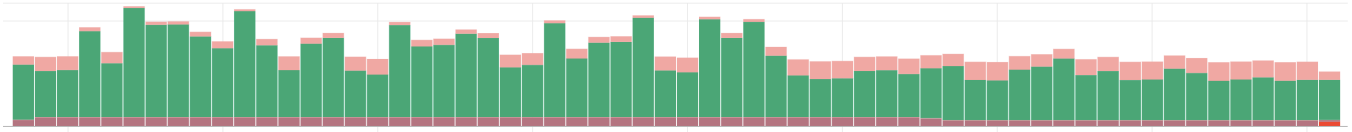

Figure 1: CO_2 footprint of a system over time.

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 our summary of 'green software' principals.

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 "all jobs are climate jobs" 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 principals, including the various sustainable architecture guidelines published by major hyperscalers [3, 11, 15], a "Green Software for Practitioners" [10] 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 we had fairly 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 principals in Section 5. Note that finding these principals and building the system was an iterative process, with much learning and rebuilding.

LOCO '24, December 3 2024, Glasgow, UK
2024.

2 BUILDING THE INFRASTRUCTURE

We took a pragmatic definition of green software as "Does the job while producing minimal CO_2e". 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 principals 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 principals beyond personal ethics. While we understand there may be legislation or accounting standards in the future, they did not exist at the time of writing, which 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 is a reasonable proxy for energy use, which can be related through grid intensity to CO_2 footprint, and cost is most definitely a good incentive for most businesses and thus employees. Low latency billing data from the cloud proved invaluable in driving cost (and thus CO_2) reduction.

2.1 The platform

We needed to provide good *infrastructure* [12, 13] that satisfies the user, while operating in a way that fulfils broader goals. We are used to compute infrastructure that allows us to get stuff done[1] 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 also 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.

[1] your mileage may vary

We found a managed spark [19] platform that fulfilled the business requirements, offered auto-scaling clusters, and later a fully serverless compute feature.

2.2 Software

The platform guided users to use pyspark [9], which transforms python code into faster JVM operations [5] that can be distributed on an auto-scaling 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* [20]. The argument is that if everyone starts everything all at once (say 00:00 UTC), 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 [4]. Our platform provided only simple cron scheduling, so we are simply staring jobs at a fixed arbitrary time that is not on the hour[2].

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 compute moved out of view.

2.4 Power

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, 14] and a better assumption might be that the carbon intensity closely follows the regional grid intensity [2]. Even this assumption is now doubtful as data centres are installing gas generators. Time-matched energy purchases (T-EACs) [1, 16] 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, 17] and cloud providers also provide customers will 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.

We reverted to `Cloud Carbon Footprint` [17] which has a methodology [18] for converting billing data to estimated CO_2 footprint. Although usually used as an online system, we were abled to instead generate a lookup table [3] that was imported into our platform, and used to augment the billing dashboard with our estimated footprint (see Figure 1).

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 used to using 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 distribution, 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 principals and activities we used as:

- **Infrastructure** which is intrinsically low carbon.
  - **Renewable** electricity.
  - **#LightSwitchOps** - switch off unused resources. [8]
  - Select the **right size** of resources.
- **Feedback** by publishing usage (cost & CO_2) data.
- **Optimise** the software.

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.

[2] https://xkcd.com/221/

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

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 (gCO_2eq/kWh)*. Electricity Maps. https://app.electricitymaps.com/map

[3] Amazon. 2024. *Sustainability Pillar - AWS Well-Architected Framework*. Amazon. https://docs.aws.amazon.com/wellarchitected/latest/sustainability-pillar/sustainability-pillar.html

[4] Apple. 2024. *Energy Efficiency Guide for iOS Apps - Minimize Timer Use*. https://developer.apple.com/library/archive/documentation/Performance/Conceptual/EnergyGuide-iOS/MinimizeTimerUse.html

[5] Doug Bagley, Brent Fulgham, and Isaac Gouy. 2024. *The Computer Language Benchmarks Game - Python 3 versus Java fastest performance*. https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/python.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] Holly Cummins. 2023. *Why Cloud Zombies Are Destroying the Planet and How You Can Stop Them*. https://hollycummins.com/cloud-zombies-qcon-london/

[9] The Apache Foundation. 2024. *PySpark Overview*. https://spark.apache.org/docs/latest/api/python/index.html#

[10] The Green Software Foundation. 2024. *Green Software for Practitioners (LFC131)*. The Linux Foundation. https://training.linuxfoundation.org/training/green-software-for-practitioners-lfc131/

[11] Google. 2024. *Design for environmental sustainability*. Google. https://cloud.google.com/architecture/framework/sustainability

[12] Saul Griffith. 2021. *Infrastructure and Climate Change*. https://longnow.org/seminars/02015/sep/21/infrastructure-and-climate-change/

[13] Saul Griffith. 2022. *Electrify: An optimist's playbook for our clean energy future*. MIT Press.

[14] 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

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

[16] 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

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

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

[19] 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.

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