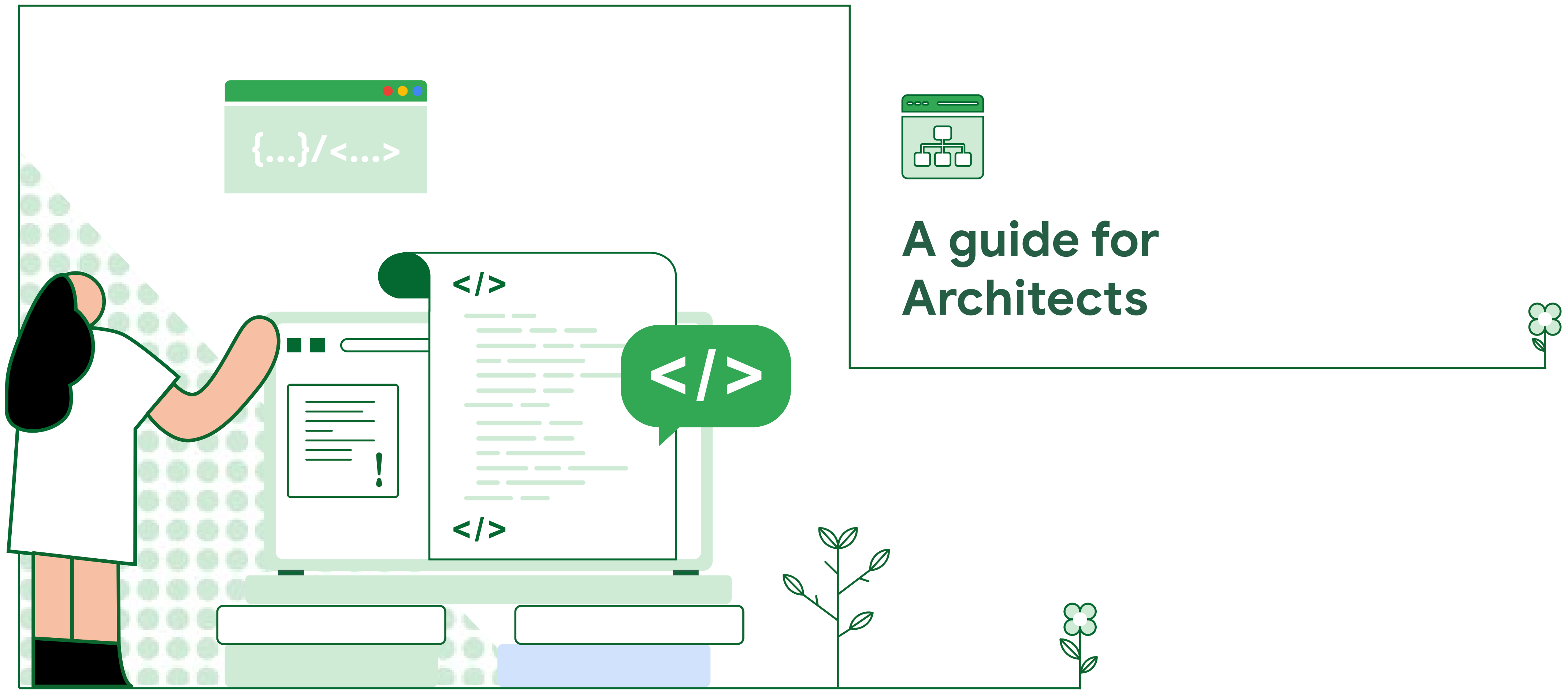
Go Green Software



A guide for Architects

Google Cloud

Contents

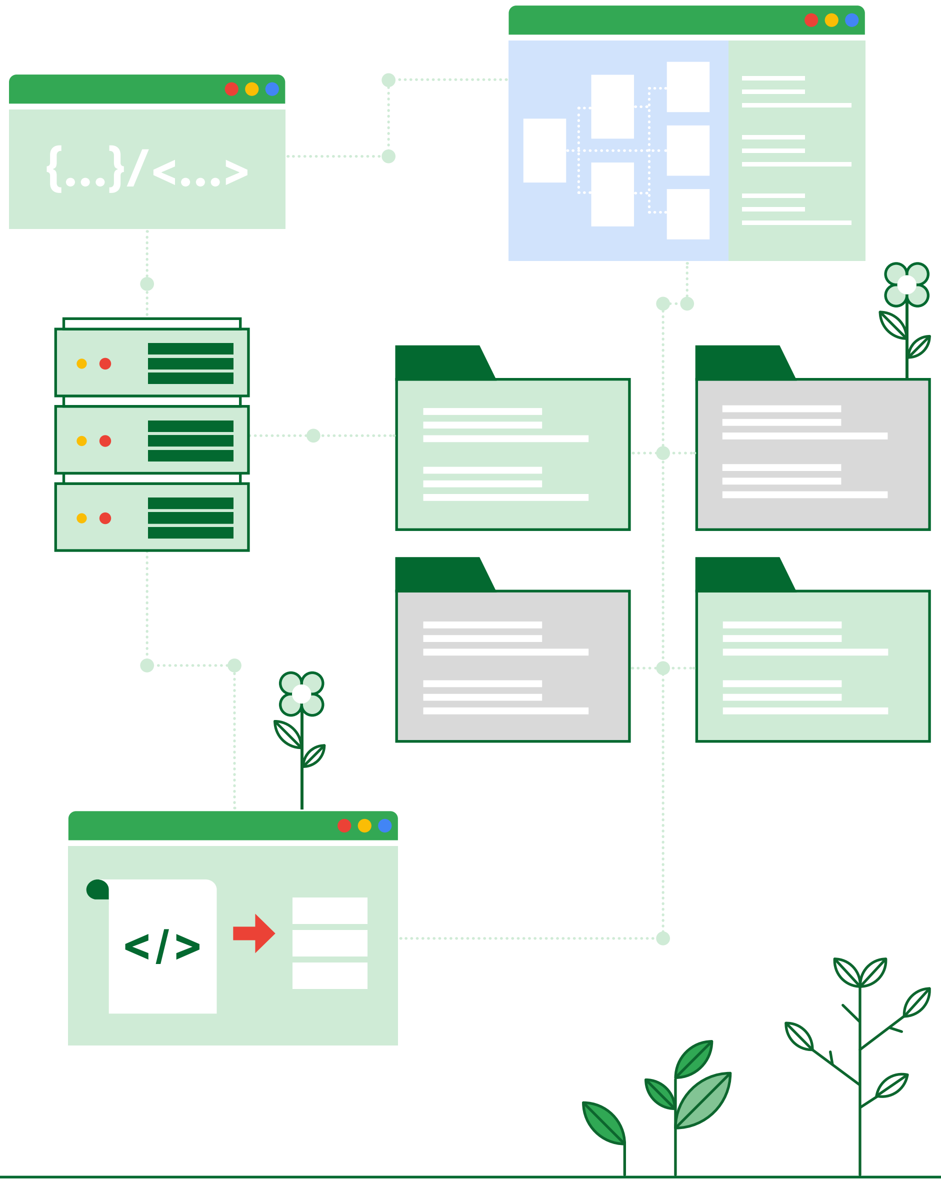
Introduction

An architect (e.g. Solution, Technical or Systems) will typically interface with multiple different personas in an organization to understand the business and technical requirements, the technologies available and the anticipated development approach through the software development lifecycle.

The architects are expected to align the requirements to a technology approach which provides the appropriate cost, reliability, scalability and now should also take into account long term sustainability.

Architects are critical for sustainability as they are one of the only technical roles who have a horizontal view of the solution. Architects working with all the project/programme personas could help inform and educate on how their system design embeds sustainable principles. The architect is also responsible for generating a set of acceptance test requirements which ultimately determines whether the high level requirements have been met, in the case of this document, how will they log and monitor the carbon efficiency of the system.

Architects must consider the impact of choosing specific design patterns from monoliths through to microservices whilst also investigating strategies on how to reduce data consumption, how to minimize compute cycles, how to optimize network traffic and when and how "jobs" need to be performed. In this next section we can identify how different strategies could be implemented to help become more efficient.

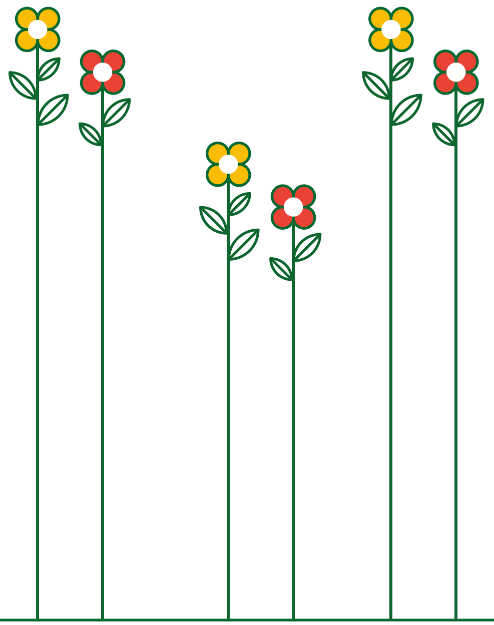


Microservices architecture

Complexity: Easy | Impact: High

Is a microservice or monolith design pattern more wasteful? Typically microservices can be automatically scaled in isolation, individually configurable and should create less waste of resources.

Using serverless or container solutions it is possible to to reduce the size of the footprint to consume less energy. However, **one side effect of the microservice pattern is the introduction of the need for multiple services to communicate with each other over various networks**. Microservices can therefore become highly distributed and it is possible to see an increase in network traffic. With this in mind the architect should **ensure they consider the payloads being returned, the proximity of each service to each other and also whether the correct transfer protocol is being consumed**. It is possible a 'chatty' microservice can end up more wasteful than a monolith.

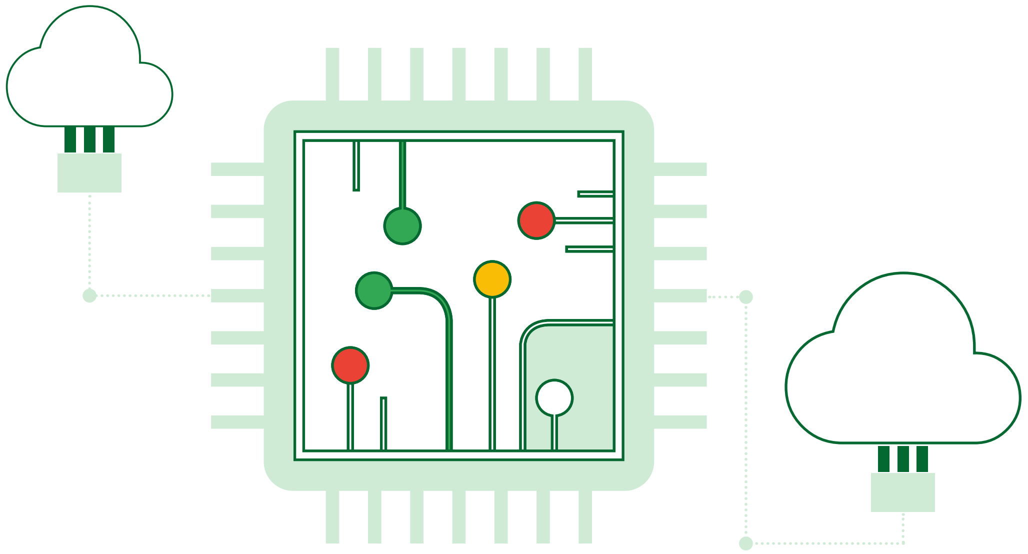


Event-driven design

Complexity: Medium | Impact: High

The adoption of RESTful APIs creates a methodology where a response is required immediately. **Real-time processing has transformed how our consumers receive data and systems are built.** However, in a number of use cases it is possible to move from a synchronous request-response paradigm to an asynchronous process.

An event driven architecture allows you to push workloads into streams to be processed and dealt with when needed by different components. The benefit of this approach is it is possible in some cases to offload these components through worker processes in other regions, later times and also batch them up. **We can use computers as and when it is needed rather than wasting it in idle states waiting**.



Caching strategies and CDNs

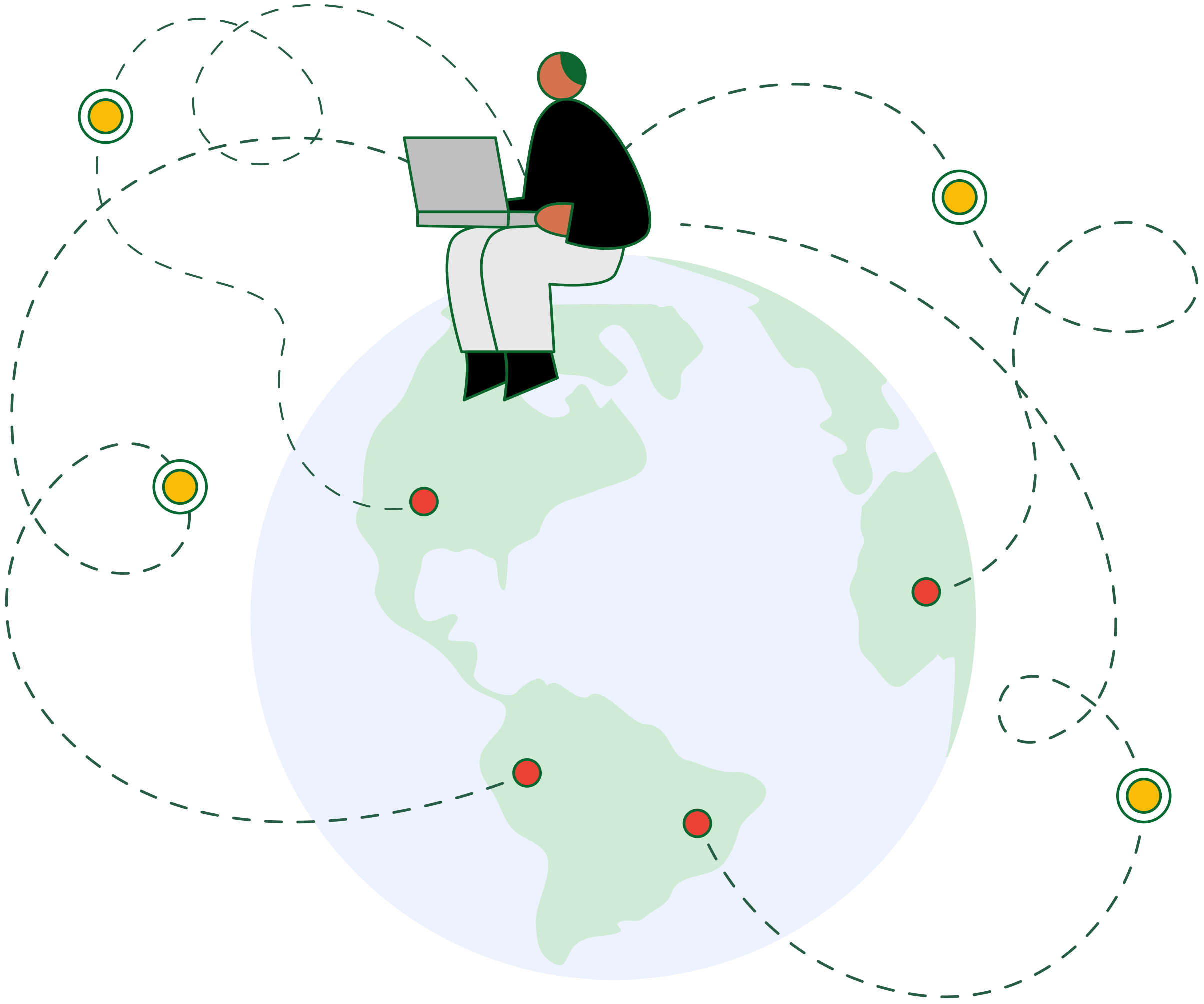
Complexity: Medium | Impact: Medium

When it comes to storing and querying with databases an architect will need to **ensure you are using the right data type**.

However, going deeper than the database engine, **the application can minimize wasteful queries by adopting a cache strategy**, a good indicator of what to cache is to find any element where multiple executions of some request will cause the same outcome. Any data which doesn't change too often that is read very frequently is ideal for a cache, resulting in less computational power required. The architect also needs to ensure that data is appropriately pruned, indexes put in place and queries that occur not only traverse the shortest route but also minimize the amount of data returned.

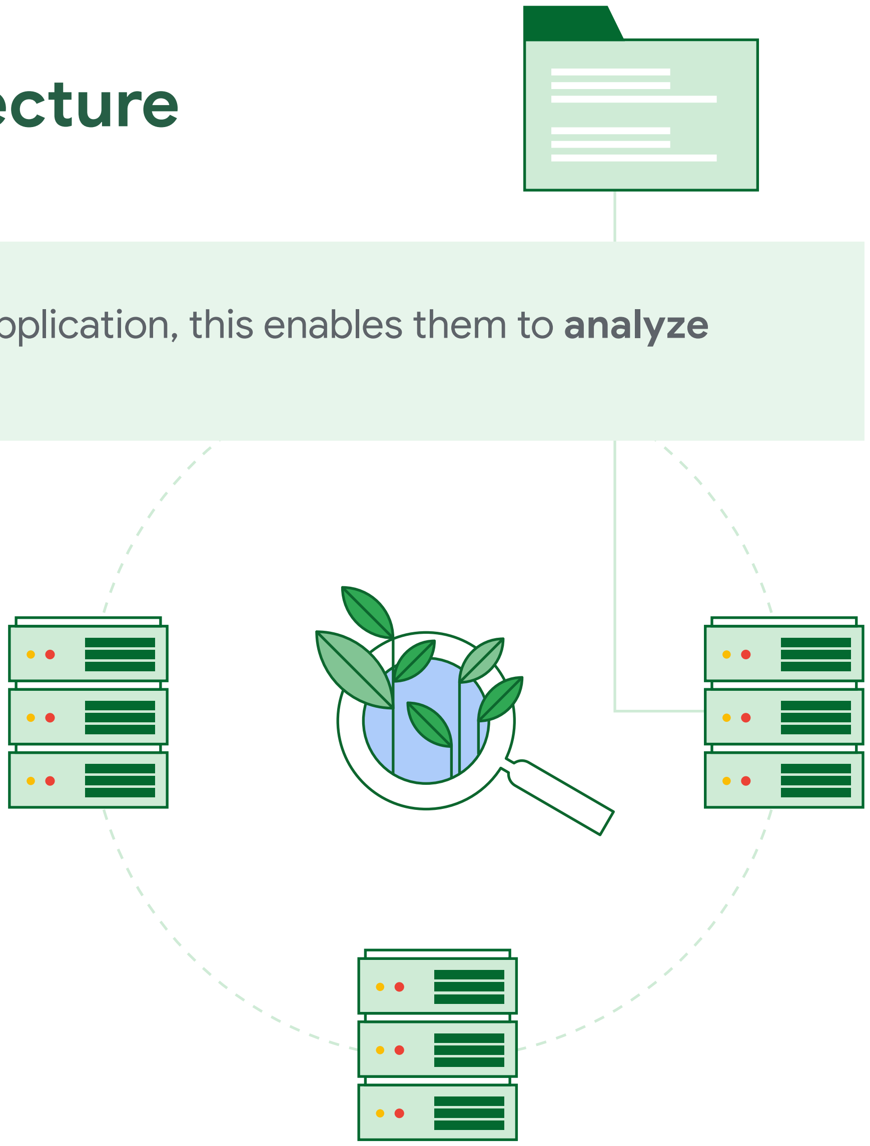
Transporting data within your network has a huge impact on your application's emissions. A content delivery network (CDN) refers to a geographically distributed group of servers and data centers that deliver content to your user from a location as close as possible to them.

If you have a global footprint as an architect, you need to ensure that where possible you are **presenting information from a location as close to the user as possible** and minimize requests traversing the globe. CDN's are also able to cache content, **caching within a CDN at the edge will ensure that fewer requests need to be processed and transported around your data center**, plus also has the added benefit of reducing latency for the user.



Logging and monitoring architecture

Complexity: Medium | Impact: Medium



An architect will always want to collect logs from their application, this enables them to **analyze usage patterns, debug errors and optimize**.

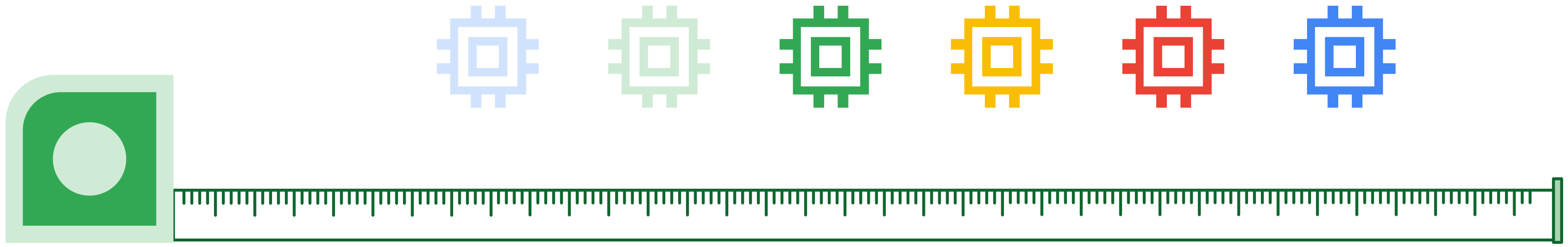
However, log collection and processing can consume significant energy if not designed appropriately. An architect should consider the granularity of the log level required and also whether this information needs to persist. The architect needs to remember their goal is to reduce data consumption, and how to minimize compute cycles. As log files grow, processing these files and deriving valuable insight becomes computationally expensive, so it's best to ensure what is required from both a business and compliance perspective and **ensure that these files are appropriately pruned, deleted or moved to cold storage and not allowed to grow out of control**.

Carbon efficiency monitoring

Complexity: Medium | Impact: High

At the beginning of this section it was mentioned that an architect should be defining the acceptance criteria when it comes to the carbon footprint of the application. **They are responsible for defining the KPIs to be met when releasing and deploying the application.**

They will need to be able to report and be able to measure and track the emission of their projects over time. Carbon Footprint[1] is an example of a tool that gives the ability to measure, report, and ultimately provides insight on how to reduce your cloud carbon emissions.



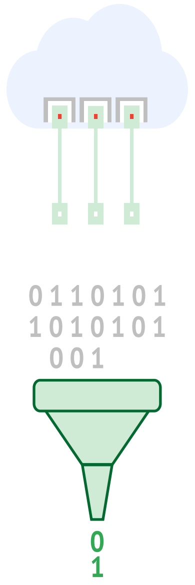
1 Google Cloud: Carbon Footprint tool

Programming language strategy

Complexity: High | Impact: Medium

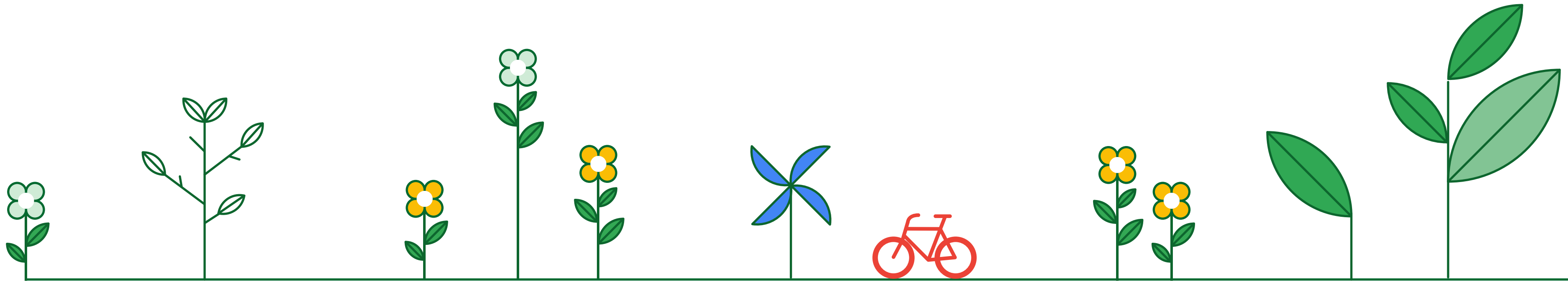
An architect should also consider the coding style adhered to by the development team.

Whilst different languages might be more optimal from a compute perspective, utilizing a “Lean Coding” approach forces the developer to use the minimum amount of processing required to deliver the application. **The developer is responsible for refactoring redundant code that is not embedded into the application through unused libraries.** The developer should ensure care is considered regarding wasteful loops that process data unnecessarily and are overly broad in their scope. Linting scripts can be deployed into the CI/CD pipeline to detect verbose code, loops that may not terminate and unused libraries.



Over the last few years I have spoken to far more early stage companies who want to ensure they are building their company and products in a sustainable way. Working with the developers we are able to share best practices at inception and ideation stages.

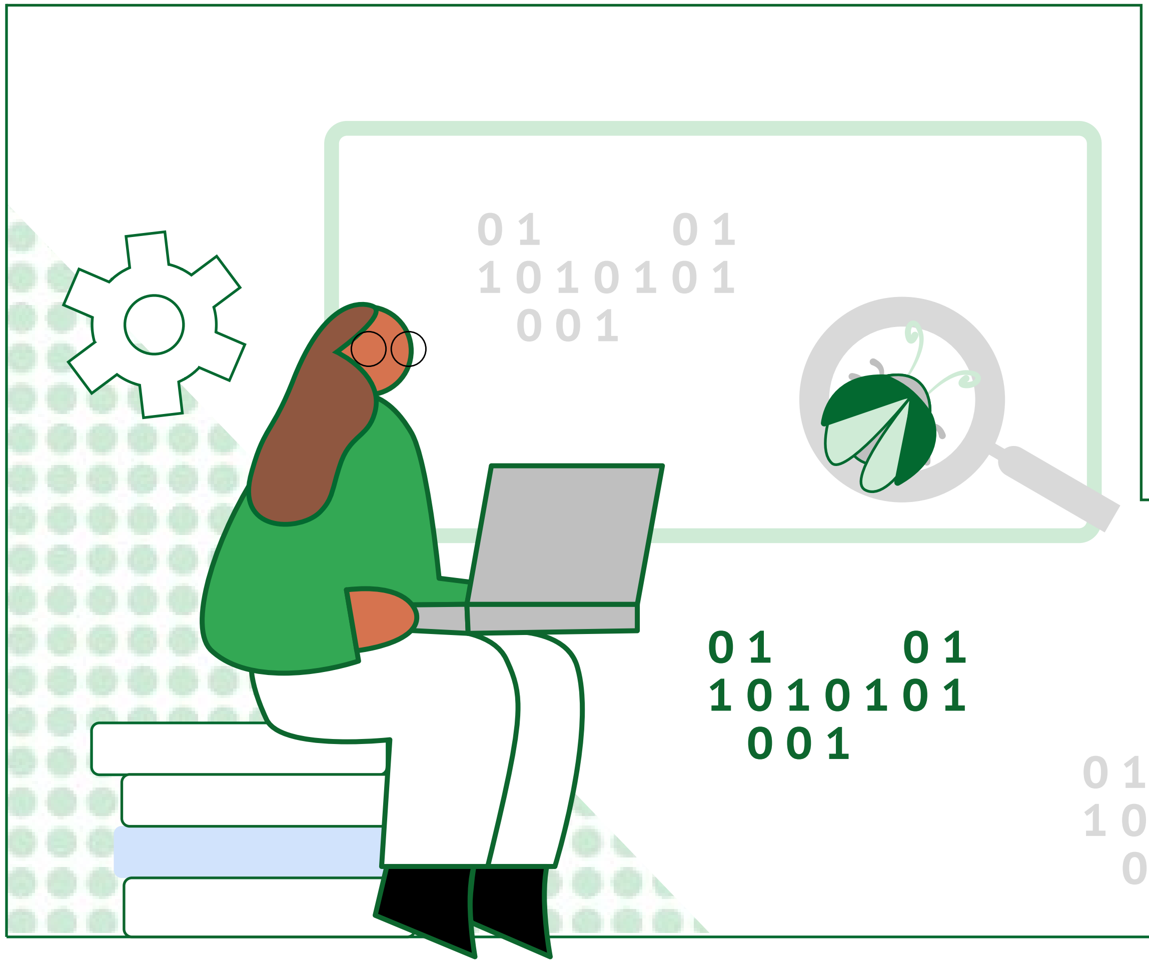
Neil Lock
Customer Engineering Manager, UK Digital Natives and Startups



Google Cloud

cloud.google.com

Go Green Software



A guide for DevOps and Site Reliability Engineers

Google Cloud

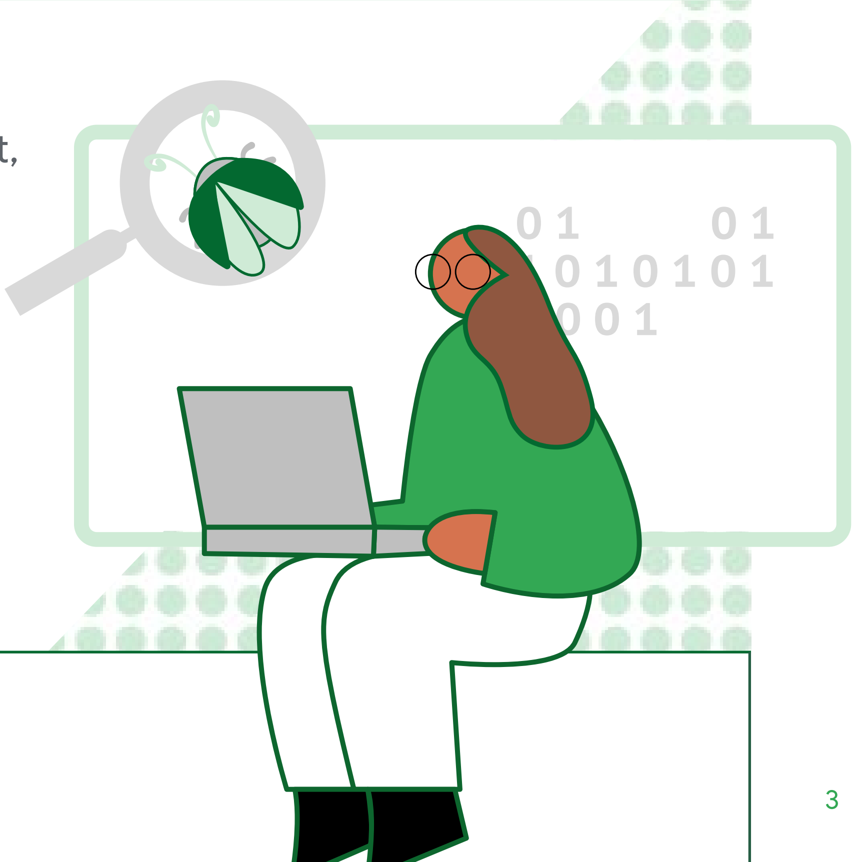
Contents

Introduction

As soon as the software has been coded, tested, built and packaged it is ready for development and execution in the cloud.

This involves many tasks that happen continuously as new software versions are shipped, such as continuous deployment, A/B testings or canary approaches, efficient execution, and real-time monitoring and resolution of issues and outages.

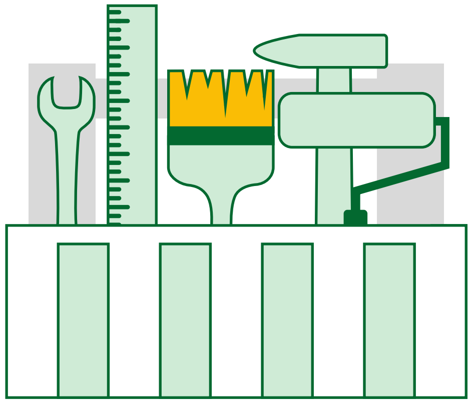
DevOps Engineers and Site Reliability Engineers (SREs), as Google calls them, typically perform these tasks. These two roles have similar concepts and cover roughly the same tasks described above. Both roles are closely aligned with software engineering.



Today, instead of simply throwing software over the fence to the operations teams, these teams get involved early and can influence the software itself. Google aims for SREs to have engineering backgrounds so they can work on code that can simplify operating the software with high reliability and availability. They may also be tasked with building tools to simplify the entire operations process.

This aspect of their work is also essential to greener operations. **Today, it takes motivation, knowledge and technical skills to implement strategies and mechanisms to run software more sustainably in the cloud.** SREs and DevOps Engineers need to be engineers interested in implementing tools and automations to achieve these gains.

Running and operating the software is one of the major sources of CO_2 emissions. The main driver is the electricity the application consumes, but the other factors such as hardware and cooling also play a role. Operations teams in the cloud have a reasonable toolbox that they can use to deploy and run a given software, with a given resource consumption profile, in a greener manner. DevOps or Site Reliability Engineers are also in the position of measuring and analyzing the data about CO_2 emissions caused by their overall operations. Both together (ie the measurement and the optimization towards more sustainable software operations) is what we at Google see as the core of **GreenOps.** Let's look at this toolbox.

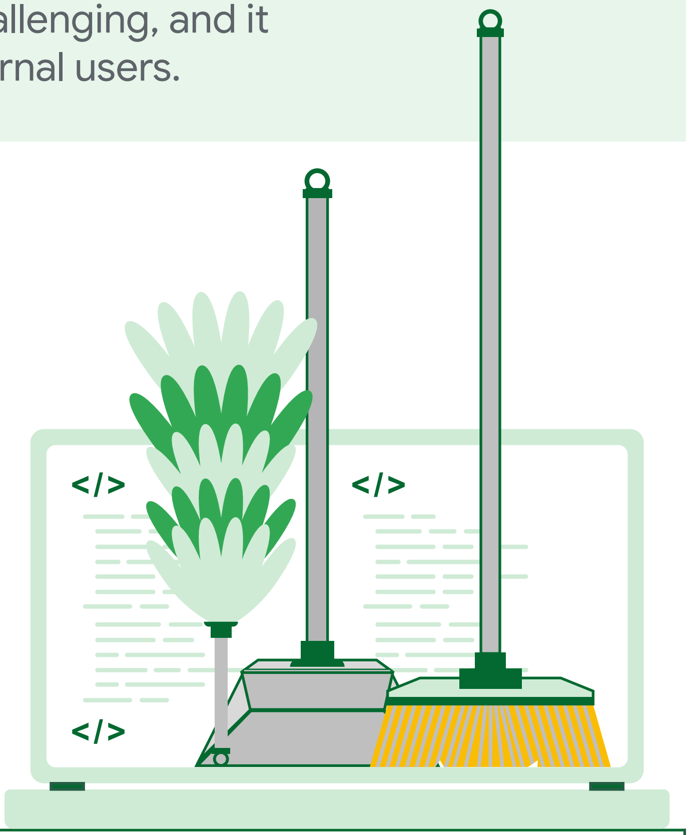


Cleaning up unused resources

Complexity: Easy | Impact: Medium | Scope: 20 - 30% of workloads

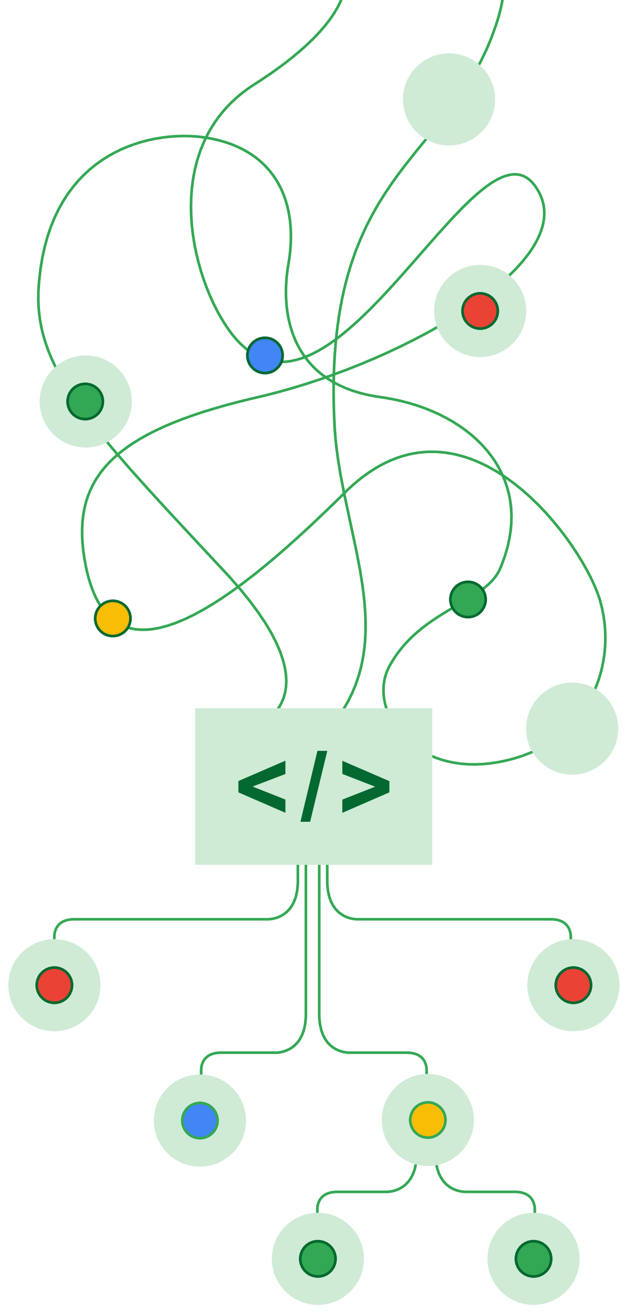
The easiest starting point for GreenOps is often cleaning up unused resources. In the cloud, where every resource is consumed with a single one click, it's easily for resources to go unused. Identifying these components can be challenging, and it can be difficult to determine if they are needed by external or internal users.

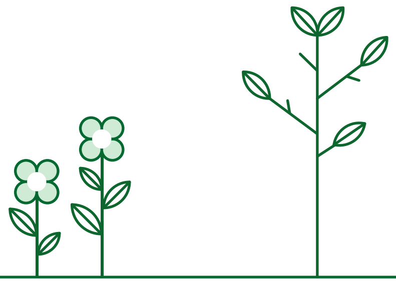
An example of a tool that can be used to help with cleaning up unused resources is Google Cloud's unattended project recommender[1] that identifies projects with low usage over the last 90 days. For a more detailed analysis, custom tools can be built using existing data sources in Google Cloud. The asset inventory API[2] allows users to export all resources in Google Cloud to a BigQuery dataset. These resources can be combined with either audit logs, which list the operations performed on a resource (e.g. start, configure, deploy) or monitoring metrics, which show the usage of a given resource.



Step by step guide

1. **Check the unattended project recommender**
2. Gather and export the list of assets or resources to BigQuery
3. Gather more data about the usage of resources
4. Implement reports to identify unused resources in your organization
5. Inform involved stakeholders about the plan to retire resources
6. **Keep backups for data**
7. **Cleanup these resources**



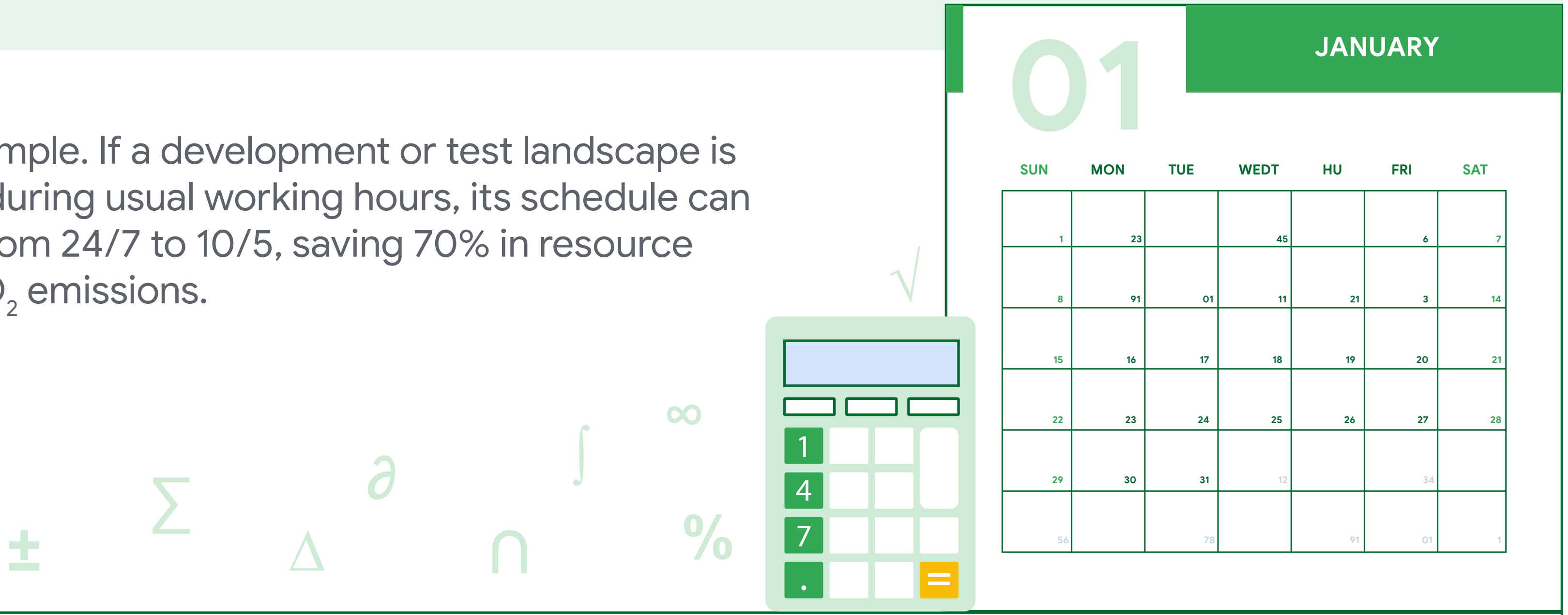


Resource scheduling

Complexity: Easy | Impact: Large | Scope: 30 - 50% of workloads

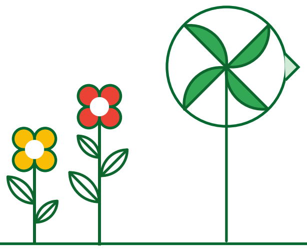
One of the benefits of Google Cloud is that it's easy and quick to start and stop resources, such as Kubernetes clusters, as needed. This means there is no need to occupy resources when you don't need them, which can save you money and reduce your carbon footprint. However, if IT organizations are coming from an on-premise world, they may see all compute resources as given and not consider scheduling their occupancy.

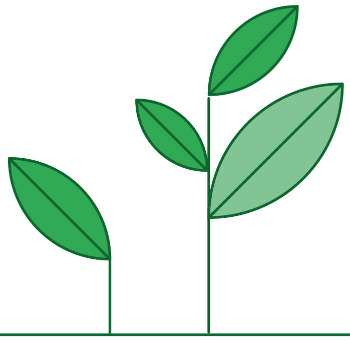
The math is simple. If a development or test landscape is only needed during usual working hours, its schedule can be reduced from 24/7 to 10/5, saving 70% in resource usage and CO_2 emissions.



Depending on the Google Cloud services used to run these applications, there are several ways to easily schedule resource usage:

- Combine Cloud Functions with Cloud Scheduler to run commands to start and stop other resources.
- Use Compute Engine's automatic schedule feature to define CRON-based patterns to start and stop instances[3]
- For Google Kubernetes Engine (GKE), you can either remove and re-provision an entire Kubernetes cluster, use more advanced approaches to configure GKE for "scale to almost zero"[4]
- Infrastructure-as-code is a great way to enable this flexibility. For example, you can use Terraform to destroy an entire environment after 6pm and re-apply it the next morning.
- When working with serverless options like Cloud Run, you can usually configure "scale to zero instanced" to perform this scheduling.

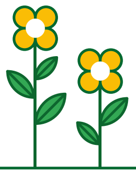


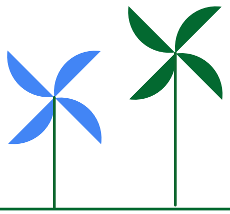


In addition to the scheduling, your users (most likely internal users such as engineers) could benefit from advanced features that give them control and consistency. A self-service portal allows users to delay a scheduled shutdown or request a restart.

Step by step guide

1. **Identify environments and software parts that are not required to run 24/7.**
2. **Define the strategy and mechanisms to schedule the involved cloud services.**
3. **Implement optimization and measure impact.**



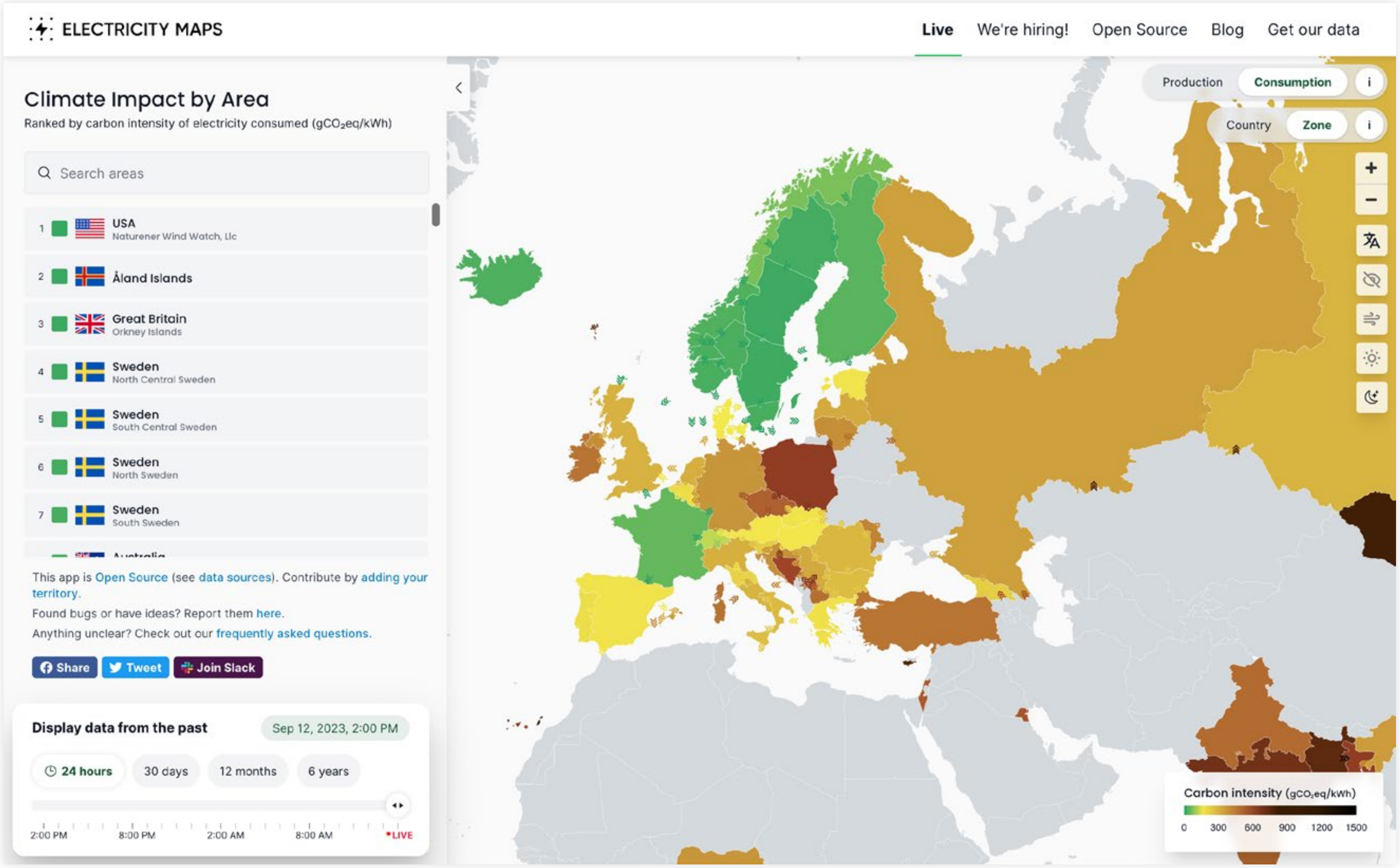


Region movement

Complexity: Medium | Impact: Large | Scope: 40 - 80% of workloads

In Google Cloud, choosing where to deploy your software is as easy as selecting a region from a dropdown menu. However, this decision can have a large impact on your carbon footprint.

Electricity grids and their carbon intensity vary between countries and regions, and the mix of energy is influenced by many factors. Watching these grids, such as on app.electricitymaps.com, can help you understand the differences and see which regions are using more sustainable energy sources like wind, solar and hydro.



Climate Impact by Area

Ranked by carbon intensity of electricity consumed (gCO_2eq/kWh)

1	🟩	USA	Naturener Wind Watch, Llc
2	🟩	Åland Islands	
3	🟩	Great Britain	Orkney Islands
4	🟩	Sweden	North Central Sweden
5	🟩	Sweden	South Central Sweden
6	🟩	Sweden	North Sweden
7	🟩	Sweden	South Sweden

This app is Open Source (see data sources). Contribute by adding your territory.

Found bugs or have ideas? Report them here.

Anything unclear? Check out our frequently asked questions.

Display data from the past — Sep 12, 2023, 2:00 PM

Image: app.electricitymaps.com

Using this knowledge, operations experts can identify regions with a greener footprint to run their software in the most sustainable way. Google Cloud provides the necessary annual data to support this decision-making[5] process, as well as a tool called Region Picker[6].

While organizations should consider carbon emissions when choosing a region, it's important to note that there are other factors that can heavily influence the ability to move a workload:

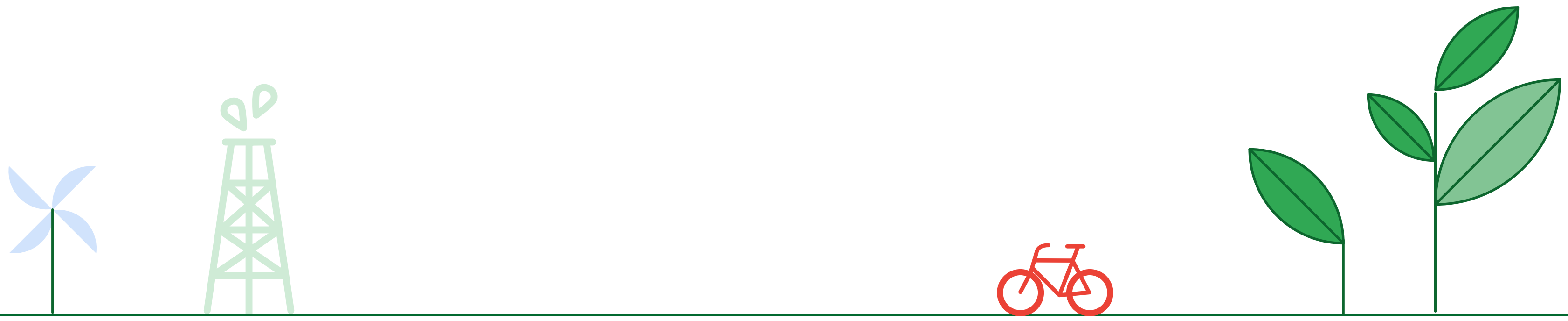
Outgoing traffic: Many software applications are consumed by users (internal or external), so you need to consider latency and data requirements when choosing a green region. To make a good decision, you need a rough understanding of how much data will travel over the network and what latency you need to achieve.

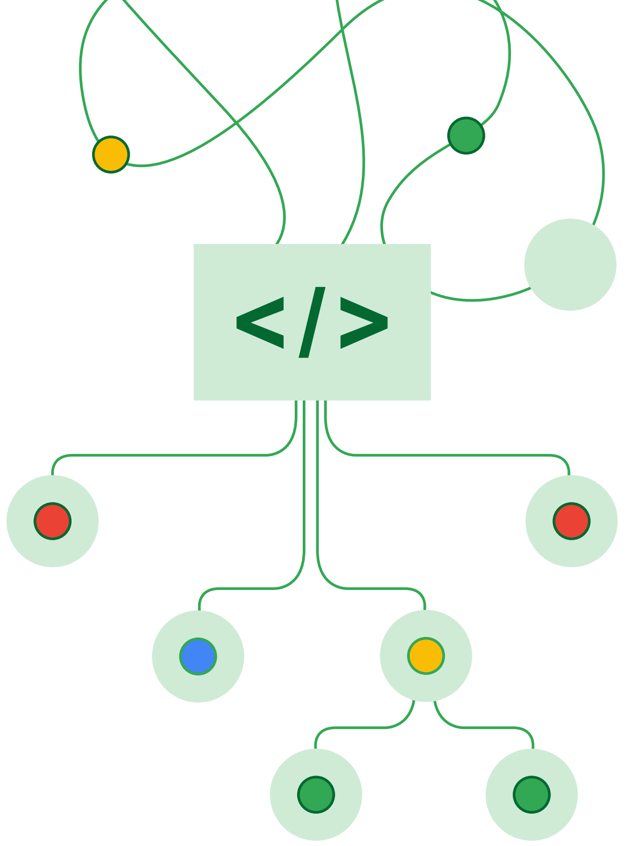
Ingoing traffic: Software modules no longer exist in isolation. They rely on other applications and services to function. For example, they may load data from somewhere or call other APIs. This is also important to estimate and evaluate when making a decision.

Data locality/privacy: Consider whether the data involved has strict data locality requirements. This may be due to data privacy regulations or because it is impractical to store a large amount of data in parallel in two locations.

Price: Google Cloud region prices typically vary. When moving to a more sustainable region, any of the following scenarios are possible: prices stay the same, prices decrease, or prices increase.

Consider these and other factors before moving an existing application to a different region. However, a simple region move is a powerful way to improve the carbon footprint of an application or entire system.





Step by step guide

1. **Identify software modules that could qualify for a move to another region based on their current size and their carbon footprint (if already measurable).**
2. Evaluate other criteria, like the outgoing and incoming traffic and the data locality needs. Use a simple classification like low, medium, high for now.
3. **Select one or two applications to get started with.**
4. Identify the Google region in your geography that ideally fits your needs.
5. Move these applications to a green region. Depending on the deployment environment it either can be moved (e.g. like in Google Compute Engine[7]) or redeployed.
6. **Measure the impact on carbon emissions but also all other metrics of the application that can be monitored, especially response latency, egress data, etc.**

Follow the sun and wind

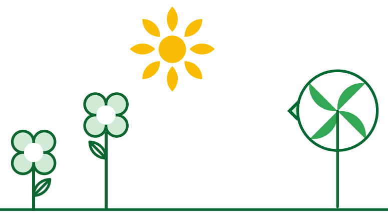
Complexity: Medium - High | Impact: Medium - Large | Scope: 20 - 30% of workloads

As we learned earlier, one of the major factors that influences an application's footprint is the grid's carbon intensity. While moving an application to a different region can optimize it in the long term, there are also opportunities to further improve it in the short term. **The idea of "follow the sun and wind" is to move workloads to times and places where there is a lot of renewable energy.**

This can be a complex approach that requires accurate real-time data, but it can also be as simple as running batch jobs at noon, when the sun's capacity is highest.

One starting point for this practice is batch jobs, which are software modules that can perform a task relatively independent of an end user need. **Batch jobs typically process data or execute individual steps as part of a regular task. For batch jobs, there are two variables that you can control:**

- The time by which the job must be started and finished
- The region in which the job is run.



For other services, APIs and applications, optimization is more complex and requires more data. These software pieces have to run all the time and serve end-user traffic, so the only dimension for improvement is the region. An optimized plan could be to deploy them in the morning to Switzerland (because hydro storages are full), move them during noon to Madrid (because sun is shining) and move them in the evening to Finland (where the wind is blowing).

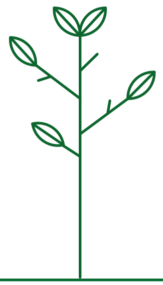
Step by step guide

1. Identify software modules that would qualify for a dynamic movement in time and place.

2. Evaluate them against criteria like the incoming and outgoing traffic, data locality, time criticality and others to identify the most suitable ones.

3. Select one or two examples to get started with.

4. Specify the strategy and its complexity:
   Level 1: Schedule and deploy the job-alike software based on a rough data foundation, like yearly averages of the hourly energy mix in certain regions.
   Level 2: Gather more detailed hourly data to determine the right times and places to deploy and run these jobs.
   Level 3: Leverage the detailed data to determine a strategy for end user facing APIs and applications.



Rightsizing

Complexity: Low - Medium | Impact: Medium - Large | Scope: 30 - 60% of workloads

One of the green software development practices is for **software engineers to be fully aware of the resource consumption profile of their software** and to optimize and document that profile. This is mostly focused on CPU and memory resources.

If this practice is not in place, operations teams have to run software that might use resources unevenly or request too many resources. This can happen with software that runs on pure VMs, but also with K8s-based workloads or even serveless solutions. Let’s dive into the details of these three areas:

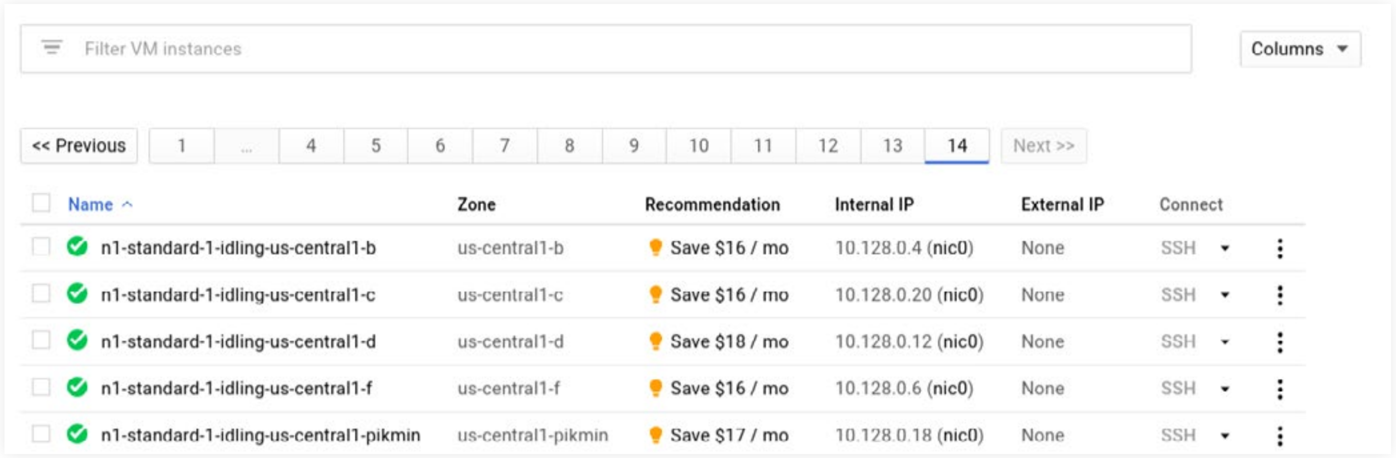
For VM-based workloads: Operations teams (in alignment with the engineering team) have to select upfront a size of the VM to run the software upfront. With Google Cloud, this can be done by selecting a machine family, which is a curated set of processor and hardware configurations optimized for specific workloads, and a machine type, which is typically a predefined configuration of CPU and memory. Additionally, Google Cloud offers custom machine types, which allow a flexible selection of CPUs and memory. Based on these options, the team decides what size to choose. If the profile of the application is unknown, or very spiky, then the VM might be chosen too big, and a lot of unused resources will be occupied.

For K8s-based workloads: K8s experts or software engineers typically specify the resource needs of a container in the K8s YAML configuration. Two data points must be provided: resource requests and limits. The first one specifies the lower limit, or the minimum amount of resources while the second specifies the upper limit[8]. Both can be specified for CPU and memory. If this specification is inaccurate, missing, or too high, then resources will again be used inefficiently.

For serverless workloads: There is typically also the need to specify a rough profile of resource needs for serverless workloads. In Cloud Run, for example, the amount of vCPUs and GB memory can be configured freely. In Cloud Functions, only the GB memory has to be chosen.

All these scenarios demonstrate that if the resource profile of an application is not clear, operations teams have a hard time operating that piece of software efficiently and will likely use more resources than needed. This leads to the idea of rightsizing.

Rightsizing involves observing a workload over a longer period of time to retrospectively identify whether the amount of requested resources is too high. For VMs, Google Cloud already offers this as a built-in feature that highlights recommendations per VM[9].



Name	Zone	Recommendation	Internal IP	External IP	Connect
✓ n1-standard-1-idling-us-central1-b	us-central1-b	Save $16 / mo	10.128.0.4 (nic0)	None	SSH
✓ n1-standard-1-idling-us-central1-c	us-central1-c	Save $16 / mo	10.128.0.20 (nic0)	None	SSH
✓ n1-standard-1-idling-us-central1-d	us-central1-d	Save $18 / mo	10.128.0.12 (nic0)	None	SSH
✓ n1-standard-1-idling-us-central1-f	us-central1-f	Save $16 / mo	10.128.0.6 (nic0)	None	SSH
✓ n1-standard-1-idling-us-central1-pikmin	us-central1-pikmin	Save $17 / mo	10.128.0.18 (nic0)	None	SSH

Rightsizing recommendation for VMs in Google Cloud

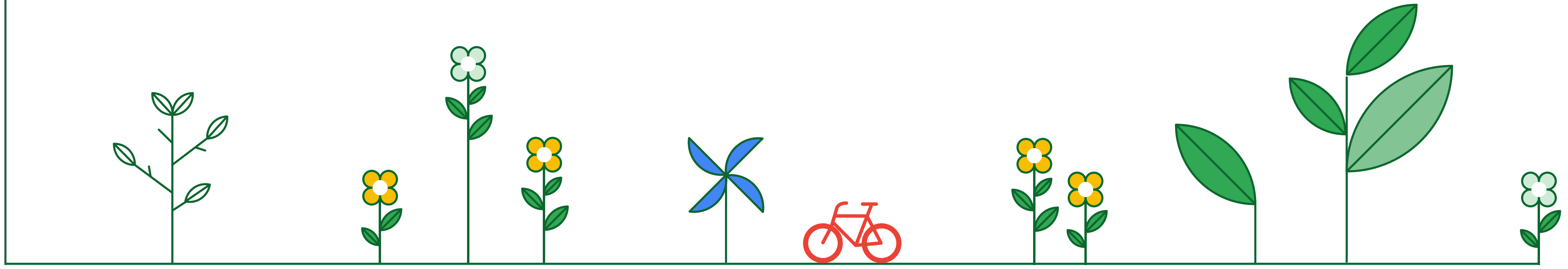
For other workloads (K8s, serverless and more) this needs to be implemented as part of monitoring the software. **Cloud Monitoring provides a foundation for this by automatically gathering monitoring metrics about CPU/memory limit and request utilization from containers running on Google Kubernetes Engine**.

Step by Step Guide

1. Check the rightsizing recommendation of VMs.
2. Configure monitoring dashboards to observe the resource utilization over time.
3. Implement regular reviews with the engineering teams to evaluate the resource requirements of a software modules.
4. Adapt the configuration and realize improvements.
5. Measure and communicate the improvements in CO_2 emissions.

More businesses are interested in building and re-building sustainably due to various external and internal pressures. I've introduced the concept of Green Software principles to numerous DevOps teams of multinational businesses that have their applications on Google Cloud. From explaining the basic terms such as what carbon intensity is to how to measure and then optimizing applications for low carbon.

Khulan Davaajav
Solutions Consultant, Google Cloud



Footnotes

1 How ML-fueled recommendations help developers optimize security, price-performance, and carbon reduction, Google blog, October 2021

2 Introduction to Cloud Asset Inventory, Google Cloud product document

3 Scheduling a VM instance to start and stop, Google Cloud Compute Engine guide

4 Scale your kubernetes cluster to (almost) zero with GKE autoscaler, Medium, Alfanso Palacios, December 2019

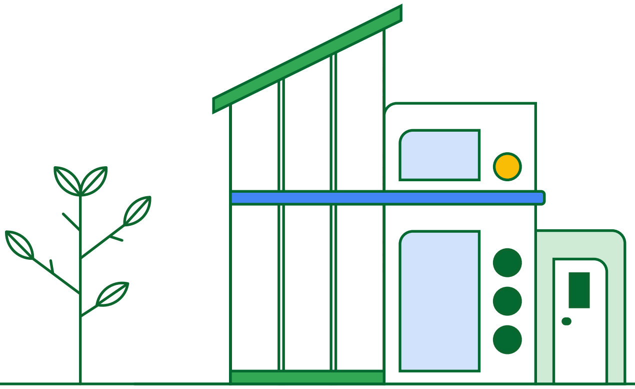
5 Carbon free energy for Google Cloud regions, Google Cloud

6 Google Cloud Region Picker, Google Cloud tool

7 Move a VM instance between zones or regions, Google Cloud Compute Engine guide

8 Resource Management for Pods and Containers, Kubernetes configuration resource

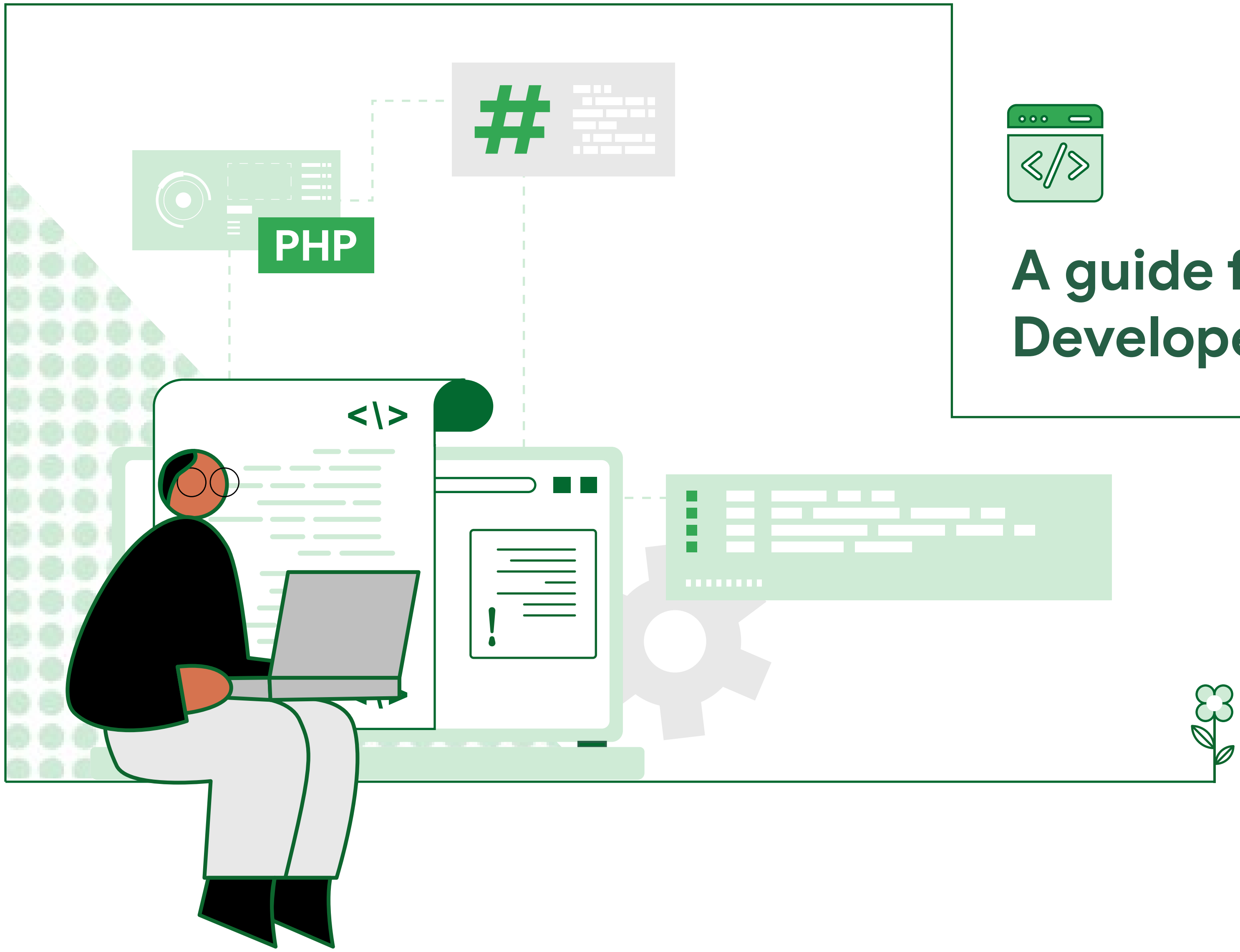
9 Save money, improve performance with new VM Rightsizing Recommendations, blog, Google Cloud, July 2016



Google Cloud

cloud.google.com

Go Green Software



A guide for Software Developers/Engineers

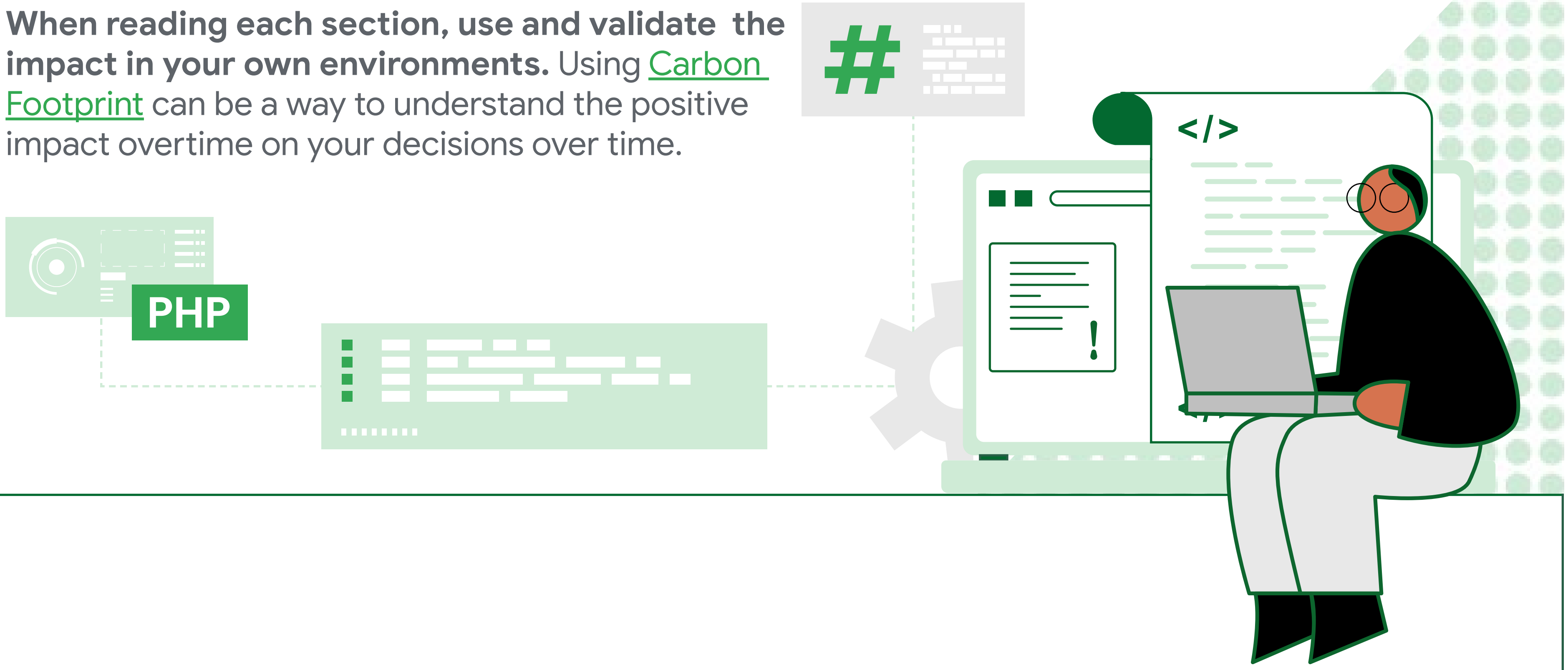
Google Cloud

Contents

Introduction

Software Engineers and Developers have the ability to reduce carbon emissions through a number of practical steps. This section covers four domains which guide developers on use of resources, the developer experience, daily activities and observations in production.

When reading each section, use and validate the impact in your own environments. Using Carbon Footprint can be a way to understand the positive impact overtime on your decisions over time.



Use cloud resources to reduce CO_2e

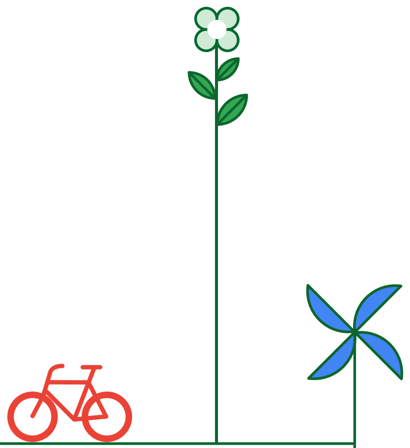
Complexity: Easy | Impact: Large

Every business aims to reduce the use of resources, as it has the potential to reduce emissions and costs. **These actions can bring long lasting benefits and value to a business, which goes way beyond just the software development.**

An important principle is to make the most of the infrastructure being used. Even when compute and storage are operationally active, free space in resources will still have a carbon footprint. Ensuring that each area is reviewed and inactive usage areas are removed is considered a key aspect of software development.

- **Development environment:** Developers can set up their development experience within a cloud to run Integrated Development Environments (IDE) and other tools, enabling the utilization for Carbon Free Energy (CFE) regions. An example of a unique approach to reducing carbon emissions is provided by the combination of a Chromebook and Cloud Workstation.
- **Physical resources:** In any development or project, the focus should be on how physical resources are maximized, especially when it comes to compute and storage. For example, Tensor Processing Units (TPUs) can be used for specific machine learning processes.

- **Limiting space:** When parameters for resource profiles are being defined, it's essential to think about "unused space", particularly in deployment models such as containers or virtual environments.
- **Resource vs concentration risk:** A careful balance is struck between maximizing utilization and concentration risk. However, making the most of a single resource will be more sustainable when combined with high availability architectures.
- **Engineered stack:** Many examples of services that enhance utilization and reduce the usage of resources like energy can be found within Google. Some instances include Tensor Processing Units (TPUs), which are optimized hardware modules for AI workloads, BigQuery has separate resource pools for compute and storage that provides dynamic workload handling. Other Common services that help decrease emissions are AlloyDB, which scales PostgreSQL workloads, reducing processing time and resources, and Cloud Run which automatically halts services when they're not in use. It is crucial for developers or engineers to seek out services designed with a lower carbon footprint as the default through integrated design. Reviewing suitable cloud services is an important area for developers and engineers to explore.
- **Usage of spot VM instances:** Preemptive instances are a great way to reduce costs and use less resources for non-production workloads. Google has researched how to optimize the usage of physical resources with preemptive instances, which can help reduce capacity. This is defined in the paper "Machine Learning Applications for Data Center Optimisation[4]"
- **Fetch only what is required:** When working with large datasets, use features like partition tables to access smaller subsets of data for processing or to transform data during ingestion via pipelines.



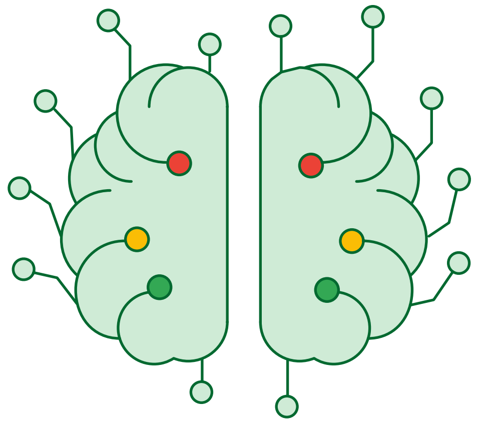
Software considerations during lifetime of development or engineering

Complexity: Easy | Impact: Large

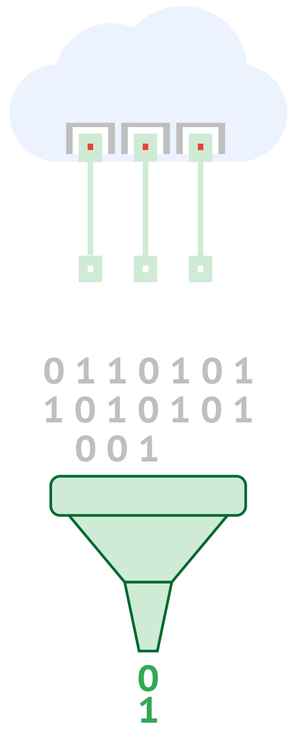
Developers use libraries, extensions, and content management systems to build solutions. **It is not uncommon for software to serve a mature enterprise for a decade or more.**

During this lifecycle, engineers and developers are constantly adapting and changing the environment. Considering the following approaches can provide opportunities to reduce CO_2e.

- **Machine Learning:** When developing machine learning, it is critical to keep the footprint of data and code required to a minimum. An recent paper "Carbon Emissions and Large Neural Network Training[5]", provides valuable insights into the impact on carbon footprint. Here are a few ways to reduce your carbon footprint when developing machine learning models: Use only the data and code that is needed, place machine learning workloads in regions with higher carbon-free energy, and limit the number of cycles in training the models.



- **Remove unused code:** Over time, as code is developed and changed, research has shown that up to 45% of features in legacy applications are never used. There are open source tools that can validate library and code usage, such as pyflakes and pylint for Python. Once unused code is identified, a backlog can be developed to remove it when the next functional change occurs, which can help reduce CO2 emissions. For example, Node.js installations often include many duplicate dependencies that can be removed.
- **Libraries with high overhead can be identified by tracing execution and understanding execution time.** This will help to find libraries that are slowing down the applications, and alternatives can be seen if they can be used. For example, in Python, trace modules can be used to investigate execution time. Resource usage is affected by any execution, so reducing the overhead of libraries can also help to reduce overall resource usage.
- Data access with a cursor for loop can be inefficient. **It is always more efficient to let the database engine handle as much of the database interaction as possible.** Consider the style of code used for database library execution. If large arrays of data are being fetched in execution loops, consider processing the data closer to its resting place.
- **Consideration for API:** One concept to consider is putting the CO2 emissions of an API call in the invoking function return. In a DevOps ecosystem, API discoverability and addressability are essential, so why not also provide CO2 emissions on execution?



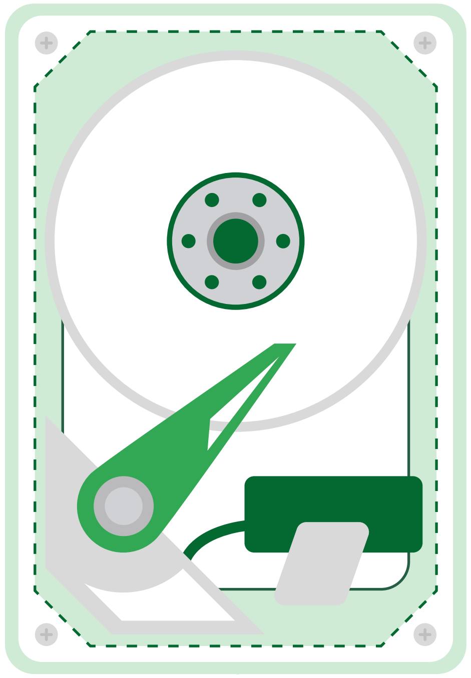
Developer daily activities

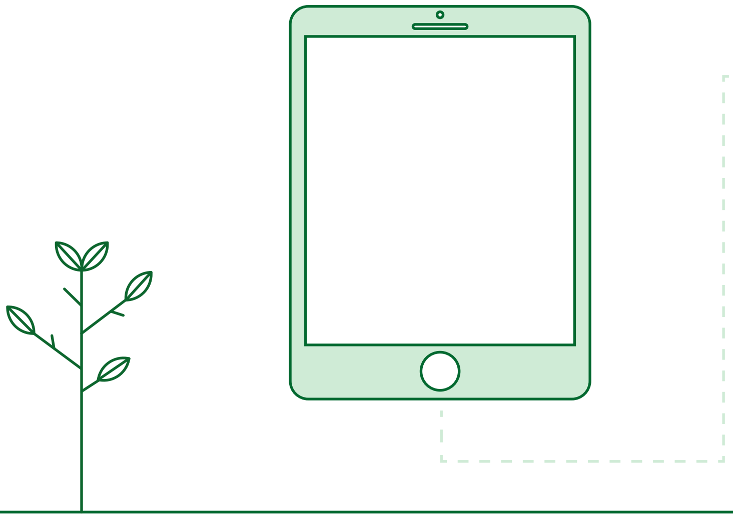
Complexity: Easy | Impact: Large

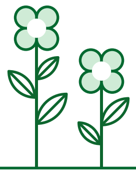
Improving developer ergonomics when building software or solutions can be a low-effort way to reduce CO_2 emissions. Here are some easy areas to investigate.

- **Use laptops for development:** Today, it is possible to develop successfully on a laptop, even a chromebook with Linux Beta. This gives developers more mobility and reduces the number of machines needed. Research has shown a laptop consumes 20-50 watts, while a desktop consumes 80-150 watts.
- **Turning off[7] hardware:** Use it or snooze it! This is especially important in a hybrid work environment, where technology can be left on at both home and the office.
- **Widescreen vs multi-screen:** The modern ultra widescreen can provide the same developer experience as multi-screen, so consider reducing energy by using a single ultra widescreen. For example, the Dell P3421W has a lifetime CO2 footprint of 688 kg, while two Dell P2423 monitors have a lifetime CO2 footprint of 1170 kg.

- **Extend +1 year:** Extending your equipment by an additional year or simply reducing the refresh point for developer workstation can reduce your carbon footprint. Embodied carbon in manufacturing can account for approximately 80% of a device's lifetime carbon emissions. Using an external keyboard and mouse with a laptop can extend the life of a laptop by reducing physical wear. In some cases, an earlier refresh is a good way to reduce emissions if the new device has a lower CO_2 emissions.
- **Recycle workstations, laptops, and other equipment at end of life to contribute to a circular economy and reduce mineral extraction:** Google developed circular economy principles for data centers and many aspects can be applied to the developer local ecosystem.







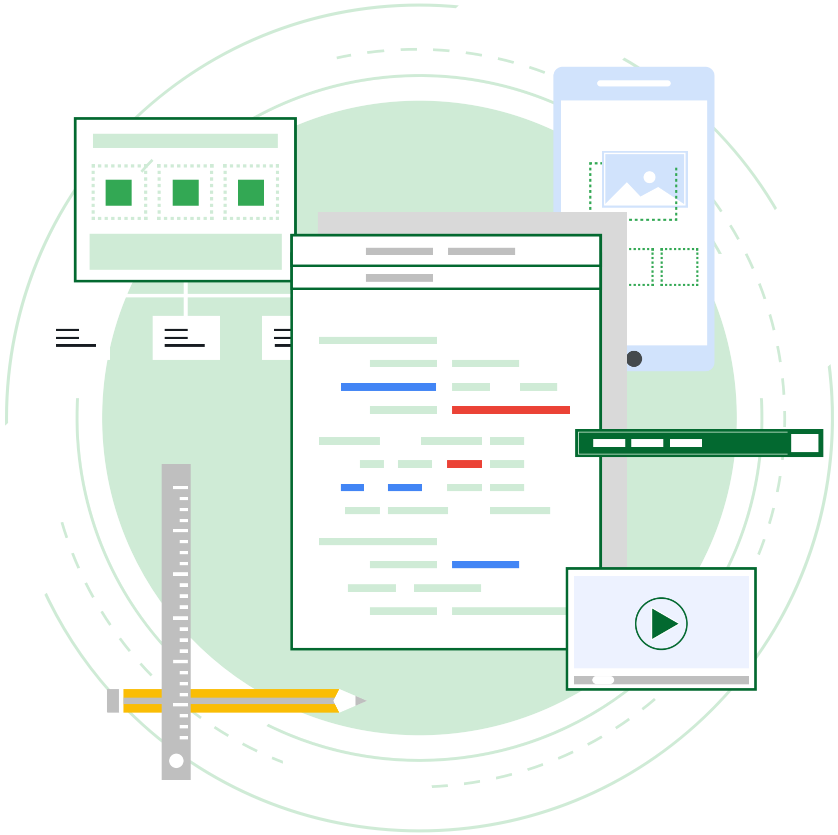
Engineer/Developer observation

Complexity: Medium | Impact: Large

Monitoring and sustainability are just as equally important as any form of operational inspection. When using CI/CD pipelines, maintenance needs to be considered to reduce carbon footprint.

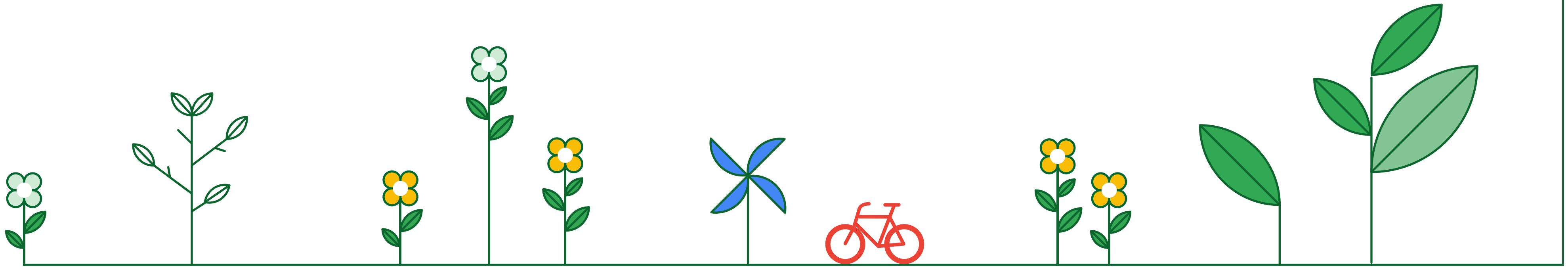
- **Define Sustainability Budgets:** Site Reliability Engineers have error budgets to monitor breaches and act accordingly. There is no reason why we can't set carbon footprint budgets for development teams for the end-to-end solutions, provide companies with visibility into the carbon usage of projects, and monitor across all solutions.
- **Constant reviews and observing carbon footprint is essential:** Sustainability monitoring should be equal to other operational aspects, and siloed thinking should be avoided.
- **Batch processing:** When performing batch processing, consider choosing a time when the carbon intensity is lowest, such as early morning or evening, to reduce the compute and data pipeline carbon footprint.

- **Developer release strategy:** Observe how developers are using the CI/CD pipelines. If constant retesting and deployment is increasing your carbon footprint, balance releasing with functional needs, especially in early development and testing phases.
  - **Maintain and prolong:** Maintain an optimized code during application lifecycle. Carefully consider embodied carbon and technological improvements to determine whether to refresh or extend the refresh of technology.
  - **Refurbish and remanufacture:** Consider the benefits of refurbishing code. Create a sustainability backlog to remove unused code.
  - **Reuse and redistribute:** Use APIs and data catalogs to promote a culture of reuse. Reuse is important because it reduces additional testing and development, as well as the number of components in production. All of these factors contribute to a lower carbon footprint.
  - **Recycle:** Recycle code with a low carbon footprint across different projects. Consider introducing sustainability testing in your development cycles or CI/CD pipelines to monitor improvements, especially for APIs.



The intersection between sustainability and technology has been increasingly top of mind for many organisations we've worked with recently. Not only has cloud raised awareness of the carbon emissions related to infrastructure usage, but has also enabled Software Engineers to leverage cloud-native principles and features like spot VMs to develop in a green way.

Nikolai Donko
Technical Account Manager, Google Cloud



Footnotes

1 Google Cloud: Cloud Workstations

2 How we're minimizing AI's carbon footprint, Google blog, Apr 2021

3 Energy and Carbon-Efficient Placement of Virtual Machines in Distributed Cloud Data Centers, Conference Paper from Proceedings of the 19th international conference on Parallel Processing, ResearchGate, August 2013

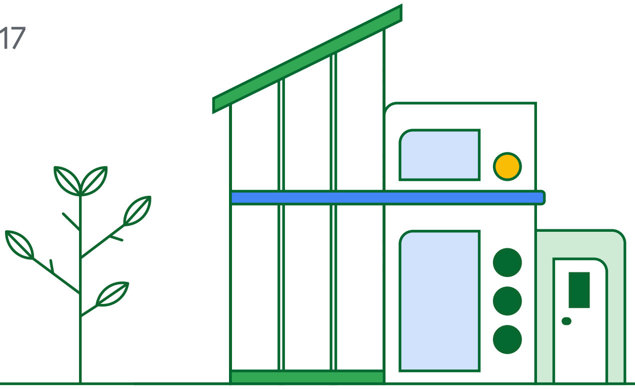
4 Machine Learning Applications for Data Center Optimization, Jim Gao, Google

5 The Carbon Footprint of Machine Learning Training Will Plateau, Then Shrink, Technology Predictions publication, February 2013

6 How Much Does Unused Code Matter for Maintenance? Research Paper, Sebastian Eder, Maximilian Junker, Elmar Jurgens, Benedikt Hauptmann, Institute of Information at Munich University, Germany, 2012

7 Will Turning Off My Monitor Save Energy? Sciencing blog, April 2017

8 How maintenance windows affect your error budget—SRE tips, Google blog, June 2020



Google Cloud

cloud.google.com