
How Kubernetes Optimization can combat climate change
Annie Talvasto

Annie Talvasto

- Sr. Product Marketing Manager at Camunda

- CNCF Ambassador
- Azure MVP

- Kubernetes & CNCF meetup co-organizer
- Startup coach
- Co-host of Cloudgossip podcast - cloudgossip.net

Welcome to the session!

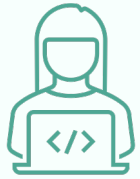
What will you learn?

Agenda

- Introduction
- Principles of Green Software
- Principles in Practice: Microservices
- CNCF Open Source Projects
- Optimization Example: Cost
- Resources

People care about sustainability – but what about at work?

Sustainability & software development



ICT industry consistently adds 2 to 6% of emissions each year since 2007 – the same as airline industry.

Modern technologies require more and more compute power.

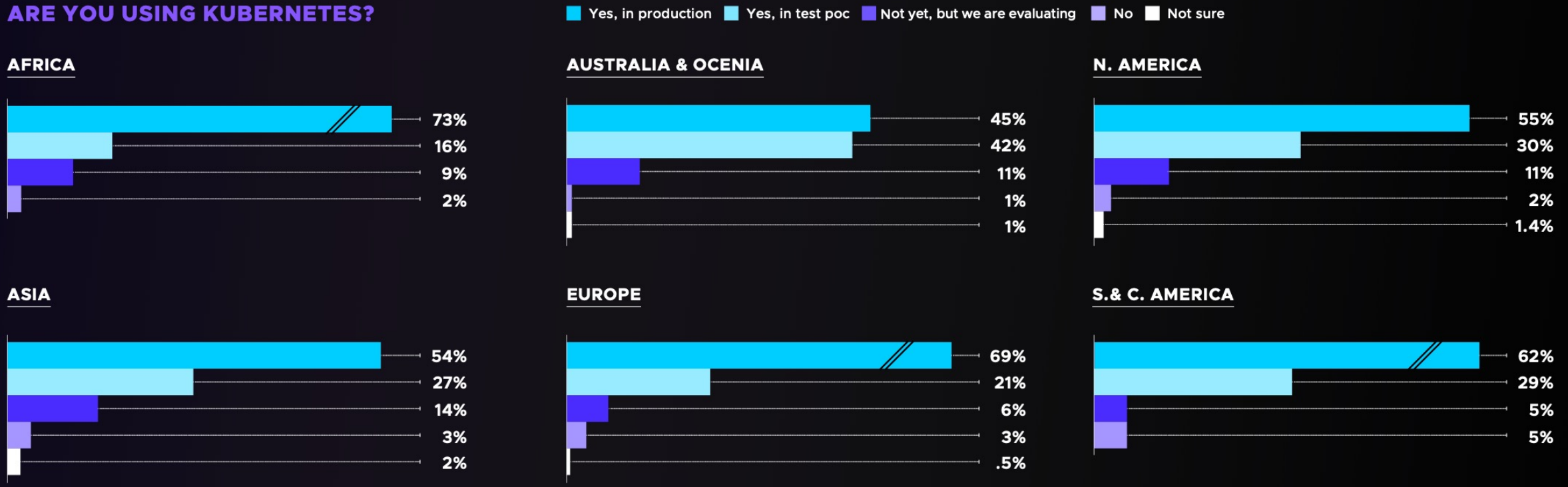
World is going to experience irreversible changes if climate change remains unchecked.

KUBERNETES HAS CROSSED THE ADOPTION CHASM TO BECOME A MAINSTREAM GLOBAL TECHNOLOGY

According to CNCF's respondents, 96% of organizations are either using or evaluating Kubernetes – a record high since our surveys began in 2016. Particularly interesting is the regional adoption of Kubernetes in production, with emerging technology hub Africa (73%) jumping ahead of other more established tech centers including Europe (69%) and North America (55%). Additionally, 93% of respondents are currently using, or planning to use, containers in production, echoing 92% in our **2020 survey**.

96% OF ORGANIZATIONS ARE EITHER USING OR EVALUATING KUBERNETES

ARE YOU USING KUBERNETES?



Search

- Home
- Getting started
- Concepts
  - Overview
  - Cluster Architecture
  - Containers
  - Windows in Kubernetes
  - Workloads
  - Services, Load Balancing, and Networking
  - Storage
  - Configuration
    - **Configuration Best Practices**
    - ConfigMaps
    - Secrets
    - Resource Management for Pods and Containers

Configuration Best Practices

This document highlights and consolidates configuration best practices that are introduced throughout the user guide, Getting Started documentation, and examples.

This is a living document. If you think of something that is not on this list but might be useful to others, please don't hesitate to file an issue or submit a PR.

General Configuration Tips

- When defining configurations, specify the latest stable API version.
- Configuration files should be stored in version control before being pushed to the cluster. This allows you to quickly roll back a configuration change if necessary. It also aids cluster re-creation and restoration.
- Write your configuration files using YAML rather than JSON. Though these formats can be used interchangeably in almost all scenarios, YAML tends to be more user-friendly.
- Group related objects into a single file whenever it makes sense. One file is often easier to manage than several. See the guestbook-all-in-one.yaml file as an example of this syntax.
- Note also that many `kubectl` commands can be called on a directory. For example, you can call `kubectl apply` on a directory of config files.
- Don't specify default values unnecessarily: simple, minimal configuration will make errors less likely.

Edit this page
Create child pa
Create an issu
Print entire se

General Configurat
"Naked" Pods versu
Deployments, and
Services
Using Labels
Using kubectl

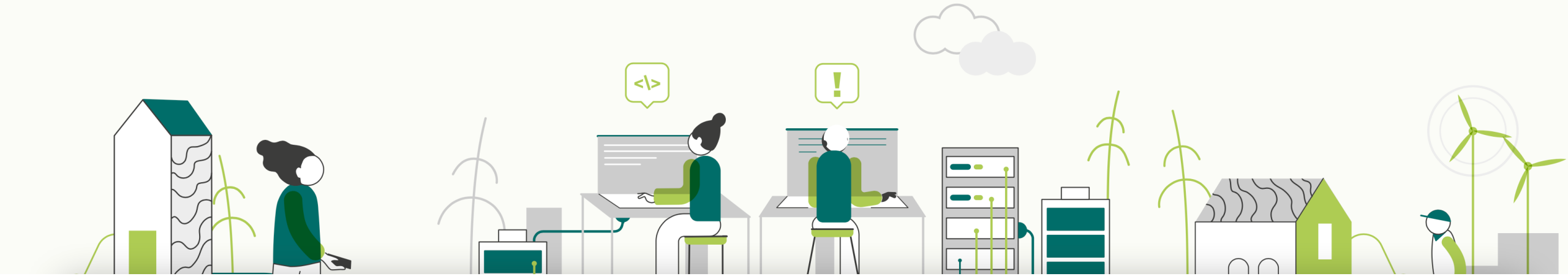
Principles of Green Software

We are building a trusted ecosystem of people, standards, tooling and best practices for

GREEN SOFTWARE

Sign up to our newsletter...

Sign up



Principles of Green software

- Carbon: Build applications that are carbon efficient.

Principles of Green software

- Carbon: Build applications that are carbon efficient.
- Electricity: Build applications that are energy efficient.

Principles of Green software

- Carbon: Build applications that are carbon efficient.
- Electricity: Build applications that are energy efficient.
  - Carbon Intensity: Consume electricity with the lowest carbon intensity.

Principles of Green software

- Carbon: Build applications that are carbon efficient.
- Electricity: Build applications that are energy efficient.
- Carbon Intensity: Consume electricity with the lowest carbon intensity.
- Embodied Carbon: Build applications that are hardware efficient.

Principles of Green software

- Carbon: Build applications that are carbon efficient.
- Electricity: Build applications that are energy efficient.
- Carbon Intensity: Consume electricity with the lowest carbon intensity.
- Embodied Carbon: Build applications that are hardware efficient.
- Energy Proportionality: Maximize the energy efficiency of hardware.

Principles of Green software

- Carbon: Build applications that are carbon efficient.
- Electricity: Build applications that are energy efficient.
- Carbon Intensity: Consume electricity with the lowest carbon intensity.
- Embodied Carbon: Build applications that are hardware efficient.
- Energy Proportionality: Maximize the energy efficiency of hardware.
- Networking: Reduce the amount of data and distance it must travel across the network.

Principles of Green software

- Carbon: Build applications that are carbon efficient.
- Electricity: Build applications that are energy efficient.
- Carbon Intensity: Consume electricity with the lowest carbon intensity.
- Embodied Carbon: Build applications that are hardware efficient.
- Energy Proportionality: Maximize the energy efficiency of hardware.
- Networking: Reduce the amount of data and distance it must travel across the network.
- Demand Shaping: Build carbon-aware applications.

Principles of Green software

- Carbon: Build applications that are carbon efficient.
- Electricity: Build applications that are energy efficient.
- Carbon Intensity: Consume electricity with the lowest carbon intensity.
- Embodied Carbon: Build applications that are hardware efficient.
- Energy Proportionality: Maximize the energy efficiency of hardware.
- Networking: Reduce the amount of data and distance it must travel across the network.
- Demand Shaping: Build carbon-aware applications.
- Measurement & Optimization: Focus on step-by-step optimizations that increase the overall carbon efficiency.

Principles of Green software

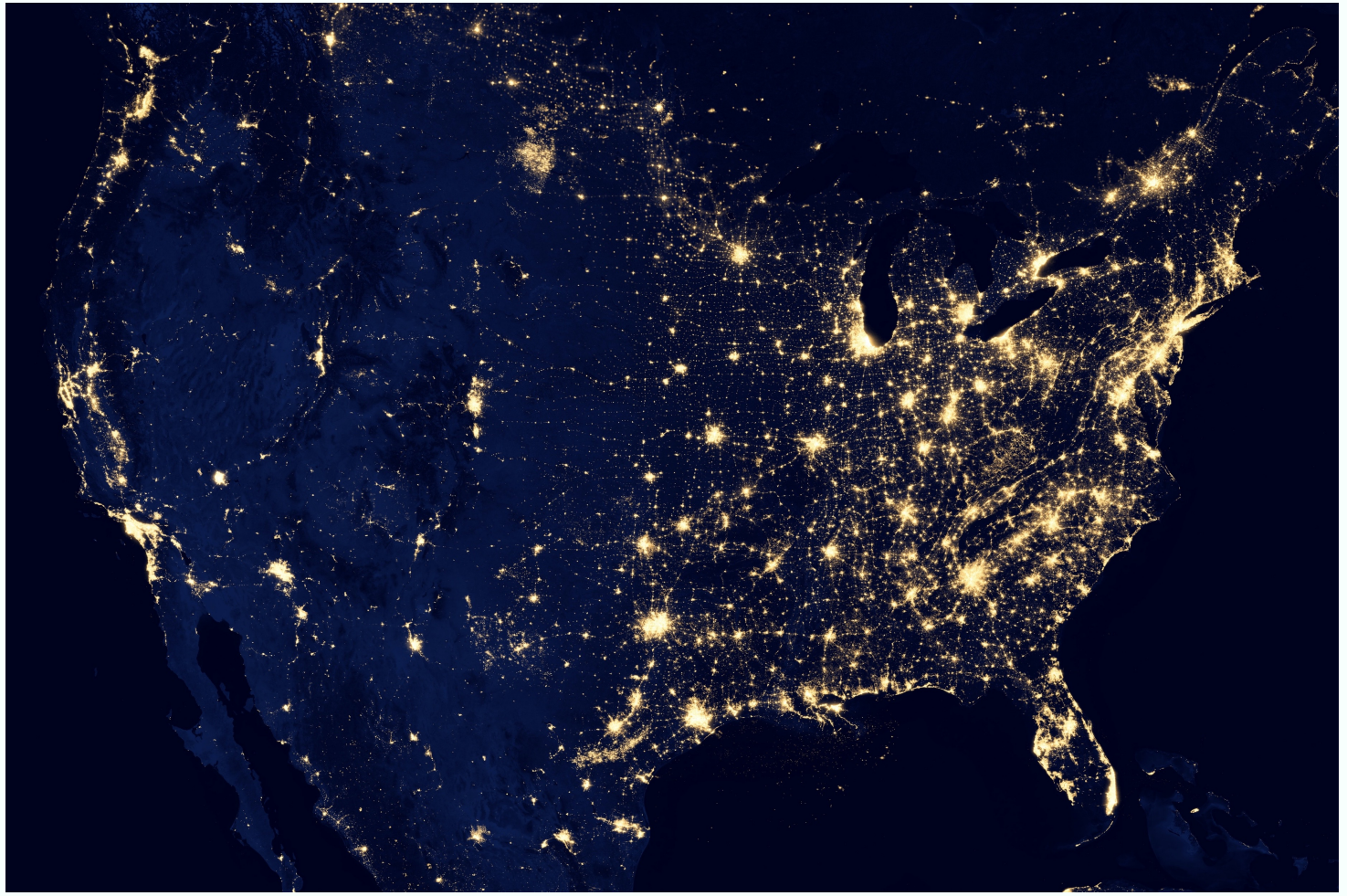
- Carbon: Build applications that are carbon efficient.
- Electricity: Build applications that are energy efficient.
- Carbon Intensity: Consume electricity with the lowest carbon intensity.
- Embodied Carbon: Build applications that are hardware efficient.
- Energy Proportionality: Maximize the energy efficiency of hardware.
- Networking: Reduce the amount of data and distance it must travel across the network.
- Demand Shaping: Build carbon-aware applications.
- Measurement & Optimization: Focus on step-by-step optimizations that increase the overall carbon efficiency.

Too many things? Focus on these:

- Carbon efficiency
  - Energy efficiency
  - Hardware efficiency
  - Carbon awareness

Energy

Location sifting

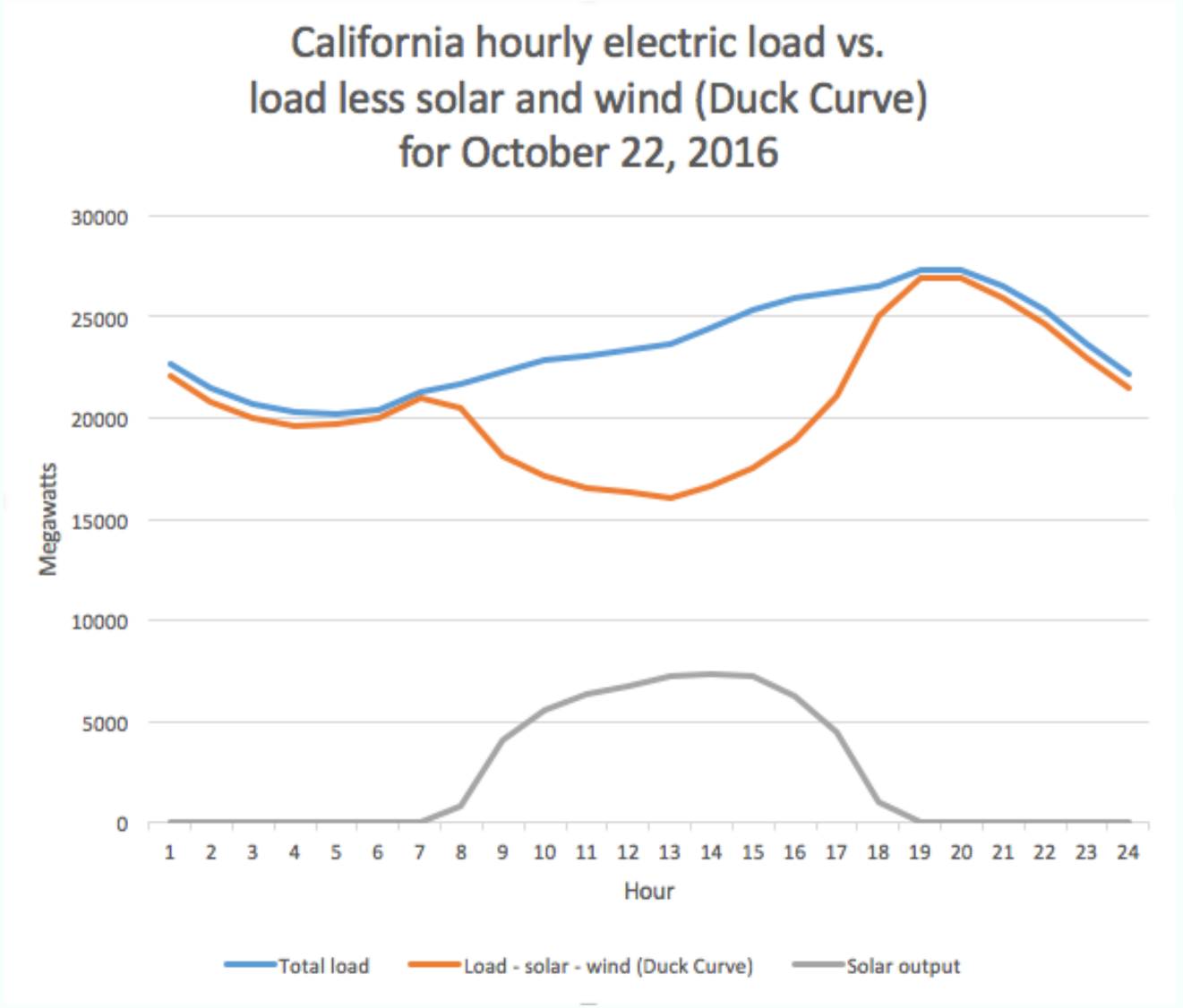


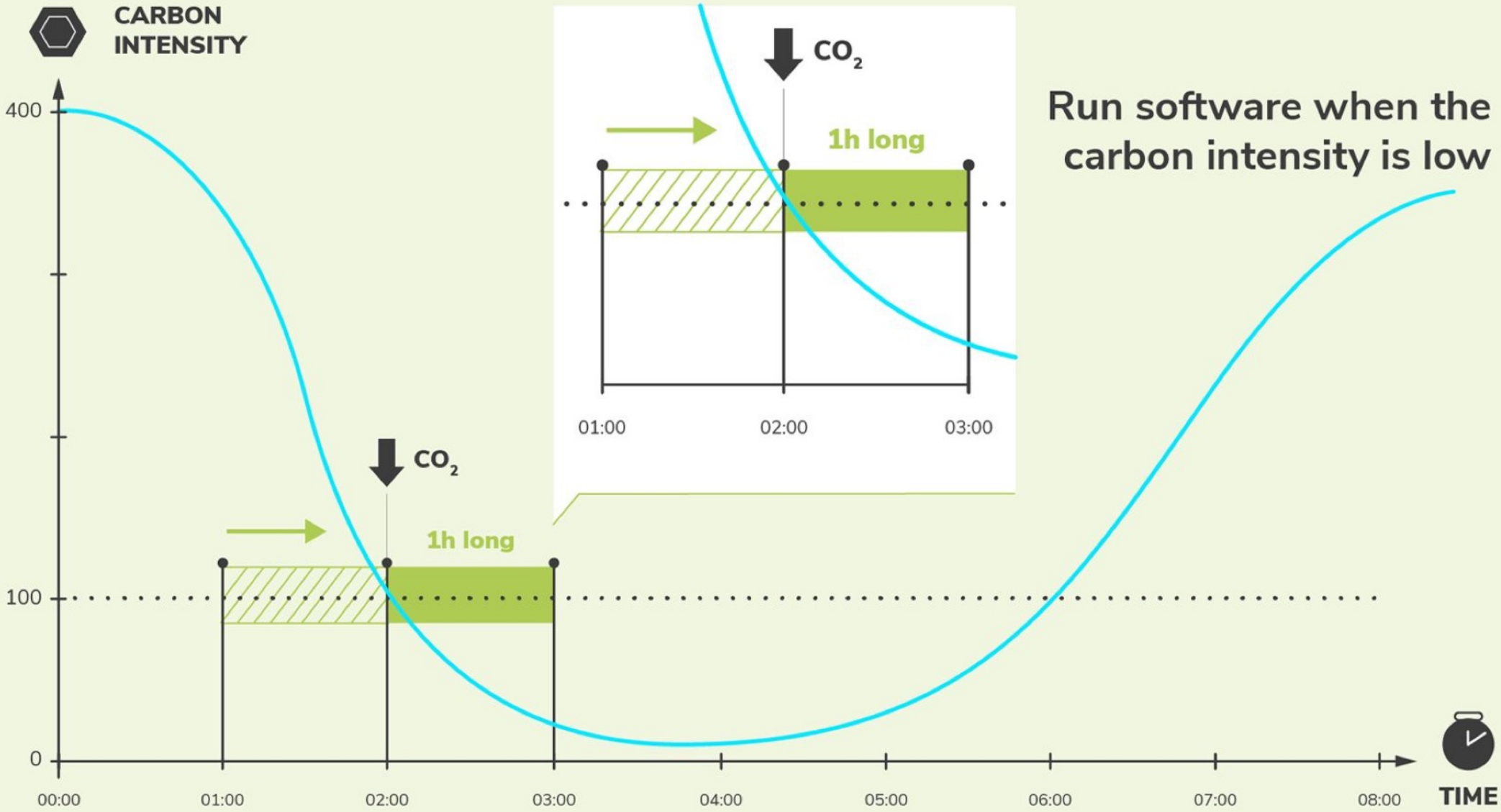
Varies by location

Green Software Foundation

greensoftware.org

Time sifting



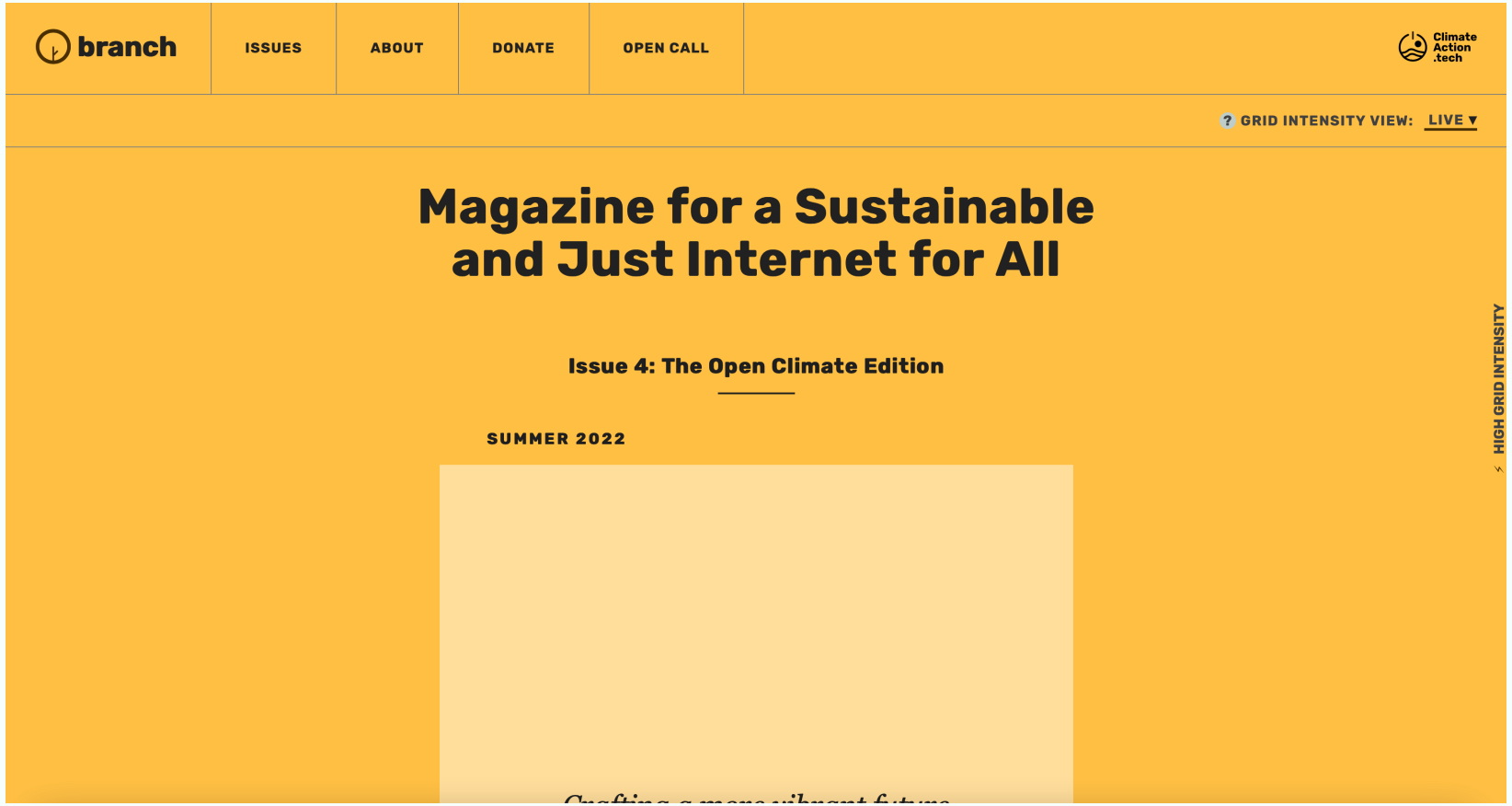


Run software when the carbon intensity is low

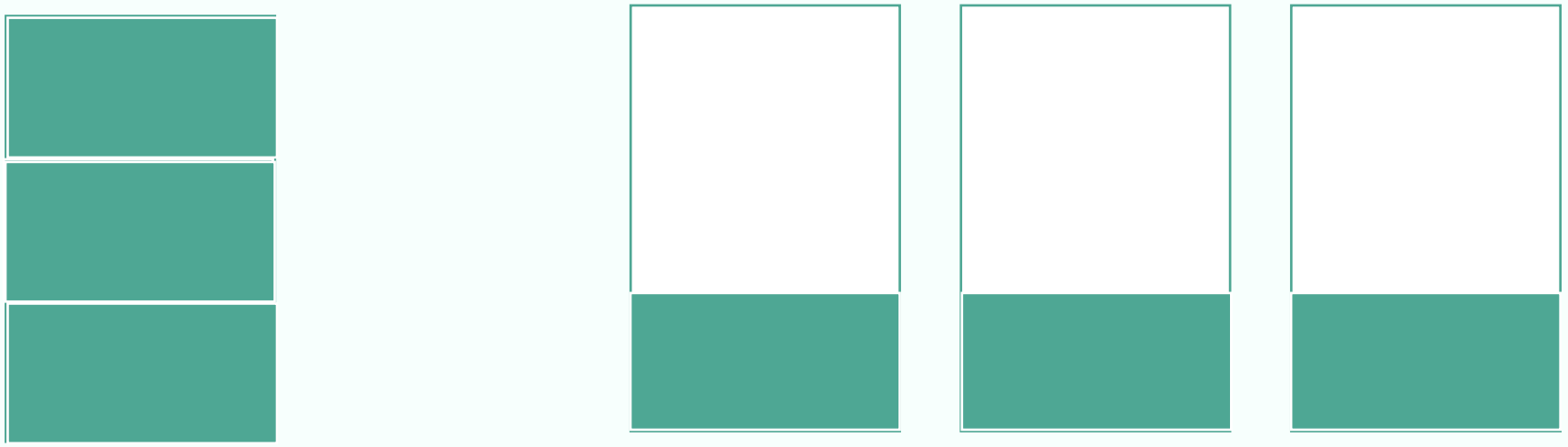
Green Software Foundation

greensoftware.org

Demand shaping



Increasing utilization



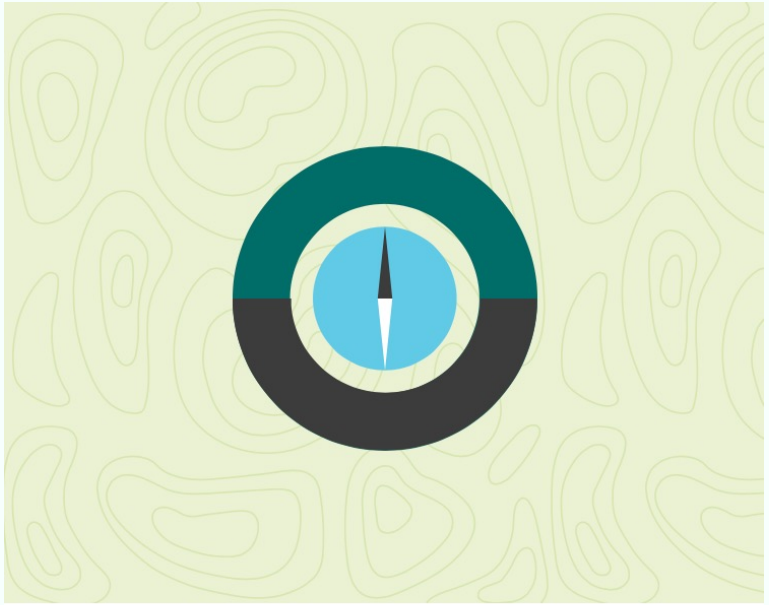
100%.

30%

30%

30%

Software Carbon Intesity - SCI



Software Carbon Intensity (SCI) Specification

The Software Carbon Intensity (SCI) Specification defines a methodology for calculating the rate of carbon emissions for a software system. The purpose is to help users and developers make informed choices about which tools, approaches, architectures, and services they use in the future. It is a score rather than a total; lower numbers are better than higher numbers, and reaching 0 is impossible.

Principles of Green Software in Practice: Microservices

@AnnieTalvasto

Microservices example (or Web Queue Worker)

- Optimize network traffic
- Increase your compute utilization
- Reduce your number of Microservices
- Optimize your database
- Understand your latency limitations

Optimize Your Network Traffic

- Traffic & architecture
- Caching headers
- CDN (Content delivery network)
- Reduce the size and optimize your bundles and static assets.
- Compression and decompression for data

Increase Your Compute Utilization

- Update workload distribution and compute resources - use less resources at a higher utilization.
- Smaller virtual machines
- PaaS
- Auto-scaling or burst capabilities
- Physical tiers & logical layers

Reduce the Number of Microservices

- Microservices architecture
- Combining services
- If two or more microservices are highly coupled, consider co-locating to reduce network congestion and latency
- Use Regions with a lower carbon intensity
- Use languages and technology stacks that optimize the efficiency of a specific microservices function.

Energy Efficiency across Programming Languages

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
ruipereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.couto@inesctec.pt

Francisco Ribeiro, Rui Rua
HASLab/INESC TEC
Universidade do Minho, Portugal
fribeiro@di.uminho.pt
rrua@di.uminho.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacome@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
Universidade de Coimbra, Portugal
jpf@dei.uc.pt

João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

Energy Efficiency across Programming Languages

Energy Efficiency across Programming Languages

How Do Energy, Time, and Memory Relate?

Rui Pereira
HASLab/INESC TEC
Universidade do Minho, Portugal
ruipereira@di.uminho.pt

Marco Couto
HASLab/INESC TEC
Universidade do Minho, Portugal
marco.l.couto@inesctec.pt

Francisco Ribeiro, Rui Rua
HASLab/INESC TEC
Universidade do Minho, Portugal
fribeiro@di.uminho.pt
rrua@di.uminho.pt

Jácome Cunha
NOVA LINCS, DI, FCT
Univ. Nova de Lisboa, Portugal
jacome@fct.unl.pt

João Paulo Fernandes
Release/LISP, CISUC
Universidade de Coimbra, Portugal
jpf@dei.uc.pt

João Saraiva
HASLab/INESC TEC
Universidade do Minho, Portugal
saraiva@di.uminho.pt

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

Optimize your Database

- Choose your database well
- Ensure you are using the best database for interacting with your data set
  - If no easy solution - reduntant copies
- Index
- Evaluate and optimize your queries
- Database cache

Understand your latency limits

- Think about what you actually need for your application
- Request/response cycle
- Worker processes vs web processes
- Worker processes in a region with lower carbon intensity.
- Delay worker process to run when the carbon intensity is the lowest

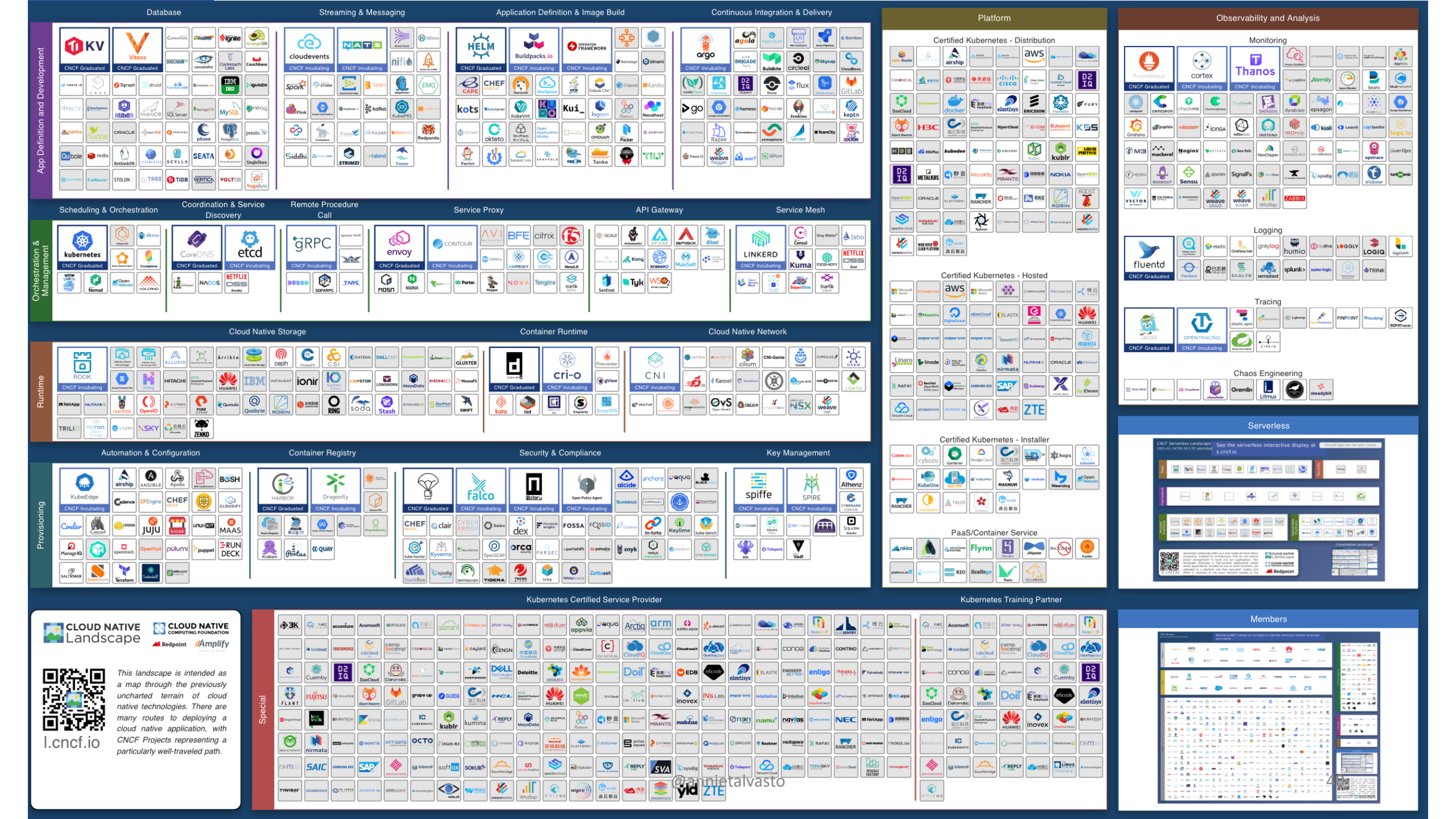
CNCF Open source projects to compliment Kubernetes

CLOUD NATIVE COMPUTING FOUNDATION

Building sustainable ecosystems for cloud native software

The Cloud Native Computing Foundation (CNCF) hosts critical components of the global technology infrastructure. CNCF brings together the world's top developers, end users, and vendors and runs the largest open source developer conferences. CNCF is part of the nonprofit Linux Foundation.

@AnnieTalvasto


Database
Streaming & Messaging
Application Definition & Image Build
Continuous Integration & Delivery
App Definition and Development
Scheduling & Orchestration
Coordination & Service Discovery
Remote Procedure Call
Service Proxy
API Gateway
Service Mesh
Orchestration & Management
Cloud Native Storage
Container Runtime
Cloud Native Network
Runtime
Automation & Configuration
Container Registry
Security & Compliance
Key Management
Provisioning
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
PaaS/Container Service
Observability and Analysis
Monitoring
Logging
Tracing
Chaos Engineering
Serverless
Kubernetes Certified Service Provider
Kubernetes Training Partner
Special
Members
CLOUD NATIVE Landscape
CLOUD NATIVE COMPUTING FOUNDATION
Redpoint
Amplify
l.cncf.io
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.
@annietalvasto
42

BLOG / COMMUNITY POST

CNCF WG Environmental Sustainability

By Max Körbächer + Leonard Pahlke

May 31, 2022

*Community post by **Max Körbächer**, Co-Founder of Liquid Reply, and **Leonard Pahlke**, Consultant at Liquid Reply*

We are pleased to announce that we have established a new working group for environmental sustainability. Our mission is to promote sustainability awareness and develop a culture within the CNCF landscape to establish sustainability best practices and standards.

Environmental sustainability is a pressing issue for humanity, and with simultaneous digitization, software is playing an increasingly important role. Making software ecologically sustainable is becoming more and more vital.

Data Centers, and therefore every compute resource accessible, are currently using 2% of the world's energy. This is conservatively expected to grow to hold 1-2% in the next couple of years, up to 12% by 2040. The growth in energy for computing is outpacing the global growth in energy production. The contributing factors include an explosion in data, the emergence of energy-intensive workloads such as AI and, the flattening of Moore's law. The energy consumption

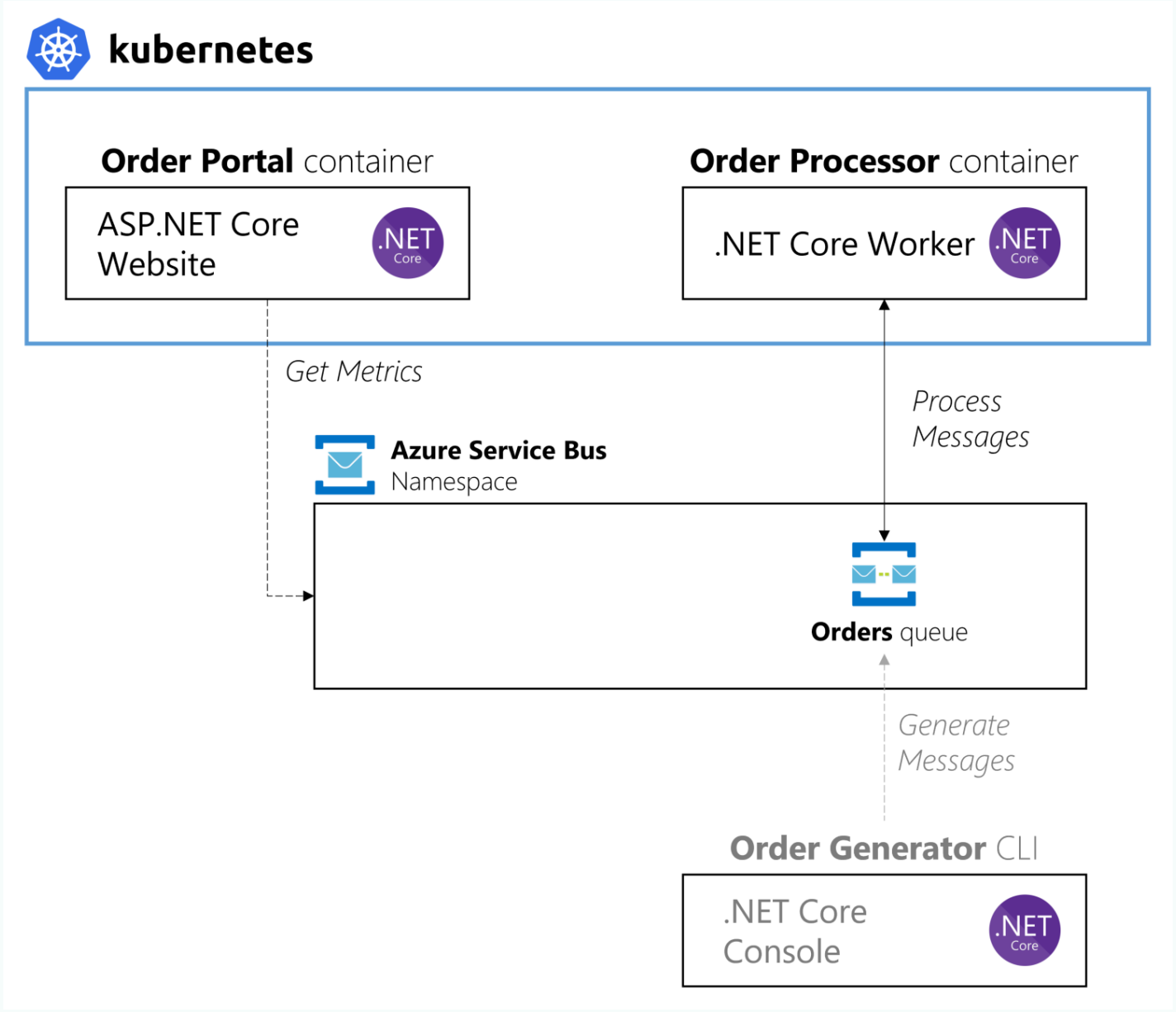
Autoscaling

What is Keda?

- Default Kubernetes Scaling is not well suited for event driven applications.
  - Keda: Event driven scale controlling that can run inside any kubernetes cluster -> Can monitor rate of the events to pre-emptively act before CPU is affected.
- You can install it into new or existing clusters.
- Provides 30+ built-in scalers, but you can build your own
  - Support for external scaler, external push or Metrics API

Demo

KEDA:
Scaling .NET Core worker with Azure Service Bus


kubernetes
Order Portal container
ASP.NET Core Website
.NET Core
Order Processor container
.NET Core Worker
.NET Core
Get Metrics
Process Messages
Azure Service Bus
Namespace
Orders queue
Generate Messages
Order Generator CLI
.NET Core Console
.NET Core

Service Mesh

LINKERD

Linkerd

- Service mesh
- Ultralight, ultrafast, security-first service mesh for Kubernetes.
- The overall goal is to reduce mental overhead of having service mesh
- What does it do?
  - Observability
  - Realiability
  - Security

What are the benefits of Linkerd?

- Thriving open source community
- Simple, minimalist design
- Deep Runtime Diagnostics
- Ultralight and ultra fast
- Installs in seconds with zero config
- Actionable service metrics

Policies

What is Kyverno?

- Designed for Kubernetes
- Policies managed as Kubernetes resources
  - Kubectl
  - Git
  - kustomize
- Validate, mutate and generate

Optimization Example: Cost

How does cloud pricing work?

- On-demand instances
- Reserved instances (+ Savings plans in AWS)
- Spot VMs/instances
- Dedicated host

8 best practices for cost optimization (or for sustainable optimization!)

- Define your requirements
- Choose the right instance types
- Verify storage transfer limitations
- Check if your workload is spot-ready
- Cherry-pick spot instances
- Bid your price on spot
- Use mixed instances
- Make multiple regions work for you

Quick tips

Observability

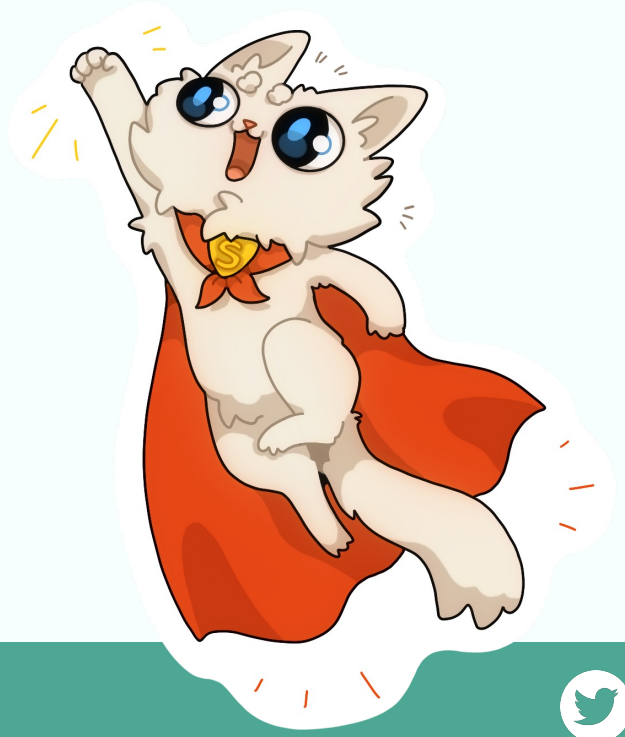
Optimization

Governance

Benefits

- Organizational improvements
- Improved performance
- Cost savings
- Sustainability goals

Saving the world !



Wrap up

- Introduction
- Principles of Green Software
- Principles in Practice: Microservices
- CNCF Open Source Projects
- Optimization Example: Cost
- Resources

Resources

- Slides & materials:
  - Github.com/annietalvasto
- Cloud Dashboards
- https://github.com/Green-Software-Foundation/awesome-green-software
- https://github.com/Green-Software-Foundation/carbon-aware-sdk
- Article about organizational view to green software:
  - https://thenewstack.io/can-reducing-cloud-waste-help-save-the-planet/
- Short course on sustainable software engineering:
  - https://docs.microsoft.com/en-us/learn/modules/sustainable-software-engineering-overview/

1300 XP

The Principles of Sustainable Software Engineering

33 min • Module • 12 Units

★★★★★ 4.8 (13,299)

Beginner | Developer | Administrator | Solution Architect | Student | DevOps Engineer | Data Scientist | Data Engineer | Database Administrator | AI Edge Engineer | AI Engineer | Technology Manager | Azure | .NET | Microsoft Power Platform

Sustainable Software Engineering is an emerging discipline at the intersection of climate science, software, hardware, electricity markets, and data center design. The Principles of Sustainable Software Engineering are a core set of competencies needed to define, build, and run sustainable software applications.

Learning objectives

In this module, you will:

- Identify the eight principles of Sustainable Software Engineering
- Understand the two philosophies of Sustainable Software Engineering

Start ⊕ Save

Prerequisites

- Familiarization with general computing concepts

Open-source tooling

- CNCF projects:
  - Event driven autoscaling: KEDA
  - Application management: Helm
  - Monitoring: Prometheus
  - Container workflow: Flux
  - Policies: Kyverno
  - Service Mesh: Linkerd, Kuma (+Meshery)
- Data on machine learning carbon footprint: https://mlco2.github.io/impact/
- A curated list of open technology projects to sustain a stable climate, energy supply, and vital natural resources: https://opensustain.tech/
- Green software foundation

Conclusion

Thank you!


when you remember to bring your reusable bag to the grocery store
STARECAT.COM