Je code : les bonnes pratiques en éco-conception de service numérique à destination des développeurs de logiciels

Cyrille Bonamy, Cédric Boudinet, Laurent Bourgès, Karin Dassas, Laurent Lefèvre, Benjamin Ninassi, Francis Vivat

Je code : les bonnes pratiques en écoconception de service numérique à destination des développeurs de logiciels

Auteurs :
Cyrille Bonamy : LEGI / CNRS
Cédric Boudinet : G2Elab / Grenoble INP
Laurent Bourgès : OSUG / CNRS
Karin Dassas : CESBIO / CNRS
Laurent Lefèvre : Inria / ENS Lyon
Benjamin Ninassi : Inria
Francis Vivat : LATMOS / CNRS

Les auteurs sont membres du Groupement de Service CNRS EcoInfo [1.1] qui travaille sur l'écoresponsabilité du numérique.

Résumé :

Cette plaquette est un complément aux 3 plaquettes de bonnes pratiques liées au développement logiciel proposées par le réseau des acteurs du DEVeloppement LOGiciel au sein de l'Enseignement Supérieur et de la Recherche : DevLOG.

Ce volet est dédié aux bonnes pratiques en termes d'écoconception de service numérique qui permettent d'appréhender, de comprendre et de réduire l'impact environnemental du numérique.

Après avoir explicité le contexte général dans une première fiche, une seconde fiche ("**Mais pourquoi ?**") met en évidence la nécessité d'intégrer une dimension environnementale dans nos conceptions de service numérique, et par conséquent dans nos développements de logiciels. La troisième fiche ("**Quand ?**") rappelle les étapes du cycle de vie d'un service numérique pour introduire les fiches de bonnes pratiques qui correspondent aux différentes étapes : "**Avant**", "**Pendant**" et "**Après**", en gardant à l'esprit que le développement est souvent itératif, et les frontières entre les différentes étapes sont perméables.

Vous trouverez à la fin de la plaquette une fiche spécifique sur les bonnes pratiques d'écoconception pour le calcul scientifique, ainsi que des fiches sur le développement sur plateforme mobile, pour le web et sur accélérateur.

Les plaquettes DevLOG existantes :

Je code : les bonnes pratiques de développement logiciel [1.2]
Je code : les bonnes pratiques de diffusion [1.3]
Je code : quels sont mes droits et obligations ? [1.4]

Remerciements pour leur relecture à : Françoise Berthoud, Rémi Cailletaud, Christophe Cossou, Loïc Maurin, Gabriel Moreau, Patrick Moreau, Olivier Ridoux, Jean-Christophe Souplet

Un service numérique, c'est :

- de l'information : les données
- des traitements : algorithme, filtrage, simulation
- des échanges d'informations
- des interfaces utilisateurs

Un service numérique repose sur :

- des infrastructures logicielles : applications, outils, bibliothèques, protocoles
- des infrastructures matérielles : serveurs, équipements réseau, terminaux, capteurs
- des personnes : développeurs, administrateurs systèmes et réseaux, chefs de projet, chercheurs

Les impacts humains [2.1]

A travers leur quotidien, les humains collaborant à la production de services numériques vont générer des impacts : ils vont par exemple utiliser du matériel informatique et se déplacer pour venir à leur bureau ou pour se réunir. Mettre en place une politique environnementale d'organisation permettant de limiter ces impacts est crucial. Par exemple, favoriser la prolongation de la durée de vie des ordinateurs et l'utilisation de matériel reconditionné va réduire l'impact de la fabrication d'équipements. D'une manière générale, contribuer à la mise en place d'une politique environnementale dans votre organisation.

Exemple de service numérique : *le workflow d'une simulation numérique.*

Il est constitué des briques suivantes :

- la préparation des données d'entrée
- le transfert des données d'entrée vers la plateforme de calcul considéré (machine locale, méso centre régional, GENCI [2.2], cloud public, GPU/CPU)
- le calcul sur la plateforme de calcul
- le transfert des données vers la plateforme de post-traitement
- le post-traitement, stockage et dissémination des données de sortie du calcul
- l'analyse et l'exploitation des données

De nombreux leviers pour réduire les impacts, mais une vision globale est nécessaire :

- limiter les transferts de données
- choix plateforme de calcul et de pré et post-traitement
- post-traiter les données au plus proche du lieu de création
- limiter les données d'entrée et/ou de sortie
- choisir des briques logicielles externes ou à redévelopper en interne

Eco-concevoir un service numérique, c'est intégrer les aspects environnementaux tout au long de son cycle de vie [2.3] (attention à bien définir l'unité fonctionnelle [2.4])

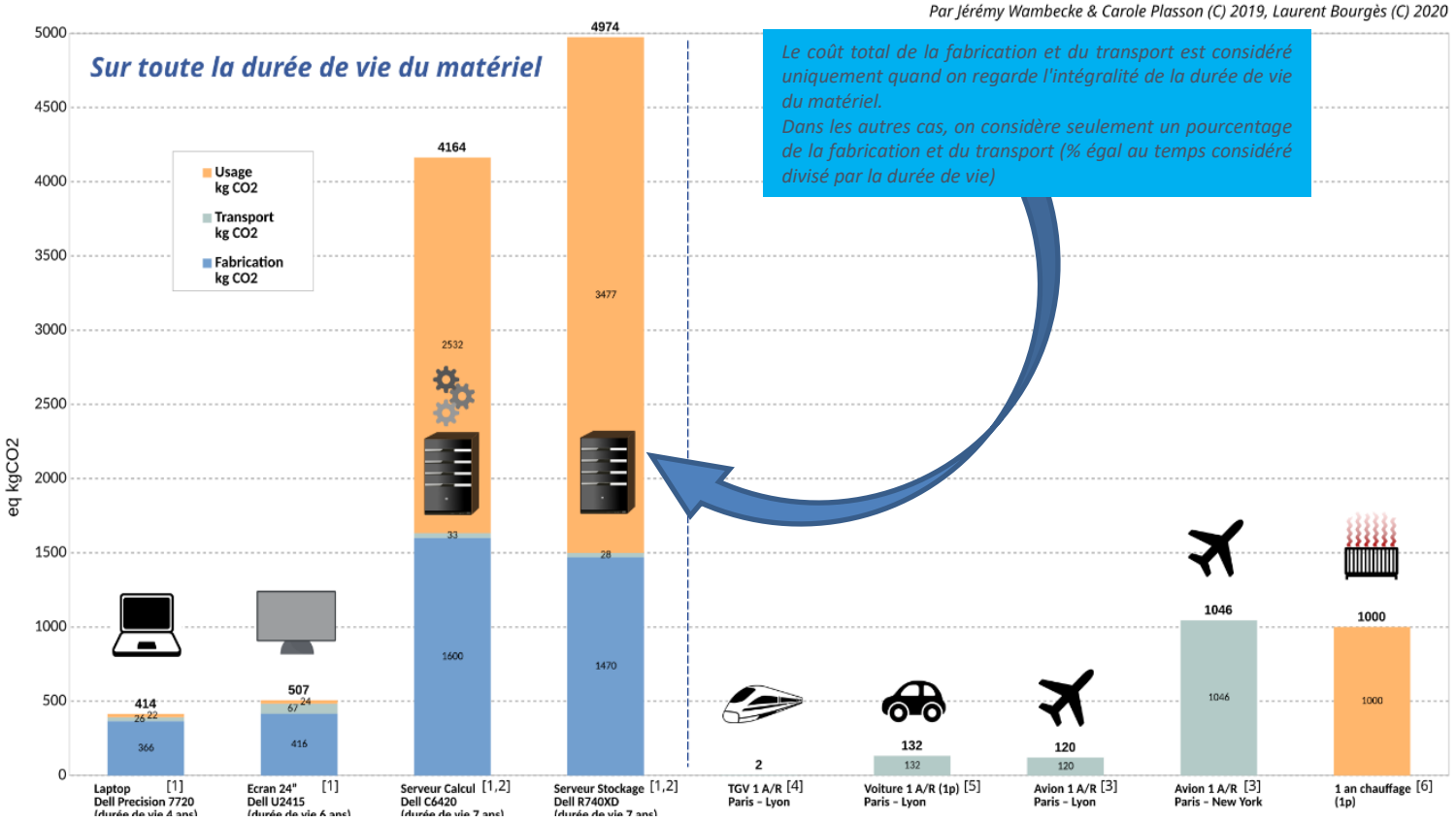
Développeurs, mais aussi chefs de projet, ingénieurs et chercheurs, tous et toutes sont concernés par cette plaquette !

Mais pourquoi ?

Pourquoi se préoccuper de l'impact environnemental du numérique ? Tout simplement parce qu'il est loin d'être négligeable. L'augmentation de la contribution du numérique dans les émissions GES (Gaz à Effet de Serre) globales mondiales est exponentielle. Il est temps de tout mettre en œuvre pour stopper cette croissance exponentielle.
Quelques références sur le sujet en page 14 (ARCEP [3.1], ADEME [3.2], EcoInfo [3.3], The Shift Project [3.4], Green IT [3.5], INR [3.6], DINUM[3.7], Boavizta[3.8])
A noter que l'impact du numérique ne se limite pas à sa déclinaison la plus médiatisée, les émissions de gaz à effet de serre et le réchauffement climatique. L'impact majeur est dû à la fabrication du matériel numérique et au traitement des déchets : pollution des sols et des nappes, diminution des ressources en eau, et même impact sur le santé et la biodiversité. Le traitement des D3E (Déchets déchets d'équipements électriques et électroniques) D3E [3.9] n'entre cependant pas dans le cadre de cette plaquette.

Comparatif d'émissions CO2

Par Jérémy Wambecke & Carole Plasson (C) 2019, Laurent Bourgès (C) 2020



[1] Données Fiches Dell (usage corrigé pour usage FR) : (https://www.dell.com/learn/us/en/uscorp1/corp-comm/environment_carbon_footprint_products)
[2] Usage à partir de la consommation moyenne (Berthoud et al. 2020) d'un nœud = 275W (C6420), 375W (R740XD) (https://hal.archives-ouvertes.fr/hal-02549565)
[3] https://eco-calculateur.dta.aviation-civile.gouv.fr/
[4] https://ressources.data.sncf.com/explore/dataset/emission-c02-tgv/table/
[5] Trajet de 473km, pour une voiture émettant 0,140 kg CO_2/km
[6] https://www.insee.fr/fr/statistiques/fichier/1281320/ip1445.pdf
Facteur d'impact : 0,108 kgCO_2/kWh (FR)

- le logiciel est au cœur du service numérique et conditionne le matériel nécessaire ainsi que sa durée de vie
- la fabrication du matériel est non négligeable d'un point de vue environnemental
- un moindre renouvellement du matériel (fortement impacté par le logiciel qu'il opère) permet de diminuer l'impact de la phase de fabrication des équipements

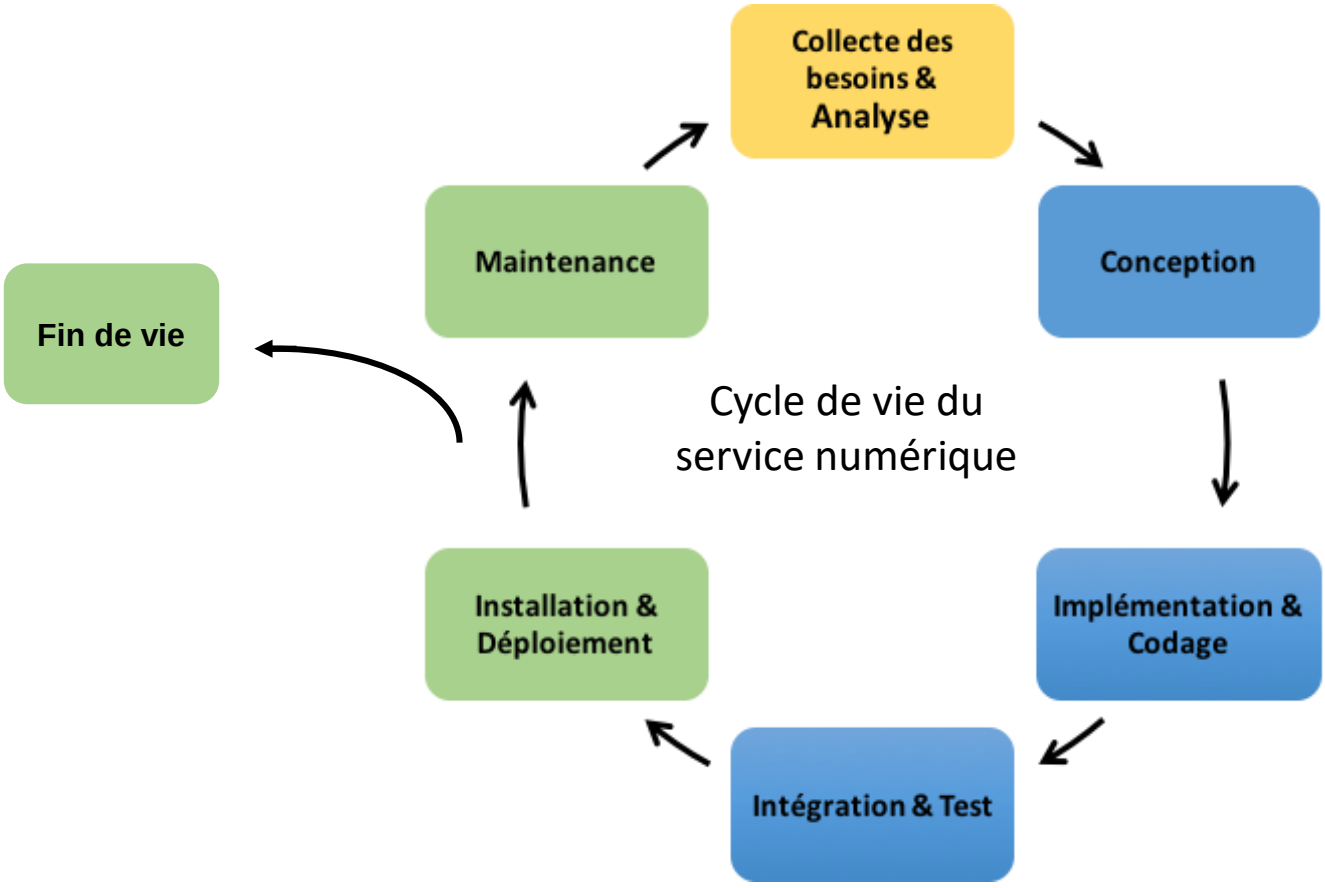
L'efficacité ou la sobriété du logiciel influe fortement sur le besoin de puissance de calcul, mémoire et stockage, ce qui permet de réduire le volume de matériels nécessaires au service numérique, donc de diminuer l'impact de la phase de fabrication des équipements et la consommation énergétique en fonctionnement dans une moindre mesure.

Exemple d'impact dans le milieu de la recherche : 5 millions d'heures.coeur de calcul (env. 25 tonnes eq CO2 [3.10]) sont équivalents à 25 Allers-Retours Paris - New York en avion selon l'aviation civile [3.11] ou 11 selon Ecolab [3.12].

Quand ?

L'écoconception d'un service numérique consiste à intégrer des contraintes environnementales dans tous les processus de développement, afin de **réduire les impacts environnementaux du service numérique pendant son cycle de vie.** Les impacts peuvent avoir lieu à toute étape du cycle de vie du service : avant son développement (définition des besoins, analyse), pendant son développement (conception, développement logiciel, tests, mise en production) et après son développement (exploitation, maintenance, fin de vie). De façon générale, plus la prise en compte des aspects environnementaux intervient tôt dans le cycle de vie, plus l'effet est important. Cette plaquette propose d'explorer différentes bonnes pratiques d'écoconception de service numérique en s'intéressant tout particulièrement au logiciel, et aux éléments à prendre en compte avant, pendant et après la phase de développement logiciel.

A noter que les bonnes pratiques proposées ne sont pas dépendantes du cycle de développement choisi : cycle en V, en W, en spirale, en cascade, modèle agile. Un modèle itératif peut permettre de diminuer les impacts d'une étape lors d'une nouvelle itération.



Agir pour réduire l'impact environnemental est possible à chaque étape

Suivre le code couleur !

Avant | **Pendant** | **Après**

S'appuyer sur les grands principes d'écoconception de service numérique :

- **simplicité** : simplifier le logiciel pour éviter les usines à gaz
  - en termes de fonctionnalités : 70 % des fonctionnalités demandées par les utilisateurs ne sont jamais ou rarement utilisées (Standish Group, 2006)
  - en termes d'interfaces utilisateurs
- **frugalité et sobriété** : limiter le nombre et la taille des éléments (images par exemple). Par exemple, dans un développement web, éviter les pages "obèses" en terme de fonctionnalités et de graphisme.
- **pertinence** = utilité (le résultat doit répondre à l'attente de l'utilisateur) x rapidité (temps de réponse pour l'utilisateur) x accessibilité (par exemple pour certains handicaps)
- **durabilité** : réutiliser tout ou partie d'un logiciel permet d'éviter de dupliquer les développements; contribuer pour le bénéfice de la communauté .

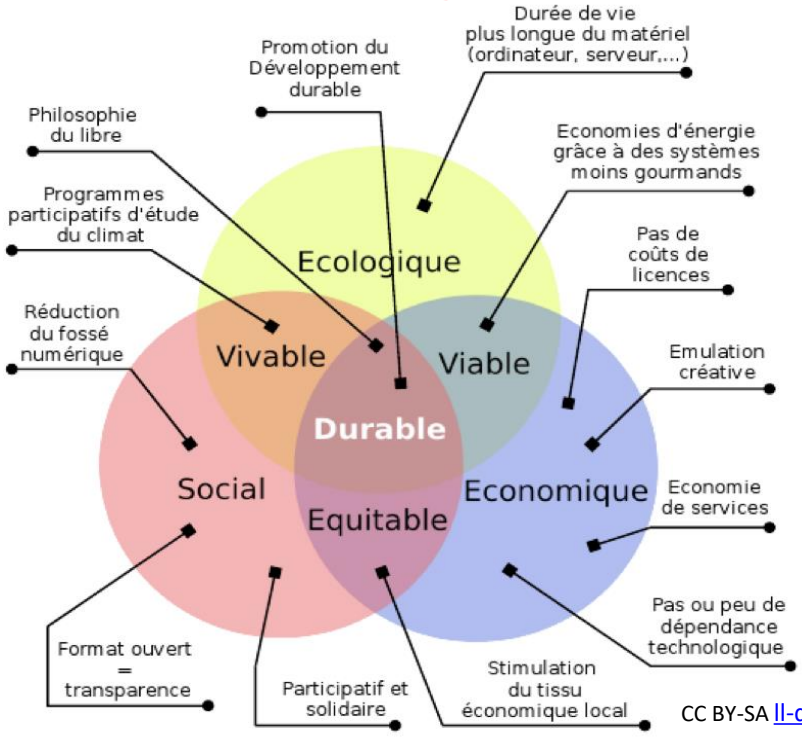
Je maitrise le nombre de fonctionnalités logicielles : éviter l'obésiciel

Trop de fonctionnalités disparates : où est passée la fonction que j'utilisais le plus souvent ? Mon logiciel est devenu un obésiciel ! Mon service numérique devient une véritable "usine à gaz" !

Plus de fonctionnalités que nécessaire : mon ancienne infrastructure ne suffit plus, je dois en changer !

Fonctionnalités justifiées et utiles : mon service numérique est léger et efficace !

Je favorise le libre : réutiliser des briques logicielles et contribuer aux communs



Je planifie la gestion du logiciel : accroître la durée de vie

Un plan de gestion logiciel (**SMP** Software Management Plan) est un outil pour la pérennisation du logiciel. Les informations sur le logiciel, sur son contexte, sur ses caractéristiques, ou sur l'organisation de sa diffusion sont ainsi regroupées, mises à jour au cours du cycle de vie, et permettent d'en faire un produit modifiable et réutilisable facilement. Opidor [5.2], Presoft [5.3] (voir aussi le Data Management Plan [5.4] Slide 7)

Prendre le temps de réfléchir à la plateforme de déploiement visée tout en considérant les contraintes du service numérique complet. Ne pas oublier de considérer les aspects sécurité, avant même le développement.

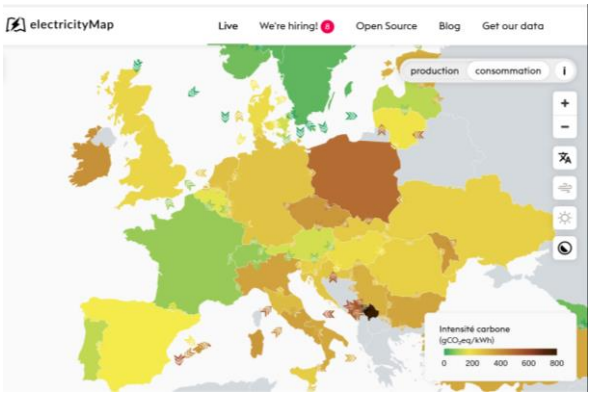
Je réfléchis au déploiement du service : s'adapter au mieux au contexte

En fonction des caractéristiques (lieu, capacités…) des plateformes disponibles :

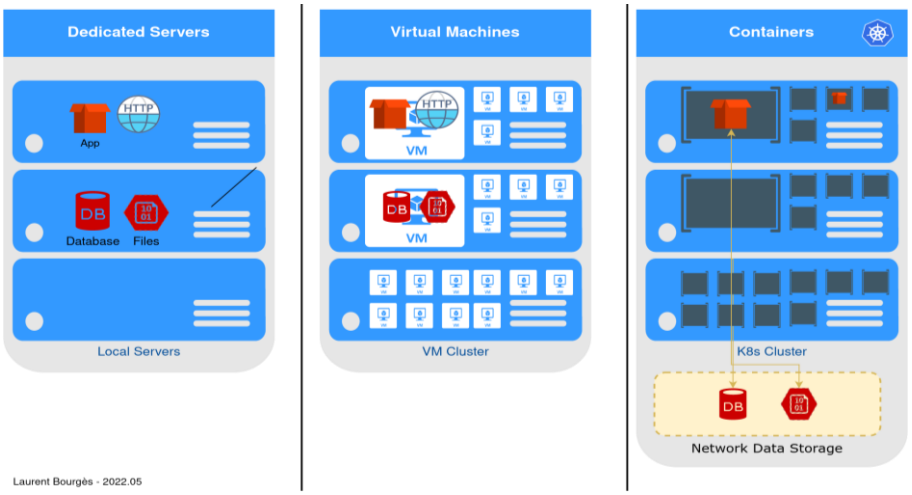
- ordinateur, serveur local, plateforme embarquée (microcontrôleur, Raspberry-Pi…)
- plateforme spécialisée : cluster virtualisation ou spécialisé HPC, GPU, FPGA, ARM64
- une Offre De Service (ODS) de site ou de tutelles (CNRS, Universités) / clouds publics

En fonction des contraintes du service :

- langages supportés, communications spécialisées (Infiniband, OmniPath, SpaceWire, Série…)
- goulots d'étranglement : accès disque, réseau, transferts mémoire
- pérennité, portabilité, sécurité, coûts à long terme, maintenance, temps de retour
- lieu géographique des usages, tutelles commanditaires du service



Electricity map / Intensité carbone [6,1]



Quelques recommandations :

- je m'appuie sur les services mutualisés (authentification, stockage, base de données…)
- j'isole les données sur des espaces de stockages distribués et sauvegardés pour éviter la duplication
- je privilégie la virtualisation et je réduis la taille des images (VM, container) : cf schéma à droite ci-dessus
- j'estime au plus près les ressources nécessaires (vcpu, mémoire, stockage) pour éviter le surdimensionnement
- j'automatise le déploiement des machines physiques et virtuelles pour garantir la reproductibilité

Je n'oublie pas les aspects sécurité de mon service

De l'indisponibilité de mon service jusqu'au vol de données personnelles de mes utilisateurs en passant par l'installation de logiciels malveillants sur mon infrastructure, les conséquences d'une faille de sécurité peuvent être nombreuses. L'application ou/et les données associées peuvent devenir obsolètes.

La sécurité doit être une préoccupation présente à toutes les étapes du cycle de vie d'un service numérique, et en lister les bonnes pratiques nécessiterait une plaquette à part entière. Je peux commencer par m'inspirer des recommandations de l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI-C [6.2], ANSSI-Rust [6.3], ANSSI-Java [6.4]).

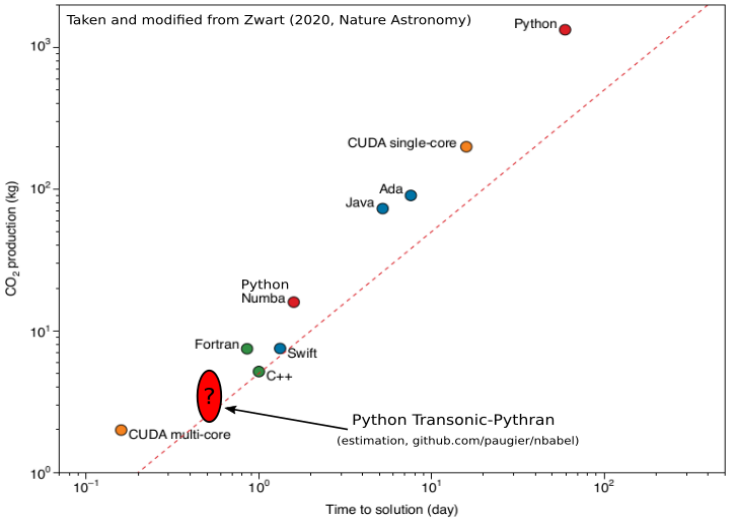
Néanmoins je reste vigilant sur l'impact des mécanismes de sécurité considérés, notamment sur l'augmentation de la consommation de ressources ou du nombre d'équipements : bien prendre en compte la criticité des services pour ne pas sur-sécuriser inutilement.

Prendre le temps de choisir un langage en considérant l'efficacité énergétique, la pérennité, la simplicité et plus généralement les contraintes globales du service numérique complet.

Je choisis mon langage et/ou ma pile logicielle : tout est affaire de compromis

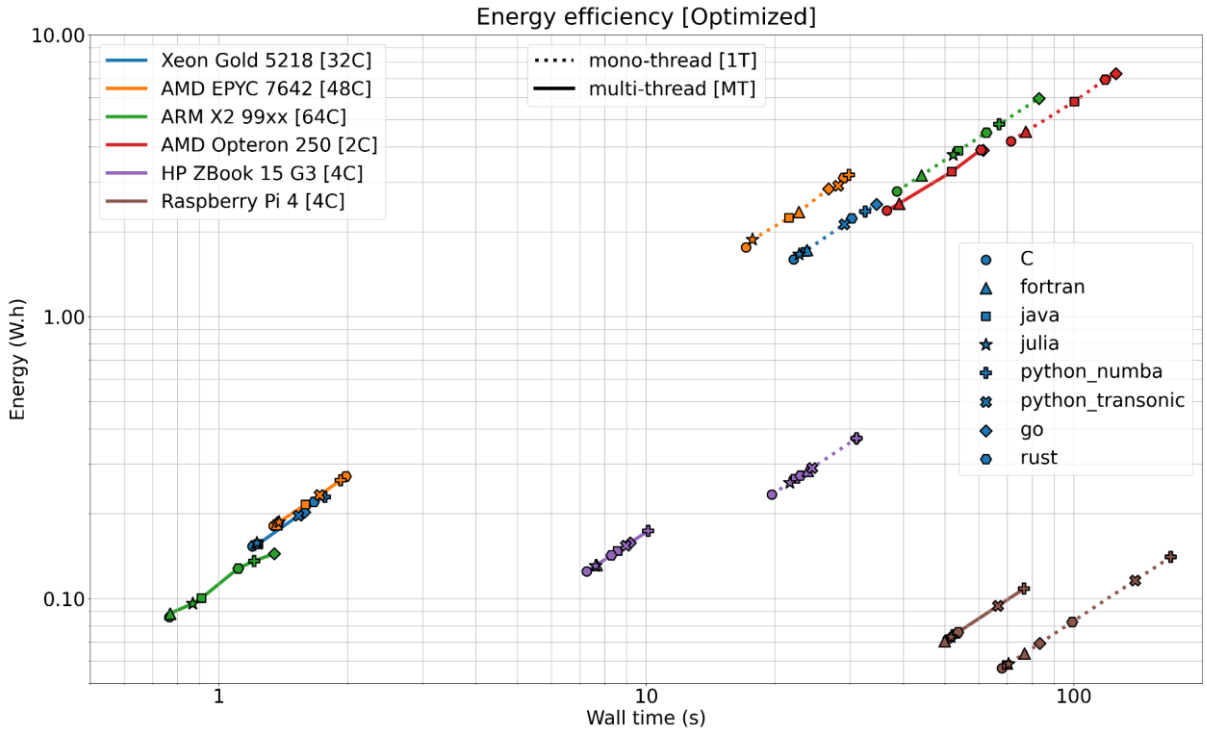
Illustrations de l'efficacité énergétique en fonction du langage :

- *à gauche : Simon P. Zwart, Nature Astronomy [7.1] avec une simulation N-Body, figure actualisée par P. Augier [7.2] et [7.3] mettant en évidence l'importance des compilateurs Python et des facteurs humains (optimisation du code, expertise et temps passé pour optimiser, tester et mesurer les codes),*
- *à droite, Rui Pereira et al, SLE17 [7.4] [7.5] basé sur le Computer Langage Benchmark Game.*



Time & Memory	Energy & Time	Energy & Memory	Energy & Time & Memory
C • Pascal • Go	C	C • Pascal	C • Pascal • Go
Rust • C++ • Fortran	Rust	Rust • C++ • Fortran • Go	Rust • C++ • Fortran
Ada	C++	Ada	Ada
Java • Chapel • Lisp • Ocaml	Ada	Java • Chapel • Lisp	Java • Chapel • Lisp • Ocaml
Haskell • C#	Java	OCaml • Swift • Haskell	Swift • Haskell • C#
Swift • PHP	Pascal • Chapel	C# • PHP	Dart • F# • Racket • Hack • PHP
F# • Racket • Hack • Python	Lisp • Ocaml • Go	Dart • F# • Racket • Hack • Python	JavaScript • Ruby • Python
JavaScript • Ruby	Fortran • Haskell • C#	JavaScript • Ruby	TypeScript • Erlang
Dart • TypeScript • Erlang	Swift	TypeScript	Lua • JRuby • Perl
JRuby • Perl	Dart • F#	Erlang • Lua • Perl	
Lua	JavaScript	JRuby	
	Racket		
	TypeScript • Hack		
	PHP		
	Erlang		
	Lua • JRuby		
	Ruby		

L'usage de Python illustre le plus grand écart (~ 100x) en efficacité selon l'environnement d'exécution (cpython, numba, pythran, pypy) et l'optimisation du code dans ce cas d'utilisation (performance-driven development).



Dans l'étude ci-dessus (C. Bonamy, L. Bourgès, L. Lefèvre [7.6], les machines à base d'ARM sont réellement efficaces quel que soit le langage considéré, et sans "portage" comme le demanderait l'utilisation de GPU. On voit aussi que deux groupes de langages se distinguent en termes d'efficacité : en tête, C, Fortran, Java et Julia, et légèrement derrière Python, Go et Rust

- langages compilés (natifs ou habituellement interprétés mais optimisés, p. ex. grâce à Numba ou PyThran pour Python) à privilégier pour les traitements lourds, haute performance ou temps réel
- langages faciles d'accès (interprétés) à privilégier pour les traitements moins contraints, afin de faciliter la maintenance, le ré-usage et ainsi la durabilité
- toutes les briques ne doivent pas forcément être écrites dans le même langage

Anticiper, prendre en compte les données pour diminuer leur impact environnemental. Tenir compte de la volumétrie et des coûts de transfert des données. Favoriser l'interopérabilité et les données ouvertes.

Je fais attention à la volumétrie des données : sobriété numérique

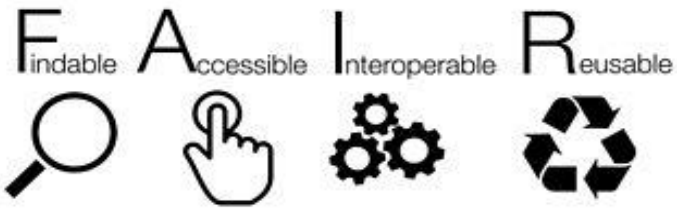
Stockage des données : ASCII ou formats binaires ?

Le format texte (ASCII brut, XML, YAML...) nécessite généralement plus de place pour stocker la même information. Par contre lorsqu'on versionne les fichiers, le phénomène peut s'inverser (stockages successifs vs. stockage différentiel). Il y a un risque d'obsolescence à l'utilisation des formats binaires non ouverts.

Volumétrie importante ?

- je privilégie le traitement au plus près des données (serveur) pour éviter de transférer les jeux de données sur les postes des utilisateurs (et besoin d'autres machines puissantes)
- je tiens compte de la volumétrie des données manipulées pour concevoir les traitements et éviter les transferts volumineux entre les couches logicielles
- je minimise la volumétrie du paquet logiciel (archive) et j'élimine les dépendances superflues.

Je planifie la gestion des données : durabilité, diminution des développements redondants



source :SangyaPundir CC BY-SA 4.0

Suivre le principe FAIR permet de réduire l'empreinte environnementale pour plusieurs raisons :

- **Interopérabilité** : les données doivent être faciles à combiner avec d'autres jeux de données, à la fois par les humains et par les systèmes informatiques. Choisir de rendre les données interopérables permet d'éviter de multiplier des données avec des formats différents : diminution du nombre de données et du coût de conversion.
- **Open Data / Reproductibilité** : rendre les données et les codes sources facilement accessibles (référence DOI pérenne) et reproductibles permet de réduire l'empreinte environnementale en évitant des doublons. Moins de stockage (utilisation de web-services), moins de développement redondant.

De nombreuses ressources existent : principes FAIR [8.1] / directive européenne [8.2] / European Open Science Cloud [8.3] / Research Data Alliance [8.4] ou encore cette infographie FAIR interactive [8.5] proposée par l'URFIST Méditerranée.

Les objectifs souvent cités d'un plan de gestion de données (DMP) [8.6] de la recherche sont les suivants : améliorer l'impact des projets de recherche et leur contribution scientifique, et même satisfaire les exigences de financeurs. Un des autres objectifs de ce plan est environnemental : un plan correctement fait permet d'aboutir à des données FAIR. Quelques exemples de DMP sur le site OPIDoR [8.7], voir aussi sprint PGD [8,8] de Datactivist.

Le Règlement Général sur la Protection des Données [8.9] définit un cadre strict sur la collecte des données personnelles à minima, ainsi que leur durée de rétention et de conservation la plus courte.
Voir aussi l'Atelier des données [8.10]

Je pense aussi aux principes FAIR pour le logiciel : FAIR4RS [8.11]

Analyser et superviser mon service numérique afin d'identifier les briques logicielles et les fonctions les plus consommatrices.

J'analyse mon code : identifier *a priori*

- les analyses statiques permettent, entre autres, de déterminer la difficulté de maintenance d'un logiciel (exemples d'outils : Sonarqube, codacy)
- les analyses dynamiques permettent de quantifier l'usage des ressources (exemples outils : gcov, gprof, perf, valgrind)

Ces analyses de logiciel - qui font partie des bonnes pratiques classiques - sont indispensables afin de réduire l'impact d'un service numérique.

Je mesure les performances : identifier en fonctionnement

- temps de chargement
- temps d'exécution
- consommation électrique
- consommation des protocoles réseau (par exemple dans l'IOT)
- taille des données
- nombre de requêtes
- performance web
- mesure émissions carbone

Exemples d'outils : Firefox web tools (trafic, requêtes, JavaScript), Apache JMeter (scénarios web), ecoindex (web), profiler intégré et adapté au langage, CodeCarbon, Green Algorithms, KDE eco)

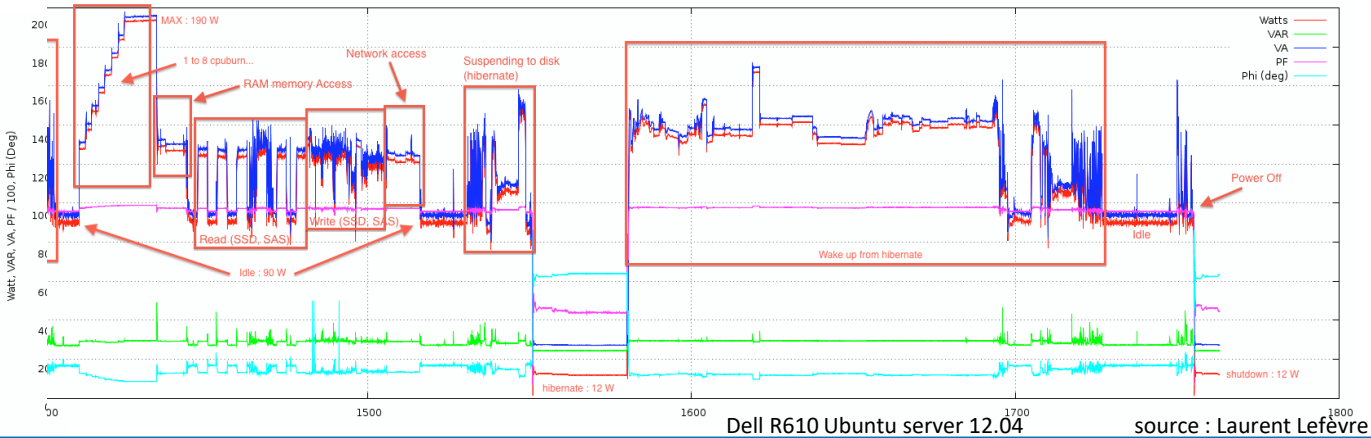
Un focus : je profile la consommation énergétique des logiciels

Chaque usage de ressource (processeur, mémoire, espace de stockage, réseau) dans mon logiciel provoque une consommation électrique.

Pour mesurer l'impact énergétique des logiciels en phase d'usage :
besoin d'équipements matériels (PDU, wattmètres) ou de sondes logicielles (powerAPI attention à la calibration idle).

Wattmètre Watts'up Pro - PDU Eaton

Je profile la consommation électrique dans le temps, pour découvrir les différentes phases intensives de mon logiciel.



Dell R610 Ubuntu server 12.04 source : Laurent Lefèvre Inria

Optimiser son logiciel, comme implémenter des algorithmes efficaces et les structures de données adéquates, permet de réduire le temps d'exécution, ce qui conduit en général à une réduction de la consommation électrique, et donc à une réduction des émissions eqCO2. Mais attention à l'effet rebond, et aux effets sur les autres éléments du service numérique. De même il faut être vigilant lorsque l'on applique les bonnes pratiques usuelles en ingénierie logicielle (gestion de version, intégration continue). Il faut les appliquer, mais avec des précautions (oui, mais....).

J'optimise mon code (oui mais...)

Attention à l'effet rebond :

Optimiser un logiciel peut induire à lancer davantage d'opérations ou traiter davantage de données, donc l'empreinte écologique du service ne sera pas réduite (Paradoxe de Jevons).

L'optimisation devrait servir simplement à réduire la consommation énergétique et des ressources, et si possible d'arriver plus vite au résultat. Chaque exécution a un impact !

Il est primordial de n'optimiser que ce qui a le plus d'impact (Loi de Pareto).

Il ne faut pour autant pas négliger la qualité des tests dans le processus d'optimisation (régressions).

J'améliore mon code (oui mais...)

L'amélioration est un problème multifactoriel : architecture, utilisation des ressources processeur, mémoire, stockage et trafic réseau. L'utilisation d'un levier d'amélioration peut nuire à l'usage d'une autre ressource.

Par exemple :

- adopter un système de cache permet d'accélérer les traitements, mais attention aux impacts sur la mémoire, le stockage et la complexité accrue du logiciel
- utiliser la compression pour réduire le trafic réseau augmente légèrement l'utilisation du CPU
- sécuriser peut augmenter le coût énergétique qui croît avec la complexité des algorithmes de sécurisation. Certains algorithmes sont déjà câblés dans les processeurs (p.ex. AES), il faut les utiliser ! Mais l'absence de sécurisation peut engendrer d'autres coûts (maintenance accrue, remise en service ...) [10.1]

Gestion de version (oui mais...)

J'utilise un outil de gestion de version, mais :

- j'évite ou limite d'y stocker les paquets binaires et les jeux de données non indispensables
- je ne place pas en gestion de version les produits de compilation ni les fichiers de sortie

Intégration continue (oui mais...)

- je réfléchis à mon Intégration Continue (CI). Je choisis un docker de taille minimum, et j'active ma CI uniquement sur certaines branches et j'envisage une exécution programmée. Ainsi je n'exécute pas tous les tests et ne produis pas tous les fichiers à chaque modification
- je surveille la durée des jobs, leur nombre, la taille des artefacts, le trafic réseau
- je privilégie les forges mutualisées

L'écoconception d'un service numérique doit prendre en compte les impacts liés au déploiement et à la production. L'amélioration continue permet de réduire les impacts environnementaux.

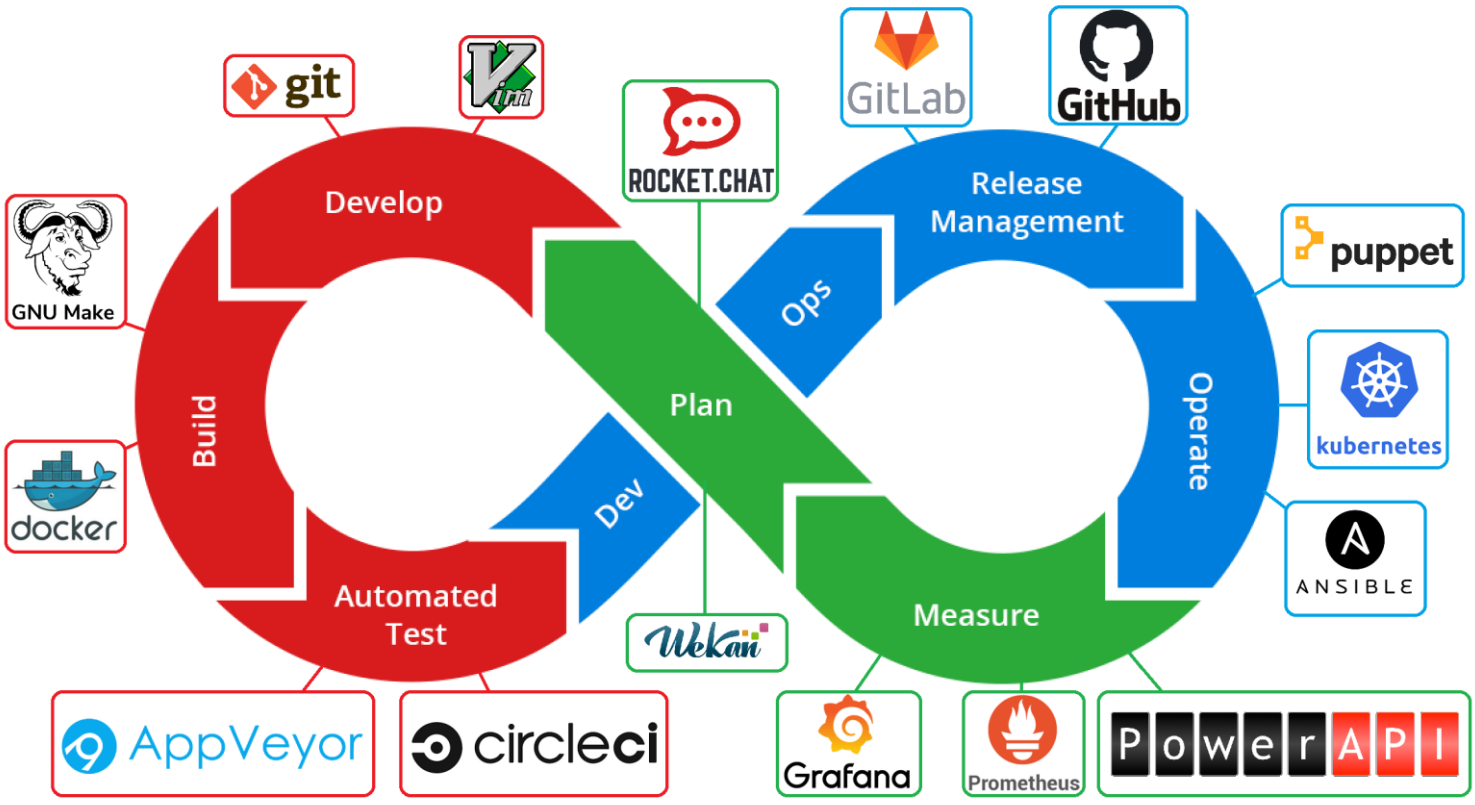
Déploiement : sobriété numérique

- je privilégie un hébergement mutualisé, labélisé COC (Code Of Conduct), au plus près des données et des utilisateurs
- je privilégie la virtualisation (moins de serveurs physiques), en minimisant l'empreinte mémoire et disque (taille des machines virtuelles, des containers), sauf cas particuliers (calcul haute performance)
- je fais attention à certains effets rebond : j'évite la multiplication des machines virtuelles, des instances du service

Production : amélioration continue en fonction des usages

- j'exploite la supervision (et les alertes) pour observer les pics CPU, ressources utilisées (disque, réseau), consommation électrique
- je modifie le service numérique pour l'adapter en fonction de l'usage observé (amélioration continue)
- je réduis les fréquences et volumes des sauvegardes, j'adopte la déduplication

Exemples d'outils de supervision : top, vmstat, zabbix, scalasca, nagios, prometheus, grafana



Exemples d'outils utilisés pour l'amélioration continue du service numérique

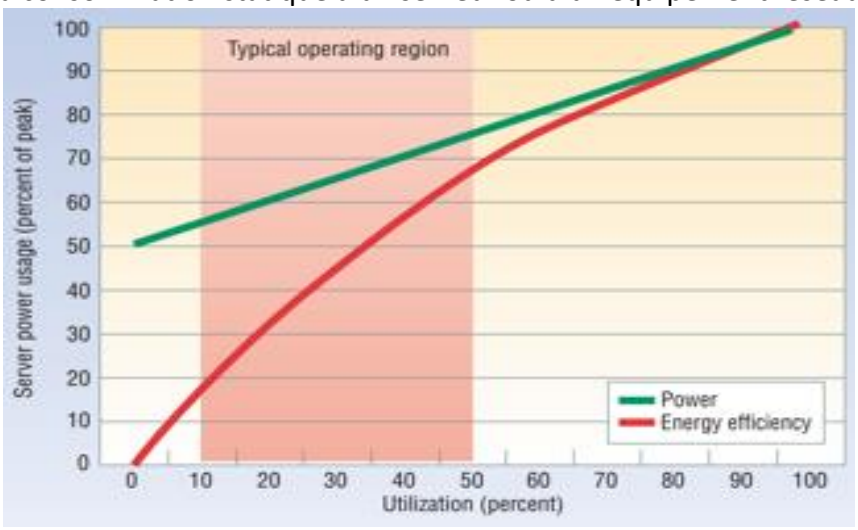
(source : PNGEgg, adaptée par C. Bonamy)

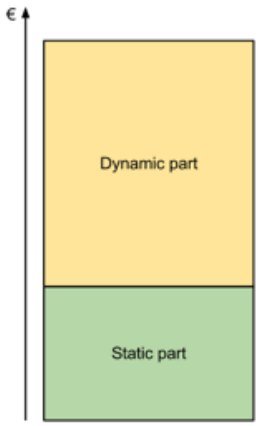
L'écoconception d'un service numérique doit prendre en compte les impacts liés à la distribution, l'utilisation et l'administration du logiciel, sans oublier les besoins des utilisateurs. Il faut s'assurer que l'efficacité énergétique et la réduction des impacts environnementaux sont conservées avec les évolutions du logiciel.

Je traque les ressources inutiles : éviter le gaspillage énergétique

La consommation statique des machines est importante : j'éteins les machines si possible

Le logiciel influence la consommation dynamique des machines, mais il y a peu de proportionnalité énergétique (un serveur inoccupé peut consommer 50 % de son budget électrique).
La consommation statique d'un serveur ou d'un équipement réseau est importante





Luiz André Barroso and Urs Hölzle, "The case for Energy-Proportional Computing", IEEE Computer, 2007 [11.1]

Mise en place de systèmes de mise en veille

- j'éteins les machines virtuelles qui ont une empreinte mémoire et CPU
- je limite le nombre de services déployés
- je favorise les extinctions d'hôtes et de ressources (cœurs de calcul, systèmes de stockage) inutilisés

Je distribue et maintiens mon code : favoriser la durabilité et la simplicité

Diffusion

- je dépose le logiciel en un endroit unique et facilement accessible, une forge.
- je privilégie les publications open source. Tout sur les licences et les démarches : plaquette licence [12.1] et etalab [12.2]

Gestion des mises à jour

- je réduis la taille des produits logiciels
- je rationalise leur nombre et leur fréquence
- rétrocompatibilité autant que possible
- je veille à éviter l'ajout constant de fonctionnalités
- workflow des mises à jour clairement défini
- Long Term Support [12.3] pour les mises à jour correctives

Je sensibilise : améliorer les usages

- en m'appuyant sur ce qui existe :
  - Ecoinfo [12.4]
  - ADEME [12.5]
  - principles green [12.6]
  - Mooc Impacts environnementaux du numérique [12.7]
  - la fresque du numérique [12.8]
- en affichant des informations de suivi de consommation
- en impliquant les utilisateurs de mon service numérique sur les impacts environnementaux de leurs usages

Un service numérique qui n'est plus utilisé va continuer à générer des impacts résiduels sur l'environnement à cause par exemple du stockage de ses données, de son hébergement, des accès réseaux réguliers à d'autres services numériques nécessaires à son fonctionnement, etc. Il est donc plus qu'important de traiter sa fin de vie, comme n'importe quel produit.

J'archive : favoriser la durabilité

Archivage

- déclaration du code source auprès de Software Heritage [13.1]
- je sauvegarde sur un stockage froid les données essentielles (= ayant de la valeur ou une obligation juridique) : traces d'exécutions réglementaires, données spécifiques, documentation, etc.

Je mets hors service : éviter les logiciels et les données « Zombies »

Les utilisateurs

- je préviens l'ensemble des utilisateurs et je m'assure que la fin de vie ne va pas avoir d'effet de bords indésirables : repli sur des solutions plus énergivores, mise en place de processus alternatifs ayant d'autres impacts environnementaux plus important, etc.

La documentation

- je supprime l'ensemble des données concernant le logiciel ... et je signale la fin de vie du logiciel dans l'espace documentaire qui lui était dédié.

Les outils de support

- je mets à jour les configurations des outils permettant le support et les évolutions (files helpdesks, outil de suivi de version, etc.) afin de supprimer toute file active inutile et de nettoyer les données concernant le service numérique arrêté.

Le logiciel

- je supprime l'installation du logiciel sur toutes les instances où il l'a été (environnements de développement/qualification/production, le cas échéant des postes de travail).

Les données

- je supprime les données stockées après archivage : bases de données et fichiers plats.

Les services nécessaires

- j'arrête et je désinstalle les services applicatifs qui n'étaient utiles que pour l'exécution du logiciel (Apache, Nginx, mysql, etc.).

Les machines virtuelles et containers

- j'arrête les containers et machines virtuelles qui n'étaient pas mutualisées avec d'autres services, et suppressions de leurs images. Les arrêter peut permettre d'éteindre une partie de l'infrastructure physique.

Les sauvegardes

- je supprime l'ensemble des sauvegardes journalières/hebdomadaires/mensuelles nécessaires au plan de reprise d'activité.

La supervision

- je supprime les références au logiciel dans les configurations des outils de supervision ainsi que l'ensemble de ses traces d'exécution.

Les dépendances mutualisées

- je vérifie que les éventuelles dépendances mutualisées (bibliothèques logicielles, assets JS ou CSS, etc.) sont encore utile à au moins un autre service numérique, sans quoi je les supprime également.

La cartographie applicative

- je supprime la partie concernant le logiciel en fin de vie dans la cartographie applicative de l'équipe ou de l'établissement afin de conserver une urbanisation du système d'information représentative de la réalité.

J'éco-conçois du code scientifique

Avant de développer :

- je choisis la licence (si possible ouverte afin de faciliter le ré-usage et la reproductibilité)
- je choisis la technologie cible et le langage en fonction des contraintes (C++, Python, Fortran, GPU, Cluster classique, ARM, Cloud, Web services)
- je choisis l'infrastructure adaptée pour chaque phase (laboratoire pour développement, méso-centre pour mise au point et validation, centre de calcul national pour production Ressources ESR [14.1])
- je réfléchis à la gestion des données associées à mon code; j'établis un plan de gestion des données (DMP)
- je conçois l'ensemble du workflow (données d'entrée, traitement, post-traitement, stockage) en pensant aux impacts environnementaux (coût des transferts réseaux...)
- j'alerte les commanditaires et autres développeurs des impacts sur l'environnement du numérique (en m'appuyant sur les instances existantes : EcoInfo)

Au tout début du développement :

- je crée un dépôt (git via un Gitlab universitaire par exemple)
- je crée les fichiers "AUTHORS", "LICENSE", "README »
- je considère des formats ouverts et efficaces pour les données de sortie du logiciel
- je mets en place l'intégration continue (Gitlab pipeline, CircleCI…), sans excès inutiles
- je prévois la création de la documentation et sa mise à jour (Sphinx, Doxygen…)
- j'envisage les éventuels plantages et prévois la possibilité de redémarrage de mes calculs

Pendant le développement :

- je code en choisissant les bonnes bibliothèques (MPI [14.2], OpenMP [14.3])
- je diffuse les sources, partage le dépôt, dès que possible
- j'assure la reproductibilité de mes expériences numériques

Après, pendant les phases de production :

Attention à l'effet rebond

- je recherche les points chauds, à l'aide d'outils existants (gprof, valgrind...)
- je fais attention au poids des I/O (entrées/sorties)
- j'analyse l'extensibilité de mon code (idéalement sur l'infrastructure d'utilisation finale) (Loi d'Amdahl [14.4])
- je me fais aider pour optimiser (exemple : Performance Optimisation and Productivity [14.5]) tout en faisant attention à l'effet rebond (toujours plus!)
- je sensibilise les utilisateurs de mon code sur les impacts environnementaux de leurs calculs
- je recherche le meilleur compromis entre temps de retour, finesse du calcul et coût eqCO2
- je réfléchis au réel intérêt de mes calculs et je n'hésite pas à redévelopper si besoin
- je pense à supprimer les données quand celles-ci ne sont plus utiles

Après, fin de vie :

- je supprime les données non essentielles; je ne conserve que le minimum pour la reproductibilité
- je diffuse mon code (au travers de software heritage [14.6]) en indiquant que le logiciel n'est plus maintenu

Je ne pense pas : "c'est un petit code, ça ne servira qu'à moi, ces bonnes pratiques ne me sont pas destinées".

J'éco-conçois du code sur GPU et/ou IA

Avant de développer :

- je fais attention au poids de l'apprentissage pour l'Intelligence Artificielle
- je n'oublie pas de considérer les alternatives à l'IA dans mon choix d'algorithme
- je sensibilise les utilisateurs au coût des calculs, et notamment au coût global, qui comprend la phase d'apprentissage, cf. article consommation énergétique de l'IA [15.1]
- je choisis la technologie et les contraintes associées en fonction de mes besoins (CUDA vs OpenCL : performance et simplicité vs généricité et verbosité)

Pendant le développement :

- je code en choisissant les bons frameworks (OpenACC [15.2], OpenCL [15.3])
- je ne redéveloppe pas ce qui existe déjà et j'utilise des bibliothèques éprouvées de haut niveau (Tensorflow, PyTorch, scikit-learn)
- la clé des accélérateurs est de réfléchir à uniformiser (pas de conditions (if/else), pas d'hétérogénéité entre les cœurs) et maximiser les traitements sur l'accélérateur (GPU) afin d'éviter les transferts mémoire CPU vers GPU (attention au temps de transferts mémoire CPU/GPU vs kernel time)
- je fais du code portable et performant (attention à trop de spécificités avec un type d'accélérateur, comme par exemple CUDA)
- je suis vigilant à l'impact des entrées/sorties (I/O) qui peut fortement dépendre de l'infrastructure cible (poste de travail vs cluster spécialisé)

Après, pendant les phases de production :

- je sensibilise les utilisateurs au coût des calculs, et notamment au coût consolidé, qui comprend les coûts liés à la fabrication des GPU.

Quelques outils

Sonde logicielle :

- Experiment impact tracker [15.4] (Henderson et al.) : une bibliothèque python qui calcule la consommation d'énergie à partir des informations GPU, CPU et DRAM.
- Carbon tracker [15.5] (Anthony et al.) : également une bibliothèque python qui calcule la consommation d'énergie à partir des informations GPU, CPU et DRAM. Dans le cas de calculs d'IA, il permet d'arrêter l'entraînement d'une expérience quand un coût environnemental fixé est dépassé.
- Code Carbon [15.6] : une troisième bibliothèque python fondée sur le même principe que les précédentes.

Estimation en ligne de la consommation d'énergie et de l'empreinte carbone associées à une expérience numérique :

- Green Algorithms [15.7] (Lannelongue et al.)
- ML CO2 Impact [15.8] (Lacoste et al.)

J'éco-conçois un site Web

Avant de développer :

Je m'imprègne du Référentiel Général d'Ecoconception de Services numériques [16.1] et je les diffuse aux autres participants du projet, notamment les représentants de la maitrise d'ouvrage (MOA, Product Owner) et du design des interfaces (UX/UI). J'invite notamment les designers à s'imprégner du Guide [16.2] réalisé par les designers éthiques. L'écoconception de développement web n'est pas une démarche individuelle, elle nécessite de mobiliser toute l'équipe impliquée dans le projet.
En effet, les gains les plus important sur le développement Web se font :

- en économisant les fonctionnalités, donc en impliquant les donneurs d'ordre et les utilisateurs pour aller vers l'essentiel
- en simplifiant au maximum les interfaces utilisateurs, dans un esprit de sobriété
- en limitant les éléments générant du trafic et de l'utilisation CPU sur les terminaux comme les vidéos, les traceurs, jeux de données, graphiques etc.

D'une manière générale, j'élimine des spécifications tout contenu ou fonctionnalité non-essentiel. Le service doit être utilisable sur un petit écran, et si possible *réactif*. Je proscris l'utilisation de tout type de Dark Pattern [16.3].
Lorsque j'attaque le développement, je vais faire un tour sur le site Les 115 bonnes pratiques [16.4].
Je me préoccupe de la sécurité à toutes les étapes du cycle de vie, et je me familiarise avec les recommandations de l'ANSSI sécurisation siteWeb [16.5].

Pendant le développement :

Je pilote la qualité du développement à partir d'indicateurs d'impacts environnementaux sur les terminaux. Pour ce faire j'utilise si possible un outil de mesure (type Greenspector [16.6] *), ou à défaut d'évaluation de l'impact environnemental de mon site (type GreenIT Analysis [16.7], Ecometer [16.8], Fruggr [16.9]* ou Greenframe [16.10]*). J'ajoute si possible l'appel à ce type d'outils dans ma chaîne d'intégration continue, en complément des outils de développement intégrés à mon navigateur que j'utilise au quotidien.

Je planifie la gestion des données au plus proche du besoin, et j'applique le principe F.A.I.R [16.11] .
J'anticipe la compatibilité et à la maintenabilité sur le long terme. Je privilégie des technologies pérennes maintenues par des communautés actives et établies, pour éviter l'obsolescence du code produit. Je fais attention aux dépendances inutiles de l'application ou des frameworks utilisés.
Je limite les accès réseaux et les transferts de données au strict minimum : je rends ainsi mon service accessible pour des connexions à faible débit, et dans l'idéal résilient aux déconnexions.
J'écris un code simple à comprendre et à mettre en œuvre, il sera aussi plus durable et plus facile à faire évoluer. C'est la base du principe KISS [16.12] : Keep It Simple and Stupid.

Après, pendant les phases de production :

Je choisis un hébergeur labellisé Code of Conduct [16.13], présentant l'indicateur d'efficacité énergétique le plus faible possible et au plus proche géographiquement de mes utilisateurs cible .
Je mets en place une politique de sauvegarde des donnée sobre et je supervise le service (CPU, IO) et les journaux (1an maximumm).
J'anticipe les usages : je suis vigilant à la transformation des comportements potentiellement induits par mon application, et je l'adapte afin de réduire leurs impacts négatifs.

* Outils (partiellement) payants

Comment éco-concevoir des plateformes embarquées et objets connectés ?

- spécifique aux objets connectés : utiliser le mode Deep-Sleep (*) entre chaque mesure
- le codage des nombres a un impact direct sur les performances et donc la consommation. Choisir la bonne représentation (virgule flottante vs fixe) et la bonne taille permet de diminuer la fréquence processeur, voire d'utiliser un processeur moins puissant
- les objets connectés échangent beaucoup de données par internet, à volume de données finales échangées identique certains protocoles ajoutent plus ou moins de données superflues (ex. MQTT, HTTP, XMPP)
- l'architecture logicielle (interruptions vs. attente active, buffers circulaires, optimisation) a un impact direct sur la charge processeur

(*) Deep-Sleep : mode particulier dans lequel le microprocesseur ne consomme plus que quelques µA et peut être réveillé par une interruption

Comment assurer la pérennité d'une appli pour mobile ?

Les recommandations des fiches précédentes s'appliquent également au développement d'applications pour mobiles, en gardant en tête qu'il vaut toujours mieux **éviter** l'installation d'une surcouche logicielle sur le terminal de l'utilisateur, qui pourrait être responsable d'une obsolescence induite sur son matériel.
Je privilégie donc la piste de l'application web plutôt que mobile.
La compatibilité d'un service numérique se construit dans la durée en demeurant utilisable sur les équipements anciens ou exotiques.

Comment assurer la résilience en sécurisant ?

De l'indisponibilité de mon service jusqu'au vol de données personnelles de mes utilisateurs en passant par l'installation de logiciels malveillants sur mon infrastructure, les conséquences d'une faille de sécurité peuvent être nombreuses. Les actions correctives, quand elles sont possibles, sont très coûteuses en temps et en énergie, et donc ont un impact pour l'environnement d'autant plus si l'application est rendue obsolète à cause d'une faille.
La sécurité doit être une préoccupation présente à toutes les étapes du cycle de vie d'un service numérique, et en lister les bonnes pratiques nécessiterait une plaquette à part entière. Je peux commencer par m'inspirer des recommandations de l'Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI [17.1]).

Néanmoins je reste vigilent sur l'impact des mécanismes de sécurité mis en œuvre, notamment sur l'augmentation de la consommation de ressources ou du nombre d'équipements : bien prendre en compte la criticité des systèmes pour ne pas sur-sécuriser inutilement.

A l'extrême

Il existe des technologies particulièrement peu consommatrices en énergie, des « low-techs » poussées à l'extrême :

CollapseOS [17.2] : un système fonctionnant sur des microcontrôleurs
Lowtech Magazine [17.3] : hébergement d'un site web uniquement visible si la production d'énergie solaire est suffisante

Références

[1.1] https://ecoinfo.cnrs.fr
[1.2] https://hal.archives-ouvertes.fr/hal-02083801/file/20191202_plaquette_developpement_V1.1.pdf
[1.3] https://hal.archives-ouvertes.fr/hal-02400300/file/20191202_plaquette_diffusion_V1.1.pdf
[1.4] https://hal.archives-ouvertes.fr/hal-02399517/file/20191202_plaquette_pi_licences_V1.1.pdf

[2.1] https://learninglab.gitlabpages.inria.fr/mooc-impacts-num/mooc-impacts-num-ressources/Partie3/FichesConcept/FC3.4.2-bonnespratiques-MoocImpactNum.html#les-impacts-humains
[2.2] https://www.genci.fr
[2.3] https://www.eco-conception.fr/static/analyse-du-cycle-de-vie-acv.html
[2.4] https://www.eco-conception.fr/static/fonction-unite-fonctionnelle-acv.html

[3.1] https://www.arcep.fr/uploads/tx_gspublication/reseaux-du-futur-empreinte-carbone-numerique-juillet2019.pdf
[3.2] https://www.institutparisregion.fr/fileadmin/NewEtudes/Etude_1806/Full_report_ENERNUM_MAY_2019-eng.pdf
[3.3] https://www.ecoinfo.cnrs.fr
[3.4] https://theshiftproject.org/wp-content/uploads/2020/10/Deployer-la-sobriete-numerique_Rapport-complet_ShiftProject.pdf
[3.5] https://www.greenit.fr/empreinte-environnementale-du-numerique-mondial/
[3.6] https://institutnr.org//inr-numerique-responsable
[3.7] https://ecoresponsable.numerique.gouv.fr/a-propos/
[3.8] https://boavizta.org
[3.9] https://ecoinfo.cnrs.fr/wp-content/uploads/2019/09/Informatique-et-développement-soutenable-LIMSI.pdf
[3.10] https://hal.archives-ouvertes.fr/hal-02549565
[3.11] https://eco-calculateur.dta.aviation-civile.gouv.fr/
[3.12] https://ecolab.ademe.fr/apps/transport

[5.1] https://ll-dd.ch
[5.2] https://dmp.opidor.fr
[5.3] http://www.france-grilles.fr/wp-content/uploads/2018/04/ModeleSMP_PRESOFTV3.2.pdf
[5.4] https://hal.archives-ouvertes.fr/hal-01241196

[6.1] https://app.electricitymap.org/map
[6.1] https://www.ssi.gouv.fr/entreprise/guide/regles-de-programmation-pour-le-developpement-securise-de-logiciels-en-langage-c/
[6.2] https://www.ssi.gouv.fr/guide/regles-de-programmation-pour-le-developpement-dapplications-securisees-en-rust
[6.3] https://www.ssi.gouv.fr/agence/publication/securite-et-langage-java

[7.1] https://arxiv.org/pdf/2009.11295.pdf
[7.2] https://rdcu.be/ciO0J
[7.3] https://github.com/paugier/nbabel
[7.4] https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf
[7.5] https://github.com/greensoftwarelab/Energy-Languages
[7.6] https://hal.archives-ouvertes.fr/hal-03607468

[8.1] https://www.force11.org/group/fairgroup/fairprinciples
[8.2] https://op.europa.eu/en/publication-detail/-/publication/7769a148-f1f6-11e8-9982-01aa75ed71a1/language-en
[8.3] https://www.ouvrirlascience.fr/portail-web-de-leosc/
[8.4] https://www.rd-alliance.org/fair
[8,5] https://view.genial.ly/5d64fbbd8352350fa3d22603
[8.6] https://hal.archives-ouvertes.fr/hal-01241196
[8.7] https://dmp.opidor.fr/
[8.8] http://opendatacanvas.org/sprint-pgd
[8.9] https://www.cnil.fr/fr/comprendre-le-rgpd
[8.10] https:// mi-gt-donnees.pages.math.unistra.fr/site/

[9.1] https://hal.archives-ouvertes.fr/hal-01192692/document

[10.1] http://www.sti.uniurb.it/events/sfm10qapl/slides/katinka_slides.pdf

Références

[12.1] https://hal.archives-ouvertes.fr/hal-02399517/file/20191202_plaquette_pi_licences_V1.1.pdf
[12.2] https://www.etalab.gouv.fr/accompagnement-logiciels-libres
[12.3] https://fr.wikipedia.org/wiki/Long-term_support
[12.4] https://ecoinfo.cnrs.fr
[12.5] https://www.ademe.fr/sites/default/files/assets/documents/guide-pratique-face-cachee-numerique.pdf
[12.6] https://principles.green
[12.7] https://www.fun-mooc.fr/fr/cours/impacts-environnementaux-du-numerique/
[12.8] https://www.fresquedunumerique.org/

[13.1] https://www.softwareheritage.org/

[14.1] https://www.edari.fr
[14.2] https://fr.wikipedia.org/wiki/Message_Passing_Interface
[14.3] https://fr.wikipedia.org/wiki/OpenMP
[14.4] https://fr.wikipedia.org/wiki/Loi_d'Amdahl
[14.5] https://pop-coe.eu
[14.6] https://www.softwareheritage.org/

[15.1] https://ecoinfo.cnrs.fr/2021/06/12/consommation-energetique-de-lutilisation-de-lia/
[15.2] https://en.wikipedia.org/wiki/OpenACC
[15.3] https://en.wikipedia.org/wiki/OpenCL
[15.4] https://github.com/Breakend/experiment-impact-tracker
[15.5] https://github.com/lfwa/carbontracker
[15.6] https://codecarbon.io/
[15.7] http://www.green-algorithms.org/
[15.8] https://mlco2.github.io/impact/#compute

[16.1] https://ecoresponsable.numerique.gouv.fr/publications/referentiel-general-ecoconception/
[16.2] https://eco-conception.designersethiques.org/guide/fr/
[16.3] https://fr.wikipedia.org/wiki/Dark_pattern
[16.4] https://collectif.greenit.fr/ecoconception-web/115-bonnes-pratiques-eco-conception_web.html
[16.5] https://www.ssi.gouv.fr/guide/recommandations-pour-la-securisation-des-sites-web/
[16.6] https://greenspector.com/fr/accueil/
[16.7] https://www.greenit.fr/2019/07/02/web-evaluez-lempreinte-dune-page-en-un-clic/
[16.8] http://www.ecometer.org/
[16.9] https://www.fruggr.io/fr/
[16.10] https://greenframe.io/
[16.11] https://fr.wikipedia.org/wiki/Fair_data
[16.12] https://fr.wikipedia.org/wiki/Principe_KISS
[16.13] https://ecoinfo.cnrs.fr/2020/05/19/guide-des-bonnes-pratiques-du-code-de-conduite-europeen-sur-les-datacentres/

[17.1] https://www.ssi.gouv.fr
[17.2] http://colapseos.org/
[17.3] https://solar.lowtechmagazine.com/fr/about.html