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, 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 : IAS / CNRS
Laurent Lefèvre : Inria / ENS Lyon
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

Contexte

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 donc 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

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, mésocentre régional, GENCI [2.1], 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 :

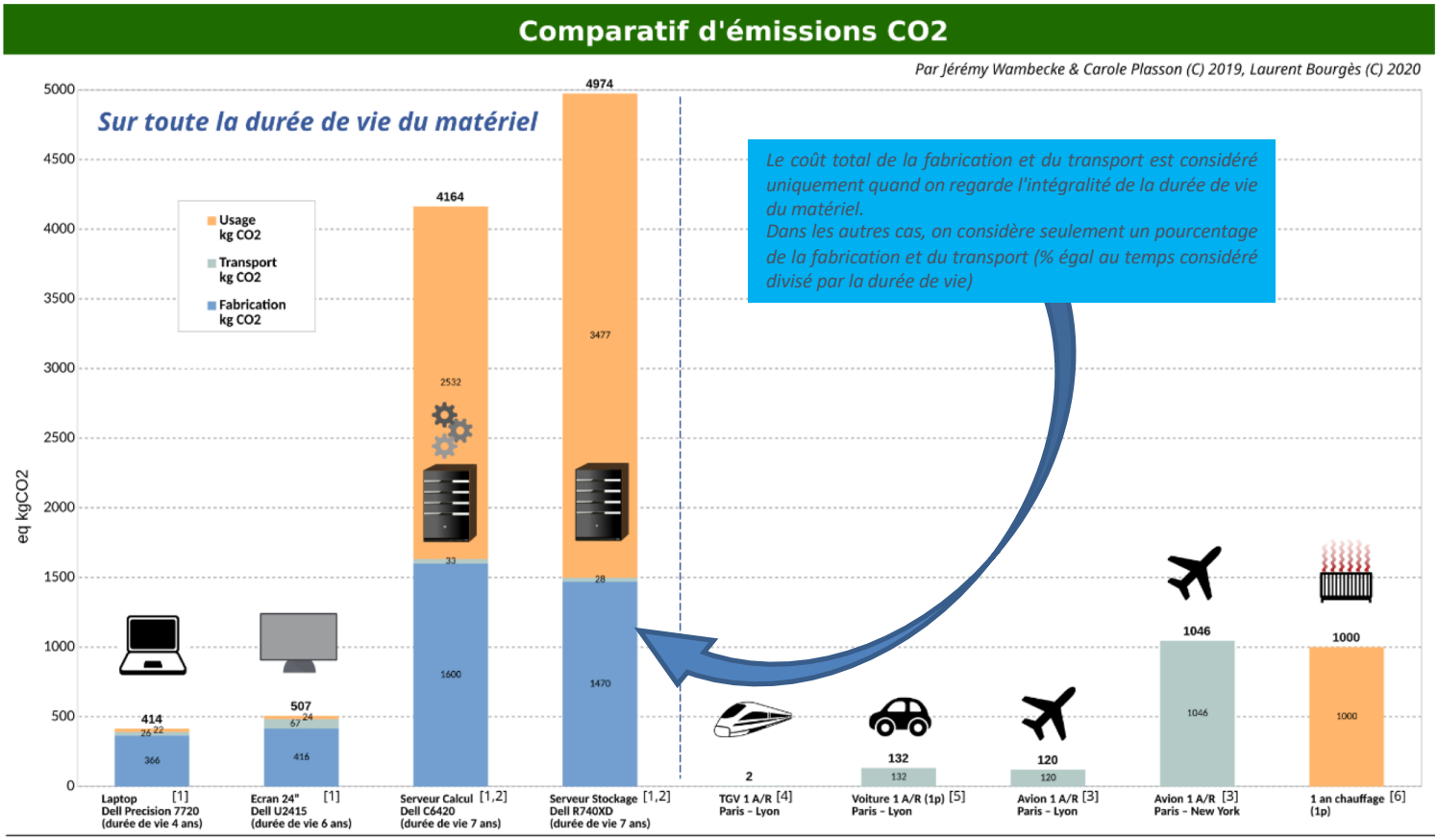
- limiter les transferts de données
- choix plateforme de calcul et de pré et post-traitement
- post-traiter les données au plus proche du lieu de création
- limiter les données d'entrée et/ou de sortie
- choisir des briques logicielles externes ou à redévelopper en interne

Eco-concevoir un service numérique, c'est intégrer les aspects environnementaux tout au long de son cycle de vie [2.2] (attention à bien définir l'unité fonctionnelle [2.3])

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

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])

Comparatif d'émissions CO2 du numérique



[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 noeud = 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-co2-tgv/table/
[5] Trajet de 473km, pour une voiture émettant 140g CO2/km
[6] https://www.insee.fr/fr/statistiques/fichier/1281320/ip1445.pdf
Facteur d'impact : 0,108 kgCO2e/kWh (FR)

- le logiciel est au coeur 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.

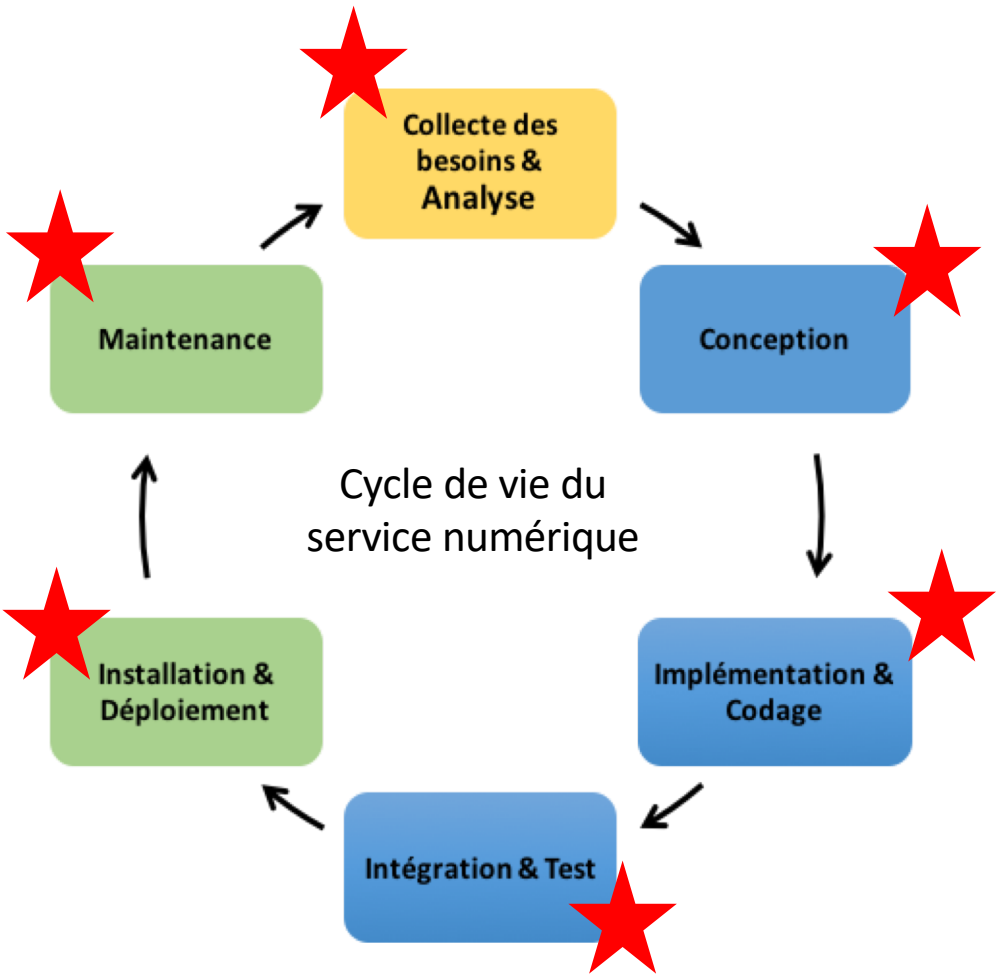
Exemple d'impact dans le milieu de la recherche : 5 millions d'heures.coeur de calcul (env. 25 tonnes eq CO2 [3.6]) sont équivalents à 25 Allers-Retours Paris - New York en avion selon l'aviation civile [3.7] ou 11 selon Ecolab [3.8].

L'écoconception d'un service numérique doit permettre de réduire les ressources nécessaires pour répondre à un besoin.

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 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 .

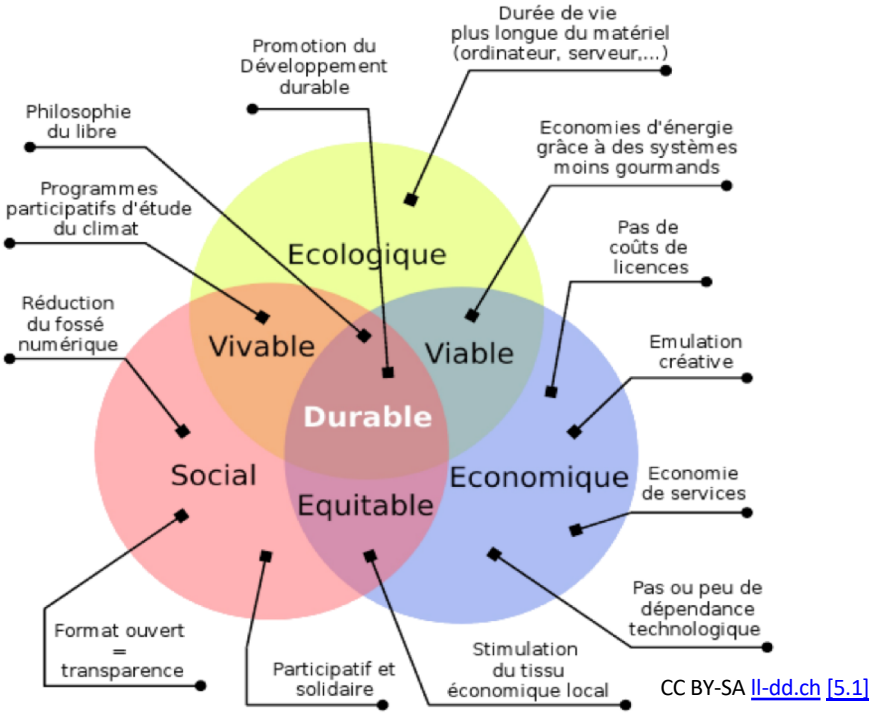
Je maitrise le nombre de fonctionnalités logicielles : éviter l'obésiciel

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 corréler le choix d'un langage et de la plateforme de déploiement tout en considérant les contraintes globales du service numérique complet.

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

En fonction des caractéristiques des plateformes disponibles :

- microcontrôleur ou processeur embarqué
- ordinateur ou serveur local
- plateforme spécialisée : Cluster spécialisé HPC, GPU, FPGA, ARM64
- offre De Service (ODS) de site ou de tutelles (CNRS, Universités)
- clouds publics
- lieu géographique de la plateforme
- capacités disponibles

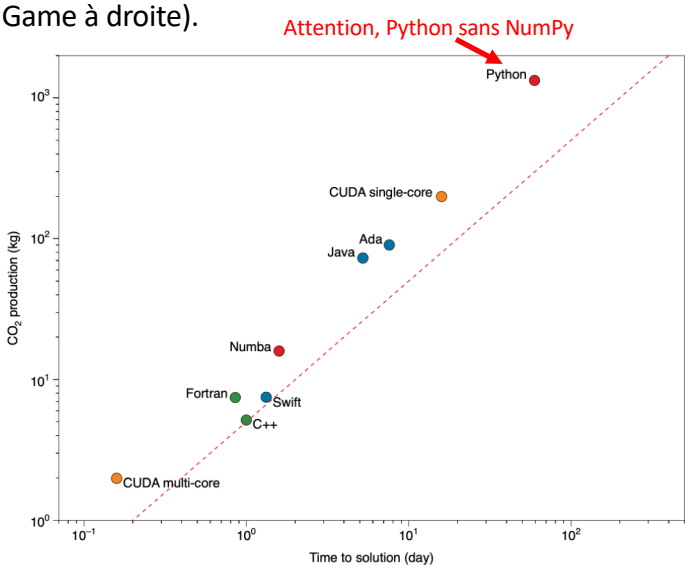
En fonction des contraintes du service :

- 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

Sans oublier les contraintes environnementales …

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

Illustrations de l'efficacité énergétique en fonction du langage (Simon P. Zwart, Nature Astronomy [6.1]) avec une simulation N-Body à gauche, et Rui Pereira et al, SLE17 [6.2] [6.3] basé sur le Computer Langage Benchmark Game à droite).



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		

- 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
- pour les langages interprétés, les performances peuvent être grandement améliorées par l'utilisation de libraires compilées (p.ex. NumPy pour python)
- paralléliser les tâches autant que possible
- 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 ?

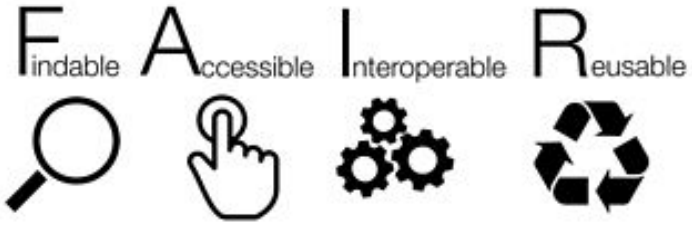
Le format texte (ASCII brut, XML, YAML...) nécessite généralement plus de place pour stocker la même information. Par contre lorsqu'on versionne les fichiers, le phénomène peut s'inverser (stockages successifs vs. stockage différentiel). Il y a un risque d'obsolescence à l'utilisation des formats binaires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

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



source :SangyaPundir CC BY-SA 4.0

- **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 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 [7.1] / directive européenne [7.2] / European Open Science Cloud [7.3] / Research Data Alliance [7.4]

Les objectifs souvent cités d'un plan de gestion de données (DMP) [7.5] de la recherche sont les suivants : assurer la reproductibilité des données, 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. Exemples de modèles de DMP sur le site OPIDOR [7.6]

Voir aussi l'Atelier des données [7.7]

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'execution
- consommation électrique
- consommation des protocoles réseau (par exemple dans l'IOT)
- taille des données
- nombre de requêtes
- performance web

Exemples d'outils : FireFox web tools (trafic, requêtes, JavaScript), Apache JMeter (scénarios web), ecoindex (web), profiler intégré et adapté au langage.

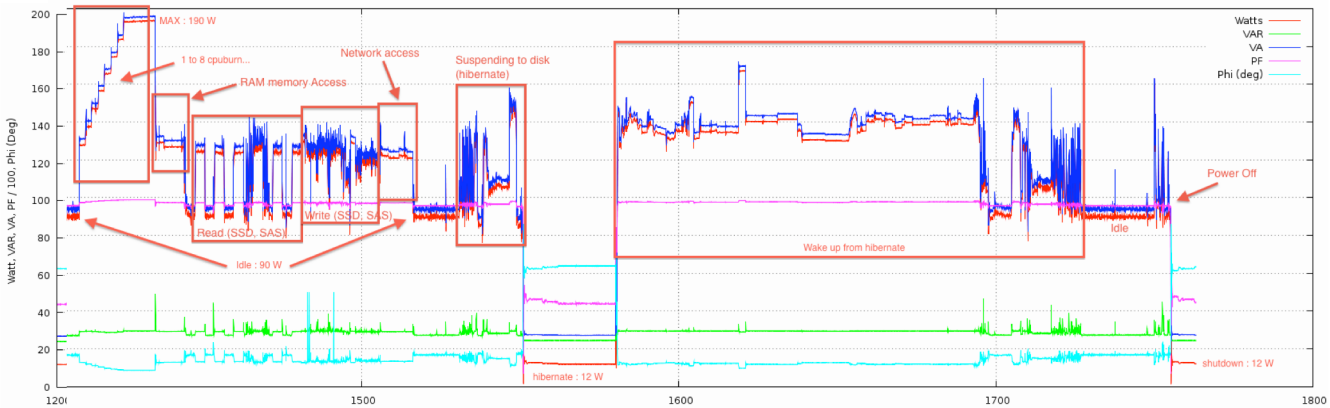
Un focus : je profile la consommation énergétique des logiciels

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.

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

Wattmètre Watts'up Pro - PDU Eaton



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 ...) [9.1]

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.

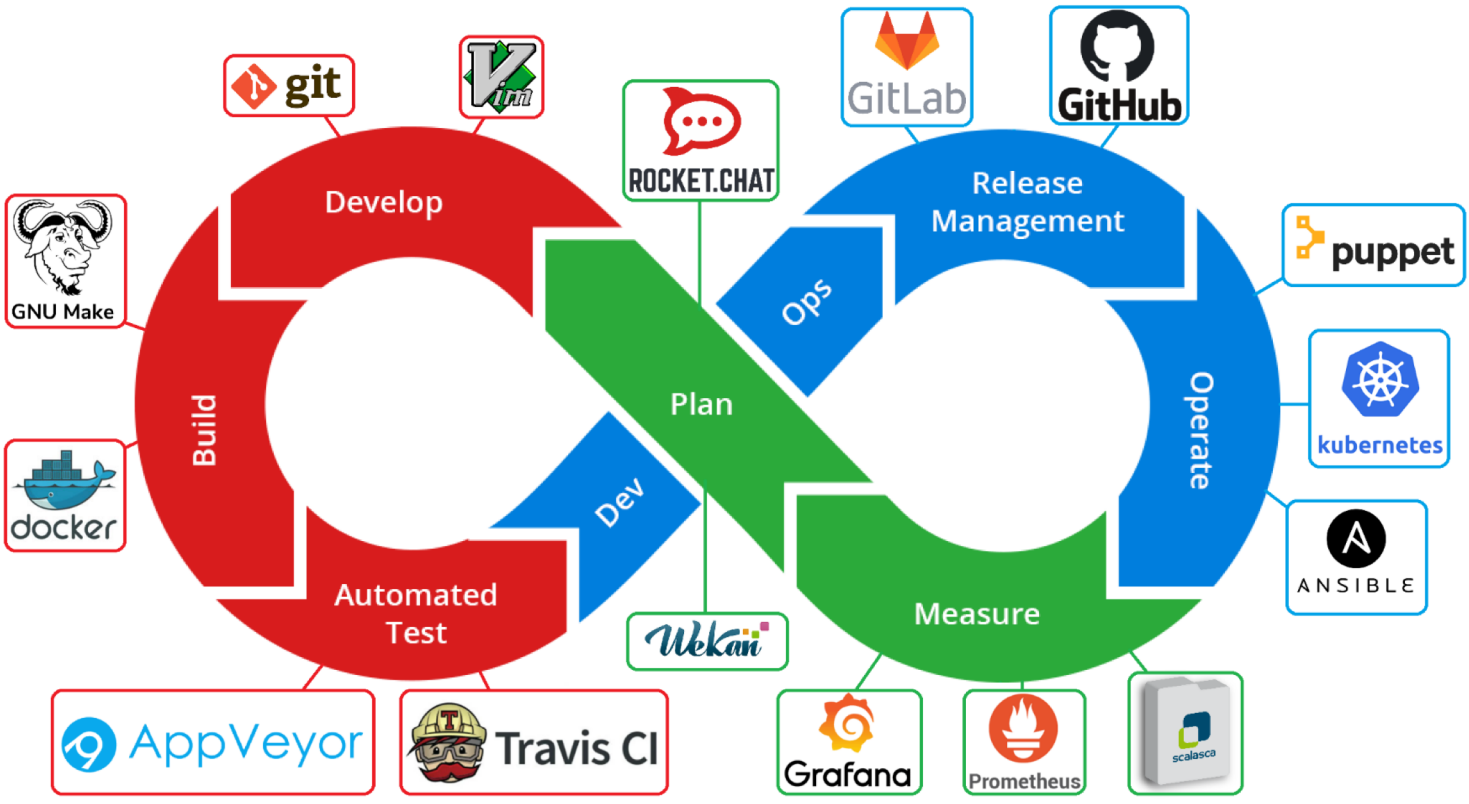
Déploiement : sobriété numérique

- 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

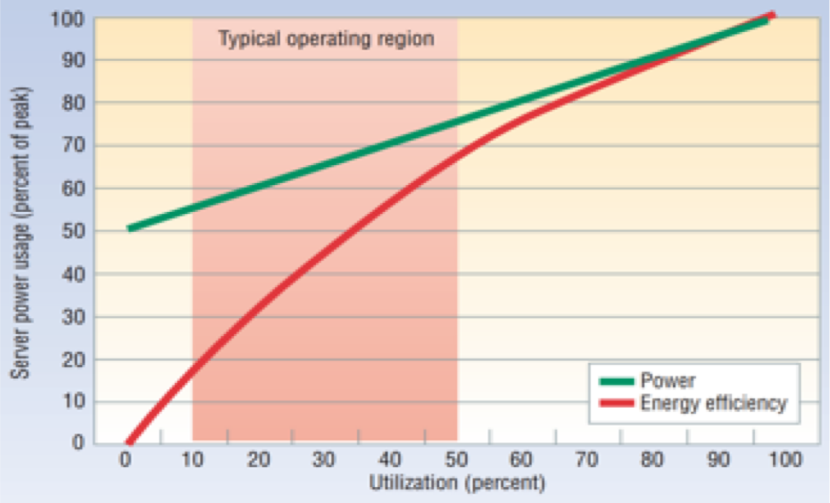
(source : PNGEgg, adaptée par C. Bonamy)

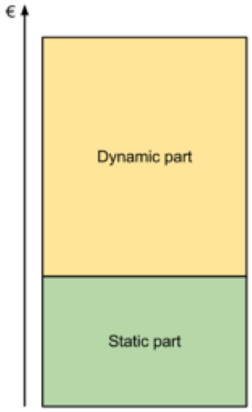
L'écoconception d'un service numérique doit prendre en compte les impacts liés à la distribution, l'utilisation et l'administration du logiciel, sans oublier les besoins des utilisateurs. Il faut s'assurer que l'efficacité énergétique et la réduction des impacts environnementaux sont conservées avec 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
- déclaration auprès de Software Heritage [11.2]

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 [11.3] pour les mises à jour correctives

Je sensibilise : améliorer les usages

- en m'appuyant sur ce qui existe :
  - Ecoinfo [11.4]
  - ADEME [11.5]
  - principles green [11.6]
- en affichant des informations de suivi de consommation
- je sensibilise les utilisateurs de mon service numérique sur les impacts environnementaux de leurs usages

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)
- plus généralement, 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 mets en place l'intégration continue (Gitlab pipeline, Travis-CI…), 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 diffuse les sources, partage le dépôt, dès que possible
- j'assure la reproductibilité de mes expériences numériques

Après, pendant les phases de production :

- 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 la scalabilité de mon code (idéalement sur l'infrastructure d'utilisation finale)
- je me fais aider pour optimiser (exemple : Performance Optimisation and Productivity [12.1]) tout en faisant attention aux effets 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

Quelques liens intéressants :

- Loi d'Amdahl [12.2]
- MPI [12.3], OpenMP [12.4], OpenACC [12.5], OpenCL [12.6]
- Comment obtenir des ressources (calcul et stockage) dans l'ESR? [12.7]

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

Comment éco-concevoir des plateformes mobiles et objets connectés ?

- spécifique aux objets connectés : utiliser le mode Deep-Sleep (*) entre chaque mesures
- 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 de downsizer le processeur
- 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 éco-concevoir mon développement Web ?

Je vais faire un tour sur ce site : Les 115 bonnes pratiques [13.1]

S'il fallait en retenir trois :
- minimiser le nombre de requêtes et le volume de données
- réduire l'impact du javascript (poste client)
- réduire la taille et le nombre des ressources web (images, styles, scripts, documents …)

Je teste mon site web avec Ecoindex [13.2] et Ecometer [13.3]
Autres outils de mesure : FireFox web tools (traffic, requetes, JS), Apache JMeter (scénarios web) (pour l'amélioration continue).

J'éco-conçois du code sur un accélérateur (GPU)

- je fais du code portable et performant (attention à trop de spécificités avec un type d'accélérateur, comme par exemple CUDA)
- je choisis la technologie et les contraintes associées en fonction de mes besoins (Cuda vs OpenCL : performance et simplicité vs généricité et verbosité)
- 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 suis vigilant à l'impact des entrées/sorties (I/O) qui peut fortement dépendre de l'infrastructure cible (poste de travail vs cluster spécialisé)
- je fais attention au poids de l'apprentissage pour l'Intelligence Artificielle
- je sensibilise les utilisateurs au coût des calculs, et notamment au coût global (qui comprend la phase d'apprentissage)

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://www.genci.fr
[2.2] https://www.eco-conception.fr/static/analyse-du-cycle-de-vie-acv.html
[2.2] 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://hal.archives-ouvertes.fr/hal-02549565
[3.7] https://eco-calculateur.dta.aviation-civile.gouv.fr/
[3.8] 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-univ-paris.archives-ouvertes.fr/UNIV-PARIS5/page/data-management-plan

[6.1] https://arxiv.org/pdf/2009.11295.pdf
[6.2] https://greenlab.di.uminho.pt/wp-content/uploads/2017/10/sleFinal.pdf
[6.3] https://github.com/greensoftwarelab/Energy-Languages

[7.1] https://www.force11.org/group/fairgroup/fairprinciples
[7.2] https://op.europa.eu/en/publication-detail/-/publication/7769a148-f1f6-11e8-9982-01aa75ed71a1/language-en
[7.3] https://www.ouvrirlascience.fr/portail-web-de-leosc/
[7.4] https://www.rd-alliance.org/fair
[7.5] https://hal-univ-paris.archives-ouvertes.fr/UNIV-PARIS5/page/data-management-plan
[7.6] https://dmp.opidor.fr/
[7.7] https://mi-gt-donnees.pages.math.unistra.fr/site/

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

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

[11.1] https://www.barroso.org/publications/ieee_computer07.pdf
[11.2] https://www.softwareheritage.org/save-and-reference-research-software/?lang=fr
[11.3] https://fr.wikipedia.org/wiki/Long-term_support
[11.4] https://ecoinfo.cnrs.fr
[11.5] https://www.ademe.fr/sites/default/files/assets/documents/guide-pratique-face-cachee-numerique.pdf
[11.6] https://principles.green

[12.1] https://pop-coe.eu
[12.2] https://fr.wikipedia.org/wiki/Loi_d'Amdahl
[12.3] https://fr.wikipedia.org/wiki/Message_Passing_Interface
[12.4] https://fr.wikipedia.org/wiki/OpenMP
[12.5] https://en.wikipedia.org/wiki/OpenACC
[12.6] https://en.wikipedia.org/wiki/OpenCL
[12.7] https://www.edari.fr

[13.1] https://collectif.greenit.fr/ecoconception-web/115-bonnes-pratiques-eco-conception_web.html
[13.2] http://www.ecoindex.fr
[13.3] http://www.ecometer.org