Concevoir des applications Java plus durables

Découvrez comment Quarkus peut réduire l'empreinte carbone et favoriser la durabilité

Sommaire

facebook.com/redhatinc
@RedHatFrance
linkedin.com/company/red-hat

Synthèse

- Le secteur des logiciels doit impérativement réduire son impact environnemental. Des solutions existent et peuvent être mises en place par les équipes de développement et de conception des applications.
- L'amélioration de l'efficacité est une solution dite « sans regret » qui permet à la fois de réduire les émissions de carbone et les coûts du cloud, et d'améliorer l'expérience utilisateur.
- Quarkus réduit considérablement l'utilisation des ressources par les applications Java(TM), tout en augmentant la productivité de l'équipe de développement. On peut ainsi s'attendre à une baisse de la consommation d'énergie par deux ou par trois par rapport aux frameworks cloud-native existants.

Durabilité et secteur informatique

Une durabilité accrue présente de nombreux avantages socioéconomiques. Outre la réduction des coûts, cela permet d'attirer des clients et de se différencier auprès des futurs candidats à l'embauche. Dans un monde de plus en plus en proie aux effets du changement climatique, c'est aussi gratifiant.

Dans le secteur de l'informatique, à l'impact environnemental démesuré, la durabilité est un sujet particulièrement pertinent. En plus de consommer de l'énergie issue en grande partie de combustibles fossiles, le secteur des TIC exploite également de précieuses ressources abiotiques telles que l'eau et les minéraux. Enfin, les équipements informatiques polluent souvent au début de leur cycle de vie, lors de leur fabrication, et génèrent des déchets électroniques en fin de vie.

Émissions de carbone issues des logiciels

Le changement climatique est un problème urgent, qui appelle une action rapide des particuliers, des entreprises et des gouvernements. Selon le sixième rapport d'évaluation du GIEC publié en 2022, une réduction des émissions de gaz à effet de serre (GES) de 45 % est nécessaire d'ici 2030 pour limiter le réchauffement de la planète à 1,5 °C au-dessus des niveaux préindustriels.

Au vu de la consommation mondiale d'énergie des ordinateurs, le secteur des TIC est un acteur essentiel dans la réduction des émissions. D'après une étude, le monde numérique consommerait 5,5 % de l'électricité mondiale et serait responsable de 3,8 % des émissions de GES associées. C'est nettement plus que l'aviation, qui représente 1,9 % des émissions mondiales de GES.

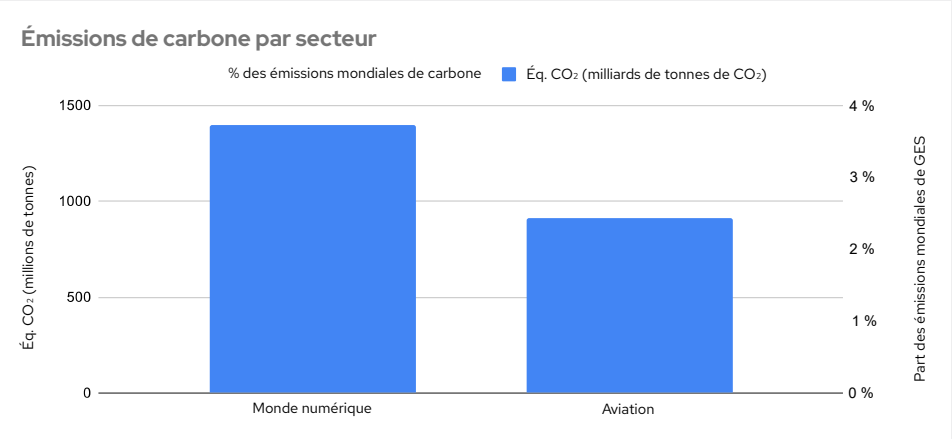


Figure 1 : comparaison des émissions par secteur[1][2]

Ici, le concept de « monde numérique » englobe les réseaux, les datacenters et les appareils des utilisateurs finaux. Qu'en est-il des datacenters pris séparément ?

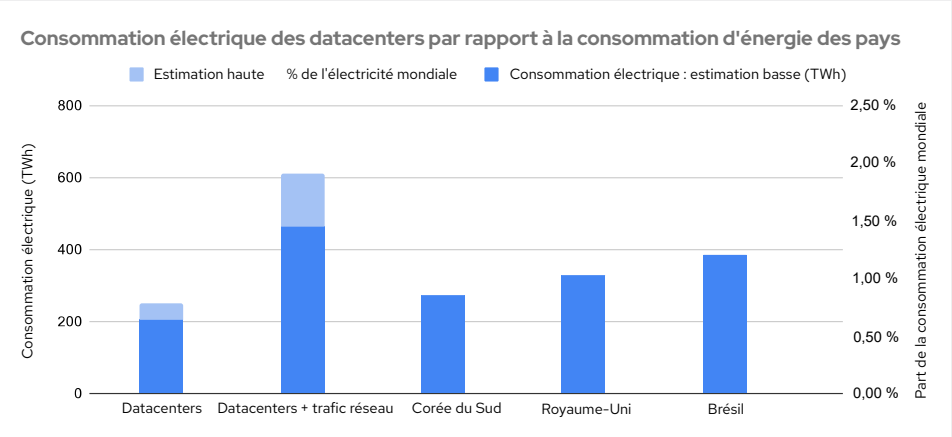


Figure 2 : comparaison de la consommation d'électricité[3][4]. La zone plus claire représente les estimations hautes, le cas échéant.

1 *« Empreinte environnementale du numérique mondial », GreenIT.fr, novembre 2019*

2 *Hannah Ritchie, « Sector by sector: where do global greenhouse gas emissions come from? », Our World in Data, 18 sept. 2020*

3 *« Data centres & networks », IEA, 2022*

4 *« Tableau « Electricity demand », Our World In Data, consulté en octobre 2022*

Au niveau mondial, la consommation d'énergie des datacenters est généralement estimée entre 200 et 250 TWh, sans même tenir compte du minage des cryptomonnaies. Cela équivaut à la consommation d'un pays de taille moyenne, comme le Brésil ou le Royaume-Uni. Le chiffre de 200 TWh fait abstraction des besoins énergétiques du trafic réseau des datacenters. En incluant ces besoins dans le calcul, la consommation d'électricité liée aux données et aux datacenters atteint environ le double de celle de la Corée du Sud.

Si la production d'électricité bas carbone est possible, elle est loin d'être la norme : l'électricité est encore en grande partie issue de la combustion d'énergies fossiles. C'était le cas de 62 % de l'électricité produite en 2021.

Ingénierie logicielle durable

Les principes de l'ingénierie logicielle durable offrent un cadre de réflexion sur la durabilité des logiciels. L'impact carbone d'un logiciel provient de trois sources :

- L'hébergement
- Le logiciel proprement dit
- Le comportement des utilisateurs

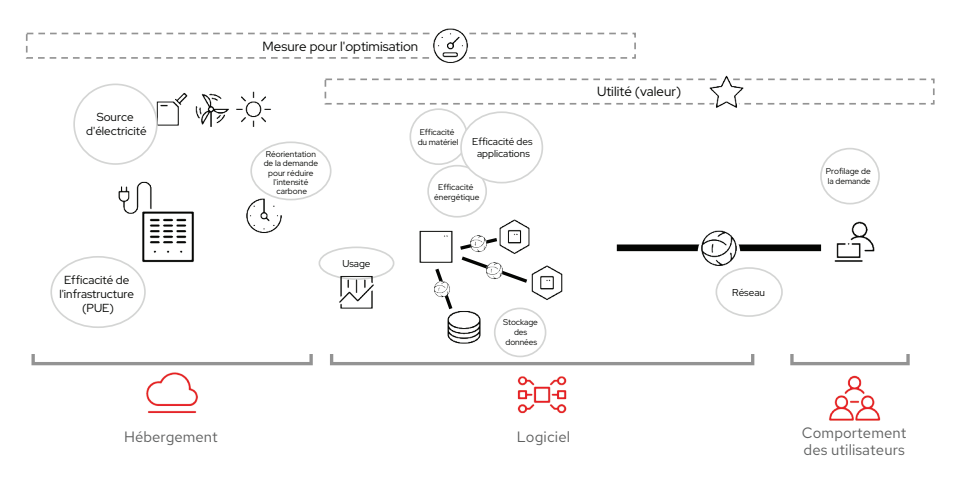


Figure 3 : présentation des principes de l'ingénierie logicielle durable[5]

Hébergement

Le processus d'exécution et l'environnement d'exécution d'une application déterminent une grande partie de son empreinte carbone.

Source d'électricité

Il existe plusieurs modes de production de l'électricité. Une partie de l'électricité provient de la combustion d'énergies fossiles, comme le charbon et le gaz, tandis qu'une autre est issue de sources renouvelables comme l'énergie hydraulique, solaire et éolienne. Le type d'électricité dont nous disposons dépend du lieu d'hébergement, des conditions météorologiques et de l'heure de la journée. Différentes sources d'électricité peuvent coexister selon les régions en fonction de l'infrastructure locale. Par exemple, dans l'État américain de Virginie qui abrite de nombreux

5 *Adaptée des « Principes de l'ingénierie logicielle durable », Principles.Green, consultés en octobre 2022*

datacenters, le bouquet énergétique est fortement dominé par le charbon, alors que dans de nombreux pays nordiques, les énergies renouvelables comme l'éolien et le solaire sont davantage présentes.

De nombreux fournisseurs de cloud publient désormais des informations sur le profil carbone de leurs régions d'activités, ce qui permet aux consommateurs de faire des choix réfléchis quant aux émissions de carbone.

Efficacité de l'infrastructure

L'efficacité de l'infrastructure influe également sur l'empreinte carbone de l'hébergement. Il s'agit d'un facteur important, en particulier lorsque l'on compare des bouquets énergétiques équivalents. En effet, un datacenter inefficace qui fonctionne à l'électricité décarbonée émet moins de carbone qu'un centre efficace dépendant des énergies fossiles.

Le point positif est que l'efficacité des datacenters s'est considérablement améliorée au cours des 20 dernières années. Grâce aux innovations dans les technologies de refroidissement, aux économies d'échelle et à la réduction des déchets, certains fournisseurs ont pu obtenir de très bons indicateurs d'efficacité énergétique (PUE ou Power Usage Effectiveness). Le PUE mesure le nombre d'unités d'énergie qui doivent être dédiées par unité de calcul dans une installation. Un PUE de 1,0 correspond à une valeur idéale et pourrait être obtenu par un datacenter qui ne présente aucune surcharge. Un datacenter classique d'entreprise présente un PUE compris entre 1,6 et 1,8, alors que certains hyperscalers atteignent une valeur de 1,1.

Logiciel

Efficacité des applications

En général, l'efficacité des applications est le facteur générateur d'émissions de carbone que les professionnels de l'informatique maîtrisent le plus. Leur travail consiste à optimiser les applications de manière à limiter les besoins en énergie et en ressources matérielles.

Stockage des données

Les données jouent un rôle significatif lorsqu'il est question d'efficacité. Le stockage des données présente un coût environnemental, et stocker plus de données que nécessaire peut avoir un impact négatif sur l'empreinte carbone.

Réseau

Tout comme les données, le trafic réseau est un aspect moins visible de l'empreinte carbone. Un réseau est constitué de nombreux ordinateurs (commutateurs, routeurs, serveurs, etc.) qui consomment de l'énergie et sont fabriqués à partir de carbone. L'objectif consiste à réduire le volume du trafic réseau entrant et sortant des systèmes, ainsi que la distance parcourue par les données.

Comportement des utilisateurs

Le niveau d'activité d'un système logiciel dépend de l'utilisation que ses utilisateurs en font. Les développeurs d'applications devraient aider les utilisateurs à adopter des écogestes, par exemple via des options simples comme un mode sombre, ou plus complexes comme un système de récompense s'ils utilisent le logiciel en période creuse. Le traitement par lots des demandes, les écomodes et la diminution de la bande passante sont autant de leviers qui permettent aux utilisateurs de réduire leurs émissions de carbone.

Considérations transversales

Utilité

Avez-vous déjà regardé un clip vidéo alors que vous vouliez seulement écouter la chanson ? Ou laissé une tâche CI/CD s'exécuter pendant des heures, sans réel objectif ? Les logiciels nous permettent de réaliser des tâches d'une très grande utilité, mais l'inverse est également vrai, et ce à grande échelle.

Mesure

Il est impossible d'optimiser ce qu'on ne peut mesurer. C'est pourquoi les entreprises doivent mesurer l'empreinte carbone de leur parc logiciel, tout comme elles mesurent les temps de réponse et les taux d'erreur. La mesure des émissions de carbone est une tâche complexe sur laquelle nous reviendrons plus tard.

Solutions de décarbonation

Solutions « sans regret »

Certaines techniques de réduction des émissions de carbone entraînent de l'inconfort ou une absence totale de confort, mais ce n'est pas une généralité. Il existe en effet des solutions « gagnant-gagnant », ou ce que le projet Drawdown appelle des solutions sans regret, qui apportent des « co-bénéfices ».

Pour prendre un exemple simple, un fournisseur de cloud a récemment indiqué qu'exécuter des charges de travail dans son datacenter de Montréal entraînait 88 % d'émissions de carbone en moins qu'à Londres. Rien de très étonnant, si ce n'est que l'hébergement coûte également 15 % moins cher à Montréal. Qu'en est-il de la latence transatlantique ? L'hébergement en Finlande génère 43 % d'émissions en moins que l'hébergement à Londres et est également 15 % moins cher. À moins qu'il n'y ait d'autres contraintes, telles que la latence ou la localisation des données, le choix d'une région à faible coût et à faible émission de carbone pour héberger un logiciel est avantageux.

Hormis ce cas particulier, de nombreuses solutions visant une réduction des émissions de carbone présentent ce double avantage d'offrir des co-bénéfices (baisse des coûts, amélioration de l'expérience utilisateur ou autres), en plus de limiter les émissions.

Arrêt des serveurs zombies

Il est étonnant de constater qu'une partie de l'électricité des datacenters est utilisée par des applications qui ne sont pas actives. Surnommées « serveurs zombies », ces applications n'ont plus aucune utilité et devraient être définitivement mises hors service, mais personne ne s'en occupe. Selon des estimations, environ 30 % des serveurs en cours d'exécution sont laissés à l'abandon et n'ont plus d'utilité.

Modèle LightSwitchOps

« Selon une estimation, en 2021, 26,6 milliards de dollars de dépenses dans le cloud public ont été gaspillés pour des charges de travail inactives. »

Kathy Stalcup
« Overprovisioning & Always-On Resources Lead to $26.6 Billion in Public Cloud Waste Expected in 2021 », *Business 2 Community*, 21 janv. 2021

En théorie, aucun système informatique ne devrait être « toujours actif » à moins qu'il ne soit réellement nécessaire. Une application inutilisée ne peut pas apporter de valeur. L'élimination de ces applications inactives pourrait réduire considérablement les émissions de GES, et ce sans poser d'inconvénients. Bien au contraire, cela entraînerait une diminution des coûts. L'utilisation d'un simple script pour éteindre les serveurs la nuit et les rallumer le matin peut permettre de réaliser environ 30 % d'économies.

Bien qu'aucune charge de travail ne devrait être active en permanence, beaucoup le sont dans les faits. Parce qu'elles ne sont pas à l'aise avec les techniques de mise à l'échelle, des entreprises peuvent exécuter en continu des applications utilisées seulement pendant les heures de bureau. Pourquoi un tel manque de confiance ? Le problème sous-jacent est souvent un manque de flexibilité. La flexibilité est la capacité à faire évoluer les ressources. Imaginez un interrupteur. Vous l'utilisez pour éteindre les lumières lorsque vous quittez une pièce et les rallumer quand vous revenez. La façon dont la lumière s'allume et s'éteint presque instantanément reflète la grande flexibilité d'une ampoule.

Le modèle LightSwitchOps décrit les architectures d'automatisation et cloud-native capables de prendre en charge des opérations fréquentes de mise à l'échelle. Le manque d'outils pour détecter les situations de charge faible ou élevée et s'y adapter freine l'adoption plus large du LightSwitchOps. On observe toutefois une évolution rapide dans ce domaine, et l'émergence de

plusieurs options permettant de gérer les instances avec mise à l'échelle automatique et même les clusters. De nouvelles normes et solutions devraient continuer à voir le jour. L'adoption des techniques DevOps, et même des workflows GitOps, peut également aider les entreprises à gagner suffisamment en confiance pour désactiver temporairement des applications.

Le LightSwitchOps apporte une solution à la fois au problème des serveurs qui doivent être redimensionnés lorsqu'ils ne sont pas utilisés, et des serveurs définitivement oubliés et abandonnés. Lorsqu'il est facile de mettre en route et d'arrêter un système, vous êtes moins tenté de le laisser tourner quand vous ne l'utilisez plus.

Modèle LightSwitchOps et Java

Le langage Java n'est pas vraiment comparable à un interrupteur. Par rapport à d'autres langages comme Go et Rust, une application Java met du temps à se lancer. Cela ne pose pas de problème pour une application à longue durée de vie, mais devient problématique pour les applications à courte durée de vie ou les déploiements serverless.

La bonne nouvelle pour la communauté Java, c'est que les choses sont en train d'évoluer : certaines applications Java affichent les mêmes performances que les interrupteurs, et sont même plus rapides. Une application REST simple native pour Quarkus démarre (et donne la première réponse) six fois plus rapidement qu'une ampoule LED de qualité.

Une application simple native pour Quarkus démarre plus rapidement qu'une ampoule LED.

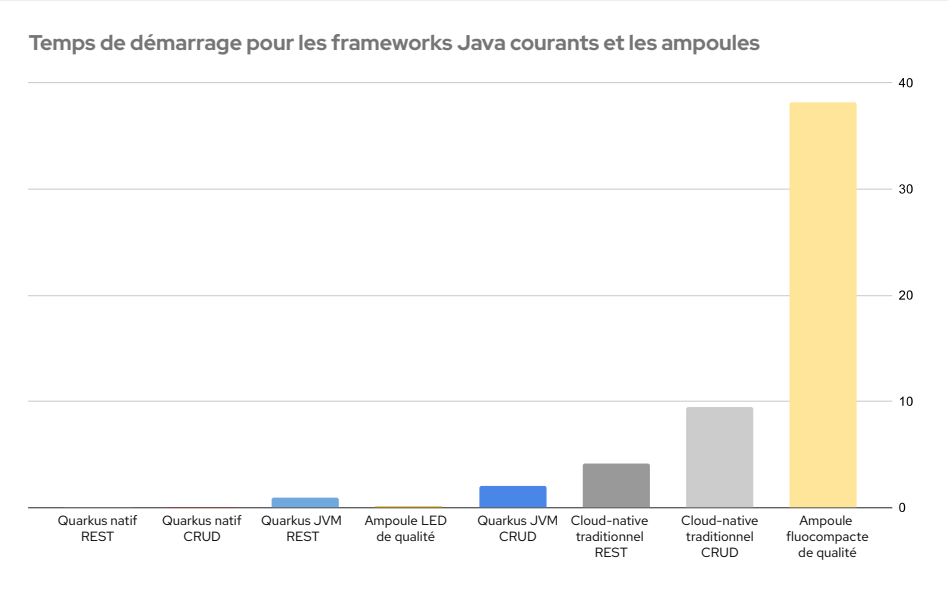


Figure 4 : temps de préchaufffage des frameworks et des ampoules. Pour les frameworks, cela équivaut au temps de démarrage plus le temps d'obtention de la première réponse. Pour les ampoules, il correspond au temps nécessaire pour atteindre la pleine luminosité.

Lorsqu'ils sont comparés à des mesures plus longues, les temps de démarrage des applications natives pour Quarkus et des LED sont trop infimes pour être perçus, comme le montre la figure.

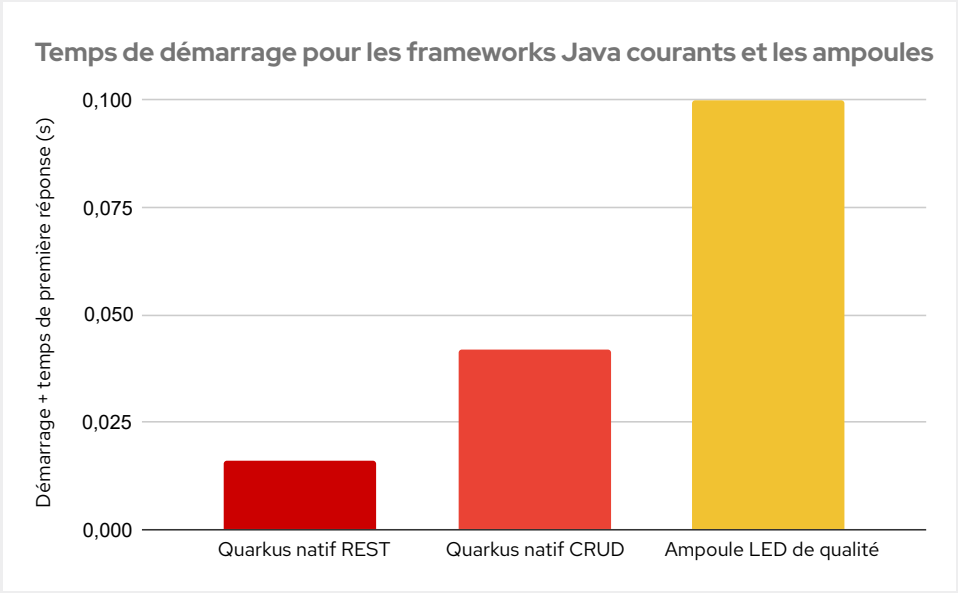


Figure 5 : temps de démarrage et de première réponse[6][7]

Efficacité

L'efficacité est une excellente solution « sans regret ». Elle permet de faire des économies, de réduire les émissions de carbone, et offre souvent des avantages supplémentaires, comme une meilleure expérience de développement, une expérience utilisateur plus harmonieuse, une accessibilité accrue et des taux de conversion utilisateur plus élevés.

Comment le framework Quarkus réduit-il les émissions de carbone ?

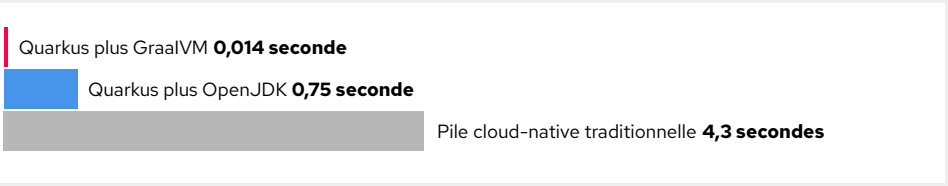
C'est là qu'intervient Quarkus. Quarkus est une pile Java native pour Kubernetes qui repose sur des normes et des bibliothèques Java reconnues, et qui est conçue pour les conteneurs et les déploiements dans le cloud. Une application Quarkus peut être exécutée sur une machine virtuelle Java (JVM) ou compilée dans un fichier binaire natif à l'aide de GraalVM.

Nous l'avons déjà évoqué, Quarkus en mode natif démarre plus vite qu'une ampoule, même sur une JVM. Une application REST Quarkus simple affiche un temps de démarrage quatre fois inférieur comparé aux piles cloud-native traditionnelles.

6 *« Power Up Delay When LED Lights are Turned On – Is This Normal? », LampHQ, consulté en octobre 2022*

7 *« Supersonic/Subatomic Java », Quarkus, consulté en octobre 2022*

Quarkus se revendique comme étant un framework « Java supersonique subatomique », une façon de mettre l'accent sur sa vitesse et sa légèreté.



Pour une application REST CRUD, le temps de démarrage de Quarkus est tout aussi réduit :

Figure 6 : comparaison des temps de démarrage des applications Quarkus[8]

En mode JVM, l'empreinte mémoire de l'application Quarkus représente un peu plus de la moitié de la mémoire d'un framework traditionnel, et environ un dixième en mode natif.

Application REST

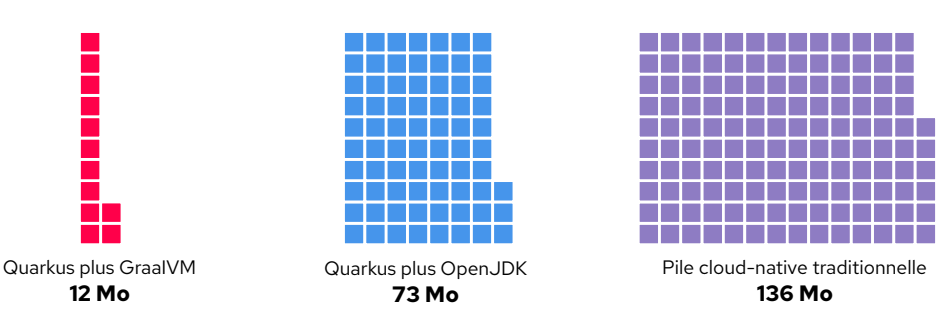


Figure 7 : mémoire utilisée par une application REST, avec Quarkus ou un environnement d'exécution cloud-native traditionnel[9]

Ces valeurs relatives à l'empreinte mémoire et au temps de démarrage entraînent-elles une réduction de la consommation d'énergie au cours de la durée de vie d'une application typique ? Pour faire court, oui.

Test de capacité

Afin de mesurer la consommation d'énergie, l'équipe d'ingénierie contrôlant les performances de Quarkus a réalisé des expériences avec l'outil RAPL (Running Average Power Limit) sur une application de base CRUD (create, read, update and delete) qui interroge une base de données

8 « Quarkus Runtime Performance », *blog de Quarkus, 7 juillet 2019*

9 « Quarkus Runtime Performance », *blog de Quarkus , 7 juillet 2019*

via une interface REST. Le test consistait à démarrer et lancer l'application, puis un générateur de charge (wrk2) envoyait des requêtes à un taux d'arrivée fixe (taux d'injection). Pendant que l'application était sous charge, la consommation d'énergie du processeur et de la DRAM était mesurée avec RAPL. L'équipe a utilisé une machine à deux processeurs de 16 cœurs chacun. L'application était le seul et unique processus exécuté sur quatre cœurs spécifiques d'un des processeurs. Le segment de mémoire n'était pas épinglé et pouvait aller jusqu'à 12 Go.

L'équipe a constaté que l'application exécutée sur le framework cloud-native traditionnel ne pouvait pas prendre en charge autant de requêtes entrantes que Quarkus (d'où les lignes plus courtes sur le graphique).

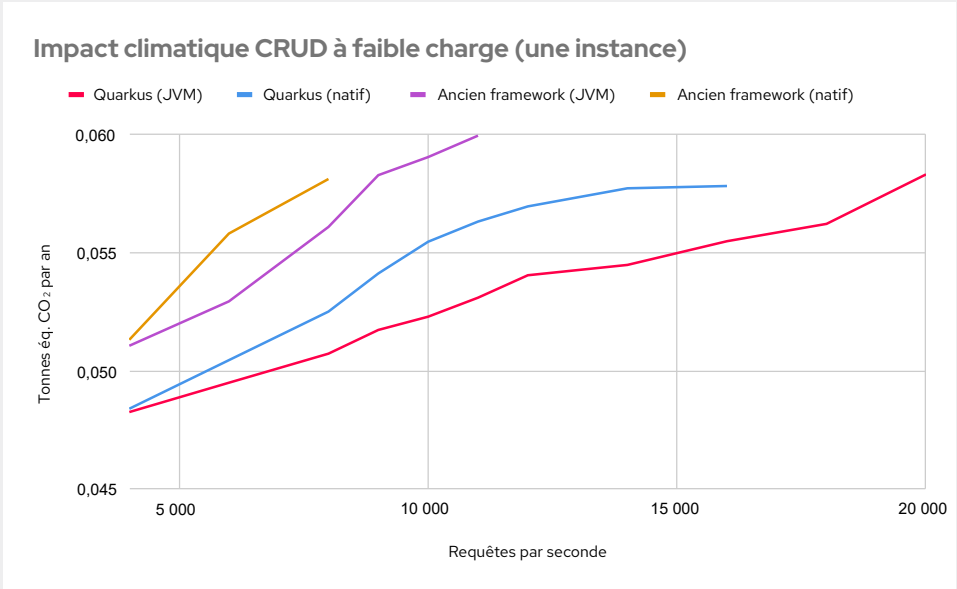


Figure 8 : empreinte carbone d'une application CRUD à une instance en fonction de la charge

Postulats :

- Les valeurs d'équivalent CO2 sont basées sur le bouquet énergétique des États-Unis.

Notez que l'axe des ordonnées de ce graphique ne commence pas à zéro. Le système présentait une consommation d'énergie de base à toutes les charges.

Pour un nombre donné de requêtes, Quarkus présentait les valeurs de consommation d'énergie et d'équivalent CO2 les plus faibles. Quarkus en mode natif arrive en deuxième position, suivi du framework cloud-native traditionnel, et enfin de la version native de ce framework (voir la figure 9).

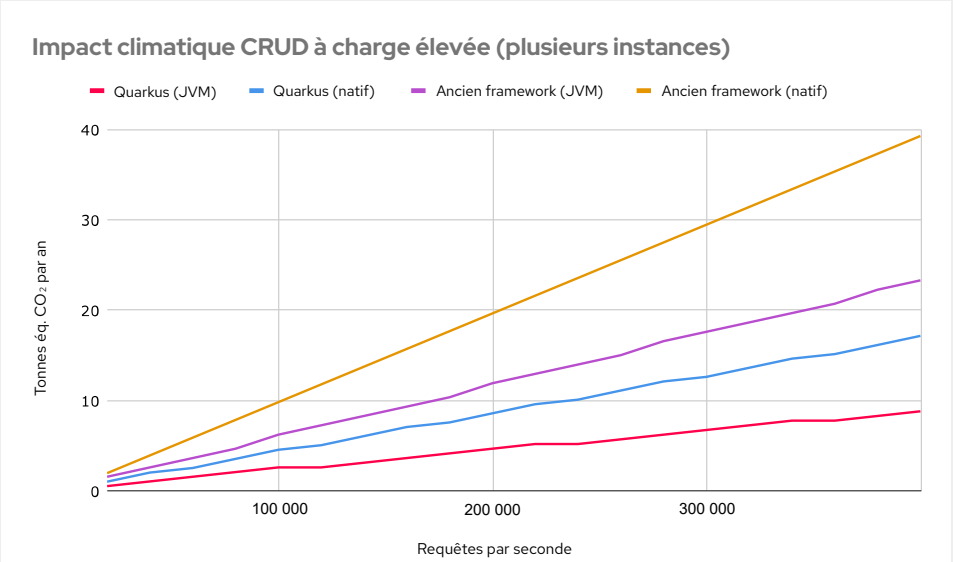


Figure 9 : empreinte carbone d'une application CRUD à plusieurs instances en fonction de la charge

Synthèse

Lors des expériences, les applications ont affiché des valeurs de consommation d'énergie (et donc d'émissions de carbone) nettement inférieures sur Quarkus par rapport à un framework cloud-native traditionnel.

Pour Quarkus et le framework traditionnel, il a fallu plus d'énergie pour gérer la charge en mode natif comparé à la JVM. Ce résultat peut surprendre. Vous verrez plus loin pourquoi (et quand) le mode natif consomme davantage d'énergie.

Ces tendances se maintiennent à la fois à faible charge (lorsqu'une seule instance d'application peut être utilisée) et à charge élevée (quand plusieurs instances sont requises).

Test de densité

Quelles sont les performances des applications dans les environnements cloud aux ressources plus limitées ? La densité des déploiements Quarkus peut être beaucoup plus élevée que celle des piles cloud-native traditionnelles, en raison de leur faible encombrement. Pour quel impact en termes d'émissions de carbone ?

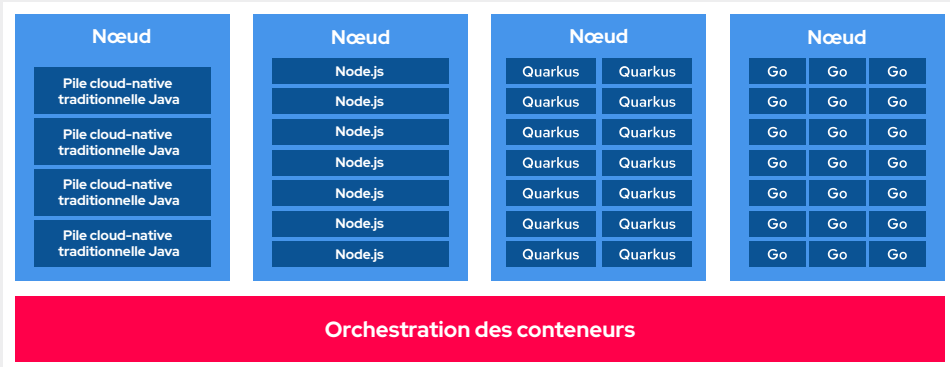


Figure 10 : schéma de l'orchestration des conteneurs

Pour le savoir, l'équipe Quarkus a déployé une application dans le cloud et y a chargé 800 requêtes par minute durant 20 jours. L'expérience a été tentée avec plusieurs tailles d'instance et frameworks d'applications. Sur un framework cloud-native courant, une instance AWS t2.medium avec deux processeurs et 4 Go de mémoire était nécessaire pour gérer la charge. Sur Quarkus, une instance beaucoup plus petite, t2.micro, avec un processeur et 2 Go de mémoire suffisait.

Tableau 1 : comparaison entre le framework traditionnel sur JVM et Quarkus

Framework	Type d'instance	Prix/heure	Éq. CO_2/heure
Ancien framework sur JVM	t2.medium (2 vCPU, 4 Go)	33,40 $	5 112 g
Quarkus sur JVM	t2.micro (1 vCPU, 1 Go)	8,40 $	2 376 g
Quarkus natif	t2.micro (1 vCPU, 1 Go)	8,40 $	2 376 g

En moyenne, Quarkus a généré deux à trois fois moins de carbone que l'ancien framework (voir figure 11).

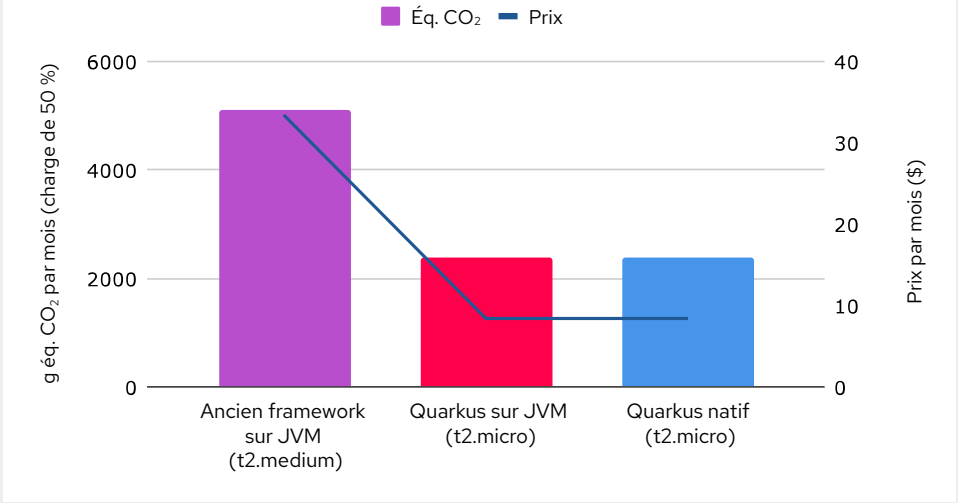


Figure 11 : les effets du choix du framework[10]

Postulats :

- La datacenter se situe dans la région us-east-1.
- La charge s'élève à 50 %.
- Les mesures ayant été effectuées entre mai et août 2021, il n'a pas été possible d'essayer une version native de l'ancien framework.

Les résultats du test de densité sont cohérents avec le test de capacité : pour exécuter la même charge de travail, Quarkus utilise environ deux fois moins de carbone que le framework traditionnel.

Source des données :

- Mesures de performance en laboratoire

L'exécution sur une instance plus petite réduit à la fois les coûts et les émissions de carbone. Les émissions liées à l'utilisation d'une application proviennent de l'exécution de la charge de travail et du matériel lui-même. L'utilisation d'un datacenter bas carbone, situé par exemple à Stockholm, en Suède, réduira les émissions issues du processus d'exécution, mais pas celles des pièces constitutives de l'équipement. Dans une démarche de respect de l'environnement, il est important de tenir compte des exigences matérielles d'une application.

Si vous savez quelle instance vous exécutez et dans quelle région, vous pouvez estimer les émissions par heure en utilisant des ensembles de données accessibles au public, tels que l'ensemble de données carbone Teads qui a été intégré à l'outil CCF de calcul de l'empreinte carbone du cloud. L'ensemble Teads contient des données pour quatre niveaux de charge prédéfinis, nous avons donc supposé ici une charge uniforme de 50 %.

Une des différences entre les deux ensembles de résultats est que, dans l'environnement contraint, avec un système à faibles spécifications et une charge limitée, le mode natif de Quarkus n'a pas généré plus d'émissions de carbone que le mode JVM. En réalité, l'application native utilisait probablement un peu moins de ressources que l'application JVM, ce qui est impossible à voir dans le tableau des résultats à cause de la granularité grossière des conversions de la charge en carbone.

10 *« Carbon footprint estimator for AWS instances », Teads Engineering, consulté en octobre 2022*

Choix de l'architecture et carbone

Quarkus en mode natif ou Quarkus JVM ?

Pourquoi les résultats des tests de capacité et de densité donnent-ils des réponses différentes à la question « L'exécution en mode natif est-elle plus économe en énergie que l'exécution en mode JVM ? ». Lors du test de densité, la charge de travail est fixe (800 transactions par seconde) et le but de l'expérience est de réduire au maximum la taille des machines. Le mode natif permet d'aller plus loin que le mode JVM, mais en raison de la granularité plutôt grossière des spécifications de la machine, cela ne se reflète pas dans les chiffres. Vous ne pouvez pas provisionner une machine avec 427 Mo de RAM.

Lors du test d'évolutivité, la machine avait des caractéristiques généreuses (même si l'application était limitée à quatre cœurs). Le système a géré autant de charges que possible, avant que la consommation d'énergie ne soit mesurée. Si une seule instance ne pouvait pas gérer toute la charge, d'autres instances étaient ajoutées. Dans ce scénario, l'efficacité carbone et le débit du framework sont étroitement corrélés. Le débit de l'application est légèrement plus élevé sur la JVM comparé au mode natif.

Cela peut sembler contre-intuitif car, en général, la rapidité entraîne une hausse de la consommation d'énergie, pas une baisse. Pour les logiciels, l'approche doit être différente. Un logiciel plus rapide en fait plus (dans un temps défini) avec les mêmes ressources système. Il est donc plus efficace tout en consommant moins d'énergie.

Cas d'utilisation pour le mode natif (en prenant en compte l'efficacité carbone)

- Faible charge de travail (aucun goulet d'étranglement dû au débit)
- Matériel ancien ou à ressources limitées (en particulier la mémoire)
- Taux de redéploiement élevé (pas de lancement d'applications avant leur arrêt)
- Charge de travail incohérente (nécessité de déployer et annuler le déploiement des instances de manière flexible)
- Modèle serverless

Cas d'utilisation pour JVM (en prenant en compte l'efficacité carbone)

- Charge de travail élevée (débit élevé requis)
- Processus à longue durée de vie (peu d'économies sur toute la durée de vie induites par le démarrage rapide du mode natif)

Toutefois, gardez en mémoire que les modes natif et JVM ne sont pas une question de choix. Vous bénéficiez d'une grande flexibilité puisqu'il est possible d'exécuter les applications Quarkus dans l'un ou l'autre mode, simplement en ajoutant un indicateur de compilation. Comme les bouquets énergétiques et la charge varient d'heure en heure, la solution idéale sera souvent un modèle hybride.

Vous pouvez changer de modèle à plusieurs reprises, sans coûts supplémentaires ou presque. C'est comme si vous vouliez diminuer les émissions de carbone en codant en C++. Cette décision doit être prise dès le départ, car il sera extrêmement difficile de revenir dessus.

Il est peu probable qu'un système de production sache exactement à l'avance quel type de charge il recevra, à moins qu'il ne s'agisse d'un système spécialisé non exposé au trafic externe. Bien souvent, cette incertitude pèsera sur la personne en charge de la stabilité d'un tel système, qui prévoira alors une capacité supplémentaire importante pour parer à toute éventualité. Pour qu'un système affiche des temps de réponse instantanés côté utilisateur, il doit traiter les requêtes en 0,1 seconde environ. Cela signifie que des instances supplémentaires doivent traiter les requêtes durant ce laps de temps, ou s'en rapprocher.

Si les serveurs de sauvegarde sont basés sur une application JVM, ils devront être démarrés longtemps à l'avance. Dans l'idéal, ils devraient également être démarrés bien en amont pour lancer l'application, afin de permettre aux optimisations JIT de commencer à s'exécuter.

Si une application est native, aucun lancement n'est nécessaire, ce qui facilite les choses. Les instances, elles, peuvent encore avoir besoin d'être démarrées à l'avance, mais moins qu'avant. Être capable de compenser les pics de charge en quelques secondes (une fois le temps de démarrage de l'infrastructure inclus) supprime une grande partie du stress lié à la planification de la capacité. Plus les instances présentent une empreinte réduite, meilleure en sera la granularité.

Un système idéal pourrait combiner des nœuds basés sur JVM (des ordinateurs surdimensionnés efficaces pour la charge de travail principale) avec des nœuds d'image native pour améliorer la flexibilité et peut-être permettre une prise en charge occasionnelle lors des mises à niveau continues. La capacité de Quarkus à compiler la même application dans les deux modes représente un atout majeur pour la création de logiciels plus durables et la planification de la capacité adaptée aux enjeux actuels.

Programmation réactive ou impérative ?

La programmation réactive génère-t-elle moins d'émissions que les modèles traditionnels ? Pas facile à dire.

Les principes de la programmation réactive découlent de la nécessité de construire des systèmes plus souples, résilients et efficaces. Ces systèmes s'appuient souvent sur un modèle d'exécution qui utilise moins de threads et avec une sympathie mécanique élevée. Donc, en principe, ils exploitent plus efficacement les ressources.

Cependant, une bibliothèque de programmation réactive moderne est souvent utilisée pour créer des applications réactives, auxquelles sont généralement allouées plus de ressources. En effet, l'application utilise moins de threads, mais ces threads doivent collaborer entre eux. Ainsi, vous avez besoin d'un protocole de contre-pression pour éviter de surcharger certaines parties de l'application. Actuellement, les bibliothèques de programmation réactives n'affichent pas une efficacité optimale en matière de processeur, de mémoire et de d'émissions de carbone.

Une contre-pression correctement mise en œuvre régulera la charge de votre système et permettra d'éviter les goulets d'étranglement et autres situations coûteuses. Bien qu'étant globalement modérées, ces charges volatiles ou « en rafales » peuvent se révéler être la principale cause de mauvaises performances et d'une faible efficacité de l'application sous-jacente. Par exemple, les charges en rafale déclenchent un parking et un déparking excessifs du noyau, ce qui s'avère inefficace. Une approche heuristique pour éviter cette situation consiste à allouer moins de ressources à l'application et à maintenir le système suffisamment occupé pour éviter les périodes d'inactivité durant une rafale. L'objectif est d'éviter le parking et le déparking du noyau lors d'une petite accalmie au milieu d'une rafale.

Quelles conséquences sur le choix de votre modèle de programmation ? Comme beaucoup de décisions relatives aux architectures, cela dépend. L'efficacité de la programmation réactive variera en fonction de l'application et du contexte d'exécution.

Mesurer l'empreinte carbone

La mesure et le suivi réguliers de l'empreinte et de l'impact carbone constituent l'un des principes de l'ingénierie logicielle durable. Il est important de mesurer les émissions, afin de savoir comment les réduire. Cependant, en obtenir une mesure précise n'est pas si simple.

Le changement climatique est causé par l'accumulation de GES dans l'atmosphère. Les GES comprennent le dioxyde de carbone (CO_2), le méthane, le protoxyde d'azote et les réfrigérants hydrofluorocarbonés. Bien que le CO_2 soit le GES le plus courant, d'autres gaz ont un effet encore

plus puissant sur le réchauffement climatique. Pour faciliter la comparaison, toutes les mesures de GES sont normalisées en équivalent dioxyde de carbone (éq. CO_2). Le terme « carbone » est souvent utilisé pour désigner l'ensemble des GES.

Voici les étapes à prendre en compte dans le calcul de l'empreinte carbone globale d'un logiciel :

- Mesure de la consommation d'énergie de la charge de travail : quelle est la taille de l'ordinateur et quel est son niveau d'activité ?
- Calcul de l'intensité carbone de l'électricité : l'électricité provient-elle de la combustion de charbon ou de gaz naturel, ou d'énergies propres (hydraulique, solaire ou éolien) ?
- Ajout de l'empreinte carbone du trafic réseau
- Ajout de la quantité de carbone contenue dans le matériel

Cette formule simple occulte plusieurs sources d'émissions indirectes tout au long du cycle de vie des logiciels, notamment le transport du matériel et la construction du datacenter, entre autres activités.

L'empreinte carbone est généralement divisée en trois scopes :

- Le **scope 1** inclut les émissions directes. Il mesure les effets directs de la consommation de combustibles ou d'autres activités émettrices de GES par une entité (par exemple, la consommation de carburant d'un propriétaire de voiture ou d'un générateur d'électricité).
- Le **scope 2** correspond aux émissions indirectes liées à la consommation d'énergie (électricité et chauffage). Il mesure l'électricité consommée.
- Le **scope 3** couvre les émissions tout au long de la chaîne de valeur, notamment les émissions liées à la fourniture du matériel, des bâtiments et des autres infrastructures, ainsi que les émissions relatives au transport dans la chaîne d'approvisionnement. Il inclut également les effets en amont de l'utilisation des services d'une entreprise.

Comme vous pouvez le constater, ces calculs deviennent vite complexes. Il est souvent impossible d'effectuer des mesures directes, donc ils reposent presque toujours sur une combinaison de mesures et d'estimations (ou modélisation). Bien que précise, la stratégie de mesure peut être intrusive, et même souvent impossible.

Kepler

Le projet Kepler, ou Kubernetes-based Efficient Power Level Exporter, est né d'une collaboration entre Red Hat et IBM Research. Il utilise un probe léger et efficace écrit en eBPF pour collecter des informations sur les systèmes et processus, y compris le temps d'exécution du processeur et les compteurs de performances, et établit une corrélation avec les lectures RAPL afin d'estimer l'électricité consommée par les pods. Les estimations sont traitées via Prometheus et peuvent être visualisées sur Grafana et d'autres consoles cloud-native.

Mesure pour l'optimisation

Lors de l'évaluation des émissions de carbone, il est important d'effectuer des calculs complets, bien que cette tâche ne soit pas facile. Durant la phase d'optimisation, en revanche, il est souvent inutile d'effectuer un calcul parfaitement précis. La priorité consiste à déterminer les émissions relatives de deux scénarios différents, ce qui peut être plus simple à calculer. Si vous pouvez évaluer la forme de l'inclinaison, vous pouvez la suivre pour réaliser des optimisations, même sans valeurs absolues.

La charge et les autres indicateurs de performances comme variables pour les émissions de carbone

Bien qu'elles restent moins précises qu'une mesure exacte, nos mesures de performance et de carbone indiquent une corrélation vague, mais utile, entre les indicateurs de performances et la consommation d'énergie. Par exemple, si une modification apportée à une application lui

permet de gérer le même volume de requêtes avec la même empreinte mémoire et une charge de processeur réduite, il est presque certain qu'elle utilise moins d'énergie (en supposant que le traitement n'ait pas été délégué ailleurs). De même, la réduction de l'empreinte mémoire ou du trafic réseau, en supposant que les autres paramètres ne changent pas, a des chances de limiter les émissions de carbone.

Le coût comme variable pour les émissions de carbone

Le coût est une autre variable potentiellement utile pour les émissions de carbone. Le coût du cloud computing étant généralement assez visible, il peut s'agir d'une heuristique utile, mais il existe des limites :

- L'électricité au charbon peut, à court terme, se révéler bon marché. Lorsque nous utilisons le coût comme variable, nous devons comparer ce qui est comparable.
- Même si les tendances générales vont dans le même sens, rien n'est linéaire. Par exemple, dans les tests de densité abordés précédemment, la différence de prix entre Quarkus et le framework cloud-native traditionnel était de 26,08 USD contre 9,15 USD, et la différence au niveau des émissions de carbone était de 7,1 g contre 3,3 g. La différence de prix était supérieure à la différence des émissions (facteur de trois contre deux). Ce ratio est propre à la région us-east-1. Dans d'autres régions qui disposent d'une énergie plus propre, de fortes variations de coûts entraîneraient des différences encore plus faibles en matière d'émissions de carbone.

Le modèle peut être rendu plus concret, en effectuant une sorte d'étalonnage. Par exemple, après répartition des dépenses en plusieurs catégories, une entreprise peut établir un rapport entre les coûts du cloud et les émissions de carbone (chaque million dépensé dans le cloud génère x tonnes d'émissions de carbone). C'est ce qu'on appelle l'évaluation économique du cycle de vie des entrées-sorties. Des facteurs d'émissions sont ainsi normalisés pour diverses catégories de dépenses liées au cloud. Le modèle est forcément approximatif, mais peut être pratique, pour la simple raison que les données financières sont très largement accessibles et que les calculs sont rapides.

Avec ces précautions, les réductions de coûts peuvent raisonnablement servir d'indicateurs pour limiter les émissions de carbone, en supposant que les autres facteurs ne changent pas.

Synthèse

Quarkus permet de réduire les émissions de carbone de plusieurs façons. Quarkus en soi est un environnement d'exécution hautement efficace, et les applications qui y sont exécutées utilisent moins de ressources et, par conséquent, émettent moins de carbone. Cette réduction des émissions de carbone est automatique, car aucune démarche n'est requise au-delà de la migration vers Quarkus (le cas échéant). Vous n'avez rien d'autre à faire que d'utiliser Quarkus.

Vous pouvez aussi aller plus loin et réduire encore plus vos émissions. Quarkus prend en charge des architectures avec un plus haut niveau d'efficacité carbone. Dans de nombreuses situations, l'option réactive fournie par Quarkus peut permettre des économies de carbone. Au niveau du matériel, vous pouvez peut-être provisionner d'autres équipements et exécuter vos charges de travail sur des machines plus petites, afin d'économiser de l'énergie lors de l'exécution et de réduire la quantité de carbone contenue dans le matériel.

Quarkus apporte une grande souplesse et offre la possibilité de mettre complètement à l'arrêt les serveurs quand ils sont inutilisés. En mode natif, les instances d'application s'exécutent presque instantanément et peuvent donc gérer un trafic en rafales sans dégradation de la qualité.

Dans de nombreux scénarios, un modèle hybride associant des instances natives et JVM offre la combinaison optimale avec un débit élevé, une haute efficacité carbone en charge et une faible consommation d'énergie au repos.

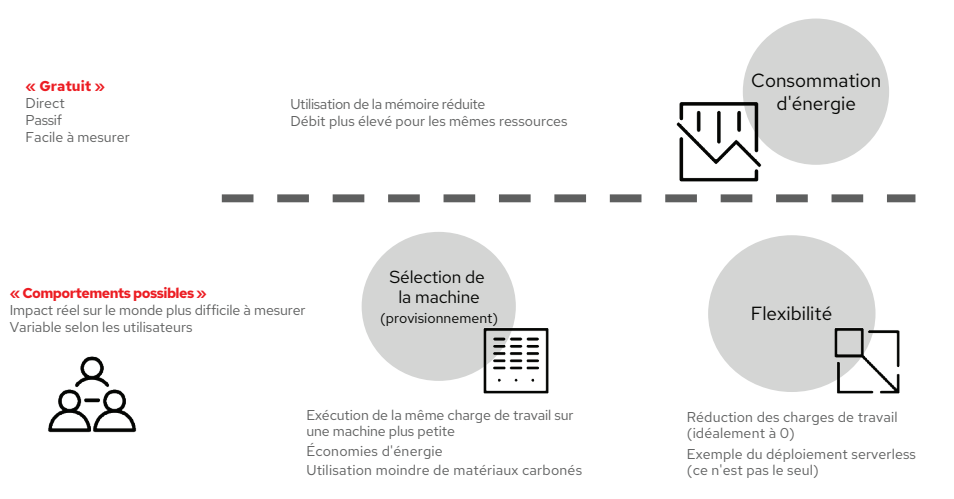


Figure 12 : comment Quarkus aide à réduire l'empreinte carbone

Témoignages clients

Voici des exemples de clients Red Hat qui ont réduit leur utilisation des ressources en adoptant Quarkus. Ces témoignages racontent comment ces économies se traduisent, depuis les tests de performance jusqu'en production.

Lufthansa

« Avec Quarkus, nous pouvons désormais gérer des déploiements trois fois plus denses, sans que cela n'ait d'incidence sur la disponibilité et le temps de réponse des services. »

Thorsten Pohl
Responsable des produits Automatisation et architecte de plateforme Division des produits numériques, AVIATAR

Lufthansa Technik utilise une plateforme numérique appelée AVIATAR. Pour faire évoluer le développement logiciel, l'entreprise a migré vers une architecture de microservices. La productivité de l'équipe de développement s'est améliorée, au détriment toutefois de l'utilisation des ressources. En effet, chacun des 100 services de l'application AVIATAR fonctionnait en triple pour assurer une haute disponibilité. Certains consommaient beaucoup de capacité mémoire et processeur. Tout cela ramené à 300 instances de service, la consommation globale de ressources de l'architecture de microservices était bien trop élevée. Lufthansa procède actuellement à la migration de ses services vers Quarkus. L'entreprise combine le mode natif et JVM, selon le service. Certains services basculent même entre les deux.

Avec Quarkus, Lufthansa a aussi pu introduire l'informatique serverless dans l'architecture, et économiser davantage de ressources. Les applications natives démarrant assez rapidement, elles peuvent être laissées à l'arrêt, sans dégrader les services.

Depuis la migration vers Quarkus, Lufthansa a remarqué que les nouveaux services n'utilisent plus d'un tiers des ressources. Par exemple, la version Quarkus de l'ancien service 0,5 cœur 1 Go ne nécessite que 200 millicœurs et 200 à 400 Mo de RAM par instance.

Vodafone

Vodafone migre une partie de son architecture depuis Spring Boot vers Quarkus, afin d'utiliser moins de ressources. Avant, avec Spring Boot, certains microservices avaient besoin de 1 Go de RAM. L'entreprise peut désormais déployer un microservice Quarkus en production avec 512 Mo de RAM. « Dans une architecture de 80 microservices, cela représente de grosses économies ! », souligne Christos Sotiriou, responsable technique DXL chez Vodafone Grèce. Il ajoute : « Par

défaut (sans optimisation), Quarkus est 50 à 60 % plus léger (en mode JVM) que Spring après les optimisations (gestion des dépendances, essai des options JVM, etc.) » Bien que l'aspect financier était la principale motivation de Vodafone, la réduction de l'utilisation des ressources aura également des avantages environnementaux.

Decathlon

Lors de la conception de sa nouvelle plateforme de messagerie VCStream, Decathlon a choisi Quarkus en partie pour les bénéfices écologiques de la faible consommation de ressources. Le système VCStream a atteint une efficacité impressionnante, avec un début mesuré de 1 million de messages par minute, par processeur et par Go de mémoire. Il s'agissait d'une nouvelle application, déployée une seule fois, donc il n'y a pas de comparaison possible avec une précédente implémentation.

En savoir plus

Pour plus d'informations sur la version Red Hat® de Quarkus, référez-vous à la présentation de Quarkus.

À propos de Red Hat

Premier éditeur mondial de solutions Open Source, Red Hat s'appuie sur une approche communautaire pour fournir des technologies Linux, de cloud hybride, de conteneurs et Kubernetes fiables et performantes. Red Hat aide ses clients à développer des applications cloud-native, à intégrer des applications nouvelles et existantes ainsi qu'à gérer et à automatiser des environnements complexes. Conseiller de confiance auprès des entreprises du Fortune 500, Red Hat propose des services d'assistance, de formation et de consulting reconnus qui apportent à tout secteur les avantages de l'innovation ouverte. Situé au cœur d'un réseau mondial d'entreprises, de partenaires et de communautés, Red Hat participe à la croissance et à la transformation des entreprises et les aide à se préparer à un avenir toujours plus numérique.

facebook.com/redhatinc
@RedHatFrance
linkedin.com/company/red-hat

Europe, Moyen-Orient et Afrique (EMEA)
00800 7334 2835
europe@redhat.com

France
00 33 1 41 91 23 23
fr.redhat.com

fr.redhat.com
#F32147_1122