RED HAT®
CONSULTING

redhat.

LIVRE BLANC

LES PRINCIPES DE LA CONCEPTION D'APPLICATIONS CONTENEURISÉES

Rédigé par Bilgin Ibryam

SYNTHÈSE

Les applications dites « conçues pour le cloud » sont des applications conçues spécialement pour être exécutées sur une infrastructure cloud. En général, ces applications sont développées sous la forme de microservices faiblement couplés qui s'exécutent dans des conteneurs gérés par une plateforme. Elles savent anticiper les défaillances et peuvent s'exécuter et évoluer même lorsque leur infrastructure sous-jacente est en panne. Toutefois, pour offrir de telles capacités, les plateformes conçues pour le cloud imposent aux applications qui s'y exécutent une série de règles et de contraintes. Ces règles permettent de s'assurer que les applications respectent certaines contraintes et permettent aux plateformes d'automatiser la gestion des applications conteneurisées.

De nombreuses entreprises ont compris l'importance du développement d'applications conçues pour le cloud. Malheureusement, elles ne savent pas toujours par où commencer. Lorsque les plateformes conçues pour le cloud et les applications conteneurisées fonctionnent en parfaite harmonie, il est possible d'anticiper les défaillances et d'exécuter ou de faire évoluer les applications même lorsque l'infrastructure sous-jacente est en panne. Ce livre blanc décrit un certain nombre de règles que vos applications conteneurisées doivent respecter pour s'intégrer parfaitement dans le cloud. Si vous suivez ces principes, vous serez sûr de pouvoir automatiser vos applications sur les plateformes conçues pour le cloud, telles que Kubernetes.

LES PRINCIPES DE LA CONCEPTION LOGICIELLE :

- KISS : « Keep it simple, stupid », ne compliquez pas les choses
- DRY : « Don't repeat yourself », ne vous répétez pas
- YAGNI : « You aren't gonna need it », vous n'en aurez pas besoin
- SoC : « Separation of concerns », séparation des préoccupations

L'APPROCHE DE RED HAT EN MATIÈRE DE CONTENEURS CONÇUS POUR LE CLOUD :

- Principe de la préoccupation unique
- Principe d'observabilité élevée
- Principe de la conformité du cycle de vie
- Principe de l'immuabilité de l'image
- Principe de la disponibilité des processus
- Principe de l'autosuffisance
- Principe du confinement de l'exécution

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

fr.redhat.com

QUELQUES BONNES PRATIQUES COURANTES EN MATIÈRE DE CONTENEURS :

- Créer des images de petite taille
- Prendre en charge des ID utilisateurs arbitraires
- Marquer les ports importants
- Utiliser des volumes pour les données persistantes
- Définir des métadonnées pour les images
- Synchroniser l'hôte et l'image

LES PRINCIPES DE LA CONCEPTION LOGICIELLE

De nombreux principes qui régissent notre vie quotidienne sont généralement considérés comme des vérités ou des croyances dont découlent d'autres principes. En ce qui concerne les logiciels, les principes sont surtout des directives abstraites qu'un développeur est censé suivre lorsqu'il conçoit son logiciel. Ils s'appliquent à tous les langages de programmation, à l'aide de différents modèles et de diverses pratiques.

En général, le développeur se base sur des modèles et des pratiques pour respecter ces principes et obtenir les résultats escomptés. Il existe un certain nombre de principes fondamentaux pour la conception de logiciels de qualité, dont découlent tous les autres. Voici quelques exemples de principes fondamentaux :

- **KISS** : « Keep it simple, stupid », que l'on peut traduire par « Ne compliquez pas les choses »
- **DRY** : « Don't repeat yourself », que l'on peut traduire par « Ne vous répétez pas »
- **YAGNI** : « You aren't gonna need it », que l'on peut traduire par « Vous n'en aurez pas besoin »
- **SoC** : « Separation of concerns », que l'on peut traduire par « Séparation des préoccupations »

Même si ces principes n'énoncent pas de règles concrètes, ils représentent un langage et un sens commun auxquels de nombreux développeurs adhèrent et se réfèrent régulièrement.

Il existe également d'autres principes, introduits par Robert C. Martin, qui concernent spécifiquement la programmation orientée objets. Il s'agit des principes SOLID pour **S**ingle responsibility (responsabilité unique), **O**pen/closed (ouvert/fermé), **L**iskov substitution (substitution de Liskov), **I**nterface segregation (isolement des interfaces), **D**ependency inversion (inversion des dépendances). Ces règles sont des principes complémentaires qui peuvent être interprétés assez librement, mais qui donnent néanmoins une ligne directrice suffisamment claire pour aider les développeurs à créer de meilleurs logiciels orientés objets. En appliquant les principes SOLID, il est plus probable de pouvoir créer un système dont les attributs sont de meilleure qualité et qui est plus facile à entretenir sur le long terme.

Les principes SOLID se basent sur des préceptes et des concepts orientés objets, tels que les classes, les interfaces et l'héritage, pour faciliter la réflexion sur la programmation orientée objets. De la même manière, il existe aussi des principes qui régissent la création des applications conçues pour le cloud et qui se basent non pas sur une classe, mais sur une image de conteneur. En suivant ces principes, il est plus facile de créer des applications conteneurisées adaptées aux plateformes conçues pour le cloud, telles que Kubernetes.

L'APPROCHE DE RED HAT EN MATIÈRE DE CONTENEURS CONÇUS POUR LE CLOUD

Aujourd'hui, il est possible de conteneuriser et d'exécuter n'importe quelle application ou presque. Par contre, il est plus complexe de créer une application conteneurisée que vous pouvez automatiser et orchestrer efficacement sur une plateforme conçue pour le cloud, telle que Kubernetes.

Les idées sous-jacentes sont inspirées de nombreux autres travaux, tels que le document « The Twelve-Factor App » qui s'intéresse aussi bien à la gestion du code source qu'aux modèles d'évolutivité des applications. Les principes qui suivent concernent plus particulièrement la conception d'applications conteneurisées basées sur des microservices pour les plateformes conçues pour le cloud, telles que Kubernetes.

Les principes pour la création d'applications conteneurisées présentés ci-dessous utilisent une image de conteneur comme base élémentaire et une plateforme d'orchestration de conteneurs comme environnement d'exécution cible. Si vous suivez ces principes, vos conteneurs se comporteront comme s'ils avaient été conçus spécialement pour le cloud dans la plupart des moteurs d'orchestration de conteneurs. Ils pourront ainsi être planifiés, mis à l'échelle et surveillés de façon automatisée. Ces principes sont présentés ici sans ordre particulier.

PRINCIPE DE LA PRÉOCCUPATION UNIQUE

Ce principe est très similaire au principe de responsabilité unique du modèle SOLID, qui indique qu'une classe ne doit avoir qu'une seule responsabilité. Il énonce que chaque responsabilité est un axe de changement et qu'une classe ne doit changer que pour une seule et unique raison. Dans le principe de la préoccupation unique, le terme « préoccupation » représente un niveau d'abstraction plus élevé que la « responsabilité » et il permet de mieux décrire la portée au niveau du conteneur et non de la classe. Si l'idée principale à l'origine du principe de responsabilité unique est la raison unique de changement, celle qui est à la base du principe de la préoccupation unique est la réutilisation et le remplacement des images de conteneurs. Si vous créez un conteneur avec une seule préoccupation et qu'il la traite avec un ensemble complet de fonctions, alors vous aurez plus de chances de pouvoir réutiliser l'image de ce conteneur dans différents contextes d'applications.

Ainsi, le principe de la préoccupation unique peut se résumer ainsi : chaque conteneur ne doit avoir qu'une seule préoccupation et la traiter correctement. Il est plus facile de respecter ce principe que son équivalent pour la programmation orientée objets, car les conteneurs ne gèrent habituellement qu'un seul processus et, dans la plupart des cas, cet unique processus ne concerne qu'une seule préoccupation.

Figure 1 : plusieurs conteneurs à préoccupation unique dans une seule unité de déploiement

Si votre microservice conteneurisé doit traiter plusieurs préoccupations, il peut utiliser des modèles « sidecar » et « init-container » par exemple pour réunir plusieurs conteneurs au sein d'une seule unité de déploiement (un « pod »), où chaque conteneur continue à ne traiter qu'une seule préoccupation. Vous pouvez aussi remplacer des conteneurs par d'autres qui traitent la même préoccupation. Par exemple, il est possible de remplacer le conteneur du serveur Web ou un conteneur de mise en œuvre de file d'attente par un conteneur plus récent et plus évolutif.

PRINCIPE D'OBSERVABILITÉ ÉLEVÉE

Les conteneurs permettent de mettre en paquet et d'exécuter des applications en les traitant comme des « boîtes noires ». Un conteneur destiné à intégrer le cloud doit quant à lui pouvoir fournir des interfaces de programmation d'application (API) pour que son environnement d'exécution puisse observer l'intégrité du conteneur et agir en conséquence. Ce prérequis est essentiel pour l'automatisation unifiée des mises à jour et cycles de vie des conteneurs, qui permet d'améliorer la résilience du système et l'expérience utilisateur.

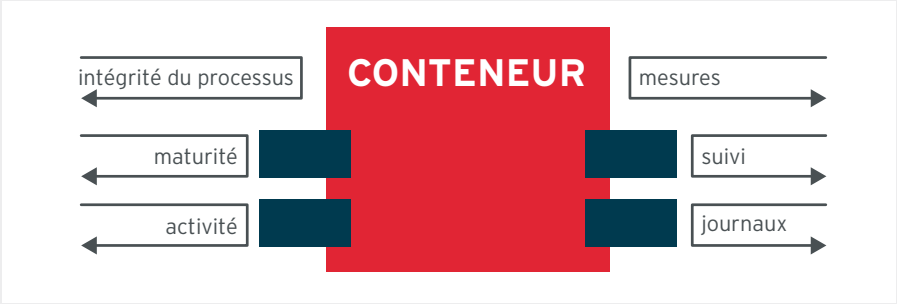

Figure 2 : un conteneur avec plusieurs API d'observabilité (les conteneurs ont plusieurs API conçues pour l'observabilité)

En pratique, votre application conteneurisée doit, au minimum, fournir des API pour les différents contrôles d'intégrité (activité et maturité). Même les meilleures applications doivent offrir d'autres possibilités d'observer l'état d'une application conteneurisée. L'application doit enregistrer les événements importants dans les flux standard STDERR (erreur standard) et STDOUT (sortie standard) pour que les journaux puissent être compilés par des outils tels que Fluentd et Logstash, puis intégrés dans des bibliothèques de traçage et de collecte de mesures, telles qu'OpenTracing, Prometheus et d'autres encore.

En résumé, traitez votre application comme une « boîte noire », mais sans oublier de la doter de toutes les API nécessaires pour aider la plateforme à observer et gérer au mieux votre application.

PRINCIPE DE LA CONFORMITÉ DU CYCLE DE VIE

Si le principe d'observabilité élevée énonce que votre conteneur doit fournir toutes les API nécessaires à la plateforme, le principe de la conformité du cycle de vie spécifie que votre application doit être en mesure de lire les événements provenant de la plateforme. Votre application ne doit toutefois pas se contenter de recevoir des événements, elle doit s'y conformer et y réagir. C'est de cet aspect qu'est tiré le nom du principe. C'est un peu comme si vous aviez un bouton « Écrire une API » dans votre application pour interagir avec la plateforme.

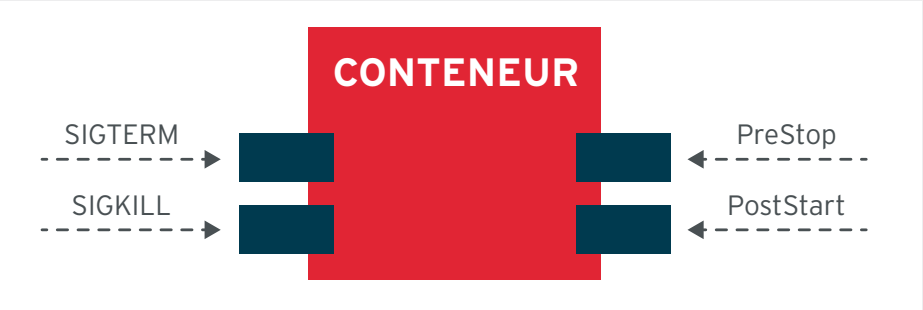

Figure 3 : un conteneur qui fournit des API et se conforme aux événements de la plateforme

La plateforme de gestion envoie plusieurs types d'événements censés vous aider à gérer le cycle de vie de votre conteneur. C'est à votre application de décider quels événements prendre en compte et s'il faut y réagir ou non.

Sachant que certains événements sont plus importants que d'autres. Par exemple, une application qui doit être arrêtée normalement doit détecter le signal SIGTERM et s'arrêter aussi vite que possible afin d'éviter un arrêt forcé par le signal SIGKILL qui suit le signal SIGTERM.

Il existe aussi d'autres événements, tels que PostStart et PreStop, qui peuvent avoir une importance pour la gestion du cycle de vie de votre application. Par exemple, certaines applications ont besoin de temps au démarrage avant de pouvoir recevoir des requêtes de service, alors que d'autres doivent libérer certaines ressources avant de pouvoir s'arrêter correctement.

PRINCIPE DE L'IMMUABILITÉ DE L'IMAGE

Les applications conteneurisées sont conçues pour être immuables et, une fois créées, elles ne doivent pas changer en fonction des différents environnements. Pour que cela soit possible, il faut recourir à des systèmes de stockage externes pour les données d'exécution et se baser sur des configurations externalisées qui varient en fonction des environnements, plutôt que créer ou modifier les conteneurs en fonction de chaque environnement. Tout changement dans une application conteneurisée doit engendrer la création d'une nouvelle image de conteneur et sa réutilisation au sein de tous les environnements. Ce même principe est également connu sous le nom de serveur/infrastructure immuable et utilisé pour la gestion des serveurs/hôtes.

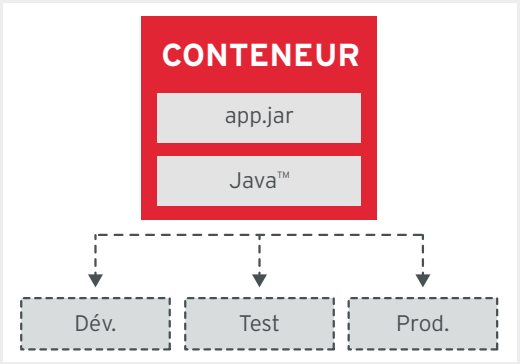

Figure 4 : les images immuables du conteneur sont utilisées au sein de tous les environnements

Ce principe vous évite de créer plusieurs images de conteneurs similaires pour différents environnements et vous permet de ne garder qu'une image de conteneur configurée pour chaque environnement. Il vous permet aussi d'effectuer des retours en arrière ou des restaurations par progression pendant les mises à jour applicatives, ce qui constitue un élément clé de l'automatisation conçue pour le cloud.

PRINCIPE DE LA DISPONIBILITÉ DES PROCESSUS

L'un des principaux avantages des applications conteneurisées réside dans le fait qu'un conteneur doit être aussi éphémère que possible et toujours prêt à être remplacé par un autre, à tout moment. De nombreuses raisons peuvent motiver le remplacement d'un conteneur : échec d'un contrôle d'intégrité, réduction de la taille d'une application, migration de conteneurs vers un nouvel hôte, manque de ressources sur la plateforme, etc.

Figure 5 : une application conteneurisée avec ouverture et fermeture rapides pour simplifier son remplacement

Aussi, les applications conteneurisées doivent conserver leur état de façon externalisée ou distribuée et redondante. Elles doivent aussi démarrer et s'arrêter rapidement, en restant toujours prêtes à subir une panne matérielle soudaine et totale.

Pour appliquer plus facilement ce principe, vous pouvez aussi créer de petits conteneurs. Les conteneurs hébergés dans un environnement conçu pour le cloud peuvent être planifiés et démarrés automatiquement sur différents hôtes. Les petits conteneurs démarrent plus rapidement, car avant de pouvoir redémarrer, les conteneurs doivent être physiquement copiés sur le système hôte.

PRINCIPE DE L'AUTOSUFFISANCE

Ce principe énonce qu'un conteneur doit contenir tout ce dont il a besoin, et ce, dès sa création. Le conteneur ne doit s'appuyer que sur le noyau Linux®. Toutes les autres bibliothèques doivent être ajoutées lors de la création du conteneur. Celui-ci doit aussi contenir l'environnement d'exécution du langage, la plateforme d'applications (le cas échéant) ainsi que les autres dépendances nécessaires à l'exécution de l'application conteneurisée.

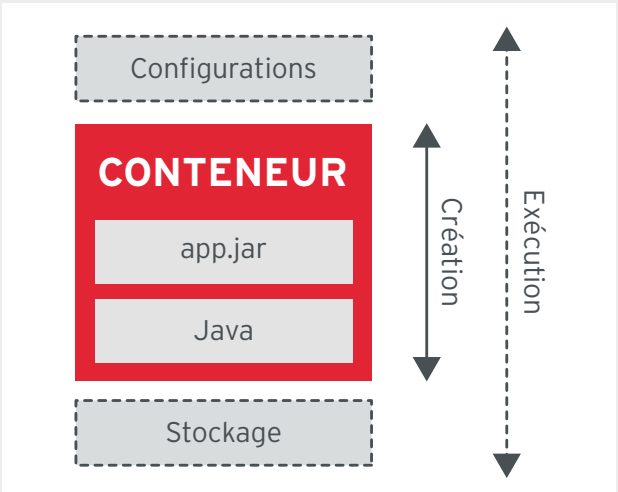

Figure 6 : les conteneurs devraient s'être sauvegardés sur toutes les dépendances au moment de la création, sauf dans les environnements spécifiques.

Les seules exceptions sont les éléments tels que les configurations, qui peuvent varier en fonction de l'environnement et qui doivent par conséquent être fournis au moment de l'exécution, par exemple via Kubernetes ConfigMap.

Certaines applications sont composées de plusieurs éléments conteneurisés. Par exemple, une application Web conteneurisée peut aussi nécessiter un conteneur de base de données. Par contre, ce principe n'implique pas la fusion des deux conteneurs. Au contraire, il recommande d'inclure dans chaque conteneur tout ce dont son contenu a besoin pour s'exécuter, par exemple le serveur Web pour l'application Web. Ensuite, lors de l'exécution, le conteneur de l'application Web dépend du conteneur de base de données et y accède selon les besoins.

PRINCIPE DU CONFINEMENT DE L'EXÉCUTION

Le principe de l'autosuffisance considère les conteneurs du point de vue de leur création et du binôme qu'ils forment avec leur contenu. Mais un conteneur ne se résume pas à une « boîte noire » unidimensionnelle sur le disque. Au moment de leur exécution, les conteneurs ont plusieurs dimensions, comme celle de l'utilisation de la mémoire, celle de l'utilisation du processeur, ainsi que d'autres dimensions de consommation de ressources.

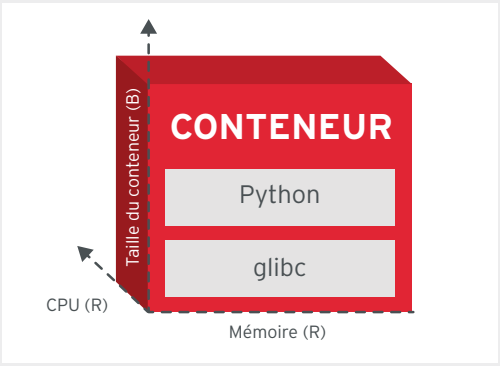

Figure 7 : un conteneur qui déclare les ressources dont il a besoin pour s'exécuter et qui respecte sa déclaration au moment de l'exécution

Le principe du confinement de l'exécution suggère que tous les conteneurs déclarent leurs besoins en matière de ressources et transmettent cette information à la plateforme. Le profil de ressources du conteneur doit contenir des informations sur le processeur, la mémoire, le réseau, l'impact du disque sur la planification, la mise à l'échelle automatique et la gestion de la capacité de la plateforme, ainsi que sur les contrats de niveau de service généraux du conteneur.

L'application ne doit cependant pas se contenter de transmettre les besoins en matière de ressources du conteneur. Elle doit aussi s'en tenir à ces ressources. Si elle n'en réclame pas plus, elle aura moins de risques d'être arrêtée ou migrée par la plateforme en cas de manque de ressources.

CONCLUSION

La conception pour le cloud n'est pas un état final à atteindre, c'est une véritable façon de travailler. Ce livre blanc a décrit un certain nombre de principes fondamentaux à suivre pour que vos applications conteneurisées puissent s'intégrer parfaitement dans le cloud.

Toutefois, ces principes ne suffisent pas à créer des applications conteneurisées de qualité. Vous devez aussi connaître les bonnes pratiques et techniques en matière de conteneurs. Si les principes décrits plus haut sont des fondamentaux qui s'appliquent à la plupart des cas d'utilisation, vous devrez faire preuve de bon sens pour savoir quand il est pertinent d'appliquer les bonnes pratiques énumérées ci-dessous. Voici quelques bonnes pratiques courantes en matière de conteneurs :

- **Créer des images de petite taille :** créez des images plus petites en supprimant les fichiers temporaires et en évitant d'installer des paquets inutiles. Ainsi, vous réduisez la taille du conteneur et vous accélérez sa création ainsi que la mise en réseau suite à la copie d'images de conteneurs.
- **Prendre en charge des ID utilisateurs arbitraires :** évitez d'utiliser la commande sudo ou de demander un ID utilisateur spécifique pour exécuter votre conteneur.
- **Marquer les ports importants :** lorsqu'il est possible d'indiquer les numéros des ports à l'exécution, faites-le en utilisant la commande EXPOSE pour simplifier l'utilisation de votre image par le logiciel ou un être humain.
- **Utiliser des volumes pour les données persistantes :** pour conserver des données après la destruction d'un conteneur, vous devez les écrire sur un volume.
- **Définir des métadonnées pour les images :** les métadonnées d'image organisées en balises, en étiquettes et en annotations simplifient l'utilisation de vos images de conteneur et améliorent ainsi l'expérience de vos développeurs lorsqu'ils utilisent vos images.
- **Synchroniser l'hôte et l'image :** certaines applications conteneurisées demandent la synchronisation du conteneur et de l'hôte au niveau de certains attributs, tels que l'heure et l'ID de la machine.

Vous trouverez ci-dessous une liste de liens vers des modèles et des bonnes pratiques qui vous aideront à mettre en œuvre plus facilement les principes expliqués ci-dessus :

- https://www.slideshare.net/luebken/container-patterns
- https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices
- http://docs.projectatomic.io/container-best-practices
- https://docs.openshift.com/enterprise/3.0/creating_images/guidelines.html
- https://www.usenix.org/system/files/conference/hotcloud16/hotcloud16_burns.pdf
- https://leanpub.com/k8spatterns/
- https://12factor.net/

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, de virtualisation, de stockage et de middleware fiables et performantes. Red Hat propose également des services d'assistance, de formation et de consulting reconnus. Situé au cœur d'un réseau mondial d'entreprises, de partenaires et de communautés Open Source, Red Hat participe à la création de technologies novatrices qui permettent de libérer des ressources pour la croissance et de préparer ses clients au futur de l'informatique.

EUROPE, MOYEN-ORIENT ET AFRIQUE (EMEA)
00800 7334 2835
fr.redhat.com
europe@redhat.com

TURQUIE
00800-448820640

ISRAËL
1-809 449548

ÉAU
8000-4449549

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