30/12/2024

L'approche Cloud-Native

HP
MICROSOFT

Table des matières

I. Time To Market

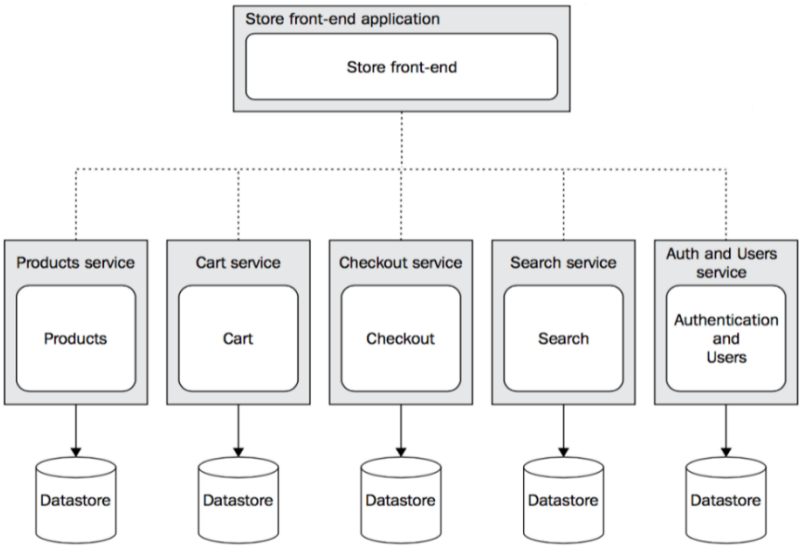
Le TTM (Time to Market) ou "délai de mise sur le marché", est le temps nécessaire pour développer, fabriquer et lancer un nouveau produit ou service. Il s'agit de la durée entre la conception d'un produit et sa mise sur le marché.

TTM est un indicateur permet de mesurer la rapidité de développement et de commercialisation d'un nouveau produit ou service par une entreprise.

II. Architecture micro-services

1.1. Définition

Les micro-services sont une **méthode de développement logiciel** utilisée pour concevoir une application comme un ensemble de services modulaires. Chaque module répond à un objectif métier spécifique et communique avec les autres modules.



1.2. Les caractéristiques

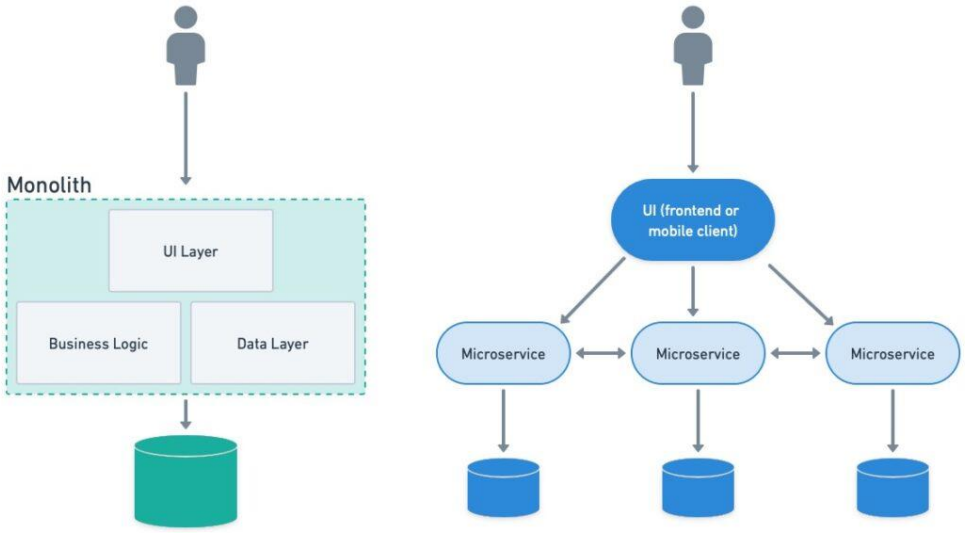
L'architecture micro-services présente les caractéristiques suivantes :

- L'application est décomposée en composants modulaires, faiblement couplés.
- L'application peut être distribuée sur plusieurs cloud et centres de données.
- L'ajout de nouvelles fonctionnalités ne nécessite que la mise à jour de ces micro-services individuels.

1.3. Les application Monolithique vs micro-services

Les applications traditionnelles ont été conçues comme des logiciels monolithiques. L'ajout de nouvelles fonctionnalités nécessite de reconfigurer et de mettre à jour tous les éléments de l'application, des processus aux communications en passant par la sécurité. Les applications monolithiques traditionnelles ont un long cycle de vie, sont rarement mises à jour et les changements affectent généralement l'ensemble de l'application. Ce processus coûteux et lourd retarde les avancées et les mises à jour dans le développement des applications d'entreprise.

Microservices est une méthode architecturale qui repose sur une série de services déployables indépendamment. Ces services ont leur propre logique métier et leur propre base de données avec un objectif précis. La mise à jour, les tests, le déploiement et la mise à l'échelle ont lieu dans chaque service.



1.4. Les avantages

a. Réduction du temps de développement

Grâce à un développement distribué, plusieurs micro-services peuvent être travaillés et développés simultanément. La durée du développement de l'application est de ce fait réduite car plusieurs développeurs peuvent intervenir sur le projet en même temps sans se déranger et nuire au travail de l'autre.

b. Évolutivité accrue

Avec les micro-services et leur indépendance en termes de développement et de déploiement, les développeurs peuvent mettre à jour un composant sans que cela n'ait d'incidence sur les autres composants de la solution.

Cela permet de faire évoluer l'application sereinement pour répondre à de nouveaux besoins et de nouvelles demandes sans entraver le fonctionnement du système.

c. Réduction des pannes

Les micro-services étant indépendants, lorsqu'un élément tombe en panne ou rencontre un problème, l'ensemble de l'application ne cesse pas de fonctionner contrairement aux applications monolithiques. Il est également plus facile d'identifier et de résoudre une panne dans ce type d'écosystème.

d. L’adaptabilité de chaque micro-service aux outils de travail

L’indépendance des composants offre la possibilité de paramétrer les microservices avec différents langages de programmation. Ainsi, les outils opérationnels utilisés par l’entreprise peuvent être rendus compatibles avec la solution globale.

De plus, l’évolution technologique ne constitue plus un problème avec ce type d’architecture étant donné que les langages d’implémentation de chacun des micro-services peuvent être changés pour s’adapter aux nouvelles innovations.

e. La flexibilité par rapport au cloud

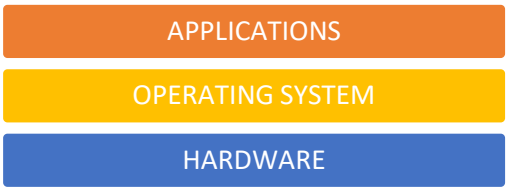
Une entreprise qui utilise l’architecture en micro-services peut réduire ou augmenter son usage du cloud en fonction de la charge de l’application. L’organisation est ainsi beaucoup plus flexible.

III. Conteneurisation

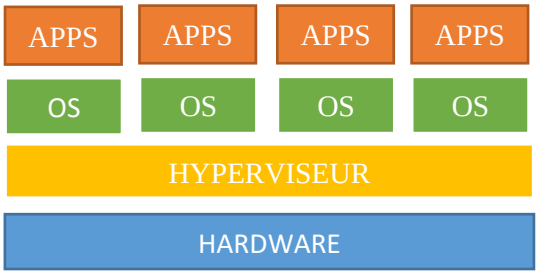
2.1. Définition de Virtualisation

La virtualisation est une technologie qui permet de créer plusieurs environnements simulés ou ressources dédiées à partir d'un seul système physique.

Le logiciel de virtualisation, appelé hyperviseur, est directement relié au matériel et vous permet de fragmenter ce système unique en plusieurs environnements sécurisés distincts. C'est ce que l'on appelle les machines virtuelles. Ces dernières exploitent la capacité de l'hyperviseur à séparer les ressources du matériel et à les distribuer convenablement.



Traditionnel Architecture



Virtual Architecture

2.2. Définition de conteneurisation

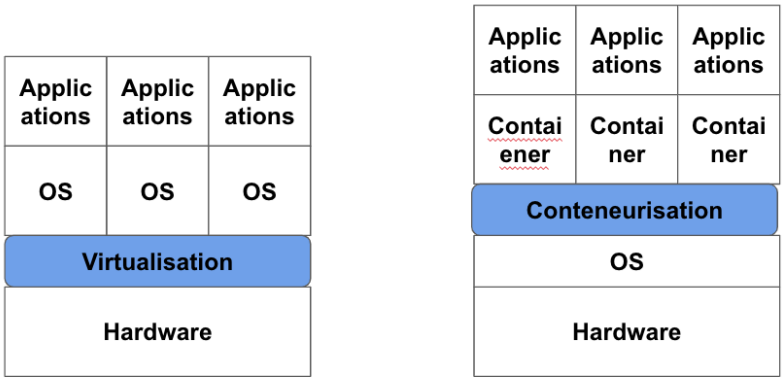
La conteneurisation consiste à rassembler le code du logiciel et tous ses composants (bibliothèques, Framework et autres dépendances) de manière à les isoler dans leur propre « conteneur ». Le logiciel ou l'application dans le conteneur peut ainsi être déplacé et exécuté de façon cohérente dans tous les environnements et sur toutes les infrastructures, indépendamment de leur système d'exploitation.

Avec les conteneurs, vous n'avez plus besoin de coder pour une plateforme ou un système d'exploitation en particulier, une méthode qui complique le déplacement des applications étant donné que le code n'est pas toujours compatible avec le nouvel environnement. De plus, ces transferts génèrent souvent des bogues, des erreurs et des problèmes qui font perdre du temps, diminuent la productivité et engendrent une grande frustration.

En plaçant une application dans un conteneur facile à déplacer entre les plateformes et infrastructures, vous pouvez l'utiliser n'importe où, car elle dispose de tout ce dont elle a besoin pour fonctionner.

2.3. Virtualisation vs Conteneurisation

La principale différence est que le conteneur fournit une virtualisation au niveau du système d'exploitation (virtualisation software) tandis que la machine virtuelle, quant à elle, fournit une virtualisation au niveau du matériel (virtualisation hardware).



Serveur virtualisé **Serveur conteneurisé**

2.4. Les avantages

La conteneurisation présente les avantages suivants :

a. Moins de surcharge

Les conteneurs requièrent moins de ressources système que les environnements de machines virtuelles classiques ou matérielles, car ils n'incluent pas les images du système d'exploitation.

b. Amélioration de la portabilité

Les applications qui s'exécutent dans des conteneurs peuvent être facilement déployées sur différents types de systèmes d'exploitation et de plateformes matérielles.

c. Meilleure cohérence des opérations

Les équipes DevOps savent que les applications qui se trouvent dans des conteneurs seront exécutées de la même manière, quel que soit l'endroit où elles sont déployées.

d. Efficacité accrue

Les conteneurs permettent de déployer, de corriger ou de faire évoluer les applications plus rapidement.

e. Optimisation du développement d'applications

Les conteneurs accélèrent les cycles de développement, de test et de production grâce à la méthodologie agile et DevOps

2.5. Orchestration des conteneurs

L’orchestration de conteneurs comprend le processus et les outils utilisés pour gérer un ensemble de conteneurs tout au long de leur cycle de vie, y compris la manière dont ils interagissent et communiquent entre eux à plus grande échelle. Ces processus sont souvent gérés par des plateformes d’orchestration de conteneurs.

IV. DevOps

3.1. Introduction

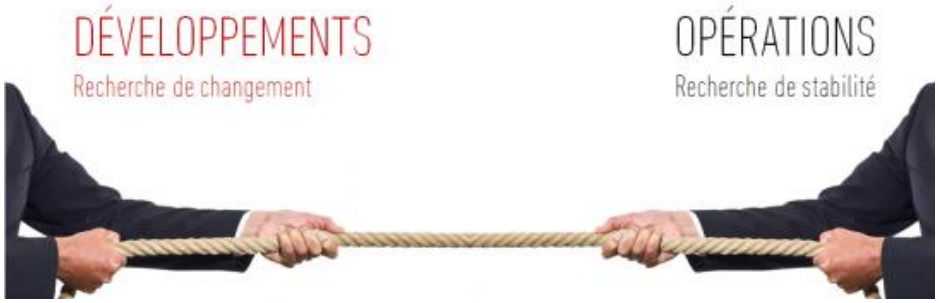
Rôle des Devs : Quand on parle des Devs, on parle de toute personne impliquée dans la fabrication du logiciel avant qu'il n'atteigne la production : les développeurs, les gestionnaires de produits, les testeurs,…

Rôle opérationnel : Il s'agit de tout le monde impliqué dans l'exploitation et la maintenance de la production : les ingénieurs systèmes, les DBAs, les ingénieurs réseaux, le personnel de sécurité, etc.

Les Développeurs (Dev) sont chargés de produire de l'innovation et délivrer les nouvelles fonctionnalités aux utilisateurs dès que possible. Les Ingénieurs d'opérations (Ops) sont chargés de s'assurer que les utilisateurs ont accès à un système stable, rapide et réactif :

- Les Dev recherchent des changements.
- Les Ops recherchent la stabilité organisationnelle.

Bien que le but ultime de Dev et Ops soit de rendre l'utilisateur satisfait (et potentiellement heureux, client payant) des systèmes qu'ils fournissent, leurs visions sur la façon de le faire restent opposées.



3.2. Définition

DevOps est les pratiques où les équipes de développement (Dev) et d'exploitation (Ops) participent ensemble à l'intégralité du cycle de vie de services : de la conception au support de production en passant par le développement.

3.3. Avant DevOps

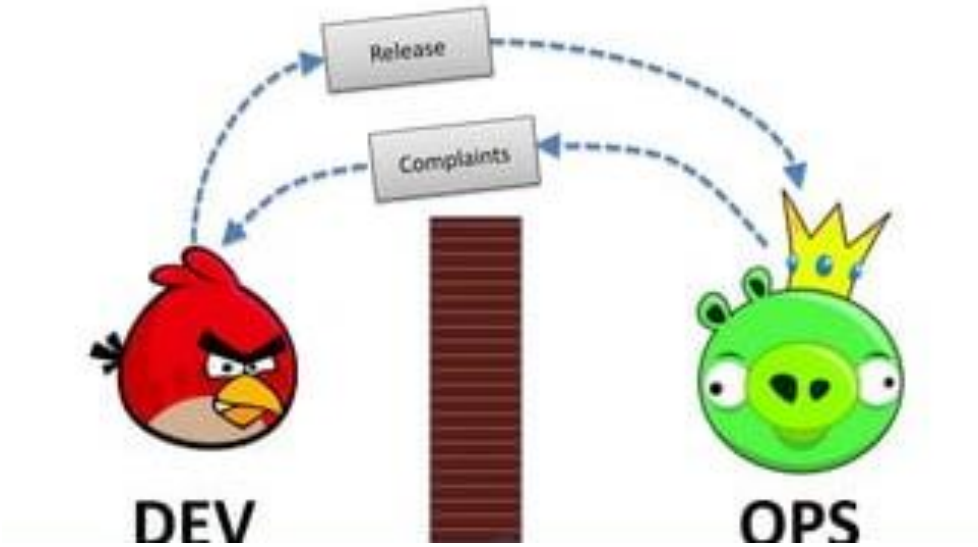
Quand les Dev et Ops vivaient dans leurs mondes isolés, cette opposition ne posait pas de problème. Les deux travaillaient « on a schedule » en collaborant seulement pendant les rares phases des releases :

- Les Devs savaient qu'ils devaient délivrer toutes les fonctionnalités désirées pour le jour « J », sinon il leur fallait attendre le prochain créneau.
- Ops avaient suffisamment de temps pour tester les nouvelles fonctionnalités avant de les envoyer en production. Ils pouvaient ensuite prendre des jours, voire des nuits, pour déployer ces releases aux utilisateurs.

Maintenant les choses ont changé, radicalement !

Les contraintes business ont évolué d'une telle manière que les utilisateurs veulent les nouvelles fonctionnalités très rapidement (Time To Market). Avec l'intégration et la livraison continue les développeurs font de nouvelles releases chaque jour. Il n'y a plus de jour « J » pour délivrer le produit comme auparavant.

Les Ops ne peuvent plus prendre des jours pour déployer des fonctionnalités une fois qu'elles ont été testées, néanmoins ils doivent maintenir la stabilité.



DevOps et ses pratiques visent à mettre fin à cette bataille entre Dév et Ops – pour parvenir à l'équilibre entre **l'innovation** et **la stabilité**. Dev et Ops doivent comprendre les bénéfices des paradigmes DevOps. Ils ont besoin de changer juste assez pour commencer à travailler ensemble et à

trouver le bon alignement entre Dev et Ops dont leur organisation a besoin, et de s'améliorer à partir de là.

3.4. Les fondements de devOps

La démarche DevOps est basée sur 4 fondements :

a. Réduire les cycles de livraison :

- Effectuer des mises en production (mises en service) fréquentes, déployer sans arrêt de service
- Par l'industrialisation de la chaine complète de production logicielle, et la livraison d'évolutions de petites tailles.

b. Optimiser les ressources :

- Par la standardisation, l'automatisation du déclenchement et de l'exécution du plus grand nombre de tâches (configuration, développement, déploiement…).

c. Améliorer la qualité :

- Le produit, le service rendu, correspond aux attentes de l'utilisateur et il est disponible
- Par des tests tout le temps et l'arrêt du processus en cas de défaut, par la refonte du logiciel sans altérer le service
- Par l'instrumentation et la supervision.

d. Mettre l'humain au centre du dispositif

- Par la mise en place d'une nouvelle culture fondée sur la collaboration et une recherche permanente de l'amélioration continue par l'apprentissage. Ce dernier fondement DevOps est à la fois le plus important mais aussi le plus difficile à mettre en place.

3.5. Cycle de vie DevOps

a. Planification

La planification permet de définir la valeur commerciale et les exigences attendues avant de lancer les développements.

Cette phase permet ainsi d'identifier et de suivre les travaux dans le temps. L'idée est d'avoir une vision claire des actions à réaliser pour atteindre les objectifs business fixés.

b. Création

La création inclut la conception logicielle et la création du code. En d'autres mots, c'est la phase de développement du produit.

c. Test

La phase de test consiste à vérifier le bon fonctionnement et faire la recette des développements réalisés durant l'étape de création.

d. Versionnage

Le versionnage aussi appelé distribution continu consiste à publier le code validé dans un référentiel de façon continue.

Cette phase permet ainsi de gérer les versions logicielles et à exploiter des outils automatisés pour compiler et intégrer le code en vue de sa mise en production.

e. Déploiement

Le déploiement aussi appelé déploiement continue, en complément du processus de distribution continue, automatise la publication d'une version prête pour la production dans un référentiel de code.

f. Exploitation

L'exploitation intervient une fois le logiciel déployé. La phase d'exploitation implique la maintenance et le dépannage des applications dans les environnements de production.

g. Surpervision

Cette phase permet d'identifier les problèmes affectant une version logicielle en production et de collecter les informations correspondantes.

V. CI-CD

4.1. Définition

L’approche CI/CD automatise le développement des applications. Tout en instaurant des éléments de surveillance pour s’assurer que l’application fonctionne bien. Et ce tout au long de la phase d’intégration, de test et de déploiement.

CI-CD est une partie du DevOps.

4.2. Intégration continue

Le Continuous integration (CI) est une pratique moderne de développement de logiciels dans laquelle des modifications incrémentielles du code sont apportées fréquemment et de manière fiable. Les étapes de construction et de test automatisées déclenchées par le CI garantissent la fiabilité des modifications du code qui sont fusionnées dans le référentiel.

Fonctionnement de l'intégration continue en 4 étapes :

Etape 1 : Le développeur code sa fonctionnalité ou le module qui lui a été défini. Après avoir conçu son module, il réalise des tests unitaires sur sa machine afin de s'assurer que tout fonctionne correctement dans son environnement. Il récupère le code sur le gestionnaire de source pour mettre à jour le code qu'il a. Il fusionne son code avec le code qu'il a récupéré puis résoudre les conflits, teste de nouveau son code sur sa machine et apporte des corrections éventuelles. Si tout est ok, il publie alors son code via son gestionnaire de code source.

Etape 2 : Le serveur d'intégration possède un service de détection de modification de code, suite à la dernière publication du développeur il prépare une tâche qui consiste à récupérer le dernier fragment de code développé et à l'intégrer dans sa plateforme. Il exécute cette tâche appelée communément Job.

Etape 3 : Une fois le job terminé, des rapports portant sur la qualité, la stabilité ou encore les divers bugs pouvant être rencontrés, sont générés et transmis à l'ensemble de l'équipe ou juste au développeur.

Etape 4 : L'équipe peut alors consulter ces rapports, les analyser, traiter des bugs s'il y en a, puis continuer à développer les autres phases du projet.

4.3. Déploiement continue

La Continuous delivery (CD) déploie toutes les modifications du code dans une construction vers l'environnement de test ou de préparation. La CD permet de libérer les builds dans l'environnement de production lorsque cela est nécessaire. En permettant à l'équipe de déployer à volonté, la CD réduit efficacement les délais de commercialisation.

VI. L'approche cloud-native

4.4. Définition

Le cloud native est une approche de la création et de l'exécution d'applications qui exploite les avantages du modèle de livraison du cloud computing. Lorsque les entreprises créent et exploitent des applications à l'aide d'une architecture "cloud native", elles commercialisent de nouvelles idées plus rapidement et répondent plus vite aux demandes des clients.

4.5. Les applications cloud-native

Une application "cloud native" est une application qui a été développé en tenant en compte les quatre principes fondamentaux :

1- **Microservice** : L'application sera organisée comme une collection de services librement services couplés.

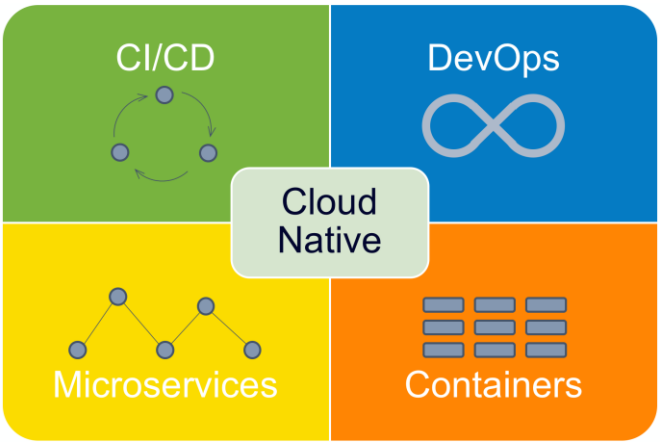
2- **Containerisation** : sont des paquets exécutables léger et autonome d'un logiciel qui comprend tout ce qui est nécessaire pour l'exécuter : code, moteur d'exécution, outils système, bibliothèques système, etc.

3- **DevOps** : est un ensemble de pratiques combinant le développement de logiciels (Dev) et les opérations (Ops) dont l'objectif est de raccourcir le cycle de vie du développement des systèmes et d'assurer une livraison continue avec une qualité logicielle élevée.

4- **CI-CD :**

- Continuous integration : La pratique consistant à fusionner le code de travail de tous les développeurs vers une ligne principale partagée plusieurs fois par jour. Chaque fusion est validée avant la fusion totale afin de détecter les problèmes éventuels. Cela permet de réduire les problèmes d'intégration et de permettre aux membres d'une équipe de travailler en collaboration pour fournir rapidement des logiciels de haute qualité.

- Continuous deployment : La pratique selon laquelle les équipes produisent des logiciels en cycles courts, ce qui garantit que le logiciel peut être publié de manière fiable à tout moment, et qu'il peut ensuite être déployé. L'objectif est de construire, tester et publier des logiciels rapidement et fréquemment.


CI/CD
DevOps
Cloud
Native
Microservices
Containers