Université Constantine 2
جامعة قسنطينة 2

Cloud Computing

– Cours 3 –

Chapitre 3 : La Conteneurisation

Dr. MENNOUR R.
Faculté des nouvelles technologies
rostom.mennour@univ-constantine2.dz

Etudiants concernés

Faculté/Institut	Département	Niveau	Spécialité
Nouvelles technologies	IFA	Master 1	SDIA

Objectifs du cours

- Décrire l'historique, la technologie et la terminologie derrière les conteneurs
- Différencier les conteneurs des serveurs bare metal et des machines virtuelles (VM)
- Illustrer les composants de Docker et leur interaction
- Identifier les caractéristiques d'une architecture de microservices
- Comprendre l'utilisation de K8s et l'orchestration des conteneurs

Plan du cours

- **L'ère des micro-services**
- **L'ère du DevOps**
- **Les conteneurs**
- **Conteneurisation Vs Virtualisation**
- **Docker**
- **L'orchestration avec K8s**
- **Conclusion**

Section 1 :
L’ère des micro-services

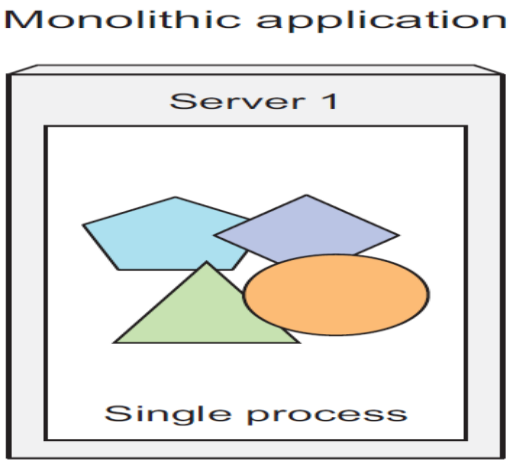
Passer des applications monolithiques aux micro-services

- **Les applications monolithiques**

Définition

Les applications monolithiques sont constituées de composants qui sont tous étroitement liés et doivent être développés, déployés et gérés comme une seule entité, car ils s'exécutent tous en tant que processus OS unique.

- Les modifications apportées à une partie de l'application nécessitent un redéploiement de l'ensemble de l'application.
- Le manque de frontières entre les parties entraîne une augmentation de la complexité et une détérioration conséquente de la qualité de l'ensemble du système.



Passer des applications monolithiques aux micro-services

- **Mettre à l’échelle les applications monolithiques**

  - L'exécution d'une application monolithique nécessite généralement un petit nombre de serveurs puissants pouvant fournir suffisamment de ressources pour l'exécution de l'application.
  - Pour faire face à des charges croissantes sur le système, vous devez soit faire évoluer les serveurs verticalement (Scaling up), soit mettre à l’échelle tout le système horizontalement (Scaling out).
  - Bien que la mise à l'échelle verticale ne nécessite généralement pas de modification de l'application, celle-ci coûte cher rapidement et, en pratique, a toujours une limite supérieure.
  - La mise à l'échelle horizontale, est relativement peu coûteuse du point de vue matériel, mais peut nécessiter de gros changements dans le code de l’application et n’est pas toujours possible.
  - Si une partie d’une application monolithique n’est pas évolutive, l’application entière devient non-évolutive (unscalable).

L’ère des micro-services

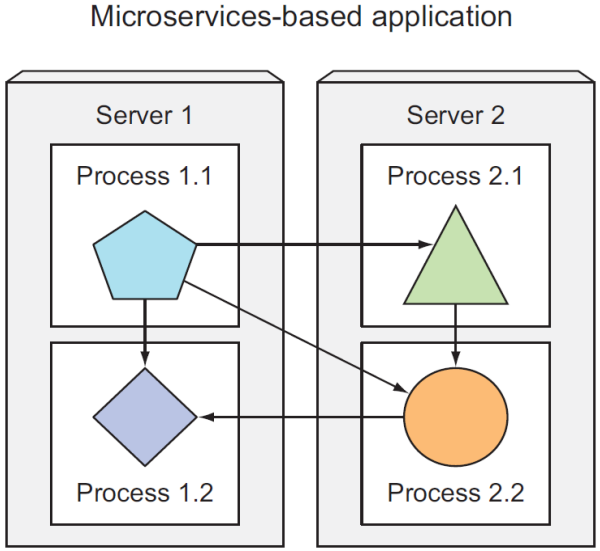
Passer des applications monolithiques aux micro-services

- **Division des applications monolithiques en micro-services**

Définition

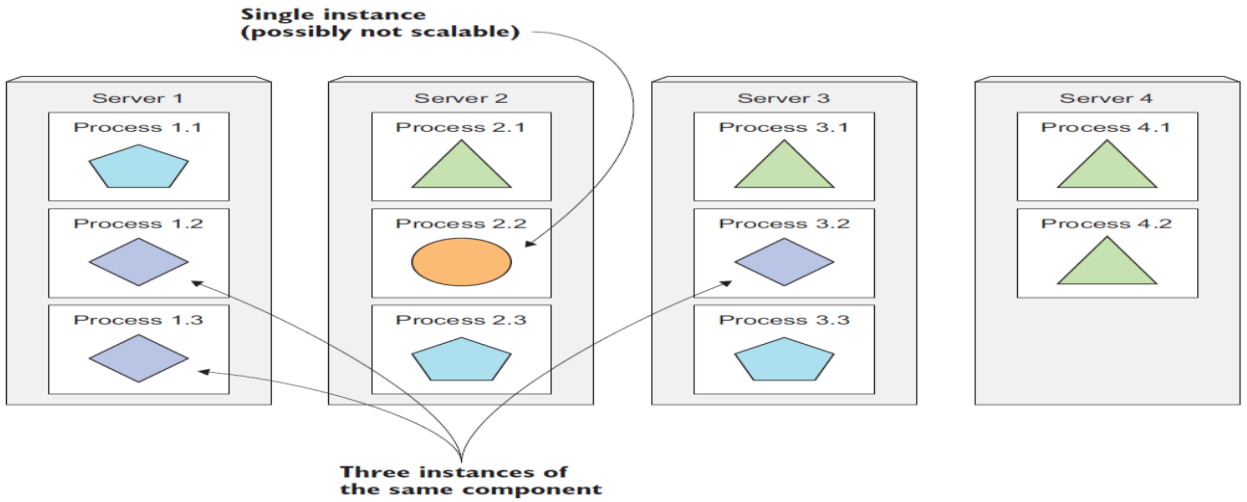
La définition de l'architecture de micro-services est simple: une application composée de «micro» services individuels. Un service, sous l’architecture micro-services, est une unité **indépendante** offrant une fonctionnalité **complète**.

- Chaque micro-service s'exécute en tant que processus indépendant et communique avec d'autres micro-services via des API simples.
- Il est possible de développer et de déployer chaque micro-service séparément.
- Chaque micro-service peut être écrit dans le langage le plus approprié pour le mettre en œuvre.
- La modification de l’un d’eux ne nécessite aucune modification ou redéploiement de tout autre service.



Passer des applications monolithiques aux micro-services

- **Mettre à l’échelle les micro-services**
  - La mise à l'échelle des micro-services est effectuée service par service, ce qui vous permet de ne mettre à l'échelle que les services nécessitant davantage de ressources, tout en laissant les autres à leur échelle d'origine.
  - Le fait de diviser l’application en micro-services vous permet de m.à.e horizontalement les pièces qui le permettent, et de m.à.e les pièces qui ne le permettent pas, verticalement plutôt qu'horizontalement.



Passer des applications monolithiques aux micro-services

- **Le déploiement des micro-services**

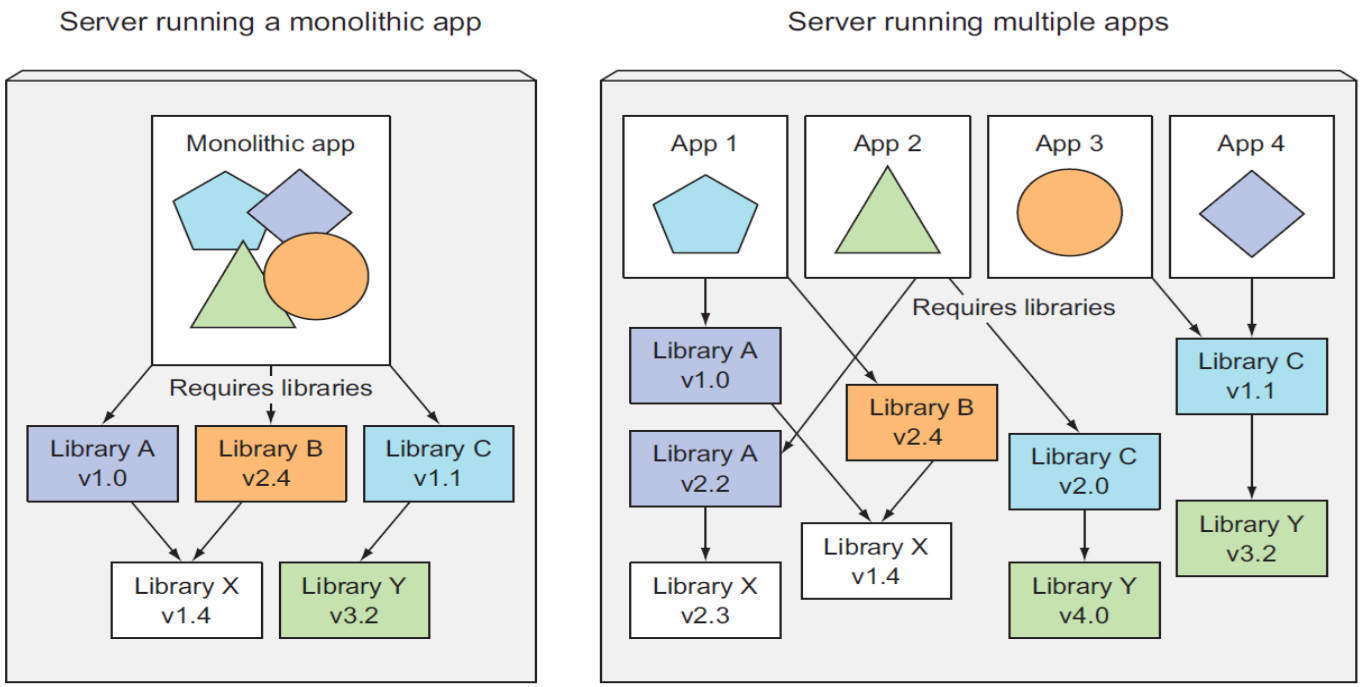
Remarque

Lorsque le nombre de composants augmente, les décisions de déploiement deviennent de plus en plus difficiles, car non seulement le nombre de combinaisons de déploiement augmente, mais également le nombre d'interdépendances entre les composants.

Les micro-services travaillent en équipe, ils doivent donc se retrouver et se parler. Lors du déploiement, quelqu'un ou quelque chose doit les configurer correctement pour leur permettre de fonctionner ensemble en tant que système unique. Avec le nombre croissant de micro-services, cela devient fastidieux et source d'erreurs, en particulier si vous considérez ce que les équipes ops / sysadmin doivent faire en cas de panne d'un serveur.

L'ère des micro-services

Passer des applications monolithiques aux micro-services



- Le déploiement d'applications liées dynamiquement qui nécessitent différentes versions de bibliothèques partagées et / ou d'autres spécificités de l'environnement peut rapidement devenir un cauchemar pour l'équipe ops qui les déploie et les gère sur des serveurs de production.
- Plus le nombre de composants à déployer sur le même hôte est important, plus il sera difficile de gérer toutes leurs dépendances pour satisfaire toutes leurs exigences.

Section 2 :
Les conteneurs

Comprendre les conteneurs

Qu’est ce qu' un conteneur ?

Les conteneurs

Comprendre les conteneurs

Before shipping containers

- Goods were shipped in a variety of vessels with no standardized weight, shape, or size.
- Transporting goods was slow, inefficient, and costly.

After shipping containers

- Uniformly sized shipping containers simplified loading, unloading, storing, and transferring between transport types.
- Abstraction of shipment details improved efficiency, increased productivity, and reduced costs.

Abstraction

Comprendre les conteneurs

- Lorsqu'une application est composée d'un petit nombre de composants volumineux, il est tout à fait acceptable d'attribuer une machine virtuelle (VM) dédiée à chaque composant et d'isoler leurs environnements en fournissant à chacun d'eux sa propre instance de système d'exploitation.
- Lorsque ces composants sont petits et que leur nombre commence à augmenter, vous ne pouvez pas leur attribuer leur propre machine virtuelle si vous ne voulez pas gaspiller des ressources matérielles et garder vos coûts en matériel bas.
- Mais il ne s'agit pas uniquement de gaspiller des ressources matérielles. Comme chaque ordinateur virtuel doit généralement être configuré et géré individuellement, un nombre croissant d'ordinateurs entraînent également un gaspillage de ressources humaines, car ils alourdissent considérablement la charge de travail des administrateurs système.

Les conteneurs

Comprendre les conteneurs

- Les conteneurs vous permettent d'exécuter plusieurs services sur le même ordinateur hôte, tout en exposant non seulement un environnement différent à chacun d'entre eux, mais également en les isolant les uns des autres, de la même manière que les machines virtuelles, mais avec beaucoup moins de surcharge.
- Lorsque ces composants sont petits et que leur nombre commence à augmenter, vous ne pouvez pas leur attribuer leur propre machine virtuelle si vous ne voulez pas gaspiller des ressources matérielles et garder vos coûts en matériel bas.
- Un processus exécuté dans un conteneur s'exécute dans le système d'exploitation de l'hôte, à l'instar de tous les autres processus. Mais le processus dans le conteneur est toujours isolé des autres processus.
- Pour le processus lui-même, il lui semble que ce soit le seul en cours d'exécution sur la machine et dans son système d'exploitation.

Remarque (Important)

La principale caractéristique du conteneur est l'isolation.

Les conteneurs

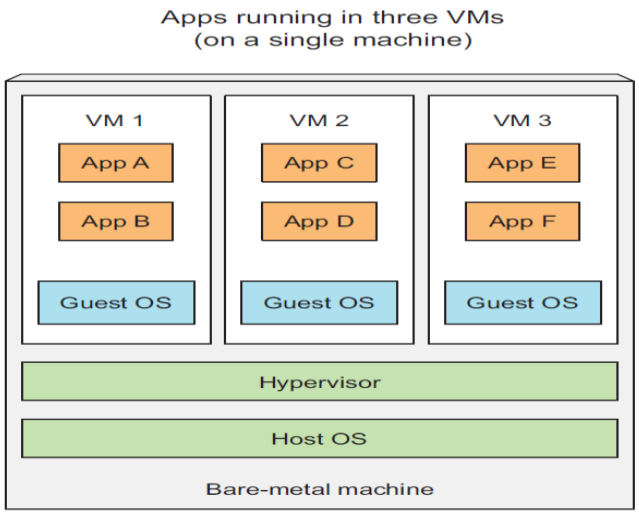
Conteneurisation Vs Virtualisation

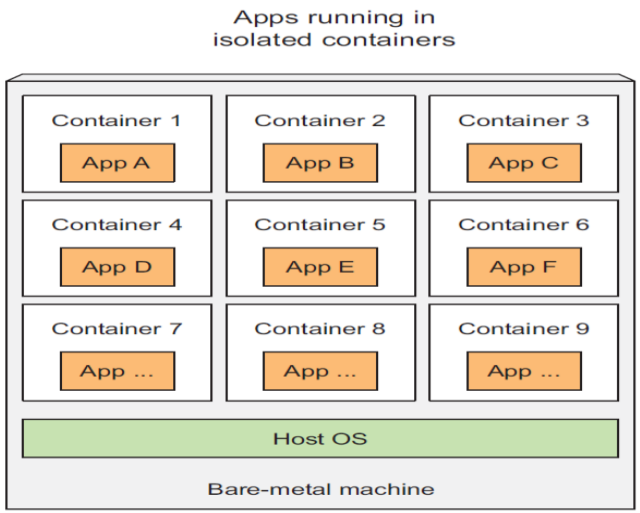
Virtualisation

Les MV sont lourdes
Nombre réduit d'instances
Lance sont propre OS
Groupe plusieurs applications
Les applications passent par l'hyperviseur
Isolation complète

Conteneurisation

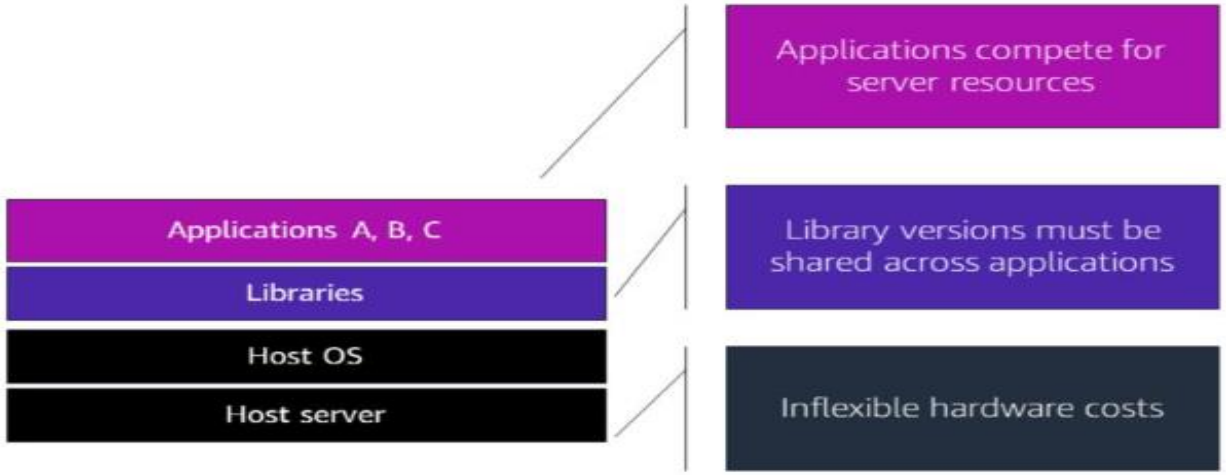
Les conteneurs sont légers
Nombre plus élevé d'instances
S'exécute sur l'OS hôte
Exécute une seule application
Les applications s'exécutent directement
Isolation partielle





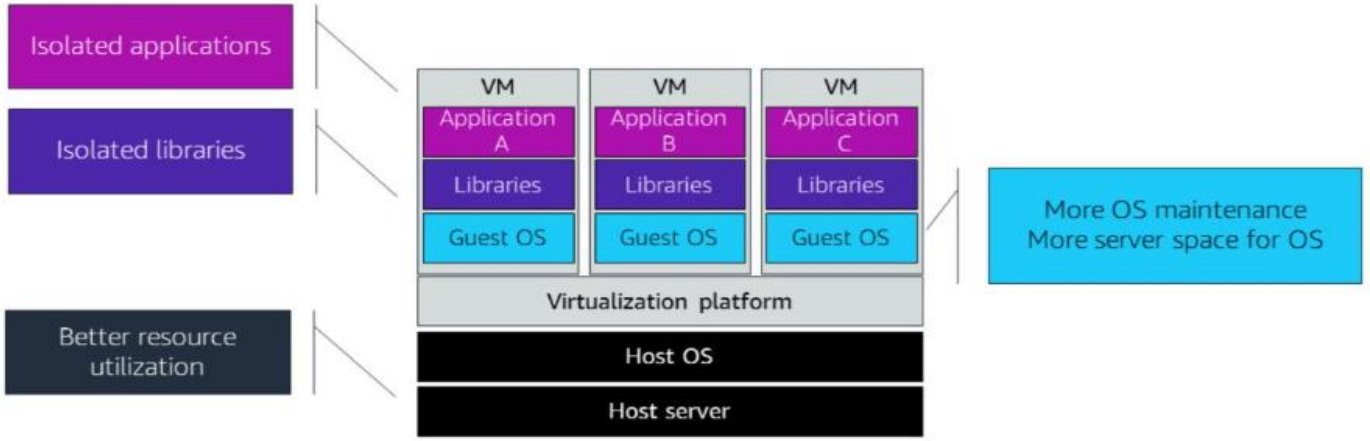
Les conteneurs

Evolution du déploiement



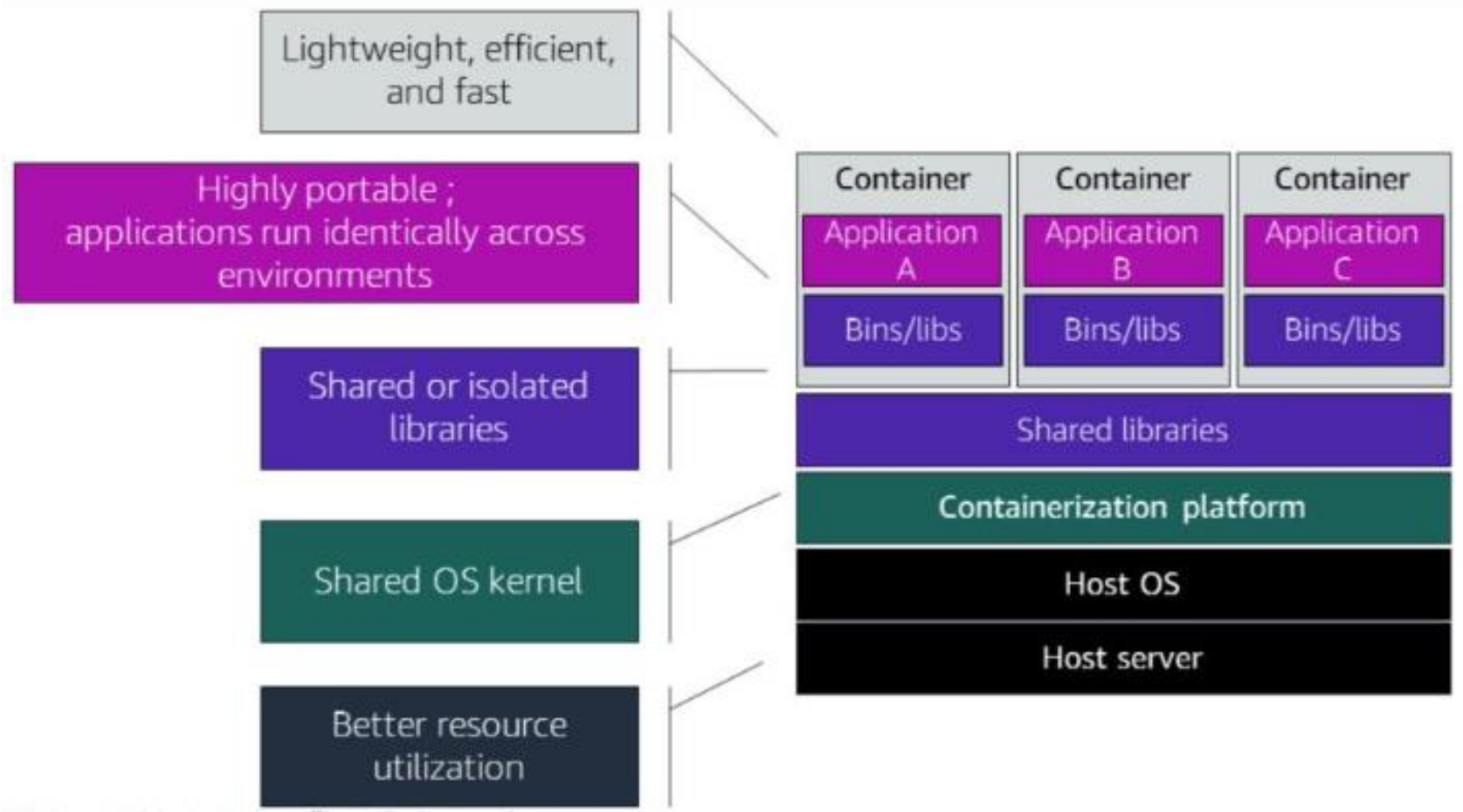
Bare metal

VMs



Les conteneurs

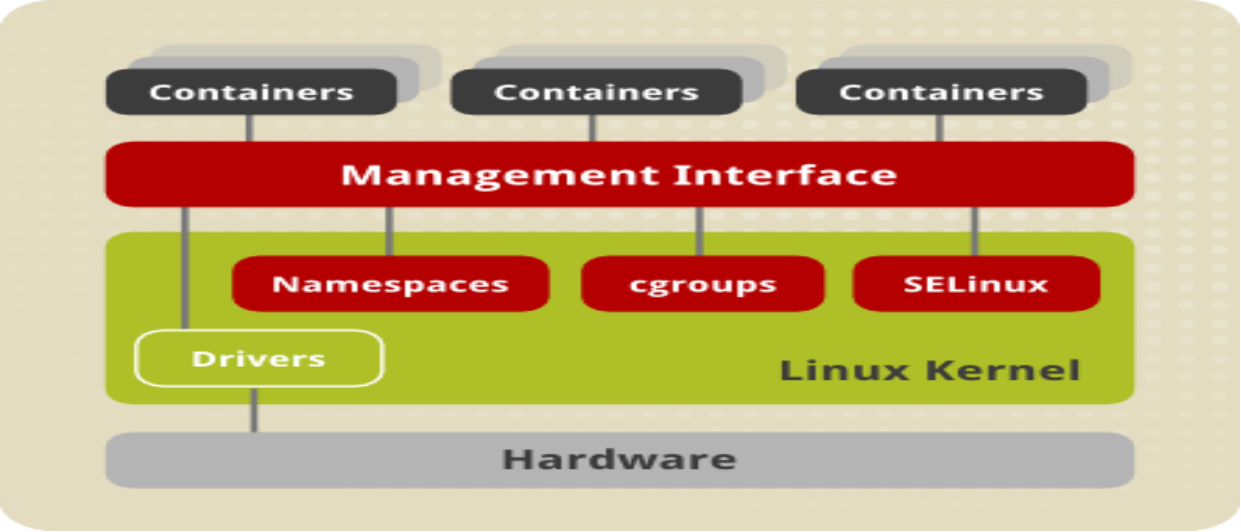
Evolution du déploiement



L’isolation des composants dans les conteneurs

comment exactement les conteneurs peuvent isoler les processus s’ils fonctionnent sur le même système d’exploitation ?

- Linux Namespaces : s'assure que chaque processus voit sa propre vue personnelle du système (fichiers, processus, interfaces réseau, nom d'hôte, etc.).
- Linux Control Groups (cgroups) : limite la quantité de ressources que le processus peut consommer (CPU, mémoire, bande passante réseau, etc.).



Les conteneurs

L’isolation des composants dans les conteneurs

- **Linux Namcespaces**

  - Par défaut, chaque système Linux a initialement un seul espace de noms. Toutes les ressources système, telles que les systèmes de fichiers, les ID de processus, les ID d'utilisateur, les interfaces réseau et autres, appartiennent à l'espace de noms unique.
  - Vous pouvez créer des espaces de noms supplémentaires et organiser les ressources entre eux. Lorsque vous exécutez un processus, vous l'exécutez dans l'un de ces espaces de noms.
  - Il existe plusieurs types d’espaces de noms, de sorte qu’un processus n’appartient pas à un seul espace de noms, mais à un espace de nom de chaque type.

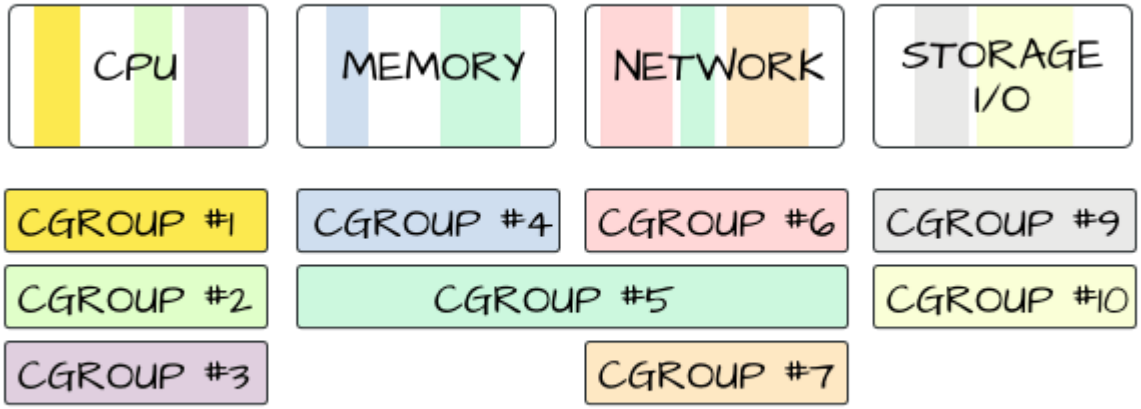
- **Mount (mnt)**
- **Process ID (pid)**
- **Network (net)**
- **Inter-process communication (ipc)**
- **UTS**
- **User ID (user)**

Les conteneurs

L’isolation des composants dans les conteneurs

- **Linux Control Groups**

  - Limite la quantité de ressources système qu'un conteneur peut consommer.
  - Un processus ne peut pas utiliser plus que la quantité configurée de CPU, de mémoire, de bande passante réseau, etc.
  - Les processus ne peuvent pas détourner des ressources réservées à d'autres processus, ce qui est similaire à l’exécution de chaque processus sur une machine distincte.

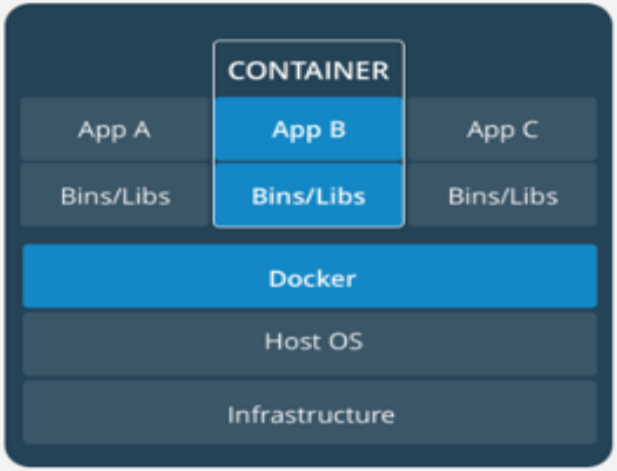


Section 4 :
Docker

Comprendre les concepts Docker

Définition

Docker est une plate-forme d'emballage, de distribution et d'exécution d'applications. Il vous permet d'emballer votre application avec tout son environnement.



Comprendre les concepts Docker

Docker comprend trois concepts principaux:

- Images : une image de conteneur Docker est un élément dans lequel vous empaquetez votre application et son environnement. Il contient le système de fichiers qui sera disponible pour l'application et d'autres métadonnées, telles que le chemin d'accès à l'exécutable à exécuter lors de l'exécution de l'image.

- Registres : Un registre Docker est un référentiel qui stocke vos images Docker et facilite le partage de ces images entre différentes personnes et ordinateurs. Lorsque vous créez votre image, vous pouvez l'exécuter sur l'ordinateur sur lequel vous l'avez créé ou vous pouvez transférer (push) l'image dans un registre, puis la télécharger (pull) sur un autre ordinateur et l'exécuter à cet emplacement. Certains registres sont publics, ce qui permet à quiconque d'en extraire des images, tandis que d'autres sont privés et uniquement accessibles à certaines personnes ou à certaines machines.

- Conteneurs : Un conteneur Docker est un conteneur Linux standard créé à partir d'une image de conteneur Docker. Un conteneur en cours d'exécution est un processus en cours d'exécution sur l'hôte exécutant Docker, mais il est complètement isolé de l'hôte et de tous les autres processus qui s'exécutent sur ce dernier. Le processus est également limité en ressources, ce qui signifie qu'il ne peut accéder et utiliser que la quantité de ressources (CPU, RAM, etc.) qui lui sont allouées.

Caractéristiques

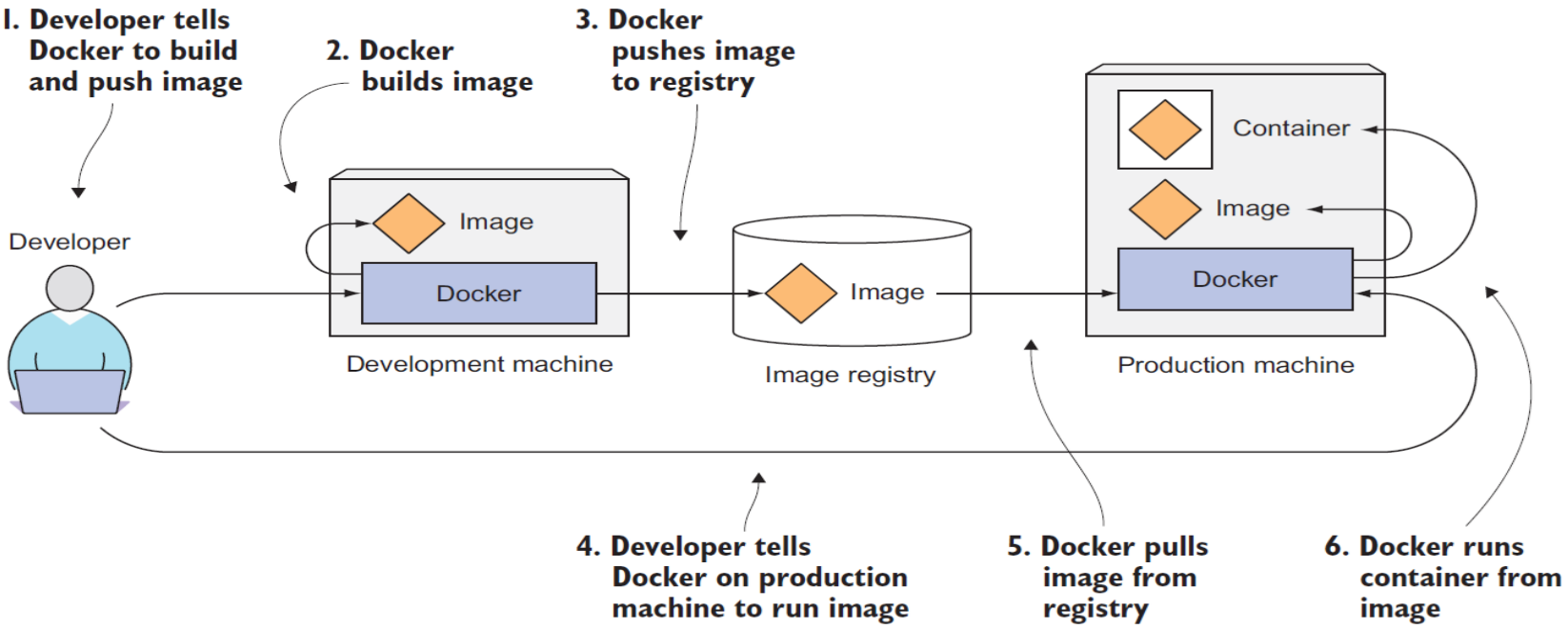
La liste suivante résume certains des avantages importants des conteneurs Docker :

- Docker est un environnement d'application d'exécution portable.
- Vous pouvez regrouper une application et ses dépendances dans un seul artefact immuable appelé image.
- Après avoir créé une image de conteneur, elle peut aller partout où Docker est pris en charge.
- Vous pouvez exécuter simultanément différentes versions d'application avec différentes dépendances.

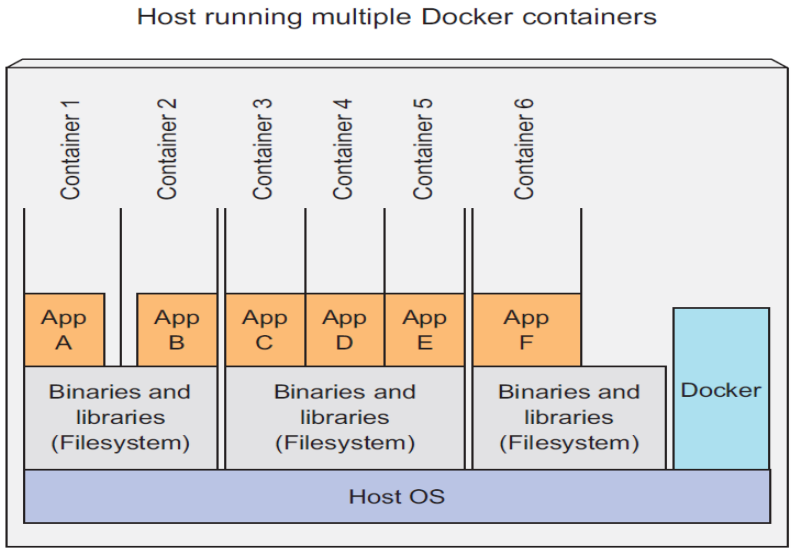
Ces avantages conduisent à des cycles de développement et de déploiement beaucoup plus rapides, ainsi qu'à une meilleure utilisation et efficacité des ressources. Toutes ces capacités sont liées à l'agilité.

Construction, distribution et exécution d'une image Docker

- Le développeur construit d'abord une image puis la pousse dans un registre.
- On peut ensuite extraire l'image sur n'importe quelle autre machine exécutant Docker et exécuter l'image.
- Docker crée un conteneur isolé basé sur l'image et exécute l'exécutable binaire spécifié dans le cadre de l'image.

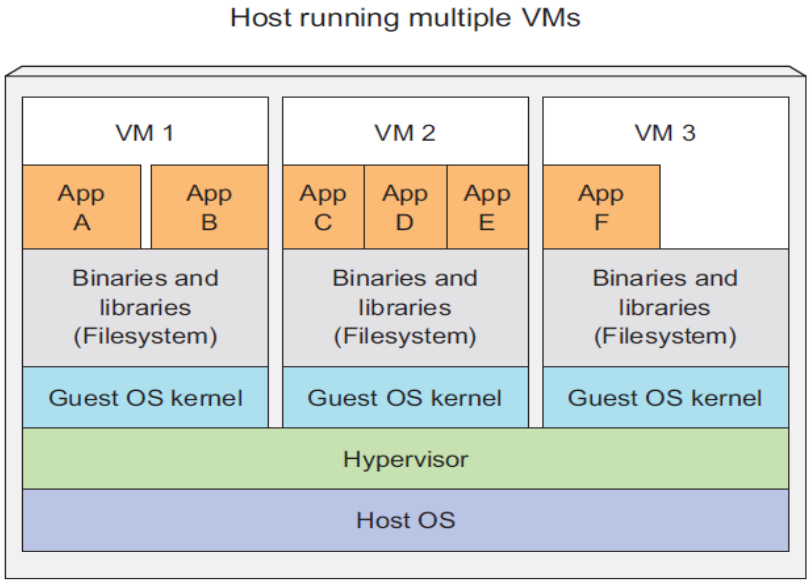


Comparaison entre images docker et images VM



les six mêmes applications fonctionnant à la fois dans des VM et en tant que conteneurs Docker. Vous remarquerez que les applications A et B ont accès aux mêmes fichiers binaires et aux mêmes bibliothèques

Cela est évident dans la machine virtuelle, car les deux applications voient le même système de fichiers (celui de la machine virtuelle). Mais nous avons dit que chaque conteneur a son propre système de fichiers isolé. Comment les applications A et B peuvent-elles partager les mêmes fichiers?



Les niveaux d'image dans Docker

- Les images dans Docker se composent de plusieurs couches (layers).
- Des images différentes peuvent contenir exactement les mêmes couches, car chaque image Docker est construite sur une autre image et deux images différentes peuvent utiliser la même image parente comme base.
- Chaque couche n'est stockée qu'une seule fois.
- Deux conteneurs créés à partir de deux images basées sur les mêmes couches de base peuvent donc lire les mêmes fichiers, mais si l'un d'eux écrit sur ces fichiers, l'autre ne voit pas ces modifications (les couches d'image sont en lecture seule).
- Lorsqu'un conteneur est exécuté, une nouvelle couche d'écriture est créée par-dessus les couches de l'image.
- Lorsque le processus dans le conteneur écrit dans un fichier situé dans l'une des couches sous-jacentes, une copie de l'intégralité du fichier est créée dans la couche la plus haute et le processus écrit dans la copie.

Les limites

- Si une application conteneurisée requiert une version de noyau spécifique, il est possible que celle-ci ne fonctionne pas sur toutes les machines. Si une machine exécute une version différente du noyau Linux ou ne dispose pas des mêmes modules de noyau, l'application ne peut pas s'exécuter sur celle-ci.
- Bien que les conteneurs soient beaucoup plus légers que les machines virtuelles, ils imposent certaines contraintes aux applications qui y sont exécutées. Les VM n'ont pas de telles contraintes, car chaque VM exécute son propre noyau.
- une application conteneurisée conçue pour une architecture matérielle spécifique ne peut s'exécuter que sur d'autres ordinateurs dotés de la même architecture. Vous ne pouvez pas conteneuriser une application construite pour l'architecture x86 et s'attendre à ce qu'elle s'exécute sur une machine ARM, même si elle exécute également Docker. Vous avez toujours besoin d'une machine virtuelle pour cela.

Section 5 :
L’orchestration avec Kubernetes (K8s)

L’orchestration avec Kubernetes (K8s)

Comprendre les origines

- Google a probablement été la première entreprise à se rendre compte qu'il lui fallait un moyen bien plus efficace de déployer et de gérer ses composants logiciels et son infrastructure pour s'adapter à l'échelle mondiale.
- C’est l’une des rares entreprises au monde à gérer des centaines de milliers de serveurs et à gérer des déploiements à une telle échelle.
- Au fil des années, Google a mis au point un système interne appelé Borg (puis un nouveau système appelé Omega), qui aidait à la fois les développeurs d'applications et les administrateurs système à gérer ces milliers d'applications et de services.
- Cela les a également aidés à utiliser beaucoup plus efficacement leur infrastructure.
- Après avoir gardé le secret de Borg et Omega pendant une décennie, Google a présenté en 2014 Kubernetes, un système open source basé sur l'expérience acquise au travers de Borg, Omega et d'autres systèmes internes de Google.

Comprendre K8s

Définition

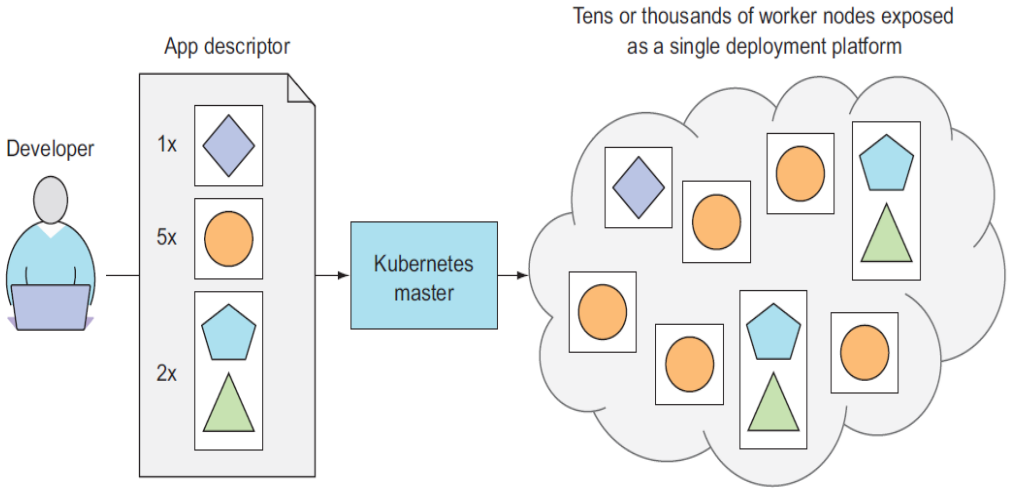
Kubernetes est une plate-forme portable, extensible et open-source pour l'automatisation et la gestion de charges de travail et de services conteneurisés. Son écosystème est vaste et en croissance rapide. Les services, le support et les outils Kubernetes sont largement disponibles.

kubernetes

Le nom Kubernetes vient du grec, ce qui signifie barreur ou pilote

L’orchestration avec Kubernetes (K8s)

Comprendre K8s



Lorsque le développeur soumet une liste d'applications au maître, Kubernetes les déploie sur le cluster de nœuds de travail. Le nœud sur lequel un composant atterrit importe peu (et ne devrait pas importer), que ce soit pour le développeur ou pour l’administrateur système.

Le développeur peut spécifier que certaines applications doivent être exécutées ensemble et Kubernetes les déploiera sur le même nœud de travail. D'autres seront réparties autour du cluster, mais elles peuvent se parler de la même manière, quel que soit le lieu où elles sont déployées.

L’orchestration avec Kubernetes (K8s)

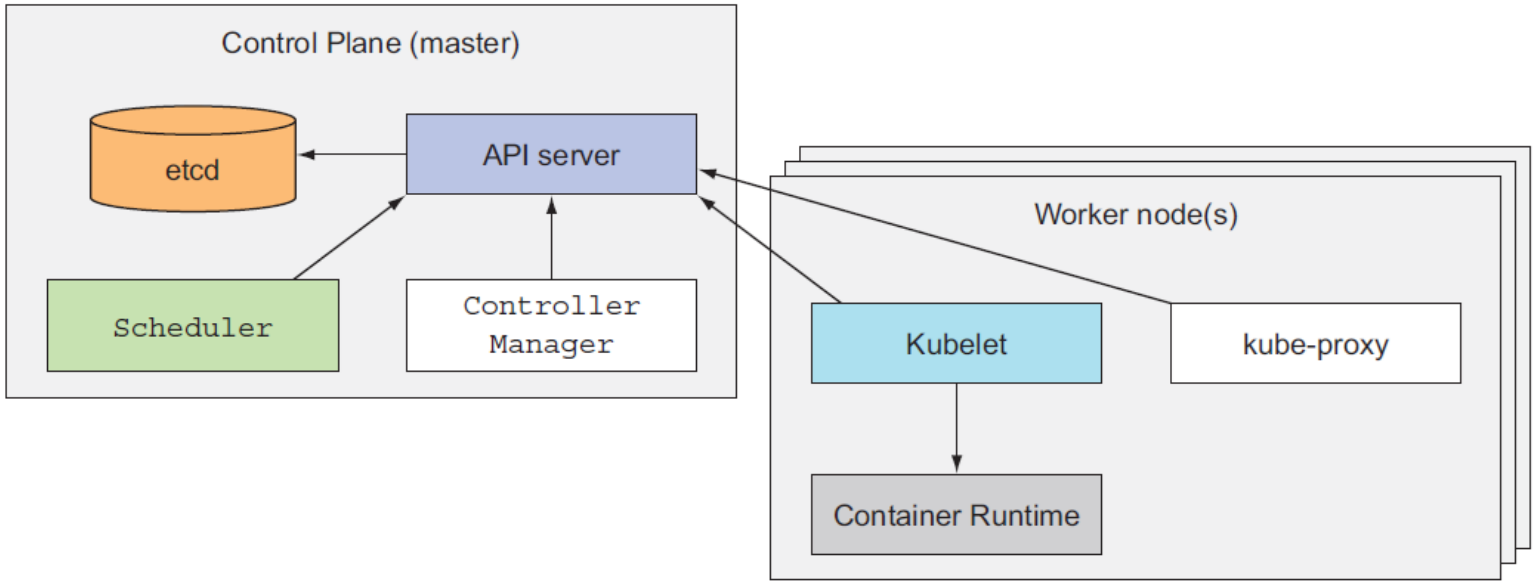
A quoi sert K8s

- Découverte de service et équilibrage de charge : Kubernetes peut exposer un conteneur en utilisant le nom DNS ou en utilisant sa propre adresse IP. Si le trafic vers un conteneur est élevé, Kubernetes peut équilibrer la charge et répartir le trafic réseau afin que le déploiement soit stable.
- Orchestration du stockage : Kubernetes vous permet de monter automatiquement un système de stockage de votre choix, tel que des stockages locaux, des fournisseurs de cloud public, etc.
- Déploiement et restauration automatisés : Vous pouvez décrire l'état souhaité pour vos conteneurs déployés à l'aide de Kubernetes et modifier l'état actuel à l'état souhaité à une vitesse contrôlée. Par exemple, vous pouvez automatiser Kubernetes pour créer de nouveaux conteneurs pour votre déploiement, supprimer les conteneurs existants et adopter toutes leurs ressources dans le nouveau conteneur.
- Emballage automatique : Vous fournissez à Kubernetes un cluster de noeuds qu'il peut utiliser pour exécuter des tâches conteneurisées. Vous indiquez à Kubernetes la quantité de ressources processeur et mémoire (RAM) requise par chaque conteneur. Kubernetes peut adapter des conteneurs sur vos nœuds pour utiliser au mieux vos ressources.
- Auto-guérison : Kubernetes redémarre les conteneurs qui échouent, les remplace, supprime les conteneurs qui ne répondent pas à la vérification de votre santé définie par l'utilisateur et ne les publie pas tant qu'ils ne sont pas prêts.
- Secret et gestion de la configuration : Kubernetes vous permet de stocker et de gérer des informations sensibles, telles que des mots de passe, des jetons et des clés ssh.

L'architecture de K8s

Au niveau matériel, un cluster Kubernetes est composé de plusieurs nœuds, qui peuvent être divisés en deux types :

- Le nœud maître, qui héberge le plan de contrôle Kubernetes qui contrôle et gère l'ensemble du système Kubernetes.
- Les Nœuds de travail exécutant les applications que vous déployez



L’architecture de K8s

- **Le plan de contrôle (Control plane)**

Le plan de contrôle est ce qui contrôle le cluster et le rend fonctionnel. Il se compose de plusieurs composants pouvant s'exécuter sur un seul nœud maître ou être répartis sur plusieurs nœuds et répliqués pour assurer une haute disponibilité. Ces composants sont :

- Le serveur API Kubernetes, avec lequel vous et les autres composants du plan de contrôle communiquez.
- Le planificateur (Scheduler), qui planifie vos applications (attribue un nœud de travail à chaque composant déployable de votre application).
- Le Controller Manager, qui exécute des fonctions au niveau du cluster, telles que la réplication de composants, le suivi des nœuds de travail, la gestion des défaillances de nœud, etc.
- etcd, un magasin de données distribué fiable qui stocke en permanence la configuration du cluster.

L’architecture de K8s

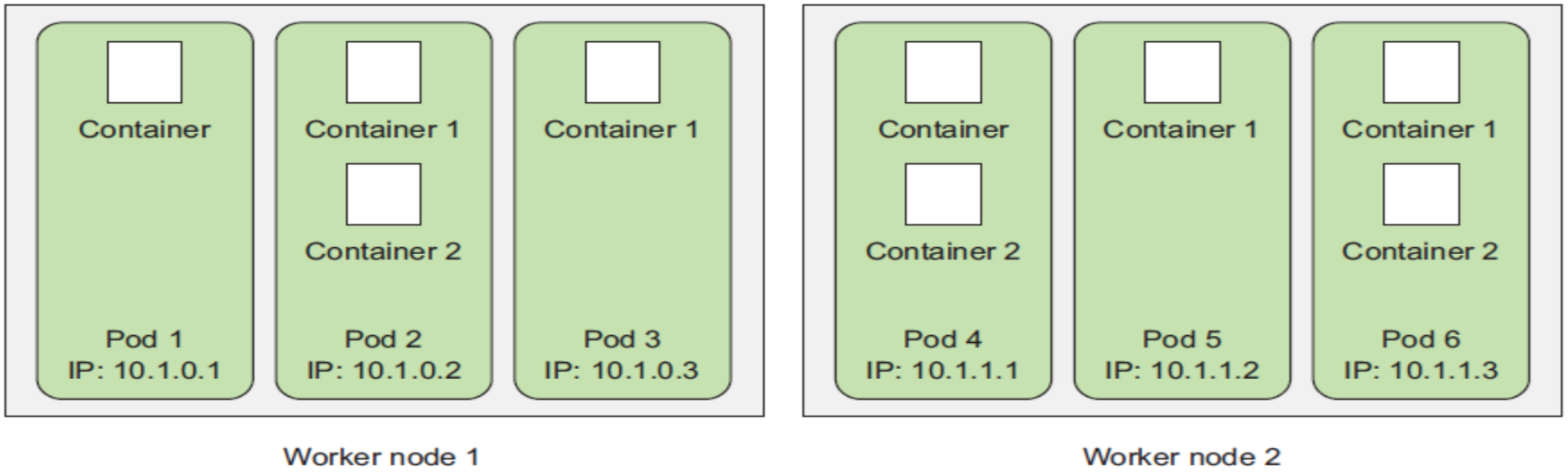
- **Les nœuds de travail**

Les nœuds de travail sont les machines qui exécutent vos applications conteneurisées. L'exécution, la surveillance et la fourniture de services à vos applications sont effectuées à l'aide des composants suivants:

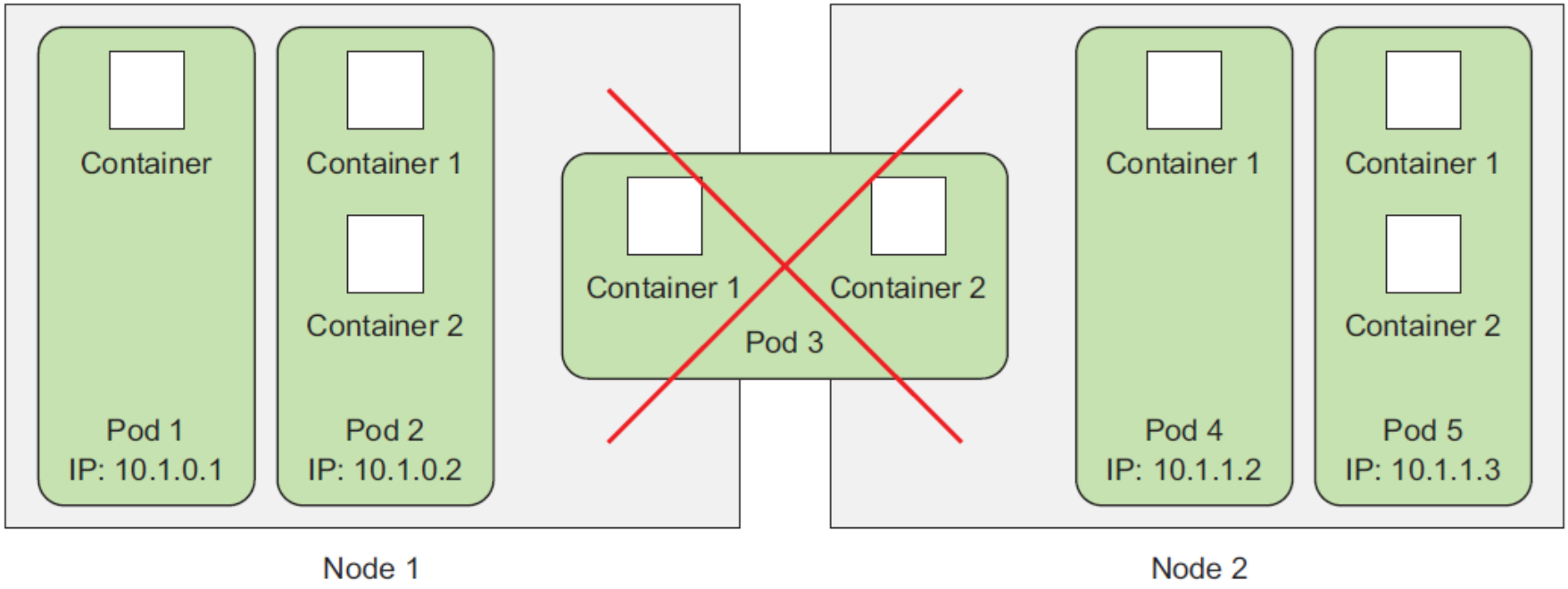
- Docker, rkt ou un autre *environnement d'exécution de conteneur*, qui exécute vos conteneurs.
- Kubelet, qui communique avec le serveur d'API et gère les conteneurs sur son nœud.
- Le proxy de service Kubernetes (kube-proxy), qui équilibre le trafic réseau entre les composants de l'application.

Le concept de Pod

- Kubernetes ne traite pas directement les conteneurs individuels. Au lieu de cela, il utilise le concept de plusieurs conteneurs colocalisés. Ce groupe de conteneurs s'appelle un pod.
- Un pod est un groupe d'un ou de plusieurs conteneurs étroitement liés qui seront toujours exécutés ensemble sur le même nœud de travail et dans le même espace de noms Linux. Chaque module ressemble à une machine logique distincte avec ses propres adresses IP, nom d’hôte, processus, etc. exécutant une seule application.
- L'application peut être un processus unique, s'exécutant dans un conteneur unique ou un processus d'application principal et des processus de support supplémentaires, chacun s'exécutant dans son propre conteneur.



Le concept de Pod

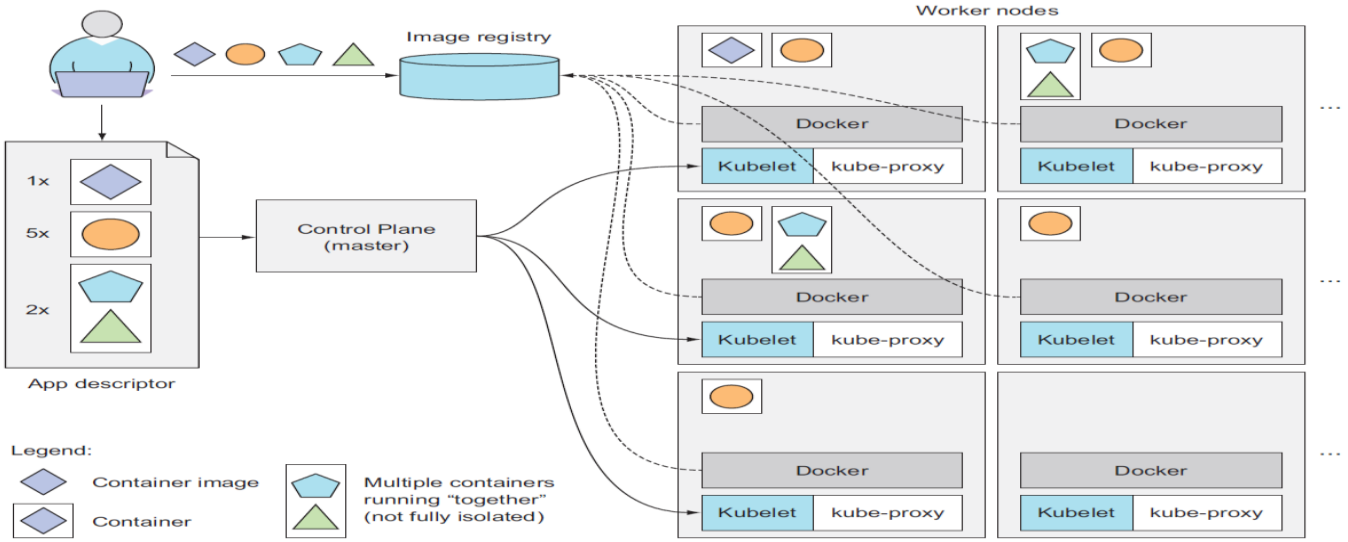


Remarque (Important)

L'essentiel sur les pods est que lorsqu'un pod contient plusieurs conteneurs, ils sont tous toujours exécutés sur un seul nœud de travail - ils ne couvrent jamais plusieurs nœuds de travail.

Exécuter des applications avec Kubernetes

- Pour exécuter une application dans Kubernetes, vous devez tout d'abord l’organiser dans une ou plusieurs images de conteneur, les envoyer dans un registre d'images, puis envoyer une description de votre application sur le serveur d'API Kubernetes.
- Lorsque le serveur API traite la description de votre application, le planificateur planifie les groupes de conteneurs spécifiés sur les nœuds de travail disponibles en fonction des ressources de calcul requises par chaque groupe et des ressources non allouées sur chaque nœud à ce moment.
- Le Kubelet sur ces noeuds donne ensuite des instructions au conteneur (Docker, par exemple) pour extraire les images de conteneur requises et exécuter les conteneurs.



L’orchestration avec Kubernetes (K8s)

Les avantages de K8s

- SIMPLIFIER LE DÉPLOIEMENT D'APPLICATIONS
- MIEUX UTILISER LE MATÉRIEL
- VÉRIFICATION DE LA SANTÉ ET AUTO-GUÉRISON
- MISE À L'ÉCHELLE AUTOMATIQUE
- SIMPLIFYING LE DEVELOPMENT D’APPLICATIONS

Section 6 : Conclusion