La révolution des containers

6 Mai 2020

Sommaire

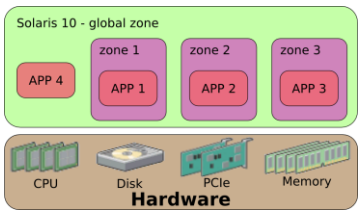
Ce qu'il faut savoir sur les containers

- Les origines
- Containers et virtualisation
- Architectures et applications types : que peut-on héberger
- Urbanisme applicatif
- Les OS dédiés
- Administration, importance de Kubernetes
- Les aspects sécurité
- Les acteurs et sphères d'influence

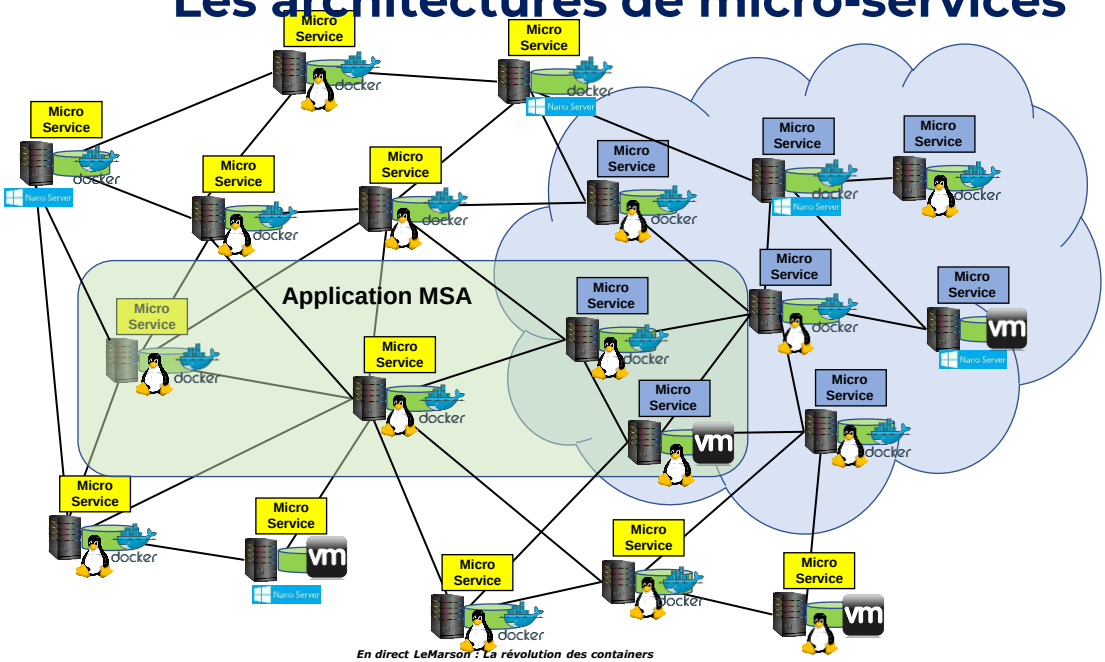
Les origines

- Technologie qui a… 40 ans
  - Commande Unix **chroot** apparue en 1979 : changement de répertoire du système de fichier racine Unix
  - Pour faire des tests, implémenter des process critiques, sans risquer de détériorer le système hôte
  - C'est le début de la séparation des environnements pour chaque process, qu'il faut recréer localement
- Les **prisons** FreeBSD en 2000 : plusieurs environnements administrables indépendamment
- Surtout les "**Solaris Containers**" de Sun en 2004
  - Technologie issue de la virtualisation d'OS des x86 et Sparc
  - Séparation en "zones" : l'OS est virtualisé et isolé des zones
- **Process containers** en 2006 (Google), devenus cgroups
- **LXC** (LinuX Containers) en 2008 : implémentation des cgroups et namespaces de Linux, fonctionne sur un seul noyau Linux
- 2017 : Adoption de Kubernetes par la CNCF (Cloud Native Computing Foundation)

Docker est un projet, lancé à l'origine par Solomon Hykes de dotCloud



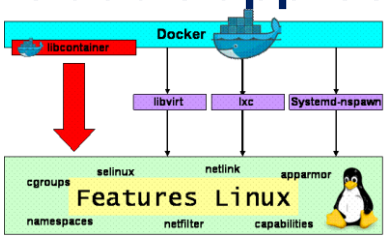
Les architectures de micro-services

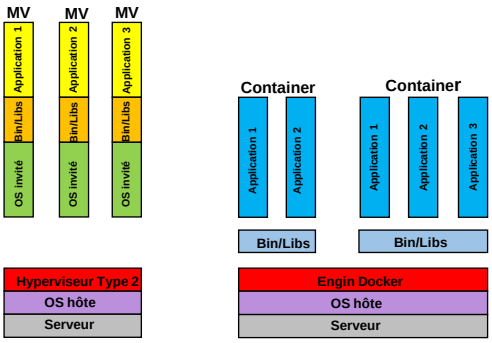


L'arrivée des containers : une autre approche

- Le principe du container est de construire une boîte applicative, dans laquelle il y aura l'application, mais aussi un minimum de « features » système dont elle va avoir besoin pour s'exécuter, sans avoir à transférer le système complet lui-même.
  - Objectif : diffuser les applications par une simple commande, sur d'autres machines, sans être contraints par la lourdeur habituelle de la virtualisation
  - Victoire du « couper/coller » container contre « installer/configurer/exécuter » de la virtualisation
  - L'idée est de construire un repositiory d'images (les build exécutables), que l'on pourra transférer à la demande sur des machines distantes, dans un Cloud, sans avoir à imaginer des procédures de migrations, complexes et coûteuses à gérer.

- Une architecture container est fondée sur trois grandes couches au-dessus du système d'exploitation : l'image applicative, le container lui-même, c'est-à-dire l'ensemble des services rendus aux applications à l'exécution (le « run time ») et la couche d'administration, telle que Kubernetes.

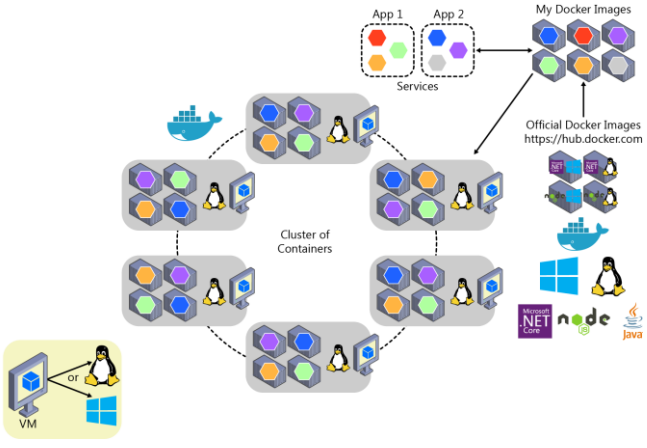




Pourquoi les containers ?

- Plate-forme de diffusion d'applications destinée aux systèmes distribués en clusters
- Environnement d'exécution complet : une application avec ses dépendances, ses bibliothèques et fichiers binaires, ainsi que les fichiers de configuration nécessaires pour l'exécuter, regroupés dans un seul package, mais **PAS** l'OS
- **Un container est une instance fixe d'une image container**

- Le marché spécifique des containers va grandir de 1,2 milliard$ en 2018 à 4,98 milliards$ en 2023 (CAGR de 32.9% pendant cette période (marketsandmarkets).
- Autre source : 8,20 milliards$ en 2025 (CAGR de 31,8 % de 2018 à 2025) (Allied Market)
- Virtualisation d'OS : 62,91 milliards$ en 2023 (Technavio), CAGR de 30%
- La virtualisation des OS serveurs représente un marché plus de 12 fois supérieur à celui des containers
- Ne pas opposer containers et virtualisation : ils se complètent



Images containers… il n'y a pas que Docker

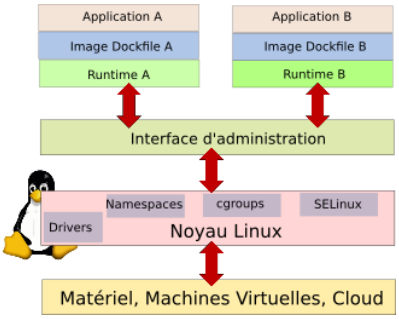
- Mais c'est quand même Docker qui mène les débats
- Rapport Sysdig : en 2018, 83% du marché était Docker (99% en 2017)
- CoreOS rkt (prononcer rocket)
  - Très proche conceptuellement de Kubernetes (les rknetes s'installent facilement)
  - Au départ 2 types d'images : Docker et appc
  - Très compatible, idéal pour la portabilité dans les Clouds publics
  - A l'origine, pas conforme OCI, mais corrigé maintenant (Rklet)
  - Quelques spécificités, dont le support TPM (Trusted Platform Modules)
- Racheté par RedHat
- Mesos Containerizer
  - Apache, en 2018, 4 % des instances
  - Compatible OCI, deux formats d'images Docker et appc
  - Un usage répandu : constructions Big Data avec Spark et Flink
  - Une image Mesos ne peut pas fonctionner seule, elle doit s'appuyer sur le framework Mesos
- LXC Container : version modernisée de LXC (1 % du marché)
  - 3 modules : lxc le runtime, lxd très important, un démon écrit en Go qui gère les containers et images avec nouvelle UI et gestion des containers et lxfuse, la gestion de fichier
  - Difficultés avec Kubernetes et OCI
- Podman, moteur de container conforme OCI (à l'origine projet kpod, à l'intérieur du projet CRI-O), avec Buildah : construction d'images et Skopeo pour la gestion des images et du registry.
- D'autres runtimes : OpenVZ, une extension Linux en 2005 fondée sur la virtualisation, dépassé aujourd'hui, containerid très important dans le contexte OCI, Hyper-V containers, Unikernel…

Architectures et applications types

- Volume : un conteneur peut être très réduit : quelques MB, alors qu'une machine virtuelle avec son système d'exploitation complet peut représenter plusieurs GB.
- Un serveur peut accueillir beaucoup plus de conteneurs que les machines virtuelles.
- Temps de démarrage d'une MV est plus long, alors que les containers démarrent instantanément (ou devraient…).
- On peut organiser les traitements de manière très différente, classique avec les MV et « juste à temps » avec les containers.
- Plutôt que d'exécuter une application complexe entière dans un seul conteneur, l'application est divisée en modules : base de données, interface utilisateur…
- Attention aux applications anciennes qui dépendent d'un kernel dépassé et de systèmes externes spécifiques
- C'est l'approche micro services : chaque module est simple, plus facile à administrer et des modifications peuvent être apportées sans avoir à reconstruire l'application entière.
- C'est le principe de l'urbanisme.

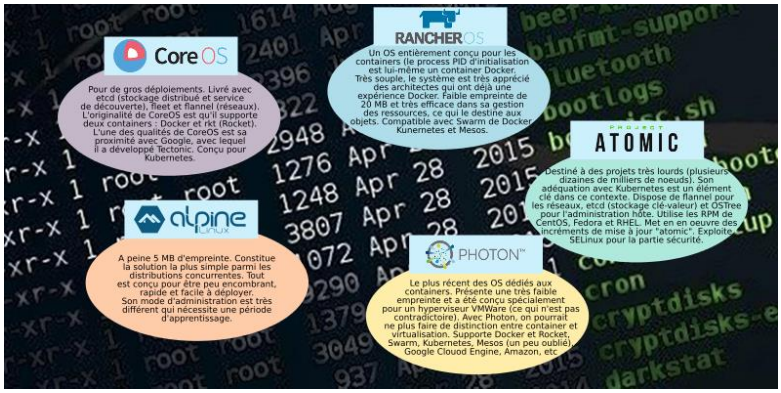
Les OS dédiés

- Les containers n'ont pas besoin de la totalité des services d'un OS classique
- « run time » d'exécution des images
- Accès au « Docker Registry.
- On dispose d'outils pour rechercher, déplacer, charger, sauvegarder et marquer les images Docker.
- On construit des images Docker avec la commande « build docker » et on édite les fichiers Dockerfiles.
- L'OS doit être aussi un complément de l'image Docker.
- Grâce à cette « proximité » entre l'OS et Docker, l'image du container aura accès aux ressources de stockage et de réseaux.
- Le noyau Linux comportera des services indispensables : les « namespaces », cgroups et SELinux (Security-Enhanced Linux).
- Deux choix possibles : on fait tout soi même (côté bidouille évident), soit on passe par un outil plus générique.



L'offre Linux pour containers

- Une dizaine de distributions Linux dédiées au support des containers Docker
- Accompagnées de services complémentaires qui peuvent être très différents.
- Deux grandes familles : celles qui sont conçues essentiellement pour traiter les couches hautes de la pile, telles que CoreOS et RedHat, avec en-dessous des couches propriétaires et celles qui sont plus ouvertes, plus flexibles, susceptibles de mieux s'adapter à des produits de complément tels que Kubernetes : RancherOS et Photon de VMWare.



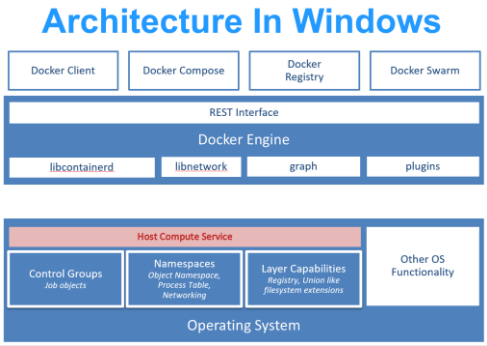
Core OS maintenant Container Linux

Microsoft et les containers

- Microsoft supporte plusieurs images de containers
  - Windows : l'ensemble complet des services système (moins les rôles serveur) et API Windows.
  - Windows Server Core : une image plus petite qui contient un sous-ensemble des API Windows Server : .NET complet, la plupart des rôles de serveur.
  - Nano Server : la plus petite image Windows Server, avec la prise en charge des API .NET Core et de certains rôles de serveur.
  - Windows 10 IoT Standard : version de Windows utilisée par les fabricants de matériel pour les petits appareils IoT qui utilisent des processeurs ARM ou x86/x64.

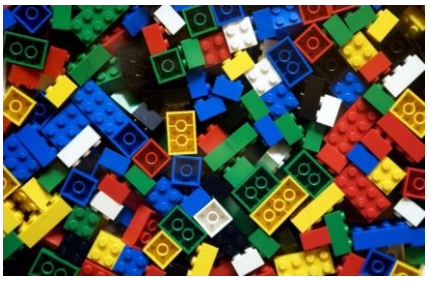
L'écosystème des conteneurs Microsoft, développement et déploiement

- Exécution de conteneurs Windows ou Linux sur Windows 10 avec Docker ou en natif sur Windows Server.
- Développement sous Visual Studio et Visual Studio Codez, prise en charge de Docker, Docker Compose, Kubernetes, Helm, etc.
- Publication d'applications en images conteneur sur l'instance DockerHub publique ou sur une instance Azure Container Registry privée pour le développement et déploiement d'instances privées : envoi (push) et tirage (pull) directement à partir de Visual Studio et de Visual Studio Code.
- Déploiement de conteneurs à grande échelle sur Azure ou d'autres Clouds :
  - Pull de l'image container à partir d'un registre de conteneurs tel qu'Azure Container Registry, déploiement avec un orchestrateur type Azure Kubernetes Service (AKS) ou Azure Fabric Service.
- Déploiement de conteneurs locaux par AKS ou Azure Stack avec OpenShift (RedHat).



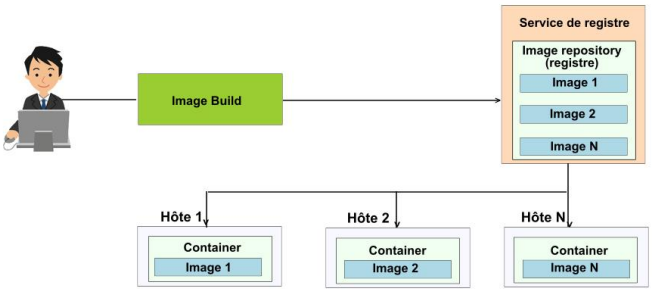
Au-dessus de tout : l'urbanisme applicatif

- A la fois un comportement et des critères techniques
- L'urbanisme applicatif est un comportement de développeur, qui tend à ce que le code qu'il produit soit modulaire, réutilisable par des tiers internes ou externes, compréhensible et donc de qualité et bien documenté, simple et répond aux exigences fonctionnelles.
- L'urbanisme applicatif est aussi un ensemble de méthodes et règles techniques, qui permettent de l'obtenir
- C'est un état d'esprit à inculquer dans l'entreprise
- SOA, MSA, Web Services, Web Components, FaaS, mashups, sont les formes les plus connues de l'urbanisme applicatif
- Quatre natures de composants
  - Présentation : chargés de la gestion de l'interface utilisateur et de la consommation des services distants.
  - Logique de domaine ou métier.
  - Logique d'accès aux bases de données : elle est constituée des composants d'accès aux données chargés d'accéder aux bases de données (SQL ou NoSQL).
  - Logique d'intégration de l'application : elle comprend notamment un canal de messagerie, essentiellement basé sur des répartiteurs de messages.



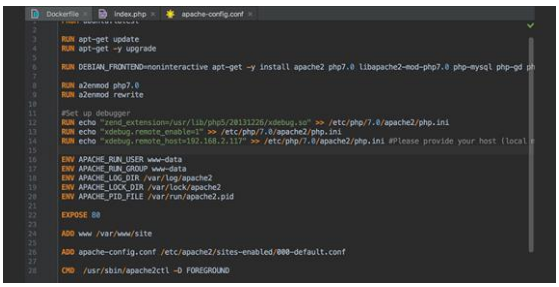
L’urbanisme via Docker

- Docker est fondé lui-aussi sur des bonnes pratiques d’urbanisation :
- Héritage, une image héritant d’une autre grâce à la commande FROM. Il existe un certain nombre d’images de référence issues de Docker Hub, Debian, Ubuntu, Microsoft, etc. Ce sera toujours un bon point de départ.
- Il existe plusieurs "registries" : Docker Hub, Docker Registry de RedHat, OpenShift (RedHat)
- Le process de fabrication d’une image doit être rapide et simple, à partir d’un Dockerfile, la cible de Docker étant les architectures de micro-services, pour lesquels les builds associés à chaque micro-service devront évoluer facilement. On dit que les Dockerfiles doivent être « éphémères ».
- Préférer des images petites.
- S'il y a beaucoup de redondances, les regrouper dans une image partagée.
- Stockage de ces fichiers : le mieux est de les placer chacun dans un répertoire distinct et vide. Grande question des bases de données.
- Pour améliorer la souplesse de fabrication des images, on pourra aussi ajouter un fichier complémentaire, un .dockerignore qui contiendra les répertoires et fichiers qui ne feront pas partie du répertoire de référence du Dockerfile.



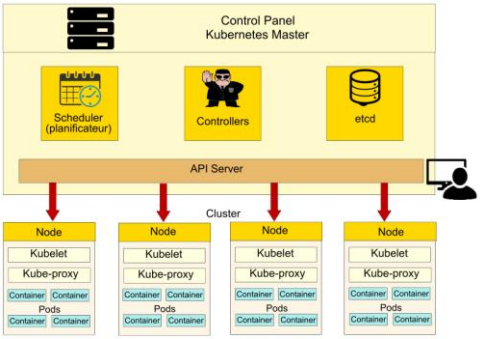
L’urbanisme via Docker

- L’idée reçue selon laquelle il n’y a rien à faire pour préparer une image, est fausse et remet en question la comparaison un peu rapide favorable au couper/coller de Docker par rapport à l’installer/configurer/exécuter de la virtualisation.
- Le secret de l’opération est le fichier Dockerfile, un fichier texte dans lequel Docker va trouver toutes les commandes pour construire son image, celle qui sera installée et exécutée, grâce à la commande docker build : c’est le cœur du système.
- Il s’agit d’un fichier de commandes, construit avec des instructions telles que :
  - RUN pour exécuter n’importe quelle commande Linux
  - ENV pour initialiser les variables d’environnement
  - ADD pour copier le nouveau fichier dans le bon répertoire de destination Docker
  - EXPOSE qui précise le port de la machine hôte.
  - mais aussi FROM, MAINTAINER, CMD, LABEL, EXPOSE, ENV, COPY, ENTRYPOINT, WORKDIR, ONBUILD,
  - VOLUME, USER…
- Il y aura autant de Dockerfile que d’environnements de développement et de langages.
- Il faudra tout prévoir et "tout" installer dans le container : un gestionnaire de servlets tel que Tomcat, etc, les commandes de saisie du mot de passe, la plate-forme MAVEN, le container léger SPRING, des frameworks spécifiques, etc.
- Mais avec des précautions.



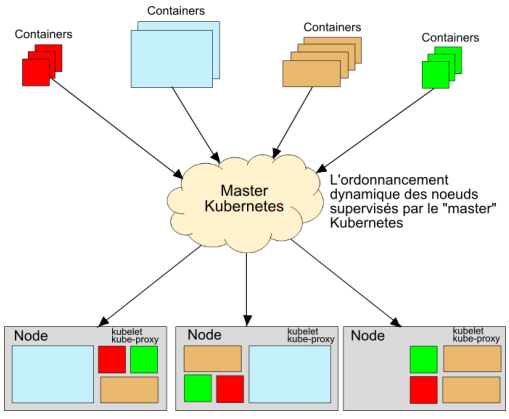
L'importance de Kubernetes pour l'administration

- Le rôle de la couche d'administration est essentielle.
- Docker n'a pas su imposer Swarm
- Kubernetes de Google, est plébiscité
- Google a cédé son produit à la communauté CNCF (Cloud Native Computing Foundation).
- Kubernetes : ensemble de composants (services), qui réalisent des opérations de supervision des containers implantés dans les nœuds d'un cluster, services qui se répartissent en deux familles, serveurs centraux et nœuds du cluster.
- Serveurs centraux : outils pour déployer, maintenir et mettre à l'échelle les applications portées par les containers (adaptation au nombre d'usagers). Ces outils sont extensibles et peuvent être combinés.
- L'entité de base gérée par Kubernetes est le « pod », constituée d'un ou plusieurs containers, qui tous sont situés dans le même nœud du cluster, identifié par son adresse IP.
- L'ensemble des applications d'un pod communiquent par divers moyens, tels qu'un sémaphore ou une mémoire partagée.
- A noter que les outils Kubernetes sont eux-mêmes empaquetés dans des containers, que ce soit sur les serveurs ou les nœuds.



Kubernetes côté serveur

- Kubernetes côté serveur
- Les composants à installer sont ceux du « control panel », qui pilotent l'activité du cluster, gèrent l'ordonnancement des opérations, la réponse aux évènements, le démarrage d'un nouveau pod, etc. C'est le cœur du système, qu'il faut installer, sachant que l'on pourra configurer plusieurs serveurs maîtres, autant pour la sécurité que pour les performances, Kubernetes pouvant être « load-balanced » entre plusieurs serveurs.
  - **kube-controller-manager** : c'est lui qui exécute les contrôleurs, le node-controller, qui maintient l'activité des nœuds et réagit quand l'un d'eux tombe en panne, le replication-controller qui maintient le nombre correct de pods pour chaque contrôleur, le endpoints-controller qui diffuse les objets vers les nœuds (pods…), service account & token controller, etc. C'est la tour de contrôle de l'activité Kubernetes.
  - **kube-scheduler** : ce service surveille les pods nouvellement créés et les affecte aux nœuds, en se basant sur les ressources nécessaires, les contraintes matérielles et logicielles, la localisation des données, etc.
  - **kube-apiserver** : c'est le front-end de Kubernetes, celui qui expose l'API à l'extérieur.
  - **etcd**, la partie stockage, à la fois pour les données manipulées (backup) que pour le maintien des configurations de chacun des containers. C'est une sorte d'annuaire clés-valeurs.



Kubernetes côté client

- Côté nœuds, d'autres services doivent être installés, chargés de la supervision des pods applicatifs, à qui ils vont attribuer les ressources nécessaires dans la plate-forme d'exécution locale (le runtime).
- **kubelet** est l'agent qui s'assure que les containers s'exécutent dans un pod et se fonde pour cela sur des PodSpecs. kubelet ne gère que les containers créés par Kubernetes.
- **kube-proxy** : l'abstraction du réseau.
- **container runtime** : une pièce essentielle responsable de l'exécution des containers, qui supporte plusieurs runtimes et formats d'images, containerid, cri-o (container léger), rklet et n'importe quelle implémentation de Kubernetes CRI (Container Runtime Interface). containerid a été intronisé au début 2019 dans CNCF.
- Kubernetes exploite aussi des addons, des services et pods, qui implémentent indépendamment certaines fonctions.

Les plates-formes de distribution Kubernetes

TECTONIC
CoreOS fournit une distribution Linux compatible Docker, avec son propre runtime qui englobe Kubernetes : CoreOS Tectonic Stack. CoreOS est la propriété de RedHat

Canonical's distribution of Kubernetes
Distribution d'Ubuntu avec Kubernetes. Compatibilité avec n'importe quel Cloud. Supervision assurée (éventuellement) par Canonical.
Version "miniature" avec Microk8s pour les tests.

Pivot Container Service
Pivotal Container Service (PKS) Point fort : sa compatibilité avec VMWre Virtualization Stack. Les containers sous PKS accèdent aux services de vSphere. PKS intègre Kubernetes et peut être géré par VMWare public ou privé.

KONTENA
Distribution Kubernetes certifiée CNCF. Une version basique gratuite compatible avec plusieurs runtime et une version payante avec les services de stockage distribué, backup, load balancing.
N'est pas liée à un produit donné.

RED HAT OPENSHIFT Container Platform
RedHat a adopté Kubernetes pour remplacer les "cartridges" Heroku. CoreOS Tectonic a été fusionné avec RedHat OpenShift.

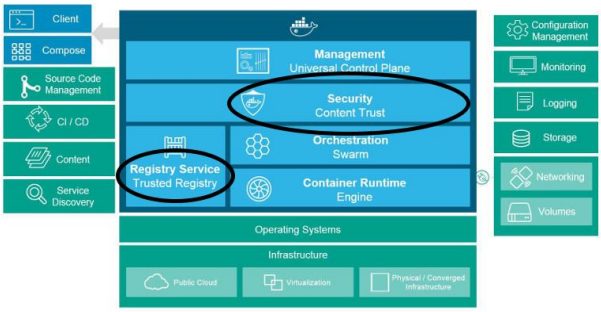
RANCHER
Rancher se place à un niveau de supervision plus élevé que Kubernetes. Peut administrer des clusters Kubernetes sur Amazon EKS, Google, Azure, etc. Comporte sa propre distribution Kubernetes et gomme de nombreux aspects fastidieux des installations Kubernetes.
Rancher propose aussi une version K3s minimaliste qui se contente de 512 Mb par instance serveur et 200 Mb sur disque.

Telekube by Gravitational
Telekube est une dsitribution Kubernetes pour les plates-formes privées SaaS. Les applications doivent être packagées en "bundles", pour être publiées dans des clusters Kubernetes. Gravity est un autre outil de Gravitational pour faire des snapshots complets de clusters Kubernetes et les réinstaller dans un autre environnement compatible Kubernetes.

SUSE CaaS Platform
Suse CaaS combine des fonctionnalités de containers, d'orchestration Kubernetes, de registry et de configuration de clusters. Cette solution remplace EKS d'Amazon et Google Kubernetes Engine.

La sécurité contestée de Docker

- Les reproches que l'on peut faire à Docker, sont liés à ses qualités.
- Le kernel est partagé entre les différentes images Docker de la machine « porteuse » : il y a un risque accru s'il est attaqué.
- Il se peut qu'un container cannibalise toutes les ressources à son profit.
  - Cet inconvénient a été corrigé depuis 2014 avec le paramètre ulimit qui permet de restreindre les ressources consommées :
  - $ docker run -i -t --rm myapp:2.0 --ulimit nofile=128:256 --ulimit nproc=32:64
  - Depuis la version 1.6 de Docker (ancienne), on peut aussi définir des valeurs ulimit par défaut pour tous les containers, en spécifiant une option –default ulimit dans le fichier de configuration Docker :
  - OPTIONS="--ulimit nofile=1280 :2560 –ulimit nproc=256 :512--"
- L'engin Docker a été mis à jour de manière à gérer des dispositions Linux de bas niveau, comme Seccomp pour filtrer les appels systèmes d'un processus Linux au niveau du noyau (existe depuis 2005) et user namespace, qui permet de créer des groupes de processus de base sous Linux.
- Jusqu'à la version 1.10, Docker comportait les namespaces PID, UTS, IPC et Network, mais il lui manquait le groupe root, de sorte qu'un utilisateur root dans un container aurait pu sortir de son image et conserver ses privilèges root sur la machine hôte. Avec la création du groupe root ce n'est théoriquement plus possible. Les namespaces Docker constituent une excellent isolation entre images Docker et machine hôte.

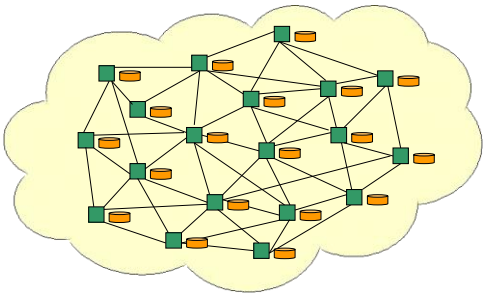


Les bonnes pratiques sécuritaires de Docker

- Créer une partition Linux séparée pour chaque nœud Docker. Car Docker place toutes ses images sous le répertoire racine dans /var/lib/docker, qu'il peut facilement saturer et par transitivité saturer le répertoire racine lui-même.
- Eviter les communications entre containers de manière à ce qu'une image « mal intentionnée » n'aille regarder ce qui se passe « ailleurs ». Il ne faut autoriser que les containers liés par la fonction link. Il suffit d'éditer le fichier /etc/default/docker et de modifier la variable DOCKER_OPTS
- Ne pas abuser du mode « privileged », très puissant mais qui permet aussi de lancer de nouvelles images sur la machine hôte…et donc de la saturer.
- Ne faire confiance qu'aux images validées par Docker et si nécessaire aller vérifier à la source, chez GitHub, que l'image que l'on installe est bien celle qui est recensée
- Pour ce qui est des ports à ouvrir sur la machine hôte, ne pas fonctionner par défaut et préférer une déclaration port par port. C'est plus long, mais plus sécurisant

Une autre manière d'imaginer une application

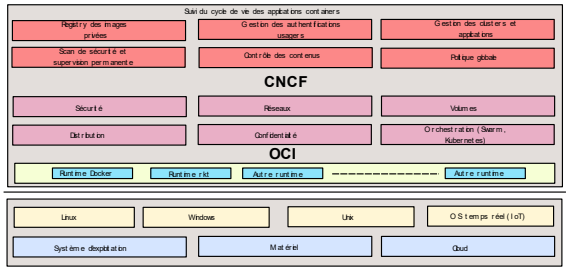
- Construire un repository de services
- Modéliser la cinématique de l'application à l'aide d'un langage formel : BPEL (XML), utilisé pour les Web Services ou un autre
- Tenir compte du fait que les services vont être orchestrés, mais qu'ils ne sont pas nécessairement prêts à l'exécution (il faut les acheminer sur un serveur disponible)
- Il faut prévoir :
  - La découverte des micro-services, via le repository (service normalement natif)
  - Le traitement de l'asynchronisme
  - La gestion des exceptions, d'autant plus complexe que l'on est dans un mode distribué
  - La gestion des files d'attentes aux micro-services, selon les modes LIFO (Last In First Out) ou FIFO (First In First Out)
  - Le traitement des formats de messages : XML, JSON, texte, CSV…
  - La prise en charge du chiffrement (confidentialité) et identité des micro-services
  - La gestion de la dégradation des applications, chaque micro-service pouvant envoyer un état de comportement (heartbeating) au repository pour l'informer de sa situation
  - La répartition de charge, qui est une conséquence du point précédent, l'idéal étant de disposer d'une API dédiée qui règlera ce problème
  - La tolérance aux fautes, indispensable pour certains micro-services, pour qui on prévoira des procédures de repli prioritaires
  - L'enregistrement des évènements de l'application dans un fichier log



Les sphères d'influence des containers

CNCF, l'Open Source pour le Cloud

- CNCF (Cloud Native Computing Foundation) créé en 2015 par 18 compagnies pour promouvoir les projets Open Source destinés au Cloud. Les deux premiers ayant été Kubernetes de Google et Promotheus, pour le « monitoring ».
- Les deux plus grands acteurs du Cloud, AWS et Microsoft ont rejoint la communauté CNCF en tant que membres « platinium » et CNCF regroupe maintenant tous les acteurs qui comptent dans le monde du Cloud.
- CNCF tourne toujours autour de Kubernetes, qui constitue son projet phare, d'orchestration des containers Docker.
- Mais d'autres projets significatifs sont venus se joindre à la communauté.
- De plus, CNCF est considéré comme le bras armé de Google dans le monde des containers et la reconnaissance d'IBM, Microsoft , AWS est avant tout celle de Kubernetes, pas du reste…



Les sphères d'influence des containers

OCI pour les containers

- OCI (Open Container Initiative) est l'autre structure phare. Fondée également en 2015, pour élaborer des spécifications autour des containers. Elle doit beaucoup à Docker, qui en a été à l'origine.
- OCI a succédé à l' « Open Container Project », lui-même lancé sous l'impulsion de Docker.
- Le forum OCI propose une norme, la 1.0, avec deux composants clés : les formats d'images et les spécifications du « runtime », pour l'exécution. C'est un début, sachant qu'il manque encore beaucoup d'éléments, dont certains essentiels comme la distribution des containers ou la méthode pour solliciter une image, hébergée dans un annuaire.
- L'un des points forts d'OCI, est le consensus qui se dessine autour de son nom, la plupart des acteurs qui comptent étant représentés : Docker, Dell, Cloud Foundry, Core OS, Google, IBM, Huawei, Intel, Mesosphere, Microsoft, Oracle, Red Hat, VM Ware, etc. Souvent des concurrents de Docker.
- Le forum OCI souhaite créer un cadre dans lequel pourront s'insérer différentes formes de containers, qu'elles viennent de Docker ou d'un autre horizon.

OPEN CONTAINER INITIATIVE

CLOUD NATIVE COMPUTING FOUNDATION