TechTrends #12

Les Conteneurs, facilitateurs de livraison jusqu'en production

Projeter, Implémenter, Exploiter

BY ENGINEERS *at* PUBLICIS SAPIENT

publicis sapient

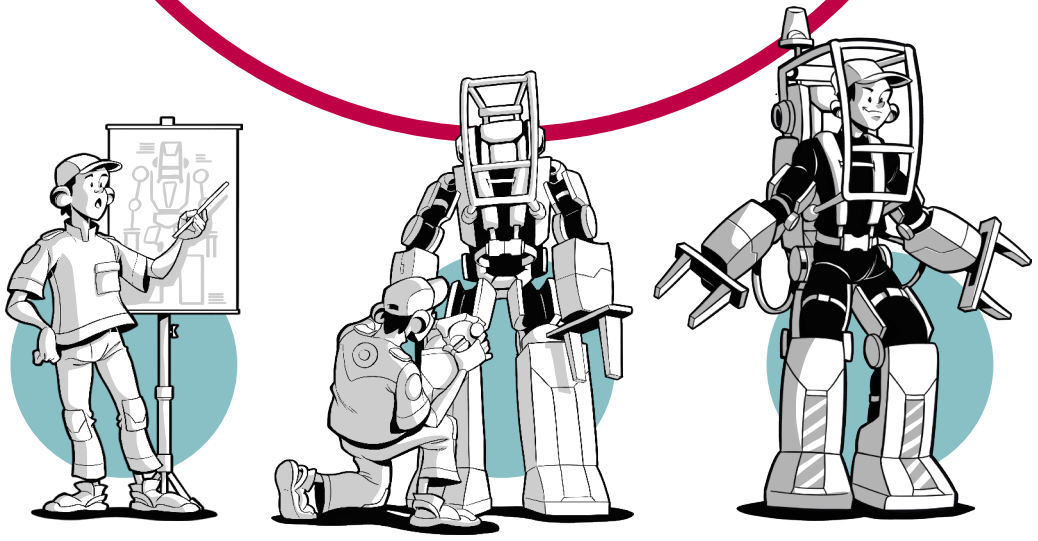
Les TechTrends sont l'expression de notre savoir-faire ; forgé sur le terrain, auprès de nos clients dans le cadre des projets que nous menons avec eux.

Fruit d'un travail collaboratif de nos consultants, vous y trouverez, nous l'espérons, les nouvelles tendances technologiques et méthodologiques ainsi que l'état de l'art de notre profession.

Nous tentons, dans le cadre de ces publications, de vous dispenser des conseils directement opérationnels afin de vous guider dans les décisions stratégiques que vous avez à prendre.

Distribués à plusieurs milliers d'exemplaires tous les ans, la collection des TechTrends s'étoffe régulièrement de nouveaux ouvrages.

Bonne lecture !



Projeter Implémenter Exploiter

Introduction

Classiquement, les développements logiciels dans les entreprises s'organisent en suivant trois grandes phases : **la construction, le déploiement et la production.** Quels que soient le langage ou la technologie utilisés, la phase de construction est bien souvent mature en termes d'outillage et de standardisation. Pour le déploiement, il existe différents **outils d'automatisation** permettant d'utiliser des solutions adaptées à chaque technologie. Si l'on s'intéresse au run, il existe aussi des solutions automatisées. Cependant, dans l'ensemble, ces outils manquent de standardisation et nécessitent encore aujourd'hui un grand nombre de tâches manuelles lors des opérations de déploiement et de run.

Les conteneurs permettent de standardiser ces différentes phases. Ils fournissent un **modèle de packaging, de déploiement et d'exécution.** Ce modèle est reproductible et portable. Il s'applique tout aussi bien aux solutions techniques sous Linux qu'à celles sous Windows.

L'utilisation de ce nouveau modèle permet de **fiabiliser l'ensemble de la chaîne de delivery** en définissant un standard commun du développement à la production tout en garantissant une portabilité optimale. Par nature, les conteneurs favorisent **l'adoption d'organisation DevOps**, car ils concernent l'ensemble de la chaîne de delivery.

Le succès des conteneurs est lié à trois facteurs : la maturité, la simplicité et le packaging. En effet, les solutions d'exécution conteneurisées ne sont pas nées de la dernière pluie. Les namespaces permettent d'isoler un ensemble de processus et de ressources sous Linux depuis 2002. C'est en 2008 que les choses s'accélèrent avec la création des cgroups, qui permettent de maîtriser les ressources allouées aux processus. Ce moment correspond à la naissance des premiers PaaS dont AppEngine, DotCloud et Mesos qui reposent tous dès le départ sur la conteneurisation. Mais c'est aussi l'année de naissance du projet LXC ou Linux Containers, qui fournit un ensemble d'outils et d'APIs facilitant la création et la gestion des conteneurs d'application ou de système.

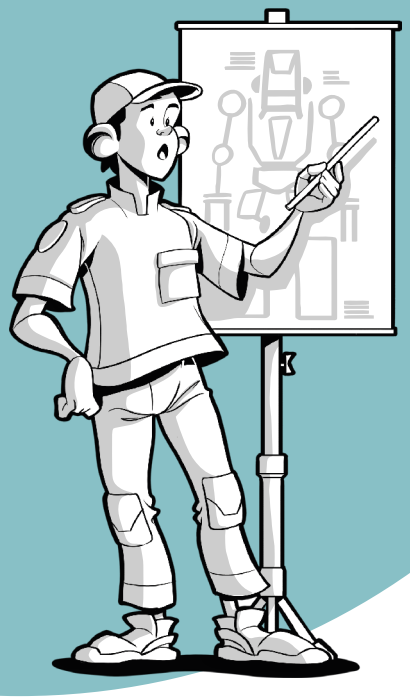
Enfin, en 2013, l'émergence de **Docker** popularise leur utilisation. Avec ce nouvel outil, la technologie est devenue abordable et facile à utiliser. Docker fournit, de plus, un modèle de packaging accompagné d'un système de distribution et de déploiement simple, rapide et efficace.

Aujourd'hui, les images, tout comme les "wars" dans l'écosystème Java, offrent la garantie d'un package portable pouvant être exécuté sur tous les systèmes supportant les cgroups et les namespaces. Les conteneurs offrent donc l'opportunité **d'innover rapidement** avec une **réversibilité à moindre effort**.

Comment pouvez-vous tirer parti de ce nouveau paradigme de packaging et de déploiement ?

- Dans quels contextes leur utilisation est-elle adaptée ?
- Quels sont les défis techniques à adresser ?
- Comment tirer le meilleur de la chaîne de delivery avec ces solutions ?

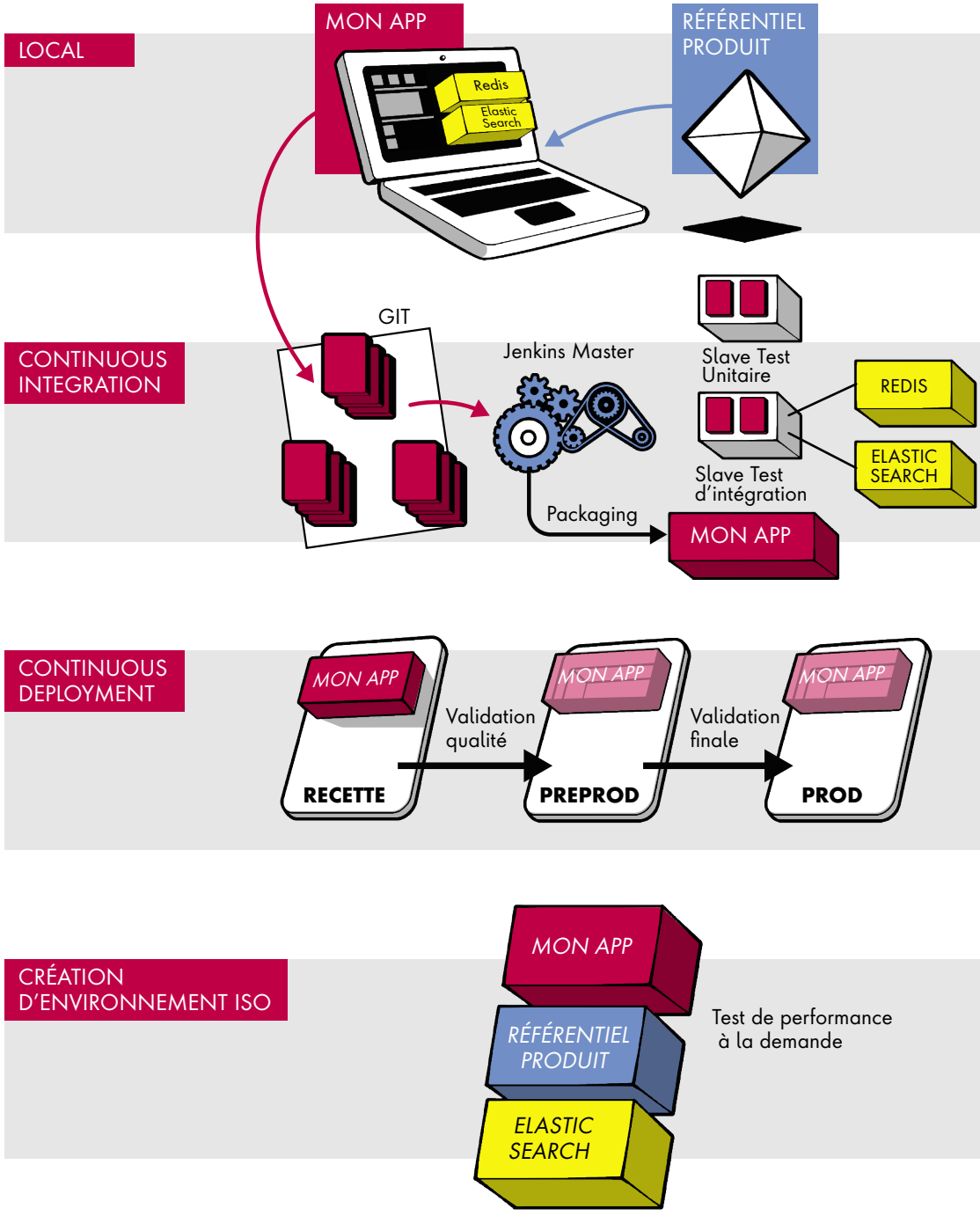
> " Les conteneurs favorisent l'adoption d'organisation DevOps, car ils concernent l'ensemble de la chaîne de delivery. "



Projeter

L'adoption d'une technologie aussi structurante que la conteneurisation n'est pas une décision à prendre à la légère.

En effet, les impacts techniques, tout autant qu'organisationnels, que nous détaillerons par la suite, vont peser sur votre SI. Alors, avant d'opérer ces changements en profondeur, mieux vaut avoir une bonne raison de le faire.
Nous allons donc explorer, dans cette première partie, quelques use cases standards qui peuvent vous inciter à prendre ce virage.



Vue d'ensemble des cas d'application d'une plateforme de conteneurs

Un environnement de développement local complexe

Les plateformes modernes sont souvent composées de très **nombreux middlewares** : serveur applicatif, messaging, stockage, indexation, cache, etc.

Face à cette profusion, le développeur a deux choix :

- Soit **installer un environnement complet en local sur sa machine**, et cet exercice, fastidieux et chronophage, relève du mythe de Sisyphe. Il est illusoire d'espérer que chaque équipier maintienne dans le temps sur son poste une plateforme de développement à jour et représentative de l'environnement cible ;
- Soit **dépendre d'un environnement de développement partagé**, sur lequel l'ensemble des développeurs de l'équipe teste leurs réalisations. Ces environnements, par nature mutualisés, deviennent rapidement des goulets d'étranglement. De plus, l'administration de ces plateformes est bien souvent diffuse entre une équipe d'opération transverse et l'équipe de développement elle-même. La première équipe intervient sans obligation de moyen ni de résultat, sa priorité étant la production, tandis que la seconde équipe s'y retrouve hors de son domaine de compétence.

L'utilisation des conteneurs permet de **créer un environnement isolé proche de la production**, directement en local sur le poste du développeur, via une simple ligne de commande. La création des images reste de la responsabilité de spécialistes des middlewares. Les développeurs peuvent ainsi se concentrer sur leur tâche première : la réalisation de user stories.

> " L'utilisation des conteneurs permet de créer un environnement isolé proche de la production, directement en local sur le poste du développeur, via une simple ligne de commande. "

Intégration continue (Continuous Integration)

Les **usines logicielles** sont un élément clé des chaînes de production logicielle sur lesquelles les problèmes de contention impactent directement et fortement la productivité des équipes de réalisation.

Historiquement, les moteurs d'intégration continue (Jenkins, GitlabCI et GoCD) ont répondu aux problèmes de contention par une distribution de charge horizontale en lançant les tâches des pipelines d'intégration sur des fermes de machines esclaves. Plus les esclaves sont nombreux, moins l'engorgement est fort.

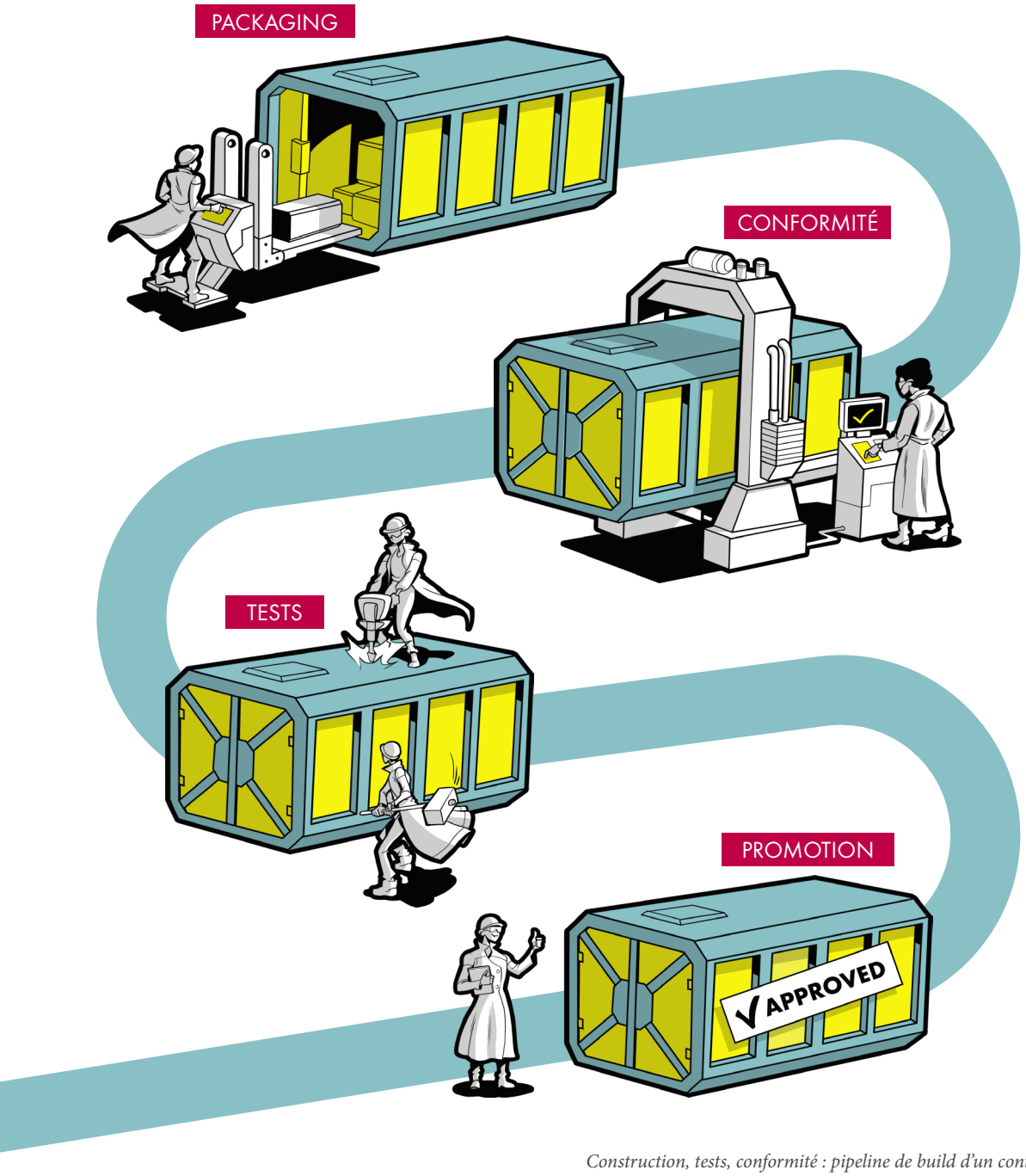
Cette promesse de la **scalabilité horizontale reste difficile à tenir**, l'élasticité des datacenters hors cloud n'étant par nature, pas infinie.

Les conteneurs apportent une solution, que les éditeurs ont immédiatement adoubée : les jobs de build sont maintenant exécutés dans des environnements isolés, en utilisant des images pré-provisionnées. Cette construction préalable **diminue d'autant le temps de build** en évitant de rapatrier l'ensemble des dépendances à chaque exécution. Les **machines esclaves sont remplacées par des conteneurs** qui peuvent s'exécuter au sein de fermes de serveurs.

L'utilisation des conteneurs pour **exécuter les tâches d'intégration continue** apporte deux autres avantages non négligeables :

- Cela permet de **garder les environnements hôtes vierges** de toute pollution venant des outils de compilation et de packaging ;
- Cela **facilite les montées de versions des dépendances et autres middlewares**. Il suffit d'utiliser une image de build mise à jour pour intégrer les nouvelles versions.

Déploiement continu (Continuous Delivery)



Construction, tests, conformité : pipeline de build d'un conteneur

Les entreprises l'ont bien compris, le véritable enjeu n'est pas de livrer les applications en recette via la plateforme d'intégration continue.
La finalité doit être de **fournir la valeur produite aux utilisateurs finaux en continu**.

Pour adresser ce nouveau défi, la norme est de construire des **pipelines de livraison** exécutés par les serveurs de CI/CD. À chaque construction d'un package, le pipeline se lance pour déployer la nouvelle version d'environnement en environnement jusqu'à la production.

Vous l'aurez bien compris, **le package est la pierre angulaire** de toute chaîne de déploiement continu. Pour garantir une portabilité maximale, il doit respecter deux principes de base : **la complétude et l'immutabilité.**

Des packages complets

Le package est ensembliste, il doit fournir l'intégralité des éléments nécessaires à son installation et son exécution. On y trouvera le programme en lui-même, le code descriptif de son infrastructure et les différents éléments de configuration.

Traditionnellement, le tout est réuni au sein d'une archive compressée accompagnée d'une note de livraison faisant office de documentation.

Ce format de packaging, bien que fonctionnel, semble désuet. La virtualisation a apporté en son temps une nouvelle solution reposant sur des VM pré-provisionnées. À l'usage, les VM se révèlent être des packages lourds, et peu portables du fait de la dépendance à l'hyperviseur.

Avec les images de conteneurs, nous avons une solution qui **inclut nativement l'application, l'infrastructure et les variables de configuration**. Outre ces avantages, les images sont particulièrement portables, de par leur adhérence faible à l'infrastructure.

Des packages immutables

Chaque construction est à l'origine d'un paquet qui doit avoir le potentiel de finir sa course en production. Le paquet **ne doit subir aucune modification** entre sa création et son déploiement final en production. La fiabilité de l'ensemble du processus de la chaîne de livraison repose sur l'idempotence du déploiement.

Pour être plus clair, un même package doit se comporter de la même façon, qu'il soit utilisé en développement, en intégration ou en production.

Les images de conteneur répondent exactement à cette caractéristique. En effet, une image est, comme son nom l'indique, figée. Elle ne contient qu'un système de fichiers en lecture seule et quelques métadonnées. Outre l'aspect immuable des images, celles-ci correspondent à un hash qui peut être associé à un tag de version bien utile lorsqu'il s'agit d'identifier ou de tracer son utilisation.

Attention toutefois, ces propriétés ne viennent pas résoudre l'éternel **problème de la gestion des configurations**. Le point crucial est d'externaliser les variables de configuration pour qu'elles puissent s'adapter à l'environnement dans lequel le conteneur sera exécuté. Par défaut, la solution consiste à reposer sur des variables d'environnement propagées à l'exécution du conteneur. Mais vous pouvez tout aussi bien vous tourner vers des volumes contenant les fichiers de configuration ou vers des solutions de service discovery.

Nous ne sommes plus qu'à une marche d'un déploiement entièrement automatisé.
Le package de livraison est portable, exécutable et configurable de l'extérieur. Il ne reste plus qu'à gérer des problématiques de plus haut niveau, comme les interactions entre les différents composants du système, le réseau, la persistance et la topologie de déploiement.

Les conteneurs appartiennent à un écosystème complet, qui, comme nous le verrons, permet de gérer efficacement ces problématiques.

Pour finir, les conteneurs offrent par leur nature portable et éphémère **l'opportunité de créer les environnements à la demande sur une infrastructure** prévue à cet effet. Nous pouvons alors faire abstraction des machines physiques sous-jacentes et les considérer comme une somme de ressources disponibles pour l'exécution de nos conteneurs.

La possibilité de sortir d'un carcan mono-langage

Aujourd'hui, le SI moderne est de plus en plus difficile à standardiser.
Là où, par le passé, il était envisageable d'aligner tous les développements sur une pile Java/Oracle, parfois agrémentée de frameworks maison destinés à cadrer (voire brider) les développeurs, les grands du Web ont démontré que la diversité est source de valeur. Ainsi, il n'est pas rare de rencontrer des SI polyglottes, en termes de langages de programmation, mais aussi de bases de données, de stockage, d'OS, etc. Certains réinventent même leur SI sous forme de **microservices**.[1]

Choisir la technologie répondant le mieux à un besoin est un enjeu majeur. Si la conteneurisation n'est pas la baguette magique qui vous permettra de réaliser cet exploit, elle peut grandement le faciliter.

Les conteneurs vont permettre en quelque sorte de rationaliser le SI, **en réfléchissant en matière de blocs fonctionnels**. Chacun de ces blocs possède une construction similaire (via un fichier descripteur), est déployé sur des machines identiques (via les outils d'orchestration), et possède des liens communs à l'ensemble (le réseau, le stockage, le monitoring).

Cela permet d'avoir une équipe qui se concentre sur le contenu du conteneur (Dev) et une équipe qui se concentre sur l'environnement (Ops). Dès lors, pourquoi ne pas laisser aux équipes de Dev la possibilité de choisir l'outillage (langage, frameworks, etc.) leur permettant d'atteindre au mieux leur but ?

Cela apporte aussi le bénéfice pour l'entreprise de disposer d'un plus grand vivier de talents, en piochant des compétences dans des écosystèmes différents.

> " Les conteneurs vont permettre en quelque sorte de rationaliser le SI, en réfléchissant en terme de blocs fonctionnels. "

1- Le TechTrends Microservices est disponible en téléchargement sur publicissapient.fr/services/engineering

Take away

Projeter

La conteneurisation propose des bénéfices immédiats dans l'ensemble du SI :

Sur le poste du développeur

sur le poste du développeur, où elle permet de rapidement installer l'ensemble des composants logiciels d'une application ;

Dans l'usine logicielle

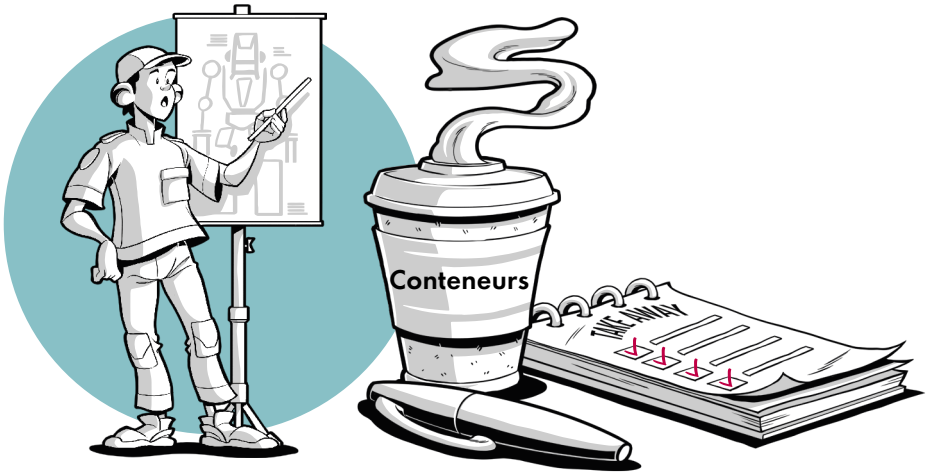
dans l'usine logicielle, où elle permet d'atteindre une élasticité horizontale à moindre coût ;

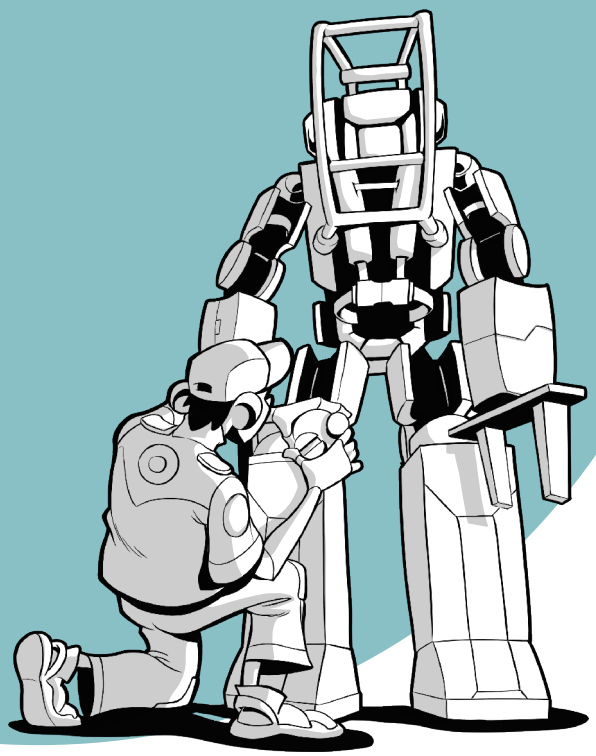
En production

en production, où elle facilite le déploiement continu ou la construction d'environnement "à la demande" ;

À l'échelle de l'entreprise

à l'échelle de l'entreprise, où elle facilite l'intégration de stacks techniques variées.





Implémenter

Votre use case cible est identifié, il est maintenant temps de vous lancer dans l'implémentation de votre plateforme.

La première étape est de choisir l'ensemble des briques qui la composera, et de s'y retrouver dans une offre devenue pléthorique. Pour y voir plus clair, nous vous proposons de décortiquer ce qu'est réellement **une plateforme de conteneurs**. En effet, la place prépondérante de Docker dans la communauté amène souvent l'amalgame : "conteneur égal Docker". C'est loin d'être le cas puisque se cache, au sein même de l'entreprise Docker, une multitude de produits.

Tout d'abord, pour s'exécuter, vos conteneurs auront **besoin d'un moteur**, qui leur permettra d'exploiter les ressources de la machine hôte. Ensuite, il est rare de voir des logiciels composés d'un module unique. Vous aurez donc **besoin d'un orchestrateur**, pour décrire et réguler le comportement d'un ensemble de conteneurs. Enfin, les conteneurs vont utiliser des ressources spécifiques de la machine hôte, à savoir **le réseau et le disque**.

Le moteur d'exécution, le minimum vital

Un conteneur **a pour mission d'isoler le fonctionnement d'une application**, constituée d'un ensemble de processus, des autres applications s'exécutant sur le même hôte système. Les critères définissant un conteneur sont simples :

- Les **processus d'un conteneur sont isolés** et n'ont pas connaissance des autres processus de la machine hôte.
- Les **fichiers présents dans un conteneur lui sont propres**, un conteneur n'a pas accès par défaut aux fichiers du système hôte. Tout partage de fichiers doit être explicite.
- Un **conteneur repose sur une image**. Une application s'exécutant dans un conteneur peut en modifier le contenu mais l'instanciation d'un nouveau conteneur démarrera toujours à partir de l'image d'origine.
- Un **conteneur n'exécute pas nécessairement un système d'exploitation complet** mais uniquement les processus strictement requis.

Le moteur d'exécution se charge de garantir ces conditions minimales. Pour ce faire, il repose sur certains prérequis techniques contenus nativement dans les systèmes d'exploitation. C'est le cas depuis de nombreuses années pour Linux (grâce à l'utilisation des cgroups et des namespaces), BSD (Jails) ou encore Solaris (Zones). Plus récemment, le partenariat entre Microsoft et Docker a permis la création des APIs nécessaires au sein de Windows Server.

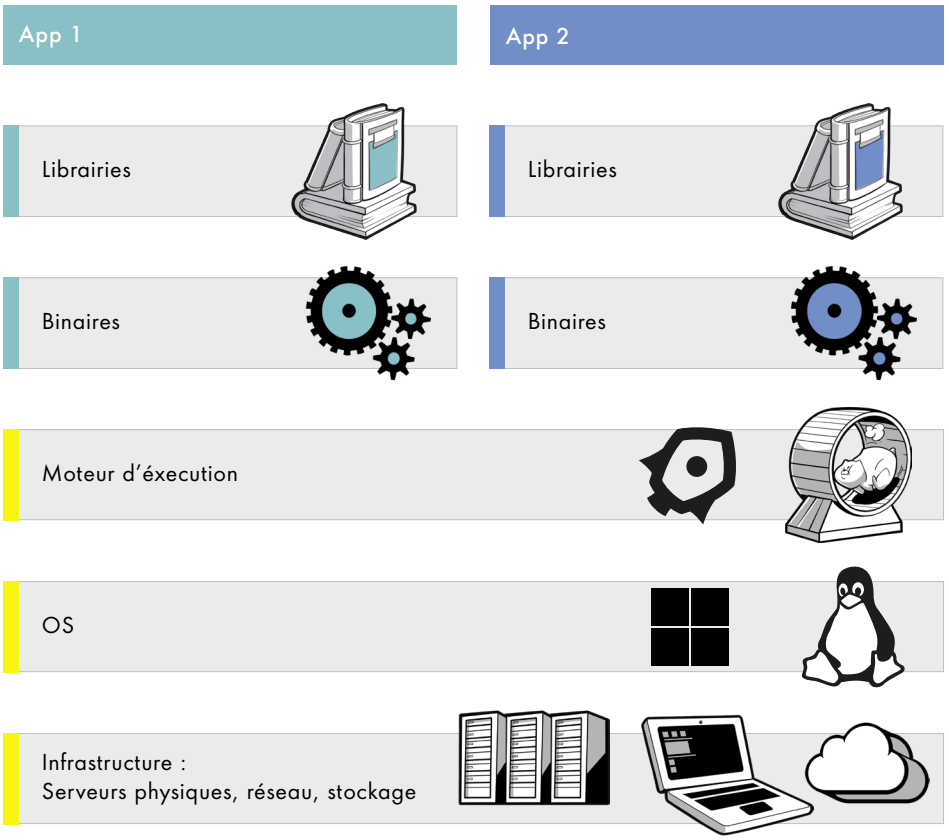
Ces briques de bas niveau sont relativement difficiles à exploiter, et la montée en puissance des conteneurs n'a pu se faire que grâce à l'apparition d'outils plus accessibles. LXC a ouvert la voie dès 2008, en initiant **la création d'outils et d'APIs visant à simplifier l'utilisation des conteneurs** sous Linux. Depuis, de nouveaux moteurs comme runC (soutenu par Docker) ou Rkt (proposé par CoreOS) sont apparus sur le marché.

Le rôle de **l'Open Container Initiative**[2], créée sous l'égide de la fondation Linux, est de contenir la folie créatrice des ingénieurs, et de favoriser la compatibilité entre les différentes solutions de conteneurs. À cette fin, sa mission est de proposer des spécifications de référence.

À ce jour, deux composants possèdent une norme : le format des images et les APIs du runtime, dont RunC est l'implémentation de référence.

Le choix du moteur, dès lors qu'il est supporté par l'OCI, ne devrait donc pas être un frein au développement de votre plateforme.

2- https://www.opencontainers.org



Une pile d'exécution riche

Un orchestrateur, pour les gouverner tous

Les architectures applicatives modernes prônent le **découpage des applications en modules de petite taille**, répondant à un besoin fonctionnel ou technique bien identifié. Ces modules doivent être dans la mesure du possible indépendants du système qui les exécute, pour ne conserver que des dépendances logiques entre eux. La notion à la mode est bien sûr celle d'architecture microservices.

Les conteneurs possèdent une caractéristique intrinsèque intéressante pour atteindre ce découpage : leur isolation. Dès lors, il est nécessaire de **résoudre le défi de la dépendance logique. C'est là qu'interviennent les orchestrateurs.**

Il est rare de voir un SI complet s'exécuter sur une seule machine, aussi puissante soit-elle.

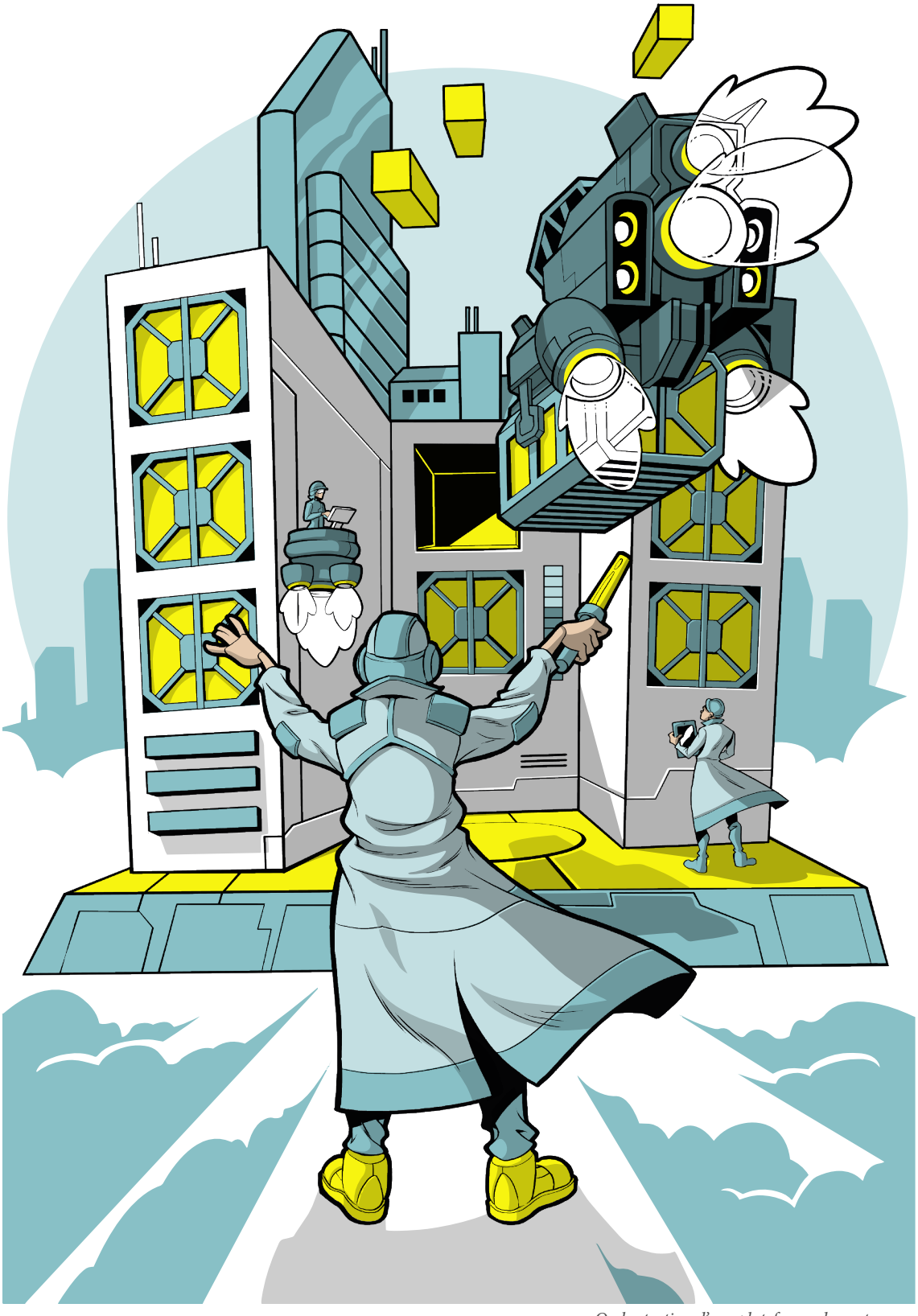
La première mission de l'orchestrateur sera donc de **placer les conteneurs sur un ensemble de machines**. Le moteur d'exécution, ainsi qu'un agent de l'orchestrateur, sont présents sur chacun des serveurs hôtes. Les ressources de ces machines (RAM, CPU, voire GPU et disques SSD) sont mises à la disposition de l'orchestrateur, dont le but premier est d'optimiser le placement des conteneurs, en tenant compte d'un certain nombre de contraintes statiques (quantité de RAM allouée, nécessité d'utiliser le GPU, etc.).

Il doit également **prendre en compte les contraintes dynamiques**, dont la résolution évolue dans le temps. On citera en exemple les contraintes d'affinité, qui imposent à deux conteneurs d'être exécutés sur le même serveur, ou d'anti-affinité qui contraignent deux instances du même service à être réparties sur des serveurs différents pour garantir la haute disponibilité.

L'orchestrateur assure aussi un **rôle de surveillant général**, vis-à-vis de l'état des conteneurs.

À partir d'un document descriptif (nombre de conteneurs souhaité, affinité ou anti-affinité, etc.) et à l'aide de Health Check périodiques, l'orchestrateur **maintient l'ensemble des conteneurs dans un état de fonctionnement optimal**. On peut ainsi facilement mettre en place des mécanismes de haut niveau, comme :

- la haute disponibilité : garantir l'exécution d'un service même en cas de défaillance d'un conteneur, voire d'un serveur ;
- le scale-in/scale-out, afin d'absorber des pics de charge à moindre coût ;
- l'optimisation de la latence en distribuant une application sur des noeuds géographiquement éloignés.



Orchestration d'une plateforme de conteneurs

Il existe aujourd'hui plusieurs solutions d'orchestration :

- Kubernetes
- Mesos/Marathon
- Swarm

Ces dernières assurent toutes les rôles définis ci-dessus. Décrire leurs différences nous obligerait à rentrer dans un niveau de détail technique hors de propos dans le cadre de ce TechTrends. Nous nous contenterons donc d'évoquer les caractéristiques externes des différents projets.

Kubernetes

Kubernetes (souvent abrégé k8s) est un produit open source développé initialement par Google, et reversé à la communauté. Le géant de l'Internet, à l'origine du développement des cgroups, utilise les conteneurs depuis près de 15 ans. Le **partage de son orchestrateur avec la communauté** constitue donc un gage de qualité.

Kubernetes possède deux particularités fortes :

- **la gestion des conteneurs est confiée aux moteurs de conteneurs**, que ce soit RunC ou Rkt ;
- Kubernetes possède **un certain nombre d'APIs** qui va lui permettre facilement de s'interconnecter aux clouds publics ou privés afin de gérer la scalabilité horizontale (ajout de noeuds au cluster).

Mesos

Mesos est la solution la plus ancienne parmi les trois citées. Sa genèse remonte à une époque où la popularité des conteneurs était nettement moindre. Le concept de Mesos est de **regrouper un ensemble de ressources (RAM, CPU)** pour proposer une vue agrégée, permettant d'exécuter des programmes (conteneurisés ou pas).

Mesos n'a pas vocation à être utilisé seul. Il **délègue l'exécution des programmes à ses frameworks**. Parmi eux, les plus utilisés sont certainement Marathon et Aurora. Ce sont eux qui, sur la base d'un fichier JSON, sont responsables de placer les conteneurs et de garantir leur exécution dans les conditions souhaitées.

Swarm

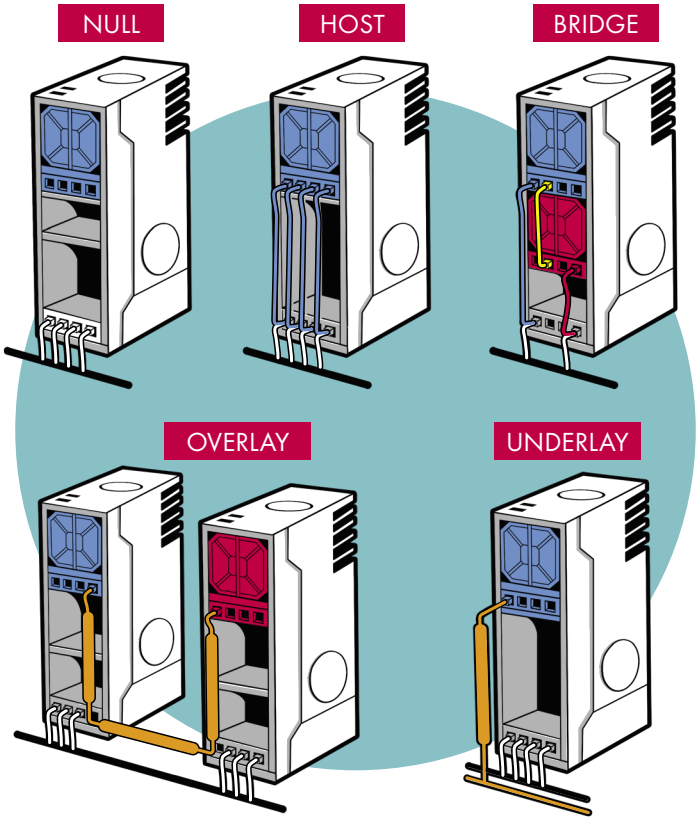
Swarm est **la solution d'orchestration fournie par Docker**. C'est également la plus jeune des trois solutions.

Il est difficile de prédire ce que sera Swarm dans un an tant son évolution a été rapide dernièrement. Au départ un composant à part, Swarm est désormais complètement intégré au service Docker. On ne parle plus de Swarm en tant que tel mais du Docker "swarm mode".

C'est donc la solution la plus "transparente" vis-à-vis de l'utilisation de conteneurs, mais aussi la moins riche (pour l'instant) et la moins modulaire.

Le réseau, fil de communication

Nous l'avons vu, les applications modernes sont souvent composées de plusieurs conteneurs communiquant entre eux. Si l'exercice est relativement aisé au sein d'une même machine, il devient nettement plus périlleux lorsque les conteneurs sont déployés sur plusieurs noeuds, comme c'est la norme avec les composants d'orchestration.



Les différents modes de communication réseau

Nous allons détailler les différents modes de communication disponibles et leurs usages bien spécifiques :

None/Null

Les conteneurs lancés dans ce mode n'ont tout simplement **aucune possibilité de communiquer avec l'extérieur.** Ils s'exécutent sans réseaux et sont, de ce fait, complètement isolés du reste du monde. Vous utiliserez donc ce mode pour exécuter des applications n'ayant pas besoin d'accéder à des services extérieurs comme des programmes de calcul par exemple.

Host

Dans ce cas, les conteneurs bénéficient de **l'accès complet et direct au réseau disponible sur le serveur qui l'exécut**e. Il n'y a donc aucune isolation des communications entre les conteneurs et la machine hôte. Si ce mode s'avère très pratique pour des phases de développement, nous déconseillons de l'utiliser dans des contextes industrialisés. Gardez en tête que l'idéal reste tout de même de pouvoir bénéficier des possibilités d'isolation offertes. Notons d'autre part que **la distribution des ports à l'écoute est rendue plus complexe** pour l'utilisateur, car il faut connaître précisément les ports disponibles sur l'hôte.

Bridge

Le runtime chargé d'exécuter les conteneurs crée dans ce cas un réseau privé virtuel sur la machine hôte. Chaque instance bénéficie de sa propre adresse sur le réseau privé et peut donc **communiquer librement avec tous les conteneurs présents au sein du même bridge**. Pour communiquer avec l'extérieur, le bridge propose de créer des règles NAT qui permettront aux conteneurs de se présenter à l'extérieur du serveur hôte comme s'ils étaient l'hôte lui-même. Le NAT se charge de rediriger le trafic entrant vers le bon conteneur.

Dans ce contexte, les communications entre les conteneurs sont bien isolées de l'hôte. **La gestion des ports est grandement simplifiée** car il s'agit maintenant de connaître les ports disponibles sur le conteneur et non sur l'hôte. Nous pourrons ainsi lancer plusieurs serveurs web sur le port 80. Les règles NAT se chargeront d'associer un port disponible de l'hôte vers le port 80 du conteneur.

Overlay

Les réseaux Overlay offrent une solution à **la communication entre conteneurs sur différents serveurs**. Le principe sous-jacent est d'établir un tunnel entre les conteneurs qui s'exécutent sur différents serveurs. Tous les conteneurs qui s'exécutent sur ce réseau peuvent discuter entre eux et bénéficient de leur propre adresse IP.

Entre les serveurs, le trafic réseau ne passera pas en clair mais sera encapsulé dans un protocole spécialisé.

Il existe plusieurs solutions d'Overlay parmi lesquelles nous citerons VXLan, la solution native Linux utilisée par Docker, Weaveworks qui fournit soit une solution packagée et payante soit une version open source. Si vous souhaitez pouvoir bénéficier des services réseau de votre fournisseur cloud, nous vous recommandons de jeter un oeil du côté de Flannel. Flannel permet de créer des routes sur AWS et Google Cloud pour gérer les communications.

Underlay

Ce dernier mode de mise en réseau est principalement pensé pour des **utilisations "on premise".** Il repose sur le principe de donner aux conteneurs accès aux interfaces physiques de leur serveur. Par exemple, MACVLan propose de créer des sous interfaces portant une adresse MAC différente pour chaque conteneur. Cette solution permet de **créer des réseaux virtuels sur lesquels les conteneurs pourront communiquer**.

Dans ce cas, le trafic des conteneurs est isolé du serveur hôte, et ce dernier ne peut pas communiquer directement avec le conteneur.

Vous l'aurez compris, il convient de se poser la question du mode de déploiement et d'utilisation de vos conteneurs pour choisir la solution de mise en réseau qui répondra le mieux à votre contexte. Nous préférons **mettre en avant les modes Bridge et Underlay** qui permettent de garantir de bonnes performances tout en offrant des possibilités d'analyses fortes car elles n'impliquent pas d'encapsuler le trafic.

Les volumes persistants

Dès lors que l'on souhaite conteneuriser une application, la question de **la gestion des données** va rapidement se poser. En effet, à moins que votre application ne soit complètement sans état, vous avez besoin de stocker des données quelque part.

Un conteneur est par essence volatile. Il ne peut donc pas stocker de données, ce qui a priori proscrit son utilisation pour des applications ayant besoin de résilience.

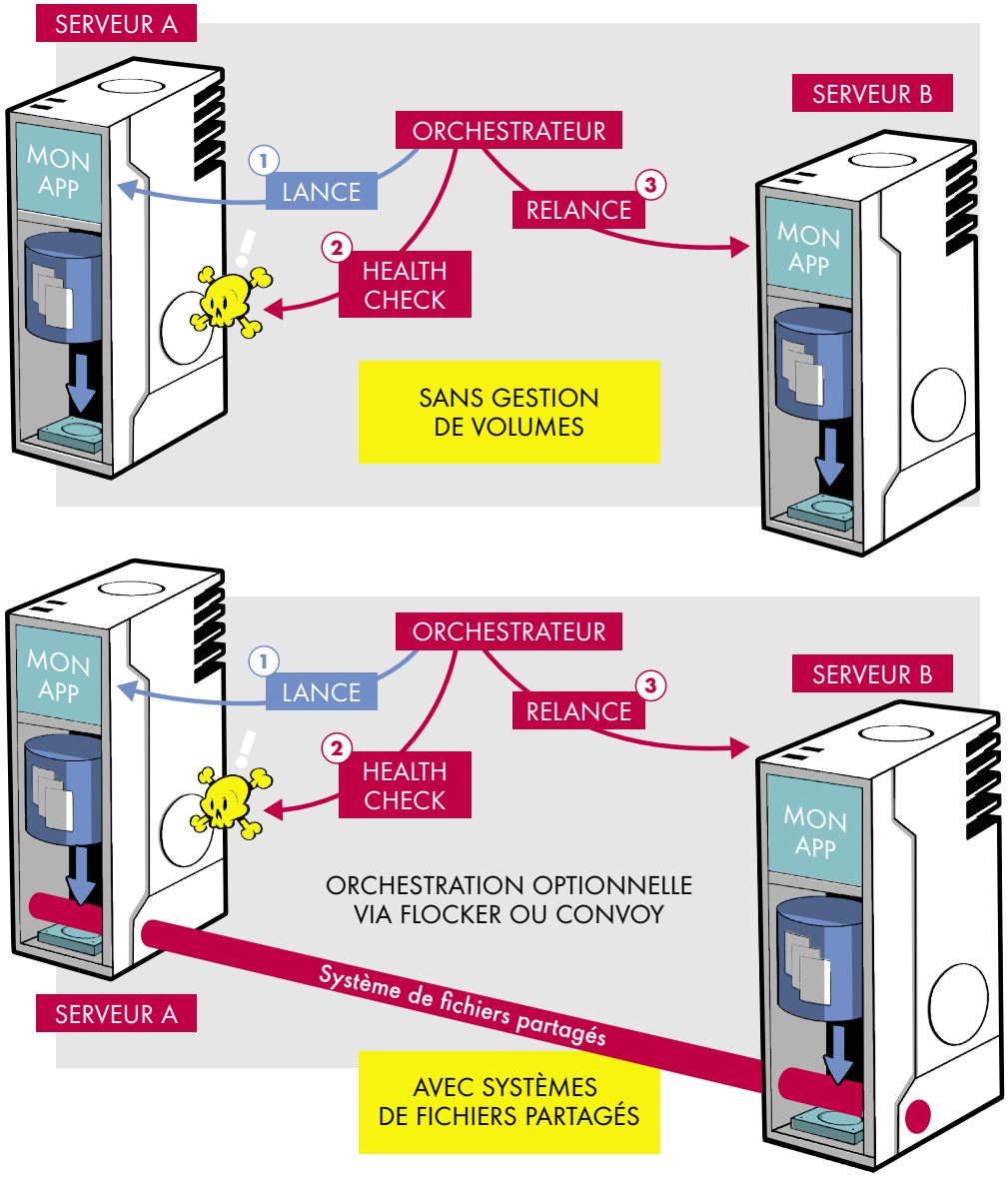
Initialement, la problématique a été adressée par une mécanique simple : les volumes. Un volume n'est rien d'autre qu'un point de montage au sein de votre conteneur faisant référence à un répertoire présent sur la machine hôte.

Les données sont accessibles depuis l'extérieur du conteneur, et la gouvernance de ces données sera donc **un enjeu majeur dans votre réflexion de conteneurisation**.

Cette mécanique simple fonctionne comme un charme dans un déploiement mono-serveur, mais, nous l'avons vu, c'est loin d'être la norme.

En règle générale, votre conteneur pourra être **instancié sur plusieurs noeuds d'un cluster.** Dans **le cas des conteneurs Stateful**, vous devrez pouvoir rattacher le volume contenant les données sur n'importe quel noeud du cluster.

" Un conteneur est par essence volatile. Il ne peut donc pas stocker de données, ce qui a priori proscrit son utilisation pour des applications ayant besoin de résilience. "



La persistance des données dans un environnement conteneurisé

Pour ce faire, il est nécessaire d'utiliser **un stockage de fichiers distribué**. Ces derniers ne sont pas une nouveauté dans le monde de l'infrastructure. L'idée est de répartir le système de fichiers sur un ensemble de noeuds. La totalité de ce "disque virtuel" est accessible depuis n'importe quelle machine. Cela permet d'exposer aux conteneurs le même répertoire, quelle que soit sa localisation sur le cluster.

Parmi les systèmes de fichiers du marché, nous préconisons **GlusterFS et Ceph** qui sont tous les deux compatibles avec les normes POSIX.

Il est aussi possible, dans le cas d'une utilisation de Docker, **d'utiliser un driver de volumes partagés**. En s'interfaçant avec Docker, ces drivers sont capables de gérer des opérations de haut niveau sur les volumes partagés. Ils permettent notamment de migrer les données avec les conteneurs à mesure qu'ils changent d'hôte au sein de votre cluster, et de réaliser des opérations de maintenance, telles que le backup/restore, le snapshot, etc.

Néanmoins, ces solutions ont vocation à disparaître avec **l'émergence d'une spécification Container Storage Interface (CSI).**

" La totalité de ce « disque virtuel » est accessible depuis n'importe quelle machine. Cela permet d'exposer aux conteneurs le même répertoire, quelle que soit sa localisation sur le cluster. "

Take away

Implémenter

Mettre en place deux composants principaux :

Un moteur d'exécution

un moteur d'exécution installé sur chacun des noeuds du datacenter. Nous pouvons citer RunC (Docker), Rkt (CoreOs) et LXC (Linux) ;

Un orchestrateur

un orchestrateur, qui garantit résilience, scalabilité et haute disponibilité. Il est aussi en charge d'optimiser le placement des ressources. Trois alternatives existent sur le marché : Kubernetes (Google), Mesos/Marathon et Swarm (Docker).

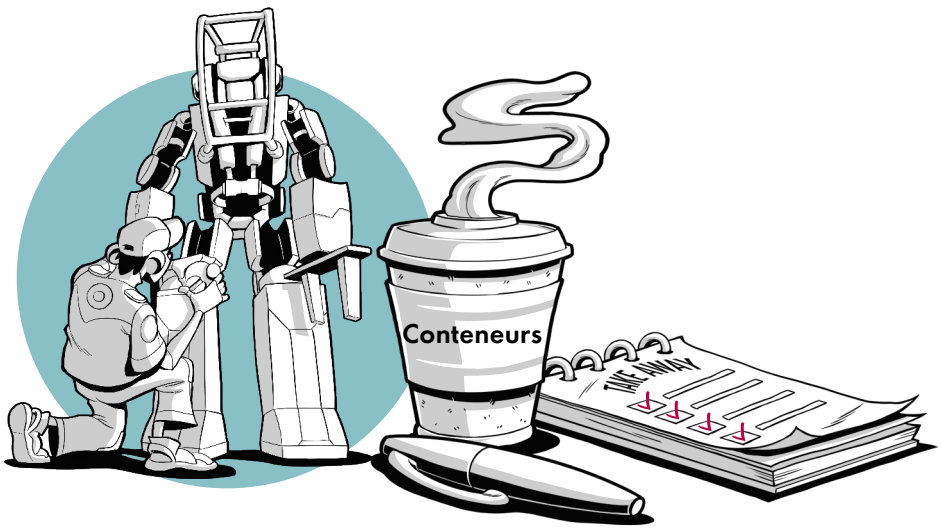
Porter une grande attention à deux composants transverses :

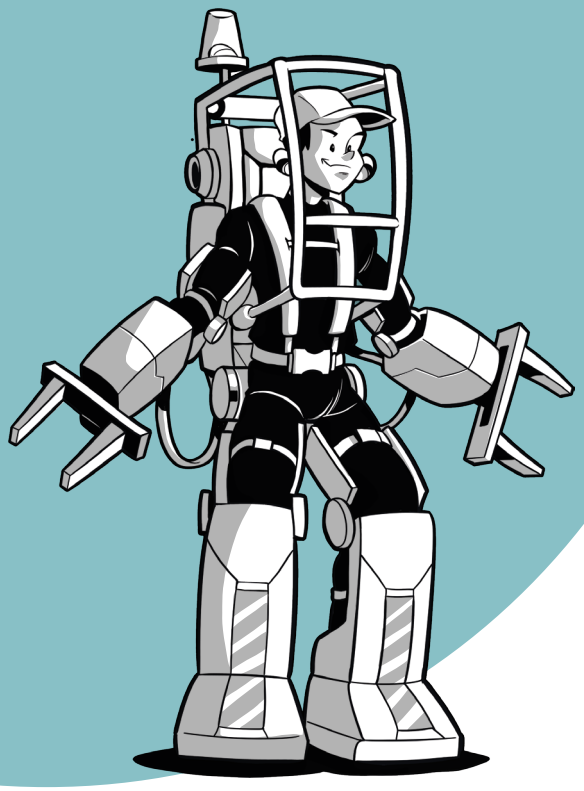
Le réseau

le réseau, vis-à-vis duquel le comportement de chaque conteneur peut être différent ;

Les volumes

les volumes, qui devront pouvoir être partagés ou déplacés en cas de conteneurs Stateful.





Exploiter

Nous avons abordé la mise en place “fonctionnelle” et technique des conteneurs.

Mais il existe un autre pan, tout aussi important, de l’adoption d’une stratégie d’entreprise basée sur les conteneurs. En effet, certains principes organisationnels risquent de se trouver bouleversés par cette nouvelle façon d’envisager le packaging des applications.

Pour autant, les bénéfices collatéraux seront à coup sûr une source de création de valeur pour votre entreprise.

Un accélérateur de l'adoption des grands principes DevOps

Le conteneur, et l'écosystème qui l'entoure, sont de formidables outils techniques.

Leur adoption entraîne également des effets vertueux sur **la collaboration entre équipes techniques, à savoir entre les développeurs (Dev) et les opérationnels (Ops)**[3]. La conteneurisation va permettre une réconciliation de ces équipes autour d'un même produit. En effet, historiquement, les Ops sont responsables des serveurs physiques, des réseaux, et de tout ce qui va tourner autour de l'infrastructure. Par extension, ils sont devenus garants du middleware permettant l'exécution des programmes développés. Cette franche séparation a créé un fossé entre les équipes, qu'il est aujourd'hui temps de combler.

L'adoption des conteneurs va permettre **un recentrage des Ops sur leur coeur de métier**, à savoir le système. Ils vont donc **mettre à la disposition des développeurs des OS**, vérifiés, durcis, et packagés par leur soin, sous forme de conteneurs.

En s'appuyant sur ce catalogue d'images OS, les développeurs vont pouvoir choisir et installer, par le biais de dépôts privés et contrôlés, les outils dont ils ont besoin pour **réaliser leurs programmes**. Ce socle, qui sera décrit sous forme de fichiers descriptifs (comme les Dockerfiles), pourra au final être validé (et donc tamponné) par les équipes d'exploitation, qui s'assureront de la conformité du système ainsi réalisé (conformité au niveau de la sécurité, de l'exploitabilité, etc.).

La nature immutable de l'image conteneurisée va permettre de protéger les secrets (mots de passe, clés privées) des environnements de production, en facilitant leur injection (par variable d'environnement ou base clé-valeur) au dernier moment par les équipes d'exploitation, permettant ainsi une **répartition naturelle des responsabilités**.

Enfin, il est possible de mettre en place **des systèmes de notification** lorsque les images de base OS sont amendées (dans le cadre du colmatage d'une faille critique de sécurité par exemple). Ces notifications permettent d'automatiser la reconstruction de tous les conteneurs dépendant de cet OS, en utilisant les mécanismes d'intégration continue. Ainsi, **les Ops peuvent directement patcher la production des Devs**, en respectant leurs standards de tests. On sort du système de ticketing, pour aller vers une chaîne de collaboration de bout en bout, dans laquelle le livrable sera passé de main en main et tamponné par tous les intervenants.

Néanmoins, ce mode de fonctionnement ne repose pas uniquement sur des outils, et si les conteneurs peuvent en être le catalyseur, une transformation DevOps passe avant tout par un **changement de culture et une augmentation drastique de la collaboration**.

3- Le TechTrends DevOps est disponible en téléchargement sur publicissapient.fr/services/engineering

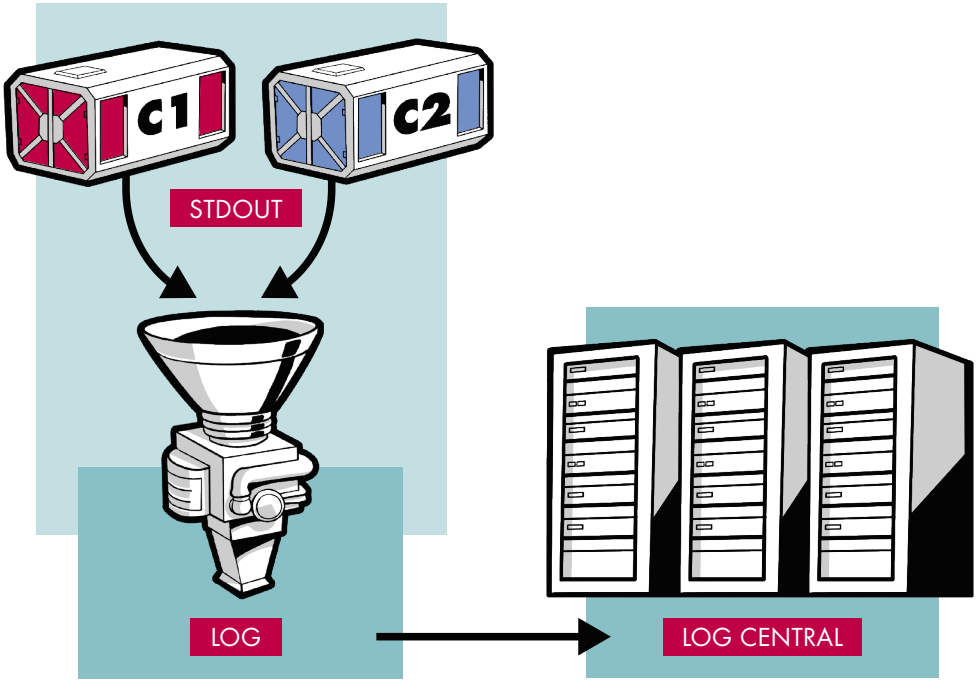
Une exploitation centrée sur le partage

La gestion des logs

Gérer les logs dans un contexte conteneurisé est moins trivial qu'il n'y paraît. Premièrement, le conteneur étant éphémère, les logs disparaissent en cas de suppression ou de crash de ce dernier. Ensuite, dans le cas d'un orchestrateur, il n'est pas toujours possible de présumer de l'emplacement physique d'un conteneur et en conséquence de l'emplacement des logs. En conclusion de ces deux points, il faut **veiller à ce que les logs soient centralisés**.

Plusieurs stratégies de gestion de logs s'offrent à nous. La première, conforme aux préconisations de 12-factor app[4], est **d'écrire tous les logs dans la sortie standard**. L'avantage est que tout passe par un canal unique ; il est donc plus simple de mettre en oeuvre un outillage pour la gestion des logs. L'inconvénient est une certaine perte de contexte que peut offrir l'utilisation de fichiers dédiés à des logs spécifiques dans un format hétérogène (audit, performance, etc.). Cette stratégie se marie très bien avec les conteneurs qui offrent une gestion native des logs inscrits sur la sortie standard et permettent de les centraliser dans un système dédié. Il est possible de les déverser dans des outils universellement répandus (comme Syslog) ou d'utiliser des formats plus récents, permettant de contextualiser plus facilement les informations (comme journald ou GELF).

4- https://12factor.net/fr/

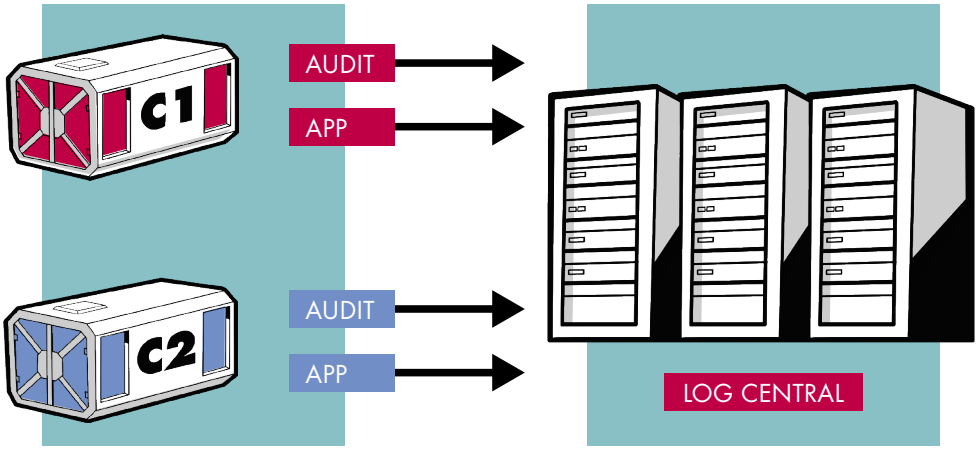


Gestion des logs via la sortie standard

“ Les Ops vont donc ***mettre à la disposition des développeurs des OS***, vérifiés, durcis, et packagés par leur soin, sous forme de conteneurs. ”

Une autre stratégie est de **diriger les logs de l'application directement dans le système de logs centralisé**. Cette approche a l'avantage d'être la plus précise, disposant du contexte dynamique de l'application, plus complet que celui du conteneur.

Elle est également intéressante si les formats proposés par le moteur de conteneurs ne sont pas compatibles avec une solution que vous auriez déjà mise en place. En contrepartie, la centralisation des logs devra être déployée localement à l'intérieur de chaque conteneur.



Déversement des logs directement dans le système centralisé

La supervision

Dans la majorité des cas, **une image contient un seul service**. Cette granularité fine permet de considérer de manière homogène l'ensemble des conteneurs déployés. En pratique, cela permet d'avoir systématiquement accès à un certain nombre de **métriques systèmes par conteneur**. Il est ainsi possible de monitorer pour chaque conteneur, le CPU, la mémoire, les accès disques, etc. Ces métriques sont exposées via des APIs, communément au format JSON et sont d'une part accessibles directement, et d'autre part - et surtout - exploitables par d'autres systèmes dédiés au monitoring.

Cependant, si les conteneurs mettent à disposition de manière uniforme des informations système, certaines informations spécifiques à l'application sont désormais contenues.

Prenons un exemple du monde Java dans lequel un certain nombre de métriques est traditionnellement exposé via le protocole JMX. Les données ne sont pas accessibles directement en utilisant les APIs fournies par le moteur de conteneurs. Il faudra alors définir une stratégie pour exposer ces informations depuis le conteneur puis y accéder depuis l'extérieur. Heureusement, dans cette démarche, **les métadonnées associées à une image** sont d'une grande aide, elles permettent par exemple de déclarer quels types de données spécifiques sont disponibles et comment y accéder.

Le monitoring comporte différentes phases : la collecte des données, la gestion des données et le requêtage. Différents outils existent pour chacune de ces phases et certains peuvent même en traiter plusieurs.

Différents **outils sont disponibles pour intégrer les métriques collectées** dans un système de monitoring global. Parmi les outils open source, notons Collectd ou Telegraf qui disposent de plugins tiers capables d'alimenter un grand nombre de solutions de stockage de métriques à partir des données des conteneurs. Des synergies apparaissent dans l'outillage ; Docker depuis la version 1.13 expose ses métriques au format Prometheus[5], qui peut directement lire les informations sans autre intermédiaire.

Il existe sur internet des solutions SaaS qui offrent des services très avancés. Datadog, par exemple, propose un panel de fonctionnalités très complet allant du suivi de production à l'analyse des performances ou l'alerting.

Les phases suivantes ne sont pas spécifiques aux conteneurs. Nous retrouvons donc, en toute logique, **les outils de monitoring** utilisés pour nos applications traditionnelles. Citons notamment InfluxDB qui propose des produits de collecte, d'analyse et d'alerting, le couple Graphite/Grafana ou encore le couple Prometheus/Grafana. Pour en savoir plus sur ces thématiques vous pouvez consulter le TechTrends DevOps[6].

Autre information très importante, **l'état de santé de l'application**. Cette notion est assez vague et le curseur de complexité varie généralement de la simple vérification de l'existence d'un processus au déroulement d'un test d'intégrité complet. Le positionnement de ce curseur dépend de chaque application. Les moteurs de conteneurs intègrent la notion de Health Check.

Dans le cas de Docker, le format d'image apporte une standardisation en spécifiant une instruction "health check" dont la valeur définit le script à utiliser. Il est intéressant de noter que le script qui vérifie l'état de santé du conteneur est livré avec celui-ci.

5- https://prometheus.io/

6- Le TechTrends DevOps est disponible en téléchargement sur publicissapient.fr/services/engineering

Une sécurité sous surveillance

Le principal argument des détracteurs de l'adoption en production d'un système à base de conteneurs reste, encore aujourd'hui, **la sécurité**. Pourtant, il existe de nombreux outils qui permettent d'envisager sereinement une plateforme conteneurisée en production. Certes, comme dans tout projet, conteneurisé ou non, la mise en place de toutes les bonnes pratiques de sécurité peut s'avérer coûteuse. Mais, au regard de ce qu'une faille de sécurité peut vous faire perdre, le calcul est rapidement fait.

En premier lieu, **votre conteneur embarque un OS**. Celui-ci peut présenter des failles de sécurité propres (est-il encore nécessaire de parler de SSL Heartbleed ?). Des outils externes permettent de "scanner" le contenu de vos conteneurs, et de le confronter à une base officielle de failles de sécurité. On citera par exemple Docker Security Scanning[7] ou la solution Clair de CoreOS. Ce genre d'outil permet aux équipes système de surveiller leurs images OS et de les patcher en conséquence.

Les OS des machines hôtes doivent bien sûr eux aussi être mis à jour régulièrement. Enfin, la protection n'est pas complète **si le moteur d'exécution** n'est pas également régulièrement patché.

Ensuite, il convient de **réduire au minimum les surfaces d'attaques possibles** sur le conteneur.
À cette fin, nous recommandons l'application de dix bonnes pratiques essentielles :

1. Rendez vos conteneurs "readOnly". Cela permettra de limiter les attaques sur les volumes partagés par l'hôte.
2. Dédiez des volumes sur des points de montage spécifiques à chaque conteneur afin d'éviter les attaques par rebond sur le fileSystem.
3. Choisissez la bonne communication réseau.
4. Exposez le minimum de ports entre le conteneur et son hôte.

> " Il existe de nombreux outils qui permettent d'envisager sereinement une plateforme conteneurisée en production. "

7- https://docs.docker.com/docker-cloud/builds/image-scan/ (https://dockr.ly/2QCvo6o)

5. Bannissez l'utilisation de root pour lancer des processus à l'intérieur du conteneur.
   Sur l'hôte, réservez un utilisateur dédié pour exécuter les conteneurs.

6. Réduisez le nombre de processus attaquables. Les conteneurs offrent des OS minimalistes (Busybox et Alpine) qui réduisent grandement le nombre de processus exécutés et donc la surface d'attaque (pas de SSH par exemple). Si vous devez utilisez un OS plus riche, il existe un outillage natif sous Linux qui permet de réduire les privilèges d'exécution des différents process. On citera Seccomp, SELinux et AppArmor. De la même façon, certains OS sont spécifiquement dédiés aux machines hôtes de conteneurs, et respectent les principes de frugalité qui favorisent une meilleure sécurité. On citera CoreOs, RancherOS, LinuxKit et RedHat Atomic.

7. Tracez toujours la provenance de vos images de base.

8. Préférez l'utilisation d'un registre privé pour héberger vos images.

9. Signez autant que possible vos images[8] pour en garantir l'intégrité.

10. Posez des limites (RAM, CPU) à vos images afin de contrer les attaques type Déni de Service.

Aujourd'hui, la sécurité d'un système de conteneurs s'est déplacée d'un terrain technique à un terrain organisationnel : tous les outils sont présents et mûrs, et **c'est maintenant à votre organisation de mûrir pour embrasser les bonnes pratiques** et garantir l'intégrité de vos logiciels et de vos données. Afin de réduire au minimum la surface d'attaque, la bonne pratique à appliquer en toute circonstance, est d'adopter le principe d'interdire par défaut tous les accès. Les Devs et les Ops n'ajouteront alors que ce qui est strictement nécessaire à l'application.

8- https://blog.docker.com/2015/08/content-trust-docker-1-8/ (https://dockr.ly/1EyCxrR)

Un pas de plus vers une hybridation cloud public/cloud privé

Ces deux dernières années, les principaux éditeurs de cloud public (AWS, Google, IBM et Azure, mais aussi OVH, Digital Ocean, etc.) ont mis les bouchées doubles pour proposer une offre “Conteneurs” digne de ce nom.

La plupart des fournisseurs proposent **un registry de conteneurs privé, ainsi qu’un orchestrateur**. On notera qu’à mi-2017, Azure possède la couverture la plus complète, en proposant Kubernetes, Mesos et Swarm, là où AWS a fait le choix de son propre orchestrateur (dont la syntaxe se rapproche de celle de Marathon).

Le mouvement entamé sur le cloud public trouve bien sûr son pendant sur le cloud privé, avec **OpenShift et CloudFoundry**.

La richesse de cet écosystème entraîne deux effets bénéfiques pour nous, utilisateurs.

Tout d’abord, il est **possible d’expérimenter une plateforme conteneurisée à moindre frais**. En effet, la surcouche “conteneur” est bien souvent gratuite dans ces offres, l’utilisateur ne payant que pour les ressources machine sous-jacentes.

Ensuite, grâce à la standardisation des packages, il devient parfaitement envisageable de **réaliser une hybridation entre votre datacenter et le cloud public**. En effet, à versions équivalentes, il n’y a aucune raison qu’un conteneur s’exécutant sur vos systèmes ait un comportement différent sur une plateforme de conteneurs cloud. De plus, certains orchestrateurs, comme Kubernetes, sont capables de gérer des noeuds partagés sur plusieurs datacenters. En partant de ce principe, des use cases comme le débordement capacitaire sur le cloud deviennent plus facilement envisageables que par le passé.

“ Grâce à la standardisation des packages, il devient parfaitement envisageable de réaliser une hybridation entre votre datacenter et le cloud public. ”

Take away

Exploiter

Embrasser

Embrasser le mouvement DevOps. La collaboration entre développeurs et exploitants est une composante essentielle de votre réussite ;

Adapter

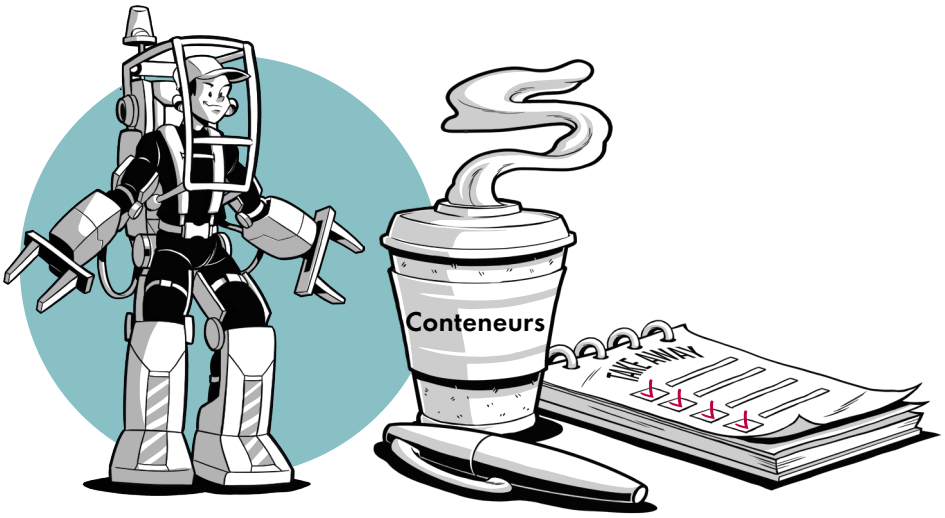
Adapter le monitoring et la métrologie. Qu'il s'agisse de logs ou de métriques, il convient de choisir des solutions adaptées au plus tôt ;

Penser

Penser sécurité. Tous les outils nécessaires à la traque des failles sont là, il suffit de s'en servir ;

Expérimenter

Expérimenter, à moindre frais, en utilisant le cloud public. Vous pourrez ensuite envisager sereinement une solution "on premise", voire une solution hybride.



Conclusion

La mise en place d'une plateforme de conteneurs n'est pas une sinécure.

L'écosystème est actuellement en plein essor, et malgré des tentatives de standardisation, nous sommes encore dans une **phase de divergence** dans laquelle chaque éditeur essaye de tirer son épingle du jeu. Même si **Docker a une franche longueur d'avance**, en particulier sur les moteurs d'exécution, la bataille fait toujours rage sur les chantiers transverses, comme les orchestrateurs.

Néanmoins, nous sommes persuadés que les solutions sont aujourd'hui mûres, et **qu'elles répondent à des besoins concrets**. Si nous devions n'en citer qu'un, nous insisterions sur la **polyvalence** que les conteneurs peuvent amener dans votre SI. Le temps des SI mono-langages, mono-frameworks est aujourd'hui dépassé. Chaque équipe de développement doit aujourd'hui pouvoir choisir ses propres outils pour être la plus performante possible. Il ne viendrait à l'idée d'aucun data scientist de programmer des algorithmes de machine learning en PHP. Cette fragmentation a nécessairement un coût, en particulier pour les équipes d'exploitants qui doivent avoir de plus en plus de cordes à leur arc. Si l'émergence des conteneurs ne neutralise pas complètement ce coût, elle permet néanmoins de l'amortir. La standardisation des plateformes d'exécution, l'unité des modules à déployer permettent de **simplifier les procédures opérationnelles**.

Et, de facto, **ce packaging commun force les équipes Dev et Ops à se rapprocher**, à nouer de plus étroites collaborations. Nous sommes convaincus que c'est la direction qu'il faut prendre, et l'adoption des conteneurs vous permettra de suivre (à marche forcée) cette voie vertueuse.

Enfin, ne nous le cachons pas, les conteneurs sont partout et risquent de s'installer longtemps dans le paysage. **Les grands acteurs du cloud les ont adoptés depuis longtemps** et démocratisent aujourd'hui leur usage. Les éditeurs prévoient tous dans les prochains mois des versions conteneurisées ou container ready, que ce soit dans le monde des OS (RedHat, Microsoft), des bases de données (Redis, Oracle, etc.) voire de certains logiciels "traditionnels" (comme le logiciel de paie Sage). Aujourd'hui, rien ne s'oppose à ce que vous lanciez dès à présent vos premières expérimentations, afin d'embrasser ce virage technologique.

Les conteneurs ne sont qu'un pas vers un nouveau type de SI, plus frugal en nombre de serveurs et où les ressources seront mieux gérées. D'ailleurs, conjointement à l'adoption massive des conteneurs, **la montée des architectures Serverless enfonce le clou,** et tend à faire disparaître la notion même de serveur d'application. La combinaison Serverless/Conteneur laisse augurer **la création d'architectures éphémères**, consommant un minimum de ressources, et bénéficiant au mieux de l'élasticité grandissante des datacenters.

Take away

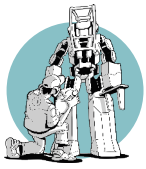
Conteneurs

Projeter

La conteneurisation propose des bénéfices immédiats dans l'ensemble du SI :

- sur le poste du développeur, où elle permet de rapidement installer l'ensemble des composants logiciels d'une application ;
- dans l'usine logicielle, où elle permet d'atteindre une élasticité horizontale à moindre coût ;
- en production, où elle facilite le déploiement continu ou la construction d'environnement "à la demande" ;
- à l'échelle de l'entreprise, où elle facilite l'intégration de stacks techniques variées.

Implémenter



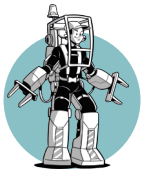
Mettre en place deux composants principaux :

- un moteur d'exécution installé sur chacun des noeuds du datacenter. Nous pouvons citer RunC (Docker)Rkt (CoreOs) et LXC (Linux) ;
- un orchestrateur, qui garantit résilience, scalabilité et haute disponibilité. Il est aussi en charge d'optimiser le placement des ressources. Trois alternatives existent sur le marché : Kubernetes (Google), Mesos/Marathon et Swarm (Docker).

Porter une grande attention à deux composants transverses :

- le réseau, vis-à-vis duquel le comportement de chaque conteneur peut être différent ;
- les volumes, qui devront pouvoir être partagés ou déplacés en cas de conteneurs Stateful.

Exploiter



- Embrasser le mouvement DevOps. La collaboration entre développeurs et exploitants est une composante essentielle de votre réussite ;
- Adapter le monitoring et la métrologie. Qu'il s'agisse de logs ou de métriques, il convient de choisir des solutions adaptées au plus tôt ;
- Penser sécurité. Tous les outils nécessaires à la traque des failles sont là, il suffit de s'en servir ;
- Expérimenter, à moindre frais, en utilisant le cloud public. Vous pourrez ensuite envisager sereinement une solution "on premise", voire une solution hybride.

À lire et à relire

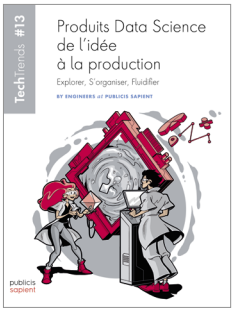
Pour télécharger l'un des TechTrends en version éléctronique (pdf), rendez-vous sur le site publicissapient.fr/services/engineering
Pour demander une version papier, envoyez un mail à techtrends@publicissapient.com

Pour en savoir plus sur le Data, le Cloud, le Web, les architectures Java, la mobilité et l'agilité, rendez-vous sur blog.engineering.publicissapient.fr

Techtrends #8
Mobile
Comprendre, Façonner, Innover

Techtrends #9
Cloud
Préparer sa migration, Séléctionner son offre, Choisir ses technologies

Techtrends #10
Studio
Démarrer son projet, Piloter son projet, Pérenniser son produit



Techtrends #13
Produits Data Science
Explorer, S'organiser, Fluidifier

Techtrends

#11 Internet of Things
#7 Back
#6 DataLab
#5 Front
#4 Craftsmanship
#3 Agilité
#2 DevOps
#1 Data

Autres parutions

Scrum Master Academy
Le Guide du Scrum Master d'élite

Les Communautés de Pratique en Pratique
Le Mini Guide

Product Academy
Le guide des Product Managers et des Product Owners d'élite

Les auteurs

Publicis Sapient Engineering

Thomas AUFFREDOU

Alexis "Horgix" CHOTARD

Pablo LOPEZ

Jean-Louis RIGAU

WeScale

Cédric HAUBER

Sébastien LAVAYSSIÈRE

Séven LEMESLE

MERCI À

Damien Baron, Anne Beauchart, Thomas Cousin, Christophe Heubès, Laetitia Janné, Aurélien Maury, Jonathan Raffre, Margot Robine, Clément Rochas, Céline Rochay, Julien Smadja, Jean-Pascal Thiery et Pauline Tirman.

Direction artistique : **Publicis Sapient Engineering**
Design graphique : **micheletmichel.com** / Illustrations: **lasaintepaire.com**

En partenariat avec WeScale

Publicis Sapient
94 Avenue Gambetta, 75020 Paris
+33 (0)1 53 89 99 99
engineering@publicissapient.com

Toutes les informations sur :
publicissapient.fr/services/engineering