**DEDICACE**

Je dédie spécialement ce document à :

A la personne qui a toujours été là pour moi et sans qui ma vie aurait été abandonner au hasard. Ma mère.

Tu es probablement la femme la plus forte que je connaisse. Puisse Dieu te garder en bonne santé !!

REMERCIEMENTS

* A monsieur ASSIDENU Kodjo, mon directeur de mémoire ;
* à monsieur TOSSOU Francis, mon maitre de stage.

Je dis un grand merci pour leurs aides incommensurables tout au long de mon stage jusqu’à la rédaction de ce document. Que **DIEU** vous bénisse et vous le rende au centuple.

AVANT-PROPOS

Il y a un certain écart entre les études et la vie réelle en entreprise. Pour comble un tant soit peu cet écart et permettre aux apprenants d’avoir une idée sur les réalités en entreprise, il est plus que nécessaire de les orientés vers les entreprises. C’est dans cet optique que le CIC a instauré un stage de 10 semaines pour les étudiants finissants. Pour notre stage, nous avons eu le choix entre plusieurs entreprises mais nous avons préféré BANKEVI GROUPE, une entreprise composée en majorité des jeunes travailleurs aussi compétents et motivés les uns que les autres pour nous inculquer l’esprit d’entreprise nécessaire pour nos projets futurs.

RESUME

Ce projet s’inscrit dans le cadre d’un stage de fin d’études au sein de l’entreprise BANKEVI GROUPE pour l’obtention de la licence professionnelle en maintenance et réseaux informatiques au CIC. L’entreprise m’a confié la mission de mettre en place un écosystème microservices prêt à l’emploi pour ses projets futurs. Un écosystème dans lequel les dépendances des applications Web fonctionnent indépendamment les uns des autres, pour que chaque module puisse être déployé indépendamment des autres. Ainsi si une fonctionnalité est plus sollicitée que les autres l’on pourra le multiplier autant de fois qu’on le souhaite sans être obligé de redéployer toute l’application à nouveau comme cela se fait dans l’entreprise avant mon stage. Les microservices se trouvent être la solution la plus appropriée pour aboutir à un résultat digne de ce nom.

ABSTRACT

INTRODUCTION GENERALE

Travailler sur un même projet en groupe n’est pas une tâche facile sans l’existence d’un système approprié pour faciliter le déroulement de ce projet. Le constat est que la plupart du temps les applications ne fonctionnent plus aussi bien en production que sur le serveur local, Le même problème subsiste entre les développeur d’un même projet. Une cause récurrente à ce problème est que les versions des dépendances de l’applications ne sont pas les mêmes sur les machines des développeurs ou ne sont pas les mêmes sur le serveur de production que sur le serveur local et pire encore l’inexistence de certaines dépendances. C’est ainsi que pour régler ce problème il devient indispensable de rendre les dépendances plus autonomes d’une part et le tout fonctionnant au sein d’un écosystème prêt à l’emploi que tous les développeurs pourront utiliser aussi bien sur le serveur local qu’en production afin de s’assurer que les applications aient les mêmes comportements non seulement chez les développeurs mais également en production. Pour venir à bout de ces problèmes on nous a confié la mission de mettre en place un écosystème microservices pour les applications Web au sein de l’entreprise BANKEVI GROUPE. Ce document a été produit afin de justifier l’élaboration du projet et s’articule autour des chapitres suivants : un premier chapitre portera sur la présentation du cadre d’étude et de stage. Dans un deuxième chapitre, nous nous intéresserons à la présentation générale du projet. Une généralité sur les microservices et leurs écosystèmes sera présentée dans un troisième chapitre. Enfin, le chapitre IV sera consacré à la réalisation. Nous couronnerons notre travail par une conclusion générale.

**Chapitre**

**1**

**:**

**Présentations du CIC,**

**BANKEVI GROUPE**

**Introduction**

Dans cette première partie, nous nous intéresserons à la présentation du centre de formation et du cadre de stage.

**1 Centre de formation : Centre Informatique et de Calcul (CIC)**

**1.1 Présentation**

Le Centre Informatique et de Calcul (CIC) est un établissement et un centre de ressources de l’Université de Lomé. Créé par arrêté N°67/MENRS du 26 Septembre 1988, le CIC apporte non seulement un appui logistique en informatique aux établissements et à l’administration de l’Université de Lomé mais offre également un cadre pour la formation informatique. Situé dans la zone sud de l’Université de Lomé, le Centre Informatique et de Calcul fait face à la Direction des Affaires Académiques et de la Scolarité (DAAS).

### **Historique**

Le CIC a été créé en 1988 avec la mission d’apporter un appui logistique et technique en informatique aux différentes écoles, facultés et aussi à l’administration de l’Université de Lomé et de former des cadres supérieurs en informatiques. Le CIC hébergeait le Centre inter Africain de Formation à la Maintenance des Equipements Micro-informatiques (CAFMICRO) qui offrait une formation POST-DUT aux techniciens supérieurs en maintenance informatique. En 1995, une filière DUT en informatique destinée à former des techniciens supérieurs en informatique a été créée.

Avec l’avènement du système LMD (Licence-Master-Doctorat) en2009, le CIC forme désormais ses étudiants dans les filières « Maintenance et Réseaux informatiques » et « Génie Logiciel », une formation à la fin de laquelle un diplôme de licence professionnelle leurs est fourni.

Tout récemment en mars 2016 un master international en informatique en partenariat avec l’université de Technologie de Belfort-Montbéliard (UTBM)a été ouvert.

### **Activités**

Le CIC abrite en son sein :

* L’académie CISCO pour la formation et la préparation du CCNA (CISCO Certified Network Associate) ;
* Le Centre Virtuel Africain (CVA) ouvert aux enseignants pour le partage des cours en ligne ;
* Le projet Pan Africain e-Network, un programme de télé-enseignement en partenariat avec quatre (04) universités Indiennes dont les universités de Madras, Delhi, Amity International et Birla Institute of Technology and Science (BITS) pour des formations en Master en informatique, des MBA, des certifications en comptabilités et en électronique.

Le CIC est aussi le siège du Réseau national d’Education et de Recherche (TogoRER) qui vise à regrouper tous les établissements d’enseignement supérieur du Togo en vue de leur offrir des services TIC et internet.

### **Organisation administrative**

Le Centre Informatique et de Calcul dépend de la présidence de l’université de Lomé et est dirigé par un directeur assisté d’un directeur adjoint.

La structure administrative est composée de :

* L’assemblée d’établissement ;
* La Commission Scientifique et Pédagogique (CSP) ;
* La Direction ;
* Le Collège des Chefs de division ;
* Le Service des examens ;
* La Cellule d’information Pédagogique (CIP) ;
* La Commission des finances et du budget ;
* La Cellule Assurance Qualité.

Le CIC dispose actuellement de quatre (04) divisions :

* ***Division enseignement*** en charge de la formation universitaire au grade Licence et Master.
* **Division système d’information** chargée de l’étude, de la conception, de la mise en œuvre de la gestion du système d’information global de l’Université de Lomé.
* **Division Maintenance informatique** chargée de la maintenance des équipements informatiques de l’Université de Lomé, de la mise en œuvre des programmes d’assemblage de micro-ordinateurs initiés par le CIC, de la réalisation des différentes études nécessaires à l’informatisation de toutes les entités de l’Université de Lomé en collaboration avec les autres divisions.
* **Division Développement** chargée des études informatiques nécessaires à l’informatisation des différentes entités de l’Université de Lomé, de la conception et du développement des applications informatiques et de la maintenance des logiciels développés.

La figure qui ci-contre présente l’organigramme du Centre Informatique et de Calcul.

Figure1 : Organigramme de CIC

### **Offres de formation**

Le Centre Informatique et de Calcul offre, dans le domaine des Sciences et Technologie (ST), deux parcours au grade Licence :

* Licence Professionnelle en informatique, Spécialité Génie Logiciel ;
* Licence professionnelle en informatique, Spécialité Maintenance et Réseaux Informatiques.

Le CIC offre également en Co-diplôme avec l’Université de Technologie de BelfortMontbéliard (UTBM) de France, une formation en Master informatique :

* Master informatique, Spécialité Génie Logiciel ;
* Master informatique, Spécialité Systèmes et Réseaux.

En dehors de ces deux principales formations, le CIC forme à la préparation au *CISCO Certified Network Associate (CCNA).*

L’accès à la formation en Licence au CIC est conditionné à la réussite au concours d’entrée ouvert aux titulaires du diplôme de Baccalauréat, séries C, D, E ou équivalant avec un excellent niveau en mathématiques.

## Présentation du cadre de stage : BANKeVI GROUPE

BANKeVI GROUPE est une société qui évolue dans plusieurs domaines d’activités avec ses départements :

* B-COM : c'est département produisant tout ce qui est lié à la visibilité et à la communication
* BANK’EVENT : c'est le département qui s'occupe des prestations évènementielles
* B-TECH : c'est le département informatique.

Fondé par deux jeunes entrepreneurs togolais aux compétences complémentaires, elle a su orienter ses activités, en ne se contentant pas de gérer des projets classiques mais en misant sur des collaborateurs aux compétences et domaines d'activités variés. Il en résulte aujourd'hui, une grande expérience, et une véritable ouverture d'esprit qui lui permet de proposer des services d’une grande qualité adaptée aux besoins de nos clients. Située dans le quartier d’Attiégou, non loin de Togo 2000, BANKeVI GROUPE est joignable aux adresses suivantes :

* BP : 01BP4929 Lomé-TOGO ;
* Téléphone : +228 22 26 79 96 ;
* E-mail : [contact@bankevi.com](mailto:contact@bankevi.com) ;
* Site web: [www.bankevi.com](http://www.bankevi.com).

### **Organigramme**

Figure 2: Organigramme de BANKEVI GROUPE

### **Missions de BANKeVI GROUPE**

BANKeVI GROUPE conçoit et gère des solutions qui améliorent l’efficacité des organisations et le pouvoir d’achat des individus. Elle est spécialisée en développement de produits et services appliqués aux besoins des entreprises et particuliers. Elle propose des solutions performantes à travers la création, le déploiement et le suivi post-lancement des produits qu’elle développe pour ses clients. A l'écoute de leurs besoins, l’équipe de BANKeVI GROUPE apporte une réponse personnalisée à la réalisation des projets de sa clientèle. Elle développe une gamme complète de services pour accompagner sa clientèle dans l’atteinte de ses objectifs de productivités :

* Une approche et création personnalisée pour chaque projet : spot publicitaire, film documentaire, histoire de vie etc.
* Une gamme complète de services : caravane, lancement de marque, prestation d’hôtesses, couverture d’évènements etc.
* Une conception intégrant la dimension commerciale : infographie, imprimerie, sérigraphie etc.
* Une équipe performante à la pointe des technologies : création d’application mobile, de logiciel, site internet etc.

Dans un domaine en perpétuel mouvement, BANKeVI GROUPE anticipe les évolutions technologiques et innovations afin de proposer un service de qualité. Convaincu de la vocation économique de l'informatique et de la bonne stratégie d’entreprise, elle adopte une approche commerciale dès la conception des projets de sa clientèle et lui propose un accompagnement au développement de sa stratégie.

### **Les Départements de BANKeVI GROUPE**

* Division B-TECH

Ce département conçoit et gère des solutions qui améliorent l’efficacité des organisations et le pouvoir d’achat des individus. Elle est spécialisée en développement de logiciel appliqué aux besoins des entreprises et particuliers. Elle propose des solutions performantes pour la création, le déploiement et la maintenance de progiciels. A l'écoute des besoins de ses clients, les équipes de la B-TECH apportent une réponse personnalisée à la réalisation de leurs projets. Ils développent une gamme complète de services pour accompagner sa clientèle dans la mise en œuvre de ses projets.

C’est dans ce département que nous avons effectué notre stage.

* Division B-COM

La publicité est un élément constitutif du mix et des stratégies marketing. Comment briser le mur de l’indifférence autour des marques et produits. Comment jeter un regard différent sur des produits souvent connus, banalisés et des codes publicitaires déjà bien établis. Comment émerger et développer un territoire d’expression qui correspond pleinement à la clientèle. La B-COM mobilise ses talents créatifs pour donner au message de sa clientèle la force et l’impact nécessaire pour produire des résultats. A l’heure où le potentiel économique d’Internet est indéniable, l’élaboration d’une stratégie Web efficace est aujourd’hui une nécessité.  
Comment construire une stratégie digitale qui soit le véritable reflet de la stratégie d’entreprises sur Internet, et qui permette d’en optimiser l’efficacité et la performance ?  
La B-COM a acquis de solides compétences dans la réalisation et la promotion de sites internet publicitaire, vitrines, marchands et responsives. Au-delà des dispositifs 100% digitaux, elle intervient dans l’élaboration de dispositifs digitaux innovants qui plongent les clients dans des univers interactifs et créatifs.

* Division B-EVENT

Quel que soit le besoin, le département évènementiel propose un panel complet de services à forte valeur ajoutée. Voici quelques exemples de prestations et d’accompagnement qu’il pourrions-vous offrir :

* Organiser des road show dans plusieurs villes pour aller à la rencontre de son public.
* Piloter la conception, le montage et la réalisation de conventions, congrès et rendez-vous business.
* Elaborer des programmes éditoriaux et d’information à travers la sélection des thématiques et des intervenants.
* Dynamiser un stand sur un salon, du design jusqu’à l’aménagement.
* Concevoir des campagnes de communication multicanales en print, web, digital et réseaux sociaux.
* Imaginer des concepts originaux pour rencontrer et engager ses clients : Petits déjeuners, soirées, voyages inventives, jeux concours, challenges ...

**Chapitre 2**

**:**

**E**

**tude préalable du sujet**

**Introduction**

Une parfaite connaissance du sujet aide à déterminer la portée du projet et celle de la solution à implémenter. Il s’avère indispensable de disposer d’un ensemble d’informations sur l’infrastructure informatique existante afin de déceler ses points forts et ses insuffisances. Ces informations vont en grande partie affecter les décisions que nous prendrons dans le choix de la solution et de son déploiement. En partant de l’étude de l’existant, nous listerons les manquements dans cet existant, définirons le contexte de notre projet et dégagerons la problématique de ce dernier.

**1 Présentation du projet**

Dans une entreprise, les développeurs travaillant sur un même projet sont souvent confrontés à un problème d’environnement, ainsi une application fonctionnant très bien sur la machine d’un des développeurs peut ne plus avoir le même comportement sur la machine d’un autre développeur, ceci ne facilite pas le travail en équipe et fait perdre du temps à l’équipe de développement. De même un site Web fonctionnant très bien sur le serveur local peut changer radicalement de fonctionnement et comporte même des bugs suite à son hébergement. Une cause à ces problèmes serait le manque d’une des dépendances de l’application. Assurer le bon fonctionnement de l’application sur tous ces environnements que ce soit en local ou chez les autres développeurs de l’équipe et surtout en production devient plus que nécessaire pour la rentabilité de l’entreprise. Pour assurer cette rentabilité, Il devient donc nécessaire de mettre en place un écosystème que les développeurs pourront tous utiliser en local pour s’assurer que le site ait le même comportement chez tous les membres de l’équipe d’une part et qu’on pourra ensuite déployer sur le serveur de production pour s’assurer que le site ait également le même comportement aussi bien en local qu’en production d’une autre part. Notre choix se porte sur les microservices qui proposent une solution à ce type de problème et au problème de dépendance en général. De plus avec un écosystème microservices les dépendances des applications fonctionnent indépendamment les uns des autres, ainsi si un service est plus sollicité que les autres l’on pourra le multiplier autant de fois qu’on le souhaite indépendamment des autres. D’où, notre projet consiste à mettre en place un écosystème microservices pour les applications Web au sein de l’entreprise BANKEVI GROUPE.

**2 Problématique**

Créer une application est une tâche qui à priori semble se limiter au codage, mais pouvoir organiser les services et dépendances dont elle a besoin en est une autre. Le constat qu’on fait aujourd’hui est que la plupart des applications peuvent ne plus avoir le comportement souhaité suite à un manque de dépendance aussi bien sur le serveur local que sur le serveur en production, ce qui complique le travail en équipe. De plus la plupart de ces applications reposent sur une architecture monolithique ce qui rend plus difficile les éventuelles évolutions dans la mesure ou toutes les fonctionnalités et dépendances des applications sont regroupées. Au fur et à mesure qu’on ajoute de plus en plus de fonctionnalités dans les applications on augmente leurs complexités et leurs tailles et cela ralentit le développement, et la vitesse à laquelle on peut tester et redéployer ces applications. Apporter des modifications peut parfois être très complexe et réclamer des semaines, voire des mois de travail pour s’assurer que la nouvelle fonctionnalité que l’on souhaite ajouter s’intègre correctement avec l’existant et ne risque pas de créer d’incidents sur d’autres composantes de l’application. Si une des dépendances de l’application est indisponible toute l’application peut se retrouver très vite hors service. La localisation de la source en cas de problème n’est pas du tout évidente. Ce qui oblige la plupart des entreprises et BANKEVI GROUPE utilisant cette approche monolithique pour le développement de leurs applications Web à préférer la stabilité à l’innovation de leurs codes. Pour être plus efficace, il est nécessaire de se poser les questions suivantes :

* Comment rendre autonome chaque dépendance des applications Web conçus par l’entreprise ?
* Comment s’assurer que l’application ait le même comportement aussi bien en local qu’en production ?
* Comment s’assurer que l’intégration d’une nouvelle fonctionnalité se fasse sans régression ou avec un impact minimal sur les autres déjà existantes ?
* Comment s’assurer que seules les fonctionnalités dont les dépendances sont indisponibles soient hors service et non toute l’application ?
* Comment multiplier une dépendance d’une application Web au besoin sans pour autant redéployer toute l’application ?

Pour pallier à cette problématique, il est nécessaire de mettre en place un système permettant de réduire sinon éliminer cette interdépendance entre les fonctionnalités et faciliter le déploiement des applications Web. Pour ce faire les microservices se voient être une solution efficace et la mieux adaptée.

1. **Objectifs**
   1. **Objectif général**

Le projet a pour objectif général la mise en place d’un écosystème microsevices pour les applications Web afin de briser sinon réduire l’interdépendance entre les fonctionnalités et services des applications Web.

* 1. **Objectifs spécifiques**

Le projet devra permettre à la fin de son exécution d’atteindre les objectifs spécifiques suivants:

* Chaque dépendance d’une application Web devra fonctionner dans un environnement propre à elle-même, ainsi l’on pourra la modifier, la supprimer et la multiplier au besoin avec un impact minimal sur les autres dépendances de l’application ;
* Concevoir un écosystème de microservices contenant les différents conteneurs de l’application ;
* Gérer l’interopérabilité entre les différents conteneurs ;
* Faciliter le déploiement des applications.

1. **Résultat attendu**

Création d’un écosystème de microservices contenant les différentes dépendances ainsi que les applications Web qui les utilisent.

1. **Etude de l’existant**
   1. **Equipements réseau**

L’entreprise dispose d’un point d’accès qui permet à toute l’équipe de BANKEVI de se connecter à internet. La connexion Internet est fournie par l’opérateur TOGOCOM via la liaison ADSL. Cette connexion à internet donne accès au serveur hébergé en ligne. Ce serveur est destiné au déploiement des projets de l’entreprise à savoir les sites web, les applications…

* 1. **Parc informatique**

L’entreprise dispose de deux ordinateurs, la première machine de marque HP destinée aux travaux du secrétaire et la seconde machine destinée au comptable. Le reste de l’équipe dispose d’ordinateurs portables personnel.

**5.3 Architecture logicielle des applications Web**

Les applications Web conçus par l’entreprise BANKEVI GROUPE sont basées sur une architecture monolithique qui consiste à regrouper ensemble toutes les fonctionnalités et dépendances des applications dans une grande couche.

1. **Critique de l’existant**

La section précédente a permis de décrire le système existant. Toutefois, cette solution présente des insuffisances que cette nouvelle section apporte sous forme de critique constructive.

Avec l’approche monolithique adoptée par l’entreprise BANKEVI GROUPE pour la conception de ses application Web, il en résulte quelques problèmes:

* Au fur et à mesure que de plus en plus de fonctionnalités sont ajoutés dans les applications, leurs complexités et leurs tailles sont augmentées et cela ralentit le développement, et la vitesse à laquelleon peut tester et redéployer.
* Apporter des modifications peut parfois être très complexe et réclamer des semaines, voire des mois de travail pour s’assurer que la nouvelle fonctionnalité que l’on souhaite ajouter s’intègrecorrectement avec l’existant et ne risque pas de créer d’incidents sur d’autres composantes de l’application.
* Une application basée sur cette approche utilise une seule base de données. Un incident au niveau de cette base de données peut rendre tous ses services indisponibles.
* Si une des dépendances de l’application est indisponible toute l’application peut se retrouver très vite hors service.
* Si un service est plus sollicité que les autres on se retrouve dans l’incapacité de multiplier seulement ce service. Pour satisfaire ce besoin accrue vers ce seul service l’on est obligé de multiplier et déployer à nouveau toute l’application, ce qui crée des ressources inutilisées dans l’application.
* La localisation de la source en cas de problème n’est pas du tout évidente.
* Cette approche oblige l’entreprise à préférer la stabilité à l’innovation de leurs codes.

Ces quelques problèmes ont un impact assez négatif sur le rendement de l’entreprise et ceci augmente considérablement le temps de création des applications Web. Face à ces problèmes résultant de l’approche monolithique il devient plus que nécessaire de s’orienter vers une nouvelle architecture répondant au mieux ces problèmes précités.

1. **Proposition et choix de solution**

**7.1 Spécification de la solution**

Pour permettre aux développeurs de livrer plus rapidement de nouveaux services aux utilisateurs, il est indispensable de rendre les dépendances des applications Web plus autonomes avec la possibilité de les déployées sur des serveurs différents et mettre en place un écosystème prêt à l’emploi dans lequel chaque application Web pourra fonctionner. Ainsi les dévéloppeurs n’auront plus à installer un tas de dépendances avant de commencer le développement de l’application ; ils n’auront qu’a instancier l’écosystème déjà en place et commencer le développement de l’application. La seule solution qui s’impose est de faire évoluer le système existant pour répondre aux exigences que nous nous étions fixées. La solution que nous adopterons sera une architecture microservices avec la technologie Docker, Il s’agit d’une approche pour développer une application composée de plusieurs petits services, chaque service constitue une application propre à elle-même et indépendant des autres, ces services ont chacun leurs processus et utilisent des mécanismes de communication légers, c’est une sous-partie d’une application qui ne va rendre qu’un seul service. Chacun de ces services doit pouvoir exécuter une partie unique et spécifique de l’application, et être déployer indépendamment des autres services. Il est important de noter que ces services peuvent être écrits dans des langages informatiques différents, et utiliser des systèmes de persistance qui le sont tout autant. Les différents microservices constituant la plateforme globale communiquent ensemble via des interfaces de programmation (API). De cette façon, les développeurs n’ont à se soucier que d’un périmètre restreint et non plus de l’application entière. Cela accélère toute la chaîne (développement, tests, mise en production).

**7.2 Evaluation financière de la solution**

La mise en place des microservices avec Docker ne nécessite pas l’achat d’une licence mais nécessite un investissement en jours hommes et en matériels informatique. Le projet se fera sur une durée de 10 semaines.

1. **Planning prévisionnel de réalisation**

Un résumé du planning prévisionnel de réalisation est établi dans le tableau si dessous :

Tableau 1 : Résumé du planning prévisionnel de réalisation

|  |  |  |
| --- | --- | --- |
| Activités | Date début | Date fin |
| **Période d’insertion (cadre du stage)** | 01 juillet 2019 | 05 Juillet 2019 |
| **Etude des enjeux liés aux microservices et leurs écosystèmes** | 08 Juillet 2019 | 12 Juillet 2019 |
| * **Etude des solutions pour la mise en place des microservices.** * **Etude des solutions pour la mise en place des écosystèmes microservices** * **Choix et étude des solutions les plus appropriées** | 15 Juillet 2019 | 19 Juillet 2019 |
| **Critique de l’existant** | 22 Juillet 2019 | 26 Juillet 2019 |
| **Elaboration du Cahier des charges** | 29 Juillet 2019 | 02 Aout 2019 |
| **Présentation et Choix de solution** | 05 Aout 2019 | 09 Aout 2019 |
| **Mise en place de l'ecosystème des microservices dans un environnement virtuel et intégration de l'application Web** | 12 Aout 2019 | 16 Aout 2019 |
| **Premier test de fonctionnement et de la stabilité des microservices par le maitre de stage** | 19 Aout 2019 | 23 Aout 2019 |
| **Correction des erreurs éventuelles** | 26 Aout 2019 | 30 Aout 2019 |
| **Second test de fonctionnement et de la stabilité des microservices par le maitre de stage** | 02 Aout 2018 | 06 Aout 2019 |

Tableau 1 :Planning prévisionnelle pour la réalisation du projet

**Conclusion**

Ce second chapitre constitue une étape primordiale pour fixer les repères du projet. Après avoir présenté l’organisme d’accueil dans et avoir reconnu ses attentes du projet, nous avons déterminé le cadre du travail ainsi que la méthodologie à emprunter lors de ce stage. Dans le prochain chapitre nous décrirons les caractéristiques de l’architecture microservices, la technologie Docker ainsi que les concepts liés.

**Chapitre**

**3**

**:**

**Les microservices et**

**la technologie Docker**

**1 L’architecture microservices**

**1.1 Définition des microservices**

Le terme "Microservice" a connu une émergence au cours des dernières années pour décrire un style d’architecture bien particulier. Cette approche consiste à développer une seule application en un ensemble de petits services, isolés, autonomes et indépendamment déployés. Ces services peuvent communiquer ensemble afin de fournir les fonctionnalités nécessaires. Les microservices sont, dans la plupart du temps, implémentés et pilotés par des équipes de petite taille avec suffisamment d’autonomie. Chacune peut changer l’implémentation de chaque microservice, ajouter ou supprimer des fonctionnalités de ce service avec un impact minimal sur les autres microservices. [8] Ce style d’architecture présente plusieurs avantages comme l’hétérogénéité technologique, la résistance contre l’échec, la scalabilité sur mesure, la facilité de déploiement, l’alignement organisationnel, la réutilisabilité.

**1.2 Caractéristique des microservices**

D’après Martin FOWLER [3], l’architecture microservices possède neuf principales  
caractéristiques qu’il est essentiel d’appliquer durant la conception et le développement  
d’une application en microservices.

**1.2.1 La division en composants via des services**

Cette caractéristique est héritée de l’architecture à base de composants. Les microservices sont indépendamment développés, testés et déployés. Un changement dans un service ne nécessite que son déploiement et n’affecte pas l’intégrité du système. Les services permettent d’éviter le couplage fort entre les différents composants en utilisant des mécanismes d’appel distants explicites.

**1.2.2 L’organisation autour des capacités métiers**

La division d’une application en microservices est très différente de la décomposition classique qui est souvent basée sur les couches techniques. Chaque microservice possède son propre code, sa propre interface et gère ses propres données.

**1.2.3 Un produit, pas un projet**

Le but de l’utilisation des microservices est de livrer rapidement un morceau de logiciel qui est alors considéré comme terminé. Dans la vision microservices, une équipe est responsable d’un produit durant tout son cycle de vie. Elle est entièrement responsable du logiciel en production.

**1.2.4 Une gouvernance décentralisée**

En effet, il est difficile de trouver une seule technologie permettant de résoudre tous les problèmes d’une façon efficace. D’où, il est préférable d’utiliser le bon outil au bon moment. Avec l’architectures microservices, nous pouvons utiliser pour chaque service le langage d’implémentation et la plateforme technologique les plus adéquats pour accomplir le besoin.

**1.2.5 Gestion de données décentralisée**

L’architecture en microservices admet l’utilisation de plusieurs bases de données. Dans le cadre d’une application monolithique, nous n’avons qu’une seule base données logique pour les entités persistantes alors que le cadre d’une application en microservices,chaque service a son propre modèle conceptuel et gère sa propre base de données.

**1.2.6 Les extrémités intelligentes et les canaux stupides**

Plusieurs entreprises s’investissent dans les canaux de communication intelligents entre les services, alors qu’avec les microservices, l’utilisation de communications stupides est favorisée. Ces communications non intelligentes ne font que transmettre les messages, alors que le microservice se charge du reste. L’intercommunication entre les microservices via des protocoles ouverts est privilégiés mais beaucoup d’autres interagissent les uns avec les autres via des appels REST ou à travers des systèmes de file d’attente.

**1.2.7 Automatisation de l’infrastructure**

Les techniques d’automatisation de l’infrastructure ont connu une évolution considérable ces dernières années. L’évolution du cloud a réduit la complexité opérationnelle de la construction, du déploiement et de l’exploitation de microservices. Les entreprises, qui ont migré vers l’architecture microservices, ont gagné de l’expérience dans la livraison continue et l’intégration continue et elles utilisent maintenant des outils d’automatisation de l’infrastructure

**1.2.8 Conception pour l’échec**

L’un des atouts majeurs des microservices est qu’ils sont conçus pour être tolérantsaux pannes. Dans une application en microservices, si un service échoue, les autres servicesne sont pas affectés et adaptent leurs fonctionnements selon l’état du système dans lequelils évoluent.

**1.2.9 Une conception évolutive**

L’un des éléments déterminants dans l’architecture en microservices, est la notion d’indépendance et d’évolutivité. En général, l’évolution d’une application consiste à l’ajout de nouvelles fonctionnalités qui se traduit par la création de nouveaux microservices et ou par la mise à jour des services existants qui implique seulement la mise à jour et le redéploiement du microservice concerné.

**1.3 Les concepts liés à l’architecture microservices**

Dans cette section, nous exposerons les concepts liés à l’architecture microservices  
à savoir la conception pilotée par le domaine, le développement à base de composants ainsi que le polyglotte de persistance.

**1.3.1 La conception pilotée par le domaine**

La conception pilotée par le domaine, dite en anglais domain driven design (DDD), est une méthodologie, une façon de concevoir une application, il n'y a absolument rien de technique dedans. La méthode se base sur la création d'un langage commun utilisé entre les différents membres d’un projet. Ce langage commun permet ainsi de lever toutes ambiguïtés dans les conversations. Le gros avantage de cette façon de travailler est que les développeurs, chefs de projet et les personnes concernées utilisent les mêmes mots pour s'exprimer et désigner un concept précis. La communication est alors améliorée puisque tout le monde discute de la même   
évidence les différents contextes bornés et leurs liaisons. Bien entendu, ce langage (dit omniprésent) peut être amené à évoluer, soit parce que le besoin métier évolue lui-même, soit parce qu'un nouveau concept qui n'avait pas été détecté apparaît. Un projet utilisant la conception pilotée par le domaine est donc voué constamment à évoluer et à subir une refactorisation continue. Afin de garantir l'évolutivité et la maintenance de l'application, il sera nécessaire d'implémenter de nombreux "design pattern" afin d'être le plus flexible possible. Il aura également la création de nombreuses classes afin de représenter les différents objets et éléments du domaine de la manière la plus explicite possible. Cela entrainera une certaine complexité technique qui freine certains développeurs à utiliser cette méthode. [9]

**1.3.2 Développement à base de composants**

Le développement à base de composant est une branche de l’ingénierie logicielle qui met l’accent sur la séparation des préoccupations à l’égard des vastes fonctionnalités disponibles à travers un système de logiciel. C’est une approche basée sur la réutilisation et la redéfinition. Cependant, un système est un ensemble de préoccupations fonctionnelles et extra fonctionnelles. Les préoccupations fonctionnelles sont les fonctionnalités métiers que le système doit assurer, alors que les préoccupations extra-fonctionnelles sont des services dont le système a besoin pour effectuer ses fonctionnalités métiers. [5] Cette situation augmente la complexité, empêche la réutilisation et gêne l’évolution des systèmes. La séparation avancée des préoccupations permet de séparer les parties extra-fonctionnelles des parties fonctionnelles d’un système. L’objectif escompté étant d’offrir une meilleure réutilisation, faciliter la maintenance, réduire la complexité des systèmes et augmenter leur évolutivité.

**1.3.3 Persistance polyglotte**

Le terme persistance polyglotte (Polyglot Persistence en anglais) [6] a été introduit par *Scott Leberknight*. Il tire son nom de la programmation polyglotte, qui favorise la coexistence de différents langages dans une même application, en tirant profit des forces de chacun d’entre eux. L’un des atouts majeurs de l’architecture microservices, est que chaque microservice peut être écrit dans un langage de programmation qui lui est propre donnant plus de rapidité et de performance. En raison de l’isolement et de l’indépendance des microservices, les services individuels peuvent être polyglottes en termes de langage de programmation.

**1.3.4 Bounded context**

Les contextes bornés donnent une solution efficace pour définir les frontières des différents microservices. L’approche “Bounded Context” est intéressante pour les microservices car:

* Les échanges sont plus nombreux entre services dans un même domaine fonctionnel.
* Elle évite d’exposer trop d’interfaces au-delà du domaine.

**1.4 Architecture monolithique et microservices**

Une architecture monolithique représente le modèle traditionnel unifié de conception d'un programme informatique. Ce contexte, « monolithique » signifie formé d'un seul bloc. Un logiciel monolithique est conçu pour être autonome ; ses composants sont interconnectés et interdépendants plutôt qu'associés de manière flexible comme dans le cas des microservices. Dans ce type d'architecture étroitement intégrée, chaque composant et ceux qui lui sont associés doivent être présents pour permettre l'exécution ou la compilation du code. Pour mettre à jour un composant du programme, il faut réécrire toute l'application, tandis que dans une application modulaire, chaque microservice peut être modifié sans que cela se répercute sur les autres parties du programme. Dans une architecture modulaire, une modification apportée à un élément a moins de risque d'entraîner des changements non prévus au sein d'autres éléments, puisque les modules sont relativement indépendants. Les programmes modulaires se prêtent aussi plus facilement à des processus itératifs que les programmes monolithiques. Cependant, les architectures monolithiques comportent également des avantages. En général, elles offrent un meilleur rendement que l’architecture de microservices, et peuvent être plus faciles à tester et à déboguer, car avec moins d'éléments, le nombre de variables à prendre en compte est moins élevé.[10]

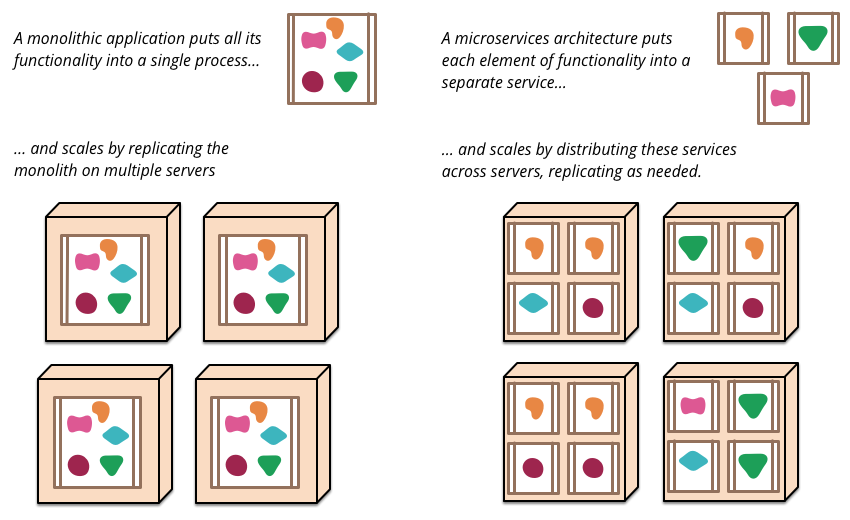


Figure : Comparaison entre architecture monolithique et microservices

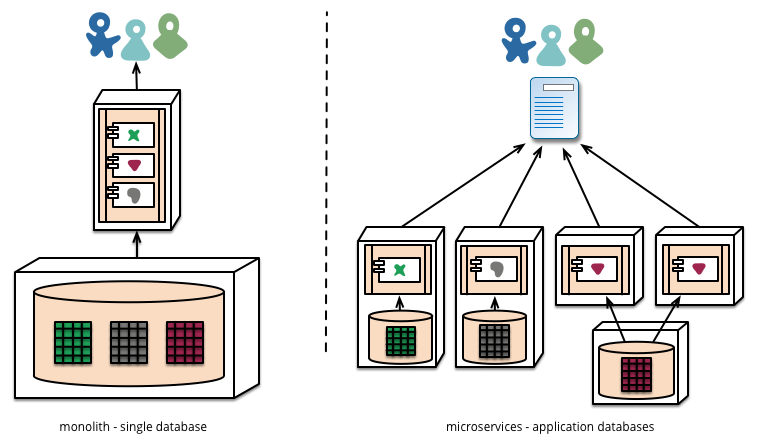


Figure : La gestion de la base de données dans une architecture monolithique et dans une architecture microservices.

**1.5 Architecture orienté service (SOA) et microsevices**

L'approche SOA est similaire aux [microservices](https://www.lemagit.fr/definition/Microservices) ; tout comme ces derniers sont semblables à une architecture logicielle *MVC* (Model-View Controller). Les idées sont les mêmes : fragmenter les applications en composants plus petits, avec une empreinte plus réduite et plus faciles à gérer. Toutefois, la plus grande différence entre ces technologies tient à l'objet et au timing de leurs introductions respectives. [L'architecture SOA est arrivée la première](https://www.lemagit.fr/conseil/Methodes-dintegration-avantages-et-inconvenients-de-REST-avec-SOA), puis vint l'architecture MVC et enfin, les microservices.

L'approche SOA (Service Oriented Architecture) est un terme purement logiciel. Son objectif consiste à faire appliquer un schéma de conception plus efficace en matière de développement d'applications. Mais la couverture s'arrête là. En effet, cette approche n'affecte pas l'infrastructure. Ce point est important, car les applications sont devenues si complexes qu'elles occupent l'intégralité de la pile, voire des datacenters entiers. L'idée des microservices consiste à étendre la mise en oeuvre de ces plus petites unités et techniques d'isolement à l'infrastructure, ainsi qu'à la conception des applications.

Les microservices sont autant une architecture d'infrastructure qu'une architecture logicielle. En fait, on pourrait supposer que le code présent dans le service est élaboré [selon des principes de type SOA](https://www.lemagit.fr/conseil/Les-architectures-orientees-services-SOA-peuvent-facilement-migrer-vers-le-cloud). Toutefois, dans le cas de microservices, l'implication tient à ce que l'infrastructure sur laquelle chaque service s'exécute soit indépendante des autres. Ainsi son autonomie est préservée sur la couche serveur autant qu’au niveau du code. Ce procédé implique également que la répartition des charges peut intervenir service par service, ce qui offre aux équipes davantage de possibilités pour optimiser les performances, comparé à la SOA. En d'autres termes, accroître la capacité pour un service revient à lancer de manière indépendante de nouvelles instances de ce service, et ainsi à offrir une évolutivité accélérée. Cela entraîne en outre une résolution plus rapide des problèmes, car dans un tel modèle, il est très probable que des services puissent s'arrêter sans entraîner l'immobilisation de toute l'application.

Mais il existe aussi d'autres points importants à prendre en compte. La gestion induite supplémentaire que requièrent les microservices par rapport à une technologie SOA est considérable. En effet, les microservices nécessitent une surveillance plus robuste, une automatisation des versions plus efficace et des équipes de développement et de gestion informatique plus disciplinées. L'approche SOA peut constituer une pratique à suivre pour le développeur, là où les microservices affecteront l'intégralité du pipeline.[11]

Cependant l’approche SOA est considérée par beaucoup comme un échec par rapport aux microservices car:

* Trop théorique: beaucoup d’architectes ont écrit des articles pour décrire cette approche sans forcément donner des indications pratiques sur la façon de casser un monolithe et d’avoir une implémentation évolutive d’un service. Pas de prise en compte des difficultés opérationnelles: l’approche trop théorique n’a pas apporté de solutions à des problématiques opérationnelles comme le déploiement, la scalabilité, le monitoring. Les implémentations de services peuvent se heurter à des difficultés à assurer des problématiques qui se résolvent plus facilement avec une approche en monolithe.
* Protocoles de communication difficiles à utiliser: l’architecture en services a, parfois été vendue par des éditeurs de middlewares qui proposaient des solutions de communications souvent couteuses et propriétaires. Ces solutions avaient une empreinte forte sur l’implémentation des services ce qui couplaient les services au middleware.
* Choix d’architecture peu évolutive et contraignante: l’utilisation de ces middlewares peut aussi rendre l’architecture en services peu évolutive et contraignante car très dépendantes des middlewares.[12]

A la différence de l’approche SOA, l’approche microservices a émergé dans le but de répondre plus facilement à des problématiques opérationnelles. L’architecture en microservices énonce des principes d’architectures en indiquant des solutions possibles et pratiques pour la plupart des problématiques.

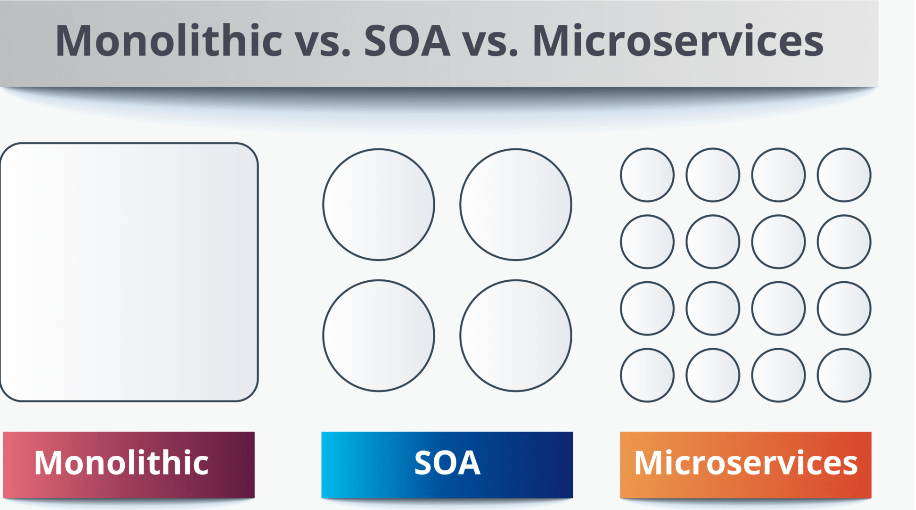


Figure : Comparaison entre architecture monolithique, SOA et microservices

**1.6 Principes des microservices**

D’une façon générale, la conception de microservices doit assurer:

* Un faible couplage: de façon à permettre de modifier les services indépendamment et d’assurer une autonomie dans leur fonctionnement. Des services faiblement couplés permettront de tirer parti au maximum de l’architecture en microservices: tolérance aux pannes, s’adapter à la charge, faciliter les déploiements.
* Grande cohésion: assurer une cohésion entre les services vise à rendre les échanges entre ces services de façon la plus cohérente possible en:
* Utilisant des interfaces claires avec des types précis: par exemple, il faut éviter d’utiliser des types comme objet pas assez précis qui laissent trop de libertés quant au type des objets. De même, il faut éviter de définir des fonctions qui ont plusieurs objectifs, il est préférable de limiter une fonction à un seul cas d’utilisation.
* Eviter les choix technologiques trop exotiques dans les communications entre services: par exemple, il faut éviter d’utiliser des middlewares qui sont généralement couteux en licence et peuvent avoir une empreinte forte dans l’implémentation des services.
* Eviter les breaking changes: limiter les breaking changes lors des évolutions des services. Des breaking changes dans les interfaces d’un service nécessitent la modification des services qui y font appel. Ces breaking changes peuvent compliquer les déploiements.
* Ne pas exposer des détails d’implémentation internes d’un service: exposer des détails de l’implémentation interne d’un service peut donner des indices sur son fonctionnement. D’autres services peuvent involontairement tirer parti de ce fonctionnement et avoir une implémentation dépendant de ce fonctionnement. Des implémentations trop dépendantes rendent le couplage plus important entre les services.[12]

## 1.7 Types de communication en microservices

Un client et des services peuvent communiquer via de nombreux types différents de communication, chacun d’eux ciblant un scénario et des objectifs différents. Au départ, ces types de communication peuvent être classés selon deux axes.

* Le premier axe définit si le protocole est synchrone ou asynchrone :
* Protocole synchrone. HTTP est un protocole synchrone. Le client envoie une requête et attend une réponse du service. Cela est indépendant de l’exécution du code client, qui peut être synchrone ou asynchrone. Le point important ici est que le protocole (HTTP/HTTPS) est synchrone et que le code client peut continuer sa tâche seulement quand il reçoit la réponse du serveur HTTP.
* Protocole asynchrone. D’autres protocoles, comme AMQP (un protocole pris en charge par de nombreux systèmes d’exploitation et environnements cloud), utilisent des messages asynchrones. Le code client ou l’expéditeur du message n’attend généralement pas de réponse. Il envoie simplement le message comme lors de l’envoi d’un message à une file d’attente RabbitMQ ou à tout autre service broker de messages.
* Le deuxième axe définit si la communication a un ou plusieurs destinataires :
* Destinataire unique. Chaque demande doit être traitée par exactement un récepteur ou un service. Le [modèle Commande](https://en.wikipedia.org/wiki/Command_pattern) est un exemple de cette communication.
* Plusieurs destinataires. Chaque demande peut être traitée par zéro à plusieurs destinataires. Ce type de communication doit être asynchrone. Le mécanisme de [publication/abonnement](https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern) utilisé dans des modèles comme [Event-driven architecture](https://microservices.io/patterns/data/event-driven-architecture.html) en est un exemple. Il s’appuie sur une interface de bus d’événements ou un répartiteur de messages durant la propagation des mises à jour des données entre plusieurs microservices via des événements. Il est généralement implémenté via un bus de service ou un artefact similaire, par exemple [Azure Service Bus](https://azure.microsoft.com/services/service-bus/), à l’aide des [rubriques et abonnements](https://docs.microsoft.com/azure/service-bus-messaging/service-bus-dotnet-how-to-use-topics-subscriptions).

Une application basée sur des microservices utilise souvent une combinaison de ces styles de communication. Le type le plus courant est une communication avec un seul destinataire, avec un protocole synchrone comme HTTP/HTTPS lors de l’appel d’un service web HTTP d’API ordinaire. Les microservices utilisent aussi en général des protocoles de messagerie pour la communication asynchrone entre microservices.[13]

**1.8 Intégration des microservices**

Le point important lors de la création d’une application basée sur des microservices est la façon dont on intègre les microservices. Dans l’idéal, il est conseillé de réduire la communication entre les microservices internes. Moins il existe de communications entre les microservices, mieux c’est. Mais dans de nombreux cas, on est souvent obligé d’intégrer les microservices d’une façon ou d’une autre. Pour ce faire, la règle essentielle est que la communication entre les microservices doit être asynchrone. Cela ne signifie pas qu’on utiliser un protocole spécifique (par exemple une messagerie asynchrone par opposition au protocole HTTP synchrone). Cela signifie simplement qu’on doit devez effectuer la communication entre microservices par propagation asynchrone des données, tout en essayant de ne pas dépendre d’autres microservices internes dans le cadre de l’opération de requête-réponse HTTP du service initial.

Si possible, ne jamais dépendre d’une communication synchrone (requête/réponse) entre plusieurs microservices, pas même pour des requêtes. L’objectif de chaque microservice est d’être autonome et disponible pour le consommateur client, même si les autres services qui font partie de l’application de bout en bout sont indisponibles ou défectueux.[13]

**1.9 Bilan sur l’architecture microservices**

Après une étude approfondie des caractéristiques et les concepts liés à l’architecture  
microservices, nous exposerons, dans cette partie, un bilan sur l’architecture microservices.

**1.9.1 Motivations de l’utilisation des microservices**

Plusieurs motivations poussent à utiliser ce style architectural :

* + La frustration de ne pas obtenir le résultat souhaité avec les architectures monolithiques dans les grands projets nous encourage à affronter les défis des microservices.
  + L’émergence de nouveaux outils d’automatisation de test, de monitoring et de déploiement nous permettent de voir les microservices sous un nouveau jour et suppriment une partie des problèmes que leur développement et leur déploiement avaient créés.
  + L’utilisation des microservices par des grandes entreprises comme Amazon, Netflix, eBay et d’autres, donne assez de confiance pour que ce style d’architecture soit prêt à être évalué et utilisé par les développeurs d’applications professionnelles.

**1.9.2 Avantages des architectures microservices**

* Elle aborde le problème de la complexité.

L'architecture microservices décompose ce qui serait autrement une application monolithique lourde en un ensemble de services gérables dont les limites respectives sont bien définies sous forme de RPC ou API à base de messages. Avec un tel niveau de modularité, les services individuels peuvent être développés plus rapidement et ils sont beaucoup plus faciles à comprendre et à maintenir.

* Elle permet aux entreprises d'optimiser les ressources consacrées au développement et aux applications.

L'architecture microservices permet à chaque service d'être développé de manière indépendante par une équipe dédiée. Elle permet aux développeurs de prendre des décisions adéquates, spécifiques aux services.

Les développeurs ne sont plus tenus d'utiliser les technologies potentiellement obsolètes qui existaient lors du lancement d'un nouveau projet. La taille relativement petite des services permet également aux développeurs de réécrire les anciens services à l'aide de technologies plus récentes et plus pertinentes.

* Elle rend le déploiement continu possible.

Étant donné que chaque microservice peut être déployé de manière indépendante, les développeurs ne doivent pas coordonner le déploiement des modifications locales spécifiques à leur service. Les mises à jour et nouvelles fonctionnalités peuvent être déployées plus rapidement et facilement, rendant le déploiement continu possible là où il ne l'était pas auparavant.

* Elle optimise le dimensionnement.

Le dimensionnement est souvent le point faible des applications monolithiques, les empêchant de répondre efficacement aux besoins de l'entreprise. Les microservices sont une manière d'isoler des fonctions parfois saturées et de les construire de manière horizontale. Les organismes peuvent également déployer uniquement le nombre d'instances dont ils ont besoin dans chaque service, mobilisant le matériel qui répond le mieux aux besoins en ressources de chaque service.

**1.9.3 Inconvénients des architectures microservices**

* Le terme « micro » est relatif.

L'objectif de l'architecture microservices est de décomposer les applications en blocs gérables, mais les blocs doivent également être de taille raisonnable. Si les services sont trop grands ou trop petits, les applications créées à l'aide de ces derniers auront des problèmes. Un équilibre parfait peut être difficile à trouver, surtout pour des développeurs novices dans ce type l'architecture.

* Une architecture fondée sur des bases de données cloisonnées pose des problèmes de mise à jour.

Les transactions commerciales mettant à jour les données de plusieurs entreprises à la fois sont assez courantes.

Ce type de transaction est simple à mettre en œuvre dans une application monolithique car il n'y a qu'une seule base de données. Toutefois, dans une application basée sur les microservices, différentes bases de données appartenant à différents services devront être mises à jour.

* Une architecture distribuée nécessite un déploiement, un dimensionnement et des tests approfondis et ciblés.

L'un des principaux avantages de l'architecture microservices, son caractère distribué, constitue également un défi considérable en matière de test, déploiement et dimensionnement des applications.

* Les changements peuvent être compliqués.

Les dépendances entre les services peuvent être source de problèmes lors de la mise en place de changements. Les développeurs doivent soigneusement planifier et coordonner le déploiement des changements pour chacun des services.

* Une architecture d'applications distribuées engendre des responsabilités distribuées et nouvelles.

Bien que l’on puisse affirmer que les avantages des microservices l'emportent sur leurs inconvénients, on ne peut nier l’enjeu considérable que constituent ses inconvénients.

Certaines entreprises tentent de décomposer leurs applications monolithiques à l'aide de codes faits maison. Cela peut fonctionner de prime abord. Au fil du temps, cependant, à mesure que le nombre de microservices, les dépendances et la complexité augmentent et que de plus en plus de développeurs sont impliqués les choses peuvent rapidement devenir très difficiles à gérer. Les entreprises doivent envisager la mise en place de la découverte de services, de la répartition de charge, de la mise en cache et du routage de la couche 7.

* L'exploitation d'une infrastructure distribuée dans un environnement automatisé entraîne de nouvelles pratiques et une nouvelle culture

Déterminer les limites entre les services et les parties qui en sont responsables peut prendre un certain temps et nécessiter des changements culturels non négligeables. Il peut être judicieux de démarrer avec une petite équipe dotée d'expériences diverses, couvrant différents domaines, depuis les langages hérités sur lesquels le monolithe a été construit jusqu’à la technologie de conteneur Linux.

En outre, la sécurité peut être menacée quand un nombre croissant de microservices entre en jeu. Une vulnérabilité au sein d'un microservice peut mettre toute une application en péril, et la nature diverse des microservices peut les rendre difficiles à sécuriser. Il est important d'envisager la mise en œuvre d'un chiffrement de bout-en-bout afin d’assurer que le maillon le plus faible de la chaîne ne peut pas causer de dommage s’il est défaillant.[14]

**1.10 Conteneurisation**

**1.10.1 Définition**

Les conteneurs et les microservices sont étroitement liés. Tout comme dans le domaine des transports, les conteneurs informatiques stockent des objets pour les transporter. Ils permettent d’expédier des applications et leurs dépendances sur de multiples systèmes d’exploitation, quels qu’ils soient. Ils garantissent que leur contenu est identique au départ et à l’arrivée, et qu’il est sécurisé, grâce à leur mise en isolation.[15]

**1.10.2 Rôle des conteneurs**

Ils servent à minimiser la complexité liée à la configuration et à l’administration applicatives, à accélérer les cycles de développement et de production applicatifs, et, grâce à leur flexibilité et à leur portabilité, ils constituent l’une des briques qui permettent de faire de l’« infrastructure as a service », c’est-à-dire d’automatiser les infrastructures IT.

## 1.10.3 Le principe de fonctionnement de la conteneurisation

La conteneurisation, à ne pas confondre avec la virtualisation est une méthode permettant d’exécuter une application dans un environnement virtuel dans une seule zone appelée Conteneur. Seul l’environnement d’exécution est virtualisé dans un conteneur (systèmes de fichiers, réseau, processeur, mémoire vive, …). L’application exécutée dans cet environnement virtuel stocke tous les fichiers, bibliothèques et librairies dans le conteneur.

Ensuite notre conteneur se connecte au noyau (kernel) d’un système d’exploitation. Il n’est donc pas nécessaire comme pour les machines virtuelles, d’installer un nouveau système d’exploitation. Ici le noyau gère les ressources de l’ordinateur et permet au différents composants matériels et logiciels de communiquer entre eux. La conteneurisation rend donc le déplacement d’application virtuelle plus simple entre des systèmes d’exploitation identiques et demande moins de ressources de mémoire, de RAM, de CPU, etc.[16]

**1.10.4 Différence entre la virtualisation et la conteneurisation**

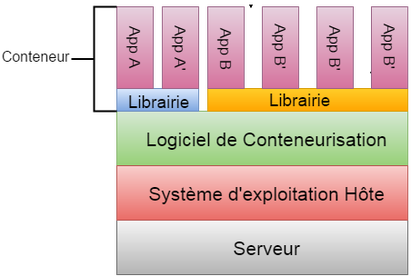
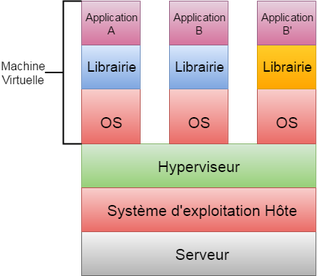


Figure : Différence entre la virtualisation et la conteneurisation

Ces deux schémas si dessus, d’un environnement de virtualisation à gauche et d’un environnement de conteneurisation à droite permettent de bien identifier les différences entre ces deux technologies. Tout d’abord, les conteneurs partagent un seul et unique système d’exploitation, de ce fait, l’échange de données entre les conteneurs est plus simple et plus rapides que pour les VM. De plus comme chaque conteneur ne contient pas de système d’exploitation propre à lui, les conteneurs sont donc réduits et prennent moins de place et moins de ressource Serveur environ 10 fois plus petit qu’un VM. [16] Le temps de création et de suppression d’un conteneur est par la même occasion réduit.

Les conteneurs facilitent l’évolution technique, par exemple si dans un environnement de VM, l’on souhaite faire évoluer les OS de plusieurs VM, il faut le faire manuellement sur chaque Machine. Ce problème n’est pas présent pour la conteneurisation car toute l’infrastructure repose sur un seul Système d’exploitation.

Mais la conteneurisation peut avoir quelques inconvénients, comme tous les conteneurs ne reposent que sur un seul système d’exploitation, la diversification des systèmes d’exploitation n’est pas possible avec la conteneurisation, « où est plus compliquée à mettre en place ». Les conteneurs sont isolés pour assurer la sécurité et empêcher les malwares de se transmettre entre les conteneurs, mais il est évident que les machines virtuelles seront toujours plus isolées que les conteneurs. Même si les conteneurs ont un grand nombre d’avantages, leur apparition ne sonne pas la fin de la virtualisation. Aujourd’hui, les machines virtuelles sont intégrées dans de nombreuses entreprises comportant des réseaux de grande taille. Pour utiliser entièrement la technologie de conteneurisation, ces entreprises devrait remanier tous leurs systèmes informatiques, ce qui est impensable. Mais de nouvelles entreprises ont vu le potentiel de la conteneurisation et on donc créer leur système informatique en fonction.

**1.10.5 Avantages de la conteneurisation**

* **Légèreté**  
  Les conteneurs occupent moins d'espace sur le serveur que les ordinateurs virtuels et ne prennent généralement que quelques secondes à démarrer.
* **Élasticité**  
  Les conteneurs sont très élastiques et il n’est pas nécessaire d’allouer une quantité donnée de ressources. Cela signifie que les conteneurs sont en mesure de faire une utilisation plus efficace et dynamique des ressources du serveur. Lorsque la demande sur un conteneur diminue, les ressources supplémentaires sont libérées pour être utilisées par d'autres conteneurs.
* **Densité**  
  La densité fait référence au nombre d'objets qu'un seul serveur physique peut exécuter à la fois. La conteneurisation permet la création d’environnements denses dans lesquels les ressources du serveur hôte sont pleinement utilisées mais non surexploitées. Par rapport à la virtualisation traditionnelle, la conteneurisation permet des environnements plus denses car les conteneurs n'ont pas besoin d'héberger leur propre système d'exploitation.
* **Performance**  
  Lorsque la pression sur les ressources est importante, les performances des applications sont bien meilleures avec les conteneurs qu'avec les [hyperviseurs](https://www.alibabacloud.com/fr/knowledge/what-is-hypervisor). En effet, avec la virtualisation traditionnelle, le système d'exploitation invité doit également répondre à ses propres besoins en matière de mémoire, en retirant une RAM précieuse de l'hôte.
* **Efficacité de la maintenance**

Avec un seul noyau de système d'exploitation, les mises à jour ou les correctifs au niveau du système d'exploitation ne doivent être effectués qu'une seule fois pour que les modifications prennent effet dans tous les conteneurs. Cela rend l'exploitation et la maintenance des serveurs plus efficaces.[17]

**1.10.6 Inconvénients de la conteneurisation**

En matière de conteneurisation, les outils arrivent petit à petit à maturité. C’est-à-dire que la conteneurisation et le support sur lequel elle s’appuie commencent (à l’intérieur du noyau Linux) à être complètement fiables. On n’observe plus les failles qui permettaient, il y a encore quelques années, de briser les protections et de s’introduire à l’intérieur d’un conteneur pour accéder frauduleusement aux ressources de la machine. Aujourd’hui, ce n’est plus possible parce que le code utilisé est abouti et que les ops sont bien mieux formés pour limiter les possibilités d’intrusion. Par exemple en limitant les droits sur les applications.

Cependant, pour les orchestrateurs de conteneurs, il existe encore des limitations. Par exemple en matière de gestion de grands volumes de données, il reste des progrès à faire pour l’intégration avec les fournisseurs de cloud. Par ailleurs, de nombreuses applications à migrer vers le cloud ne peuvent pas bénéficier des avantages de l’orchestration des conteneurs pour le passage à l’échelle et la portabilité. Il faudrait les réécrire pour les adapter à cette nouvelle approche, mais cela demanderait trop de temps.

**1.10.7 Les applications utilisant les conteneurs**

Presque toutes les applications qui doivent être modifiées et redéployées rapidement et fréquemment conviennent parfaitement à la conteneurisation. Les applications utilisant une architecture de microservices constituent également un choix naturel.

## 1.10.8 Une standardisation de conteneurisation

## L’initiative d’une standardisation « l’Open Container Project » rassemble un grand nombre d’entreprises telle que *Docke*r, *CoreOS*, *Google,* *Microsoft*, *VMware*, *Intel, IBM, Cisco, Red Hat* et bien d’autres. Avec la domination du marché de la conteneurisation par Docker, ce modèle est rapidement devenu un standard, mais ces entreprises souhaitent aujourd’hui établir un standard ouvert et formel. Docker proposera son code source comme base mais cette standardisation reviendra à la Linux Fondation. Le but est que cette technologie ne soit pas attribuée à une seule entreprise mais que n’importe qui puissent y accéder. Malgré que ce ne soient pas la première tentative de standardisation, l’Open Container Project à l’avantage d’obtenir l’appui précieux de Docker et d’un impressionnant nombre d’acteurs. [16]

**2 La technologie Docker**

**2.1 Définition du Docker** :

Il s'agit d'un logiciel libre pour automatiser le déploiement d'applications dans des conteneurs logiciels. La firme de recherche sur l'industrie 451 Research a défini le Docker comme un outil qui permet d'empaqueter une application et ses dépendances dans un conteneur isolé, Ce logiciel pourra aussi être exécuté sur n'importe quel serveur Linux. Et par la suite, la portabilité d’exécution et la flexibilité d'une application s'étendent, que ce soit sur la machine locale, , une machine nue, un Cloud public ou privé etc. Le Docker est utilisé pour créer et gérer des conteneurs, ce qui permet la simplification de la mise en œuvre de systèmes distribués en permettant l'exécution de multiples applications, tâches de fond et autres processus. L’exécution sera de façon autonome sur une seule machine physique ou à travers un éventail de machines isolées. Ceci permet le déploiement des nœuds en tant que ressources sur besoin

La figure suivante montre l'architecture du Docker :

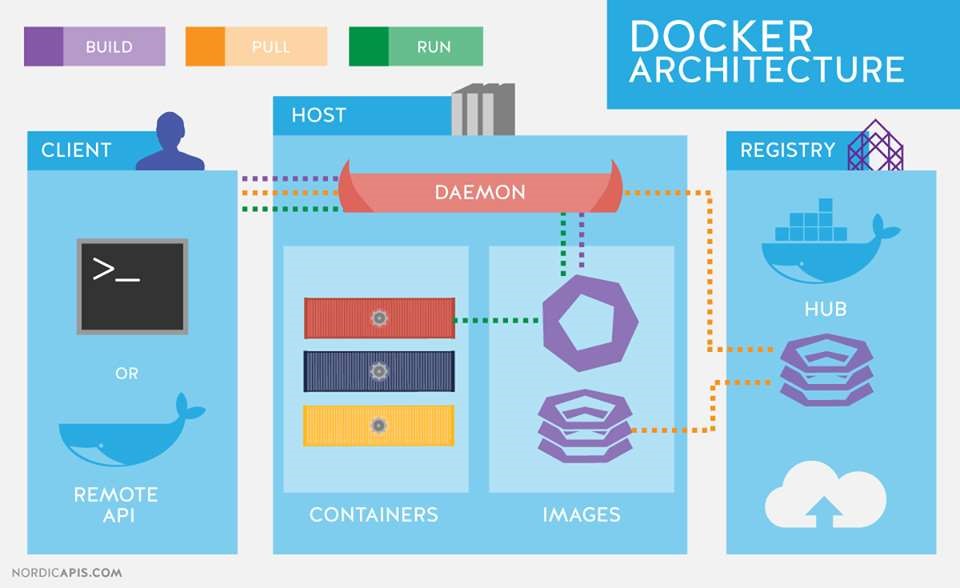


Figure 2 : Architecture du docker

Docker contient trois composants :

* Docker Daemon : Il a la fonction d’un root et il orchestre essentiellement tous les conteneurs
* Docker images : Il s’agit des images pour système d'exploitation virtuels, techniquement Docker images ont de manière moins d’empreinte que les images d'OS réels. Les systèmes de fichiers sont sauvegardés et empilés, et par la suite ils peuvent être unifiés à la dernière révision à tout moment. Toutes ces opérations d'écriture sont également mises en cache vu la nature du système des fichiers. À savoir, si Docker découvre qu'on ne dispose pas des changements jusqu'à une certaine révision, il joue le tout jusqu'à ce que la révision, notamment du cache en appliquant les écritures. Ceci permet un énorme avantage de performance sur le temps de provisionnement. En plus de tous ces avantages, on peut échanger les images avec d'autres personne.
* Docker repositories :
* C’est le composant écosystème du docker. Autrement dit c'est le référentielle. Les images doivent être sauvegardées quelque part, que ce soit privé ou public.

**2.2 Histoire de Docker**

Dans le cadre d’un projet interne de dotCloud, l'entreprise française Solomon Hykes a développé Docker. Cette entreprise propose une plate-forme en tant que service, avec les contributions d'Andrea Luzzardi et Francois-Xavier Bourlet. Docker est une évolution basée sur les technologies propriétaires de dot Cloud.

* Depuis Mars 2013, Docker a été distribué en tant que projet open source.

Au 18 novembre 2013, le projet a été favorisé plus de 7 300 fois sur GitHub (Il présente le 14e projet le plus populaire), avec plus de 900 forks et 200 contributeurs.

* Au 9 mai 2014, le projet a été favorisé plus de 11 000 fois sur GitHub, avec plus de 1900 forks et 420 contributeurs.
* En octobre 2015, le projet a été favorisé plus de 25 000 fois sur GitHub, avec plus de 6500 forks et 1 100 contributeurs.
* En septembre 2016, le projet a été favorisé plus de 34 000 fois sur GitHub, avec plus  
  de 10 000 forks et 1 400 contributeurs[2]

**2.3 Pourquoi utiliser Docker**

Le choix du Docker est justifié par:

* La distribution des applications est facilitée
* Les applications en Dev/Qualif/Prod ont le même Comportement.
* Rapide déploiement, lancement et arrêt.
* Linux et Windows (en preview dans Windows Server 2016)
* Reconstruire d'un container à partir d'un simple fichier"Docker file"
* Gérer des containers identiques sur toutes les plateformes en utilisant peu d'outils.
* Des API sont disponibles pour piloter l'ensemble depuis d'autres applications.

**2.4 Outils libres de Docker :**

* DOCKER MOTEUR/CLIENT : permet l'utilisation de docker en cli.
* DOCKER MACHINE : Permet le lancement de Docker, Il crée automatiquement un environnement virtuel.
* DOCKER COMPOSE : permet le lancement des applications multi-containers.
* DOCKER SWARM : permet la gestion des containers Docker dans un cluster.
* DOCKER REGISTRY : c'est une application pour gérer des images locales.
* NOTARY : Il s'agit des outils pour la signature des images par le fournisseur et la vérification de l'intégrité par le client.
* LIBNETWORK : c'est un outil d'abstraction réseau qui permet la communication et l'isolation entre les conteneurs, même sur des nœuds différents.

La figure ci-dessous montre la configuration le docker client :

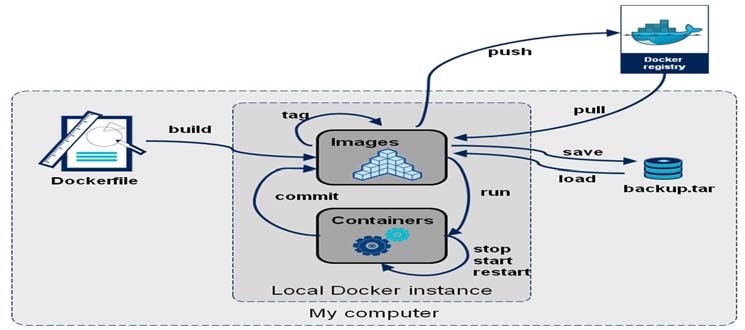


Figure 3 : Docker Moteur / client

**2.5 Caractéristiques de Docker**

Docker a plusieurs caractéristiques :

* Docker Combine et adapte des différents concepts et outils éprouvés (conteneurs, scripts, gestion de version, gestion de paquets) avec une approche DevOps
* Principal intérêt : utilisation simple et autosuffisance
* Logiciel libre écrit en Go
* Uniquement des conteneurs pour le noyau Linux, pour l'instant
* Disponibilité sous Linux, sous Windows et Mac via la Docker Toolbox fournissant une machine virtuelle Linux minimaliste, et sous de nombreuses infrastructures nébuleuses.

**2.6 Comparaison entre virtualisation classique et Docker:**

Pour la virtualisation, les hyperviseurs créent une copie complète du système d’exploitation. Cette copie fonctionne sur un espace matériel virtuel. L’hyperviseur sera donc responsable de tous les échanges de données. Exécuter plusieurs machines virtuelles sur un même serveur requiert de hautes performances et de ressources suffisantes pour assurer le fonctionnement de plusieurs machines virtuelles. (Chose difficile avec la virtualisation classique)

Tandis que les conteneurs permettent :

* Exécution avec une vitesse plus rapide probablement 2 à 3 secondes au lieu de minutes
* Utilisation de la mémoire du conteneur au lieu de disques
* Encombrement minimal - à la place de milliers de Mo, il est des dizaines de Mo
* De nombreuse exécution - Possibilité d'exécuter 100 conteneurs en parallèle sur une seule boîte.
* Résilience - En cas d’un accident, on peut relancer immédiatement
* Sécurité - Dans le cas où un conteneur à une attaque DDOS, les autres conteneurs ne seront pas affectés
* Sauvegarde - en utilisant de fichiers file-system

La différence de vitesse d'exécution entre la virtualisation et le Docker :

Virtualisation Docker



Déploiement dans des minutes Déploiement dans quelques secondes pendant un moins Pendant quelques heures/minutes

Agile Devops

Docker étend le format de conteneur Linux standard, LXC, avec une API de haut niveau proposant une solution de virtualisation qui effectue les processus de façon isolée. Docker fait usage de LXC, Cgroups, et le noyau Linux lui-même. Contrairement aux machines virtuelles traditionnelles, un conteneur Docker ne contient pas de système d'exploitation, se basant sur les fonctionnalités du système d’exploitation fournies par l'infrastructure sousjacente.

Le schéma ci-dessous compare le mode Docker au celui de virtualisation classique :

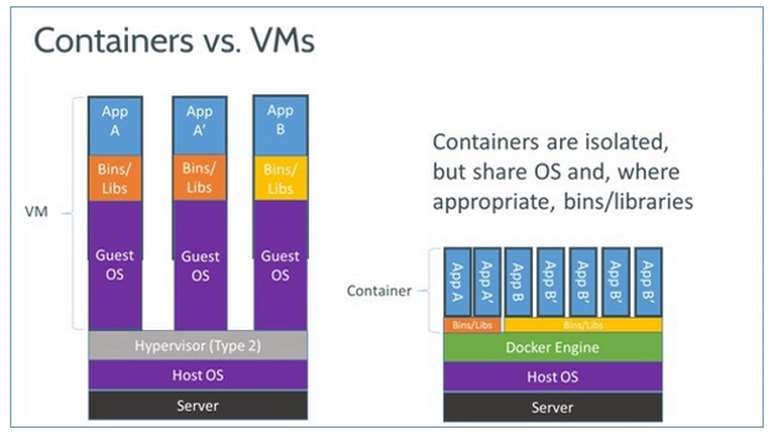


Figure 4 : comparaison au mode de virtualisation classique

**2.7 Les avantages et inconvénients de la solution de Docker :**

**2.7.1 Avantages**

Dans le domaine du Web ainsi que dans le domaine de l'IT, l’utilisation de Docker possède plusieurs avantages. Les services IT fournissent de plus en plus de services orientés web, les conteneurs pourraient être un élément clef permettant la transition des services IT vers le cloud.

* Légèreté des conteneurs

Par opposition à un serveur virtuel sous Linux, le conteneur n’a besoin que de quelques centaines Mo de disques. De même l’empreinte mémoire est réduite, car nous n’avons que de la mémoire employée pour l’application (pas de couche OS). Il est ainsi avec démarrage plus rapide aussi le conteneur peut être facilement déplacé d’une machine à une autre.

* Rapidité et facilité de déploiement des applications

Après avoir récupérer un template en une commande, et afin d'exécuter le conteneur il nous suffit une autre commande, celui-ci démarre en quelques secondes. Lors du démarrage, des paramètres peuvent être ajoutés en les transmettant au conteneur. Par exemple, la spécification de l’accès à la base de données ou d’autres paramètres d’automatisation. L’idéal consiste à mettre en place une gestion de configuration, grâce à elle, les conteneurs deviennent autonomes. Ces conteneurs seront gérés comme un package logiciel. Cette facilité aide le développeur à provisionner les environnements qui sont déployés en production par exemple sans connaissance particulière d’infrastructures ou d’administration système.

* Portabilité et multicloud

Il y a peu de formats d’enveloppes virtuelles qui soient nativement multi-Cloud. Il est toujours compliqué de récupérer sur sa ferme VMware son POC fait sur AWS, ou encore migrer sa machine virtuelle Open stack sur une Ressource Group Azure.

## 2.7.2 Inconvénients

## 2.8 Les Facteurs et les concepts :

## 2.8.1 Facteurs

## Code base

## Un logiciel de suivi de version (git, mercurial, ...) gère les codes.

## Une application = code source

## Dependencies

## Préciser clairement toutes les dépendances.

## Le système cible n'est pas censé inclure de programme pré-installé.

## Absence de dépendances implicites.

## Config

## On considère une configuration, tout ce qui diffère d'un environnement à l'autre (dev, qualif, prod, autre site).

## Les éléments de configuration doivent être passé par des variables d'environnement.

## Aucune référence ne figure à la configuration dans le code.

## Backing Services

## Il s'agit d’une ressource externe au conteneur (base mysql, smtp, activemq, memcache, ...).

## L'accès à ces ressources se fait en paramètre. Pas de différence entre les services locaux et distants.

## Build, release, run

## L'application et l'environnement sont recréés avant tout déploiement d'une nouvelle version.

## On apporte aucune modification sur l'application déployée.

## A chaque version déployée on accorde un numéro de version unique (timestamp, numero de commit, ...).

## Création de conteneurs (Build)

## Notion d'image qui permet une construction multicouche des conteneurs

## Il est possible de créer une nouvelle image tout en modifiant une image existante

|  |  |
| --- | --- |
| Build | Génération automatisée d'images avec langage de script spécialisé dans un Docker file, exemple :  FROM docker/whalesay:latest  RUN apt-get -y update && apt-get install -y fortunes CMD /usr/games/fortune -a | cowsay |
| Commit | Création d'une image à partir des modifications lors de l'exécution d'un conteneur |
| History | Affichage de l'historique d'une image (couches avec possibilité de dépiler) |
| save, load, export, import | Sauvegarde et rechargement d'images et de conteneurs |

Tableau 1 : Creation de conteneurs

1. **Dépôts de conteneurs (Ship)**

|  |  |
| --- | --- |
| Search | Rechercher d'images sur le Docker Hub |
| pull, push | Récupérer / envoyer d'image depuis / sur un dépôt (par défaut le Docker Hub) |

Tableau 2 : Dépots de conteneurs

* Plusieurs images officielles (gcc, python, ruby, java, golang, , haskell, centos, perl, , debian, fedora, ubuntu, …) .
* Plusieurs images publiques d'utilisateurs ou d’organisations.
* Versionnage des images .
* Possibilité d'employer son propre dépôt avec Docker Registry, notamment disponible sous forme d'image Docker officielle.

1. **Exécution et supervision de conteneurs (Run)**

|  |  |
| --- | --- |
| Daemon | Lancer la partie serveur (démon) de l'architecture client/serveur de Docker |
| Run | Lancer une commande dans un nouveau conteneur construit à partir d'une image |
| start, stop, restart, pause, unpause, kill | Contrôler l'exécution des conteneurs |
| ps, logs, top, stats | Surveiller des conteneurs |

Tableau 3 : Exécution et supervision de conteneurs

* Plusieurs conteneurs se lient via des liens réseau
* Le partage et la persistance de données se font en utilisant de volumes pour
* Coordonner l'exécution de multiples conteneurs (Docker Compose) est possible Port binding
* L'application donne un service qui écoute sur un port.

1. **Processus**

* L'application est effectuée dans l'environnement d'exécution en tant qu'un ou plusieurs processus.
* Stockage de toutes les données dans une ressource externe (base de données).
* Le non stockage local des variables de sessions utilisateurs.

1. **Port binding**

L'application donne un service qui écoute sur un port.

1. **Concurency** 
   * Possibilité de mettre à l'echelle chaque application. Pour répartir la charge, les conteneurs peuvent être lancés x fois.
   * Le programme dans le conteneur ne peut pas être lancé en tâche de fond.
   * L'arrêt du conteneur suite à l'arrêt du programme.
2. **Disposability** 
   * Le conteneur doit être à usage unique.
   * Il peut être lancé très rapidement.
   * Un arrêt inopportun ne doit pas détériorer les données.
3. **Dev/prod parity** 
   * Dès qu'il finit la saisie du code le développeur doit pouvoir le déployer rapidement.
   * Le responsable du développement doit être plus proche du déploiement (DevOps).
   * Maintenir la production et le développement aussi semblables que possible en utilisant les mêmes outils. Pour limiter les surprises en production, éviter de prendre des backends différents en prod et en dev (ex: base de données, ...)
4. **Logs** 
   * Pour la visualisation et l'archivage à long terme, les applications doivent externaliser leurs journaux
   * Posibilité d'afficher les journaux dans la sortie standard de l'application, mais pas dans un fichier du conteneur.
5. **Admin process** 
   * Les commandes d'administration doivent être saisies dans un environnement semblables aux autres processus d'exploitation.
   * Mêmes variables d'environnement, même conteneur, mais en mode conversationnel.

**2.8.2 Concepts**

1. **Layers**

* Les conteneurs et leurs images sont dissociés en couches (layers)
* Les layers peuvent être remployés entre différents conteneurs
* Optimisation de la gestion de l'espace disque.
  1. **Layers : Une Image**



Figure 5 : décomposition d'une image en layers

* 1. **Layers : Un Conteneur**

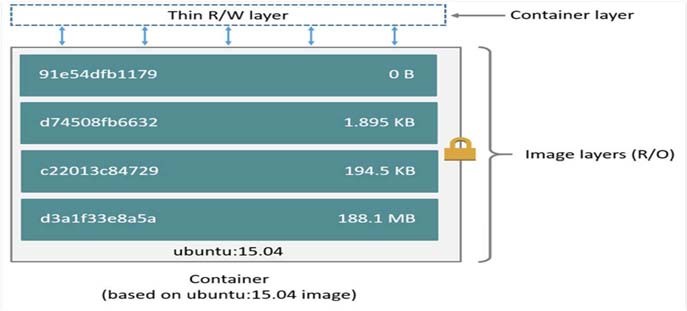


Figure 6 : Un conteneur est la somme d'une image et un layer r/w

* 1. **Layers : Plusieurs Conteneurs**

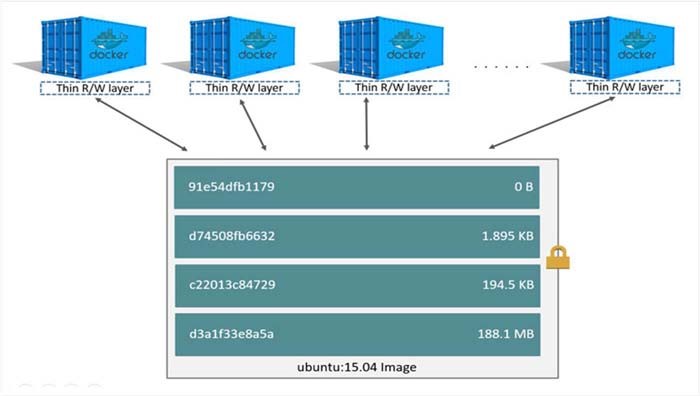


Figure 7 : Une image pour créer plusieurs conteneurs

* 1. **Layers : Répétition Des Layers:**

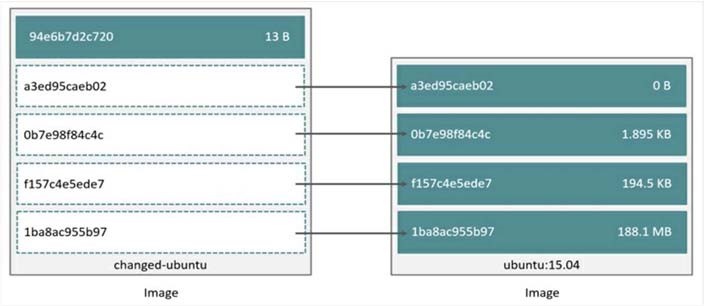


Figure 8 : Les layers sont indépendants de l'image

1. **Stockage :**

Le stockage se fait pour les Images Docker, données des conteneurs, volumes ainsi pour des multiples backends (extensibles via plugins) tels que:

* + - AUFS
    - Device Mapper
    - Over lay FS
    - NFS(Via plugin Convoy)
    - Gluster FS(Via plugin Convoy)
  1. **Stockage AUFS** 
     + Stable : moyenne performance écriture
     + Non intégré dans le Kerne lLinux (mainline)
  2. **Stockage Device Mapper** :
     + Basé sur Logiciel Volume Manager
     + Thin Provisionning et snapshot
     + Intégré dans le Kernel Linux
     + Stable : moyenne performance écriture
  3. **Stockage Over layFS** :
     + Successeur de AUFS
     + Performances accrues
     + Relativement stable mais moins éprouvé que AUFS ou Device Mapper

* 1. **Stockage PLUGINS** :
     + Étendre les drivers de stockages disponibles
     + Utilisation des systèmes de fichier distribués (GlusterFS)
     + Partage des volumes entre plusieurs hôtes docker

1. **Volume :** 
   * + Assurer une persistance des données
     + Le volume est indépendant des conteneurs et des layers
     + On distingue deux types de volumes :
       - Conteneur : Les données sont stockées dans un data conteneur
       - Hôte : Montage d'un dossier de l'hôte docker dans le conteneur
     + Le volume peut être partagé entre plusieurs conteneurs

* Volume : PLUGINS

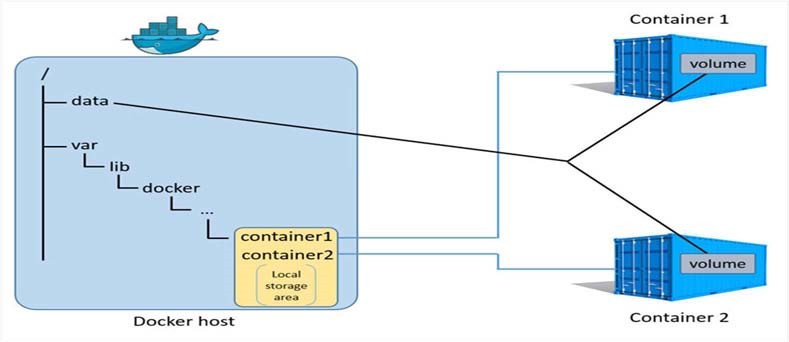


Figure 9 : Un volume monté sur deux conteneurs

* + - * Permet le partage de volumes entre différents hôtes
      * Fonctionnalité avancées : migration, snapshot, restauration

1. **Réseaux** :

|  |  |  |
| --- | --- | --- |
| NETWORK ID | NAME | DRIVER |
| 42f7f9558b7a | Bridge | Bridge |
| 6ebf7b150515 | None | Null |
| 0509391a1fbd | Host | Host |

Tableau 4 : Réseaux

* 1. Réseau Bridge :

Un réseau Bridgé est le plus employé dans Docker. Les réseaux Bridgés créés par les utilisateurs sont identiques au réseau bridge par défaut créé lors de l’installation de Docker (Docker0). Certes, de nouvelles fonctionnalités sont ajoutées, telle que la gestion du DNS...

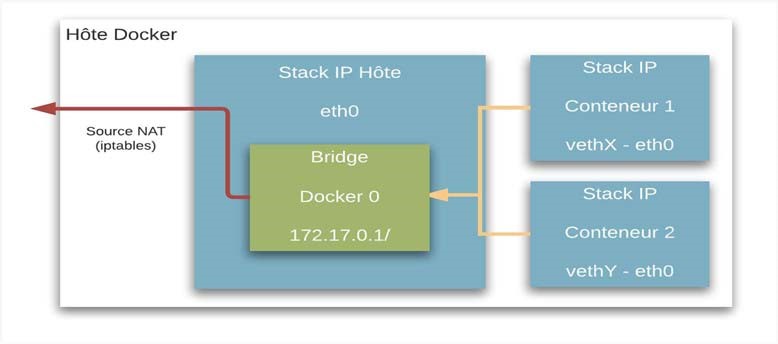


Figure 10 : Réseaux Bridge

* 1. **Évolutions du réseau Docker** :
     + - Les composants réseau sont refactorisés (lib network)
       - Système de plugins : multi host et multi backend (overlay network)
  2. **Réseau Multihost Overlay** :

C'est un réseau multi hôtes pour Docker, pour faire communiquer les conteneurs sur un réseau « interne » commun. Plusieurs réseaux overlay peuvent être créés.

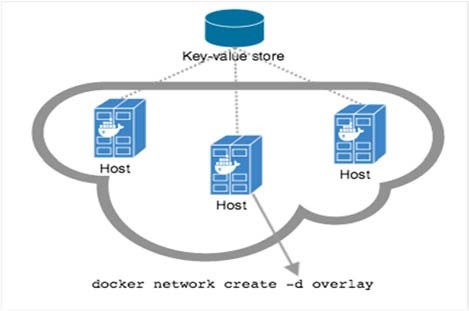


Figure 11 : Réseaux Multi Host

1. **Publication De Ports :** 
   * + - Se fait dans le cas d'un réseau diffèrent de l'hôte
       - De même, dans le cas où les conteneurs ne sont pas accessibles depuis l'extérieur
       - Les ports peuvent être publiés depuis l'hôte vers le conteneur
       - L'hôte est utilisé comme proxy de service

Ce schéma montre la publication de ports :

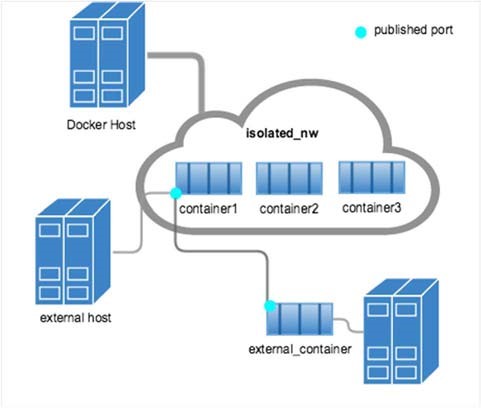


Figure 12 : schéma de publication des ports

1. **Links :** 
   * + - Communication des conteneurs d'un même réseau via IP
       - Les liens pour lier deux conteneurs par nom
       - Les liens peuvent être remplacés par les discovery services

##### **Configuration système requise et systèmes pris en charge de Docker**



Figure 12 : Configuration système requise et systèmes pris en charge de Docker

**2.10 Quelques concurrents de Docker**

Dans cette section, nous présenterons quelques alternatifs à Docker, leurs caractéristiques, avantages et inconvénients :

## Rkt de CoreOS

## LXC

## LXD de Canonical

## Linux-VServer

## OpenVZ/Virtuozzo 7[1]

### **2.10.1 RKT de CoreOS**

Sur le marché de la virtualisation par conteneurs, **le concurrent le plus féroce de Docker** est l’environnement d‘exécution [*rkt*](https://coreos.com/rkt/)(prononcé : « Rocket ») du distributeur Linux *CoreOS*. C**’est un système d’exploitation open source. Basé sur le kernel Linux, cet OS permet de déployer et de gérer des applications au sein de containeurs** [3]

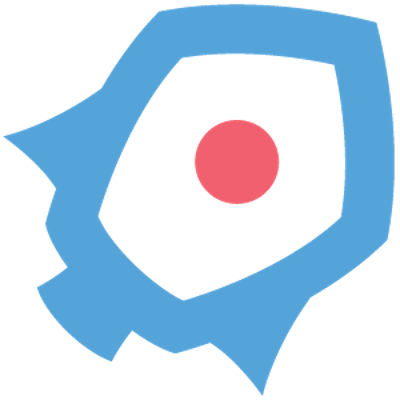


Figure 13: RKT de CoreOS

**2.10.1.1 Quelques informations sur RKT CoreOS**

* Le projet de logiciel a été présenté en 2014
* En février 2016, *CoreOS* a publié *rkt* version 1.0, la première version stable de l’environnement d‘exécution des conteneurs.[1]

Contrairement à Docker, rkt supporte d’autres formats en plus de ses propres images de conteneurs. L’environnement d’exécution est **compatible avec les images Docker** et offre un moyen de convertir n’importe quel format de conteneur en ACI avec l’outil open source [*Quay*](https://quay.io/).

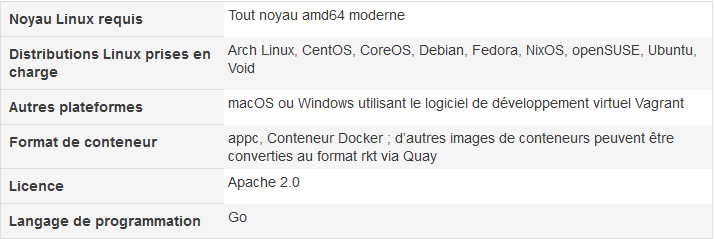
La définition d’un standard autour de ce que doit être un container ayant été mise en place aux débuts de Docker, les équipes de Container Linux ont fortement participé à son développement.

Toutefois, l’évolution de Docker ne correspondait pas à ce que les développeurs de CoreOS envisageaient à la base :

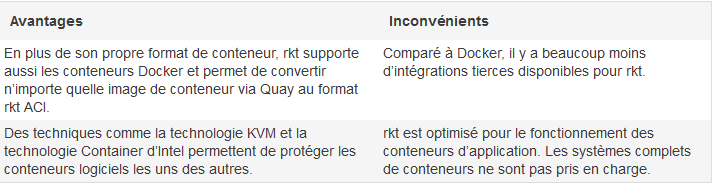
* Docker tourne avec un daemon qui gère l’ensemble des composants. Si ce processus disparait, les conteneurs disparaissent (la situation a changé en Juin 2016 avec la sortie de la version 1.11 de Docker, avec le Docker Engine séparé du daemon gérant les containers).
* De simple conteneur, Docker est devenu tout un écosystème packagé dans un seul binaire là encore, la situation a changé depuis ce constat : création d’images, orchestrateur de containeurs.

De ce constat est né le projet RKT.[4]

**2.10.1.2 Configuration système requise et systèmes pris en charge**



**2.10.1.3 Quelques avantages et inconvénients**



### **2.10.2 LXC**

LXC, est un ensemble d’outils, de templates, de bibliothèques, qui forment ensemble une **interface d’espace utilisateur (userspace) avec les fonctions de conteneur natives du noyau Linux**. LXC offre aux utilisateurs Linux un moyen pratique de créer et de gérer des applications et des systèmes de conteneurs.

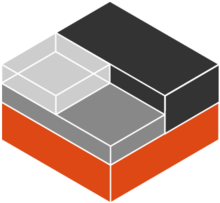
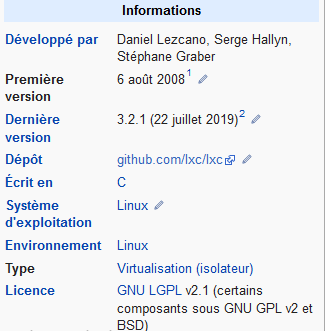
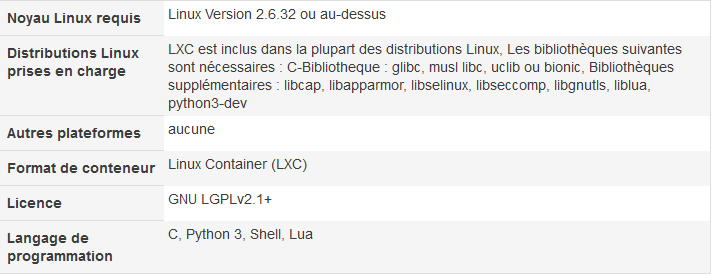


Figure 14: LXC

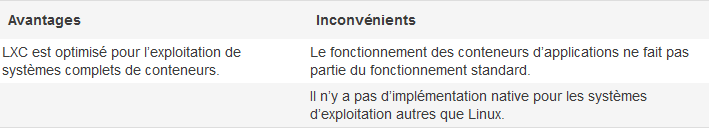
**2.10.2.1 Quelques informations sur LXC**



**2.10.2.2 Configuration système requise et systèmes pris en charge**



**2.10.2.3 Quelques avantages et inconvénients**



### **2.10.3 LXD de Canonical**

**LXD** est un logiciel libre développé par Canonical pour simplifier la manipulation de conteneurs de logiciels à la manière d'un hyperviseur de VM. C'est une surcouche logicielle de [LXC](https://doc.ubuntu-fr.org/lxc). Il fait partie du projet global [Linux Containers](https://linuxcontainers.org/fr/) de gestion de conteneurs[5].

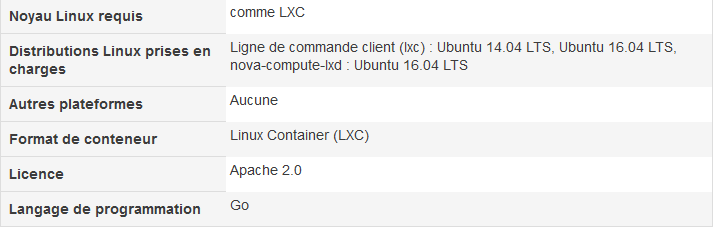


Figure 15 : LXD

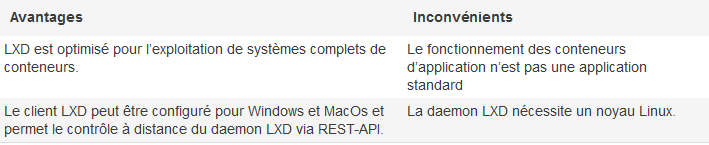
**2.10.3.1 Quelques informations sur LXD Canonical**

En tant que projet suiveur de LXC, le distributeur Linux Canonical a lancé LXD en novembre 2014. Le projet s’appuie sur la technologie des conteneurs Linux et l’étend par le processus Daemon LXD. Le logiciel est une sorte d**’hyperviseur de conteneur**. L’objectif du projet logiciel est d’offrir aux utilisateurs une **expérience utilisateur basée sur la technologie des conteneurs Linux similaire à celle des machines virtuelles**. [1]

**2.10.3.2 Configuration système requise et systèmes pris en charge**



**2.10.3.3 Quelques Avantages et inconvénients de LXD Canonical**



### **2.10.4 Linux-VServer**

Linux-VServer est une technique de virtualisation au niveau du système d’exploitation qui utilise des techniques **d’isolation du noyau Linux similaires aux conteneurs logiciels**. Plusieurs unités virtuelles sont exploitées sur un noyau Linux commun, dont les ressources (systèmes de fichiers, CPU, adresses réseau et mémoire) sont divisées en partitions compartimentées. Cette technologie est utilisée, par exemple, par les fournisseurs d‘hébergement Web qui veulent offrir à leurs clients des machines virtuelles séparées sur une base matérielle commune.[1]

[](https://www.google.com/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&ved=2ahUKEwiVxPLe9ZjkAhVRxIUKHb8bCUUQjRx6BAgBEAQ&url=https://fr.wikipedia.org/wiki/Linux-VServer&psig=AOvVaw3HzF6Gxr4CfCrQfSoYIWzv&ust=1566647193795514)

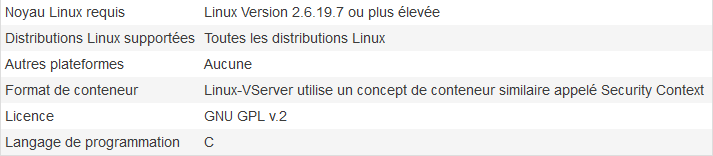
Figure 16: Linux-VServer

**2.10.4.1 Quelques informations sur Linux-VServer**

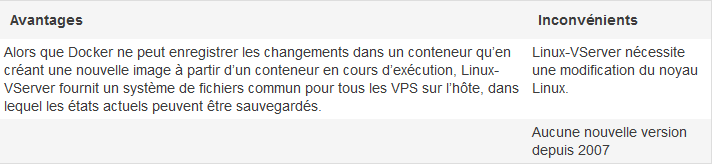
Cette technologie open source est assez récente sous Linux. La version 0.0 date d'octobre 2001. Jacques Gélinas, canadien à l'origine de plusieurs projets bien connus sous Linux (linuxconf, umsdos) a démarré le projet. Cette solution est spécifique à Linux, mais non liée à la plateforme IA-32. Le développement des versions s'est ralenti fin 2002. De nombreuses modifications ont alors fait leur apparition, et le projet s'est transformé en un projet communautaire. Le leader de ce projet est devenu Herbert Poetzl à partir d'octobre 2003. Depuis ce moment de nombreuses évolutions ont vu le jour. La version 1.0 est sortie le 1er novembre 2003, suivie de la version 1.2 le 5 décembre 2003. Cette branche est toujours active pour le noyau Linux 2.4 (version 1.2.10 pour noyau 2.4.29). La version 2.0 est sortie le 7 août 2005 pour le noyau 2.6.12. Cette version apporte de nombreux perfectionnements.[6]



**2.10.4.2 Configuration système requise et systèmes pris en charge**

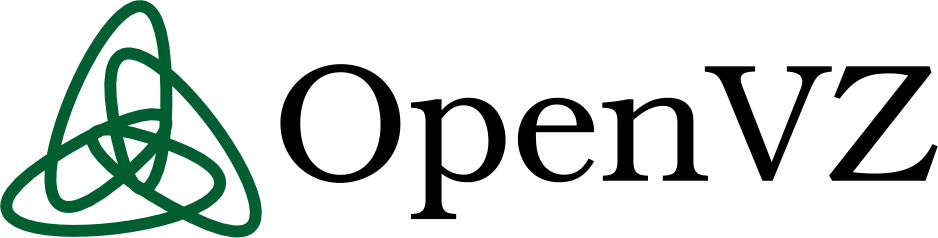


**2.10.4.3 Quelques avantages et inconvénients**



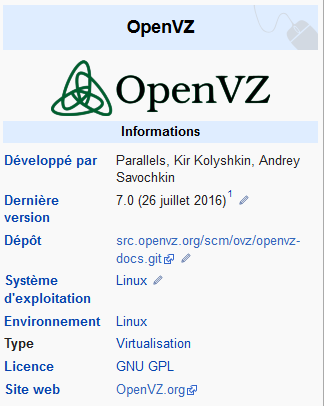
### **2.10.5 OpenVZ/Virtuozzo 7**

Le logiciel est basé sur Red Hat Enterprise Linux 7 (RHEL) et permet le fonctionnement de systèmes invités qui peuvent être implémentés sous forme de machines virtuelles ou bien sous forme de conteneurs



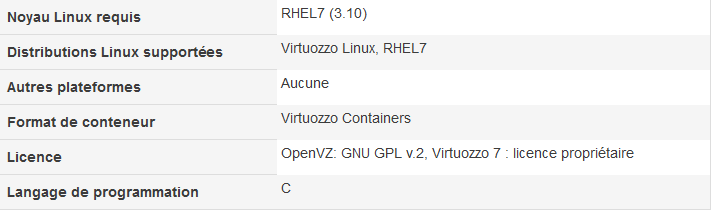
**Figure 17 : Linux-VServer**

**2.10.5.1 Quelques informations sur OpenVZ/Virtuozzo 7**

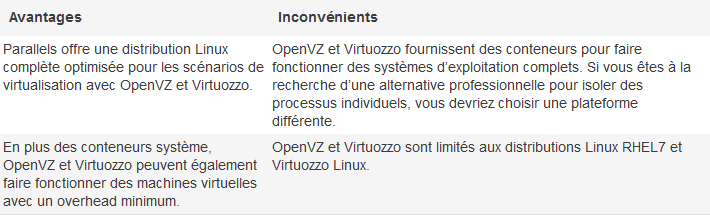


**Figure 18 : OpenVZ**

**2.10.5.2 Configuration système requise et systèmes pris en charge**



**2.10.5.3 Quelques avantages et inconvénients**



**2.10.6 Tableau comparatif des différentes solutions**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **Docker** | |  | | --- |   **Rkt de CoreOs** | LXC | | **LXD** | | --- | |
| Technologies de virtualisation | OS-Level | OS-Level, Hypervisor | OS-Level | |  |  | | --- | --- | |  | OS-Level | |
| Full System Container | Non | |  |  | | --- | --- | |  | Non | | |  |  | | --- | --- | |  | Oui | | |  |  | | --- | --- | |  | Oui | |
| |  |  | | --- | --- | | App Container |  | | Oui | Oui | |  |  | | --- | --- | | Non |  | | |  |  | | --- | --- | | Non |  | |
| |  |  |  |  | | --- | --- | --- | --- | | Licence |  |  |  | | Apache 2.0 | Apache 2.0 | GNU LGPLv2.1+ | Apache 2.0 |
| Format de conteneur | Docker-Container | appc, Docker-Container | Linux Container (LXC) | Linux Container (LXC) |
| Plateforme supportée | Linux, Windows, Mac0S, Microsoft Azure, Amazon Web Services (AWS) | Linux, Windows, MacOS | Linux | |  |  | | --- | --- | |  | Linux | |
| Dernière version | |  |  | | --- | --- | |  | Avril 2017 | | Février 2017 | Janvier 2017 | Mars 2017 |
| Patch du noyau Linux requis | Non | |  |  | | --- | --- | |  | Non | | Non | Non |
| Langage de programmation | Go | Go | C, Python 3, Shell, Lua | Go |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Linux-VServer** | |  | | --- |   **OpenVZ** |
| Technologies de virtualisation | OS-Level | OS-Level, Hyperviseur |
| Full System Container | Oui | |  |  | | --- | --- | |  | Oui | |
| |  |  | | --- | --- | | App Container |  | | Non | Non |
| |  |  |  |  | | --- | --- | --- | --- | | Licence |  |  |  | | |  |  | | --- | --- | |  | GNU GPL v.2 | | |  |  | | --- | --- | |  | GNU GPL v.2 | |
| Format de conteneur | Security Context | Virtuozzo Containers |
| Plateforme supportée | |  |  | | --- | --- | |  | Linux | | Linux (uniquement Virtuozzo Linux, RHEL7) |
| Dernière version | |  |  | | --- | --- | |  | Avril 2007 | | Juillet 2016 |
| Patch du noyau Linux requis | Oui | |  |  |  | | --- | --- | --- | | |  |  | | --- | --- | |  | Distribution indépendante | | |
| Langage de programmation | C | C |

## 2.10.7 Technologie d’orchestration des conteneurs

Pour l’orchestration de l’écosystème microservices, le choix se fera entre les technologies suivantes :

* Docker-compose
* Kubernetes
* Docker Swarm

2.10.7.1 Docker Compose

Docker Compose est un outil qui permet de décrire et gérer en ligne de commande plusieurs conteneurs comme un ensemble de services interconnectés. L’outil fournit des programmes en ligne de commande qu’on utilise pour gérer le cycle de vie complet des applications. Ainsi l’on pourra démarrer, modifier, supprimer un ensemble de conteneurs en une seule commande.

**2.10.7.2 Kubernets**

Kubernetes est un projet Open Source créé par Google en 2015. Il permet **d’automatiser le déploiement et la gestion d’applications multi-container à l’échelle**. Il s’agit d’un système permettant d’exécuter et de coordonner des applications containerisées sur un cluster de machines. C’est une plateforme conçue pour gérer entièrement le cycle de vie des applications et services containerisés en utilisant des méthodes de prédictibilité, de scalabilité et de haute disponibilité.

**2.10.7.3 Docker Swarm**

Swarm est un logiciel développé par les développeurs de Docker, il permet de consolider n’importe quel nombre d’hôtes Docker dans **un cluster et la gestion centrale de celui-ci** ainsi que l’orchestration des conteneurs.

**2.10.8 Choix pour la mise en place des microservices**

Après l’étude des avantages des technologies de mise en place des microservices, notre choix se porte sur Docker car non seulement Docker est Open Source mais elle **est également la plateforme de conteneurs la plus populaire et la plus utilisée*.*** La plateforme a rencontré un franc succès auprès de nombreuses entreprises. Aujourd’hui, selon les créateurs de Docker, plus de 3,5 millions d’applications ont été conteneurisées en utilisant cette technologie, et plus de 37 milliards d’applications containerisées ont été téléchargées. De même, d’après le système de monitoring cloud DataDog, 18,8% des utilisateurs avaient adopté la plateforme en 2017. De son côté, RightScale estime que l’adoption de la plateforme dans l’industrie du Cloud a augmenté de 35% en 2017 à 49% en 2018. Des géants comme *Oracle* et *Microsoft* l’ont adopté, au même titre que presque toutes les entreprises du Cloud.

Selon de *451 Research*, l’essor de Docker n’est pas près de s’arrêter. Ces analystes estiment que le marché des conteneurs va littéralement exploser d’ici 2021. Les revenus seront multipliés par 4 avec un taux de croissance annuel de 35%, passant de 749 millions de dollars en 2016 à 3,4 milliards de dollars en 2021[7]

**2.10.9 Choix pour l’orchestration des conteneurs**

**Chapitre 4**

**:**

**L**

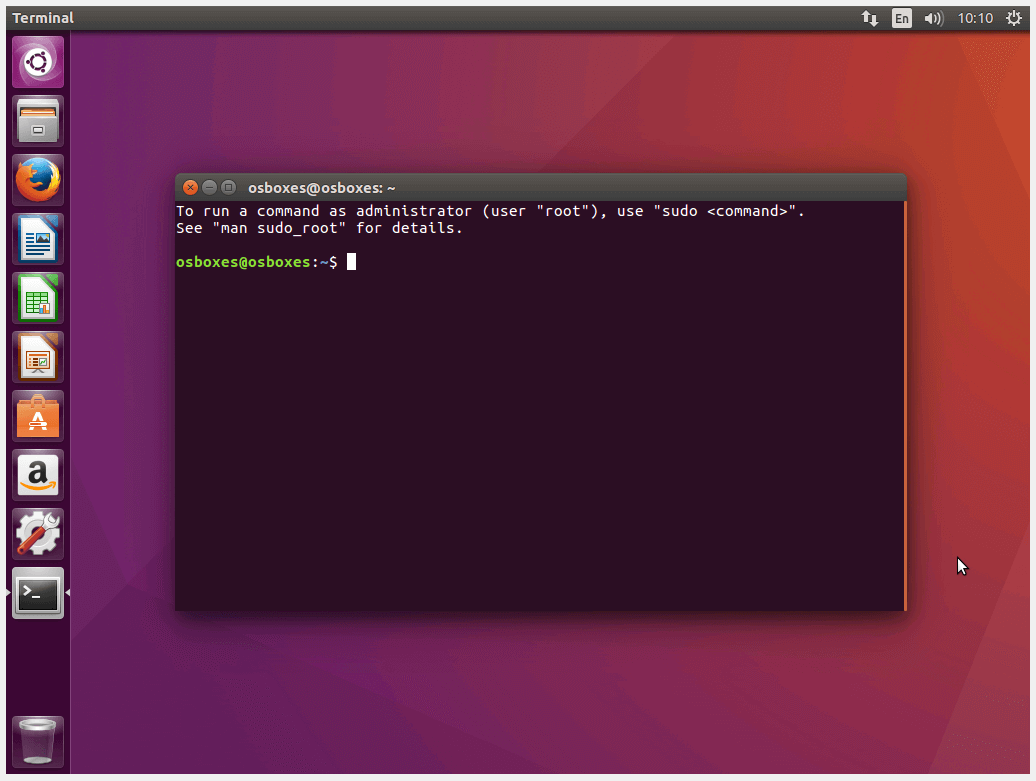
**a mise en œuvre**

### **I-Avant l’installation de Docker**

Avec Unbntu 16.10, le processus d’installation suit les mêmes étapes. Il est recommandé aux utilisateurs de Unbuntu 14.04 (LTS) d’installer les paquets linux-image-extra-\* packages avant l’installation de Docker. Ceux-ci permettent au Docker-Engine d’accéder aux pilotes de mémoire AUFS.

Le gestionnaire de paquets intégré *APT* (Advanced Packaging Tool) offre une méthode pratique pour fournir des mises à jour à un système Linux. Pour installer les **paquets additionnels pour Unbuntu 14.04**, on suit ces étapes :

1. **Ouvrir le terminal :** On démarre Ubuntu et ouvrir le terminal, par exemple en utilisant la combinaison de touches [STRG] + [ALT] + [T].



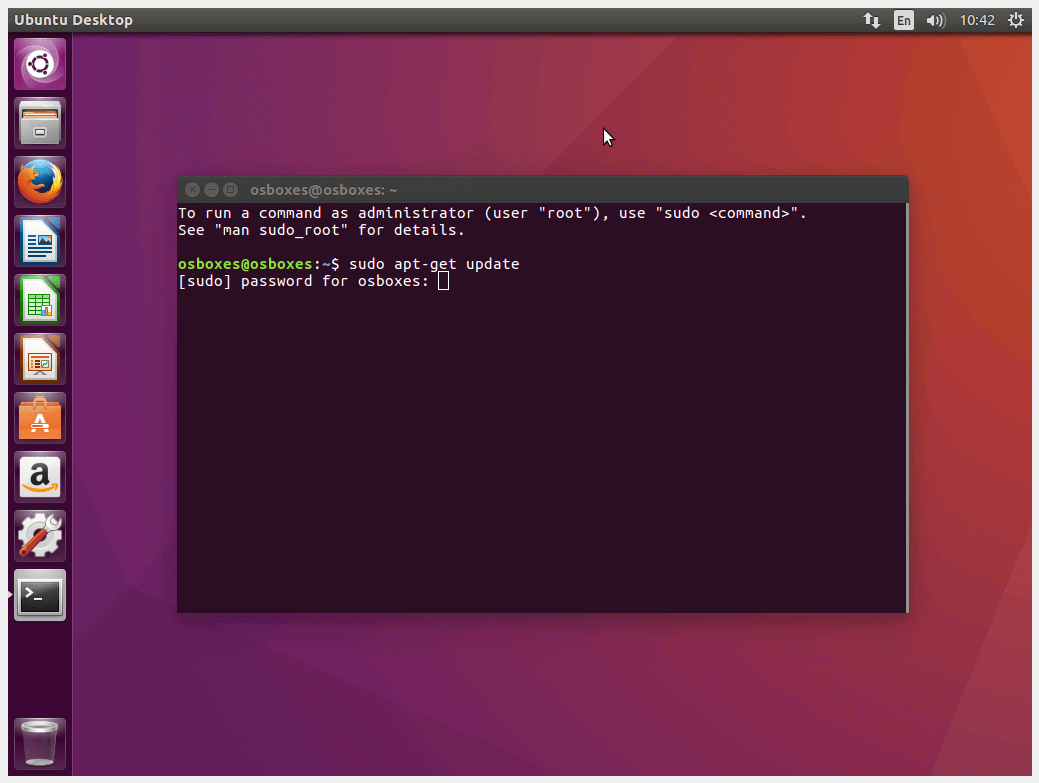
**2. Mettre à jour les listes de paquets :** On entre la commande suivante pour mettre à jour l’index des paquets locaux du système d’exploitation. On confirme avec la touche [ENTER].

$ sudo apt-get update

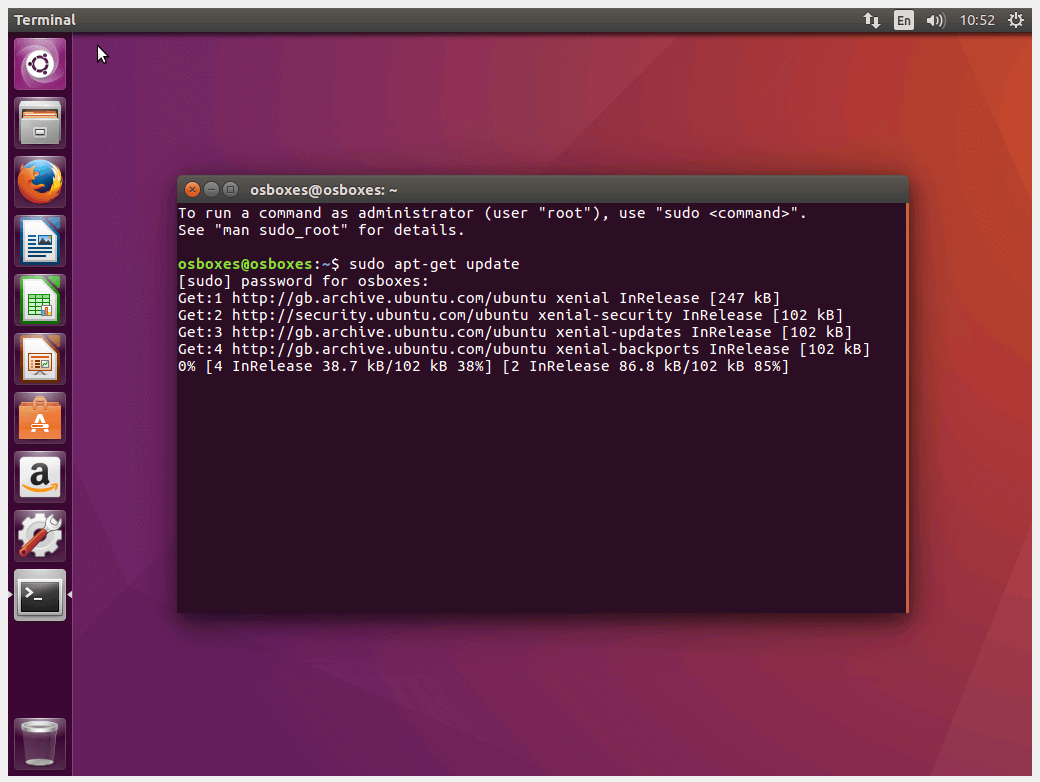
La commande apt-get update n’installe pas les nouveaux paquets : L'option update **met à**

**Jour la liste des fichiers disponibles dans les dépôts APT présents dans le fichier de configuration /etc/apt/sources.list**. L'exécuter régulièrement est une bonne pratique, afin de maintenir à jour la liste de paquets disponibles. [18] L’ajout sudo permet d’exécuter des commandes en tant qu’administrateur (Superuser „root“). Certaines commandes peuvent nécessiter des droits Root. Dans ce cas, Ubuntu demande le mot de passe administrateur. Alternativement, nous pouvons en permanence passer au rôle d’administrateur via sudo -s.

Pour installer la plateforme de conteneur Docker, nous avons besoin des droits root pour le système hôte en question.



Si on est identifié comme utilisateur Root par mot de passe, Ubuntu lance le processus de mise à jour. Le statut est affiché dans le terminal.



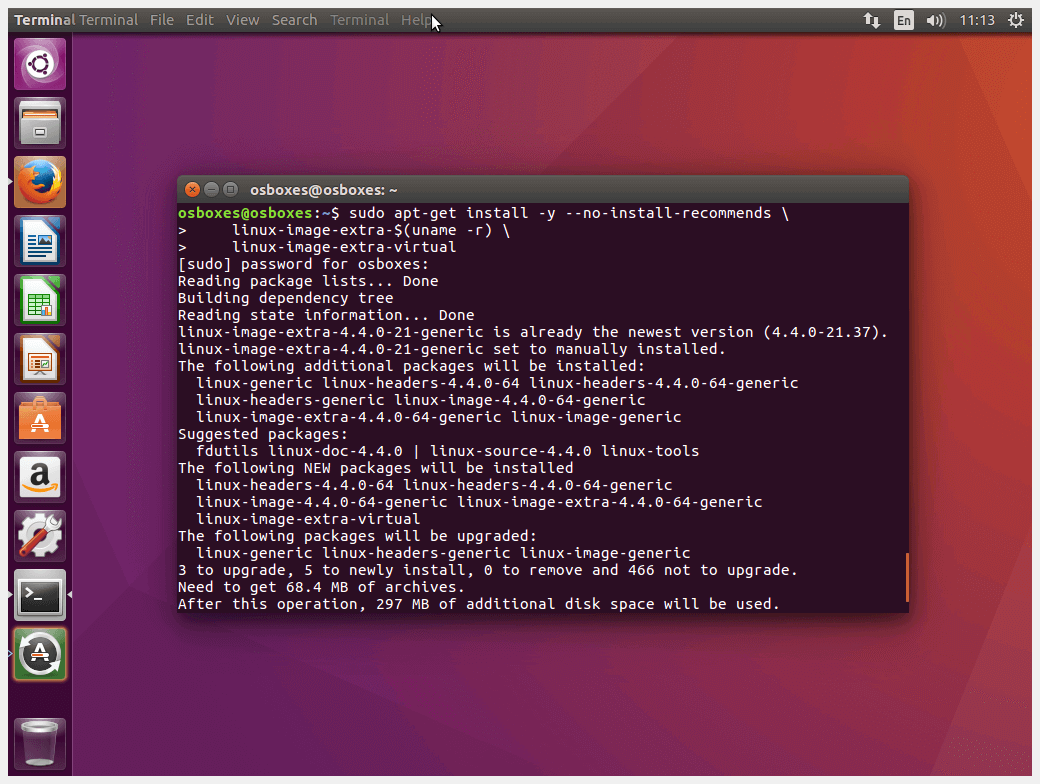
**3. Installation des paquets supplémentaires :** Toutes les descriptions de paquets ont été mises à jour, nous pouvons maintenant procéder à l’installation de nouveaux paquets. Le gestionnaire de paquets APT fournit pour ce faire la commande apt-get install "PaketName". Pour charger les paquets supplémentaires recommandés pour unbuntu 14.04 à partir du dépôt Ubuntu et les installer sur le système, nous devons entrer la commande suivante dans le terminal et confirmez avec [ENTER].

$ sudo apt-get install -y --no-install-recommends \

linux-image-extra-$(uname -r) \

linux-image-extra-virtual

Si nous utilisons des commandes avec l’option -y, toutes les questions interactives sont automatiquement répondues par « Oui ». L’option --no-install-recommends empêche Ubuntu d’installer automatiquement les paquets recommandés.



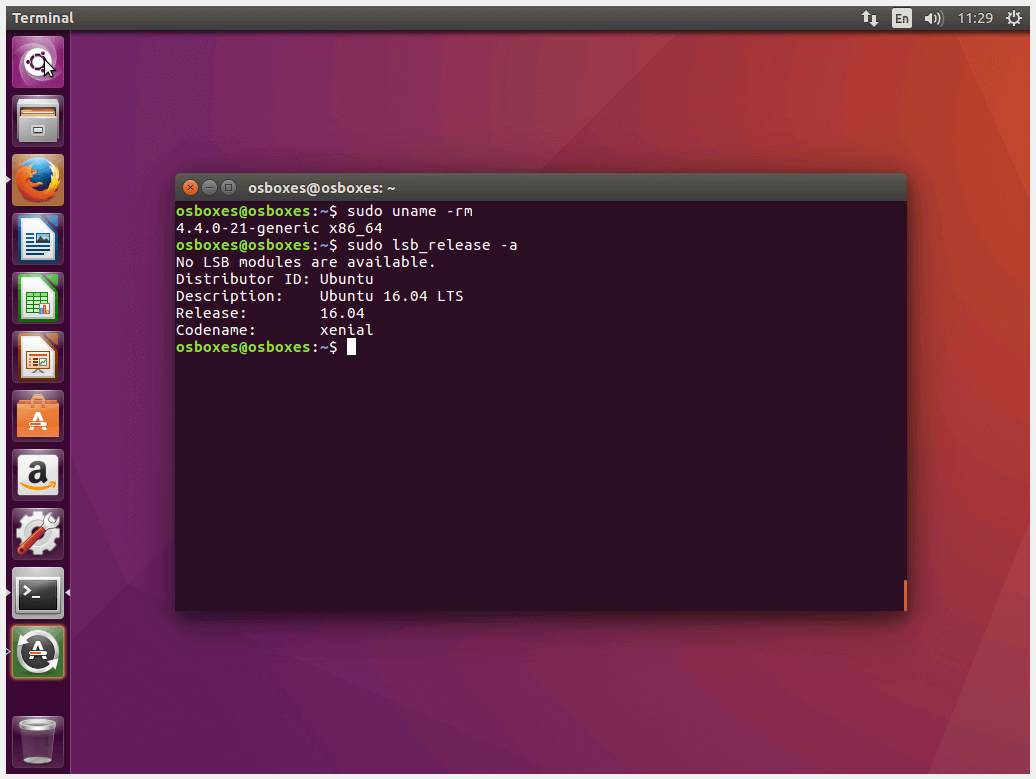
Après avoir téléchargé les paquets supplémentaires pour Unbuntu 14.04, toutes les fonctionnalités de la plateforme Docker sont également disponibles sur cette version d’Ubuntu.

Si l’on ne connait pas la version Ubuntu du système utilisé ou bien si on est pas sûr de pouvoir fournir l’architecture 64 bits requise pour une installation Docker. La version du noyau et l’architecture du système peuvent être déterminées dans le terminal Ubuntu à l’aide de la commande suivante :

$ sudo uname -rm

La version **Ubuntu** correspondante, le **Release** ainsi que le surnom peuvent être affichés par l’entrée suivante :

$ sudo lsb\_release –a



**II-Installation de Docker**

**1-Installation sous linux**

### **1.1 Installation manuelle via le paquet DEB**

En principe, Docker peut être téléchargé sous forme de paquet DEB et être installé manuellement. Le paquet d’installation requis est disponible à l’adresse URL suivante :[*apt.dockerproject.org/repo/pool/main/d/docker-engine/*](https://apt.dockerproject.org/repo/pool/main/d/docker-engine/) [Télécharger Docker sous forme de paquet DEB].

On télécharge le fichier DEB de la version Ubuntu souhaitée et on lance le processus d’installation avec cette commande via le terminal Ubuntu :

$ sudo dpkg -i /home/stage/package.deb

On installe le paquet *DEB* sauvegardé dans le dossier /home/stage/. Dans le cas d’une installation manuelle, toutes les mises à jour du logiciel doivent aussi être effectuées manuellement. La documentation de Docker vous recommande donc d’utiliser le référentiel propre de Docker. Ceci permet d’installer la plateforme à conteneurs confortablement à partir du terminal Ubuntu et de la maintenir à jour.

1.2 Installation à partir du dépôt Docker

Le meilleur moyen d’accéder à la plateforme de conteneurs est de l’installer à partir du dépôt Docker. Voici la configuration du système pour que le gestionnaire de paquets APT puisse accéder au dépôt Docker via HTTPS.

1.2.1. **Installer les paquets : On entre** la commande suivante pour installer les paquets nécessaires pour accéder au Docker Repository :

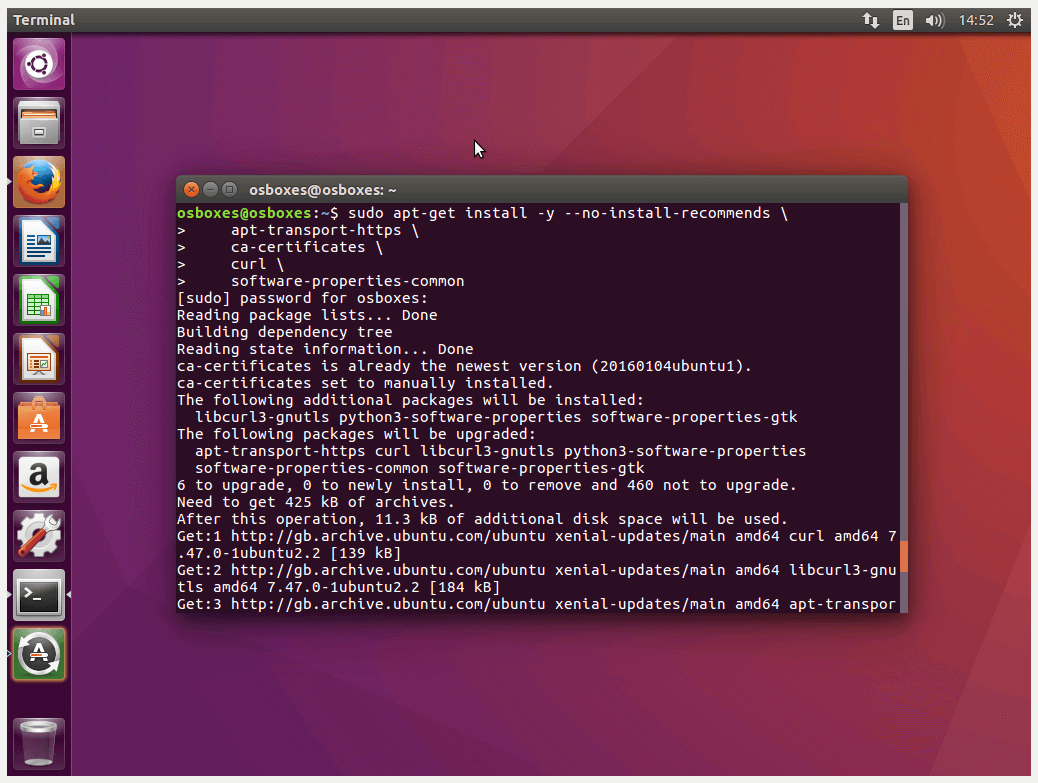
1 $ sudo apt-get install -y --no-install-recommends \

2 apt-transport-https \

3 ca-certificates \

4 curl \

5 software-properties-common



**1.2.2 Ajout de la clef GPG :** ajouter la clef GPG officielle de Docker

$ curl -fsSL https://apt.dockerproject.org/gpg | sudo apt-key add –

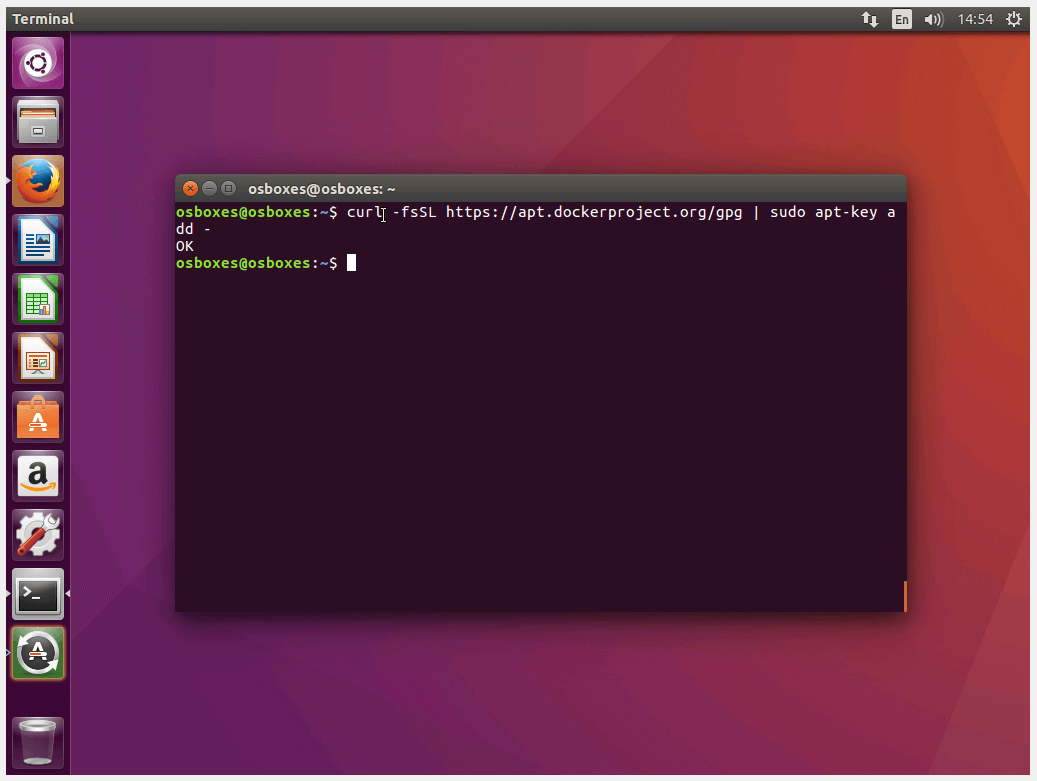
**1.2.3 Vérification de la clef GPG :** Nous devons nous assurer que la clé GPG correspond à l’ID suivant : 5811 8E89 F3A9 1289 7C07 0ADB F762 2157 2C52 609D. On utilise la commande ci-dessous :

$ apt-key fingerprint 58118E89F3A912897C070ADBF76221572C52609D

Le message suivant s’affiche ensuite dans le terminal :

pub 4096R/2C52609D 2015-07-14

Key fingerprint = 5811 8E89 F3A9 1289 7C07 0ADB F762 2157 2C52609D uid Docker Release Tool (releasedocker)<docker@docker.com>



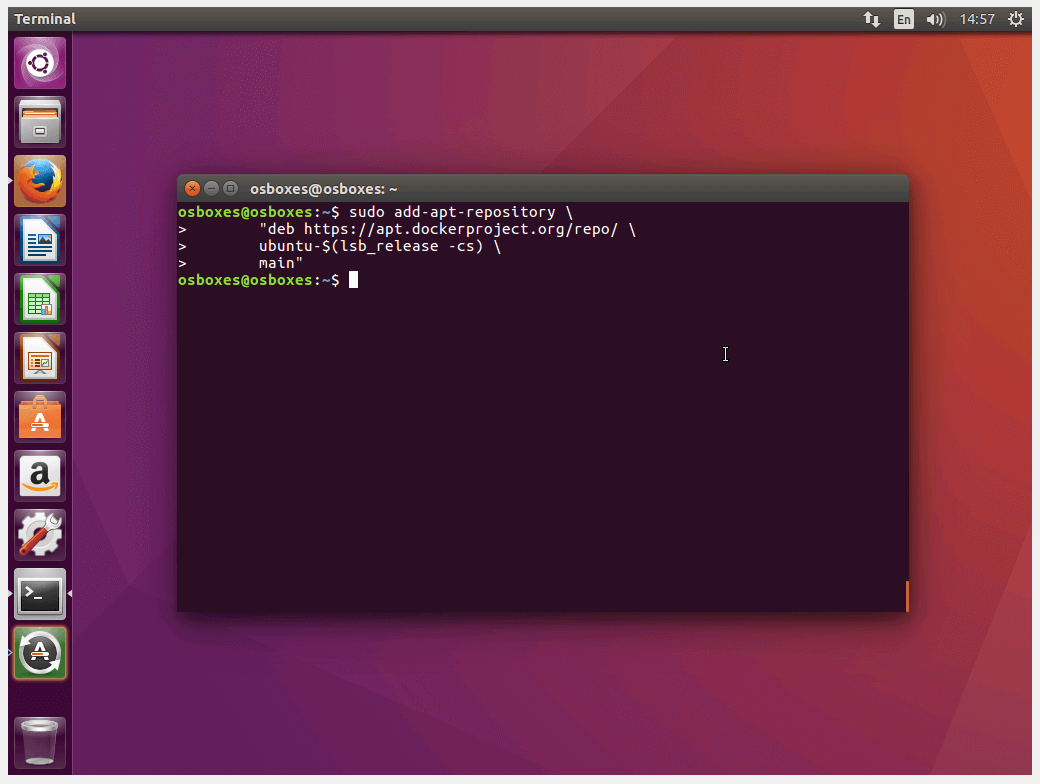
**1.2.4 Configuration du dépôt Docker** : pour garantir un accès sécurisé au dépôt Docker, entre la commande suivante :

1 $ sudo add-apt-repository \

2 "deb https://apt.dockerproject.org/repo/ \

3 ubuntu-$(lsb\_release -cs) \

4 main"



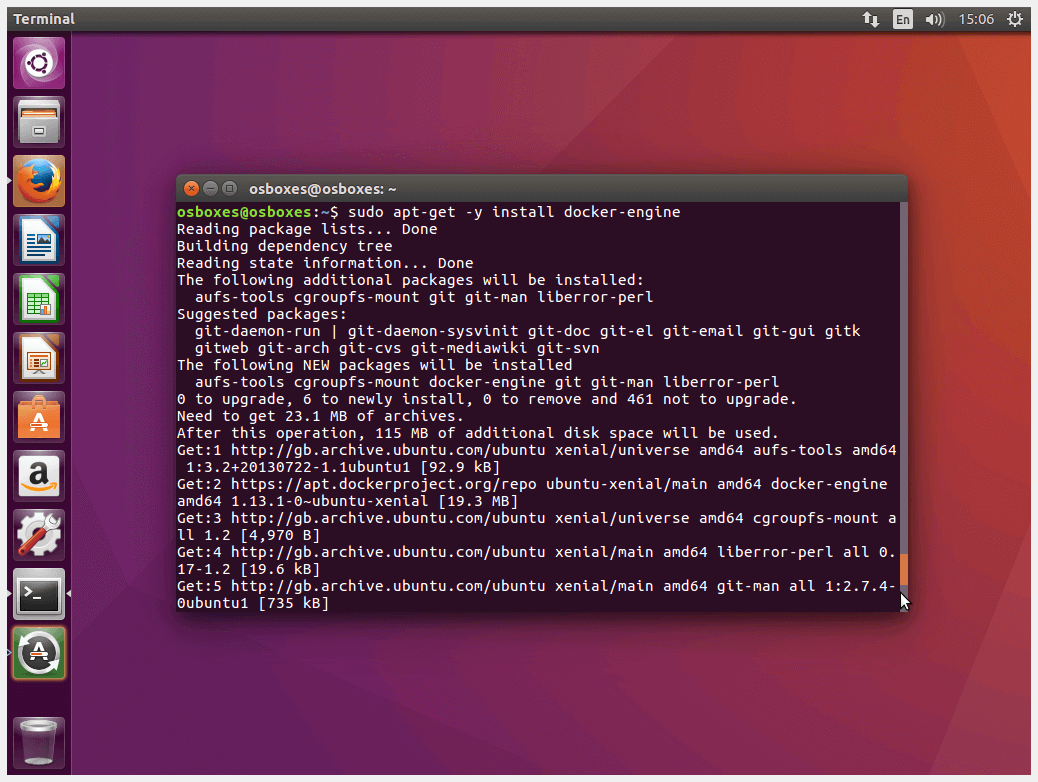
Le système est maintenant entièrement préconfiguré pour installer la plateforme de conteneurs à partir du dépôt Docker. Comme alternative au dépôt stable, on peut également utiliser le dépôt test de Docker. On charge le fichier /etc/apt/sources.list et on remplace le mot main par testing. Il l’est pas recommander d’utiliser le référentiel de test sur les systèmes de production.

**1.2.5 Mettre à jour l’index des paquets :** avant de procéder à l’installation du moteur Docker, il est recommandé de mettre à jour l’index des paquets de votre système d’exploitation. Utilisez à nouveau la commande :

$ sudo apt-get update

**1.2.6 Installation du Docker à partir du référentiel :** il y a deux façons de charger le moteur Docker à partir du référentiel Docker et de l’installer sur le système Ubuntu. Pour charger la dernière version du moteur Docker, on utilise la commande :

$ sudo apt-get -y install docker-engine



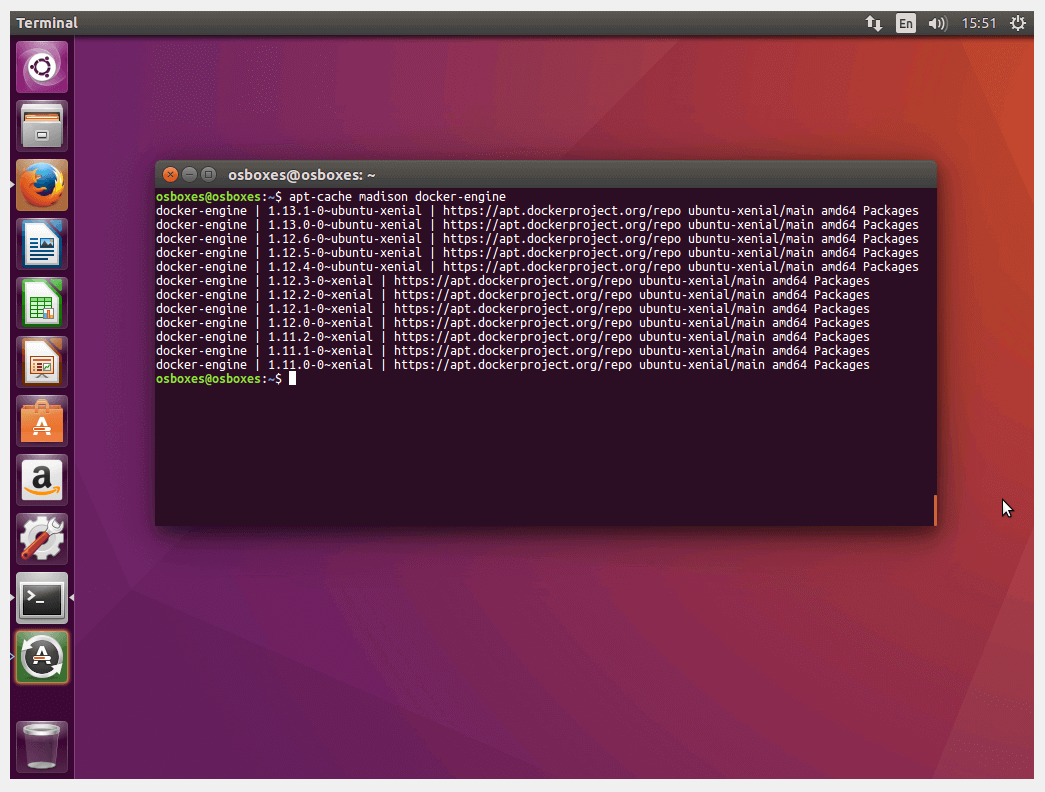
La plateforme à conteneurs est prête à l’emploi une fois le processus d’installation terminé.

Le démon Docker démarre automatiquement. S’il y avait une version plus ancienne de la plateforme de conteneur sur votre système avant l’installation de Docker, elle a été remplacée par le logiciel nouvellement installé.

Comme alternative à la dernière version, il est possible **d’installer n’importe quelle ancienne version du moteur Docker**. Ceci a du sens, par exemple, lorsqu’il est utilisé sur des systèmes de production. Ici, les utilisateurs préfèrent parfois les versions établies avec de bonnes valeurs d’expérience aux versions logicielles plus récentes.

Un **aperçu des versions de Docker disponibles** pour votre système peut être obtenu avec la commande suivante :

$ apt-cache madison docker-engine



Pour installer une version spéciale de Docker, il suffit d’ajouter la **chaîne de version** (par exemple 1.12.5-0) à la commande d’installation en l’ajoutant au nom du paquet (dans ce cas docker-engine) séparé par un signe égal.

$ sudo apt-get -y install docker-engine=<VERSION\_STRING>

1.3 Installation à partir du Repository Unbuntu

Les utilisateurs qui ne souhaitent pas utiliser le repository Docker ont également la possibilité de charger la plate-forme de conteneurs **à partir du repository du système d’exploitation.** Utilisez la ligne de commande suivante pour créer un [paquet Docker fourni par la communauté Ubuntu](http://packages.ubuntu.com/xenial/docker.io) pour l’installer :

$ sudo apt-get install -y docker.io

Cette methode est celle utilisé pour la réalisation de notre projet.

**2 Installation sous Windows**

### **2.1 Docker4Windows**

Docker4Windows est encore en version bêta, mais reste totalement utilisable au quotidien. Pour le moment il n'est compatible qu'avec Windows 10 (version pro, Enterprise et Education), si nous sommes sur une autre version de Windows il faudra passer par docker-toolbox ou faire une installation manuelle.

Avant de commencer l'installation de docker4windows, nous devons activer hyper-v. Pour ce faire, clic droit sur le menu d'application -> panneau de configuration -> Programmes et fonctionnalités -> Activer ou désactiver des fonctionnalités Windows -> Cocher Hyper-V -> OK. On redémarre le PC, et normalement nous avons Hyper-V installé. On peut passer à l'installation de docker4windows, et là c'est vraiment simple, télécharger l'exécutable à l’adresse <https://download.docker.com/win/beta/InstallDocker.msi> et l’installer. Ainsi doit apparaitre normalement dans la zone des notifications, une petite baleine, cela veut dire que Docker est bien installé.

### **2.2 Docker-toolbox**

L'installation de docker-toolbox est rapide, il suffit de télécharger [docker-toolbox](https://download.docker.com/win/stable/DockerToolbox.exe) à l’adresse <https://download.docker.com/win/stable/DockerToolbox.exe> et l’installer. Ceci installera toute la panoplie du super-docker, c'est-à-dire, virtualbox avec une VM boot2docker, le client docker, docker-machine et docker-compose pour Windows. Puis également, si on le souhaite, kitematic, qui est un GUI pour installer des applications via docker.

## 1.3 Installation sous Mac

### **1.3.1 Docker4Mac**

Docker4Mac est encore en version bêta, mais reste totalement utilisable au quotidien. L'installation de docker4Mac est vraiment simple, téléchargez l'exécutable à l’adresse <https://download.docker.com/mac/beta/Docker.dmg> et ensuite l’installer.

### **1.3.2 Docker-toolbox**

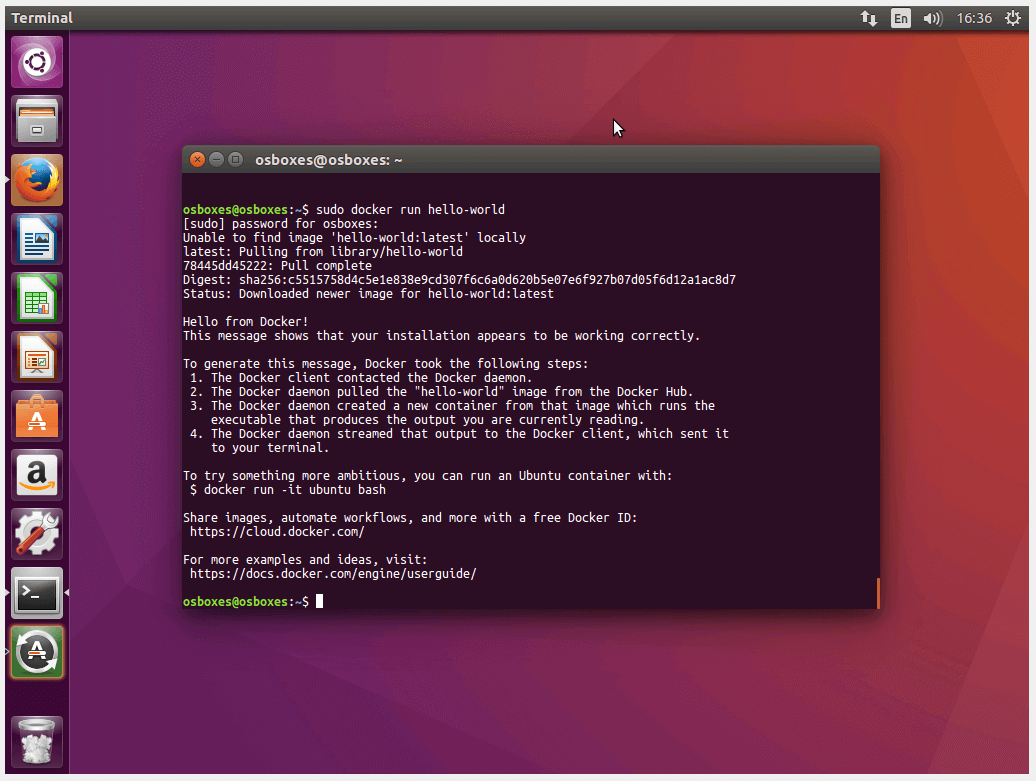
Il faut télécharger l'installateur « docker toolbox » ici : <https://www.docker.com/products/docker-toolbox> , ensuite on exécute le pkg et installer docker comme indiqué à l’adresse <https://docs.docker.com/engine/installation/mac/>. Ensuite on crée une VM docker, ça va créer une vm dans virtualbox qui aura pour nom docker (On pourra vérifier dans virtualbox)

1.4- Test pour le bon déroulement de l’installation

Après avoir terminé avec succès le processus d’installation, on doit s’assurer que la plateforme à conteneurs fonctionne correctement. L’équipe de développement fournit un simple conteneur hello-world à cet effet. On **vérifie l’installation de Docker** en entrant la commande suivante dans le terminal Ubuntu et en confirmant avec [ENTER] :

$ sudo docker run hello-world

La commande docker run indique au daemon docker de trouver et de démarrer un conteneur nommé hello-world. Si **l’installation de votre Docker est correctement configurée**, L’on verra apparaitre ce qui correspond à la capture d’écran ci-dessous.



L’interprétation de cette **édition de terminal** est la suivante : pour exécuter la commande hello-world, le démon Docker cherche d’abord dans les fichiers locaux du système pour trouver l’image de conteneur correspondante. Puisque l’on exécute le conteneur hello-world pour la première fois, la recherche du démon est infructueuse. On recevra donc le message « Impossible de trouver l’image » ou « unable to find image » en anglais.

$ sudo docker run hello-world

[sudo] password for osboxes:

Unable to find image 'hello-world:latest' locally

Si Docker ne peut pas trouver une image souhaitée sur le système local, le démon lance un **processus de téléchargement** (pulling) à partir du référentiel Docker.

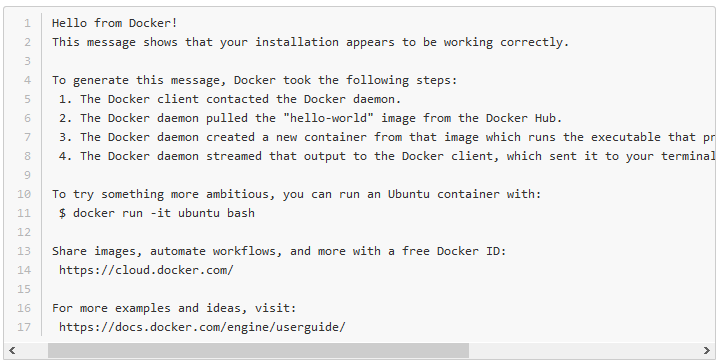
latest: Pulling from library/hello-world

78445dd45222: Pull complete

Digest: sha256:c5515758d4c5e1e838e9cd307f6c6a0d620b5e07e6f927b07d05f6d12a1ac8d7

Status: Downloaded newer image for hello-world:latest

Après un téléchargement réussi, On recevra le message : "Télécharger une nouvelle image pour hello-world:latest". Le **conteneur est alors mis en marche**. Il contient un simple script hello-world avec le message suivant des développeurs :



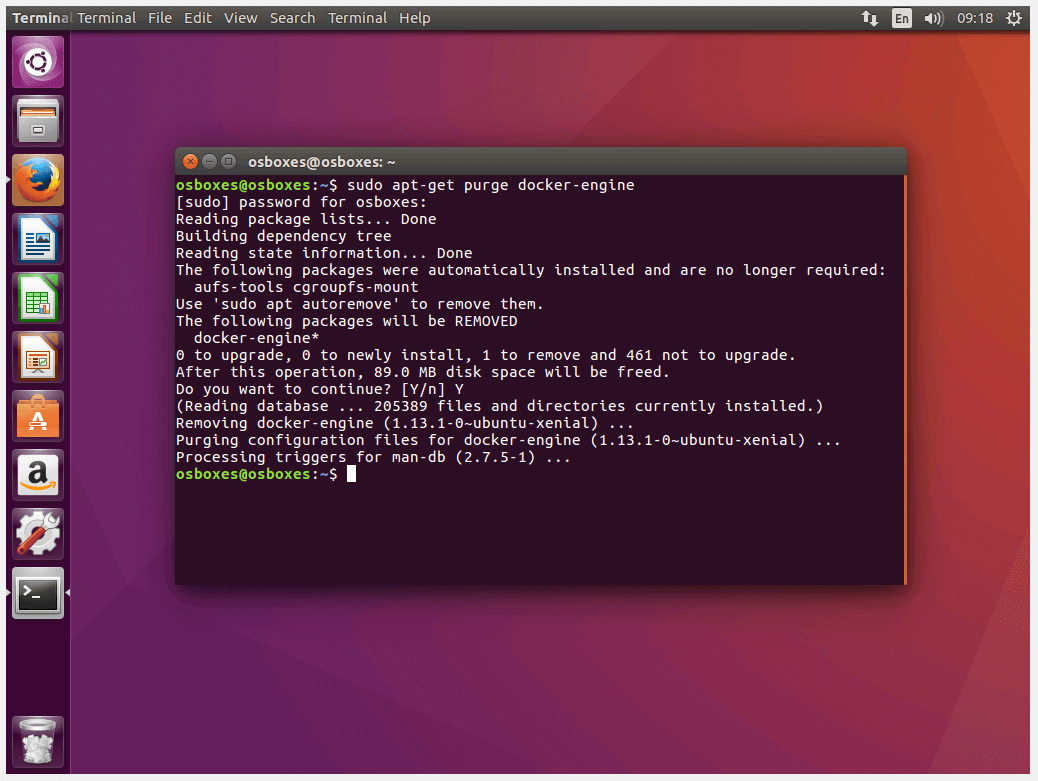
Ce texte signifie simplement : L’installation Docker fonctionne parfaitement.

1.5 Désinstaller Docker

De la même manière que vous installez le moteur Docker sur le terminal, vous pouvez **désinstaller** la plateforme à conteneurs de cette façon. Si vous souhaitez supprimer le paquet Docker de votre système, entrez la commande suivante dans le terminal Ubuntu et confirmez avec [ENTER] :



Une fois que les informations nécessaires à la désinstallation ont été lues, il sera demandé de confirmer à nouveau la commande



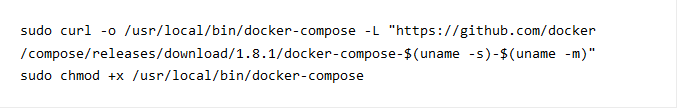
Pour continuer, on entre "Y" (Yes) et appuyez sur [ENTER]. On sélectionne "n" pour annuler la désinstallation. Les images et les conteneurs ne sont **pas automatiquement supprimés** lorsque le moteur Docker est désinstallé. On utilise cette commande pour les supprimer :



Si des **fichiers de configuration** supplémentaires ont été installés, ils doivent être supprimés manuellement.

**III-Installation de Docker Compose**

Pour la réalisation du projet nous allons installer ce nouvel outil docker compose qui ne fait pas parti des outils de base lors de l’installation de Docker



Nous allons pouvoir vérifier que l’installation de docker compose a bien fonctionné en testant une fonction très simple de docker-compose qui permet d’afficher le numéro de version de l’outil installé :

docker-compose -v

IV-Mise en place des microservices

Pour la réalisation du projet, on a besoin de 5 conteneurs :

* Un conteneur NGINX
* Un conteneur MYSQL
* Un conteneur PHP
* Un conteneur PHPMYADMIN
* Un conteneur DNS



* [MySQL](http://www.mysql.com/) *est un* système de gestion de base de données. Son rôle est de stocker les données, sous forme de tables, et de permettre la manipulation de ces données à travers le langage de requête SQL



* PHP est un langage de script. Il permet, de décrire dans une page web, un affichage dynamique d'information, c'est-à-dire que le texte affiché peut dépendre de variables. Nous avons par exemple la possibilité d'afficher dans une page renvoyée au navigateur, la date du jour. Les instructions PHP sont généralement contenues dans des fichiers d'extension php. Ces fichiers peuvent contenir du HTML, entremêlé avec le code PHP. Quand un navigateur demande un tel fichier, le serveur Apache exécute les instructions PHP, qui produisent une page HTML. Une fois la page HTML générée, le serveur la renvoie au navigateur, qui ne voit qu'une page HTML.



NGINX est le serveur web. Son rôle est d'écouter les requêtes émises par les navigateurs (qui demandent des pages web), de chercher la page demandée et de la renvoyer.

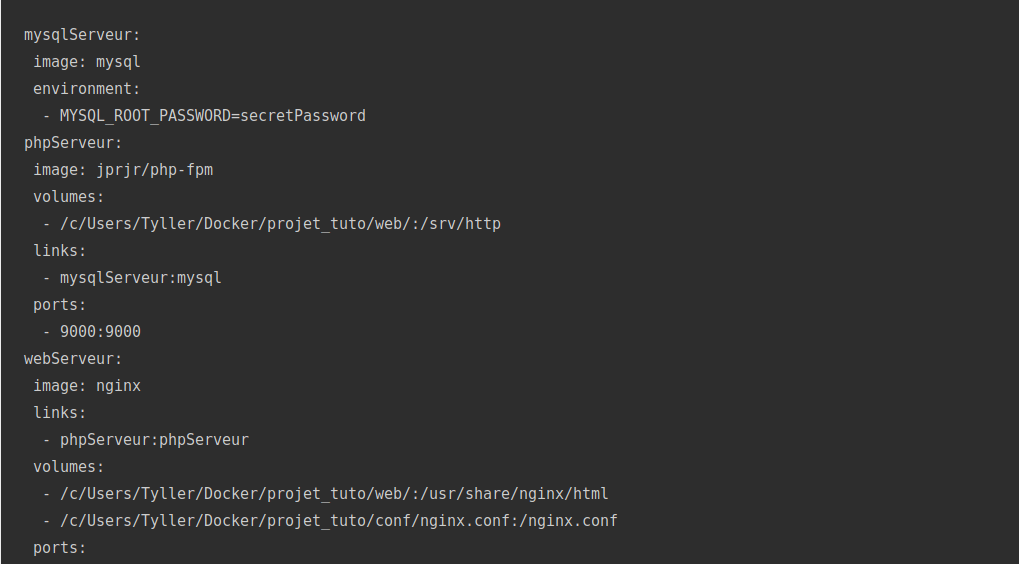


phpMyAdmin est une application web permettant de gérer les bases de données MySQL.



Le serveur DNS (Domain Name System, ou Système de noms de domaine en français) est un service dont la principale fonction est de traduire un nom de domaine en adresse IP. Pour simplifier, le serveurDNS agit comme un annuaire que consulte un ordinateur au moment d'accéder à un autre ordinateur via un réseau.

V.1 Mise en place des conteneurs avec Docker-compose

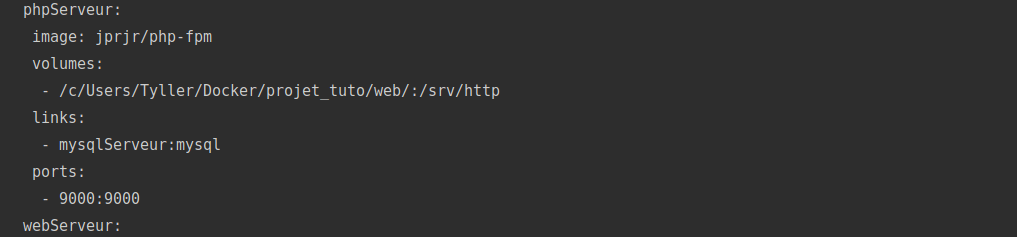


Explication

Mysql



Php

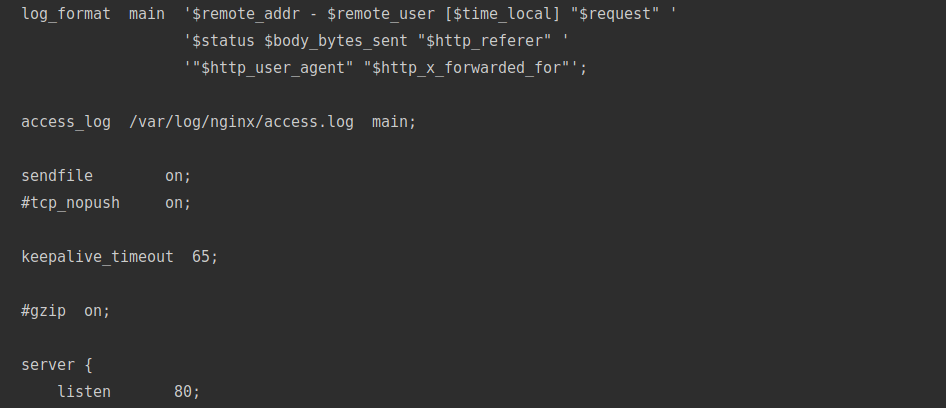


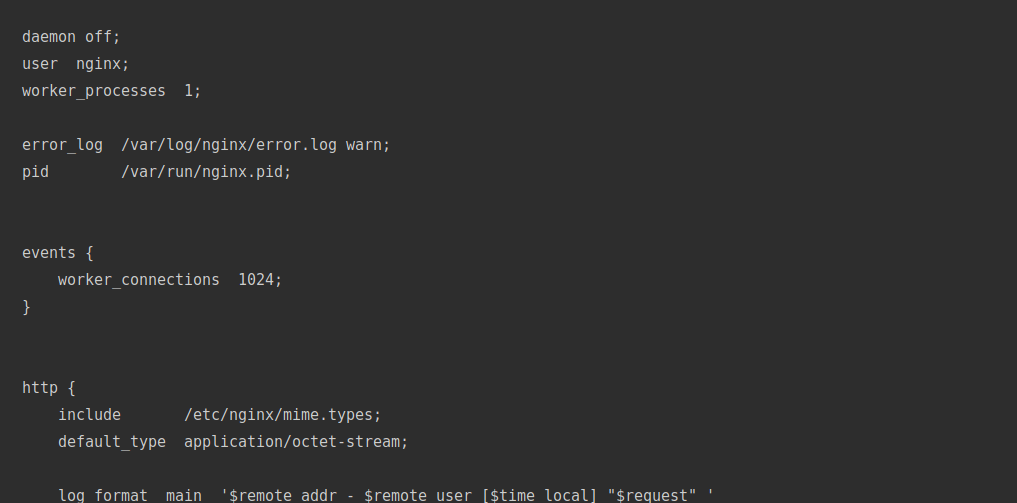
Web

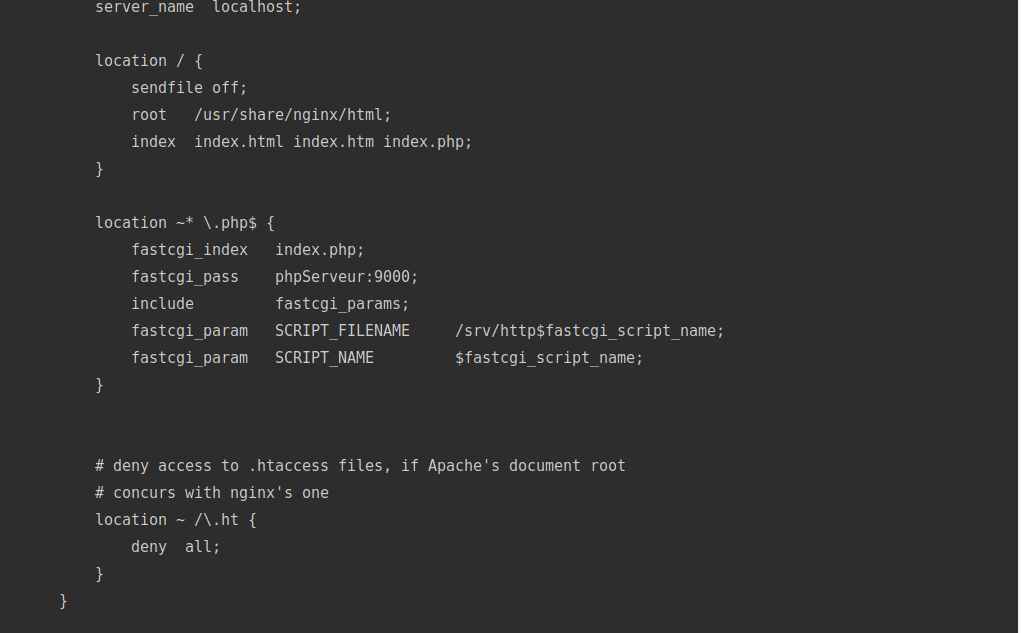


PhpMyadmin



Configuration de Nginix





Références Bibliographique

[1] « Aperçu des alternatives à Docker », *1&1 Digitalguide*. [En ligne]. Disponible sur: https://www.ionos.fr/digitalguide/serveur/know-how/apercu-des-alternatives-a-docker/. [Consulté le: 09-août-2019].

[2] « Docker », 11-sept-2017. .

[3] +Bastien L, « CoreOS : tout savoir sur l’OS Container Linux », *LeBigData.fr*, 12-juin-2018. .

[4] « Introduction à RKT », *Le blog des experts WeScale - DevOps, Cloud et passion*, 23-janv-2017. [En ligne]. Disponible sur: https://blog.wescale.fr/2017/01/23/introduction-a-rkt/. [Consulté le: 21-août-2019].

[5] « lxd [Wiki ubuntu-fr] ». [En ligne]. Disponible sur: https://doc.ubuntu-fr.org/lxd. [Consulté le: 21-août-2019].

[6] « Linux-VServer — Wikilivres ». [En ligne]. Disponible sur: https://fr.wikibooks.org/wiki/Linux-VServer. [Consulté le: 21-août-2019].

[7] +Bastien L, « Docker : tout savoir sur la plateforme de containérisation », *LeBigData.fr*, 14-mai-2019. .

[8] C. Posta, « Microservices for Java Developers », p. 129.

[9] J. DECOOL @jdecool, « La conception pilotée par le domaine », *Jérémy DECOOL (@jdecool), Ingénieur Etudes et Développement à Lyon*, 30-mars-2015. [En ligne]. Disponible sur: https://www.jdecool.fr/blog/2015/03/30/la-conception-pilotee-par-le-domaine.html. [Consulté le: 16-août-2019].

[10] « Que signifie architecture monolithique? - Definition IT de Whatis.fr », *Whatis.com/fr*. [En ligne]. Disponible sur: https://whatis.techtarget.com/fr/definition/architecture-monolithique. [Consulté le: 21-août-2019].

[11] « Les microservices s’apparentent-il plutôt à une architecture SOA ou MVC ? », *LeMagIT*. [En ligne]. Disponible sur: https://www.lemagit.fr/conseil/Les-microservices-sapparentent-il-plutot-a-une-architecture-SOA-ou-MVC. [Consulté le: 16-août-2019].

[12] M. Montantin, « Microservices en 10 min: Concevoir des microservices (partie 1) », *Blog .NET*, 28-juin-2017. .

[13] nishanil, « Communication dans une architecture de microservices ». [En ligne]. Disponible sur: https://docs.microsoft.com/fr-fr/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture. [Consulté le: 21-août-2019].

[14] « Microservices : avantages et inconvénients de la nouvelle structure de serveurs web », *LeMagIT*. [En ligne]. Disponible sur: https://www.lemagit.fr/tribune/Microservices-avantages-et-inconvenients-de-la-nouvelle-structure-de-serveurs-web. [Consulté le: 16-août-2019].

[15] « Comment la conteneurisation informatique accélère le développement des applications - Hello Future Orange », *Hello Future*, 28-janv-2019. [En ligne]. Disponible sur: https://hellofuture.orange.com/fr/comment-la-conteneurisation-informatique-accelere-le-developpement-des-applications/. [Consulté le: 16-août-2019].

[16] « La Conteneurisation », *Phelep Jérémy*, 21-juin-2017. .

[17] « Qu’est-ce que la Conteneurisation et l’Ochestration de Conteneur? - Alibaba Cloud ». [En ligne]. Disponible sur: https://www.alibabacloud.com/fr/knowledge/what-is-containerization. [Consulté le: 16-août-2019].

[18] « apt-get [Wiki ubuntu-fr] ». [En ligne]. Disponible sur: https://doc.ubuntu-fr.org/apt-get. [Consulté le: 19-août-2019].