

Chebihi Fayçal

DOCTEUR CHERCHEUR EN INFORMATIQUE, FORMATEUR, CONSULTANT ET CHEF DE PROJETS EN NOUVELLES TECHNOLOGIES 15 ANS D'EXPÉRIENCES

**Philosophie et Culture DevOps**

**Développement des applications modernes**

**Gestion des versions avec GIT**

**Déploiement et orchestration des conteneurs**

**Les outils d'intégration continue**

**Projet de CI/CD**



# Philosophie et Culture DevOps

# Introduction

---

- imaginez un monde où product owners, Development, IT Operations, et Infosec travaillent ensemble.
- Travaillant vers un objectif commun
- tout en obtenant une stabilité, une fiabilité, une disponibilité et une sécurité de classe mondiale.
- Les équipes interfonctionnelles testent rigoureusement leurs hypothèses sur les fonctionnalités qui raviront le plus les utilisateurs et feront progresser les objectifs organisationnels.
- Cela permet aux organisations de créer un système de travail sûr, où de petites équipes sont capables de développer, tester et déployer rapidement et indépendamment du code et de la valeur
- **Ce sont les résultats qui résultent de DevOps**

# Problématique

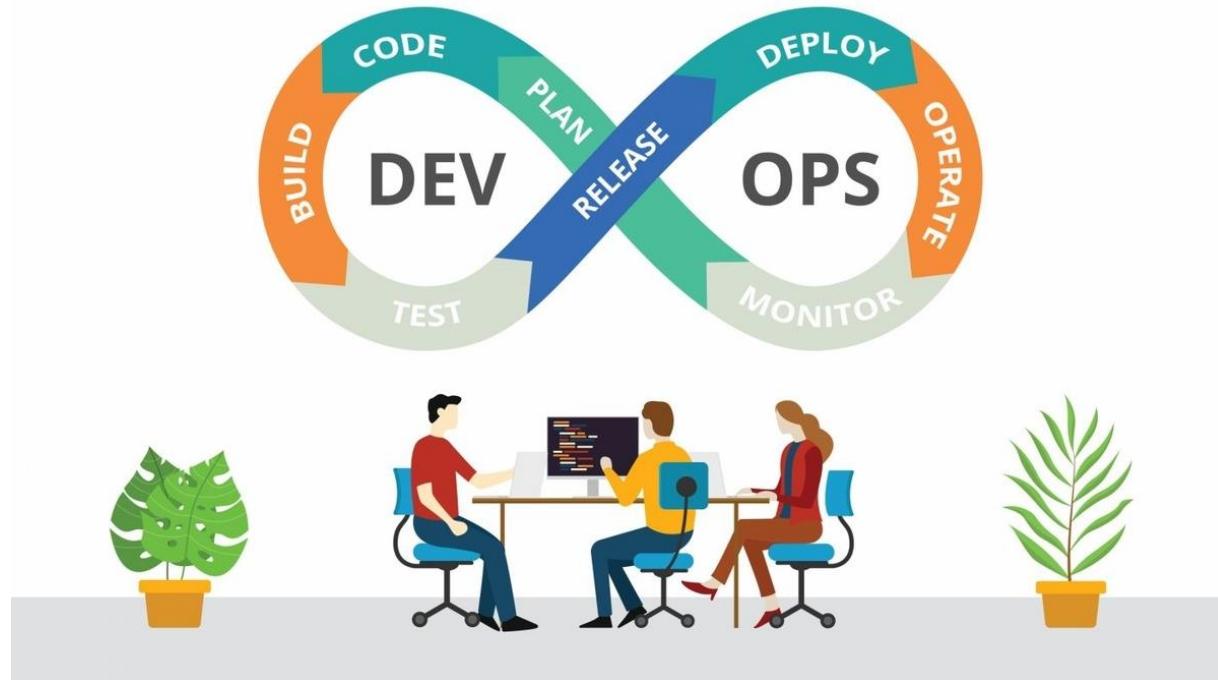
---

- le système dans lequel nous travaillons est en panne => résultats extrêmement médiocres
- le développement et les opérations informatiques sont des adversaires
- les tests et les activités Infosec n'ont lieu qu'à la fin d'un projet
- trop d'efforts manuels et trop de transferts, nous laissant toujours en attente
- => Délais extrêmement longs + qualité de notre travail + déploiements de production chaotique
- impacts négatifs sur nos clients et notre entreprise

# Solution ?

---

- Changer notre façon de travailler (**DevOps**)



# Historique

---

- la révolution manufacturière des années 1980 (pratiques **Lean**)
  - les délais moyens de commande des usines de fabrication étaient de six semaines
  - moins de 70 % des commandes expédiées à temps
  - 2005, avec la mise en œuvre généralisée des pratiques Lean
  - les délais moyens de livraison des produits étaient tombés à moins de trois semaines
  - plus de 90 % des commandes expédiées à temps
- Dans les années 2000 (Principes et pratiques **agiles**)
  - le temps nécessaire pour développer de nouvelles fonctionnalités était tombé à des semaines ou des mois.
  - mais le déploiement en production nécessitait encore des semaines ou des mois
  - souvent avec des conséquences catastrophiques. résultats

# Historique

---

- Avec l'introduction de DevOps (2010) + banalisation matériel (Cloud)
- Le déploiement est finalement devenu routinier et à faible risque
- Rapidement déployées en production en quelques heures ou quelques minutes
- Effectuer des expériences pour tester des idées commerciales (feedback)
- Découvrant quelles idées créent le plus de valeur pour les clients et l'organisation dans son ensemble
- Ensuite développées en fonctionnalités pouvant être déployées rapidement et en toute sécurité en production.
- Aujourd'hui, les organisations qui adoptent souvent les principes et pratiques DevOps **déployer des changements des centaines voire des milliers de fois par jour**

# Historique

	1970s–1980s	1990s	2000s–Present
Era	Mainframes	Client/Server	Commoditization and Cloud
Representative technology of era	COBOL, DB2 on MVS, etc.	C++, Oracle, Solaris, etc.	Java, MySQL, Red Hat, Ruby on Rails, PHP, etc.
Cycle time	1–5 years	3–12 months	2–12 weeks
Cost	\$1M–\$100M	\$100k–\$10M	\$10k–\$1M
At risk	The whole company	A product line or division	A product feature
Cost of failure	Bankruptcy, sell the company, massive layoffs	Revenue miss, CIO's job	Negligible

# Le problème : Quelque chose dans votre organisation doit avoir besoin **Amélioration**

---

- La plupart des organisations ne sont pas en mesure de déployer des changements de production en quelques minutes ou heures.
- Ils ne sont pas non plus en mesure de déployer des centaines ou des milliers de modifications en production par jour
- ils ont du mal à se déployer mensuellement ou même trimestriellement
- Les déploiements de production ne sont pas non plus routiniers, impliquant plutôt des pannes et des combats chroniques et héroïques

# The Core, Chronic Conflict

---

- Conflit entre le développement et les opérations informatiques
- Entraînant des délais de mise sur le marché toujours plus lents pour les nouveaux produits et fonctionnalités.
- la dette technique (**technical debt**) décrit comment les décisions que nous prenons conduisent à des problèmes qui deviennent de plus en plus difficiles à résoudre au fil du temps.
- les objectifs souvent **concurrents** du développement et des opérations informatiques.
  - développement : Répondre à l'évolution rapide du paysage concurrentiel
  - opérations : Fournir un service stable, fiable et sécurisé au client

# Downward Spiral in Three Acts

---

- Le premier acte (opérations): objectif est de maintenir le fonctionnement des applications et de l'infrastructure afin que notre organisation puisse offrir de la valeur aux clients.
- Le deuxième acte : compenser des promesses non tenue
- Troisième acte : tout devient un peu plus difficile
  - tout le monde devient un peu plus occupé
  - le travail prend un peu plus de temps
  - les communications deviennent un peu plus lentes
  - les files d'attente de travail deviennent un peu plus difficiles. plus long
  - de petites actions provoquent de plus gros échecs
- => nos cycles de livraison de produits continuent d'évoluer de plus en plus lentement
- => les retours sur le travail de chacun deviennent plus lents et plus faibles
- => nous ne sommes pas non plus en mesure de fournir un service stable et fiable à nos clients

# DevOps : Briser la spirale descendante

---

- Petites équipes de développeurs implémentent indépendamment leurs fonctionnalités
- Valident leur exactitude dans des environnements de type production
- Les déploiements de code sont routiniers et prévisibles
- les déploiements se produisent tout au long de la journée ouvrable
- les opérations informatiques travaillent pendant les heures normales
- En créant des boucles de rétroaction rapides à chaque étape du processus, chacun peut voir immédiatement les effets de ses actions.
- Chaque fois que des modifications sont validées dans le contrôle de version, des tests automatisés rapides sont exécutés dans des environnements de type production
- Garantissant en permanence que le code et les environnements fonctionnent comme prévu et sont toujours dans un état sécurisé et déployable.

# DevOps : Briser la spirale descendante

---

- Les tests automatisés aident les développeurs à découvrir rapidement leurs erreurs (quelques minutes).
- La télémétrie de production omniprésente dans nos environnements de code et de production garantit que les problèmes sont détectés et corrigés rapidement.
- => tout le monde se sent productif
- => l'architecture permet aux petites équipes de travailler en toute sécurité
- => architecturalement découplées du travail des autres équipes
- => versions de produits et de fonctionnalités de haut niveau deviennent routinières
- techniques de lancement sombres : déploiement invisible pour tout le monde sauf les employés internes => **tester et de faire évoluer la fonctionnalité jusqu'à ce qu'elle atteigne l'objectif commercial souhaité**

# DevOps ?

---

- DevOps peut être considérée comme :
- dérivée du Lean !!!
- de la théorie des contraintes et du mouvement Toyota Kata !!!
- comme la suite logique du parcours logiciel Agile qui a commencé en 2001 !!!

# The Lean Movement

---

- Des techniques telles que la cartographie des flux de valeur, les tableaux kanban et la maintenance productive totale ont été codifiées pour le système de production Toyota dans les années 1980. En 1997.
- Les principes Lean se concentrent sur la façon de créer de la valeur pour le client grâce à la pensée systémique:
  - créant une constance dans l'objectif
  - adoptant la pensée scientifique
  - créant un flux et une traction
  - assurant la qualité à la source
  - dirigeant avec humilité et en respectant chaque individu

# The Agile Manifesto

---

- Le Manifeste Agile a été créé en 2001 par dix-sept experts
- Ils voulaient créer un ensemble de valeurs et de principes qui capturent l'avantage de ces méthodes plus adaptatives, par rapport aux processus de développement de logiciels tels que le développement en cascade et les méthodologies telles que le Rational United Process, ...
- L'un des principes clés était : livrer fréquemment des logiciels fonctionnels, de quelques semaines à quelques mois, avec une préférence pour les délais plus courts.
- Agile est reconnu pour avoir considérablement augmenté la productivité et la réactivité de nombreuses organisations de développement.
- de nombreux moments clés de l'histoire de DevOps se sont également produits au sein de la communauté Agile ou lors de conférences Agile

# Le mouvement de livraison continue

---

- S'appuyer sur la discipline de développement de la construction, du test et de l'intégration continus.
- Garantir que le code et l'infrastructure sont toujours dans un état déployable
- Le code enregistré dans le tronc peut être déployé en toute sécurité en production
- Cette idée a été présentée pour la première fois lors de la conférence Agile de 2006

# Toyota Kata

---

- Gérer les gens pour l'amélioration, l'adaptabilité et les résultats supérieurs.
- la communauté Lean a raté le plus important pratique de tous => **kata** d'amélioration
- Chaque organisation a des routines de travail et que le kata d'amélioration nécessite de créer une structure pour la pratique quotidienne et habituelle du travail.
- la pratique quotidienne améliore les résultats
- définition des résultats cibles sur une cadence et d'amélioration continue du travail quotidien

# La chaîne de valeur de technologique

---

- Processus requis pour convertir une hypothèse commerciale en un service ou une fonctionnalité reposant sur la technologie qui apporte de la valeur à l'entreprise.
1. L'entrée de processus est la formulation d'un objectif commercial
  2. Les équipes de développement qui suivent un processus Agile ou itératif typique transformeront cette idée en spécification de fonctionnalité.
  3. Implémentée dans le code dans l'application ou le service en cours de construction
  4. Le code est ensuite vérifié dans le référentiel de contrôle de version
  5. Chaque modification est intégrée et testée avec le reste du système logiciel

# Focus sur le délai de déploiement

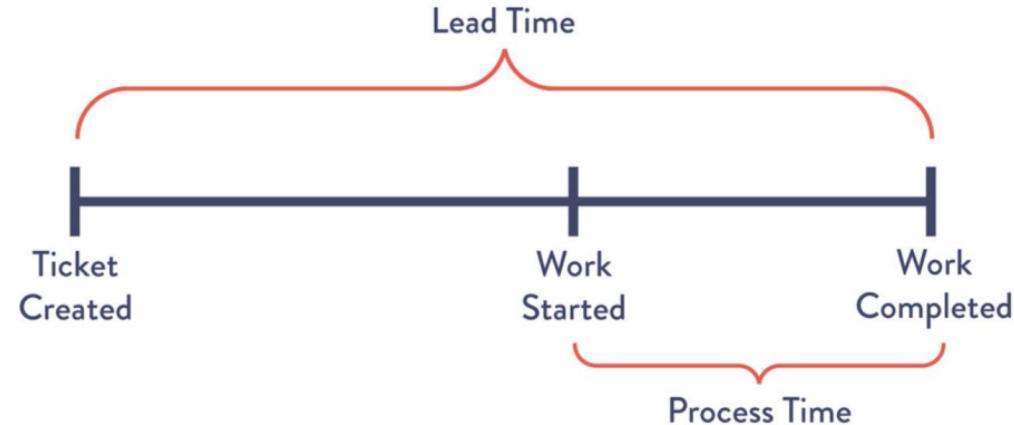
---

- flux de Valeur : commence lorsqu'un ingénieur vérifie un changement dans le contrôle de version et se termine lorsque ce changement est exécuté avec succès en production.
  - apportant de la valeur au client et générant des commentaires et une télémétrie utiles.
- La première phase de travail qui comprend la conception et le développement
  - nécessitant souvent des degrés élevés de créativité
  - une grande variabilité des temps de traitement
- la deuxième phase du travail, qui comprend les tests, le déploiement et les opérations (Lean Manufacturing)
  - prévisible et mécaniste
  - obtenir des résultats de travail avec une variabilité minimale

# Délai de déploiement

---

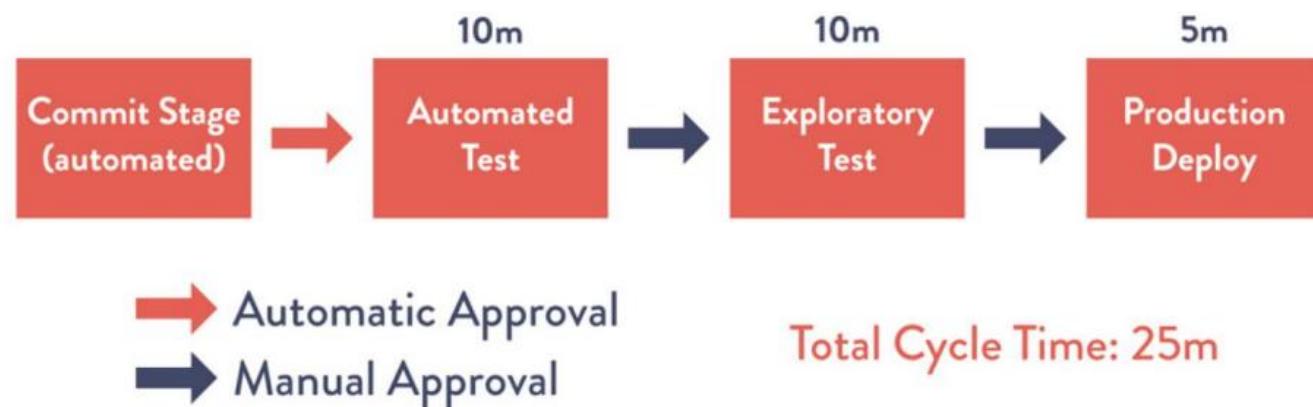
- le déploiement et les opérations se déroulent simultanément avec la conception/le développement => permettant un flux rapide et de haute qualité.
- Cette méthode réussit lorsque nous travaillons par petits lots et intégrons la qualité dans chaque partie de notre flux de valeur.
- Dans la communauté Lean, le délai d'exécution est l'une des deux mesures couramment utilisées pour mesurer les performances dans les flux de valeur, l'autre étant le temps de traitement



# Délai de déploiement

---

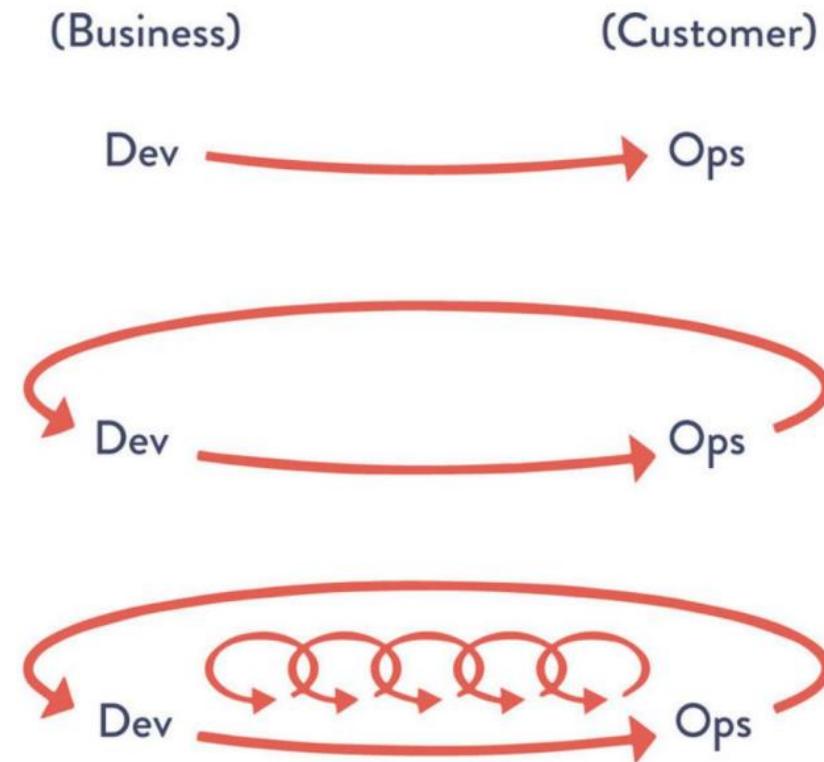
- Dans l'idéal DevOps : des délais de déploiement de quelques minutes
- les développeurs reçoivent des commentaires rapides et constants sur leur travail
- => implémenter, d'intégrer et de valider rapidement et indépendamment leur code
- => déployer le code dans l'environnement de production



# The Three Ways

---

- ensembles des principes sous-jacents à partir desquels tous les comportements et modèles DevOps observés sont dérivés



# First Way : Flow

---

- Permet un flux de travail rapide de gauche à droite du développement aux opérations jusqu'au client.
- En accélérant le flux à travers le **flux de valeur** technologique nous réduisons le délai requis pour répondre à toutes les demandes internes ou clients.
- Les pratiques qui en résultent incluent :
  - Des processus de construction, d'intégration, de test et de déploiement continus
  - La création d'environnements à la demande **IaC**
  - la limitation des travaux en cours (**WIP**)
  - la création de systèmes et d'organisations pouvant être modifiés en toute sécurité

# Second Way : Feedback

---

- permet le flux rapide et constant de rétroaction de droite à gauche à toutes les étapes de notre flux de valeur.
- Amplifier les commentaires pour éviter que les problèmes ne se reproduisent
- Détection et une récupération plus rapides
- => nous créons de la qualité à la source et générerons ou intégrerons des connaissances là où elles sont nécessaires.
- Cela nous permet de créer des systèmes de travail toujours plus sûrs
- Cela maximise les opportunités pour notre organisation d'apprendre et de s'améliorer

# Third Way : Continual Learning and Experimentation

---

- Permet la création d'une culture génératrice de confiance élevée qui soutient une approche dynamique disciplinée et scientifique de :
  - l'expérimentation
  - la prise de risques
- => Nous créons des systèmes de travail toujours plus sûrs
- => Prendre risques et réaliser des expériences qui nous aident à apprendre plus vite que nos concurrents et à gagner sur le marché
- => nous concevons également notre système de travail de manière à pouvoir multiplier les effets des nouvelles connaissances
- transformant les découvertes locales en améliorations globales

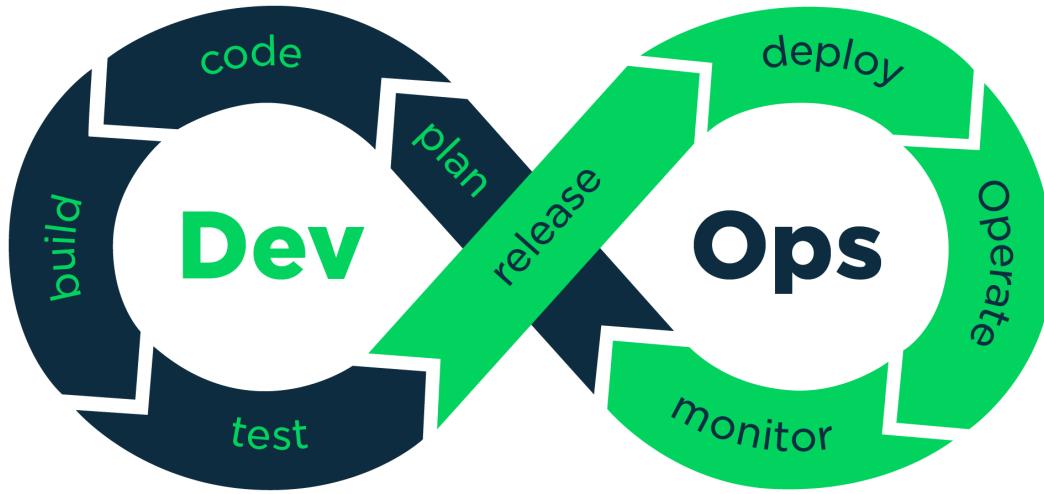


---

# Développement des applications modernes

---

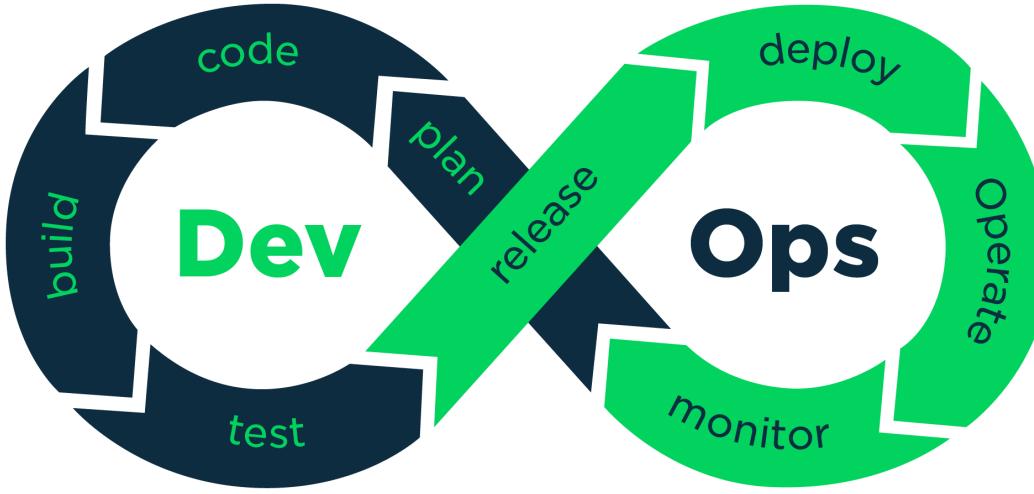
- DevOps ?
- D'Agile vers DevOps
- L'architecture à microservices
- Les Middleware
- Les plateformes Cloud
- Les concepts de CI/CD



---

# DevOps ?

---

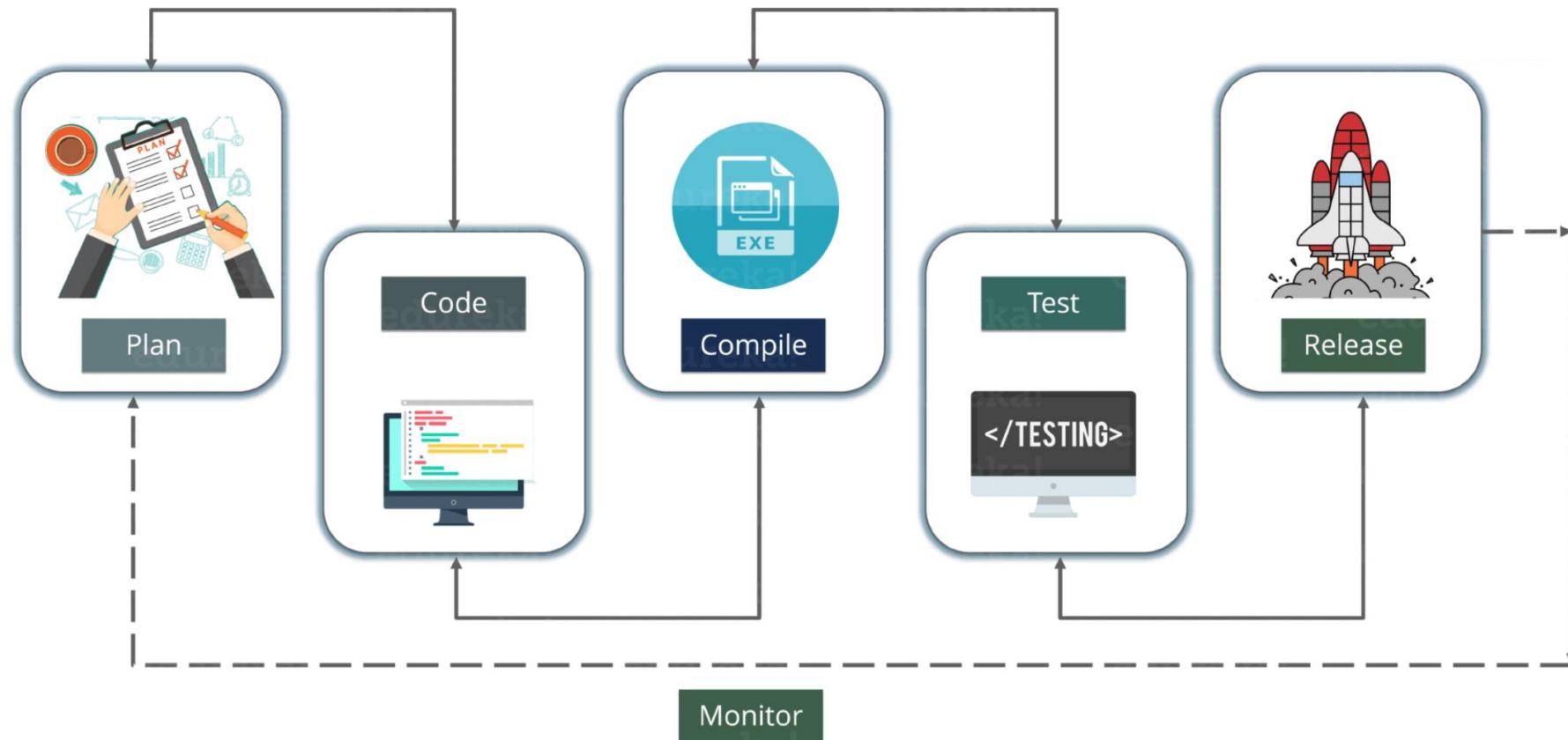


---

# Pourquoi apprendre DevOps

---

# Vue à 360 degrés du SDLC (Software Development Life Cycle)



# Devenir plus précieux pour l'entreprise :

---

- La plupart des entreprises à des fins d'optimisation des coûts recherchent des personnes possédant une grande variété de compétences.
- Avec DevOps, vous devenez plus précieux pour l'entreprise, car vous connaissez divers outils et technologies utilisés pour le développement, les tests et le déploiement.



# Séparez-vous de la foule

---

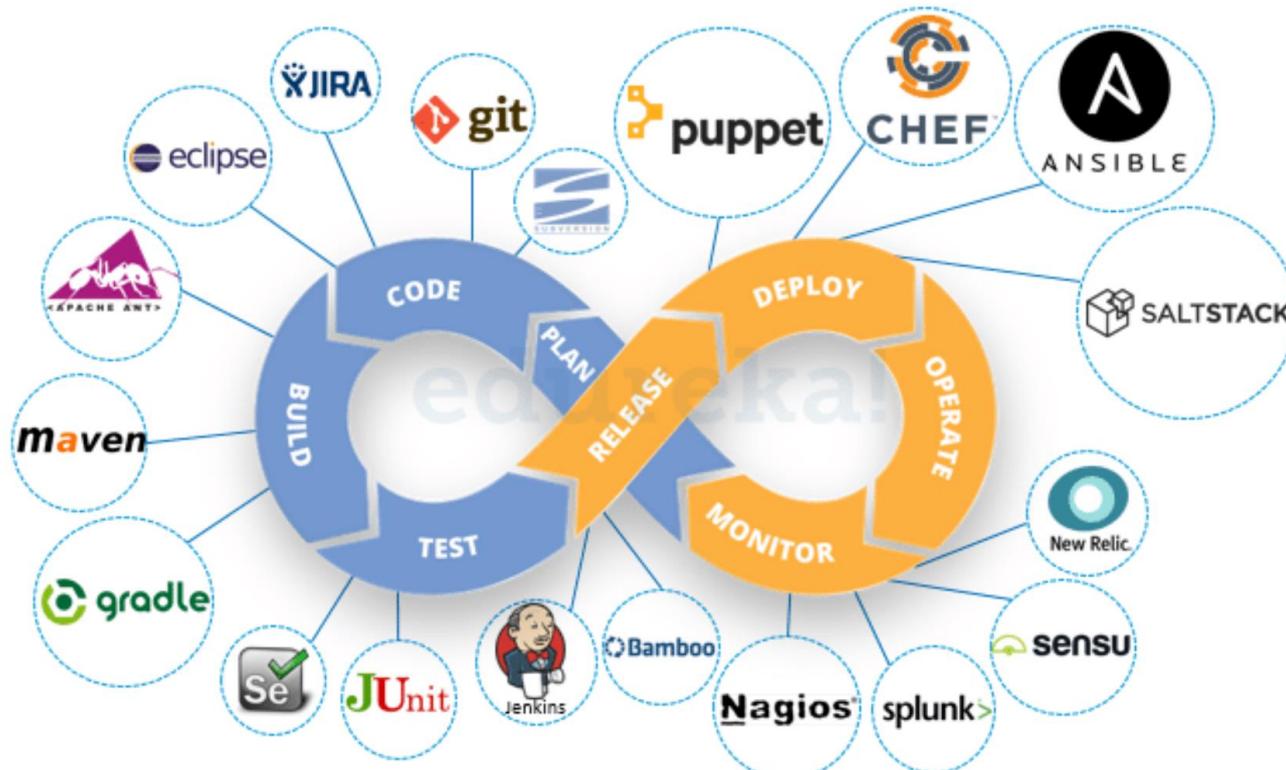
➤ Avec les connaissances DevOps, vous pouvez offrir quelque chose d'unique à n'importe quelle organisation. Cela vous donne un avantage sur les autres personnes lors d'un entretien.



# Exposition à divers outils et technologies tendance

---

- DevOps implique différentes phases, et pour chaque phase, plusieurs outils sont disponibles.



# Des versions plus rapides

---

- DevOps rend le processus SDLC vraiment agile, ce qui garantit des livraisons dans le temps.
- Les organisations peuvent analyser assez rapidement le comportement des utilisateurs et intégrer ces changements dans la prochaine version.
- Cela donne aux organisations un avantage sur leurs concurrents et les utilisateurs obtiennent un meilleur produit.
- Cela se produit en raison des différentes phases impliquées dans DevOps et des multiples outils disponibles.
- Cela permet la livraison continue et parfois même le déploiement continu.
- Chez Amazon, les ingénieurs déplacent du code toutes les 11,7 secondes en moyenne.



# Croissance de carrière rapide

---

- La mise à niveau est une nécessité, surtout de nos jours où la technologie évolue à un rythme rapide
- Vous devez devenir plus précieux pour l'organisation, c'est là que DevOps peut jouer un rôle très vital. Cela peut augmenter votre croissance de carrière.
- Vous pouvez devenir Release Manager, Project Manager, Automation Architect ou même DevOps Evangelist.



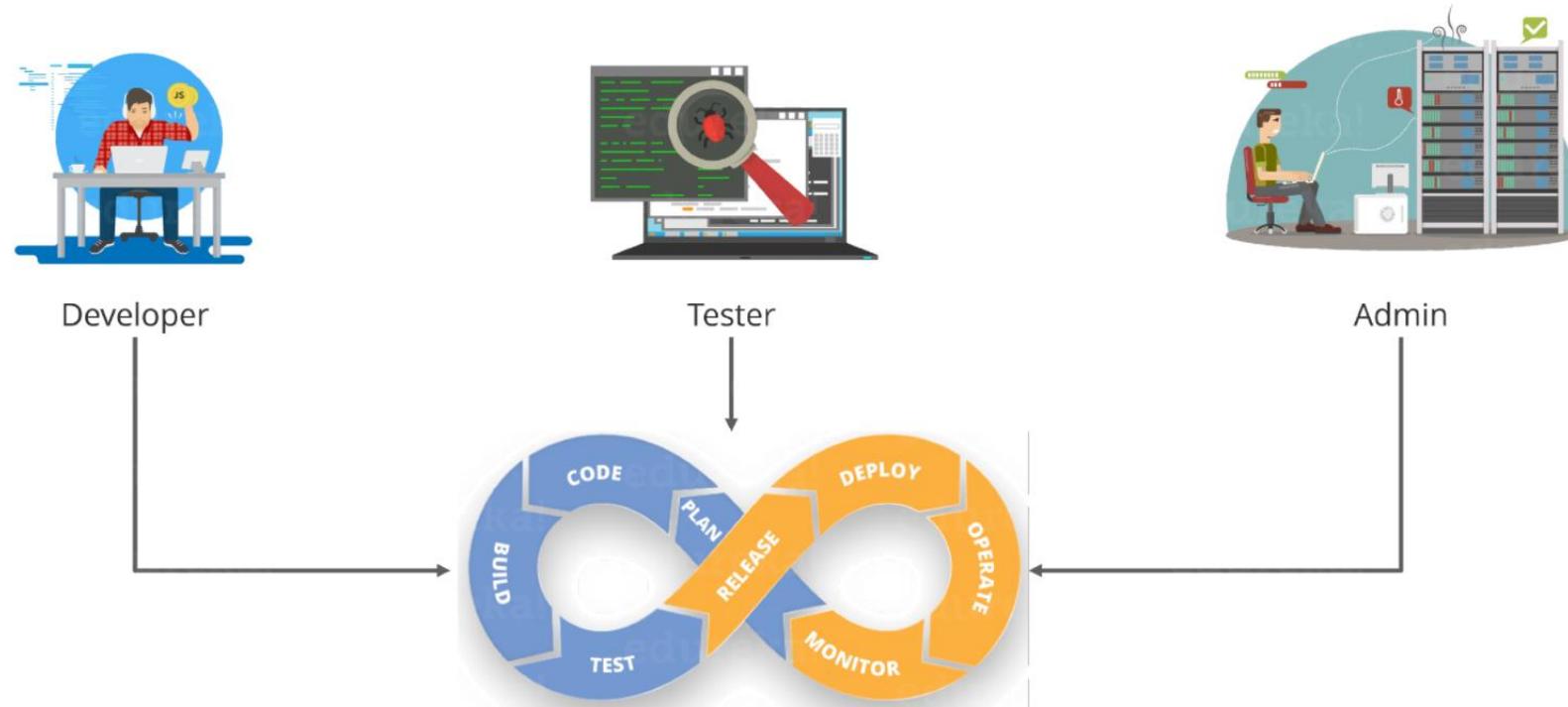
# Facile à obtenir un emploi

---

- Il y a beaucoup de demandes de professionnels DevOps, mais actuellement il n'y a pas assez de personnes pour répondre aux exigences souhaitées pour les rôles et responsabilités DevOps.
- Cela laisse une énorme fenêtre d'opportunités pour quiconque cherche à se démarquer et DevOps peut être une bonne opportunité de carrière pour eux.
- DevOps offre les avantages dont nous avons tous besoin sur le marché actuel et une personne qui est bonne dans ce domaine sera certainement très demandée et jouira d'une carrière fructueuse.
- Cela augmente vos chances d'être embauché facilement, à condition que vous ayez une bonne connaissance pratique des divers outils et technologies DevOps.

# Tout le monde peut apprendre DevOps

- Des personnes d'horizons différents peuvent apprendre DevOps.
- Même un débutant avec des connaissances de base de Linux et un langage de script peut apprendre DevOps.



# Qu'est-ce que DevOps ?

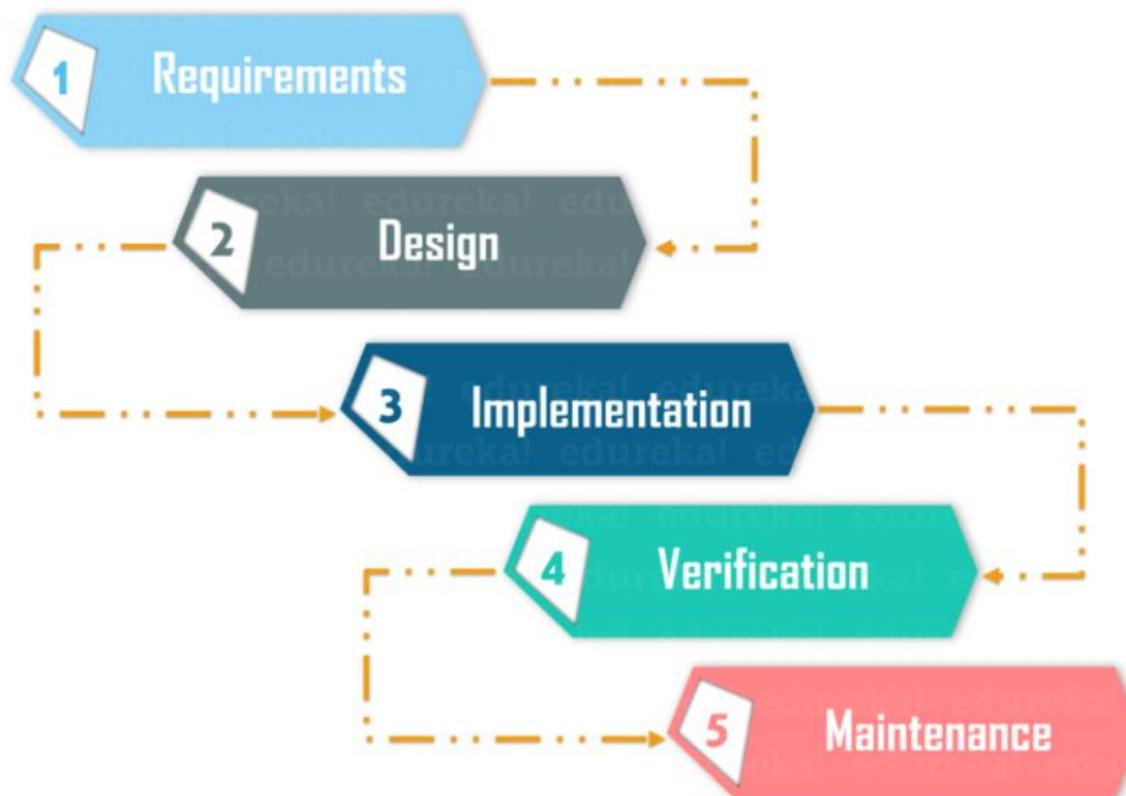
---

- ❑ Le terme DevOps est une combinaison de deux mots, à savoir Développement et Opérations. DevOps est une pratique qui permet à une seule équipe de gérer l'ensemble du cycle de vie du développement d'applications, c'est-à-dire le développement, les tests, le déploiement et la surveillance.
- ❑ L'objectif ultime de DevOps est de réduire la durée du cycle de développement du système tout en fournissant des fonctionnalités, des correctifs et des mises à jour fréquemment en étroite synchronisation avec les objectifs de l'entreprise.
- ❑ DevOps est une approche de développement logiciel à l'aide de laquelle vous pouvez développer des logiciels de qualité supérieure rapidement et avec plus de fiabilité. Il se compose de diverses étapes telles que le développement continu, l'intégration continue, les tests continus, le déploiement continu et la surveillance continue.

# Approches de développement logiciel

---

□ Modèle de cascade:



# Approches de développement logiciel

## Méthodologie Agile



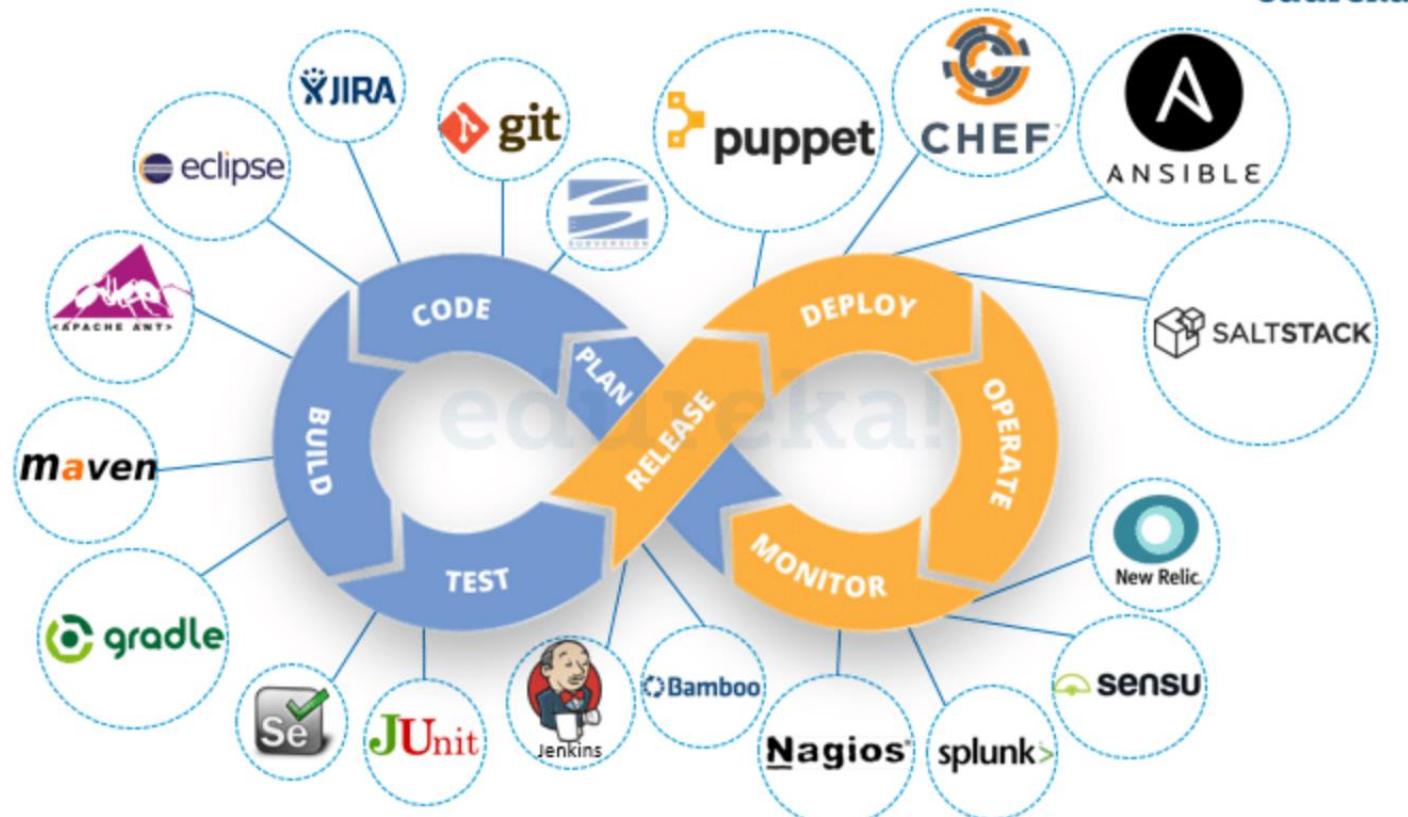
# Méthodologie Agile

---

- ❑ En Agile, une entreprise publie l'application avec des fonctionnalités hautement prioritaires dans la première itération.
- ❑ Après sa sortie, les utilisateurs finaux ou les clients vous donnent leur avis sur les performances de l'application.
- ❑ Ensuite, vous apportez les modifications nécessaires à l'application ainsi que de nouvelles fonctionnalités et l'application est à nouveau publiée, ce qui correspond à la deuxième itération.
- ❑ Vous répétez toute cette procédure jusqu'à ce que vous obteniez la qualité logicielle souhaitée.

# Étapes et outils DevOps

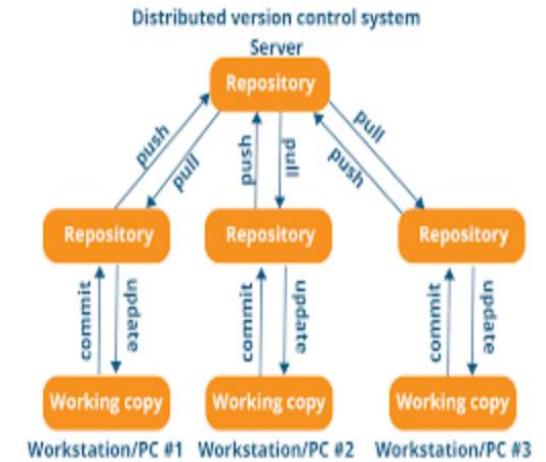
- Comme mentionné précédemment, les différentes étapes telles que le développement continu, l'intégration continue, les tests continus, le déploiement continu et la surveillance continue constituent le cycle de vie DevOps. Voyons maintenant chacune des étapes du cycle de vie DevOps une par une.



# Développement continu (Continuous Development )

---

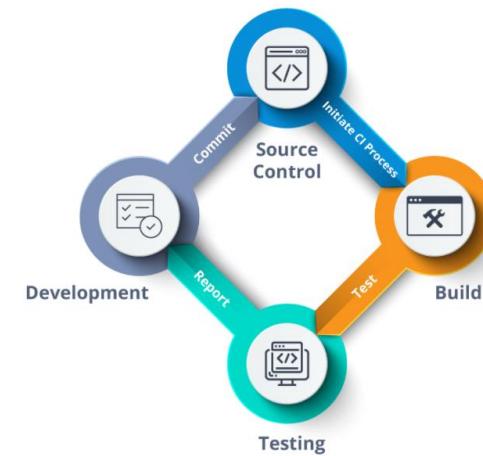
- ❑ Outils : **Git, SVN, Mercurial, CVS**
- ❑ C'est la phase qui implique la « planification » et le « codage » du logiciel.
- ❑ Ici, aucun outil DevOps n'est requis pour la planification, mais il existe un certain nombre d'outils pour maintenir le code.
- ❑ Le code peut être dans n'importe quelle langue, mais vous le maintenez à l'aide des outils de contrôle de version.
- ❑ Ce processus de maintenance du code est connu sous le nom de « Source Code Management »
- ❑ Une fois le code développé, vous passez à la phase d'intégration continue.



# Intégration continue (Continuous Integration)

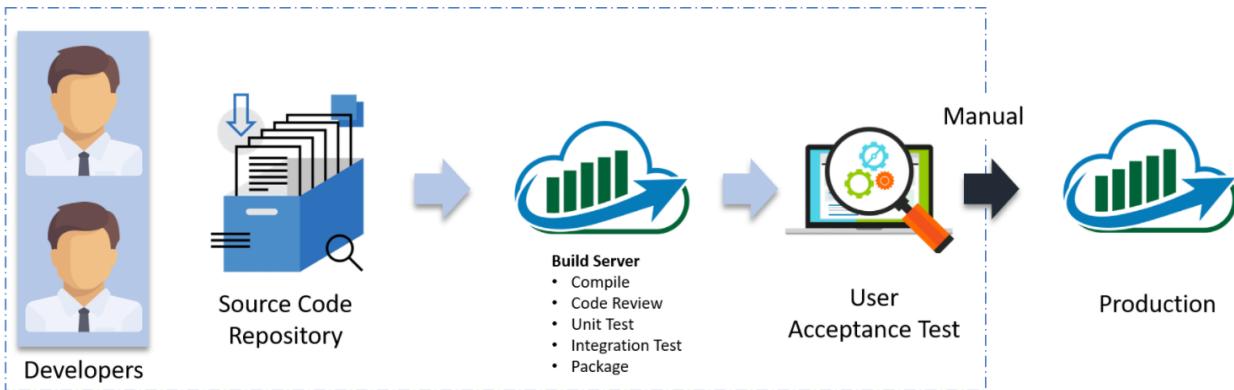
---

- Outils: **Jenkins, TeamCity, Travis**
- Cette étape est au cœur de l'ensemble du cycle de vie DevOps.
- C'est une pratique dans laquelle les développeurs ont besoin de valider(commit) les modifications du code source plus fréquemment.
- Vous construisez ensuite chaque commit et cela permet une détection précoce des problèmes s'ils sont présents.
- Le code de construction implique non seulement la compilation, mais également la revue de code, les tests unitaires, les tests d'intégration et l'empaquetage.
- Le code prenant en charge les nouvelles fonctionnalités est continuellement intégré au code existant
- Dans cette étape, vous utilisez les outils de construction/empaquetage du code dans un fichier exécutable afin de pouvoir le transmettre aux phases suivantes.



# Tests continus (Continuous Testing)

- ❑ Outils: Jenkins, Selenium TestNG, Junit

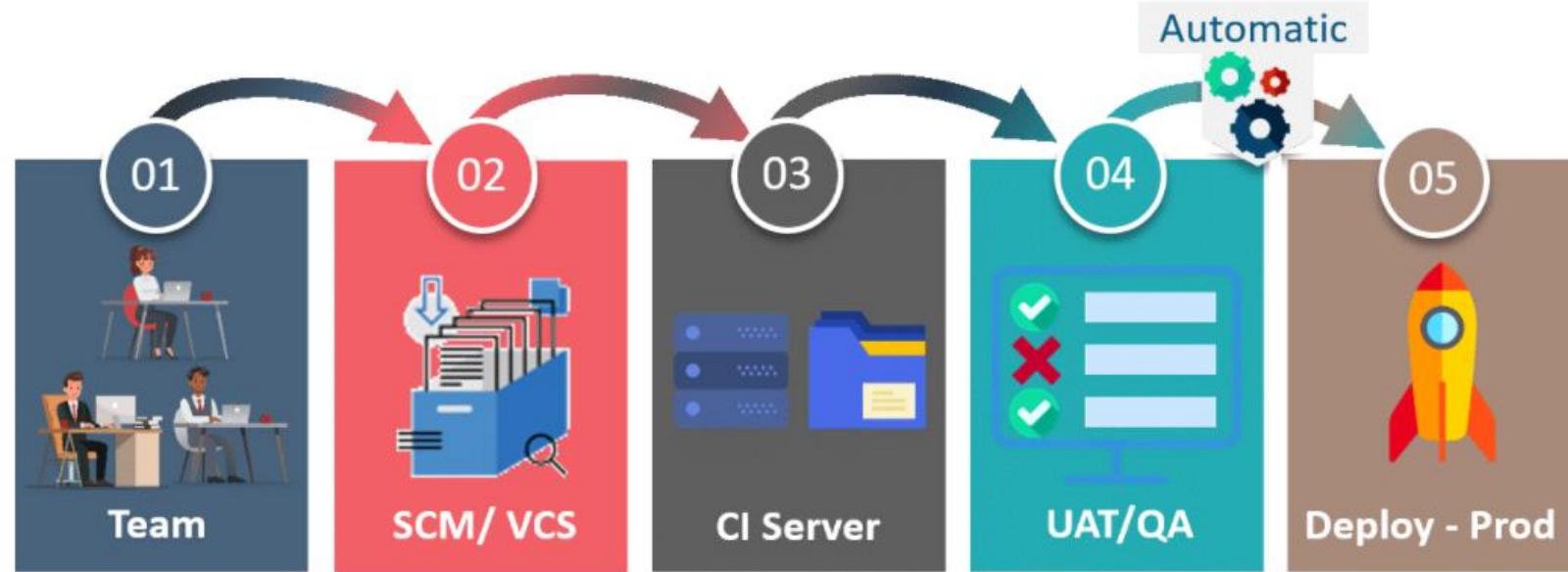


- ❑ Selenium est utilisé pour les tests d'automatisation et les rapports sont générés par TestNG. Vous pouvez automatiser toute cette phase de test à l'aide d'un outil d'intégration continue appelé Jenkins.
- ❑ Vous pouvez écrire le code Selenium en Java pour tester votre application (maven,ant).
- ❑ L'ensemble de ce processus peut être automatisé à l'aide de Jenkins.

# Déploiement continu (Continuous Deployment)

## ❑ Outils:

- ✓ Configuration Management – Chef, Puppet, Ansible
- ✓ Containerization – Docker, Vagrant



# Déploiement continu (Continuous Deployment)

---

## ❑ Configuration Management:

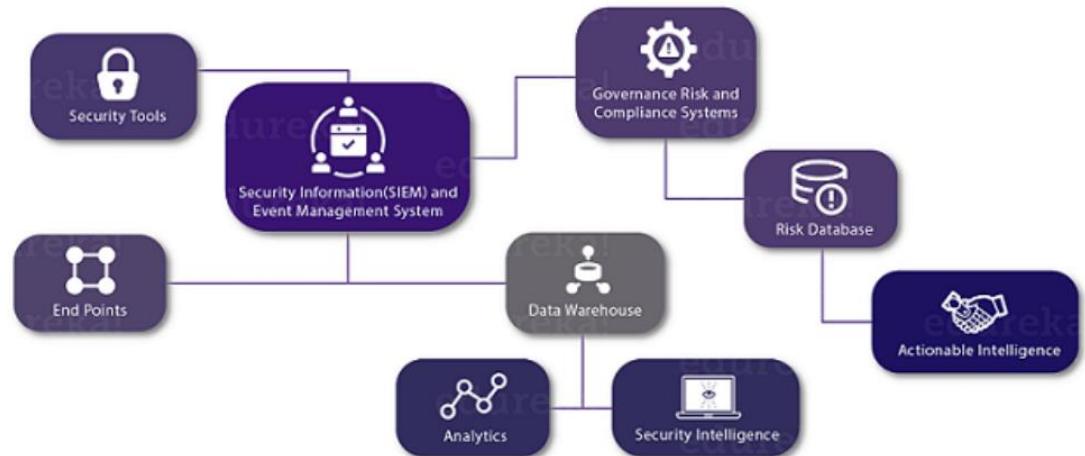
- ❑ C'est l'acte d'établir et de maintenir la cohérence des exigences fonctionnelles et des performances d'une application.
- ❑ c'est l'acte de libérer les déploiements sur les serveurs, de planifier les mises à jour sur tous les serveurs et, surtout, de maintenir les configurations cohérentes sur tous les serveurs.

## ❑ Containerization :

- ❑ Les outils de conteneurisation aident à produire une cohérence entre les environnements de développement, de test, de Dev ainsi que de production
- ❑ Ils aident également à augmenter et à réduire rapidement le nombre des instances.

# Contrôle continu (Continuous Monitoring)

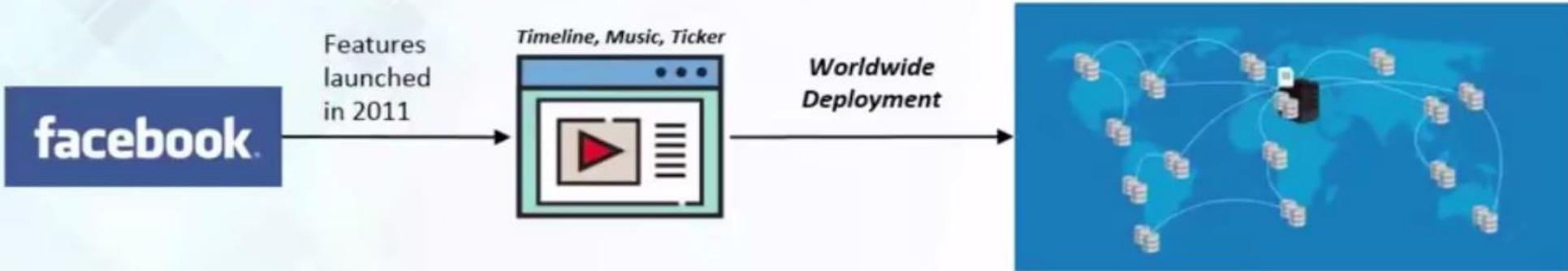
- ❑ Outils : Splunk, ELK Stack, Nagios, New Relic
- ❑ Surveillez en permanence les performances de votre application
- ❑ Enregistrez des informations vitales sur l'utilisation du logiciel
- ❑ Traitez ensuite ces informations pour vérifier le bon fonctionnement de l'application
- ❑ Résolvez les erreurs système telles que la mémoire insuffisante, le serveur inaccessible, etc



# Qu'est-ce qu'un ingénieur DevOps ?

---

- ❑ Un ingénieur DevOps est une personne qui comprend le cycle de vie du développement logiciel et qui a une compréhension directe de divers outils d'automatisation pour le développement de pipelines numériques (pipelines CI/CD).
- ❑ Un ingénieur DevOps travaille avec les développeurs et le personnel informatique pour superviser les versions de code. Ce sont soit des développeurs qui s'intéressent au déploiement et aux opérations réseau, soit des administrateurs système qui ont une passion pour les scripts et le codage et qui passent au développement où ils peuvent améliorer la planification des tests et du déploiement.



### Challenges they faced that day

- Features released to 500 million users → Heavy Website traffic → Server Meltdown
- Mixed responses from users which lead to no conclusion

# Cas d'utilisation 2011 : FaceBook

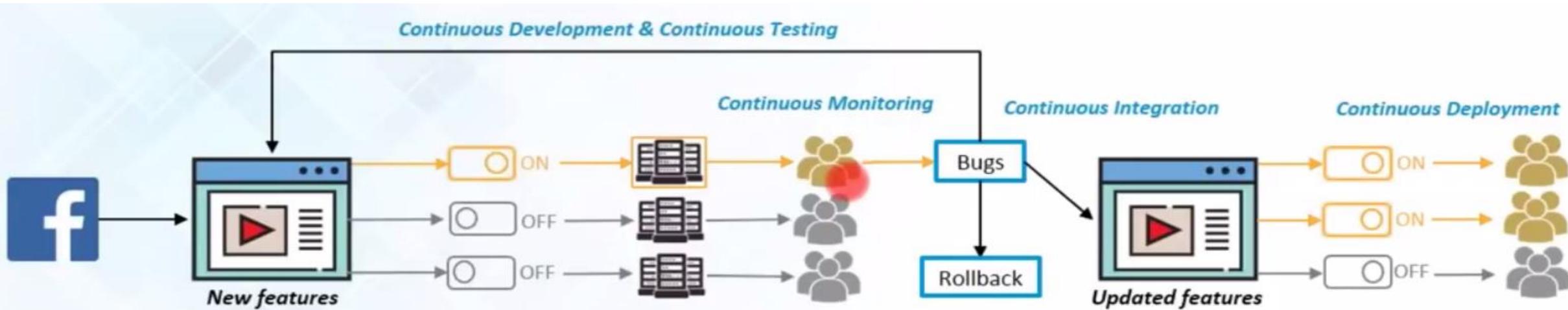
**Post This They Came Up With  
The Dark Launching Technique**

**FaceBook propose Dark Launching**

### **According to Dark Launching Technique:**

- The new features are first deployed on a smaller & specific user base.
- They are continuously monitored and the feedbacks are continuously developed and tested.
- Once the features are stable, they are deployed on other user bases in multiple releases.

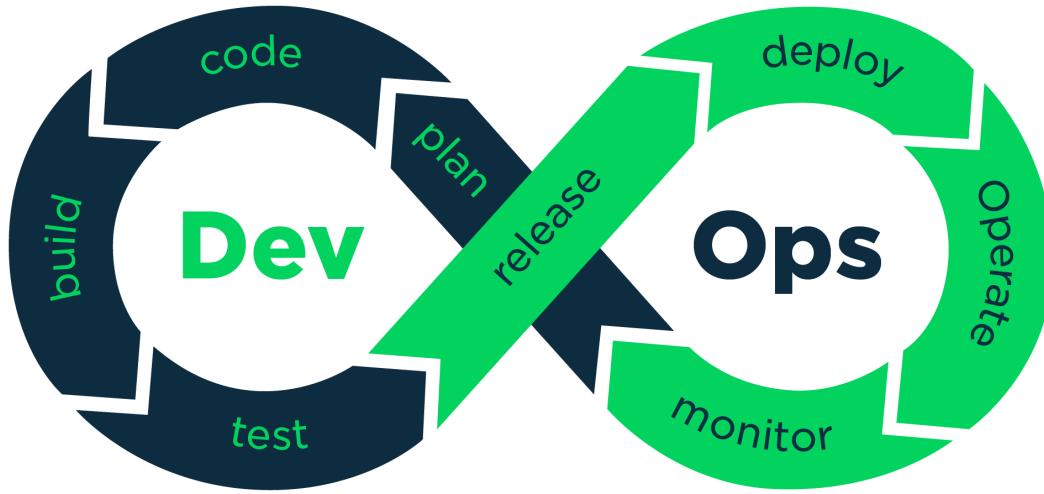
# The Dark Launching Techinique



To Implement Dark launching, the below activities are fundamental as they lie at the heart of the DevOps lifecycle:

- Continuous Development
- Continuous Testing
- Continuous Integration
- Continuous Deployment
- Continuous Monitoring

# The Dark Launching Techinique



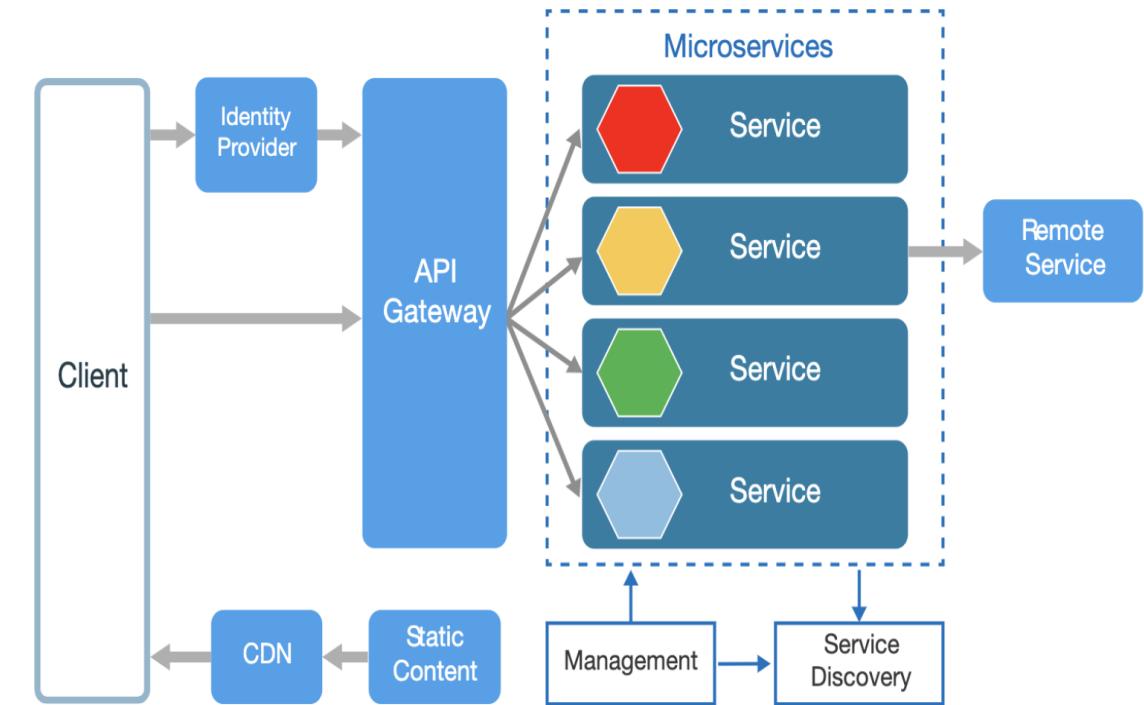
---

# Architectures Micro services

---

# Architectures Micro services

- Développez l'intégralité de votre application dans de petits composants indépendants et faiblement couplés.
- Chaque micro-service doit répondre à une seule exigence métier.
- Les fonctionnalités d'une grande application sont séparées en petits composants. (regardez image)
- Les composants sont développées indépendamment et communiqueront les unes avec les autres sur des API privées bien définies.



# Micro-services vs monolithique

---

- Une application monolithique n'est qu'une seule application (où tout est réuni)
- Une application de micro-service, sont de nombreuses applications minuscules qui travaillent ensemble.
- Avantages du micro-service :
  - ❑ Les développeurs peuvent travailler indépendamment sur différents composants.
  - ❑ **Résistance** : Si un service tombe en panne, il ne supprimera pas l'intégralité de l'application.
  - ❑ Les services peuvent être mis à l'échelle indépendamment.
  - ❑ Déploiements indépendants.
  - ❑ **La flexibilité** : Différents langages / frameworks peuvent être utilisés pour différents composants.

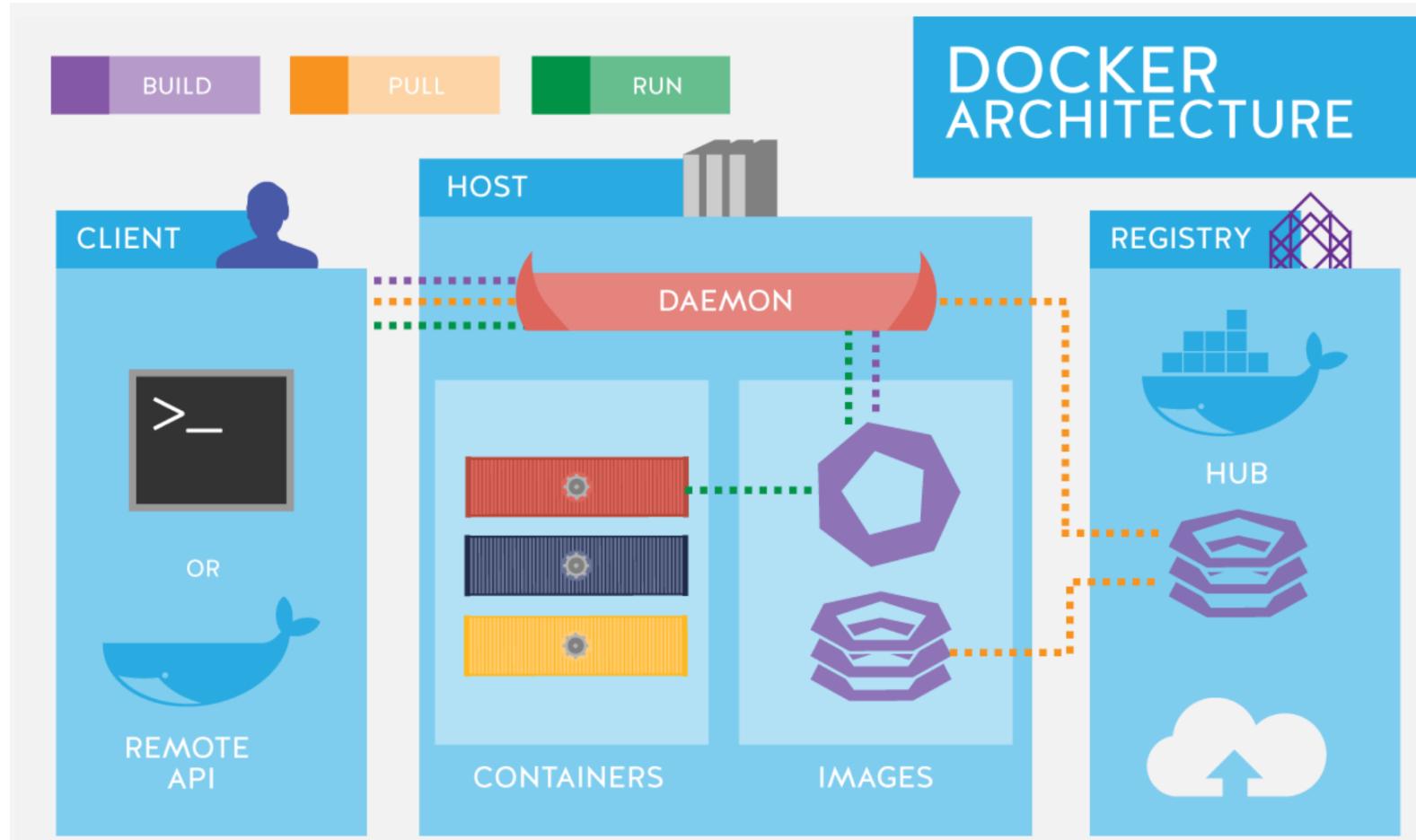
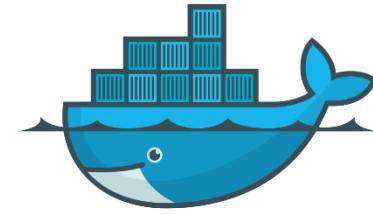
# Micro-services vs monolithique

---

## ➤ Inconvénients du micro-service :

- ❑ Le débogage devient un peu plus difficile. Lorsque vous travaillez sur le service A, vous devrez peut-être ouvrir le service B pour corriger les problèmes.
- ❑ **Tests** : Les outils existants ne sont pas nécessairement conçus pour fonctionner avec des dépendances de service.
- ❑ **Latence** : Pendant que les services communiquent entre eux.
- ❑ **Bases de données distribuées** : avoir différentes tables dans différentes bases de données pourrait être un défi. Pourrait être un mérite pour certains.

# Architectures Docker



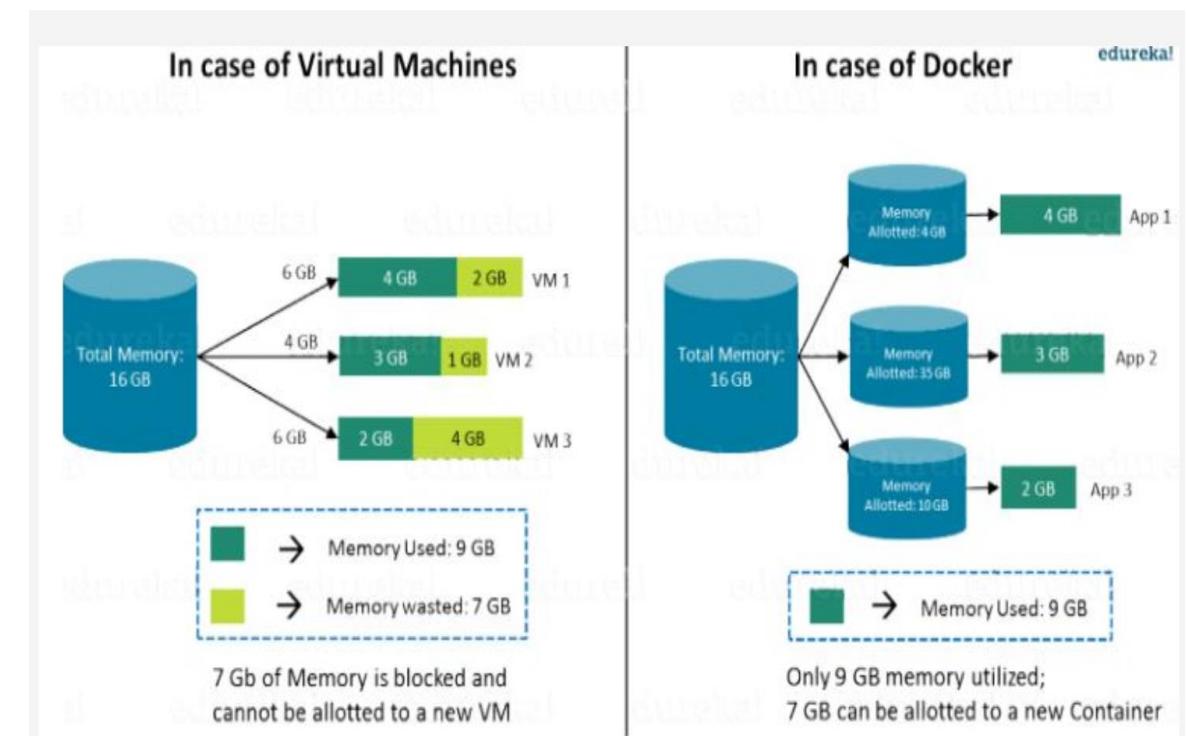
# Conteneurs Docker

---

- Un conteneur est une unité logicielle standard qui conditionne une application avec toutes ses dépendances et bibliothèques.
- Cela facilite le déploiement d'applications dans des conteneurs, quelle que soit la configuration matérielle ou le système d'exploitation.
- Contrairement aux machines virtuelles, les conteneurs utilisent le système d'exploitation hôte plutôt que de regrouper un système d'exploitation avec l'application.
- Il fournit un écosystème complet d'outils pour créer des images de conteneurs, extraire ces images de registres connus, déployer des conteneurs et gérer ou orchestrer des conteneurs en cours d'exécution sur un cluster de machines.
- Docker a popularisé l'adoption des conteneurs. Même l'expression "dockerize mon application" est devenue courante.

# Avantages de l'utilisation de Docker

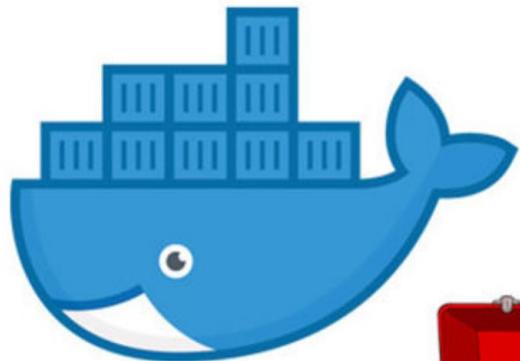
- Docker provides a consistent runtime across all phases of a product cycle (development, tests et deployment).
- Docker est open source. Il y a une liberté de choix puisque tout type d'application (hérité, cloud natif, monolithique, 12 facteurs) peut s'exécuter dans un conteneur Docker.
- La sécurité est intégrée au moteur Docker par défaut. Il est alimenté par certains des meilleurs composants tels que le *containerd*



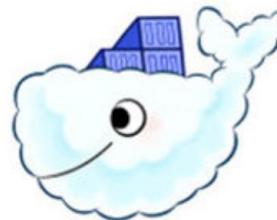
# Les composants de l'écosystème Docker?

---

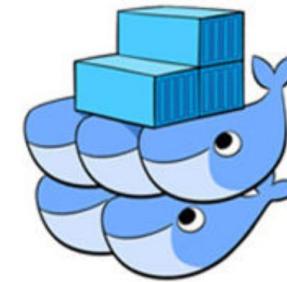
Docker Engine



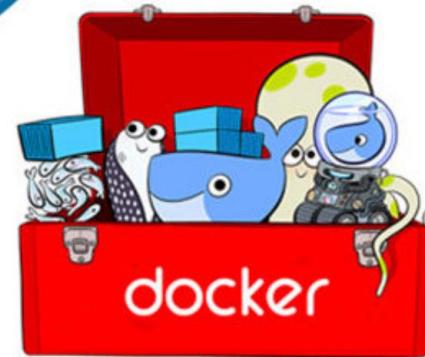
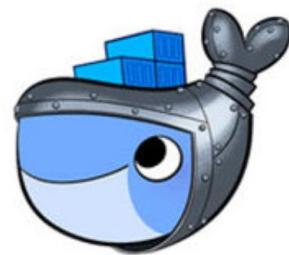
Docker Cloud



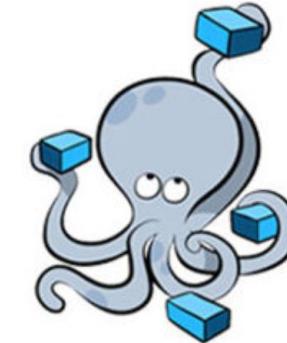
Docker Swarm



Docker Machine



Docker Compose





---

# Gestion des versions avec GIT

---

# GIT ?

---

- ❑ Git est un outil de système de contrôle de version distribué gratuit et open source.
- ❑ conçu pour tout gérer, des petits aux très grands projets, avec rapidité et efficacité.
- ❑ Il a été créé par Linus Torvalds en 2005 pour développer le noyau Linux.
- ❑ Git possède les fonctionnalités, les performances, la sécurité et la flexibilité dont la plupart des équipes et des développeurs individuels ont besoin.
- ❑ Il sert également d'outil DevOps de contrôle de version distribué important.

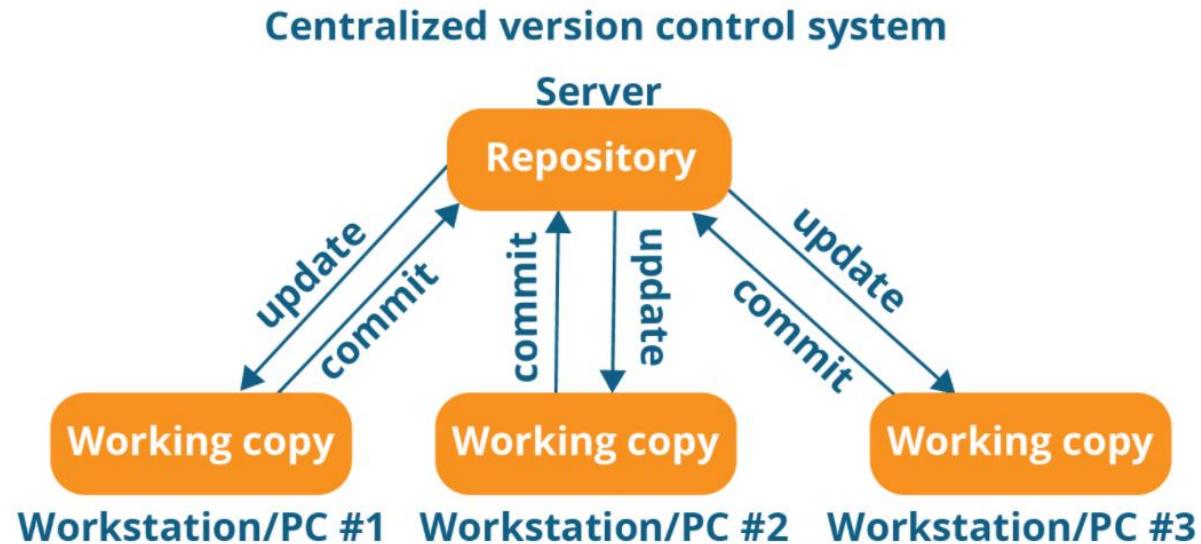
# Quel est le but de Git ?

---

- ❑ Git est principalement utilisé pour gérer votre projet, comprenant un ensemble de fichiers de code/texte susceptibles de changer.
- ❑ Le contrôle de version est la gestion des modifications apportées aux documents, aux programmes informatiques, aux grands sites Web et à d'autres collectes d'informations.
- ❑ There are two types of VCS:
  - ❑ Système de contrôle de version centralisé (CVCS)
  - ❑ Distributed Version Control System (DVCS)

# VCS centralisé

- Le système de contrôle de version centralisé (CVCS) utilise un serveur central pour stocker tous les fichiers et permet la collaboration en équipe.
- Il fonctionne sur un référentiel unique auquel les utilisateurs peuvent accéder directement à un serveur central.



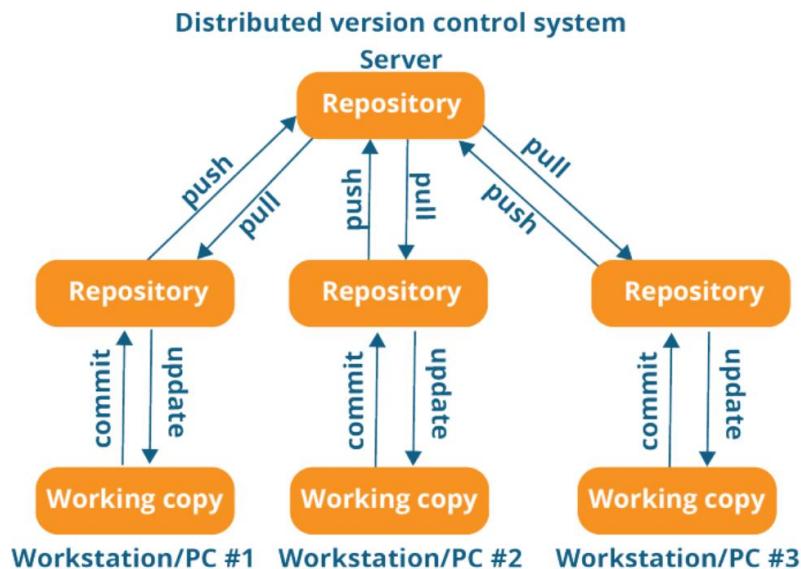
# VCS centralisé

---

- Chaque programmeur peut extraire ou mettre à jour ses postes de travail avec les données présentes dans le référentiel ou peut apporter des modifications aux données ou valider dans le référentiel.
- Même s'il semble assez pratique de maintenir un seul référentiel, il présente des inconvénients majeurs. Certains d'entre eux sont:
  - ✓ Il n'est pas disponible localement ; ce qui signifie que vous devez toujours être connecté à un réseau pour effectuer une action.
  - ✓ Étant donné que tout est centralisé, dans tous les cas, le plantage ou la corruption du serveur central entraînera la perte de toutes les données du projet

# VCS distribué

- ❑ Ces systèmes ne reposent pas nécessairement sur un serveur central pour stocker toutes les versions d'un fichier de projet.
- ❑ Chaque contributeur a une copie locale ou « clone » du référentiel principal
- ❑ Chacun maintient son propre référentiel local qui contient tous les fichiers et métadonnées présents dans le référentiel principal.



# VCS distribué

---

- ❑ Chaque programmeur gère lui-même un référentiel local, qui est en fait la copie ou le clone du référentiel central sur son disque dur
- ❑ Ils peuvent valider et mettre à jour leur référentiel local sans aucune interférence.
- ❑ Ils peuvent mettre à jour leurs référentiels locaux avec de nouvelles données du serveur central par une opération appelée « pull » et affecter les modifications apportées au référentiel principal par une opération appelée « pousser » à partir de leur référentiel local.
- ❑ Avantages:
  - ❑ Toutes les opérations (sauf push & pull) sont très rapides car l'outil n'a besoin d'accéder qu'au disque dur.
  - ❑ La validation de nouveaux ensembles de modifications peut être effectuée localement sans manipuler les données du référentiel principal
  - ❑ Étant donné que chaque contributeur dispose d'une copie complète du référentiel du projet, ils peuvent partager les modifications les uns avec les autres s'ils souhaitent obtenir des commentaires avant d'affecter les modifications dans le référentiel principal.
  - ❑ Si le serveur central tombe en panne à tout moment, les données perdues peuvent être facilement récupérées à partir de l'un des référentiels locaux du contributeur

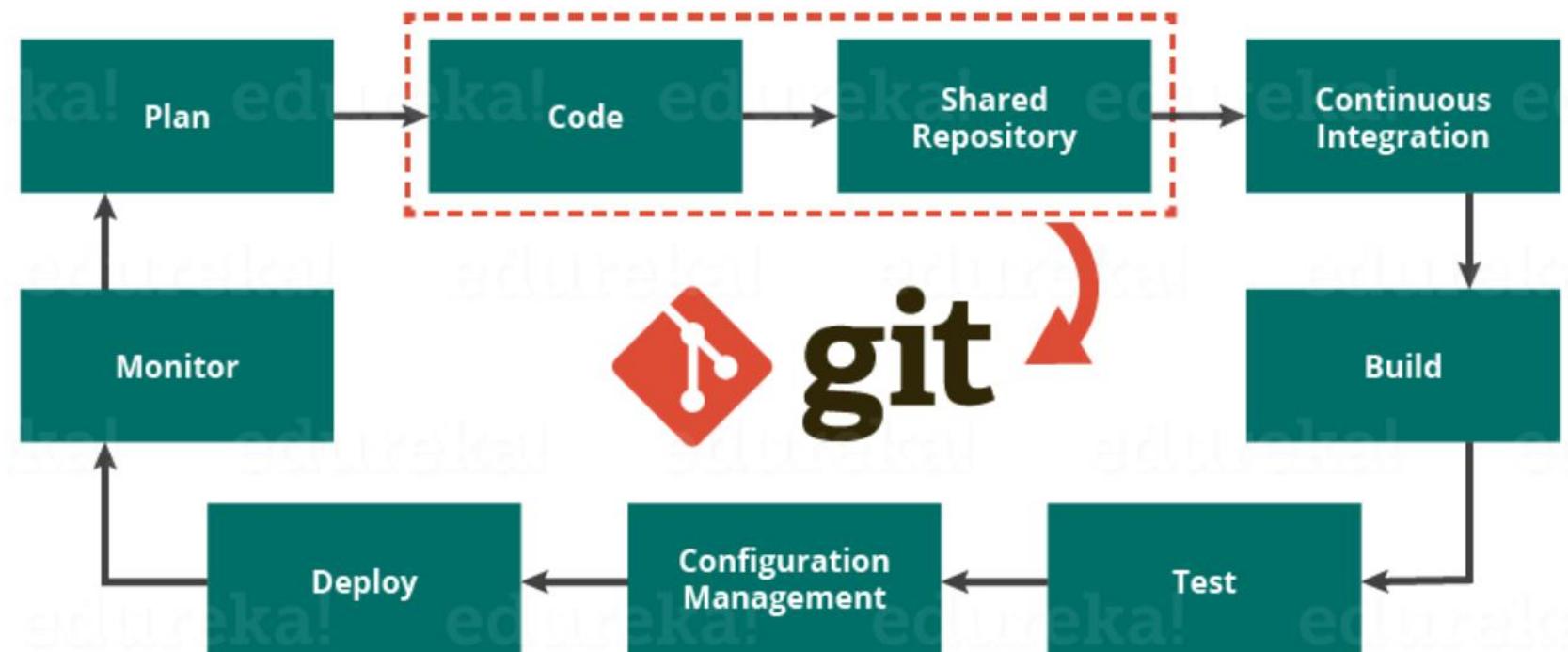
# Qu'est-ce que Git ?

---

- ❑ Git est un outil de contrôle de version distribué qui prend en charge les flux de travail non linéaires distribués en fournissant une assurance de données pour le développement de logiciels de qualité
- ❑ Git fournit toutes les fonctionnalités du VCS distribué à l'utilisateur mentionné précédemment
- ❑ Les dépôts Git sont très faciles à trouver et à accéder
- ❑ Pourquoi Git :
  - ❑ Gratuit et open source
  - ❑ Rapide
  - ❑ Évolutive
  - ❑ Fiable
  - ❑ Sécurise (clé SHA1)
  - ❑ Économique
  - ❑ Prend en charge le développement non linéaire (branchement et fusion)

# Rôle de Git dans DevOps

- DevOps est la pratique consistant à apporter de l'agilité au processus de développement et aux opérations.



# Entreprises populaires utilisant Git

---

- ❑ Git a gagné en popularité par rapport aux autres outils de contrôle de version disponibles sur le marché comme Apache Subversion (SVN), Concurrent Version Systems (CVS), Mercurial etc.
- ❑ Certaines entreprises qui utilisent Git pour le contrôle de version sont : Facebook, Yahoo, Zynga, Quora, Twitter, eBay, Salesforce, Microsoft et bien d'autres.
- ❑ Dernièrement, tous les nouveaux travaux de développement de Microsoft ont été intégrés aux fonctionnalités de Git. Microsoft migre .NET et nombre de ses projets open source sur GitHub qui sont gérés par Git.

# Github / GitLab

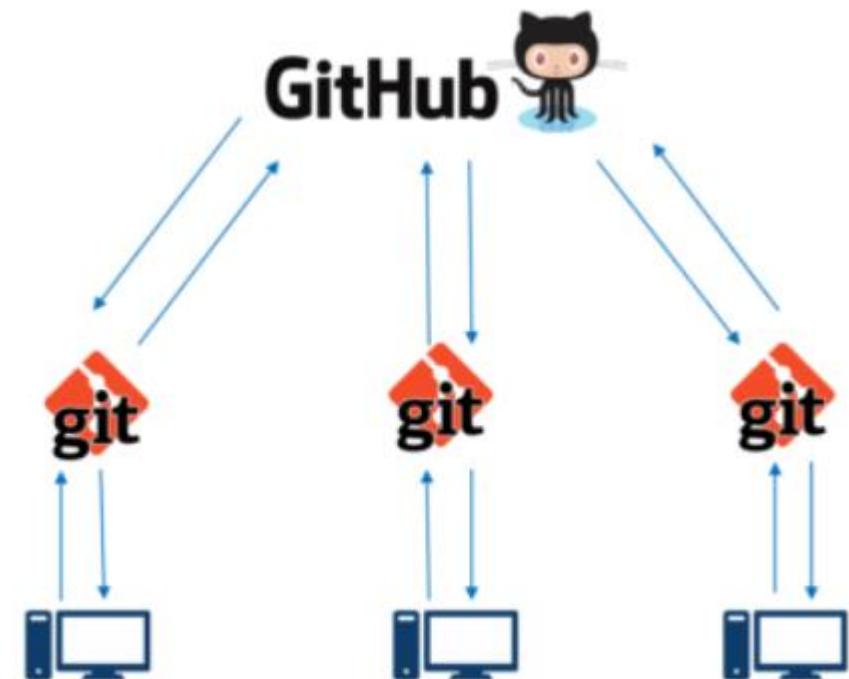
---

- ❑ Des services de partage de fichiers ou de code pour collaborer avec différentes personnes.
- ❑ GitHub est un logiciel très utilisé qui pour le contrôle de version.
- ❑ Prenons un exemple :
  - ❑ Supposons que plus de deux développeurs de logiciels travaillent sur le même fichier et qu'ils souhaitent le mettre à jour simultanément.
  - ❑ Malheureusement, la personne qui enregistre le fichier en premier aura la priorité sur les autres.
  - ❑ Github documente les modifications et les reflète de manière organisée pour éviter tout chaos entre les fichiers téléchargés
- ❑ Par conséquent, en utilisant le référentiel centralisé GitHub, cela évite toute confusion et travailler sur le même code devient très facile

# GIT & GITHUB

---

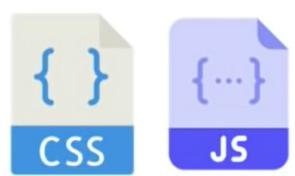
- ❑ GitHub est un référentiel central et Git est un outil qui vous permet de créer un référentiel local.
- ❑ Git est un outil de contrôle de version qui vous permettra d'effectuer toutes sortes d'opérations pour récupérer des données du serveur central ou y envoyer des données
- ❑ GitHub est une plate-forme d'hébergement de base pour la collaboration de contrôle de version.
- ❑ GitHub est une entreprise qui vous permet d'héberger un référentiel central sur un serveur distant.



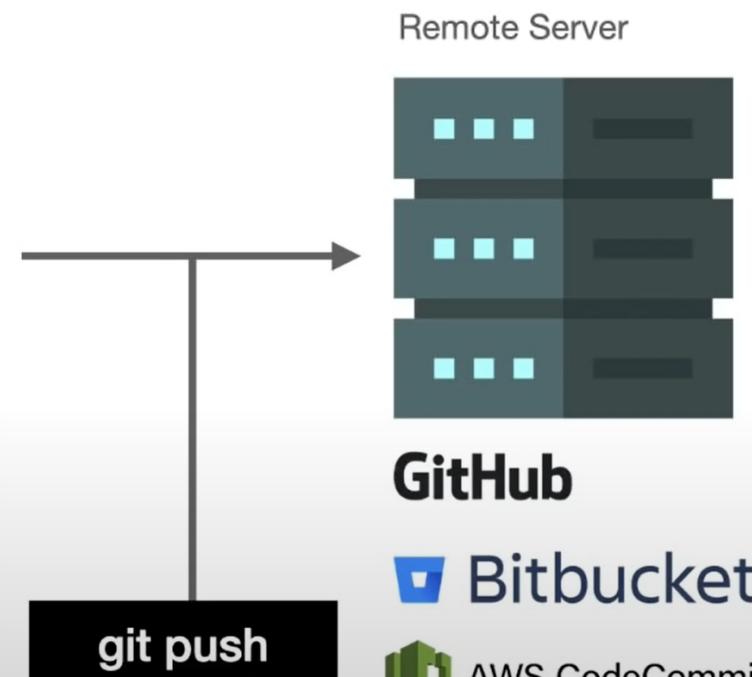
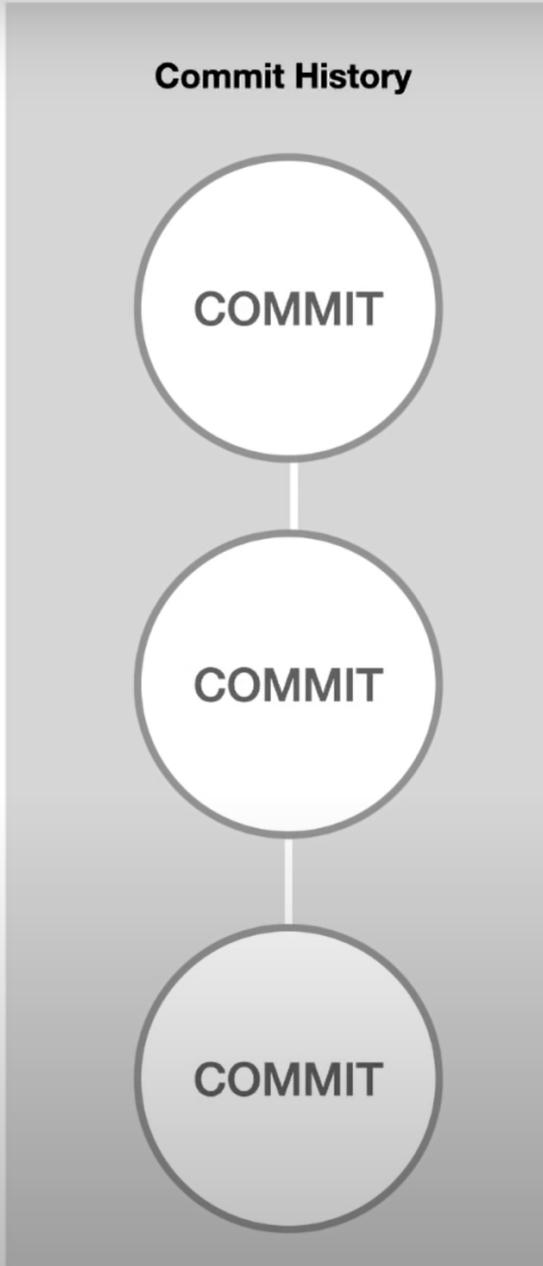
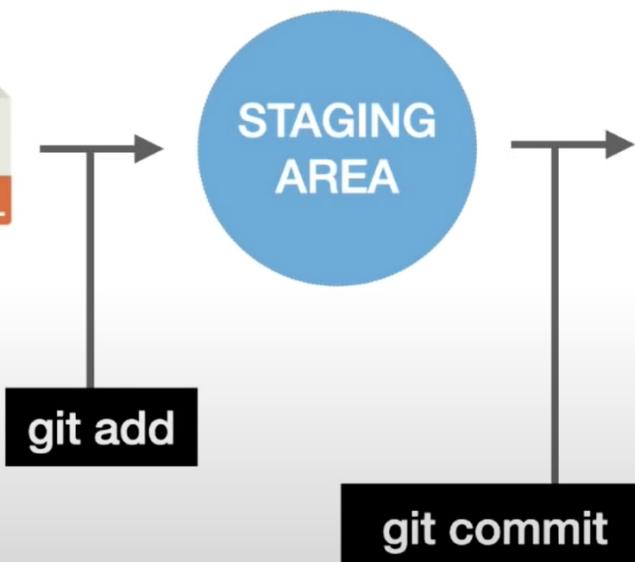


# GIT PUSH

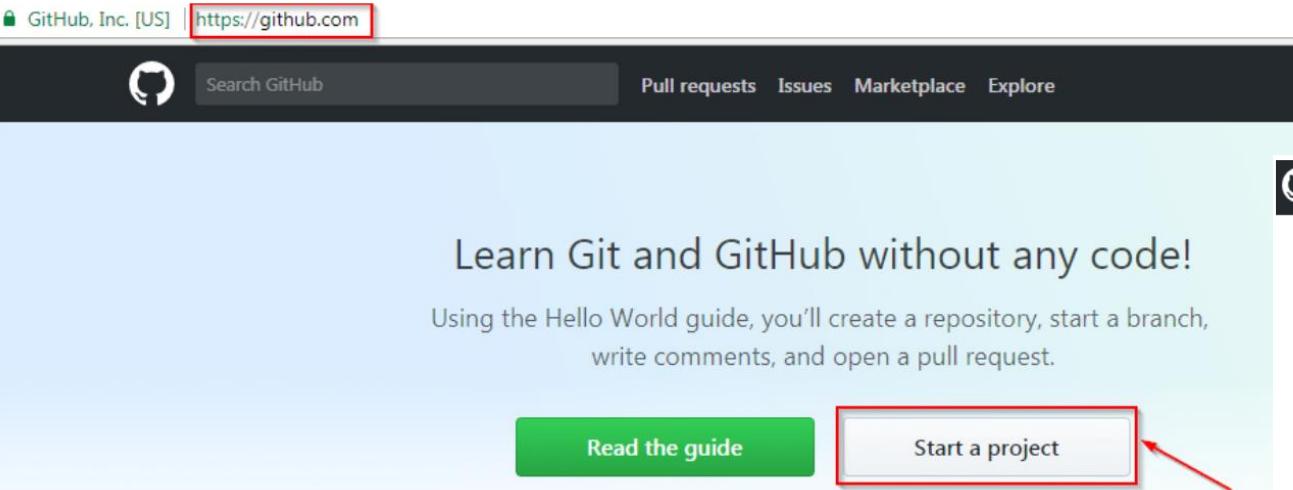
## Working Directory



Local machine



# Create a GitHub Repository



The screenshot shows the GitHub homepage. At the top, there's a navigation bar with links for Pull requests, Issues, Marketplace, and Explore. Below the navigation bar, a banner reads "Learn Git and GitHub without any code!" and "Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request." Two buttons are present: "Read the guide" (green) and "Start a project" (white). A red box highlights the "Start a project" button, and a red arrow points from it to the "Create a new repository" form on the right.

GitHub, Inc. [US] | <https://github.com>

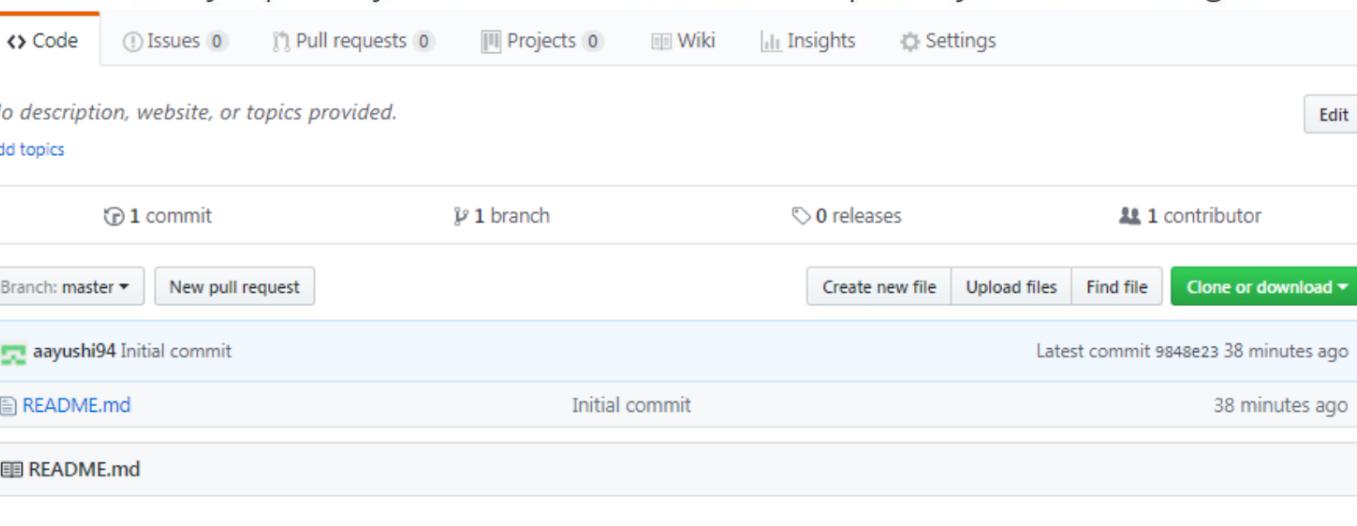
Search GitHub Pull requests Issues Marketplace Explore

Learn Git and GitHub without any code!

Using the Hello World guide, you'll create a repository, start a branch, write comments, and open a pull request.

Read the guide Start a project

- Enter any repository name and click on "Create Repository". You can also give



The screenshot shows a GitHub repository page for "ayayushi94/GitHub-Tutorial". The top navigation bar includes links for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Insights, and Settings. Below the navigation bar, a message says "No description, website, or topics provided." There's a "Edit" button. The main area displays repository statistics: 1 commit, 1 branch, 0 releases, and 1 contributor. It also shows a "Clone or download" button. Below this, a list of files includes "ayayushi94 Initial commit" (a file icon) and "README.md" (a text icon). The "README.md" file is described as an "Initial commit" made 38 minutes ago.

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided.

Edit

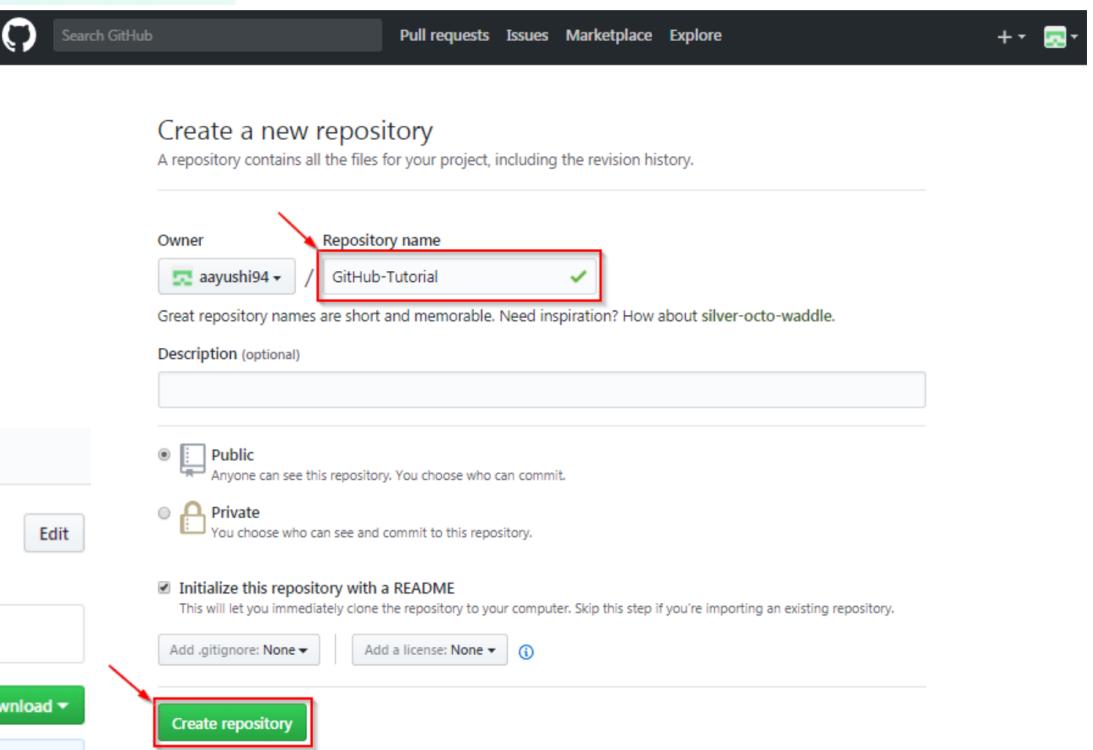
1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

ayayushi94 Initial commit Latest commit 9848e23 38 minutes ago

README.md Initial commit 38 minutes ago

README.md



The screenshot shows the "Create a new repository" form. It starts with a header "Create a new repository" and a sub-header "A repository contains all the files for your project, including the revision history." Below this, there are fields for "Owner" (set to "ayayushi94") and "Repository name" (set to "GitHub-Tutorial"). A red box highlights the "Repository name" field, and a red arrow points from it to the "Create repository" button at the bottom. The form also includes a "Description (optional)" field, a "Public" radio button (selected), a "Private" radio button, and a checkbox for "Initialize this repository with a README".

Search GitHub Pull requests Issues Marketplace Explore

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: aayushi94 Repository name: GitHub-Tutorial

Great repository names are short and memorable. Need inspiration? How about silver-octo-waddle.

Description (optional)

Public Anyone can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

Initialize this repository with a README

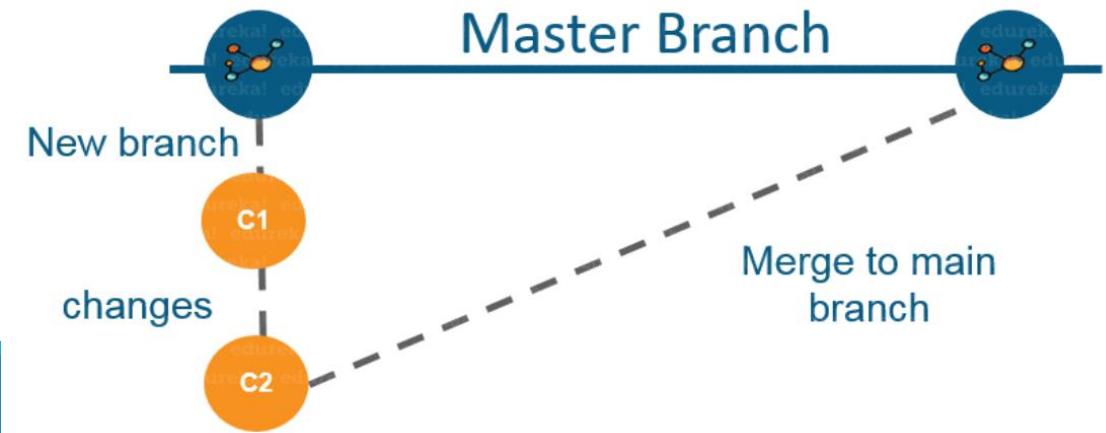
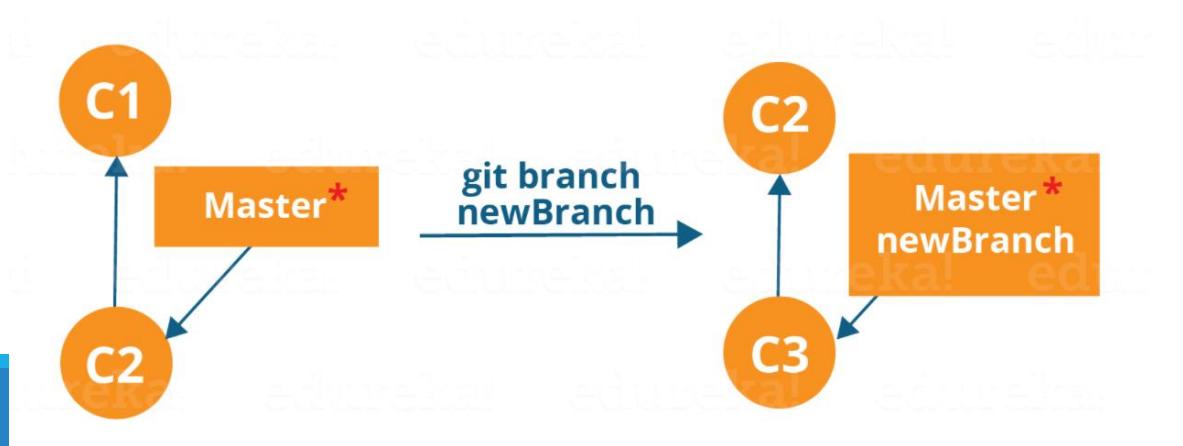
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None Add a license: None

Create repository

# Créer des Branches et effectuer des opérations

- ❑ Branching:
  - ❑ Les branches vous aident à travailler sur différentes versions d'un référentiel en même temps.
  - ❑ Supposons que vous souhaitez ajouter une nouvelle fonctionnalité (qui est en phase de développement) et que vous craignez en même temps de modifier ou non votre projet principal.
  - ❑ Les branches permettent d'aller et venir entre les différents états/versions d'un projet
  - ❑ vous pouvez créer une nouvelle branche et tester la nouvelle fonctionnalité sans affecter la branche principale
  - ❑ Une fois que vous avez terminé, vous pouvez fusionner les modifications de la nouvelle branche vers la branche principale

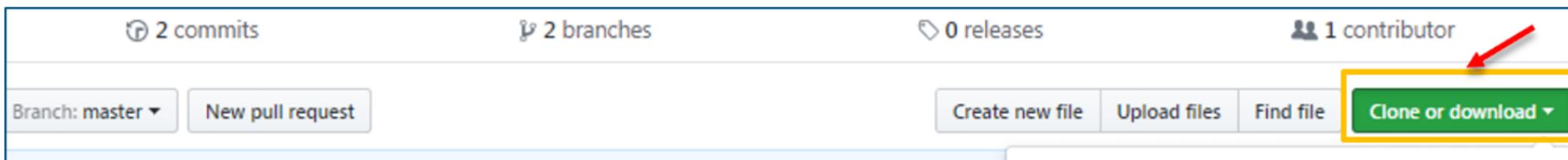
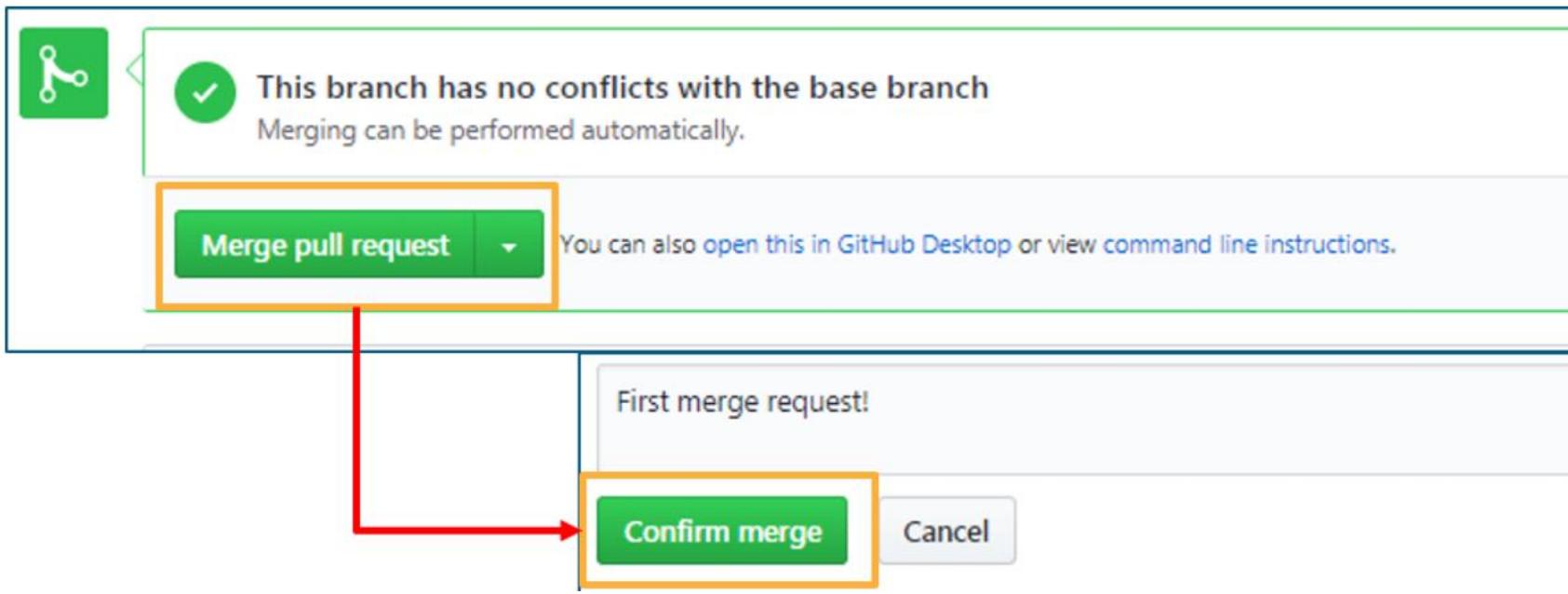


# Commit & pull

The screenshot shows the GitHub interface for a repository named "GitHub-Tutorial". The user is editing the file "README.md". A red box highlights the commit message: "Hey! This is for tutorial purpose." Below the editor, a "Commit changes" button is highlighted with a red box. The commit message "This is my first change!" is shown in a blue box. At the bottom, two options are available: "Commit directly to the readme--changes branch." and "Create a new branch for this commit and start a pull request." The "Commit changes" button is again highlighted with a red box.

The screenshot shows the GitHub interface for a repository named "aayushi94 / GitHub-Tutorial". Step 1 highlights the "Pull requests" tab. Step 2 highlights the "New pull request" button. Step 3 highlights the comparison between branches "readme--changes" and "master". Step 4 highlights the "Create pull request" button. Step 5 highlights the "First pull" message in the pull request details, which reads "This is my first pull request!".

# Megre & clone



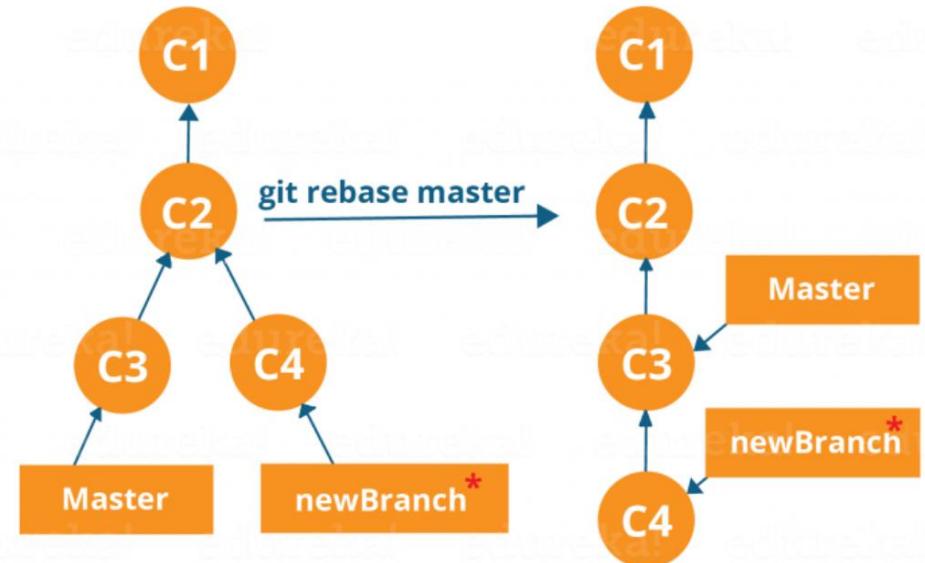
# Forking

---

- ❑ Supposons que vous ayez besoin de code présent dans un référentiel public, sous votre référentiel et votre compte GitHub.
- ❑ Les modifications apportées au référentiel d'origine seront répercutées dans le référentiel fork.
- ❑ Si vous apportez une modification au référentiel forké, elle ne sera pas reflétée dans le référentiel d'origine tant que vous n'aurez pas fait une demande d'extraction (pull)

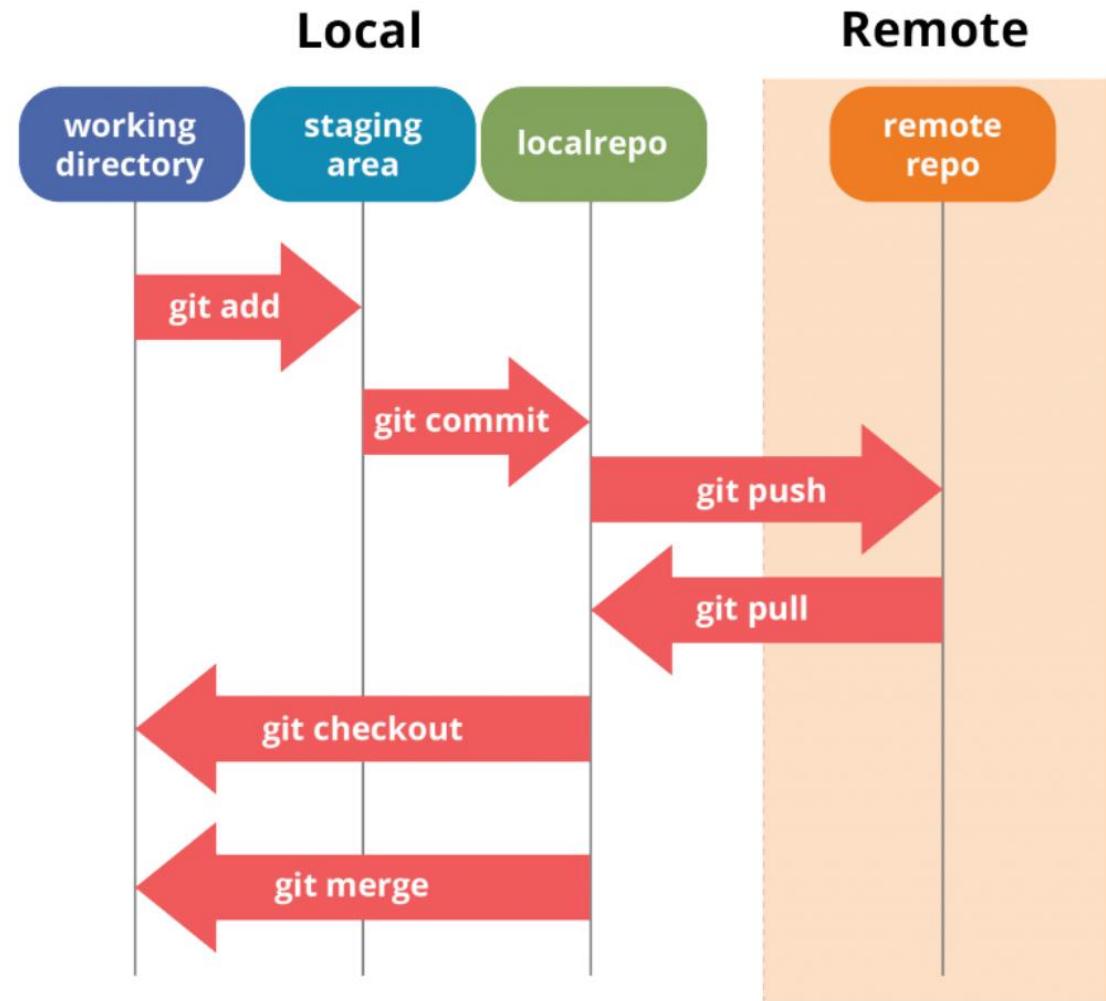
# Rebasing

- ❑ Le rebasage prend un ensemble de commits, les copie et les stocke en dehors de votre référentiel
- ❑ L'avantage du rebasage est qu'il peut être utilisé pour créer une séquence linéaire de commits.
- ❑ Le journal de validation ou l'historique du référentiel reste propre si le rebasage est effectué.
- ❑ Exemple :
  - ❑ notre travail de newBranch est placé juste après master et nous avons une belle séquence linéaire de commits.

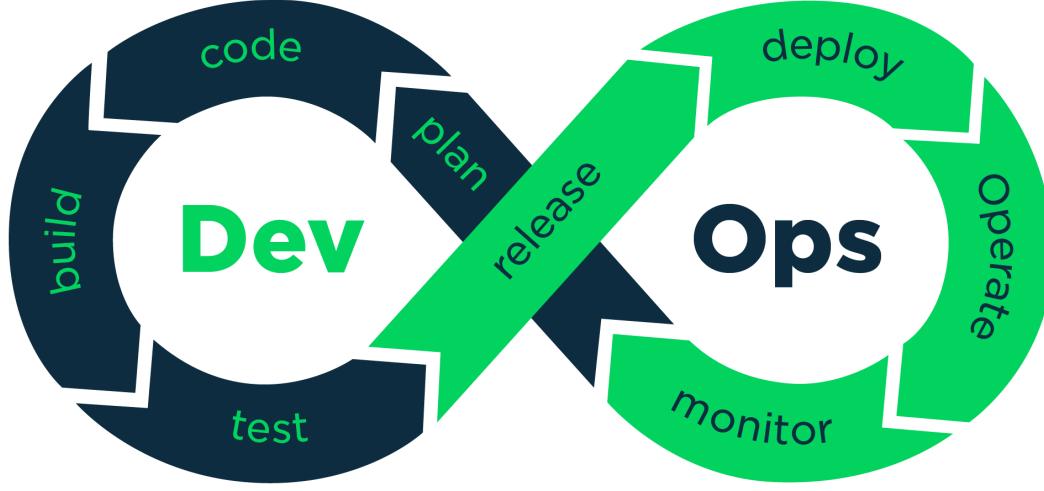


# Git commands

---







---

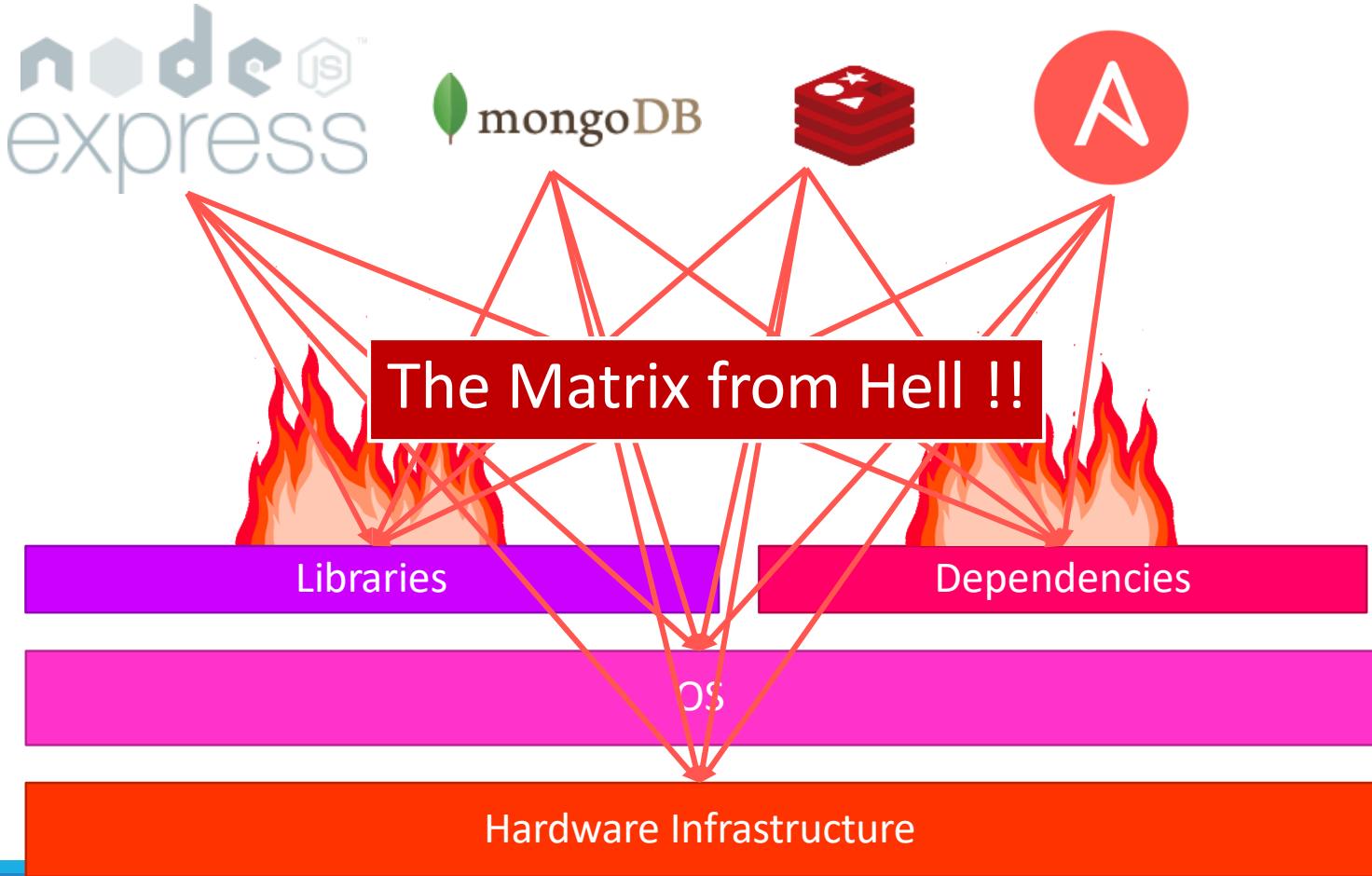
# Docker

---

# Pourquoi avez-vous besoin de docker?

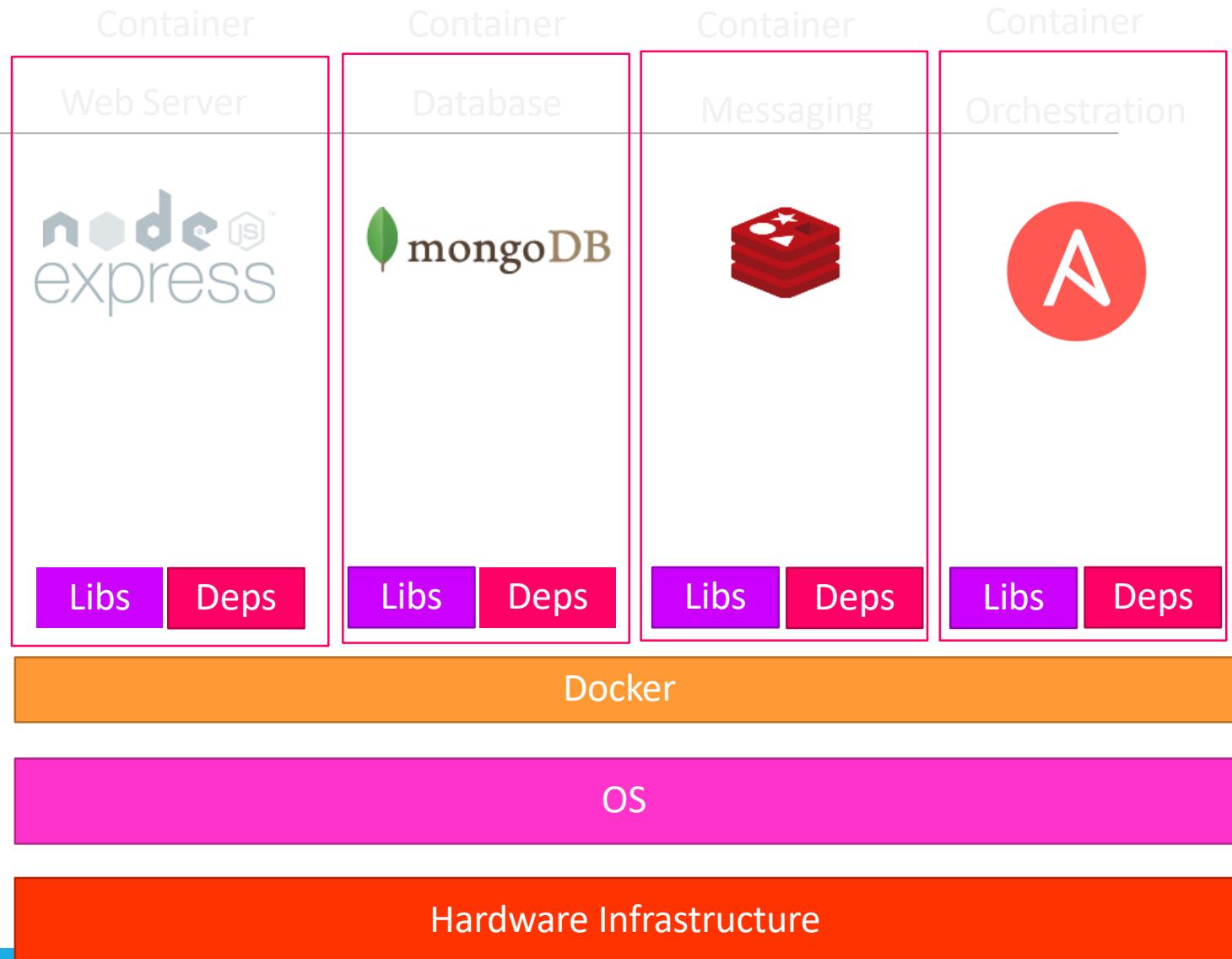
---

- Compatibilité / dépendance
- Temps de configuration ↑
- Différents environnements  
Dev/Test/Prod

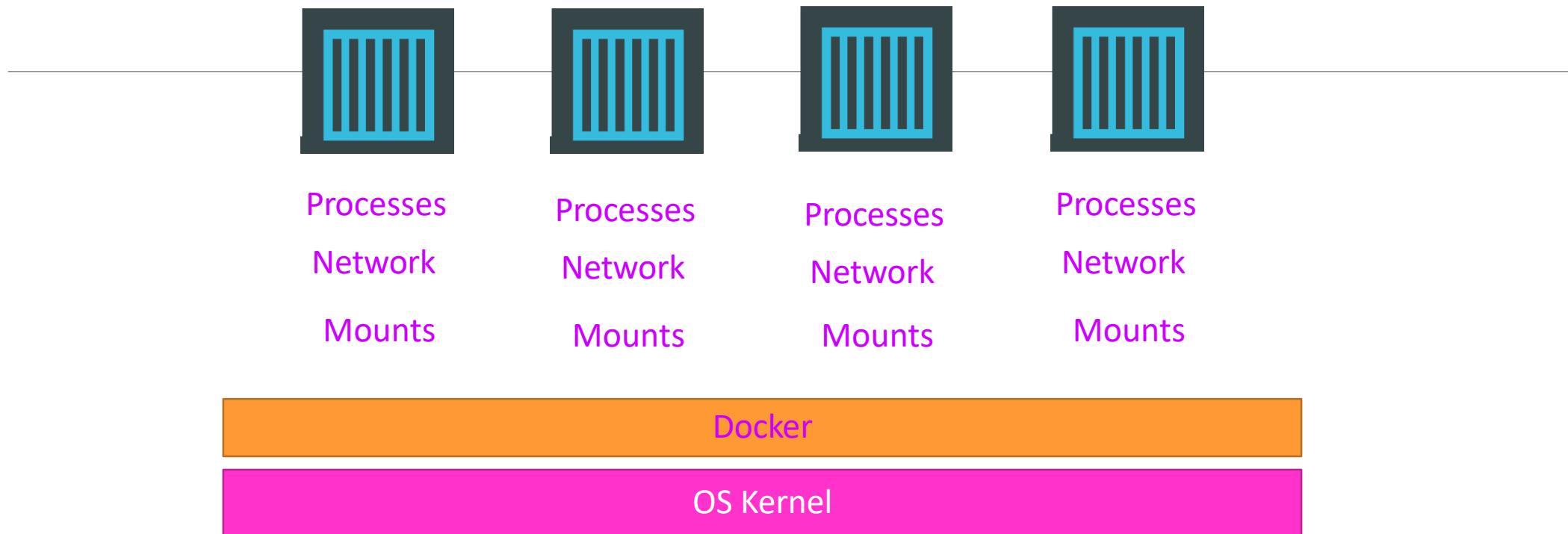


# Qu'est-ce que ça peut faire?

- Conteneuriser les application
- Exécutez chaque service avec ses propres dépendances dans des conteneurs séparés



# Que sont les conteneurs?



# Système d'exploitation

---



OS

Software

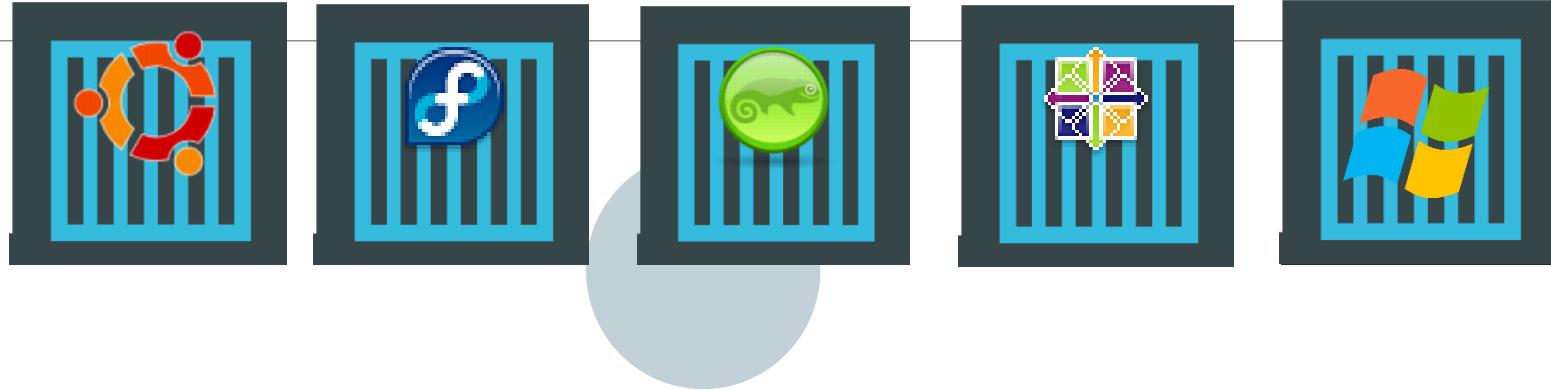
Software

Software

Software

OS Kernel

# Partager le noyau

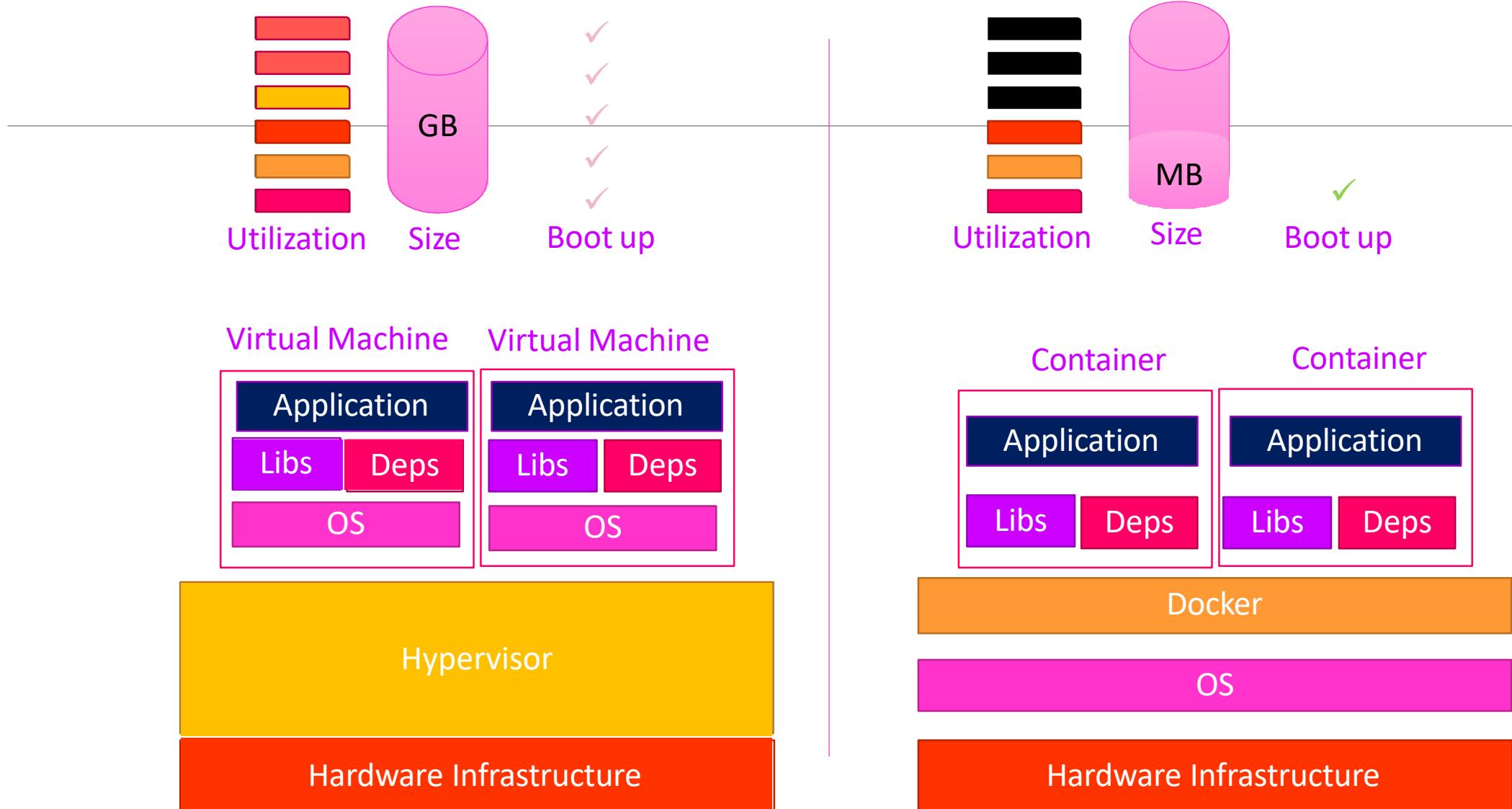


Docker

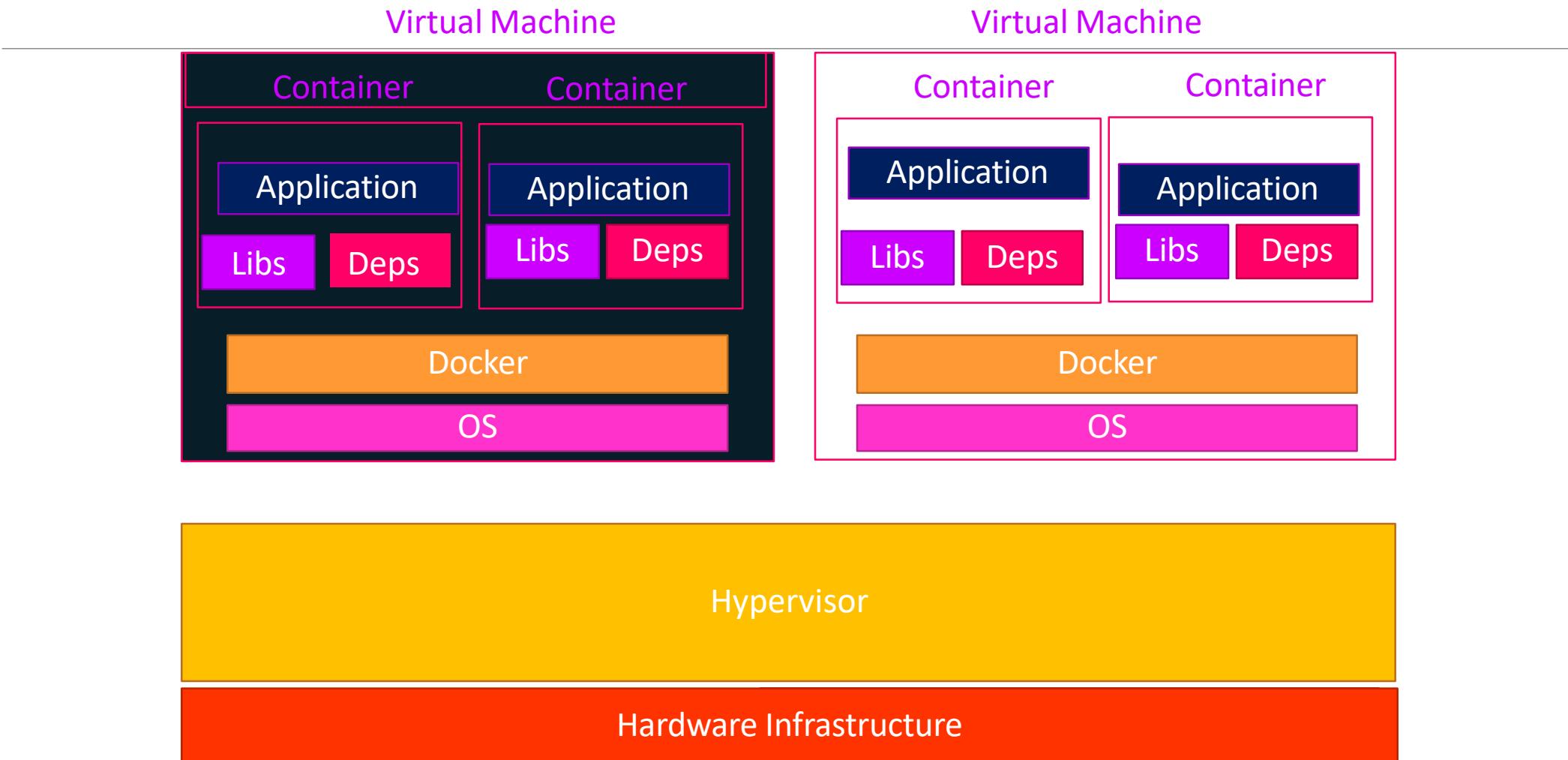


OS - Ubuntu

# Conteneurs vs Virtual Machines



# Conteneurs & Virtual Machines

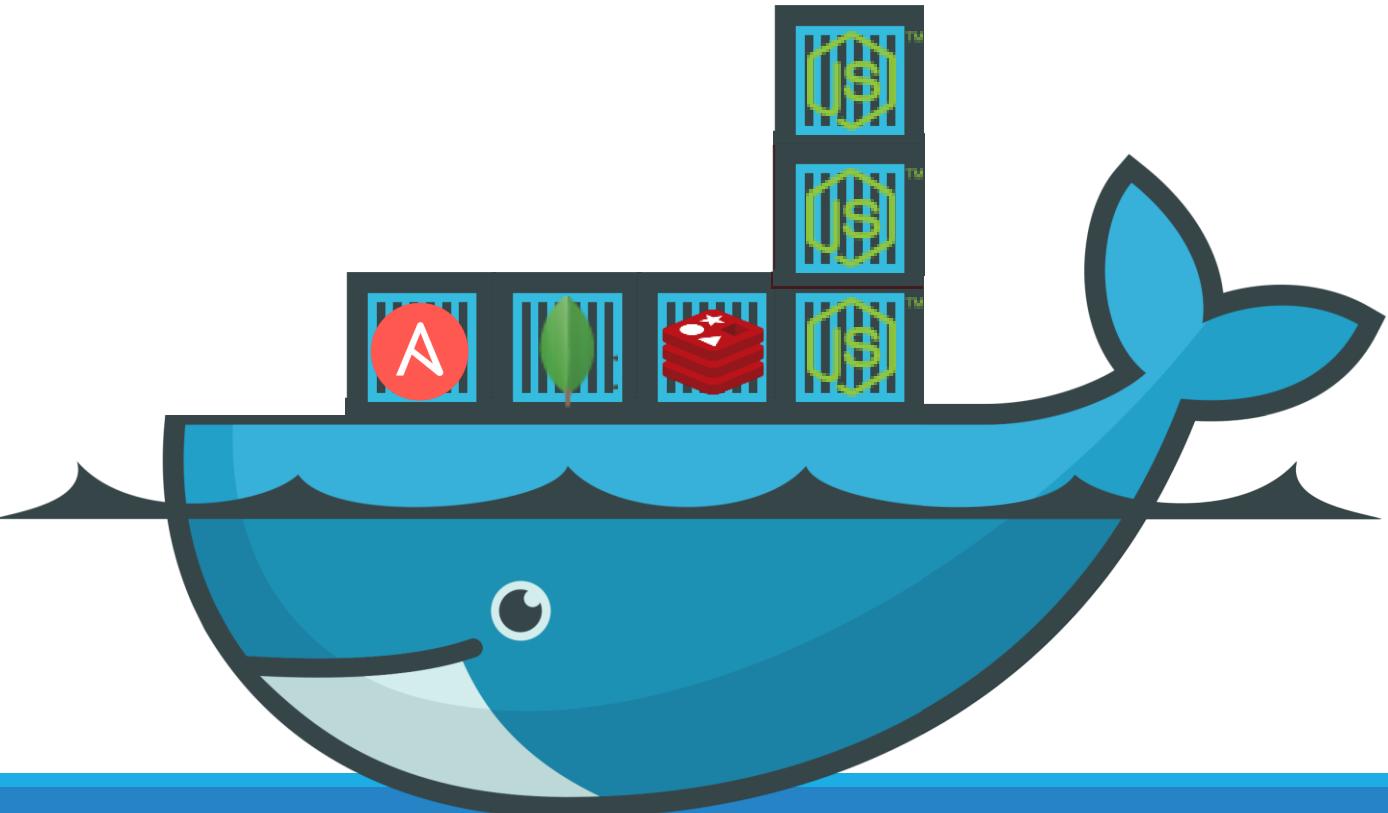


# Comment ça marche?



Public Docker registry - dockerhub

```
docker run ansible  
docker run mongodb  
docker run redis  
docker run nodejs  
docker run nodejs  
docker run nodejs
```



# Conteneur vs image



Package  
Template  
Plan



Docker Container #1



Docker Container #2



Docker Container #3

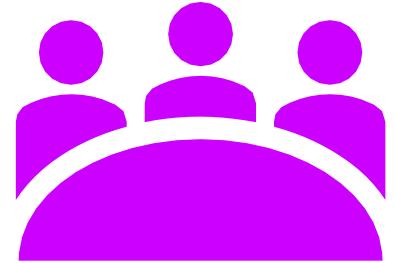
---

d o c k e r

# Getting Started

# Editions Docker

---



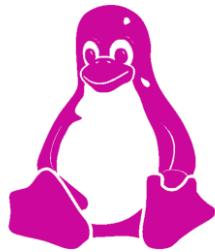
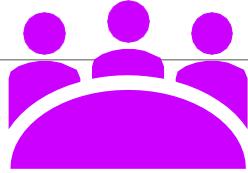
Community Edition



Enterprise Edition

# Community Edition

---



Linux



MAC



Windows



Cloud

---

# docker

## commandes

# Run – Commencer un Conteneur

```
▶ docker run nginx
```

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
fc7181108d40: Already exists
d2e987ca2267: Pull complete
0b760b431b11: Pull complete
Digest:
sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Status: Downloaded newer image for nginx:latest
```

# ps – liste des Conteneurs

```
▶ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
796856ac413d	nginx	"nginx -g 'daemon of..."	7 seconds ago	Up 6 seconds	80/tcp	silly_sammet

```
▶ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
796856ac413d	nginx	"nginx -g 'daemon of..."	7 seconds ago	Up 6 seconds	silly_sammet
cff8ac918a2f	redis	"docker-entrypoint.s..."	6 seconds ago	Exited (0) 3 seconds ago	relaxed_aryabhata

# STOP – Arrêter un conteneur

```
▶ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
796856ac413d	nginx	"nginx -g 'daemon of..."	7 seconds ago	Up 6 seconds	80/tcp	silly.sammet

```
▶ docker stop silly.sammet
```

```
silly.sammet
```

```
▶ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
796856ac413d	nginx	"nginx -g 'daemon of..."	7 seconds ago	Exited (0) 3 seconds ago	silly.sammet
cff8ac918a2f	redis	"docker-entrypoint.s..."	6 seconds ago	Exited (0) 3 seconds ago	relaxed_aryabhata

# Rm – Supprimer un Conteneur

```
▶ docker rm silly_sammet
```

```
silly_sammet
```

```
▶ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
cff8ac918a2f	redis	"docker-entrypoint.s..."	6 seconds ago	Exited (0) 3 seconds ago	relaxed_aryabhata

# images – Liste des images

► docker images				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
nginx	latest	f68d6e55e065	4 days ago	109MB
redis	latest	4760dc956b2d	15 months ago	107MB
ubuntu	latest	f975c5035748	16 months ago	112MB
alpine	latest	3fd9065eaf02	18 months ago	4.14MB

# rmi – Supprimer des images

```
▶ docker rmi nginx
```

```
Untagged: nginx:latest
Untagged: nginx@sha256:96fb261b66270b900ea5a2c17a26abbfabef95506e73c3a3c65869a6dbe83223a
Deleted: sha256:f68d6e55e06520f152403e6d96d0de5c9790a89b4cfcc99f4626f68146fa1dbdc
Deleted: sha256:1b0c768769e2bb66e74a205317ba531473781a78b77feef8ea6fd7be7f4044e1
Deleted: sha256:34138fb60020a180e512485fb96fd42e286fb0d86cf1fa2506b11ff6b945b03f
Deleted: sha256:cf5b3c6798f77b1f78bf4e297b27cf5b6caa982f04caeb5de7d13c255fd7a1e
```

! Supprimer tous les conteneurs dépendant pour  
effacer une image

# Pull – Télécharger une image

```
▶ docker run nginx
```

```
Unable to find image 'nginx:latest' locally
latest: Pulling from library/nginx
fc7181108d40: Already exists
d2e987ca2267: Pull complete
0b760b431b11: Pull complete
Digest:
sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Status: Downloaded newer image for nginx:latest
```

```
▶ docker pull nginx
```

```
Using default tag: latest
latest: Pulling from library/nginx
fc7181108d40: Pull complete
d2e987ca2267: Pull complete
0b760b431b11: Pull complete
Digest:
sha256:96fb261b66270b900ea5a2c17a26abbfabe95506e73c3a3c65869a6dbe83223a
Status: Downloaded newer image for nginx:latest
```

```
▶ docker run ubuntu
```

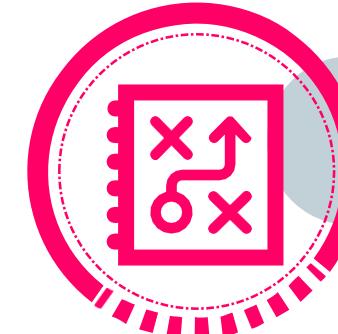
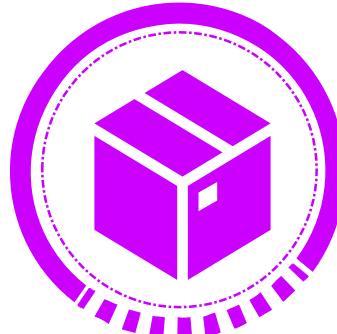
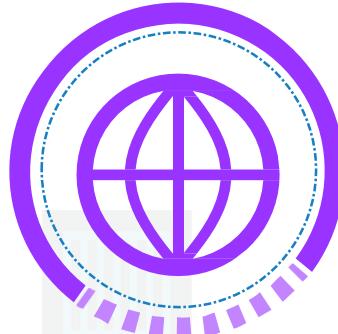
```
▶ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
--------------	-------	---------	---------	--------	-------

```
▶ docker ps -a
```

CONTAINER ID 45aacca36850	IMAGE ubuntu	COMMAND "/bin/bash"	CREATED 43 seconds ago	STATUS Exited (0) 41 seconds ago	PORTS
------------------------------	-----------------	------------------------	---------------------------	-------------------------------------	-------

► docker run ubuntu



► docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
--------------	-------	---------	---------	--------	-------

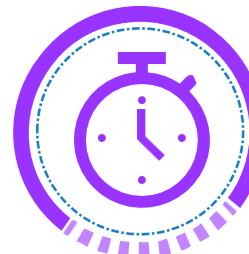
► docker ps -a

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
45aacca36850	ubuntu	"/bin/bash"	43 seconds ago	Exited (0) 41 seconds ago	

# Ajouter une commande

```
▶ docker run ubuntu
```

```
▶ docker run ubuntu sleep 5
```



# Exec – Exécuter une commande

```
▶ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	NAMES
538d037f94a7	ubuntu	"sleep 100"	6 seconds ago	Up 4 seconds	distracted_mcclintock

```
▶ docker exec distracted_mcclintock cat /etc/hosts
```

```
127.0.0.1      localhost
::1      localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.18.0.2      538d037f94a7
```

# Run – Attacher et détacher

```
▶ docker run kodekloud/simple-webapp
```

```
This is a sample web application that displays a colored background.  
* Serving Flask app "app" (lazy loading)  
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

```
▶ docker run -i kodekloud/simple-webapp
```

```
a043d40f85fefa414254e4775f9336ea59e19e5cf597af5c554e0a35a1631118
```

```
▶ docker attach a043d
```

---

d o c k e r  
r u n

RUN -

# tag

```
docker run redis
```

```
Using default tag: latest
latest: Pulling from library/redis
f5d23c7fed46: Pull complete
Status: Downloaded newer image for redis:latest

1:C 31 Jul 2019 09:02:32.624 # o000o000o000o Redis is starting o000o000o000o
1:C 31 Jul 2019 09:02:32.624 # Redis version=5.0.5, bits=64, commit=00000000, modified=0, pid=1, just started
1:M 31 Jul 2019 09:02:32.626 # Server initialized
```

```
docker run redis:4.
```

TAG

```
Unable to find image 'redis:4.0' locally
4.0: Pulling from library/redis
e44f086c03a2: Pull complete
Status: Downloaded newer image for redis:4.0
```

```
1:C 31 Jul 09:02:56.527 # o000o000o000o Redis is starting o000o000o000o
1:C 31 Jul 09:02:56.527 # Redis version=4.0.14, bits=64, commit=00000000, modified=0, pid=1, just started
1:M 31 Jul 09:02:56.530 # Server initialized
```

# RUN - STDIN

```
~/prompt-application$ ./app.sh  
Welcome! Please enter your name: Mumshad
```

```
Hello and Welcome Mumshad!
```

```
docker run kodekloud/simple-prompt-docker
```

```
Hello and Welcome !
```

```
docker run -i kodekloud/simple-prompt-docker
```

```
Mumshad
```

```
Hello and Welcome Mumshad!
```

```
docker run -i kodekloud/simple-prompt-docker
```

```
Welcome! Please enter your name: Mumshad
```

```
Hello and Welcome Mumshad!
```

# Run – PORT mapping

```
docker run kodekloud/webapp
```

```
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

http://172.17.0.2:5000

Internal IP

```
docker run -p 80:5000 kodekloud/simple-webapp
```

```
docker run -p 8000:5000 kodekloud/simple-webapp
```

```
docker run -p 8001:5000 kodekloud/simple-webapp
```

```
docker run -p 3306:3306 mysql
```

```
docker run -p 8306:3306 mysql
```

```
docker run -p 8306:3306 mysql
```

```
root@osboxes:/root # docker run -p 8306:3306 -e MYSQL_ROOT_PASSWORD=pass mysql
docker: Error response from daemon: driver failed programming external connectivity on endpoint boring_bhabha (5079d342b7e8ee11c71d46): Bind for 0.0.0.0:8306 failed: port is already allocated.
```



http://192.168.1.5:80

80      8000      8001

5000

5000

5000

IP: 172.17.0.2

Web APP  
Docker Container

IP: 172.17.0.3

Web APP  
Docker Container

IP: 172.17.0.4

Web APP  
Docker Container

IP: 172.17.0.5

MySQL  
Docker  
Container

IP: 172.17.0.6

MySQL  
Docker  
Container

IP: 172.17.0.6

MySQL  
Docker  
Container

8

3

0

6



Docker Host

3

0

6

8

3

0

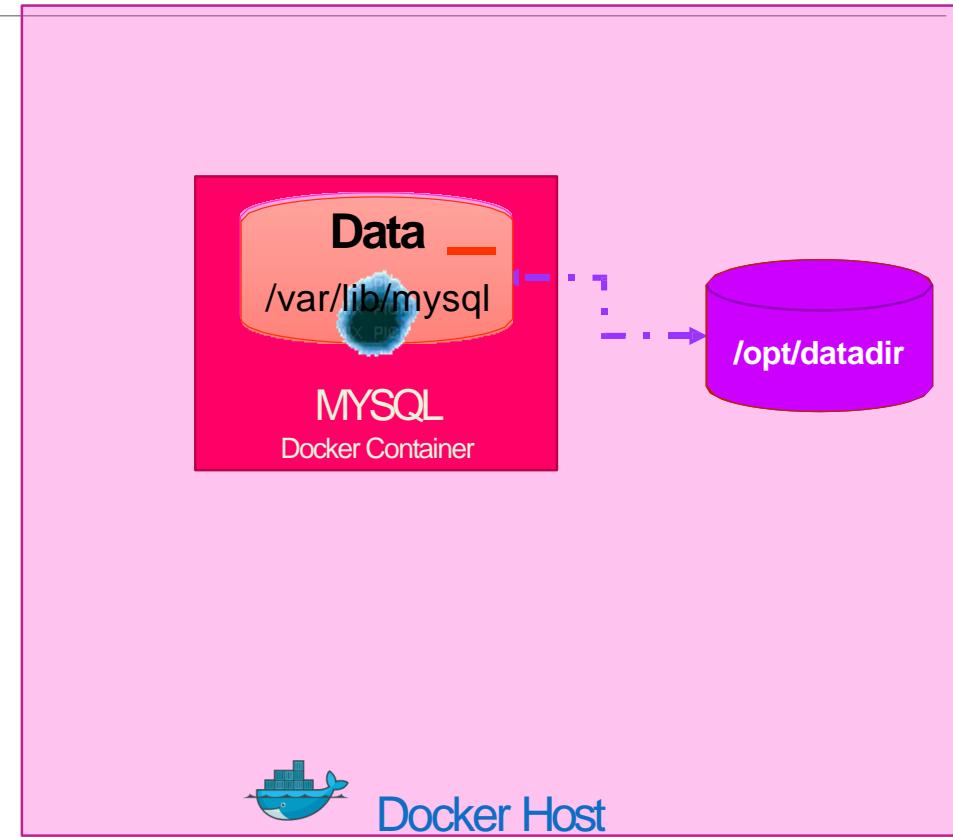
6

# RUN – Volume mapping

```
docker run mysql
```

```
docker stop mysql  
docker rm mysql
```

```
docker run -v /opt/datadir:/var/lib/mysql mysql
```



# Inspecter le Conteneur

```
▶ docker inspect blissful_hopper
```

```
[  
 {  
     "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",  
     "Name": "/blissful_hopper",  
     "Path": "python",  
     "Args": [  
         "app.py"  
     ],  
     "State": {  
         "Status": "running",  
         "Running": true,  
     },  
     "Mounts": [],  
     "Config": {  
         "Entrypoint": [  
             "python",  
             "app.py"  
         ],  
     },  
     "NetworkSettings": {...}  
 }]
```

# Logs de Conteneur

```
▶ docker logs blissful_hopper
```

```
This is a sample web application that displays a colored background.  
A color can be specified in two ways.
```

1. As a command line argument with --color as the argument. Accepts one of red,green,blue,blue2,pink,darkblue
2. As an Environment variable APP\_COLOR. Accepts one of red,green,blue,blue2,pink,darkblue
3. If none of the above then a random color is picked from the above list.  
Note: Command line argument precedes over environment variable.

```
No command line argument or environment variable. Picking a Random Color =blue  
* Serving Flask app "app" (lazy loading)  
* Environment: production  
WARNING: Do not use the development server in a production environment.  
Use a production WSGI server instead.  
* Debug mode: off  
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

---

docker  
variables de  
L'environnement

# Variables de l'environement

app.py

```
import os
from flask import Flask

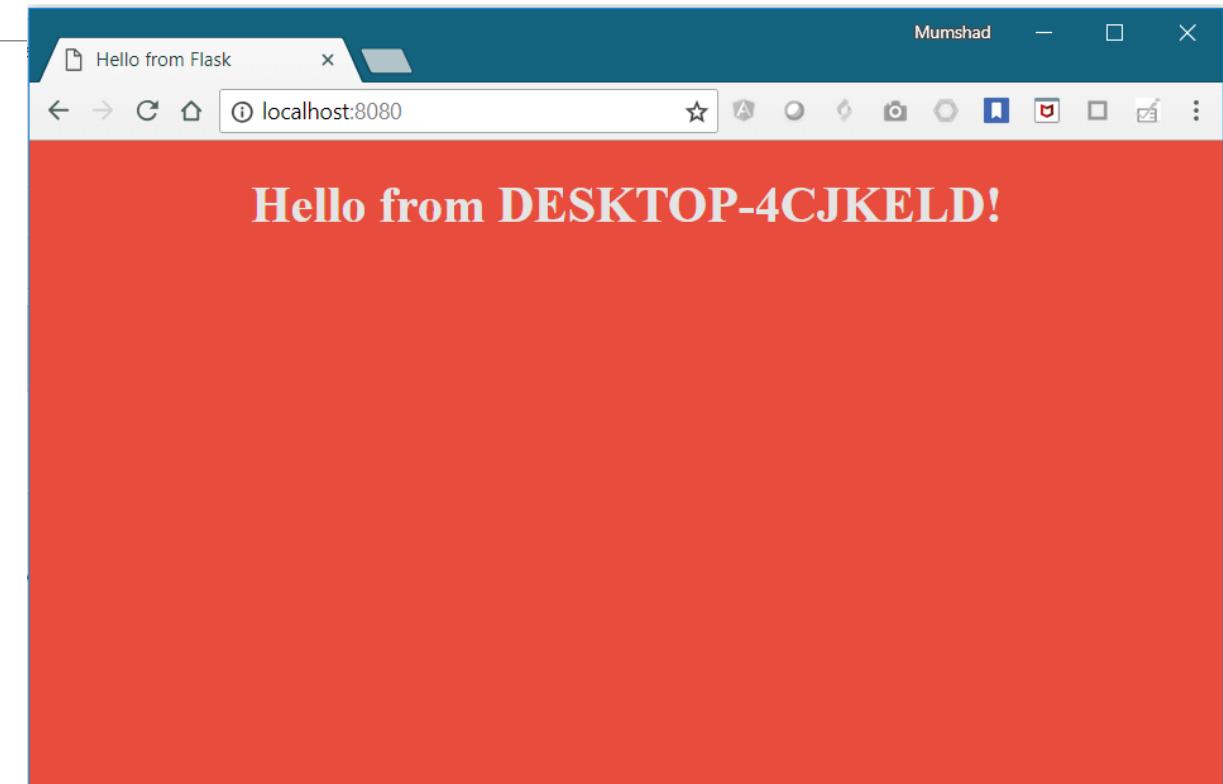
app = Flask(__name__)

...
...

color = "red"

@app.route("/")
def main():
    print(color)
    return render_template('hello.html', color=color)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8080")
```



▶ python app.py

# Variables de l'environement

app.py

```
import os
from flask import Flask

app = Flask(__name__)

...
...

color = "red"

@app.route("/")
def main():
    print(color)
    return render_template('hello.html', color=color)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8080")
```

# Variables de l'environement

app.py

```
import os
from flask import Flask

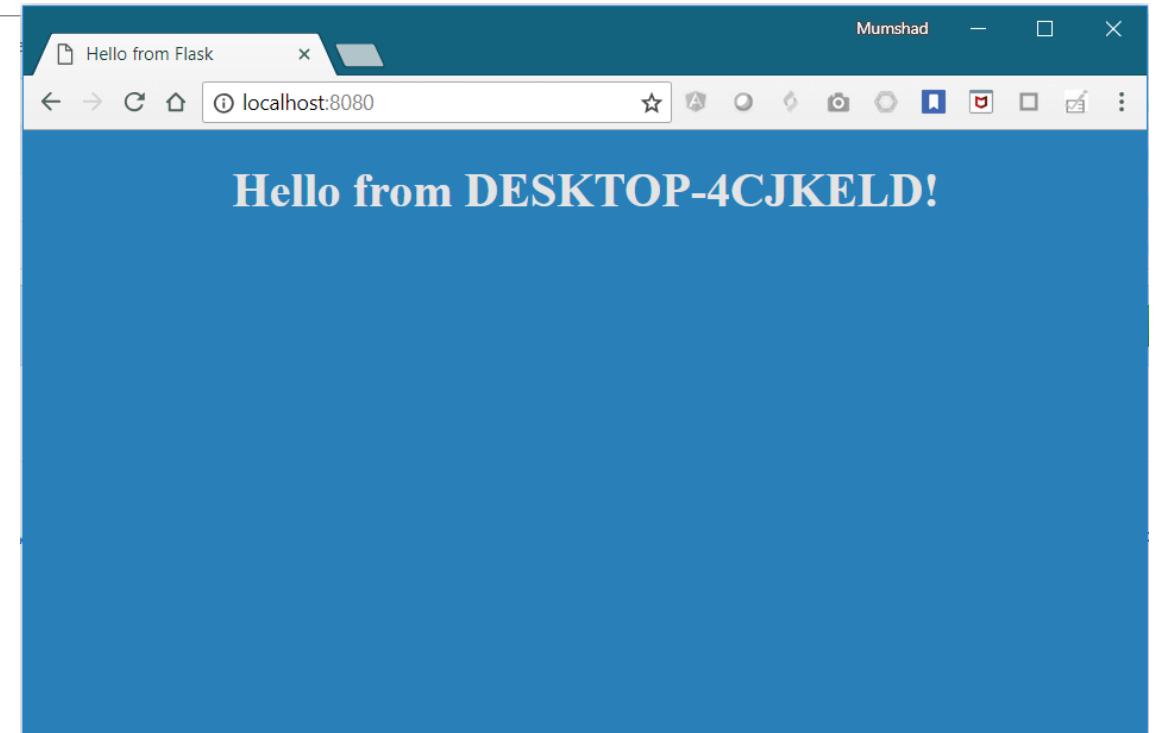
app = Flask(__name__)

...
...

color = os.environ.get('APP_COLOR')

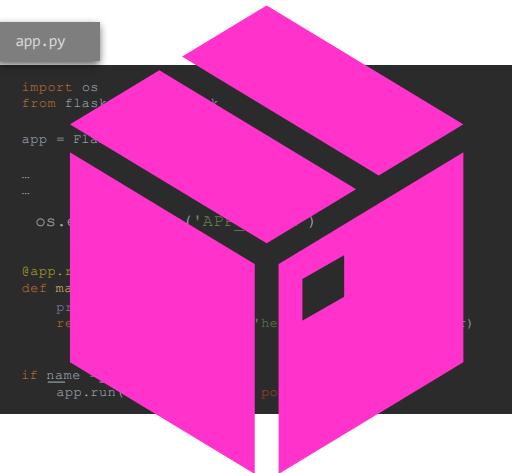
@app.route("/")
def main():
    print(color)
    return render_template('hello.html', color=color)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8080")
```

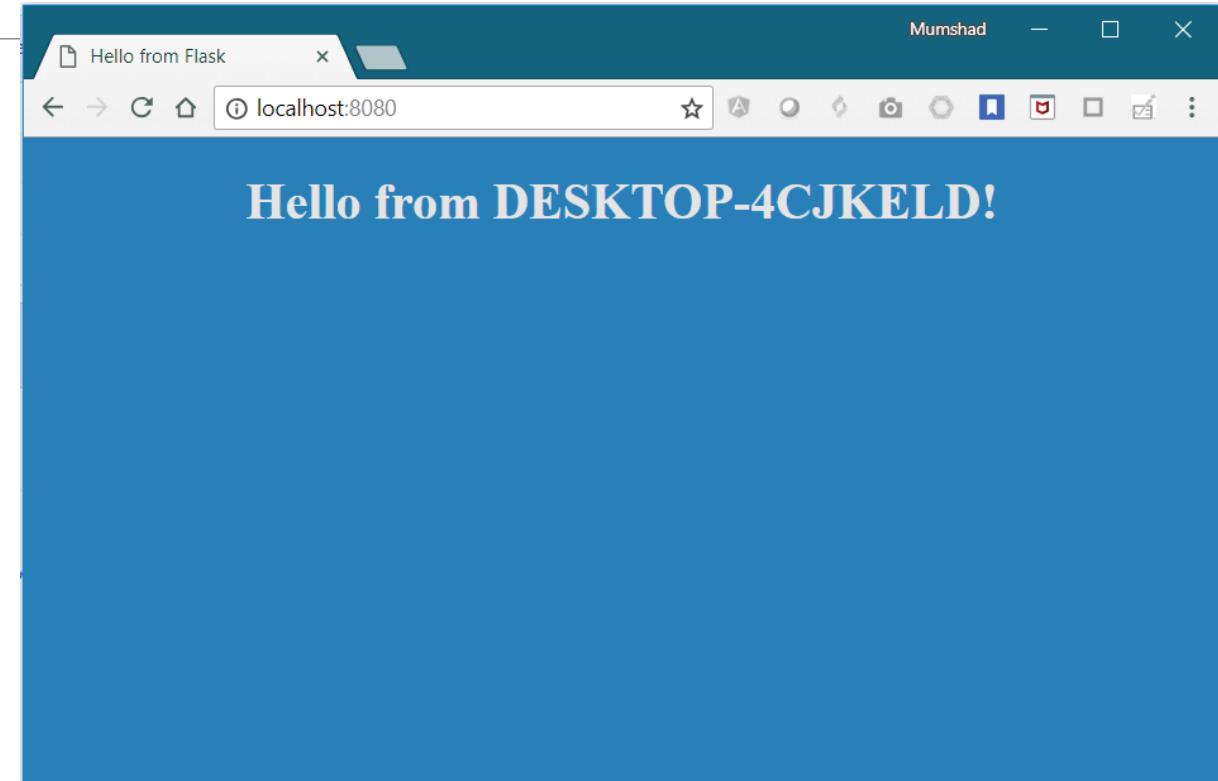


▶ export APP\_COLOR=blue; python app.py

# Variables de l'environement en Docker



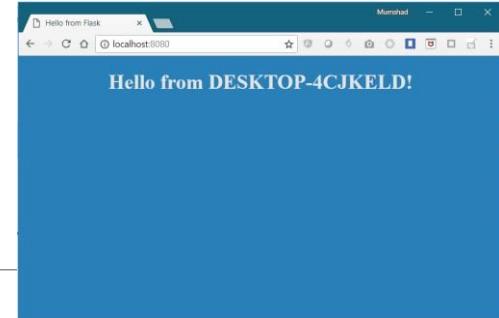
```
app.py
import os
from flask import Flask
app = Flask(__name__)
...
os.environ['API_NAME'] = 'API_1'
@app.route('/')
def main():
    print("Hello from " + os.environ['API_NAME'])
    return "Hello from " + os.environ['API_NAME']
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```



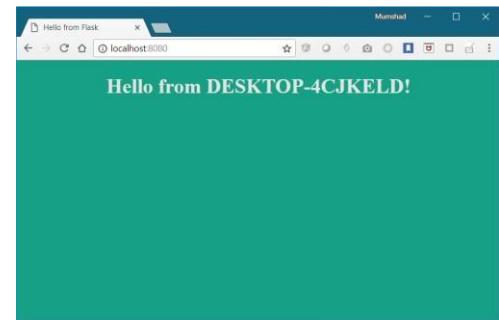
```
▶ docker run -e API_NAME=blue
```

# Variables ENV en Docker

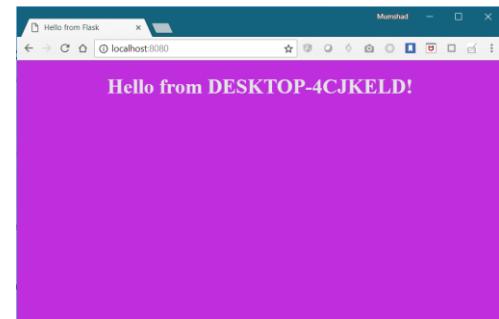
```
▶ docker run -e APP_COLOR=blue simple-webapp-color
```



```
▶ docker run -e APP_COLOR=green simple-webapp-color
```



```
▶ docker run -e APP_COLOR=pink simple-webapp-color
```



# Inspecter variable de l'environnement

```
docker inspect blissful_hopper
```

```
[  
  {  
    "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",  
    "State": {  
      "Status": "running", "Running": true,  
    },  
  
    "APP_COLOR=blue",  
    "MOUNTPOINT=/config";{  
      "KEY=0D96DF4D4110E5C43FBFB17F2D347EA6AA65421D",  
      "PYTHON_VERSION=3.6.6",  
      "PYTHON_PIP_VERSION=18.1"  
    },  
    "Entrypoint": [  
      "python",  
      "app.py"  
    ],  
  },  
]
```

---

d o c k e r  
i m a g e s

# Qu'est-ce que je conteneurise?



# Comment créer son propre image?

## Dockerfile

```
FROM Ubuntu  
  
RUN apt-get update  
RUN apt-get install python  
  
RUN pip install flask  
RUN pip install flask-mysql  
  
COPY . /opt/source-code  
  
ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
```

1. OS - Ubuntu

2. Mettre à jour apt repo

3. Installer les dépendances en utilisant apt

4. Installer les dépendances Python en utilisant pip

5. Copier le code source vers le fichier /opt

6. Exécuter le serveur web en utilisant la commande “flask”

```
docker build Dockerfile -t mmumshad/my-custom-app
```

```
docker push mmumshad/my-custom-app
```

Registre  
Docker

# Fichier Docker

Dockerfile

INSTRUCTION

ARGUMENT

Dockerfile

FROM Ubuntu

RUN apt-get update

RUN apt-get install python

RUN pip install flask

RUN pip install flask-mysql

COPY . /opt/source-code

ENTRYPOINT FLASK\_APP=/opt/source-code/app.py flask run

Commencer par la base OS  
ou  
Une autre image

Installer les dépendances

Copier le code source

Spécifier le point  
d'entrée

# Architecture en couches

Dockerfile

```
FROM Ubuntu

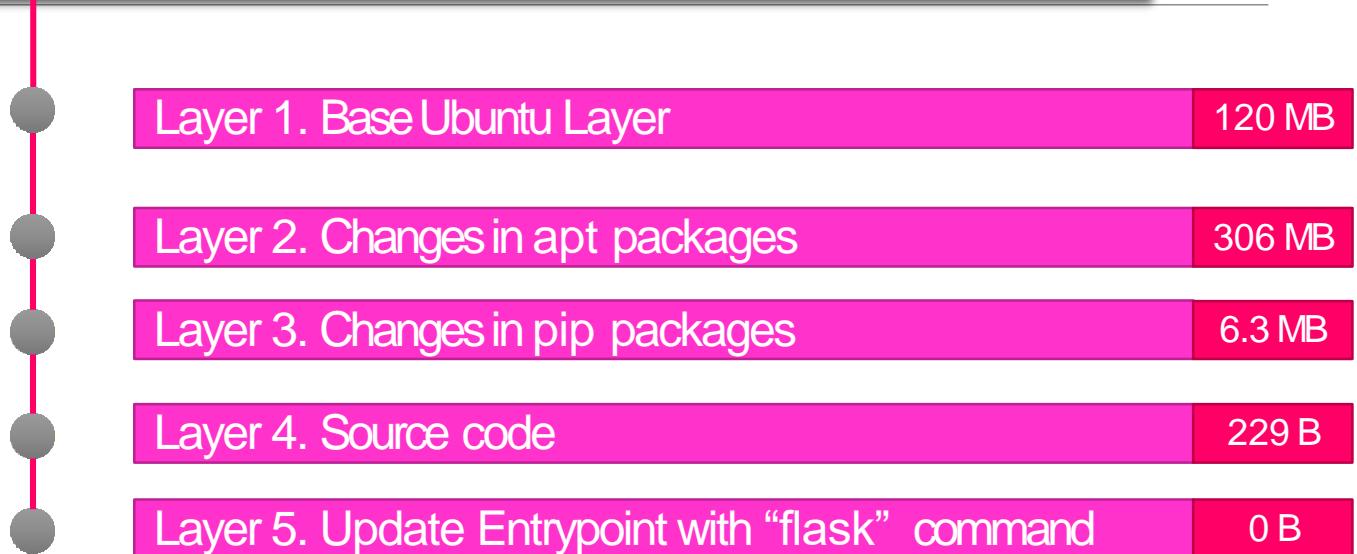
RUN apt-get update && apt-get -y install python

RUN pip install flask flask-mysql

COPY . /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask run
```

```
docker build Dockerfile -t mmumshad/my-custom-app
```



```
root@osboxes:/root/simple-webapp-docker # docker history mmumshad/simple-webapp
IMAGE          CREATED      CREATED BY
1a45ba829f10  About an hour ago /bin/sh -c #(nop)  ENTRYPOINT ["/bin/sh" ... 0B
37d37ed8fe99  About an hour ago /bin/sh -c #(nop) COPY file:29b92853d73898... 229B
d6aaebf8ded0  About an hour ago /bin/sh -c pip install flask flask-mysql 6.39MB
e4c055538e60  About an hour ago /bin/sh -c apt-get update && apt-get insta... 306MB
ccc7a11d65b1  2 weeks ago   /bin/sh -c #(nop) CMD ["/bin/bash"] 0B
<missing>     2 weeks ago   /bin/sh -c mkdir -p /run/systemd && echo '... 7B
<missing>     2 weeks ago   /bin/sh -c sed -i 's/^#\s*/(deb.*universe\... 2.76kB
<missing>     2 weeks ago   /bin/sh -c rm -rf /var/lib/apt/lists/* 0B
<missing>     2 weeks ago   /bin/sh -c set -xe  && echo '#!/bin/sh' >... 745B
<missing>     2 weeks ago   /bin/sh -c #(nop) ADD file:39d3593ea220e68... 120MB
```

# Sortie de build Docker build

```
root@osboxes:/root/simple-webapp-docker # docker build .
Sending build context to Docker daemon 3.072kB
Step 1/5 : FROM ubuntu
--> ccc7a11d65b1
Step 2/5 : RUN apt-get update && apt-get install -y python python-setuptools python-dev
--> Running in a7840dbfad17
Get:1 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:4 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [46.3 kB]
Get:5 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:6 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [440 kB]
Step 3/5 : RUN pip install flask flask-mysql
--> Running in a4a6c9190ba3
Collecting flask
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Collecting flask-mysql
  Downloading Flask_SQLAlchemy-1.4.0-py2.py3-none-any.whl
Removing intermediate container a4a6c9190ba3
Step 4/5 : COPY app.py /opt/
--> e7cdab17e782
Removing intermediate container faaaaaf63c512
Step 5/5 : ENTRYPOINT FLASK_APP=/opt/app.py flask run --host=0.0.0.0
--> Running in d452c574a8bb
--> 9f27c36920bc
Removing intermediate container d452c574a8bb
Successfully built 9f27c36920bc
```

# Echec

Couche 1. Base de couche Ubuntu

Couche 2. Changes in apt packages

Couche 3. Changes in pip packages

Couche 4. code Source

Couche 5. Mise à jour de point d'entrée par la commande “flask”

```
docker build Dockerfile -t mmumshad/my-custom-
```

app

```
root@osboxes:/root/simple-webapp-docker # docker build .
Sending build context to Docker daemon 5.12kB
Step 1/5 : FROM ubuntu
--> ccc7a11d65b1
Step 2/5 : RUN apt-get update && apt-get install -y python python-pip
--> Using cache
--> e4c055538e60
Step 3/5 : RUN pip install flask
--> Running in aacdaccd7403
Collecting flask
  Downloading Flask-0.12.2-py2.py3-none-any.whl (83kB)
Removing intermediate container aacdaccd7403
Step 4/5 : COPY app.py /opt/
--> af41ef57f6f3
Removing intermediate container a49cc8befc8f
Step 5/5 : ENTRYPOINT FLASK_APP=/opt/app.py flask run --host=0.0.0.0
--> Running in 3d745ff07d5a
--> 910416d360b6
Removing intermediate container 3d745ff07d5a
Successfully built 910416d360b6
```

# Que pouvez-vous conteneuriser?



curl://



## Conteneuriser tout!!!



CMD  
VS

POINT D'ENTREE

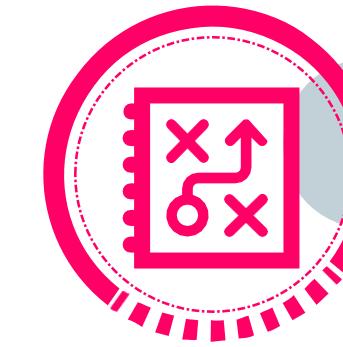
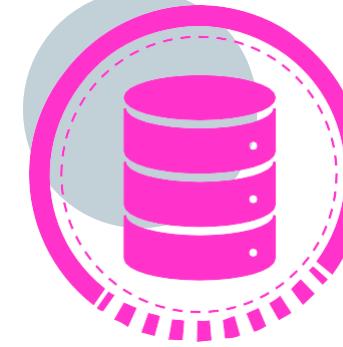
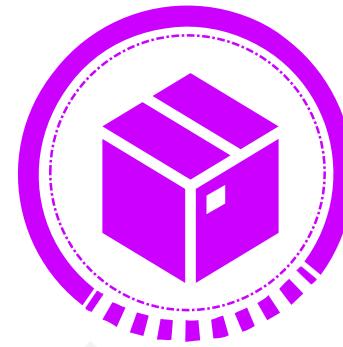
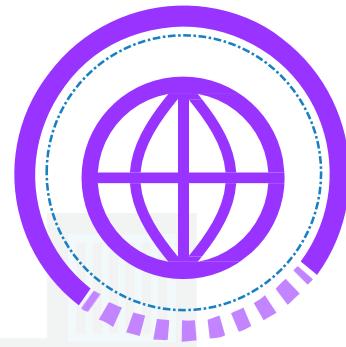
```
▶ docker run ubuntu
```

```
▶ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
--------------	-------	---------	---------	--------	-------

```
▶ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
45aacca36850	ubuntu	"/bin/bash"	43 seconds ago	Exited (0) 41 seconds ago	



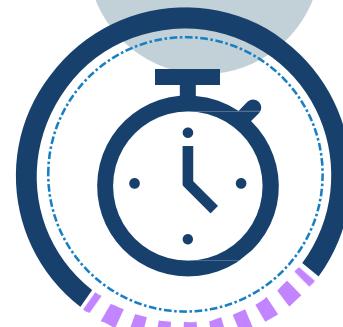
```
# Install Nginx.  
RUN \  
    add-apt-repository -y ppa:nginx/stable && \  
    apt-get update && \  
    apt-get install -y nginx && \  
    rm -rf /var/lib/apt/lists/* && \  
    echo "\ndaemon off;" >> /etc/nginx/nginx.conf && \  
    chown -R www-data:www-data /var/lib/nginx  
  
# Define mountable directories.  
VOLUME ["/etc/nginx/sites-enabled", "/etc/nginx/certs", "/etc/nginx/conf.d"]  
  
# Define working directory.  
WORKDIR /etc/nginx  
  
# Define default command.  
CMD ["nginx"]
```

```
ARG MYSQL_SERVER_PACKAGE_URL=https://repo.mysql.com/yum/mysql-8.0-community/docker/x86_64/  
ARG MYSQL_SHELL_PACKAGE_URL=https://repo.mysql.com/yum/mysql-tools-community/el/7/x86_64/  
  
# Install server  
RUN rpmkeys --import https://repo.mysql.com/RPM-GPG-KEY-mysql \  
    && yum install -y $MYSQL_SERVER_PACKAGE_URL $MYSQL_SHELL_PACKAGE_URL libpwquality \  
    && yum clean all \  
    && mkdir /docker-entrypoint-initdb.d  
  
VOLUME /var/lib/mysql  
  
COPY docker-entrypoint.sh /entrypoint.sh  
COPY healthcheck.sh /healthcheck.sh  
ENTRYPOINT ["/entrypoint.sh"]  
HEALTHCHECK CMD /healthcheck.sh  
EXPOSE 3306 33060  
CMD ["mysqld"]
```

```
# Pull base image.  
FROM ubuntu:14.04  
  
# Install.  
RUN \  
    sed -i 's/# \(.*multiverse$\)/\1/g' /etc/apt/sources.list && \  
    apt-get update && \  
    apt-get -y upgrade && \  
    apt-get install -y build-essential && \  
    apt-get install -y software-properties-common && \  
    apt-get install -y byobu curl git htop man unzip vim wget && \  
    rm -rf /var/lib/apt/lists/*  
  
# Add files.  
ADD root/.bashrc /root/.bashrc  
ADD root/.gitconfig /root/.gitconfig  
ADD root/.scripts /root/.scripts  
  
# Set environment variables.  
ENV HOME /root  
  
# Define working directory.  
WORKDIR /root  
  
# Define default command.  
CMD ["bash"]
```

▶ docker run ubuntu [COMMAND]

▶ docker run ubuntu sleep 5



FROM Ubuntu

CMD sleep 5

CMD command param1

CMD ["command", "param1"]

CMD sleep 5

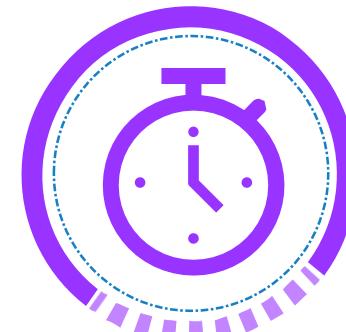
CMD ["sleep", "5"]

CMD ["sleep 5"]



```
▶ docker build -t ubuntu-sleeper .
```

```
▶ docker run ubuntu-sleeper
```



4

```
FROM Ubuntu
```

```
CMD sleep 5
```

Command at Startup: sleep 10

```
▶ docker run ubuntu-sleeper sleep 10
```

```
FROM Ubuntu
```

```
ENTRYPOINT [ "sleep" ]
```

Command at Startup:

```
▶ docker run ubuntu-sleeper
```

```
sleep: missing operand  
Try 'sleep --help' for more information.
```

Command at Startup:

```
FROM Ubuntu
```

```
ENTRYPOINT ["sleep"]
```

```
CMD ["5"]
```

▶ docker run ubuntu-sleeper

```
sleep: missing operand
Try 'sleep --help' for more information.
```

Command at Startup:

▶ docker run ubuntu-sleep 1r010

Command at Startup:

▶ docker run --entrypoint sleep 20ubuntu-sleep 1pe0r 10

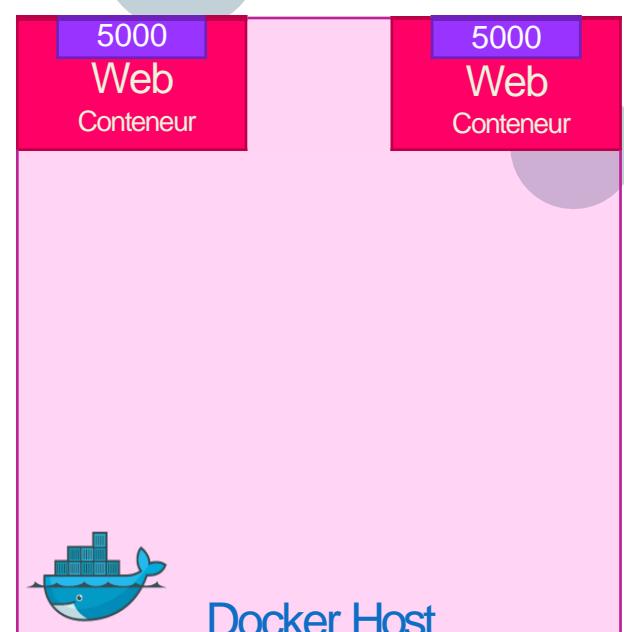
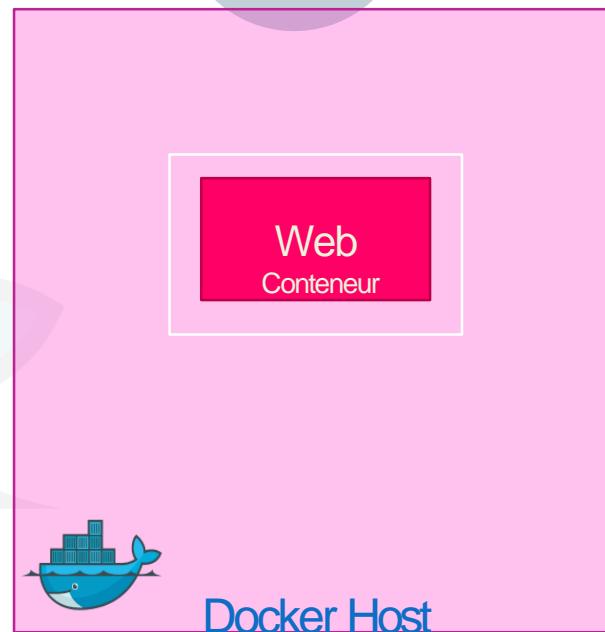
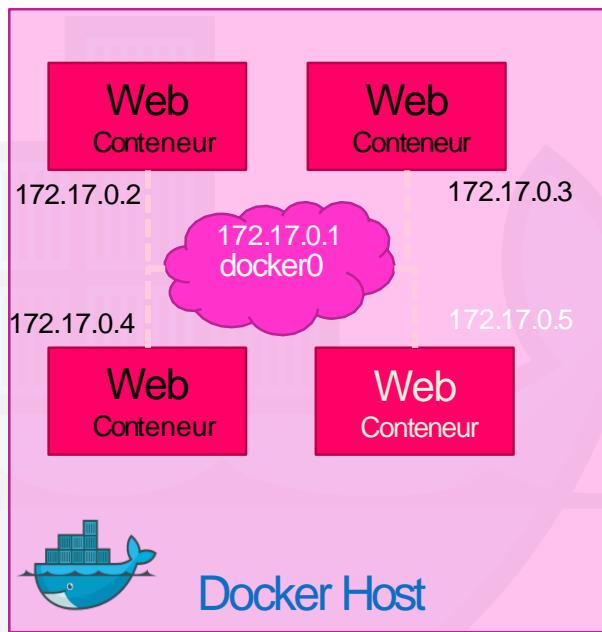
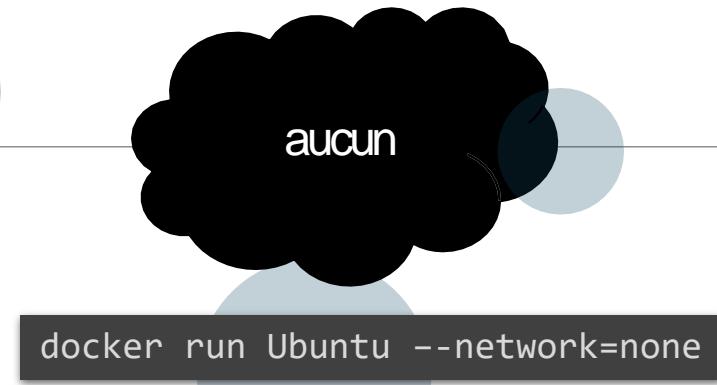
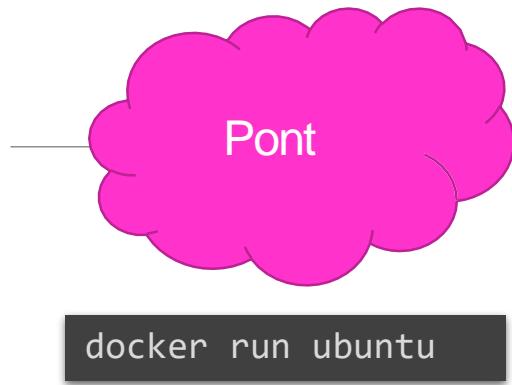
Command at Startup:

---

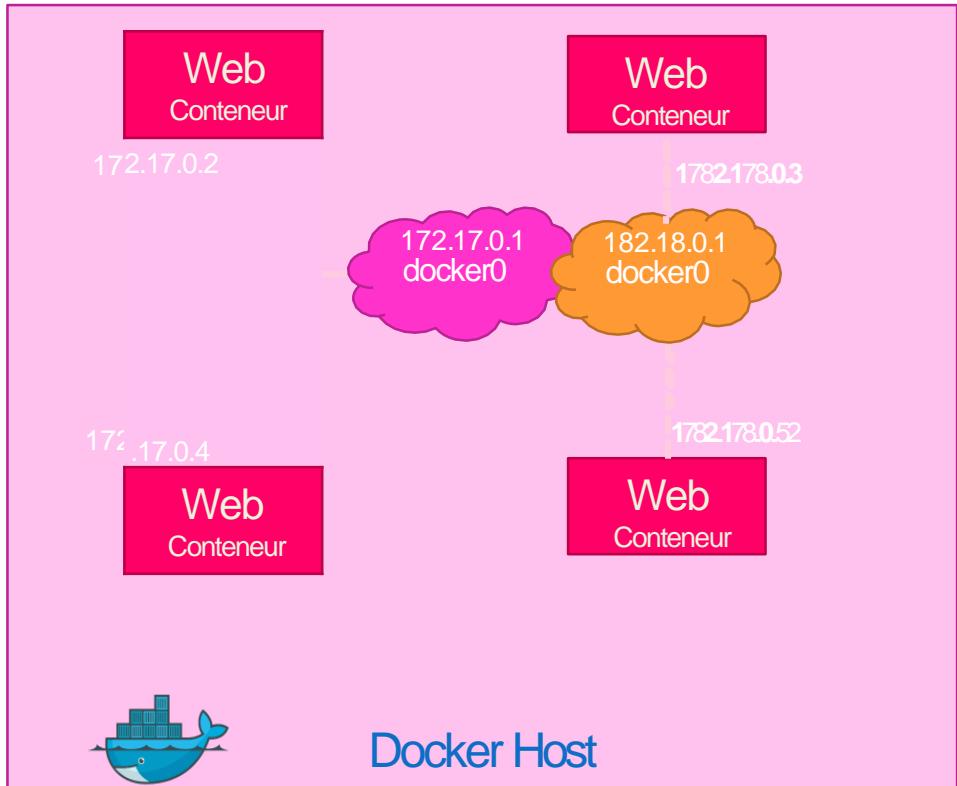
D o c k e r

# La mise en réseau

# Mise en réseau par défaut



# Réseaux définis par l'utilisateur



```
docker network create \
--driver bridge \
--subnet 182.18.0.0/16
custom-isolated-network
```

```
docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
dba0fb9370fe	bridge	bridge	local
46d476b87cd9	customer-isolated-network	bridge	local
6de685cec1ce	docker_gwbridge	bridge	local
e29d188b4e47	host	host	local
mmrho7vzb9rm	ingress	overlay	swarm
d9f11695f0d6	none	null	local
d371b4009142	simplewebappdocker_default	bridge	local

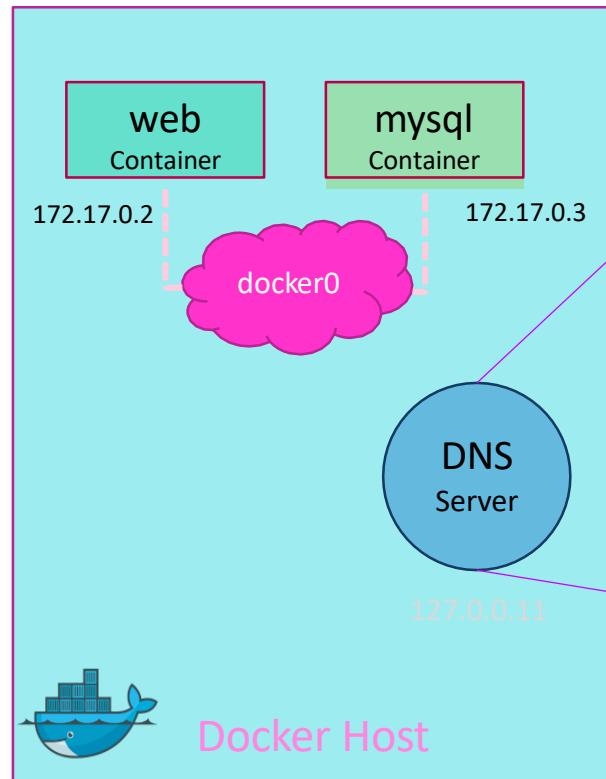
# Inspecter le réseau

```
docker inspect blissful_hopper
```

```
[  
{  
  "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",  
  "Name": "/blissful_hopper",  
  
  "NetworkSettings": {  
    "Bridge": "",  
    "Gateway": "172.17.0.1",  
    "IPAddress": "172.17.0.6",  
    "Gateway": "172.17.0.1",  
    "MacAddress": "02:42:ac:11:00:06",  
    "IPAddress": "172.17.0.6",  
    "MacAddress": "02:42:ac:11:00:06",  
  }  
}  
}
```

# DNS Intégré

```
mysql.connect(  mysql      )
```



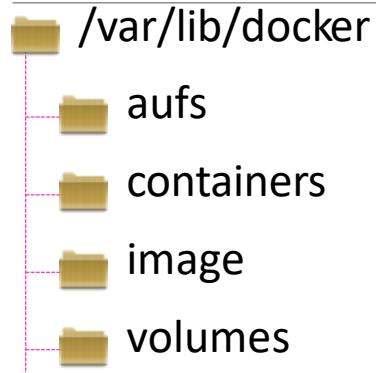
Host	IP
web	172.17.0.2
mysql	172.17.0.3

---

d o c k e r

# Espace de stockage

# Système de fichiers



# Architecture en couches

Fichier Docker

```
FROM Ubuntu

RUN apt-get update && apt-get -y install python

RUN pip install flask flask-mysql

COPY . /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app.py flask
run
```

```
docker build Dockerfile -t mmumshad/my-custom-app
```

Fichier Docker 2

```
FROM Ubuntu

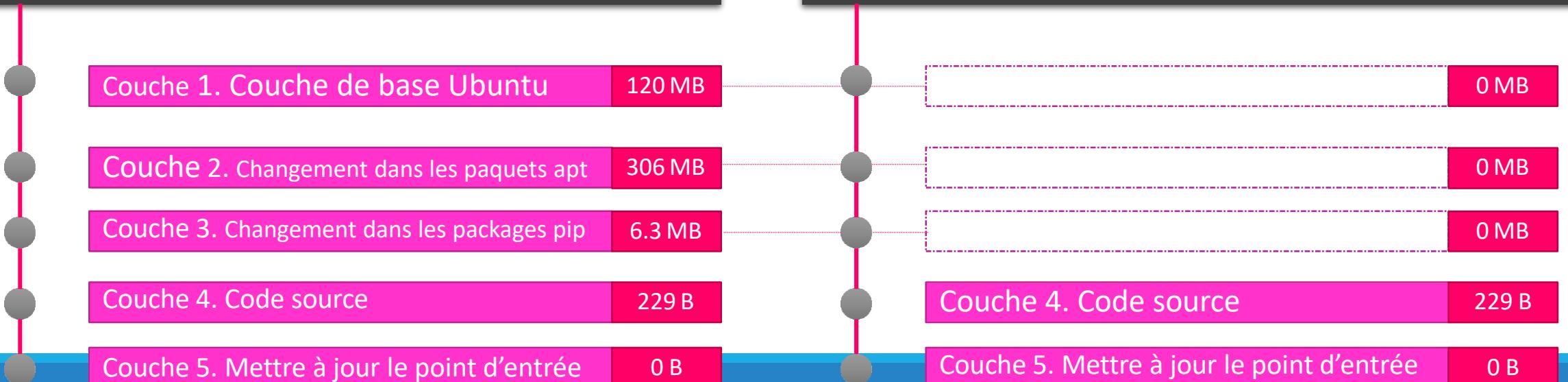
RUN apt-get update && apt-get -y install python

RUN pip install flask flask-mysql

COPY app2.py /opt/source-code

ENTRYPOINT FLASK_APP=/opt/source-code/app2.py flask
run
```

```
docker build Dockerfile2 -t mmumshad/my-custom-app-2
```



# Architecture en couches

Couche Conteneur

Lire / Ecrire

Couche 6. Couche Conteneur

```
docker run mmumshad/my-custom-app
```

Couches Image

Lire seulement

Couche 5. Mettre à jour point d'entrée avec la commande "flask"

Couche 4. Code Source

Couche 3. Changes in pip packages

Couche 2. Changes in apt packages

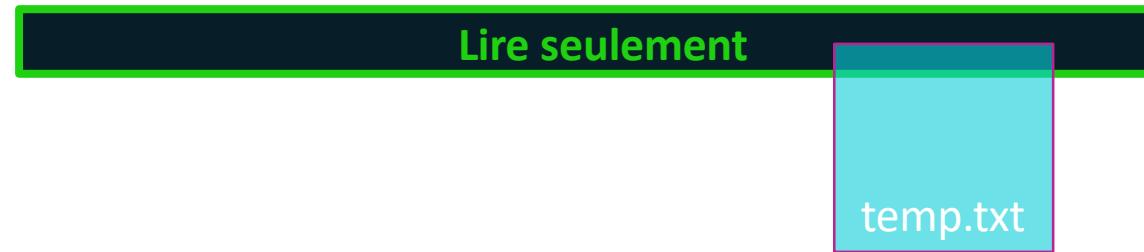
Couche 1. Couche de Base Ubuntu

```
docker build Dockerfile -t mmumshad/my-custom-app
```

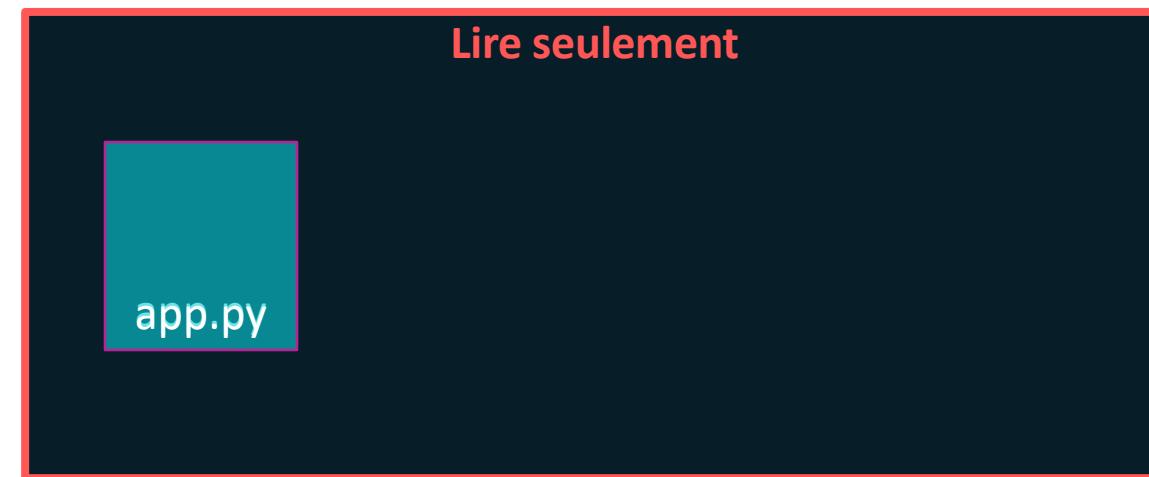
# COPIER SUR ECRIRE

---

Couche Conteneur

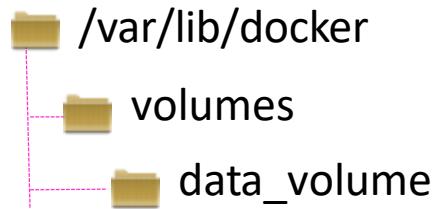


Couches Image



# volumes

```
docker volume create data_volume
```

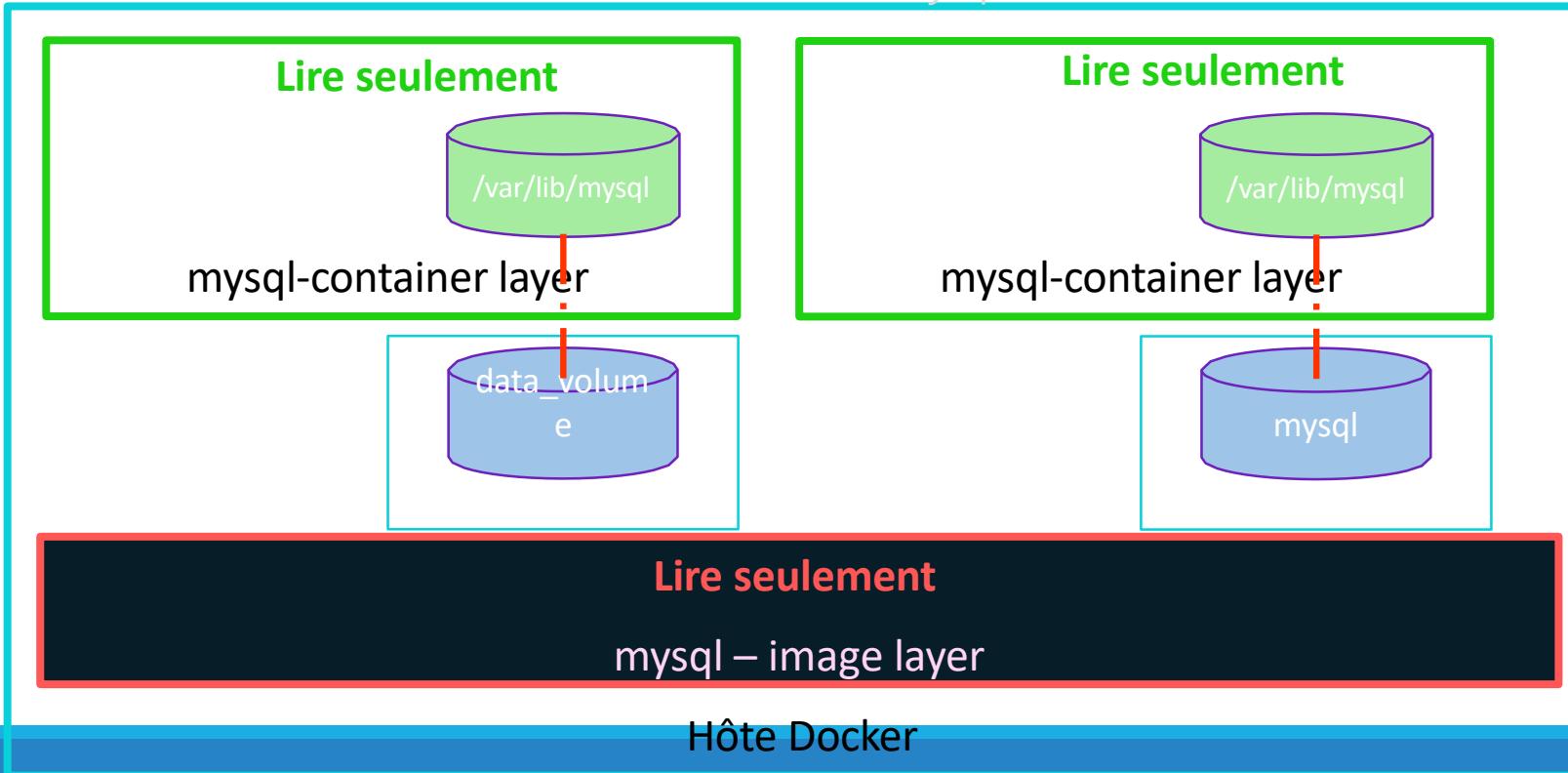


```
docker run -v data_volume:/var/lib/mysql mysql
```

```
docker run -v data_volume2:/var/lib/mysql mysql
```

```
docker run -v /data/mysql:/var/lib/mysql mysql
```

```
docker run \
--mount type=bind,source=/data/mysql,target=/var/lib/mysql
mysql
```



---

d o c k e r

composer

# Composer Docker

```
docker run mmumshad/simple-webapp
```

```
docker run mongodb
```

```
docker run redis:alpine
```

```
docker run ansible
```

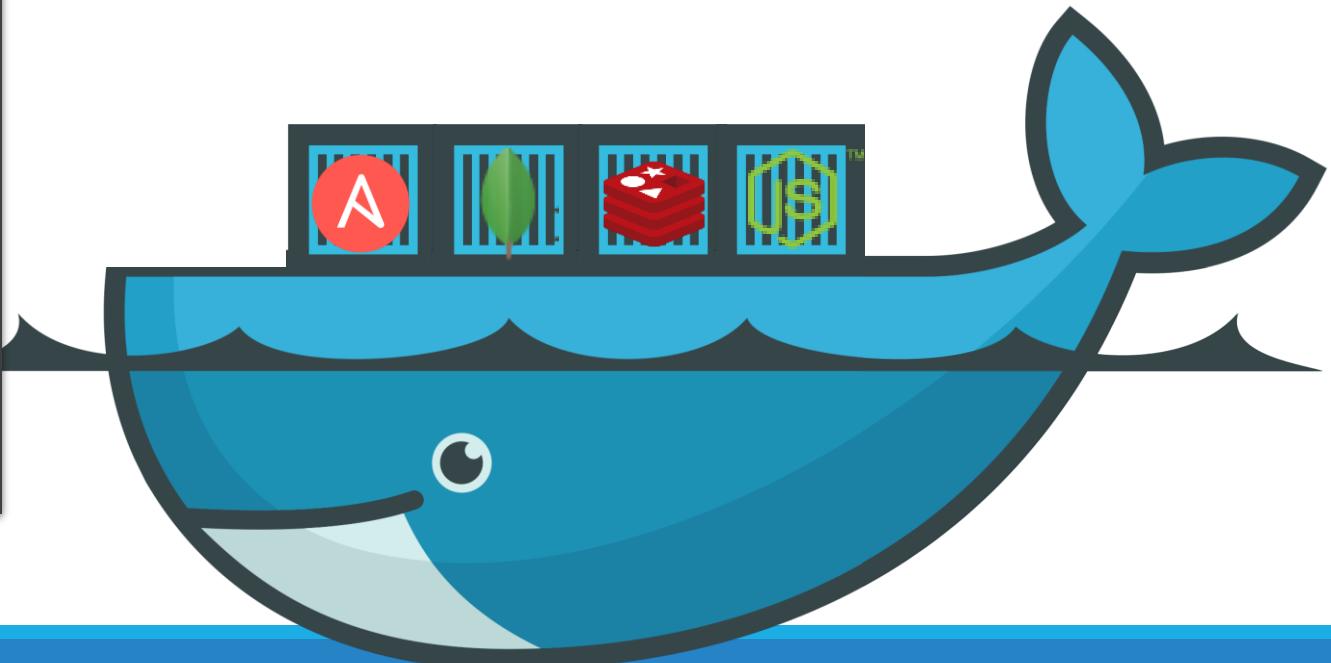
docker-compose.yml

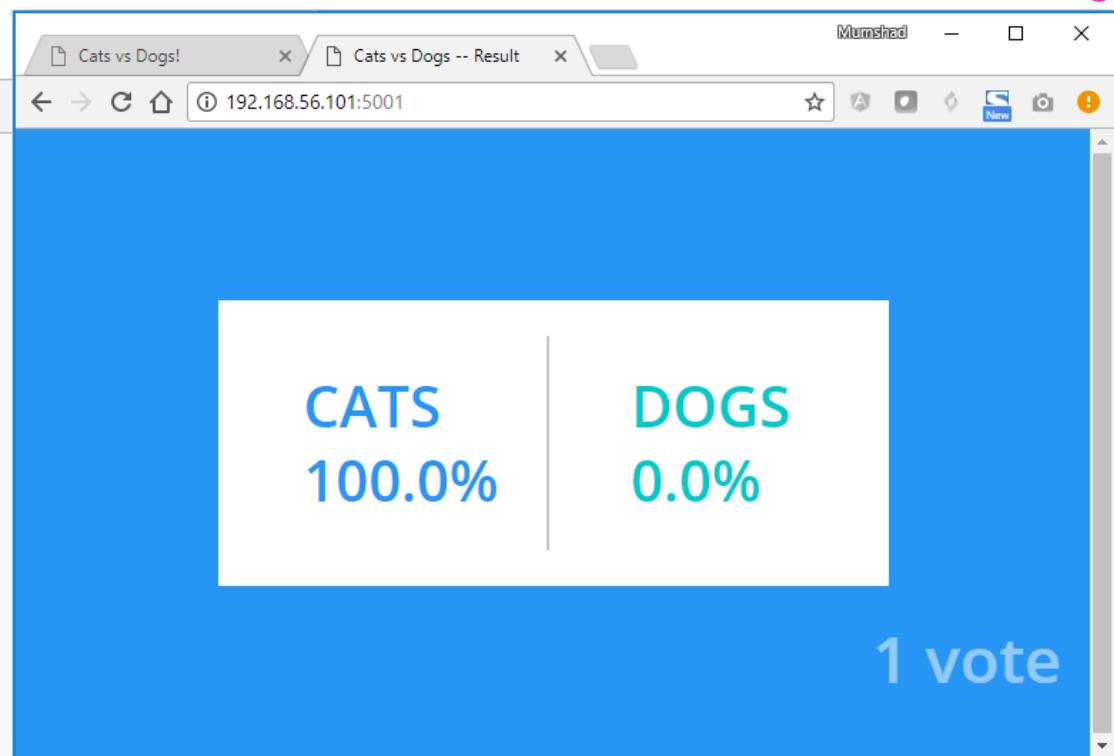
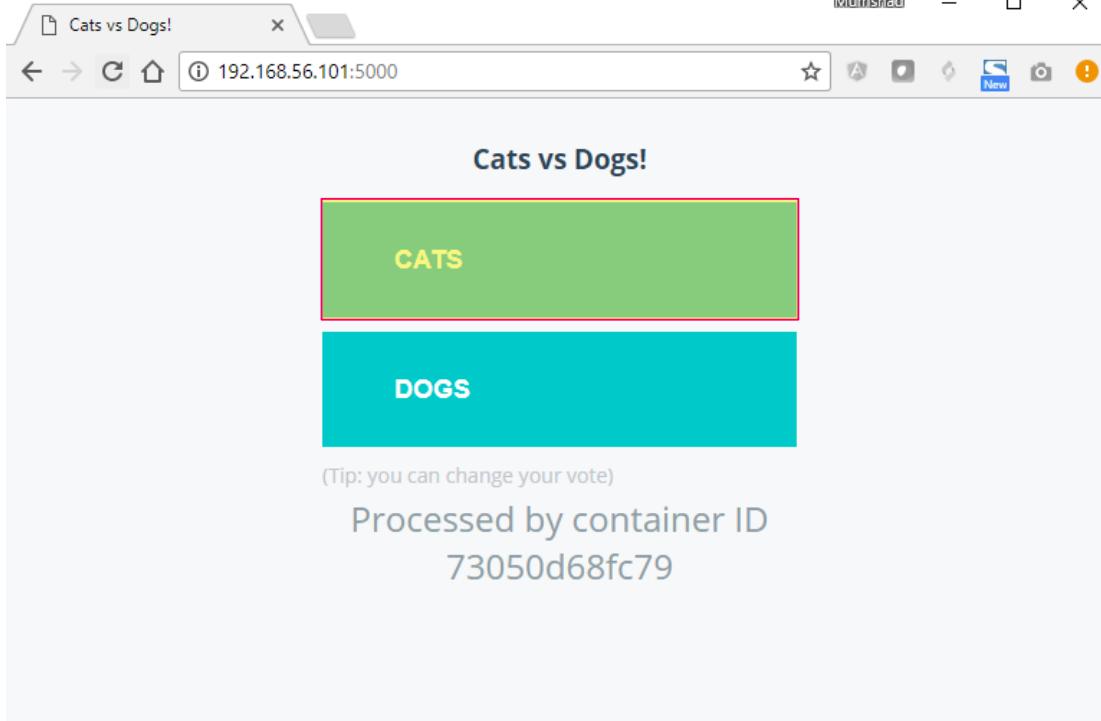
```
services:  
  web:  
    image: "mmumshad/simple-webapp"  
  database:  
    image: "mongodb"  
  messaging:  
    image: "redis:alpine"  
  orchestration:  
    image: "ansible"
```

```
docker-compose up
```



Public Docker registry - dockerhub

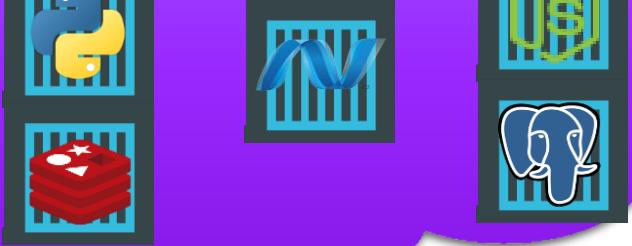




CATS	DOGS
1	0



# Docker Exécuter --liens



```
docker run -d --name=redis redis
```

```
docker run -d --name=db
```

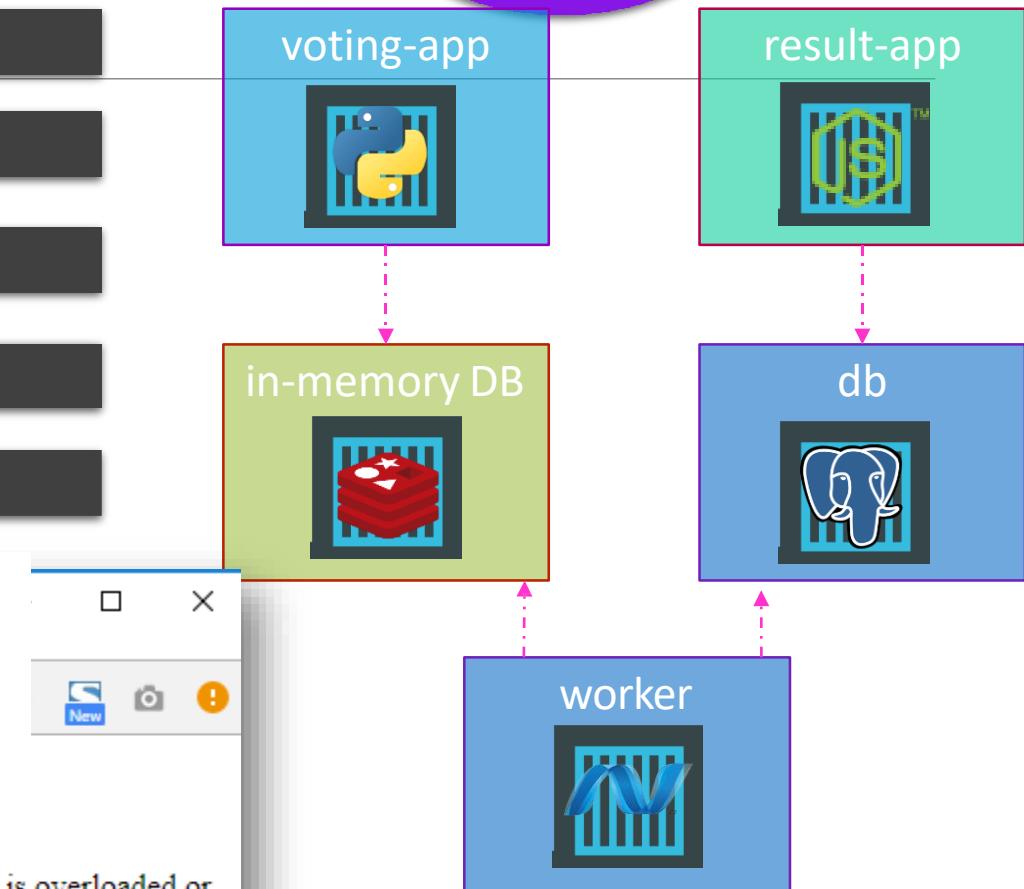
```
docker run -d --name=vote -p 5000:80 --votilinkng-rediapp  
sub
```

```
docker run -d --name=result -p 5001:80 -reslinkt-db:dpbb
```

```
docker run -d --name=worker workmkdb:db --link redis:redis
```

```
try {  
    Jedis redis = connectToRedis("redis");  
    Connection dbConn = connectToDB("db");  
  
    System.out.println("Watching vote queue");  
  
    redis.set("key", "value");  
}  
  
127.0.0.1      localhost  
::1      localhost ip6-localhost ip6-loopback  
fe00::0 ip6-localnet  
ff00::0 ip6-mcastprefix  
ff02::1 ip6-allnodes  
ff02::2 ip6-allrouters  
172.17.0.2      redis 89cd8eb563da  
172.17.0.3      ebcae9eb46bf
```

The server encountered an error while processing this request. There is an error in the configuration of the server or the server is overloaded or has insufficient resources.



Deprecation Warning

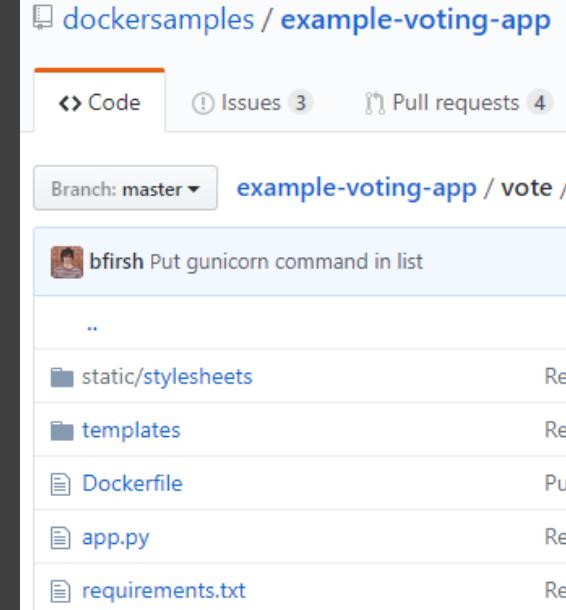
# Docker composer - construire

docker-compose.yml

```
redis:
  image: redis
db:
  image: postgres:9.4
vote:
  image: voting-app
  ports:
    - 5000:80
  links:
    - redis
result:
  image: result
  ports:
    - 5001:80
  links:
    - db
worker:
  image: worker
  links:
    - db
    - redis
```

docker-compose.yml

```
redis:
  image: redis
db:
  image: postgres:9.4
vote:
  build: ./vote
  ports:
    - 5000:80
  links:
    - redis
result:
  build: ./result
  ports:
    - 5001:80
  links:
    - db
worker:
  build: ./worker
  links:
    - db
    - redis
```



# Docker composer - versions

docker-compose.yml

```
redis:  
    image: redis  
  
db:  
    image: postgres:9.4  
  
vote:  
    image: voting-app  
    ports:  
        - 5000:80  
    links:  
        - redis
```

docker-compose.yml

```
version: 2  
services:  
    redis:  
        image: redis  
    db:  
        image: postgres:9.4  
    vote:  
        image: voting-app  
        ports:  
            - 5000:80  
        depends_on:  
            - redis
```

docker-compose.yml

```
version: 3  
services:
```

version: 1

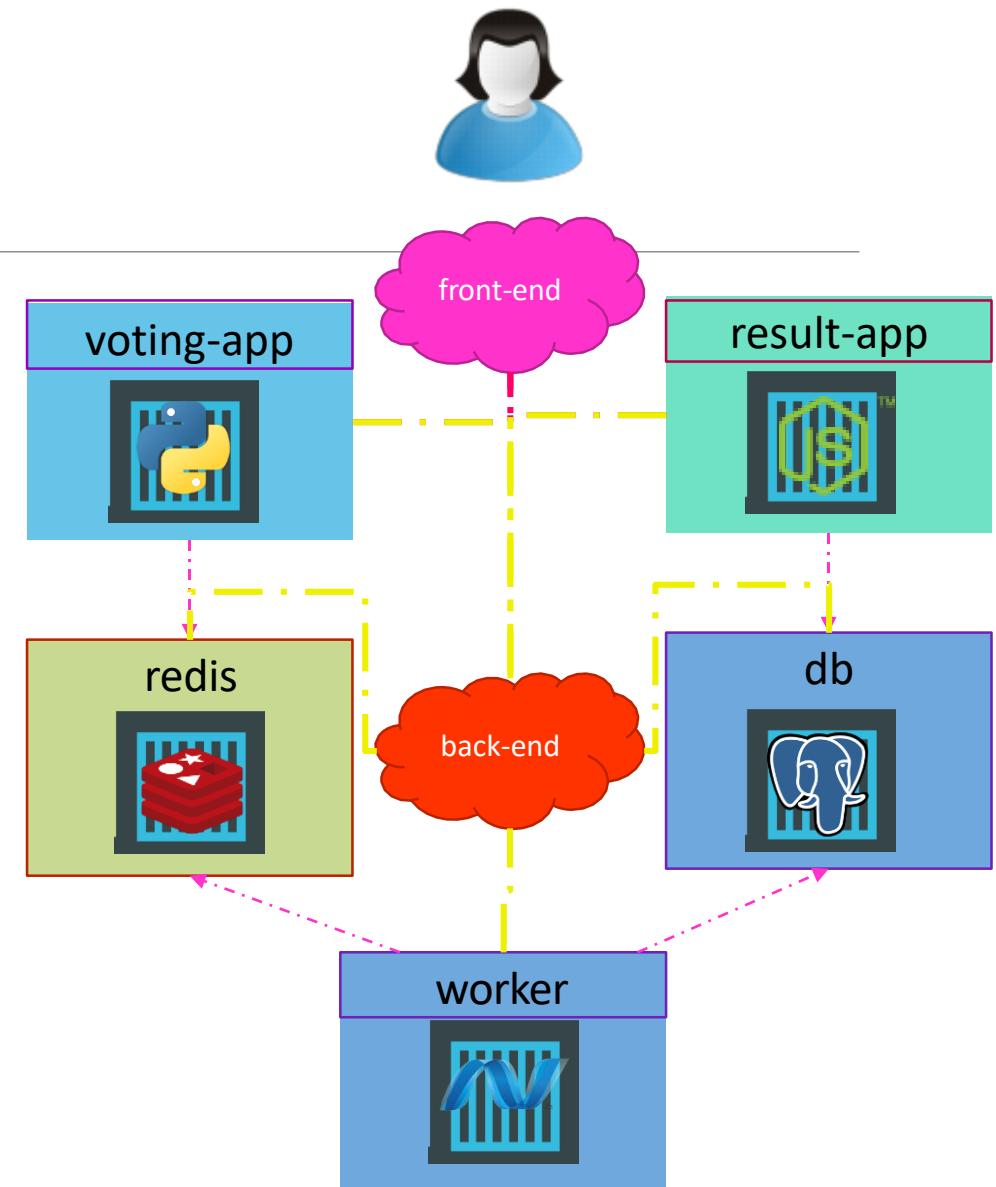
version: 2

version: 3

# Docker composer

docker-compose.yml

```
version: 2
services:
  redis:
    image: redis
    networks:
      - back-end
  db:
    image: postgres:9.4
    networks:
      - back-end
  vote:
    image: voting-app
    networks:
      - front-end
      - back-end
  result:
    image: result
    networks:
      - front-end
      - back-end
networks:
  front-end:
  back-end:
```



---

d o c k e r

---

# Enregistrement

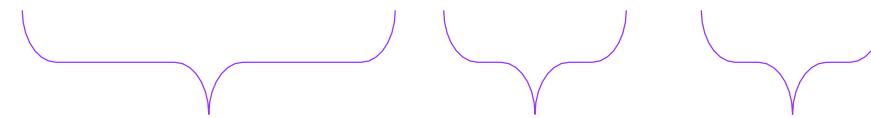
mag

e

---

▶ docker run nginx

**image:** docker.io/nginx/nginx



Registry      User/      Image/  
                  Account    Repository

gcr.io/kubernetes-e2e-test-images/dnsutils

# Enregistrement privé

```
▶ docker login private-registry.io
```

```
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to  
https://hub.docker.com to create one.
```

```
Username: registry-user
```

```
Password:
```

```
WARNING! Your password will be stored unencrypted in /home/vagrant/.docker/config.json.
```

```
Login Succeeded
```

```
▶ docker run private-registry.io/apps/internal-app
```

# Déployer enregistrement privé

```
▶ docker run -d -p 5000:5000 --name registry registry:2
```

---

```
▶ docker image tag my-image localhost:5000/my-image
```

```
▶ docker push localhost:5000/my-image
```

```
▶ docker pull localhost:5000/my-image
```

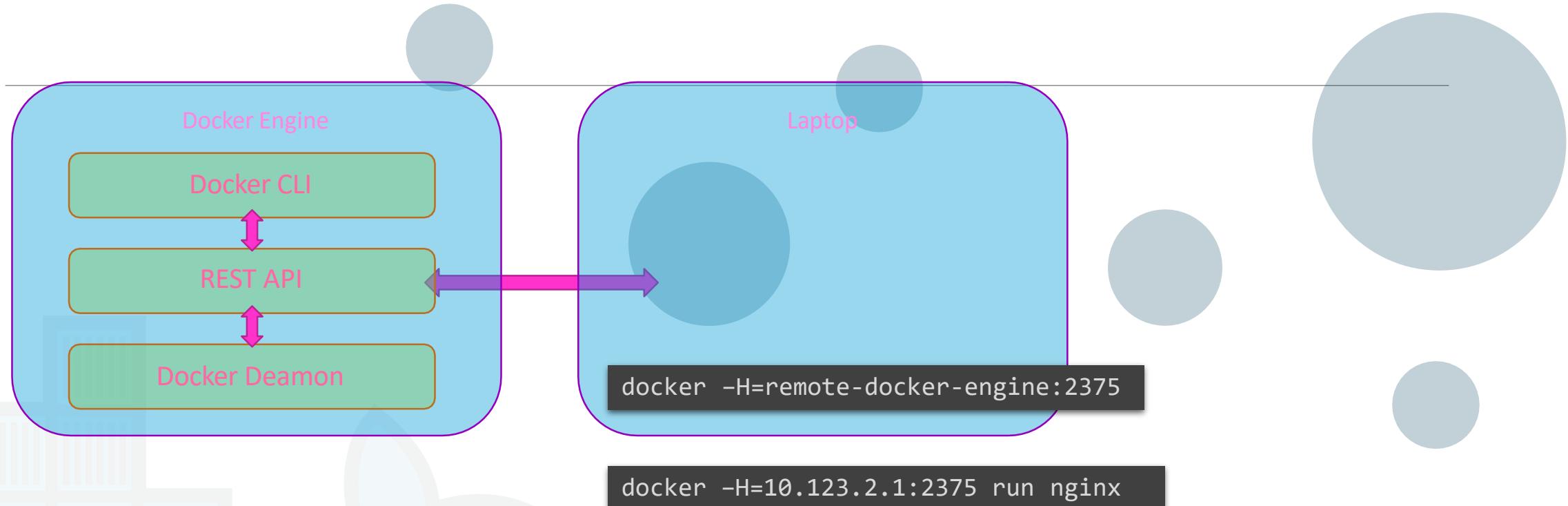
```
▶ docker pull 192.168.56.100:5000/my-image
```

---

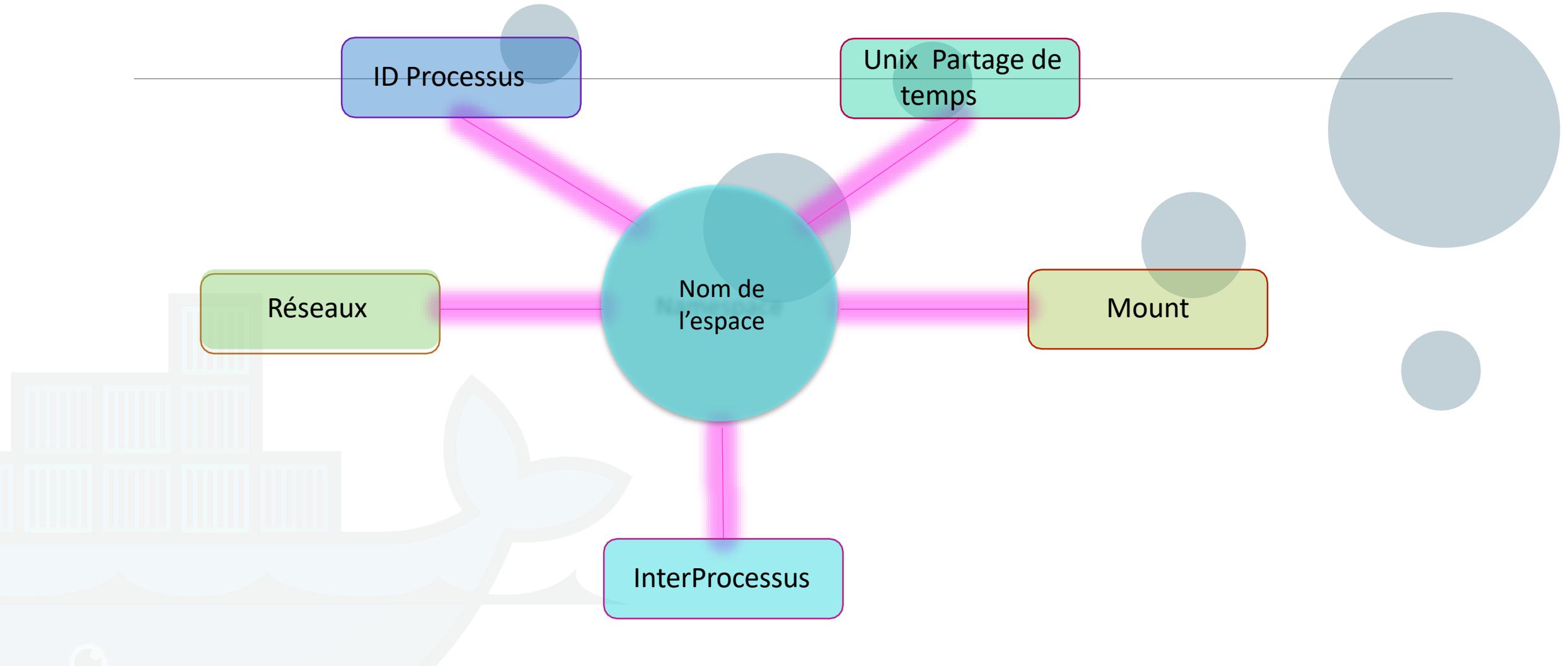
d o c k e r

Moteur

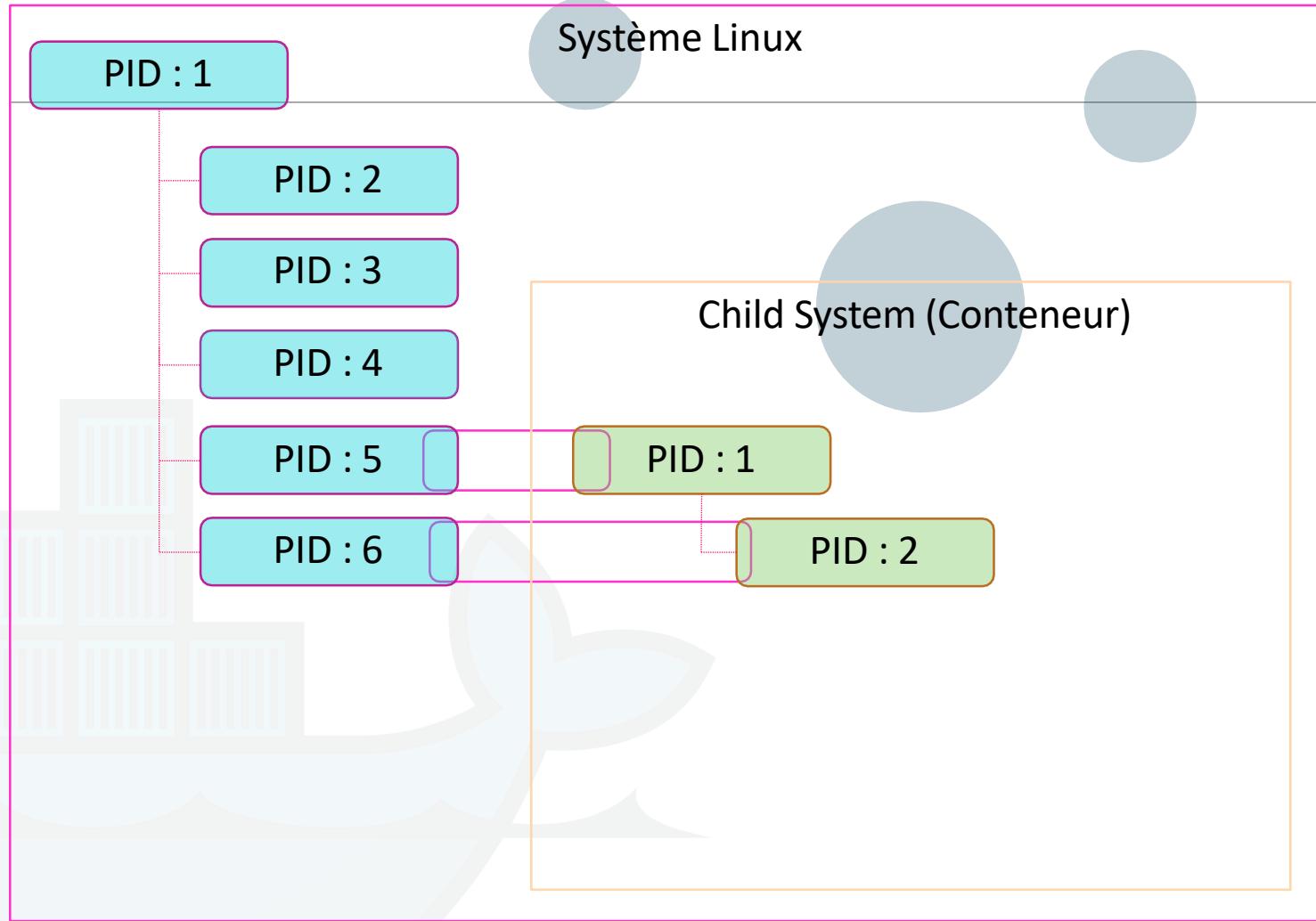
# Moteur Docker



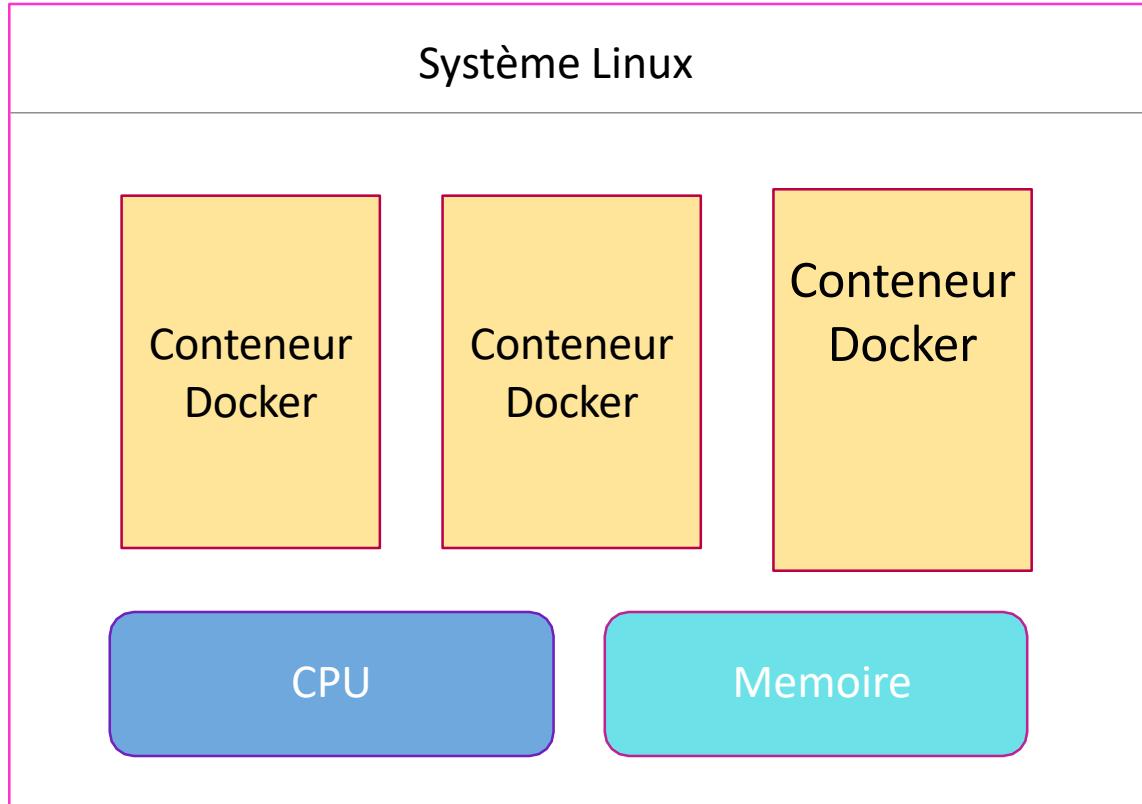
# Conteneurisation



# Nom de l' espace - PID



# cgroupes



```
docker run --cpus=.5 ubuntu
```

```
docker run --memory=100m ubuntu
```

---

d o c k e r

# Sous Windows

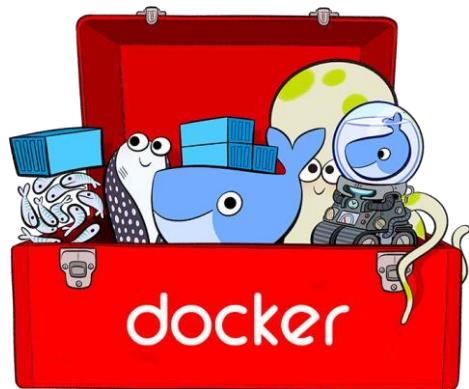
# Docker sur windows

- 
1. Docker sur Windows en utilisant Docker Toolbox
  2. Docker Desktop pour Windows

# 1. Docker Boites à outils



- 64-bit operating
- Windows 7 or higher.
- Virtualization is enabled

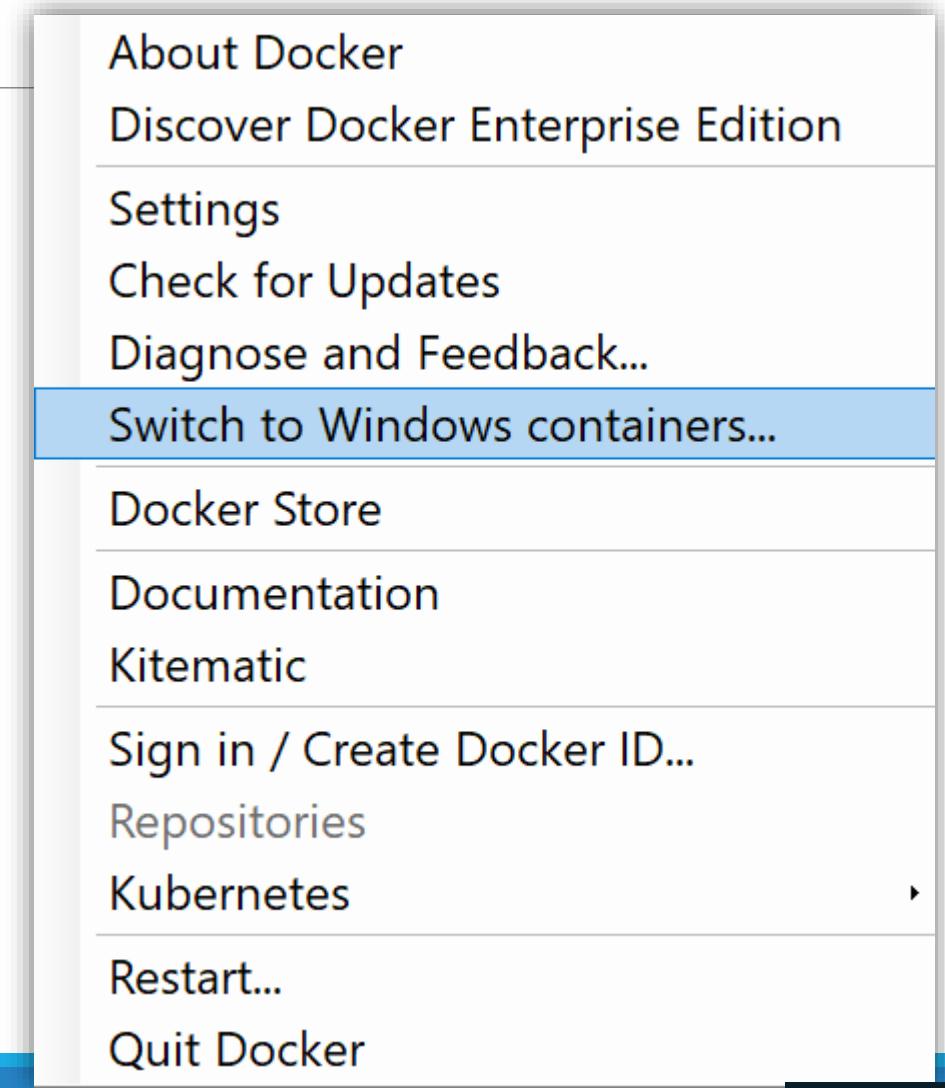


- Oracle Virtualbox
- Docker Engine
- Docker Machine
- Docker Compose
- Kitematic GUI

## 2. Docker Desktop pour Windows

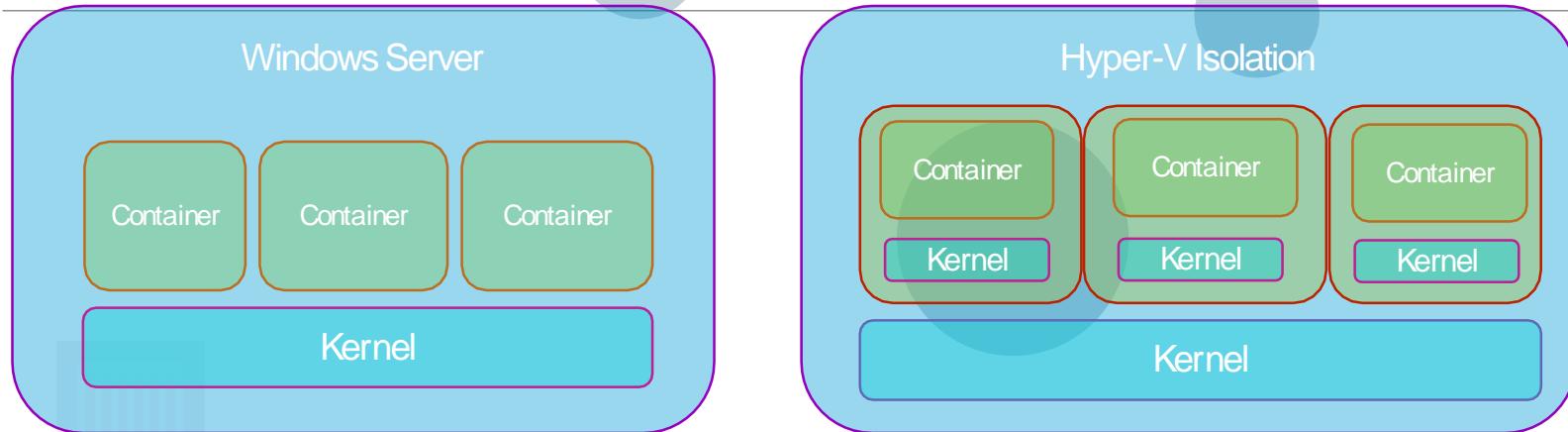


Linux Containers (Default)  
Or  
Windows Containers



# Conteneurs Windows

## Types Conteneur

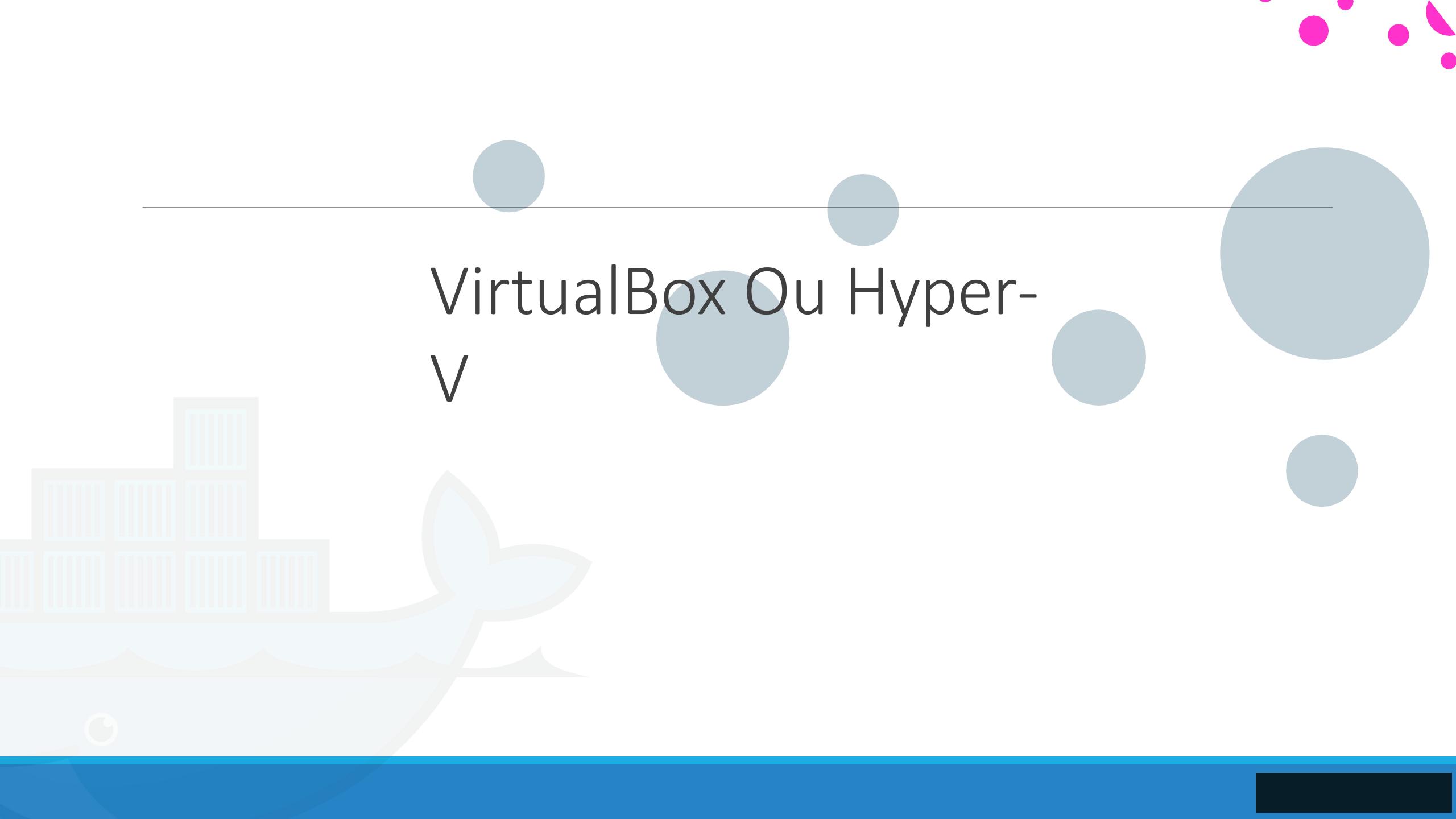


## Base Images:

- Windows Server Core
- Nano Server

## Support

- Windows Server 2016
- Nano Server
- Windows 10 Professional and Enterprise (Hyper-V Isolated Containers)



VirtualBox Ou Hyper-V



---

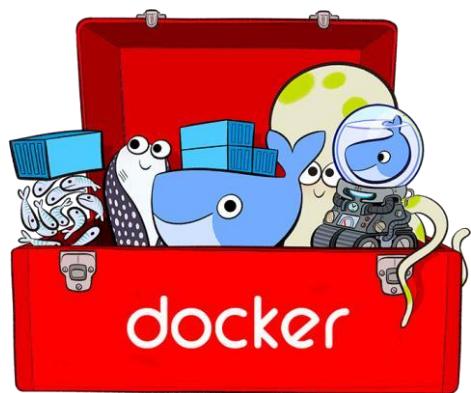
d o c k e r

# Sur Mac

# Docker sur Mac

- 
1. Docker sur Mac en utilisant la boite à outil Docker
  2. Docker Desktop sur Mac

# 1. Boîte à outil Docker



## 2. Docker Desktop pour Mac



HyperKit

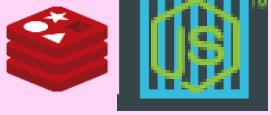
Conteneurs Linux

---

conteneur

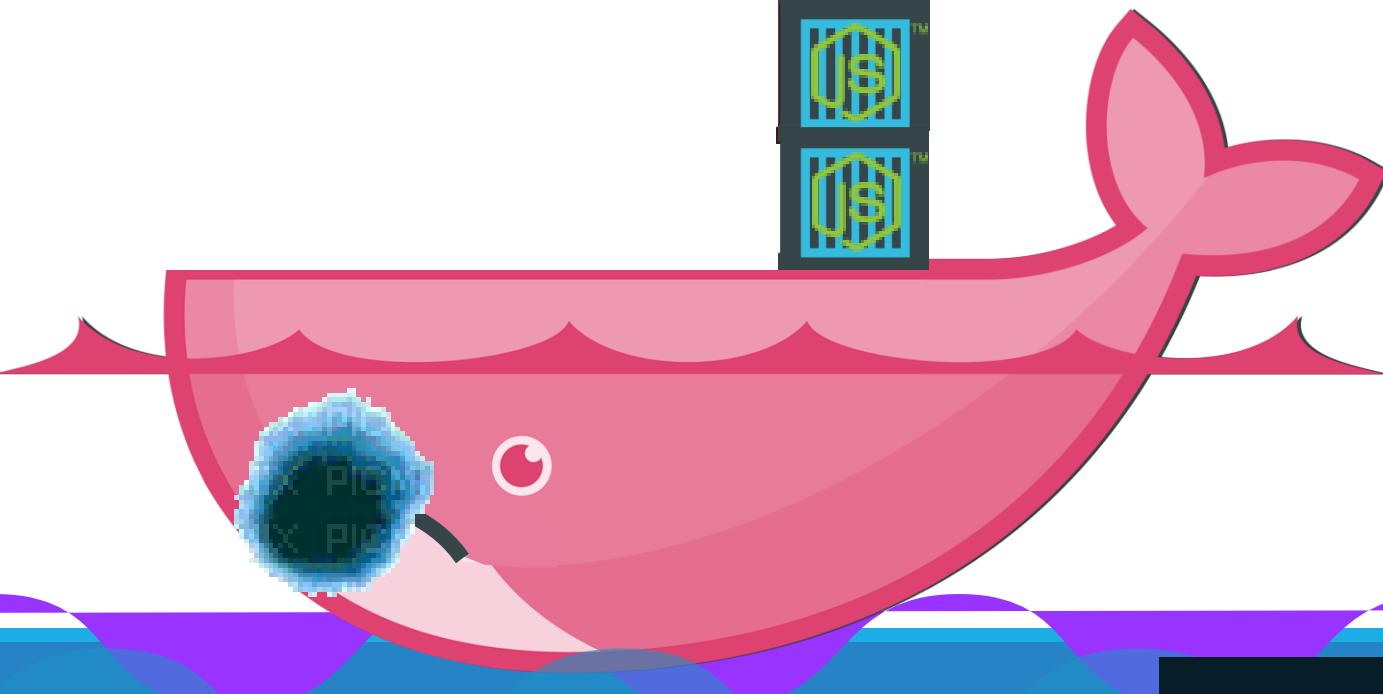
orchestration

# Pourquoi orchestrer?



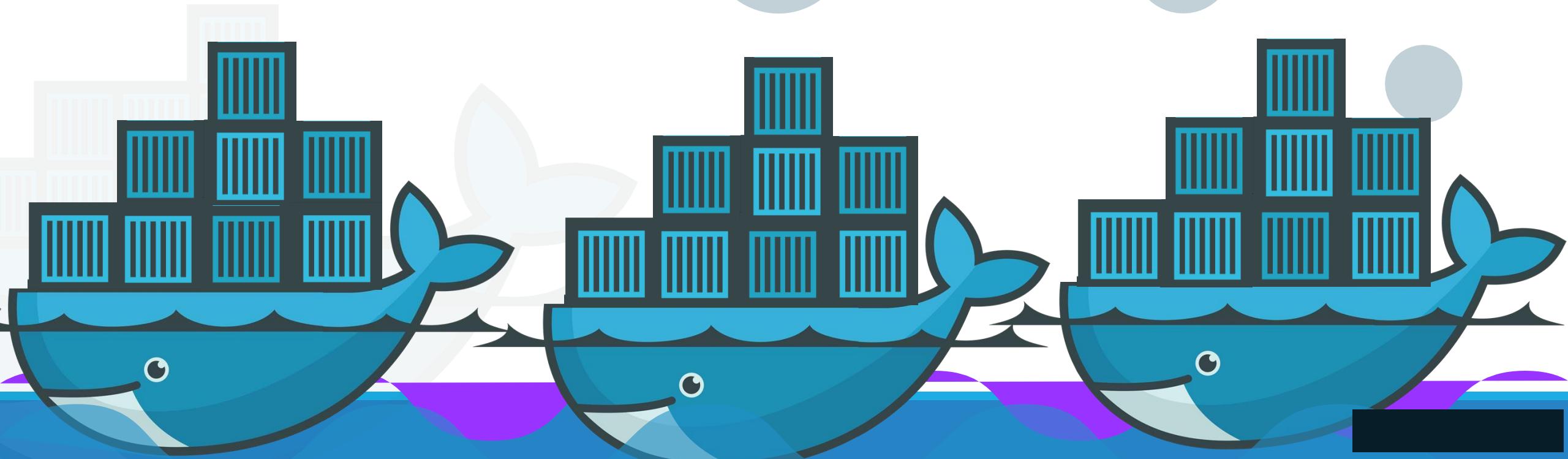
Public Docker registry - dockerhub

```
docker run nodejs  
docker run nodejs  
docker run nodejs  
docker run nodejs
```



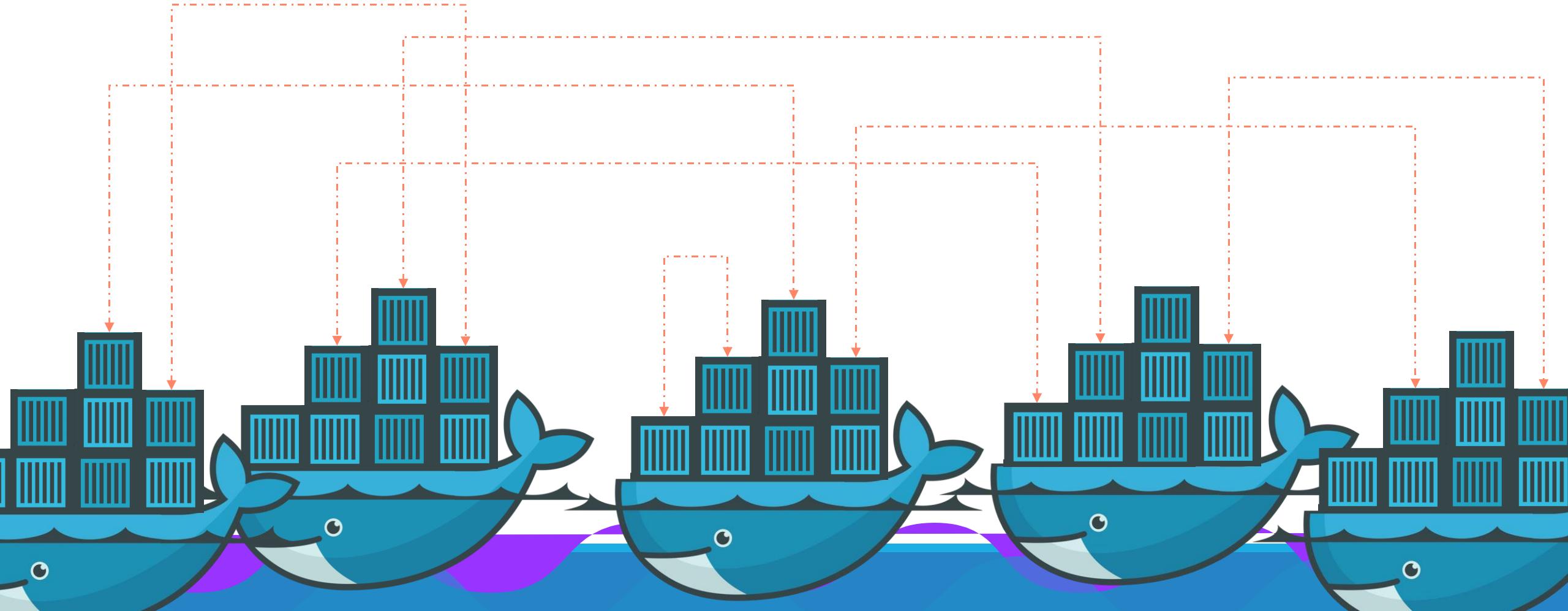
# Orchestration de Conteneur

```
docker service create --replicas=100 nodejs
```



# Orchestration de Conteneur

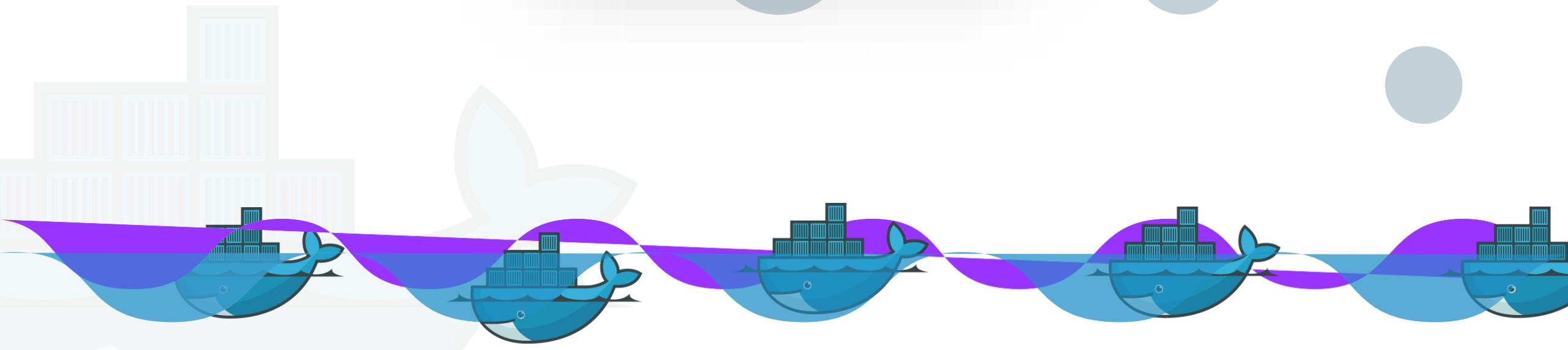
```
docker service create --replicas=100 nodejs
```





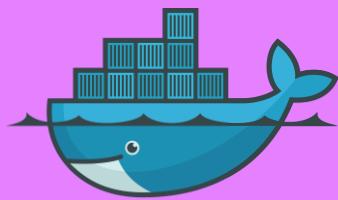
# d o c k e r

# Swarm



# Solutions

---



Docker Swarm



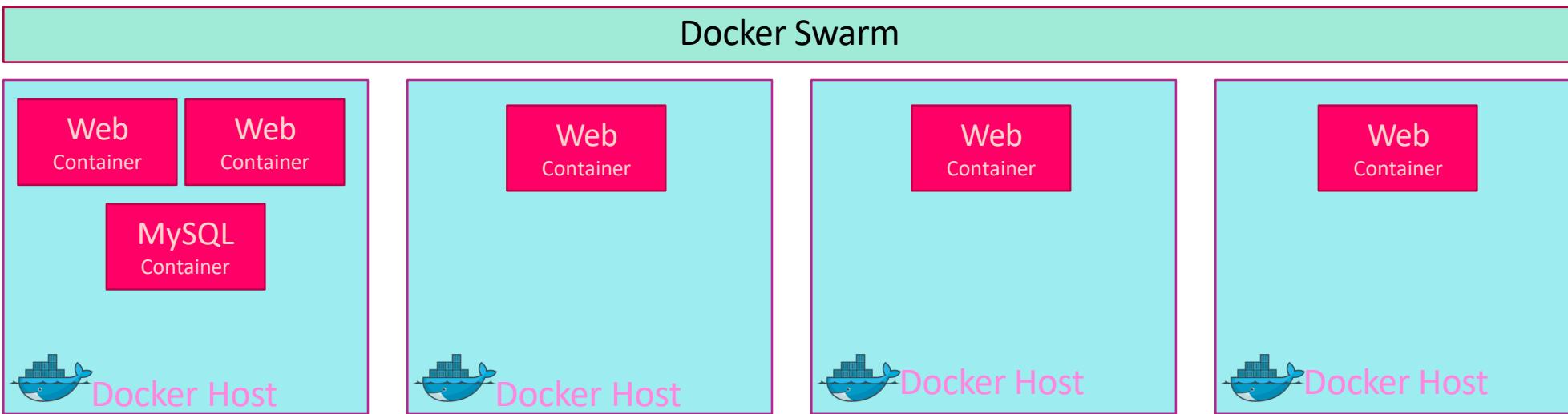
kubernetes



MESOS

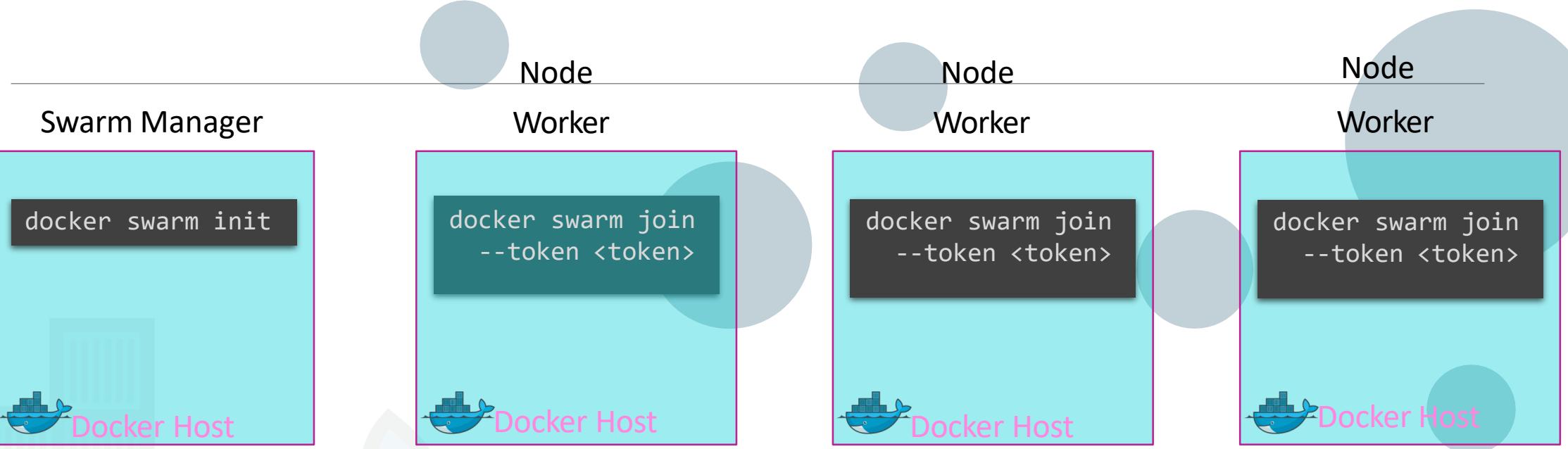
# Docker swarm

---



# Setup

## swarm



```
root@osboxes:/root/simple-webapp-docker # docker swarm init --advertise-addr 192.168.1.12
Swarm initialized: current node (0j76dum2r56plxfne4ub1ps2c) is now a manager.
```

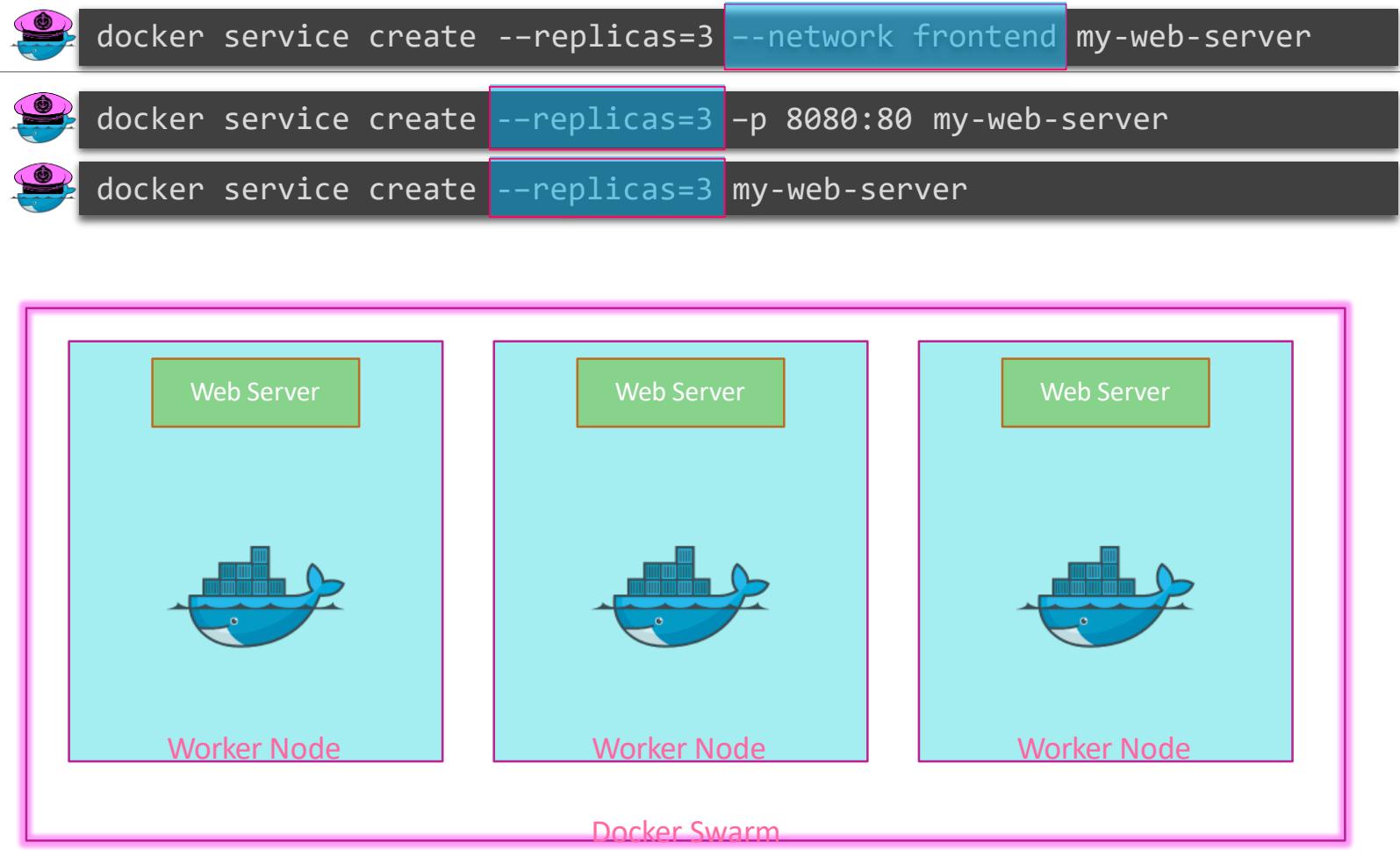
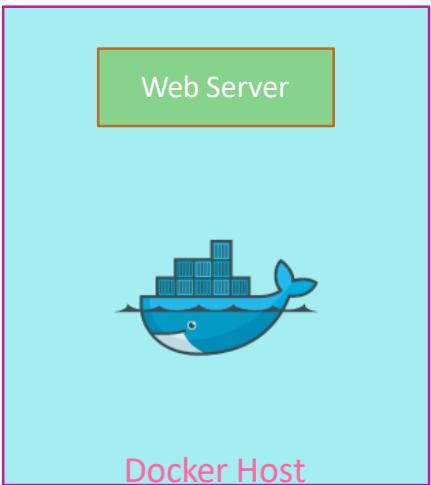
To add a worker to this swarm, run the following command:

```
root@osboxes:/root/simple-webapp-docker # docker swarm join --token SWMTKN-1-35va8b3fi5krpdskefqqxgttmulw3z828daucr7y526ne0sgu-2eek9qm33d4lxzoq6we9i8izp 192.168.1.12:2377
```

To add a manager to this swarm, run '`docker swarm join-token manager`' and follow the instructions.

# Docker service

```
docker run my-web-server
```



# kubernetes



```
 docker run my-web-server
```



```
 kubectl rolling-update my-web-server --rollback
```

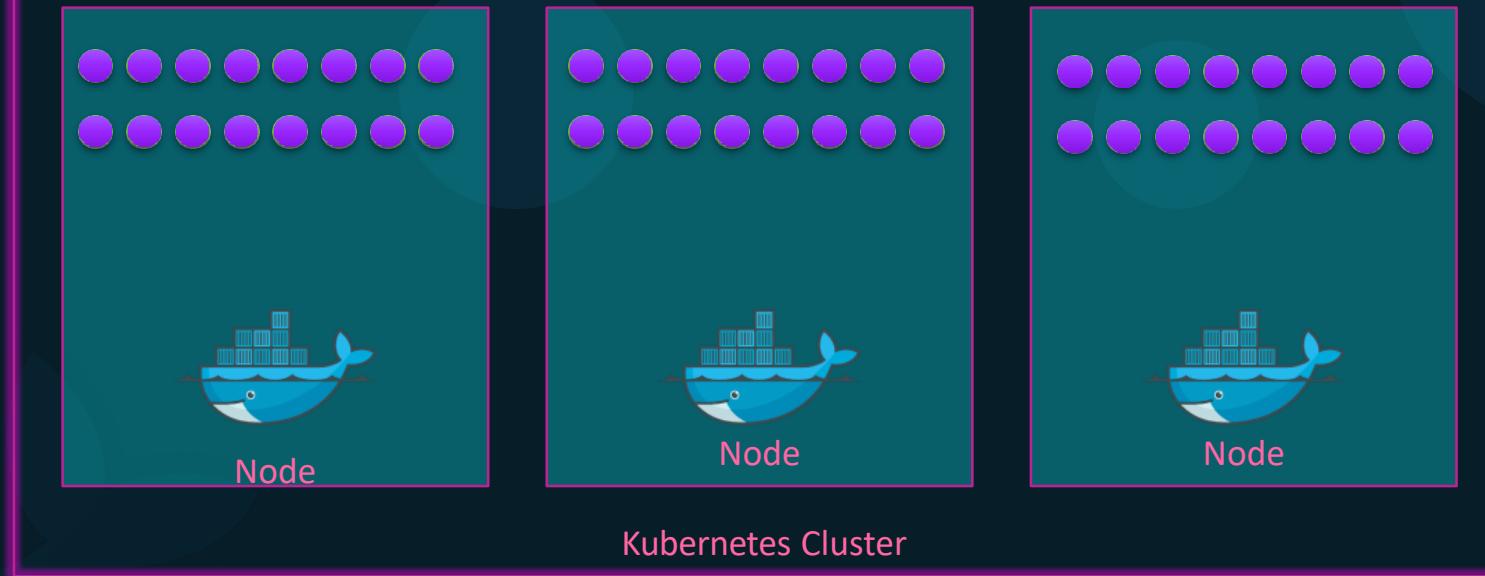
```
 kubectl  my-web-server --image=web-server:2
```

```
 kubectl scale  my-web-server
```

```
 kubectl run --replicas=1000 my-web-server
```

A

B



Security

POD AutoScalers

Cluster AutoScalers

Network

 weaveworks

 flannel

 cilium

 NSX  
vmware



Storage

 ceph

 GlusterFS

SCALEIO

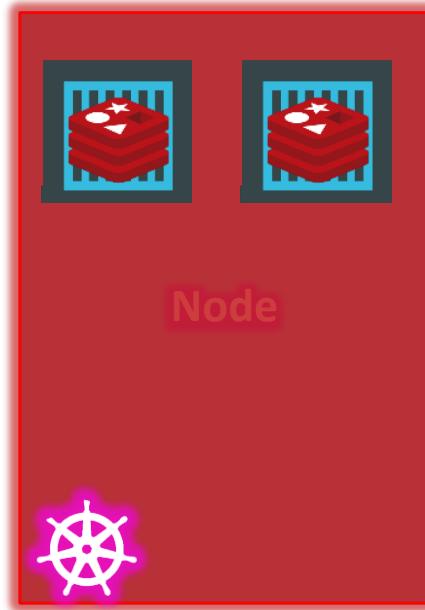
 Amazon Web Services™



 Flocker™  
by ClusterHQ®

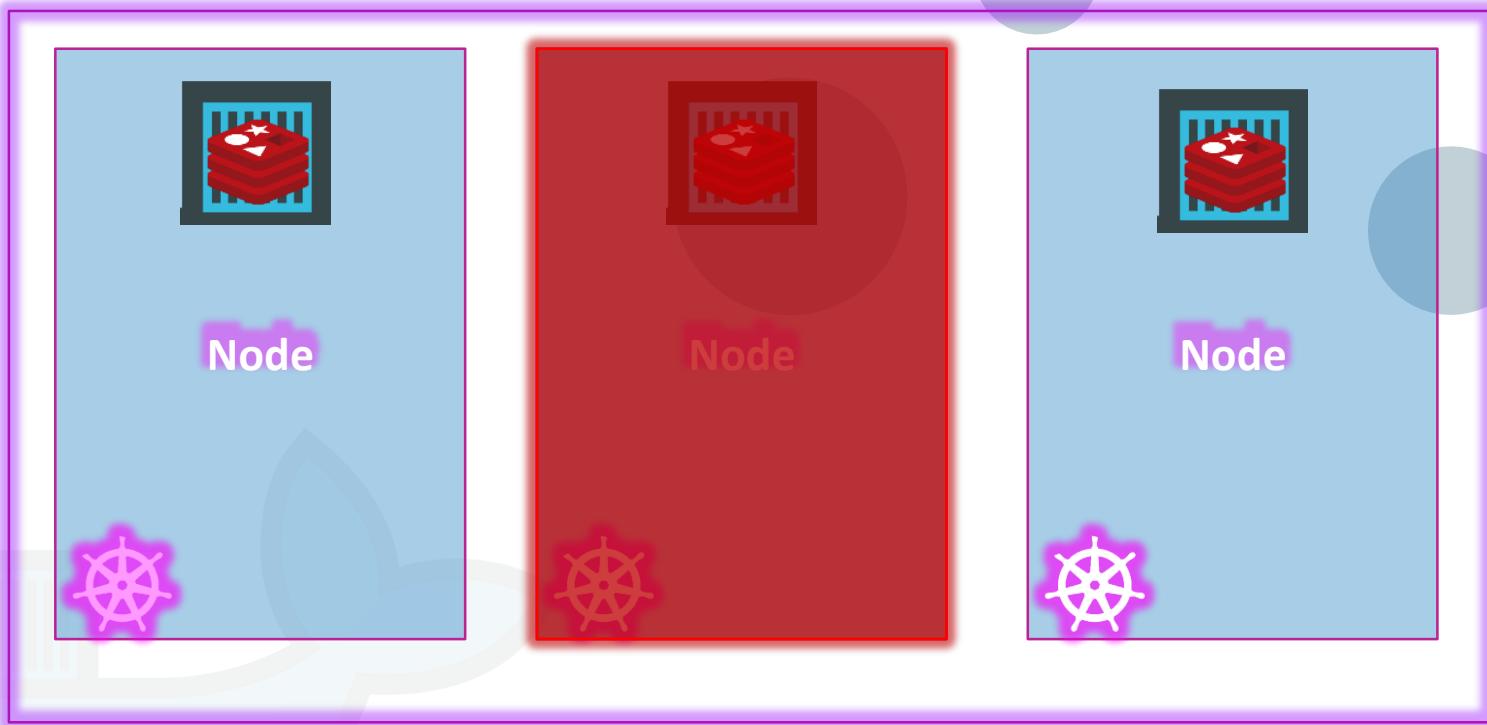
# Nodes (Minions)

---

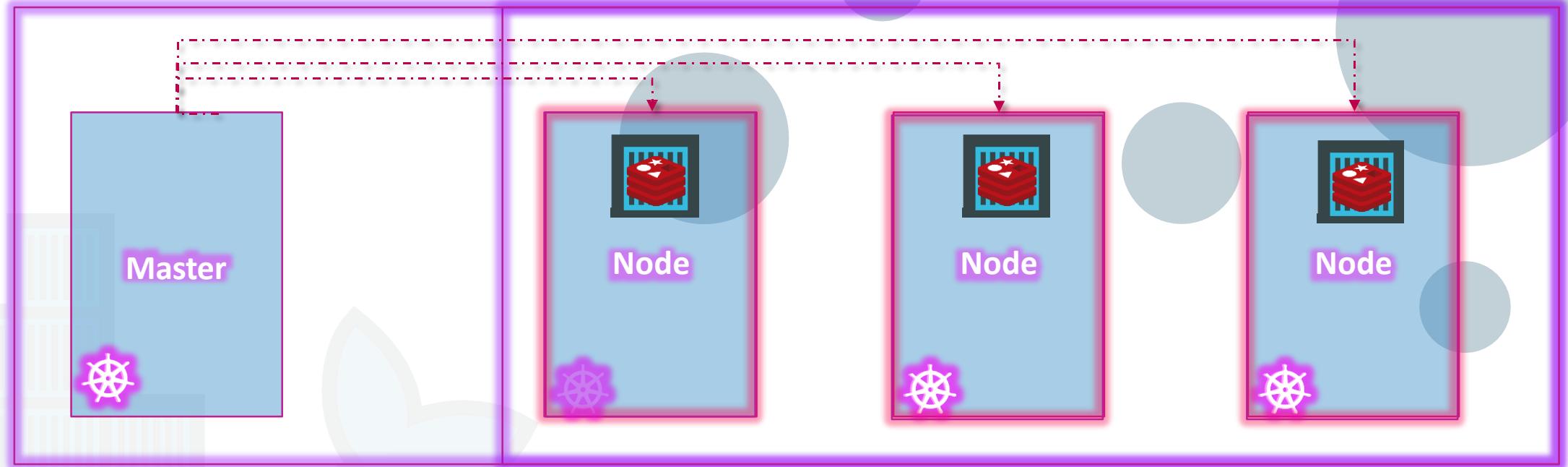


# Cluster

r



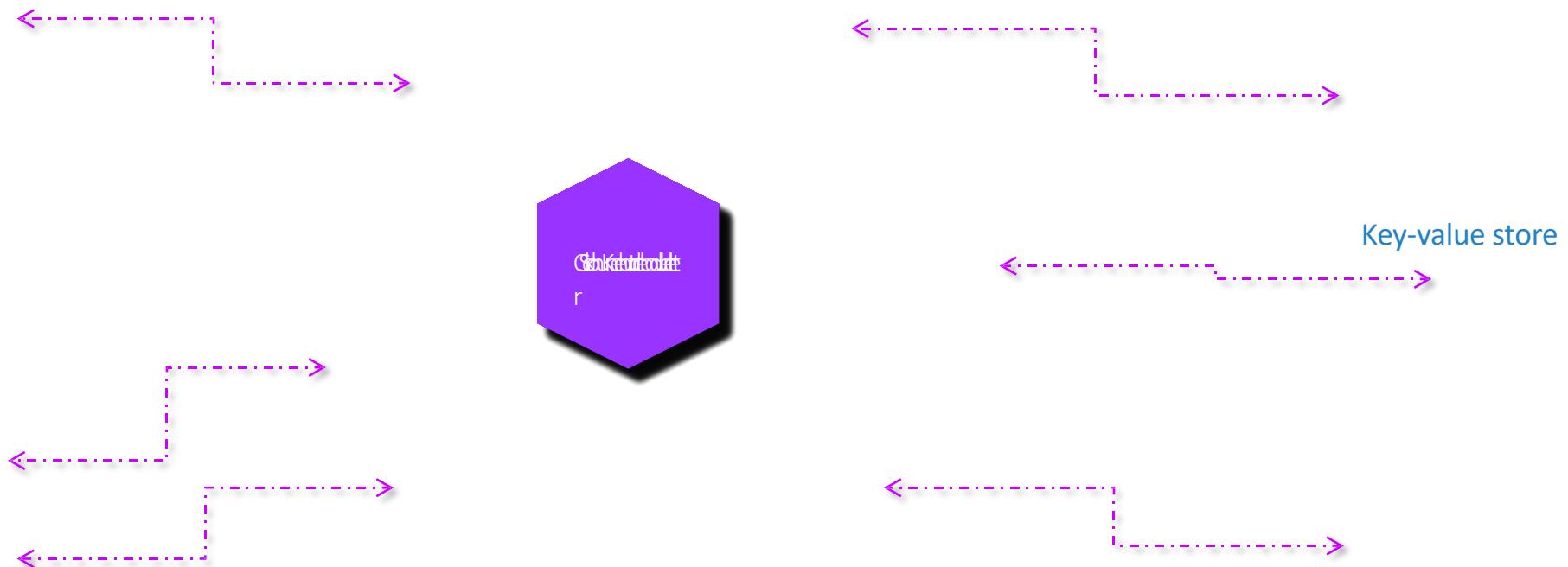
# Master r



# Components

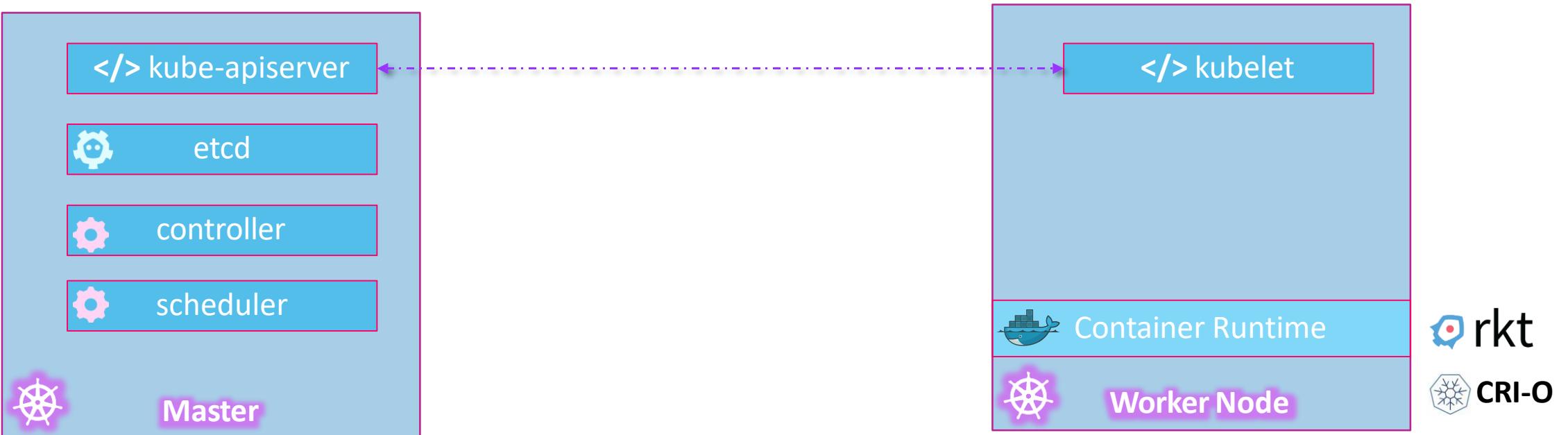
ts

---



# Master vs Worker Nodes

---



# Kubectl

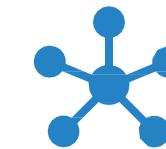
|

---

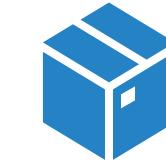
```
kubectl run hello-minikube
```



```
kubectl cluster-info
```



```
kubectl get nodes
```



```
kubectl run my-web-app --image=my-web-app --replicas=100
```



---

# CONCLUSION







---

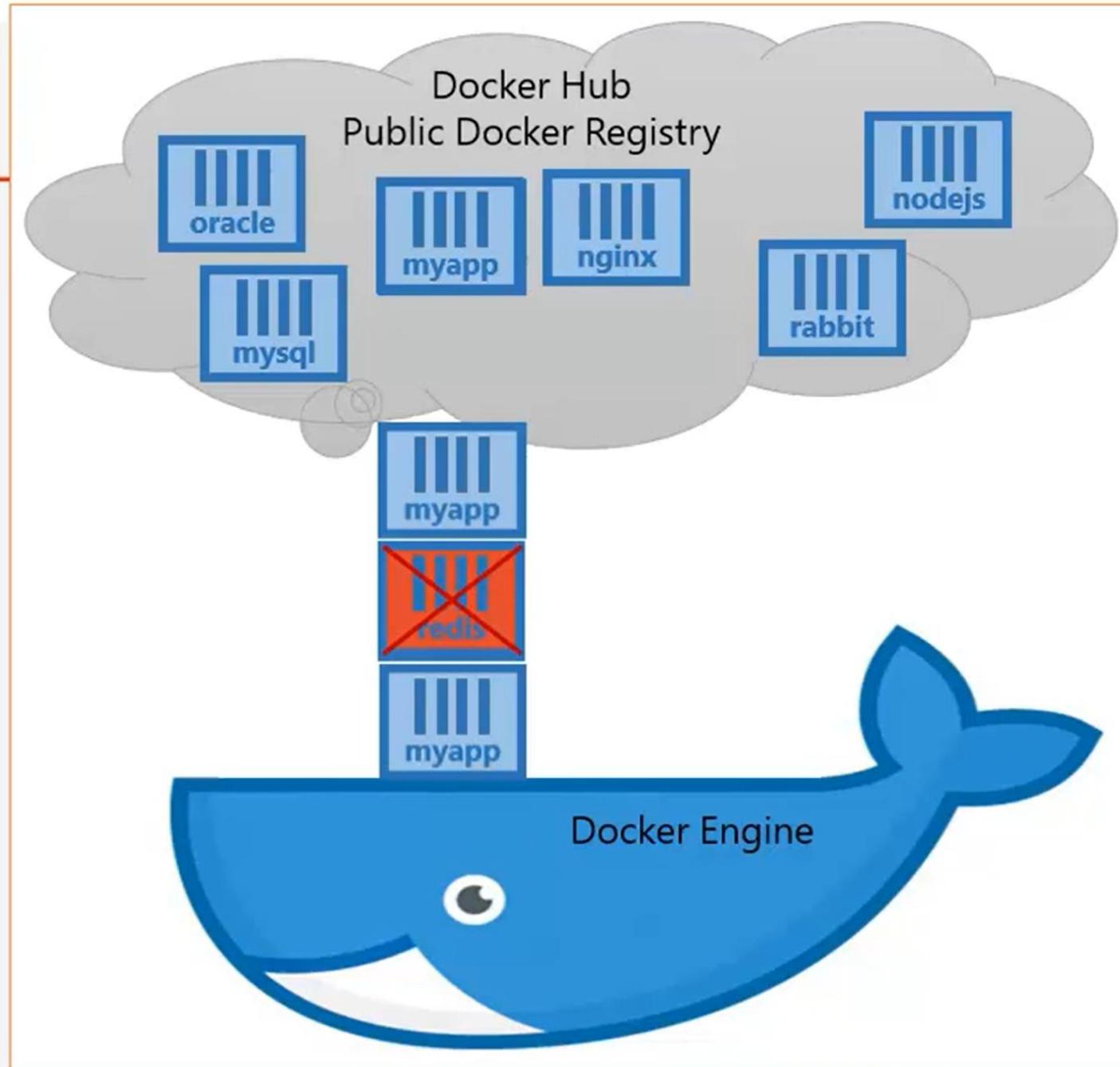
# Kubernetes

---

## Orchestration des conteneurs

```
$ docker run myapp  
$ docker run myapp  
$ docker run myapp
```

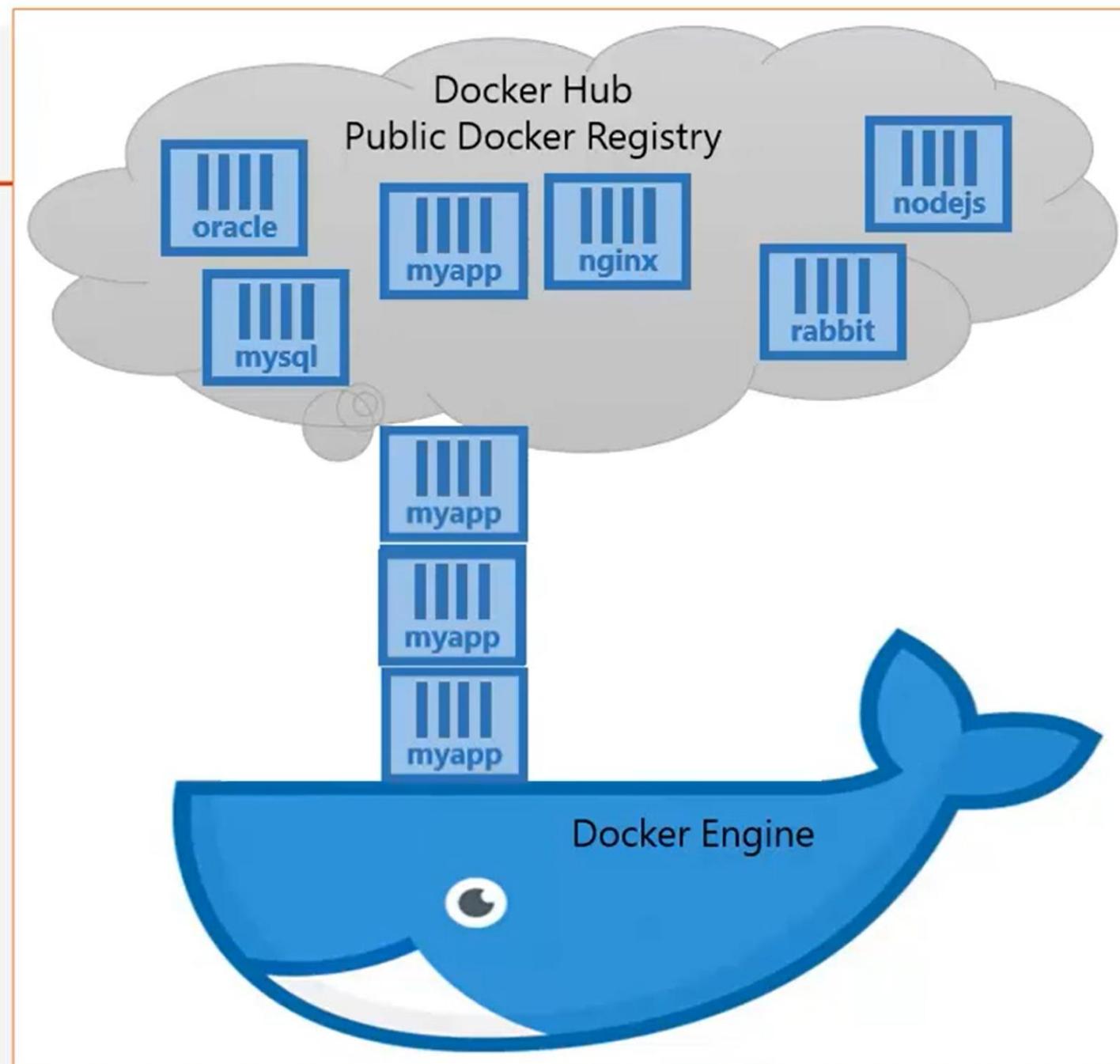
- Problème de Disponibilité et de monté en charge :
  - Il faut démarrer plusieurs instances en fonction de la charge
  - Si une instance tombe en panne, il faut la remplacer
  - Si Docker Engine tombe en panne, tous les conteneur vont cracher.
- L'orchestration des conteneurs est une solution pour ce problème



## Orchestration des conteneurs

```
$ docker run myapp  
$ docker run myapp  
$ docker run myapp  
$ docker run myapp
```

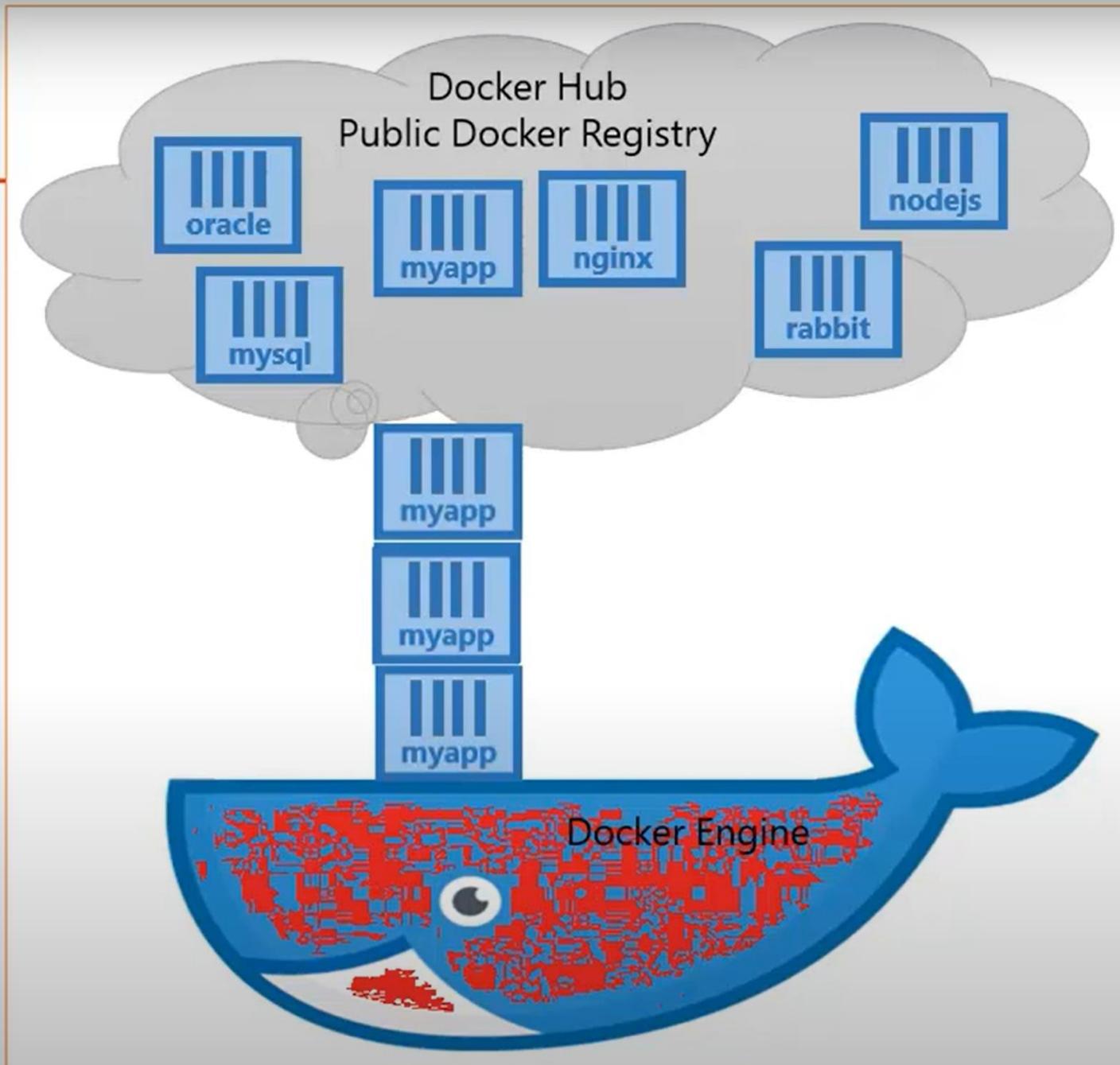
- Problème de Disponibilité et de monté en charge :
  - Il faut démarrer plusieurs instances en fonction de la charge
  - Si une instance tombe en panne, il faut la remplacer
  - Si Docker Engine tombe en panne, tous les conteneur vont cracher.
- L'orchestration des conteneurs est une solution pour ce problème



## Orchestration des conteneurs

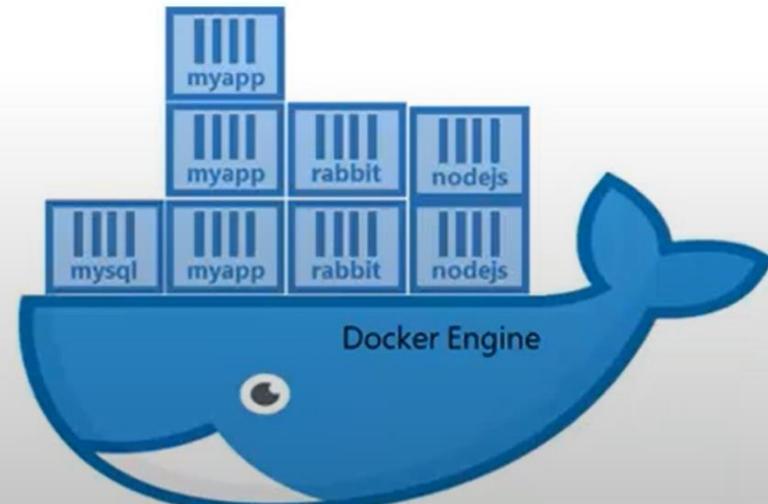
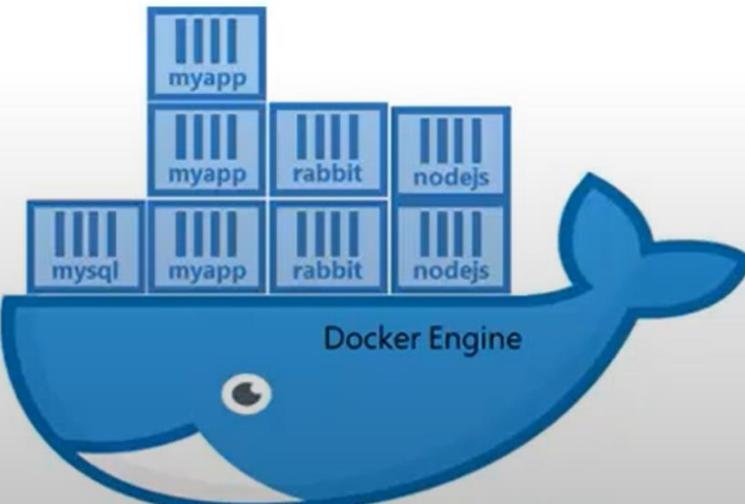
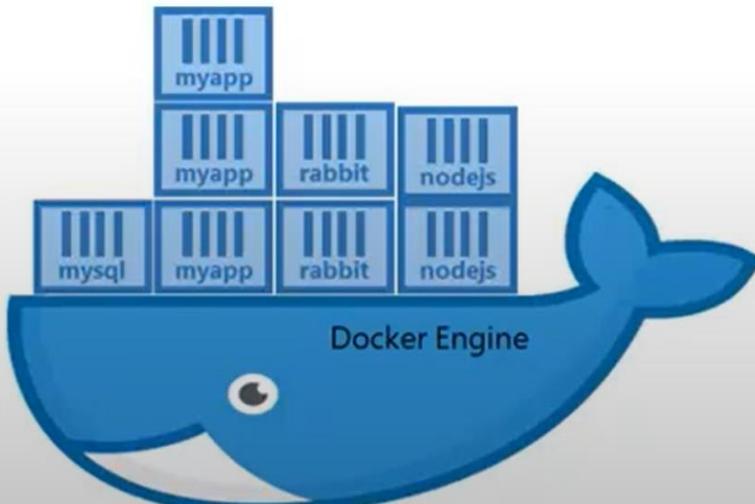
```
$ docker run myapp  
$ docker run myapp  
$ docker run myapp  
$ docker run myapp
```

- Problème de Disponibilité et de monté en charge :
  - Il faut démarrer plusieurs instances en fonction de la charge
  - Si une instance tombe en panne, il faut la remplacer
  - Si Docker Engine tombe en panne, tous les conteneur vont cracher.
- L'orchestration des conteneurs est une solution pour ce problème



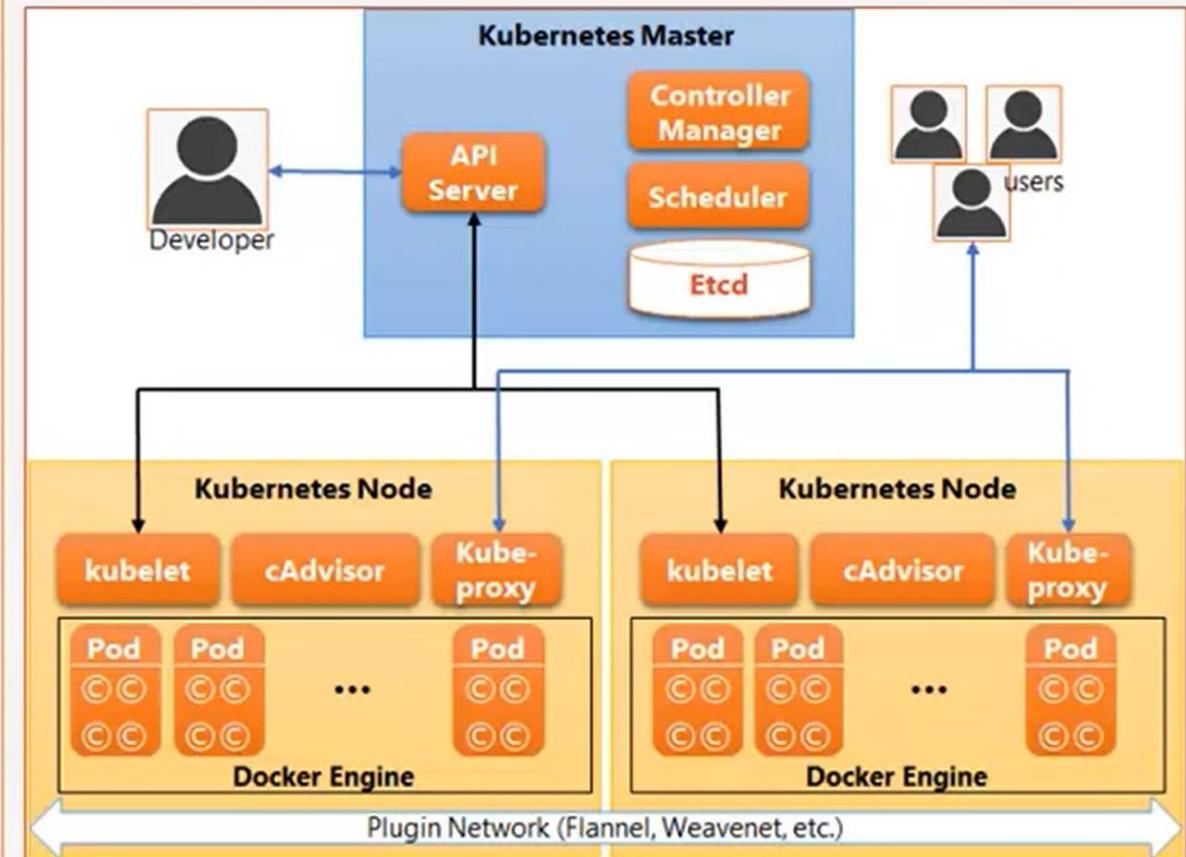
# Orchestration des conteneurs

- Consiste en un ensemble d'outils et de scripts qui permettent l'automatisation des opérations :
  - Scalabilité des conteneurs, RéPLICATION des conteneurs, Haute disponibilité, bilan de santé des conteneurs etc.
- Exemples d'outils :
  - Docker Swarm
  - Kubernetes
  - MESOS



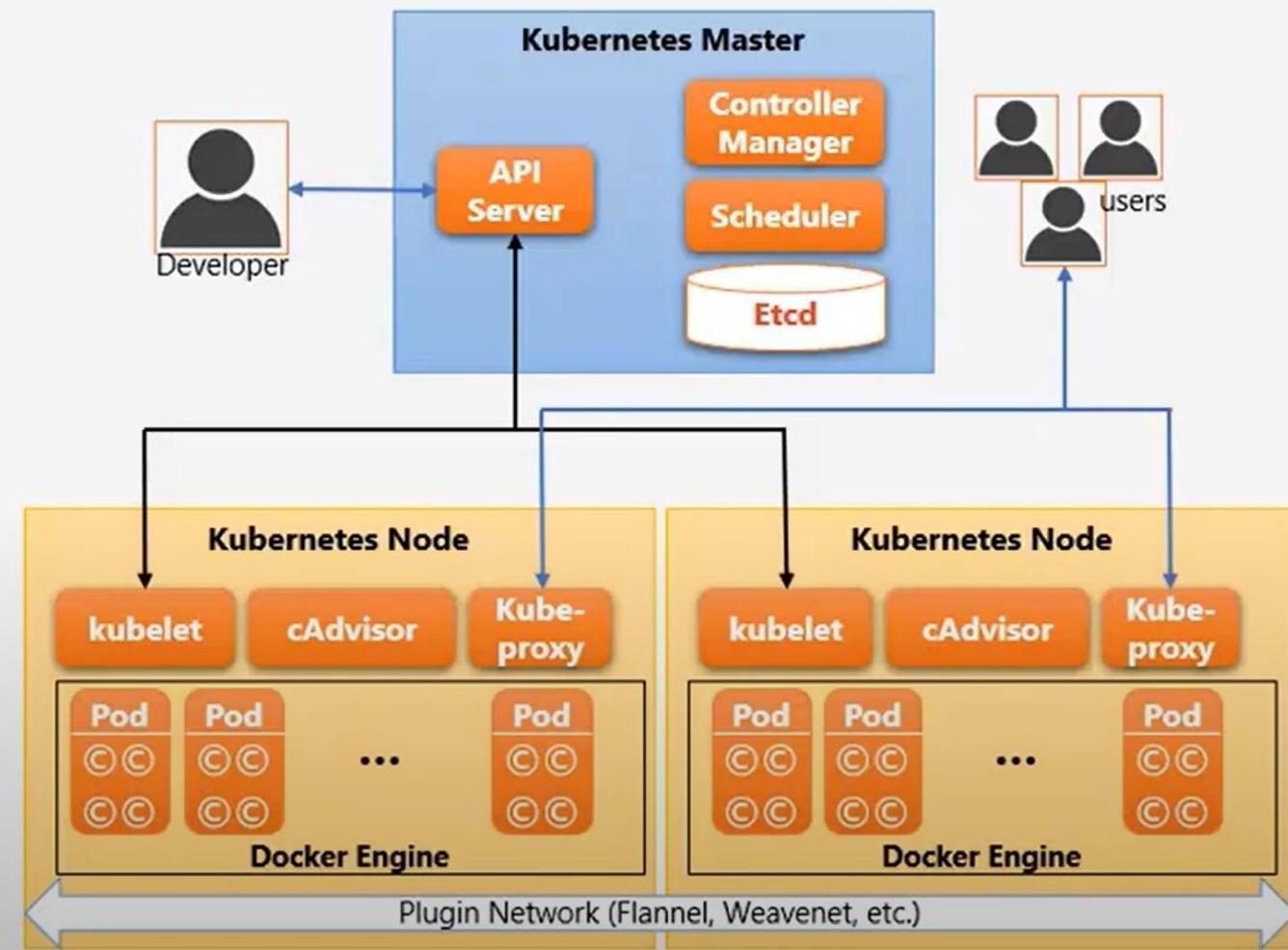
# KUBERNETES

- Outils d'orchestration des conteneurs
- Mot d'origine Grèque signifiant « Timonier », ce qui gouverne la barre du bateau.
- Inspiré du projet Borg dont google s'en sert depuis des années pour gérer ses conteneurs.
- Développé par google avec ses partenaires : Redhat, CoreOS, IBM, Microsoft, MesoSphere, Vmware, HP, etc...
- Ecrit en langage Go
- Peut tourner en :
  - Local, Cloud (GCE, Azure, AWS, ..), Machines Physiques
- Kubernetes REST API pour exposer ses fonctionnalités.
- L'objectif est de gérer directement des applications et non pas de machines



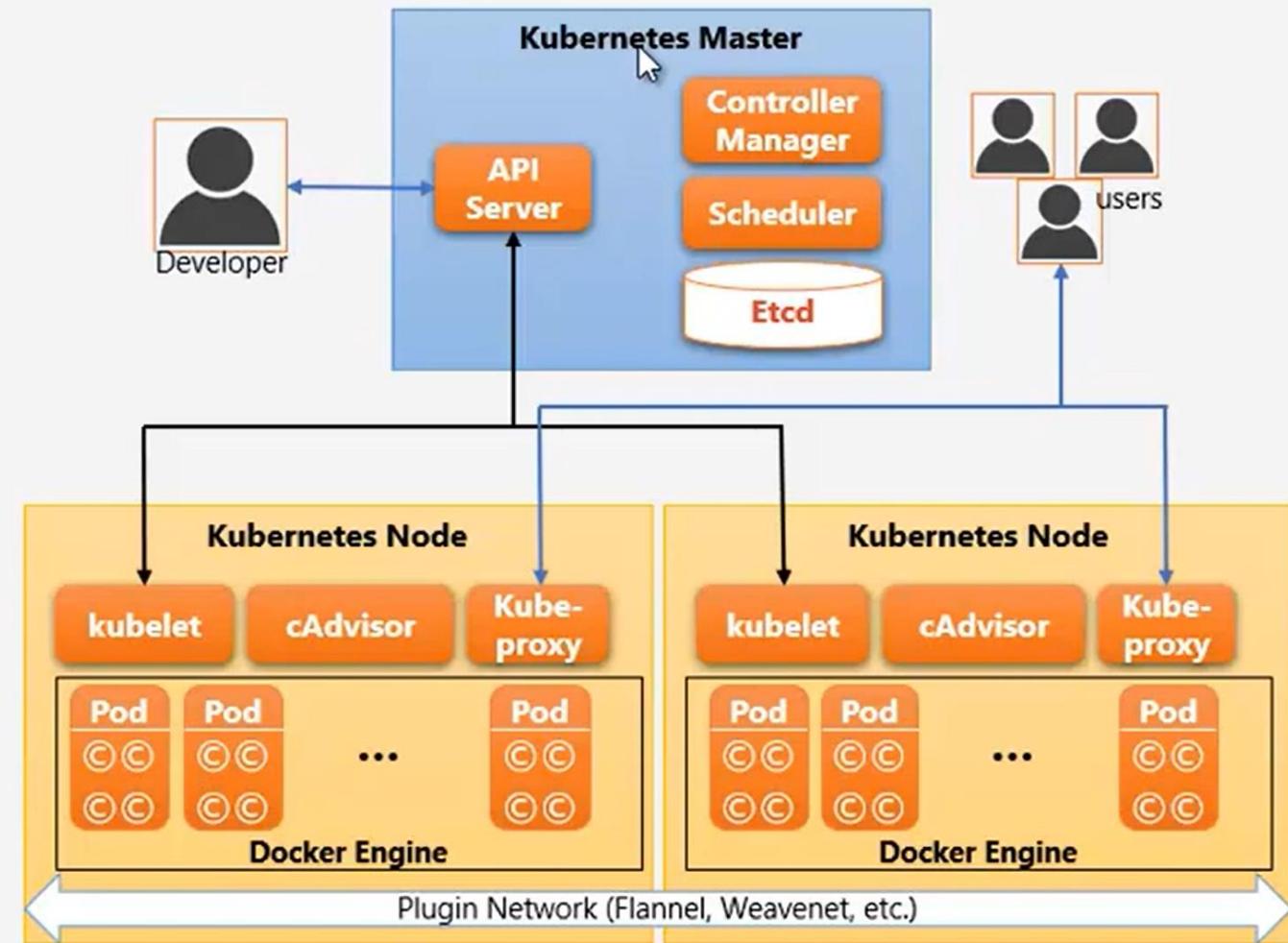
# Qu'est qu'on peut faire avec KUBERNETES

- Déployer une application sous forme de conteneurs d'une manière très rapide et prévisible
- Automatiser le déploiement et la réplication des containers.
- Organiser les conteneurs en groupes et faire du Load balancing
- Déclarer l'architecture cible (Nombre de pods, contenu, stratégie de mise à jour, etc.) et laisser le système atteindre et à maintenir cette cible
- Faire des mises à jour sans interrompre le service
- Séparer l'application de l'architecture sous-jacente.
- Déetecter les problèmes et les résoudre tout seul



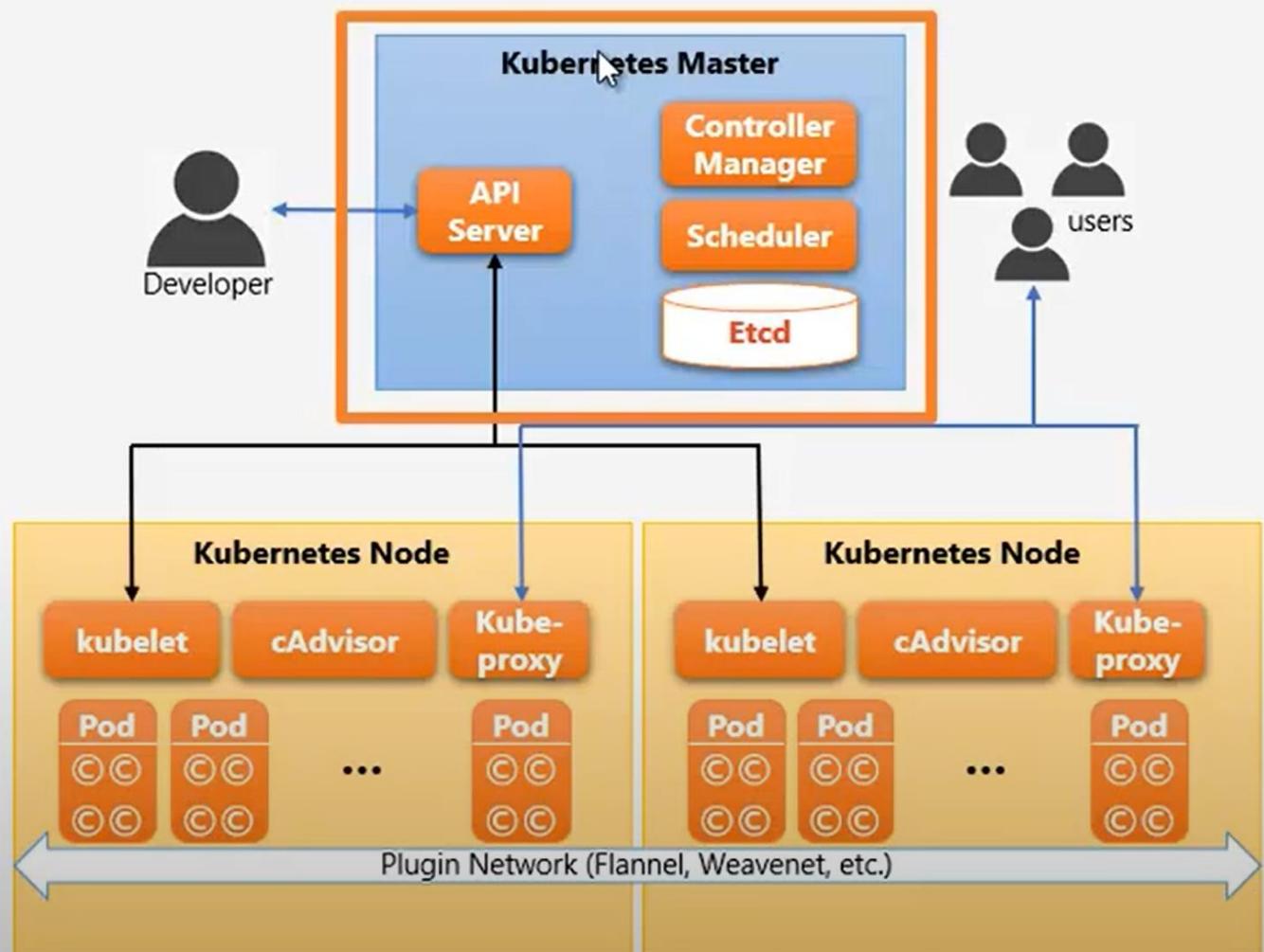
## Concepts de base

- Cluster : Ensemble de machines physique ou virtuelle utilisées par Kubernetes pour faire tourner des applications.
- Node : Une machine physique ou virtuelle dans laquelle seront déployés des Pods. Chaque Node doit exécuter un Container Engine comme Docker.
- Pod: Groupe de containers déployés ensemble
- Namespace : Segmenter Pod, Volume, Secrêts, ..
- Kubectl : Application en ligne de commande permettant d'intéragir avec Kubernetes.



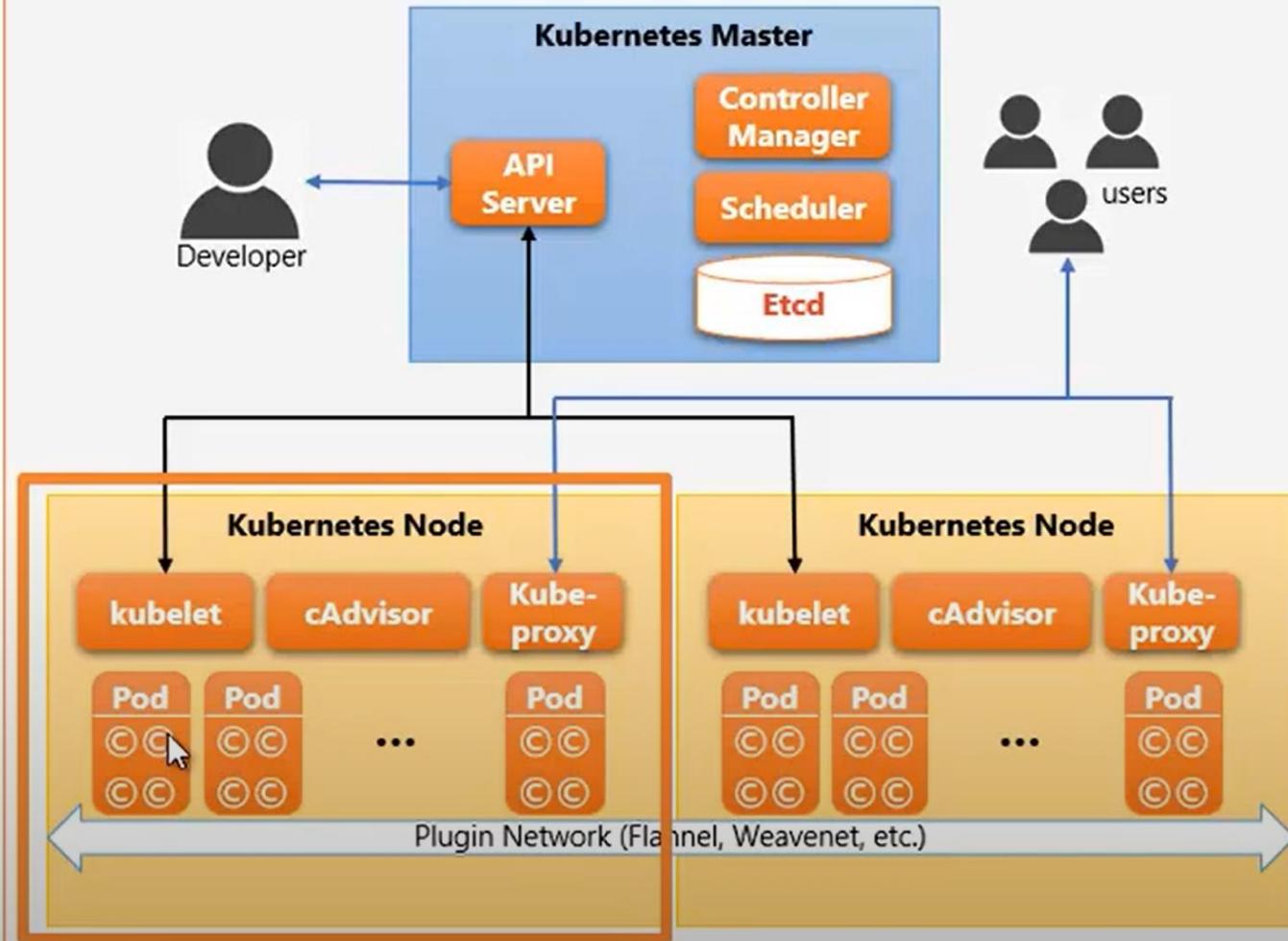
## Kubernetes Master

- Les différents composants du plan de contrôle de Kubernetes sont décrits ci-dessous:
  - Etcd : Unité de stockage Distribuée persistante et légère
  - API Server: API REST de communication avec les composants internes et externes
  - Scheduler: L'ordonnanceur qui permet de sélectionner quel Node devrait faire tourner un Pod
  - Controller manager : processus dans lequel s'exécutent les contrôleurs principaux de Kubernetes tels que DaemonSet Controller et le Replication Controller.



## Kubernetes Node

- Les composants d'un Worker Node de Kubernetes sont:
  - Kubelet : Responsable de l'état d'exécution du nœud ( c'est-à-dire, d'assurer que tous les conteneurs sur un nœud sont en bonne santé organisés en Pods sous la direction du plan de contrôle)
  - Kube-proxy : Responsable d'effectuer le routage du trafic vers le conteneur approprié en se basant sur l'adresse IP et le numéro de port de la requête entrante.
  - cAdvisor : Agent qui surveille et récupère les données de consommation des ressources et des performances comme le processeur, la mémoire, ainsi que l'utilisation disque et réseau des conteneurs du Node.



## Pod

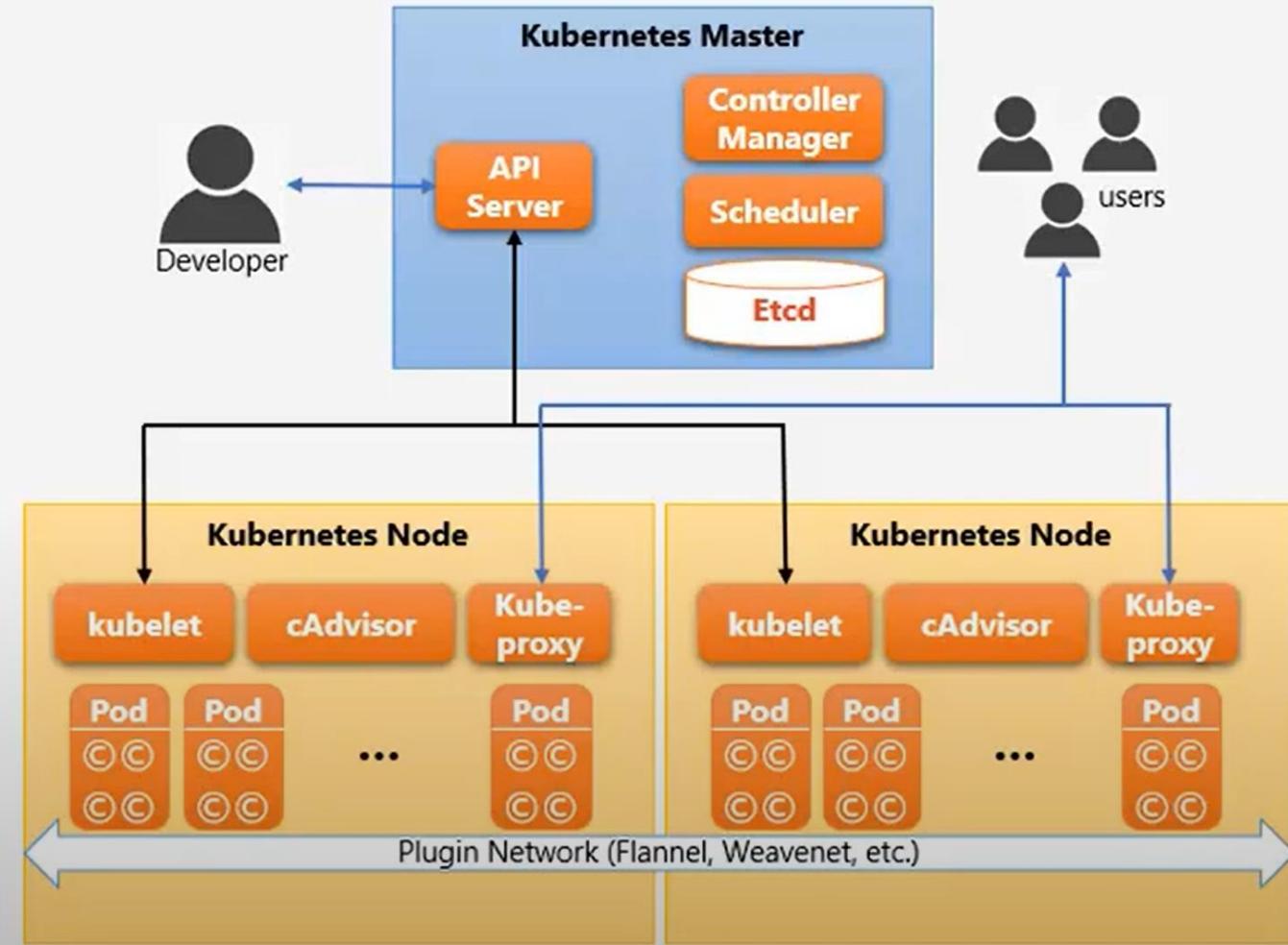
---

- Groupe de containers déployés ensemble
- Les **containers du Pod** :
  - Sont toujours démarrés, arrêtés et répliqués en groupe
  - Partagent le même Network, Namespace, IP et Ports
  - Peuvent communiquer ensemble en utilisant localhost
  - Peuvent partager des données via des shared volumes
- Le Pod est l'entité de base qu'on va répliquer et qu'on va mettre sur différentes nœuds du cluster.

```
// Créer un pod
$ kubectl create -f mypod.yml
//Lister tous les pods
$ kubectl get pods
//Supprimer un pod
$ kubectl delete Pod_ID
```

## Volume

- Espace de stockage
  - qui peut être attaché
    - à un conteneur
    - À un groupe de conteneurs dans un même Pod
  - Peut être persisté indépendamment du conteneur
  - Par exemple, il permet aux conteneurs de partager un même répertoire dans un même Pod.



## Déploiement du contrôleur

- Elément central de Kubernetes
- C'est le fichier dans lequel on décrit la cible que l'on veut atteindre.
  - Les conteneurs qu'on veut mettre dans le Pod
  - Les détails de réPLICATION
  - Les stratégies qu'on va utiliser pour les mises à jour
- C'est le déploiement du contrôleur de Kubernetes qui va se charger de maintenir cet état.
  - Par exemple , s'il y a un Pod qui plante, il va le désactiver et l'enlever du load balancer et créer un autre à côté pour démarrer sur un autre nœud pour que ça puisse continuer à fonctionner dans les mêmes conditions exigées dans le descripteur de déploiement.

myDeployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
      ports:
        - containerPort: 80
```

```
$ kubectl create -f myDeployment.yaml
```

## Déploiement du contrôleur

- Dans cet exemple :
  - Un déploiement nommé nginx est créé, indiqué par le champ metadata: name.
  - Le déploiement crée trois pods répliqués, indiqués par le champ replicas.
  - Le modèle de pod, ou le champ spec: template, indique que ses pods sont libellés app: nginx.
  - La spécification du modèle de pod, ou du champ template: spec, indique que les pods exécutent un conteneur, nginx, qui exécute l'image Docker Hub nginx dans sa version 1.7.9.
  - Le déploiement ouvre le port 80 pour une utilisation par les pods.
- En résumé, le modèle de pod contient les instructions suivantes pour les pods créés par ce déploiement :
  - Chaque pod est libellé app: nginx.
  - Créez un conteneur et nommez-le nginx.
  - Exécutez l'image nginx dans la version 1.7.9.
  - Ouvrez le port 80 pour envoyer et recevoir du trafic.

## Deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80
```

## Service

---

- Permet d'accéder à un groupe de Pods avec une même adresse IP statique. Un peut plus que le load balancer
  - On identifie les Pods par des labels (Selectors) qui seront les cibles du loadbalancer
  - Peut faire du load balancing
  - Différents Types :
    - ClusterIP
    - NodePort
    - LoadBalancer

## Installing Kubernetes with Minikube

<https://kubernetes.io/docs/tasks/tools/install-minikube/>

- **Kubectl** est un outil en ligne de commande de kubernetes, kubectl, vous permet d'exécuter des commandes dans les clusters Kubernetes. Vous pouvez utiliser kubectl pour déployer des applications, inspecter et gérer les ressources du cluster et consulter les logs, ....
- **Minikube**, qui est un outil qui fait tourner un cluster Kubernetes à un nœud unique dans une machine virtuelle sur votre machine. C'est une machine virtuelle qui avec l'essentiels des packages de:
  - Docker
  - Kubernetes

- Installer cubeclt :
  - <https://kubernetes.io/docs/tasks/tools/install-kubectl/>
- Installer miniKube :
  - <https://kubernetes.io/docs/tasks/tools/install-minikube/>



---

# Jenkins

---

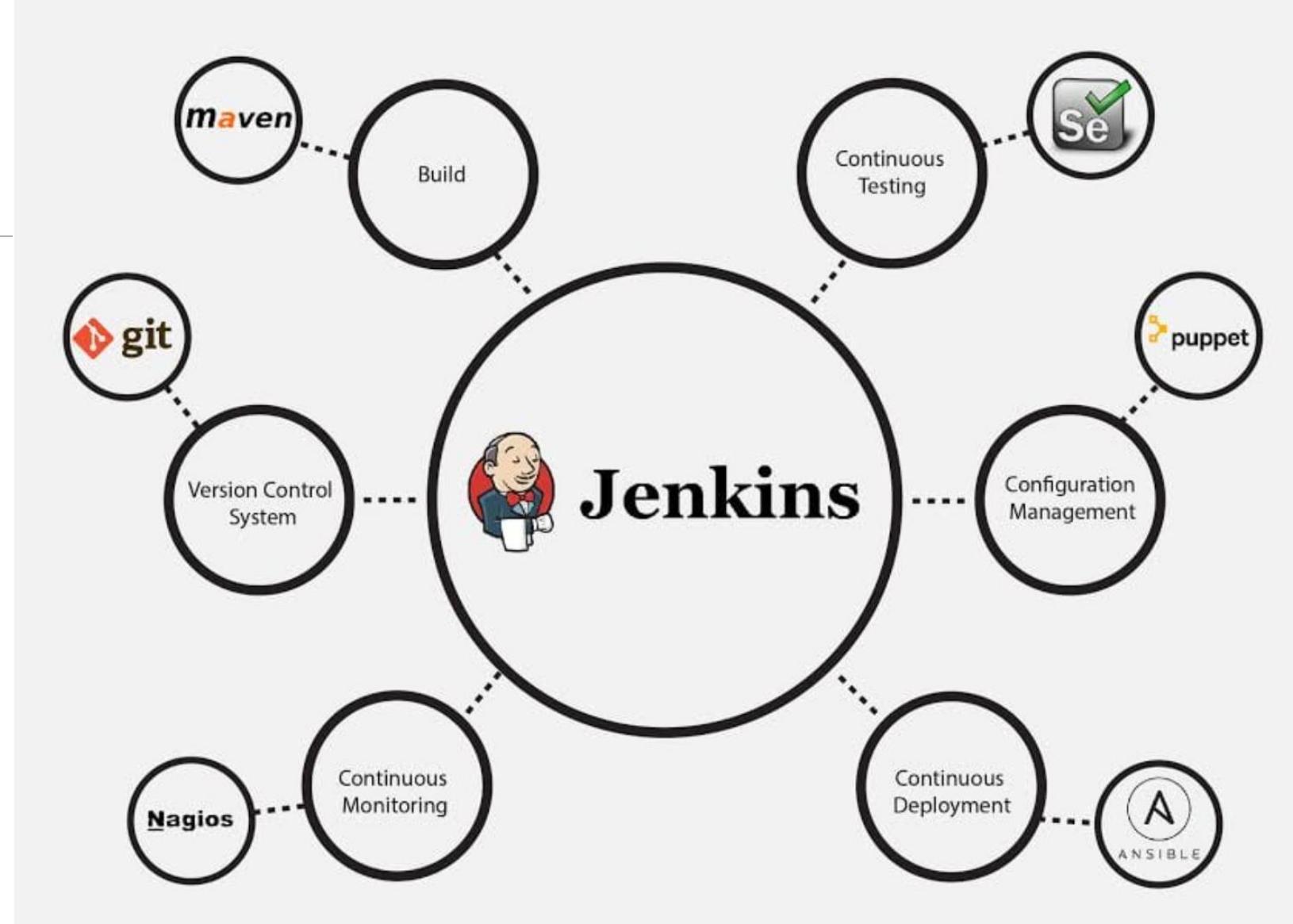
# Qu'est-ce que Jenkins

---

- ❑ Jenkins est un outil d'automatisation open source écrit en Java avec des plugins construits à des fins d'intégration continue.
- ❑ Jenkins est utilisé pour créer et tester vos projets logiciels en continu, ce qui permet aux développeurs d'intégrer plus facilement les modifications au projet et aux utilisateurs d'obtenir plus facilement une nouvelle version.
- ❑ Il vous permet également de livrer votre logiciel en continu en s'intégrant à un grand nombre de technologies de test et de déploiement.
- ❑ Jenkins réalise l'intégration continue à l'aide de plugins

# Jenkins

---



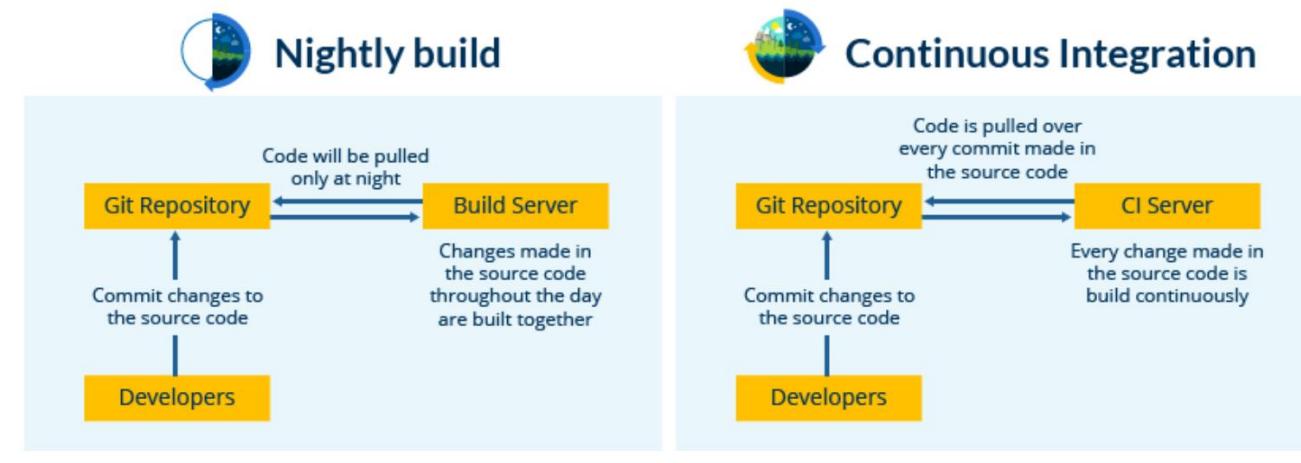
# Les avantages de Jenkins

---

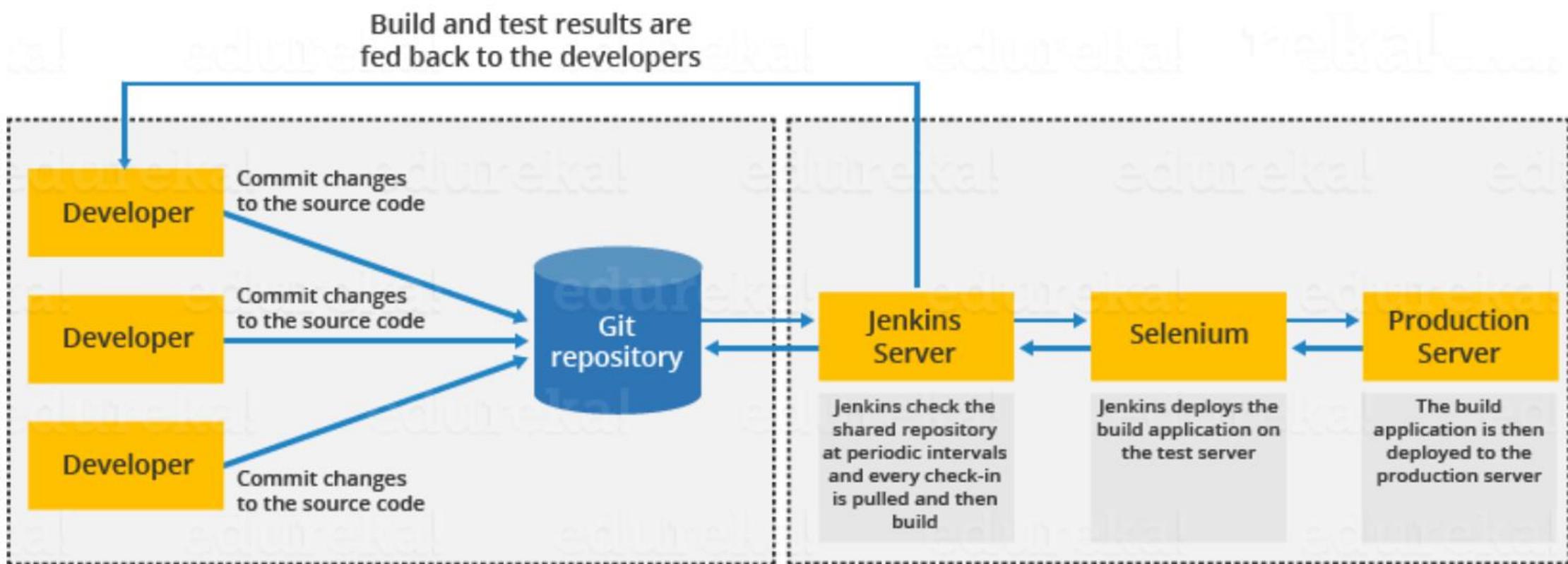
- ❑ C'est un outil open source avec un grand soutien de la communauté.
- ❑ C'est facile d'installer.
- ❑ Il dispose de plus de 1000 plugins pour faciliter votre travail. Si un plugin n'existe pas, vous pouvez le coder et le partager avec la communauté.
- ❑ C'est gratuit.
- ❑ Il est construit avec Java et est donc portable sur toutes les principales plates-formes.

# Caractéristiques de Jenkins

- ❑ **Adoption** : Jenkins est répandu, avec plus de 147 000 installations actives et plus d'un million d'utilisateurs dans le monde.
- ❑ **Plugins** : Jenkins est interconnecté avec plus de 1 000 plugins qui lui permettent de s'intégrer à la plupart des outils de développement, de test et de déploiement.
- ❑ **Continuous Integration** : L'intégration continue est une pratique de développement dans laquelle les développeurs sont tenus de valider les modifications du code source dans un référentiel partagé plusieurs fois par jour ou plus fréquemment



# Continuous Integration With Jenkins

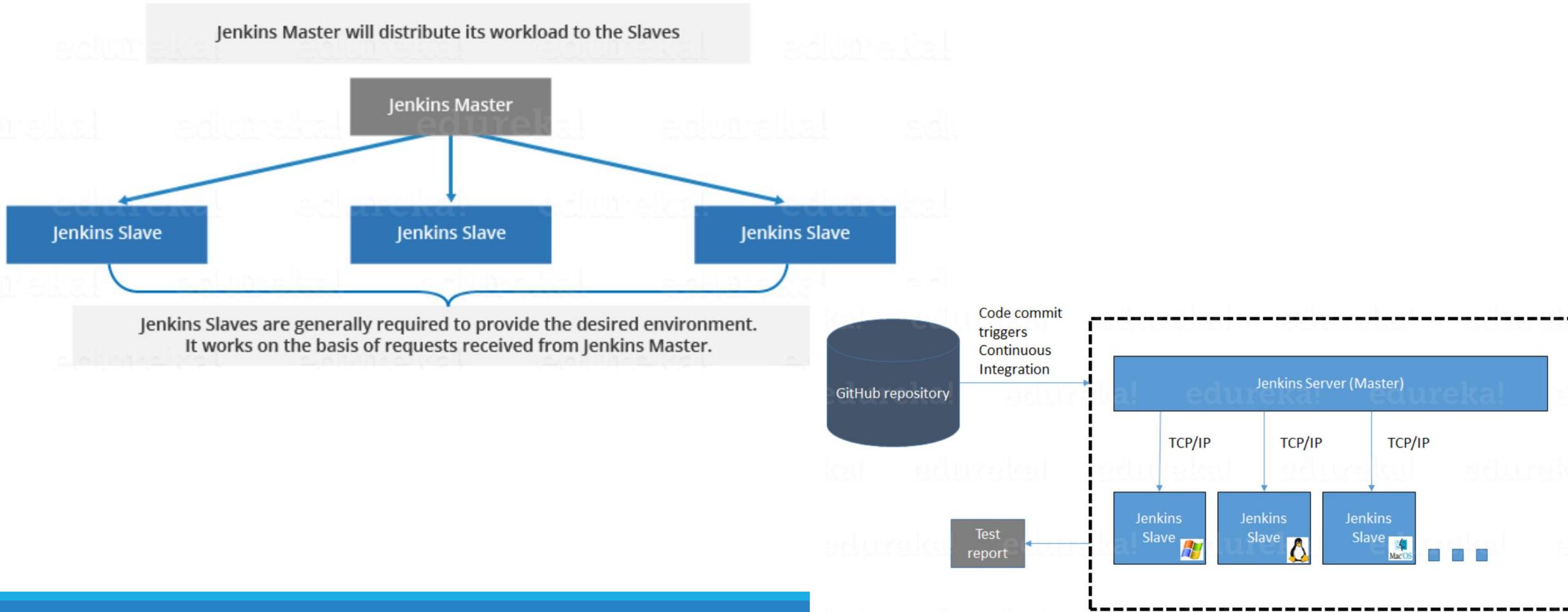


# Architecture distribuée Jenkins

---

- ❑ Jenkins utilise une architecture maître-esclave pour gérer les builds distribués. Dans cette architecture, le maître et l'esclave communiquent via le protocole TCP/IP.
- ❑ Le serveur Jenkins principal est le maître. Le travail du Master consiste à gérer :
  - ❑ Planification des travaux de construction.
  - ❑ Envoi des builds aux esclaves pour l'exécution réelle.
  - ❑ Surveillez les esclaves (éventuellement en les mettant en ligne et hors ligne selon les besoins).
  - ❑ Enregistrement et présentation des résultats de la construction.
  - ❑ Une instance maître de Jenkins peut également exécuter des tâches de build directement.
- ❑ Un esclave est un exécutable Java qui s'exécute sur une machine distante. Voici les caractéristiques des esclaves Jenkins :
  - ❑ Il entend les requêtes de l'instance Jenkins Master.
  - ❑ Les esclaves peuvent fonctionner sur une variété de systèmes d'exploitation.
  - ❑ exécute des tâches de construction envoyées par le maître.

# Architecture distribuée Jenkins

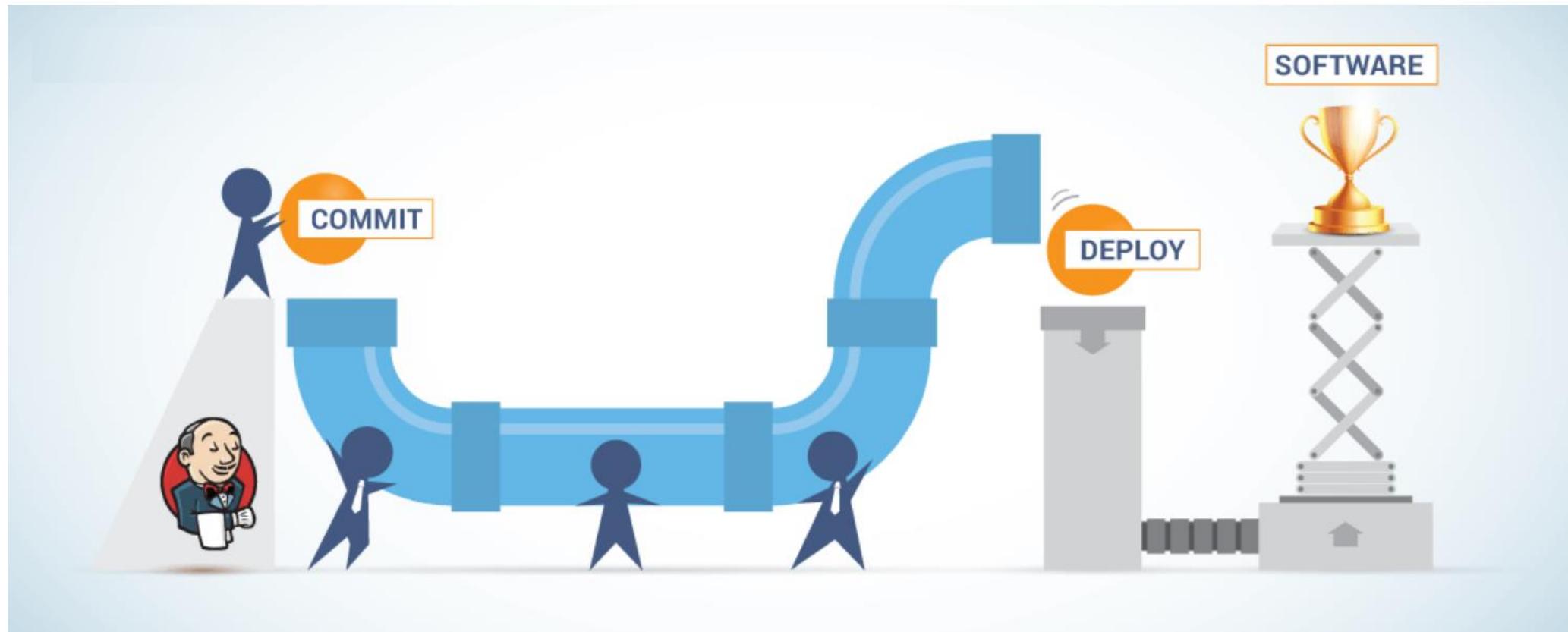


# Pipeline de construction de Jenkins



# Continuous delivery With Jenkins

---



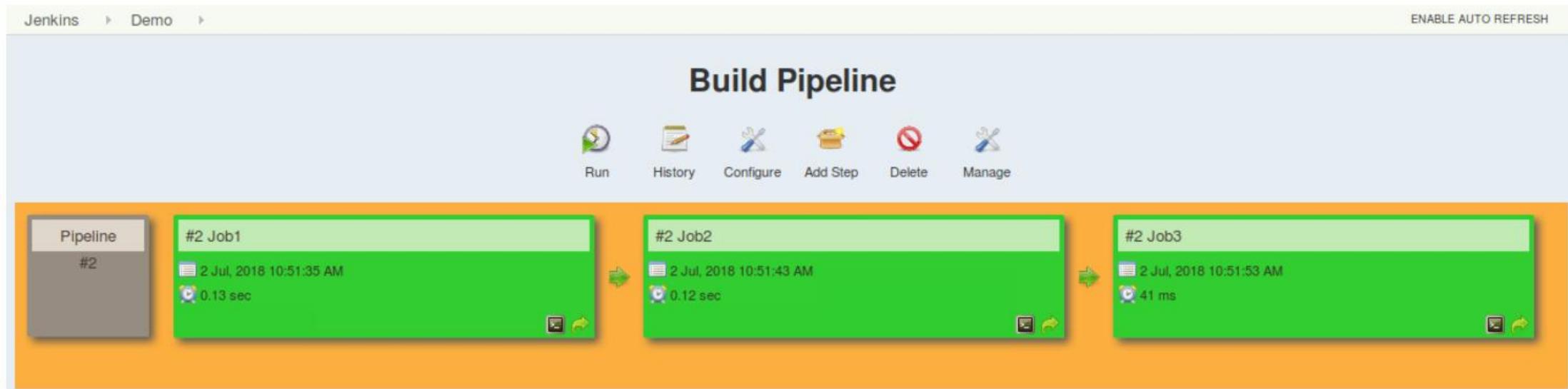
# Continuous delivery With Jenkins

---

- ❑ La livraison continue est la capacité de publier un logiciel à tout moment. C'est une pratique qui garantit que le logiciel est toujours dans un état prêt pour la production.
- ❑ La livraison continue garantit que le logiciel est construit, testé et publié plus fréquemment
- ❑ Il réduit le coût, le temps et le risque des versions logicielles incrémentielles.
- ❑ Pour effectuer une livraison continue, Jenkins a introduit une nouvelle fonctionnalité appelée pipeline Jenkins

# Jenkins pipeline

- ❑ Un pipeline est un ensemble de tâches qui met le logiciel du contrôle de version entre les mains des utilisateurs finaux à l'aide d'outils d'automatisation.
- ❑ Il s'agit d'une fonctionnalité utilisée pour incorporer la livraison continue dans notre flux de travail de développement logiciel.



# Pipeline as code

---

- ❑ La principale caractéristique de ce pipeline est de définir l'intégralité du flux de déploiement via le code.
- ❑ Tous les travaux standard définis par Jenkins sont écrits manuellement en un seul script.
- ❑ Instead of building several jobs for each phase, you can now code the entire workflow and put it in a Jenkinsfile.

# Avantages du pipeline Jenkins

---

- ❑ Il modélise des pipelines simples à complexes en tant que code en utilisant Groovy DSL (Domain Specific Language)
- ❑ Le code est stocké dans un fichier texte appelé Jenkinsfile qui peut être archivé dans un SCM (Source Code Management)
- ❑ Améliore l'interface utilisateur en incorporant les entrées de l'utilisateur dans le pipeline
- ❑ Il est durable en termes de redémarrage non planifié du maître Jenkins
- ❑ Il peut redémarrer à partir des points de contrôle enregistrés
- ❑ Il prend en charge les pipelines complexes en incorporant des boucles conditionnelles, des opérations de fourche ou de jointure et en permettant l'exécution de tâches en parallèle
- ❑ Il peut s'intégrer à plusieurs autres plugins

# Types of Jenkins pipeline

---

## ❑ Declarative pipeline syntax:

- ❑ Le pipeline déclaratif est une fonctionnalité relativement nouvelle qui prend en charge le concept de pipeline en tant que code.
- ❑ Ce code est écrit dans un fichier Jenkins qui peut être archivé dans un système de gestion de contrôle de source tel que Git.

## ❑ Scripted pipeline syntax:

- ❑ Le pipeline scripté est une manière traditionnelle d'écrire le code.
- ❑ le fichier Jenkins est écrit sur l'instance de l'interface utilisateur Jenkins

# Pipeline concepts

---

## ❑ Pipeline:

- ❑ Il s'agit d'un bloc défini par l'utilisateur qui contient tous les processus tels que la construction, le test, le déploiement,...

```
pipeline {  
    }  
}
```

## ❑ Node:

- ❑ Un nœud est une machine qui exécute un workflow complet. C'est un élément clé de la syntaxe du pipeline scripté.

```
node {  
    }  
}
```

## ❑ Agent:

- ❑ Un agent est une directive qui peut exécuter plusieurs builds avec une seule instance de Jenkins
- ❑ Répartissez la charge de travail entre différents agents et exécutez plusieurs projets au sein d'une même instance Jenkins.

## ❑ Docker:

- ❑ Ce paramètre utilise le conteneur docker comme environnement d'exécution pour le pipeline ou une étape spécifique.

```
pipeline {  
    agent {  
        docker {  
            image 'ubuntu'  
        }  
    }  
}
```

---