ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

DevOps : Développement et Opération

Prof. Samya Bouhaddour

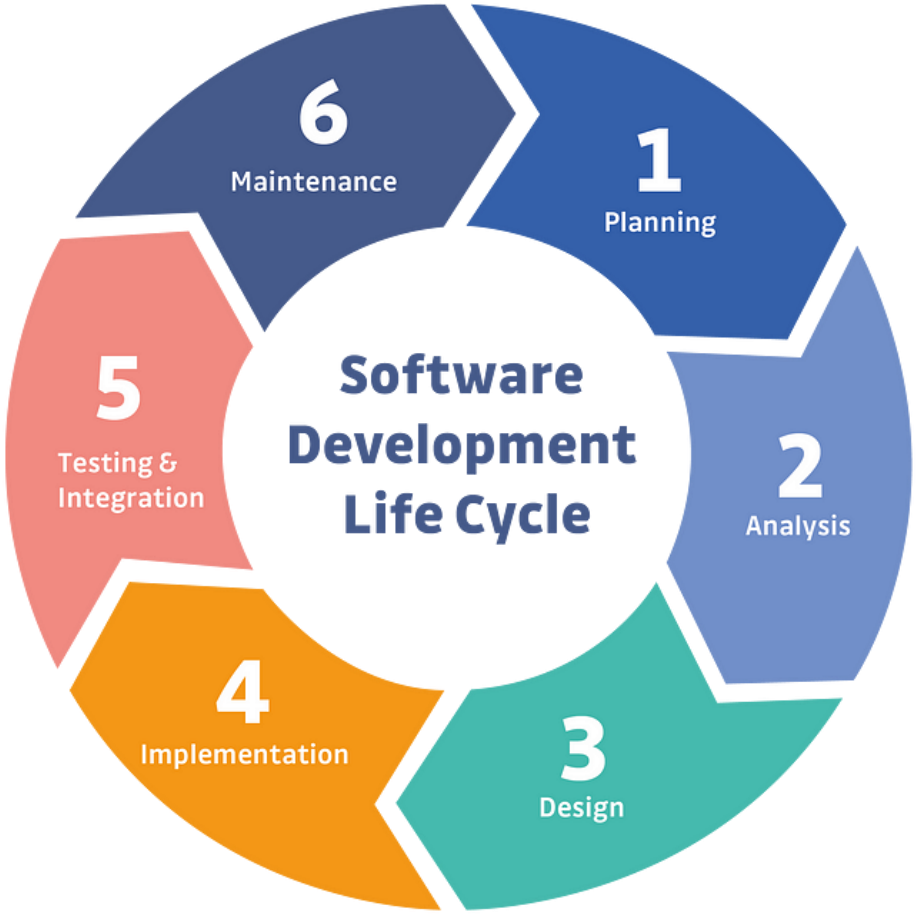
A.U. : 2023-2024

PLAN

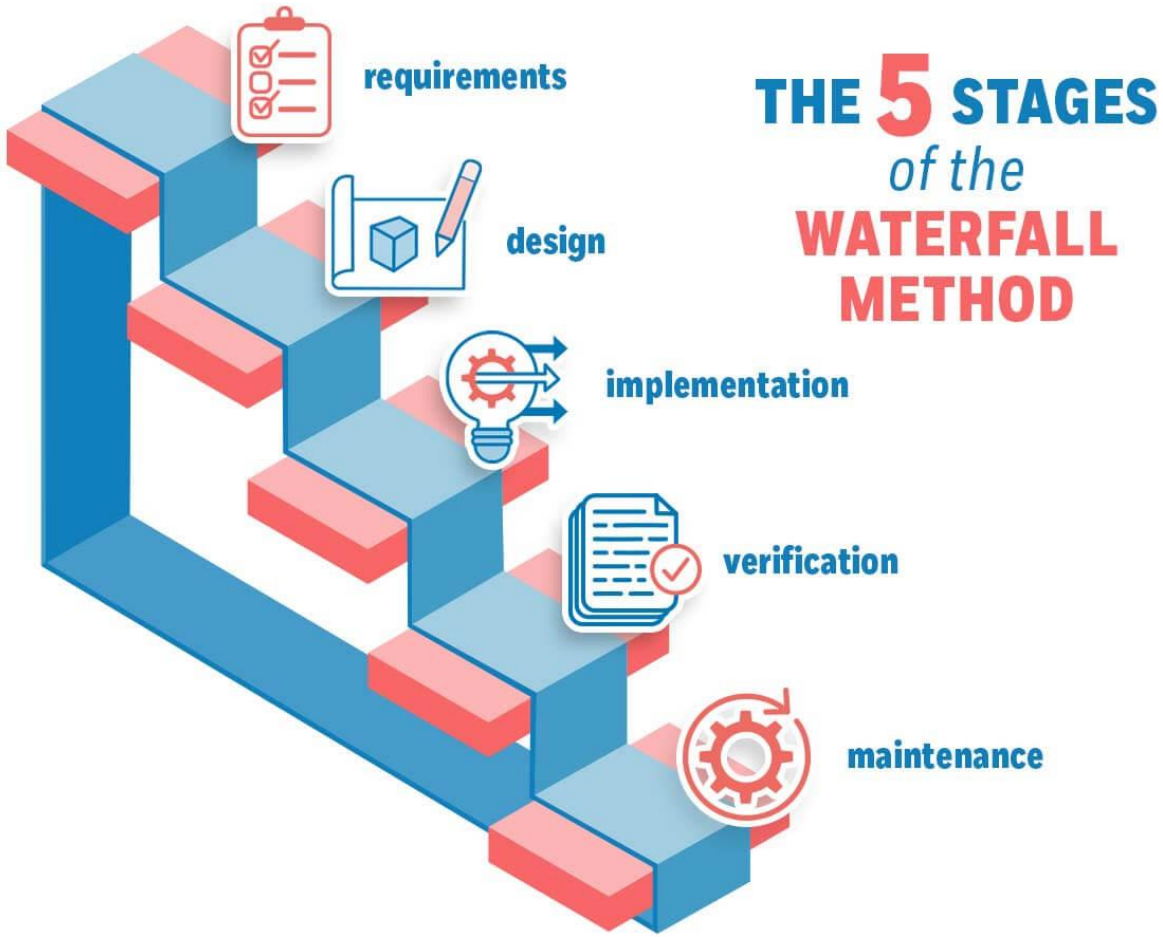
- Introduction au concept DevOps
  - Introduction à DevOps
  - Introduction à la plateforme Azure DevOps
- Contrôleur de code source GIT
  - Définition d'un contrôleur de code source
  - Application de commandes de base GIT
  - Partage d'un répertoire distanciel
  - Gestion des branches
  - Fast-Forward / No-Fast-Forward
  - Rebase / Rebase Interactif
- Planification
  - Utilisation d'Azure Board
- Outil de conteneurisation 'Docker '
  - Le concept de la virtualisation
  - Définition d'outil d'orchestration des conteneurs
  - Dockerfile
  - Docker-image
  - Conteneur Docker
  - Crétaion des images
  - Docker-Compose
  - Kubernetes
- Gestion de pipeline

Introduction à DevOps

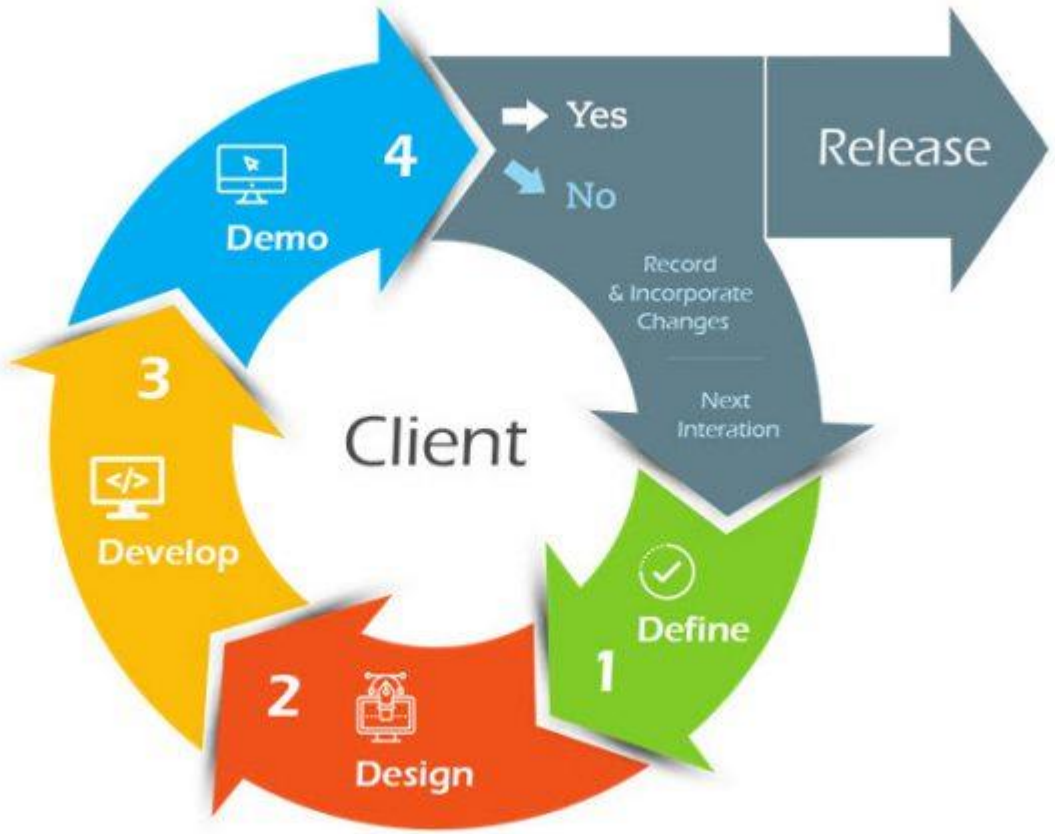
SDLC : Software Development Lice Cycle



WATERFULL : MODÈLE EN CASCADE



AGILE


4
Demo
Yes
No
Release
Record
& Incorporate
Changes
Next
Interation
3
Develop
Client
Define
1
2
Design

Le terme génie logiciel désigne l'ensemble des **méthodes***, des **techniques*** et **outils*** concourant à la production d'un logiciel, au delà de la seule activité de programmation.

Ensemble des méthodes, des techniques et outils * : Le génie logiciel ne se limite pas à une seule approche ou méthode. Il englobe un large éventail de méthodes, de techniques et d'outils utilisés pour développer un logiciel de manière efficace et de haute qualité. Cela comprend la planification, la conception, la construction, les tests, la maintenance et la gestion de projet.

LES DÉFIS

La taille des projets: pour certains, des millions de ligne de code (MLOC)

Contribution à des projets existants

- Peu de projets démarrés from scratch
- Plus de valeur d'ajouter 100 LOCs à un grand projet largement utilisé que d'écrire 10000 LOCs dans son coin. *Exemples de ces projets : systèmes d'exploitation, bases de données massives, ERP, plateformes de médias sociaux, etc.*
- La taille des projets impacte divers aspects : complexité du développement, ressources requises, planification, gestion du code et des versions, performances et maintenance.
- Gérer efficacement de grands projets exige une planification minutieuse, une coordination d'équipe efficace, ainsi que des outils et des processus appropriés.

LES DÉFIS

Réutilisation de code existant

Est ce que le concepteur d'une voiture commence par réinventer la roue?

- Pratique courante et bénéfique dans le développement logiciel.
- Évite de réinventer la roue à chaque projet.
- Recherche et utilisation de bibliothèques, frameworks et modules existants.
- Permet d'économiser du temps et des ressources.
- Favorise la cohérence, la fiabilité et la qualité du logiciel.
- Utilisation de composants éprouvés et souvent maintenus par une communauté plus large.

LES DÉFIS

Collaboration avec d’autres développeurs

Comment interagir?

- Utilisation d'outils de gestion de versions comme Git.
- Plateformes de gestion de projets telles que GitHub ou GitLab.
- Communication via des forums de discussion et des réunions virtuelles.

Comment fournir du code réutilisable?

- Assurer une documentation claire et complète du code.
- Adopter une conception modulaire et bien encapsulée.
- Publier le code sur des plateformes accessibles aux autres développeurs.
- Encourager une communication ouverte sur les normes de codage et les bonnes pratiques.

LES DÉFIS

Utilisateurs/Clients :

- Compréhension des besoins et des attentes des utilisateurs.
- Collecte de retours d'expérience et de feedbacks.
- Validation des fonctionnalités du logiciel en collaboration avec les utilisateurs.
- Développement de produits offrant une valeur réelle et une expérience utilisateur positive.

LES ENJEUX

L’industrie du logiciel, c’est 5% de projets “from scratch” et 95% de projets existants.

Le travail consiste alors à:

- ✓ **Réutiliser :** *Utiliser des composants existants pour éviter de recréer des fonctionnalités déjà développées.*
- ✓ **Faire évoluer :** *Ajouter de nouvelles fonctionnalités ou améliorer les existantes pour répondre aux besoins changeants.*
- ✓ **Etendre :** *Augmenter les capacités ou la portée d'un système pour inclure de nouveaux cas d'utilisation.*
- ✓ **Adapter :** *Modifier un logiciel pour qu'il fonctionne dans de nouveaux environnements ou avec de nouvelles technologies.*
- ✓ **Maintenir :** *Assurer le bon fonctionnement du logiciel en appliquant des correctifs et en fournissant un support continu.*
- ✓ **Réorganiser :** *Restructurer le code existant pour améliorer sa qualité et sa facilité de maintenance.*

DEVOPS

Définition simple :

Ensemble de techniques et d'outils facilitant le passage du développement à la production.

Bien plus que ça:

- Modèle de fonctionnement de l'entreprise
- Modèle d'interactions entre les équipes
- Intégration du retour sur expérience
- Une "culture"

Nous en resterons à la définition simple !

DEVOPS

Relation entre Dev et Ops

Dev: Equipes de développeurs logiciels
Ops: Equipes en charge de la mise en production des produits

Antagonisme fort

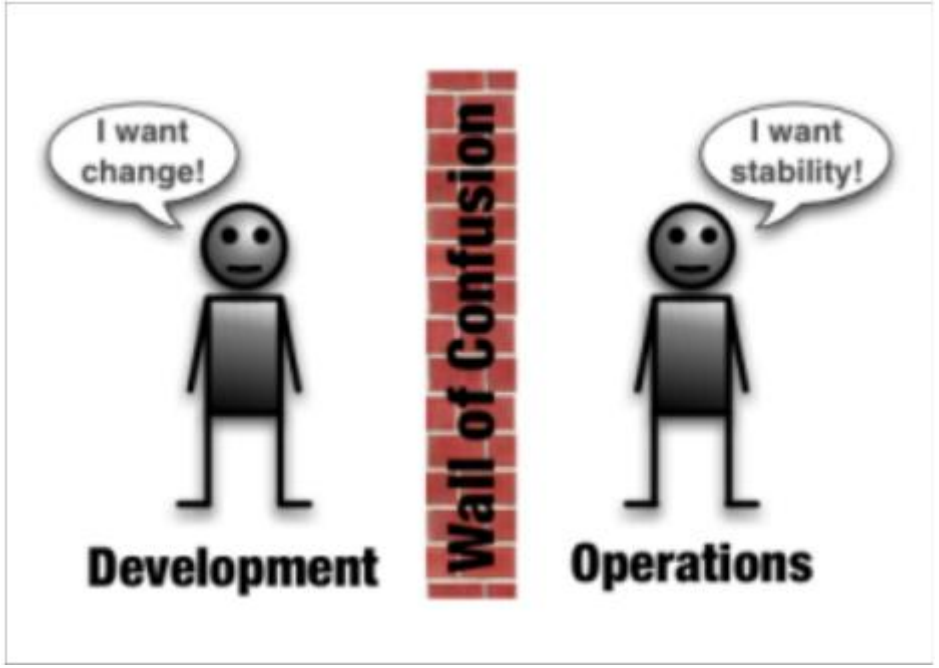
Dev: Modifications aux moindres coûts, le plus rapidement possible
Ops: Stabilité du système, qualité

L'automatisation est au cœur de l'approche DevOps !

Le DevOps est un **ensemble de pratiques** qui met l'emphase sur **l'automatisation des processus** entre les équipes de développement, et les équipes en charge du maintien en conditions opérationnelles de l'application développée.

- gain de confiance des équipes entre elles ;
- accélération des livraisons et des déploiements ;
- résolution des tickets plus rapide ;
- gestion plus efficace des tâches non planifiées…

Méthodes agiles



DevOps : Automatisation

Intégration continu

- Intégration automatique des modifications du code.
- Reconstruire et tester le logiciel à chaque modification.
- Détection rapide des problèmes.
- Amélioration de la qualité du code.
- Réduction des conflits d'intégration

Une méthode de développement logiciel dans laquelle le logiciel est reconstruit et testé à chaque modification apportée par un programmeur

Livraison continue

- Déploiement rapide et fiable des modifications validées.
- Livraison sans délai excessif ni risque.
- Réactivité accrue aux besoins des utilisateurs.
- Amélioration continue de l'application.
- Réduction des délais entre développement et mise en production

Une approche dans laquelle l'intégration continue associée à des techniques de déploiement automatiques assurent une mise en production rapide et fiable du logiciel.

Déploiement continue

- Automatisation totale du processus de mise en production.
- Mise à jour en temps réel de l'application.
- Pas de temps d'arrêt ni de risque d'erreur humaine.
- Agilité maximale dans les mises à jour.
- Capacité à répondre rapidement aux évolutions du marché

Une approche dans laquelle chaque modification apportée par un programmeur passe automatiquement toute la chaîne allant des tests à la mise en production. Il n'y a plus d'intervention humaine

L'intégration continue

L'intégration continue se réfère à plusieurs pratiques :

Construire une version fonctionnelle chaque jour :

Garantit la disponibilité d'une version testable du logiciel quotidiennement.

Assure une vision claire de l'état du prototype.

Exécuter les tests quotidiennement :

Permet de détecter rapidement les erreurs et les problèmes de compatibilité.

Réduit les risques d'introduire des défauts majeurs.

Commiter les changements quotidiennement :

Favorise une approche incrémentale du développement.

Facilite la collaboration et la gestion des versions.

Système d'observation des changements :

Automatise la vérification des modifications dans le dépôt.

Réalise des actions en réponse aux changements détectés.

La livraison continue

Les étapes principales d'une procédure de livraison continue sont:

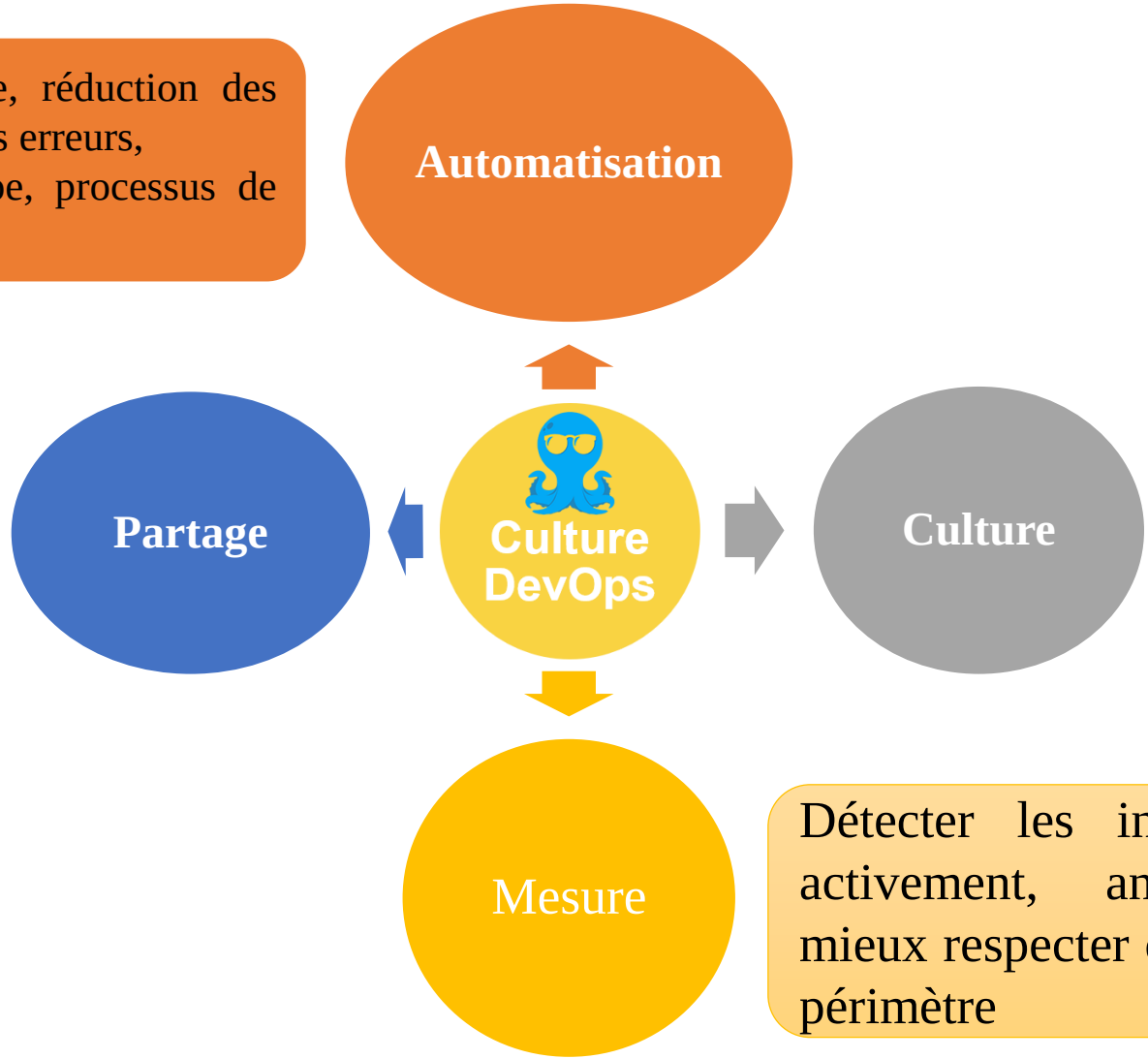
1. **Commit d'une modification**
2. **Exécution des tests unitaires (Intégration continue)**
3. **Tests de validation fonctionnelle (acceptance test)**
   - Tests black-box
   - Testent si le logiciel répond bien au besoin du client
   - Peuvent aussi être automatisés
4. **Tests de performances**
   - Testent si le logiciel peut répondre à la charge
   - Performance et passage à l'échelle
5. **Tests exploratoires**

Tests non-automatisés effectués par des experts

6. **Le déploiement en production**

Définition

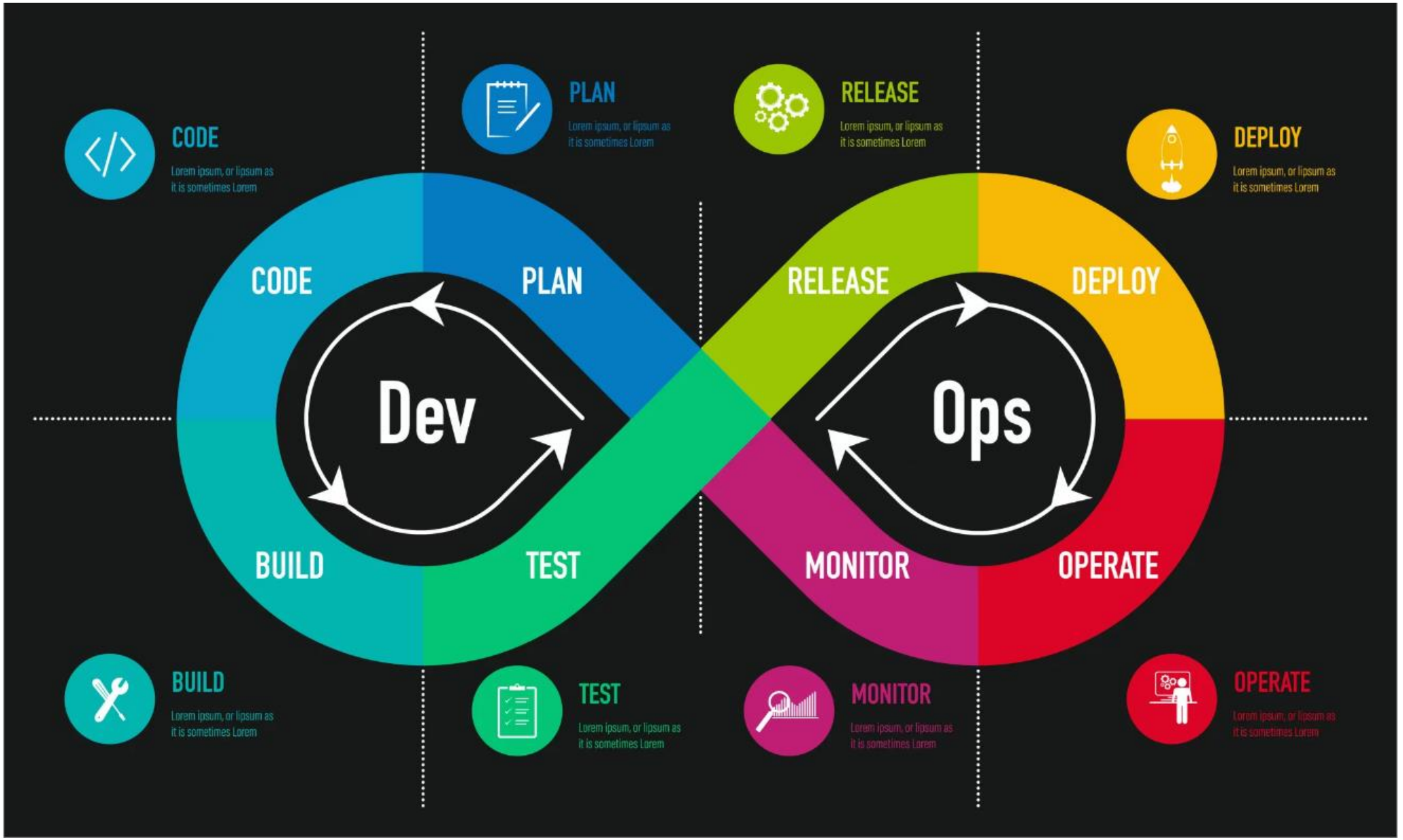
Diminution du temps de cycle, réduction des coûts de livraisons, réduction des erreurs, diminution du stress des équipe, processus de livraison fiable et rapide



Etre tous orientés vers une même cible, cible partagée entre business , développement et exploitation. Développer le «tous ensemble»

Favoriser le mode projet, l'agilité, la créativité et la mise en avant des compétences de chaque membres des équipes

Détecter les incidents pro activement, anticiper pour mieux respecter coûts, délais, périmètre


CODE
Lorem ipsum, or lipsum as it is sometimes Lorem
PLAN
Lorem ipsum, or lipsum as it is sometimes Lorem
RELEASE
Lorem ipsum, or lipsum as it is sometimes Lorem
DEPLOY
Lorem ipsum, or lipsum as it is sometimes Lorem
CODE
PLAN
RELEASE
DEPLOY
Dev
Ops
BUILD
TEST
MONITOR
OPERATE
BUILD
Lorem ipsum, or lipsum as it is sometimes Lorem
TEST
Lorem ipsum, or lipsum as it is sometimes Lorem
MONITOR
Lorem ipsum, or lipsum as it is sometimes Lorem
OPERATE
Lorem ipsum, or lipsum as it is sometimes Lorem

Avantages à adopter l’approche DevOps

- Accélération et amélioration de la fourniture des produits
- Résolution plus rapide des problèmes et complexité réduite
- Stabilité accrue des environnements d'exploitation
- Meilleure utilisation des ressources
- Automatisation accrue
- Meilleure visibilité sur les résultats du système
- Innovation renforcée

Introduction à AZURE DevOps

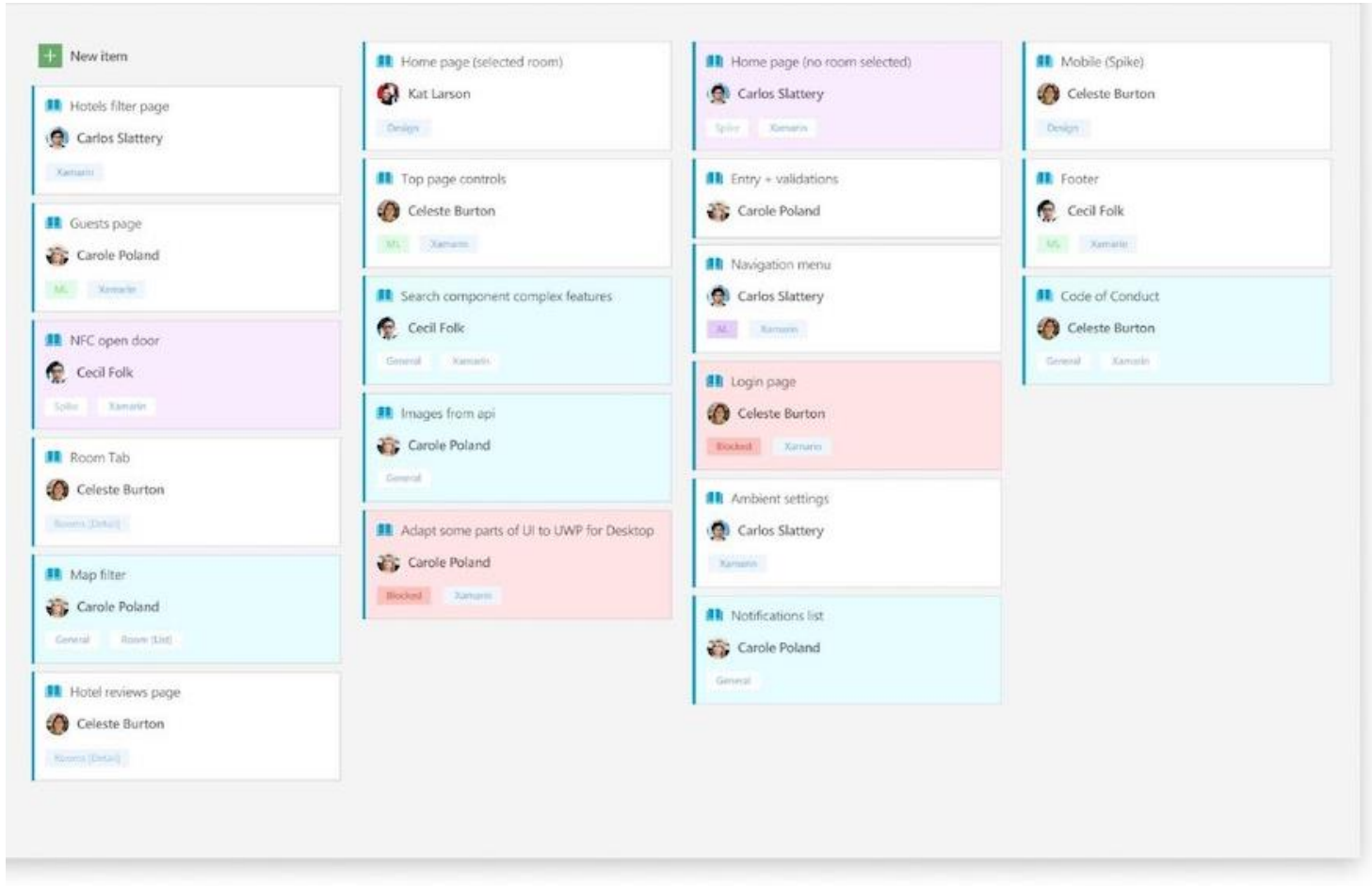
Définition

- Azure DevOps fournit des services de développement pour aider les équipes à planifier le travail, à collaborer au développement de code et à créer et déployer des applications.
- Azure DevOps fournit des fonctionnalités intégrées auxquelles vous pouvez accéder via votre navigateur Web (azure.microsoft.com) ou votre client IDE (VS).

- **Azure Repos :** Pour le partage du code source.
- **Azure Boards:** Outils de la planification.
- **Azure Pipelines :** Workflow de build et déploiement.
- **Azure Test Plans :** Automatisation et exécution des tests.
- **Azure Artifacts:** Création de package pour déploiement.

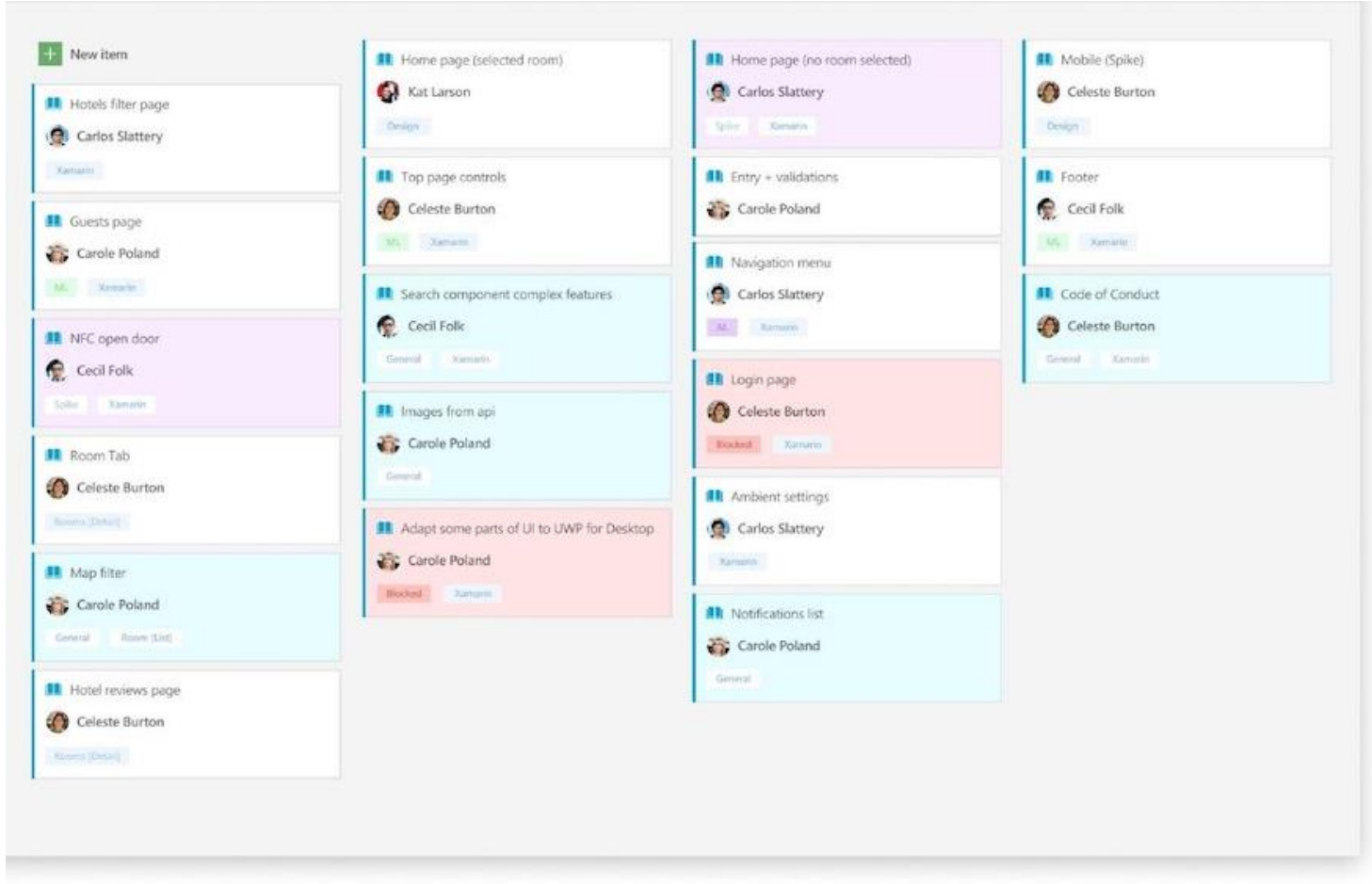
Azure Boards

Suivez le travail effectué avec des tableaux Kanban configurables, des backlogs interactifs et des outils de planification puissants.
Grâce à une traçabilité et à des rapports inégalés, Boards est la solution idéale pour toutes vos idées, petites ou grandes.



Azure Repos

Suivez le travail effectué avec des tableaux Kanban configurables, des backlogs interactifs et des outils de planification puissants.
Grâce à une traçabilité et à des rapports inégalés, Boards est la solution idéale pour toutes vos idées, petites ou grandes.



Contrôleur de code source

LES DÉFIS

Comment gérez-vous actuellement un projet ?

- L'envoyer à travers un message sur Facebook, ... (Très mauvaise idée)
- L'envoyer par mail (Un peu moins)
- Utiliser une Dropbox, Google Drive, ... (Déjà mieux mais toujours risqué ou manque de fonctionnalités)

SOLUTION

UTILISER UN SYSTÈME DE GESTION DE VERSION DÉCENTRALISÉ


geek & poke
YESTERDAY IT WORKED
YOU KNOW YOU'RE IN A SOFTWARE PROJECT

DÉFINITION

- Source contrôle est la pratique du suivi et de la gestion des modifications du code
- Les sources contrôles fournissent un historique de développement du code et aident à résoudre les conflits lors de la fusion de contributions provenant de plusieurs sources.
- Le contrôle des sources protège le code source des dégradations occasionnelles suite aux erreurs humaines.
- Les avantages incluent: la réutilisabilité, la traçabilité, portabilité, l'efficacité, la collaboration et l'apprentissage.

CE QUE PERMET DE FAIRE UN SYSTÈME DE CONTRÔLE DU CODE SOURCE

- Conserver tout le code source
- Prendre note de tous les changements effectués au code source et à sa documentation

  => permet de retourner à une version antérieure (qui, elle, fonctionnait ! !)
- Identifier quels fichiers ont été modifiés
- Déterminer qui a modifié un bout de code
- Comparer des versions
- Fusionner des versions développées de façon concurrente
- Identifier et gérer les releases*, les versions**, les branches de développement ***

Release* : des versions stables et officielles du logiciel, généralement destinées à être utilisées par les utilisateurs finaux

Versions** : Les versions correspondent à différentes itérations du logiciel, avec des fonctionnalités ajoutées, des bogues corrigés, etc.

Branches de développement ***: sont des versions du code source qui évoluent séparément les unes des autres.

Par exemple, il peut y avoir une branche de développement pour travailler sur de nouvelles fonctionnalités tandis qu'une autre branche est dédiée à la correction de bugs. Un système de contrôle de code source permet de gérer ces différentes entités de manière organisée, en les identifiant clairement et en facilitant leur gestion.

LES DIFFÉRENTS SCCS : SCCS CENTRALISÉ VS. SCCS DISTRIBUÉ

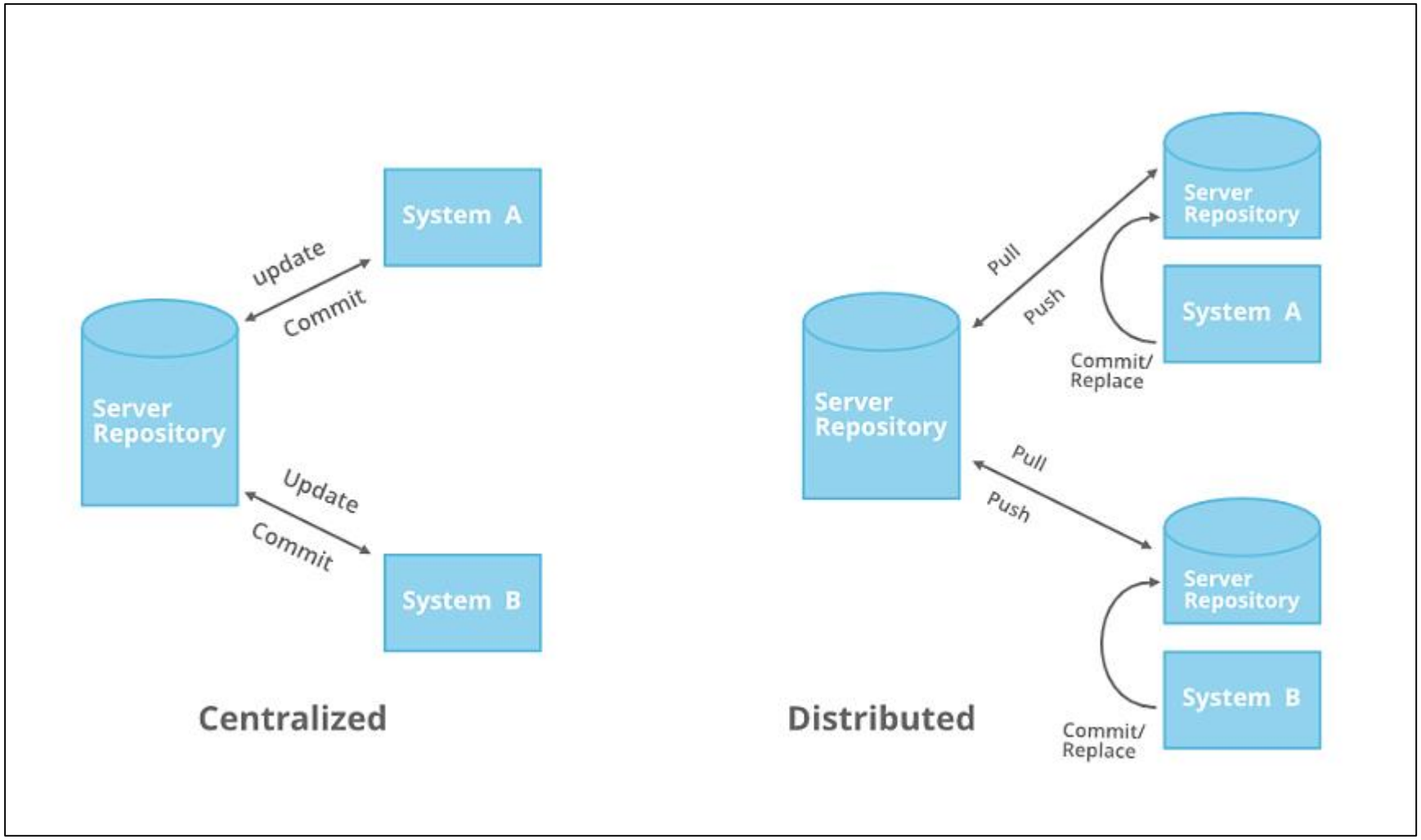
SCCS Centralisé

Tout l'historique des changements est conservé sur un serveur central (distant), duquel n'importe qui peut obtenir la version la plus récente ou envoyer les changements les plus récents.

SCCS distribué

Chaque usager a une copie locale de tout l'historique des changements. Il n'est donc pas nécessaire d'être connecté à un réseau/serveur pour sauvegarder des changements. En outre, n'importe quel usager peut se synchroniser avec n'importe quel autre.

Les différents SCCS : SCCS Centralisé vs. SCCS Distribué



LES AVANTAGES ET LES INCONVÉNIENTS DU SCCS DISTRIBUÉ

AVANATGES

- Chaque développeur a son espace privé — son sandbox
- On peut travailler — et faire des commits ! — hors ligne
- Plusieurs opérations sont très rapides — exécution locale
- La création et fusion de branches est efficace

INCONVÉNIENTS

- Quand même préférable d'avoir un dépôt central (sauvegarde)
- Pas nécessairement «une» version «la plus récente»

DÉMARRER AVEC GIT

- Git est un système de contrôle de version distribué gratuit qui permet aux programmeurs de suivre les modifications du code, via des "instantanés" (commits), dans leur état actuel.
- L'utilisation de validations permet aux programmeurs de tester, déboguer et créer de nouvelles fonctionnalités en collaboration. Tous les commits sont conservés dans ce que l'on appelle un « référentiel Git » pouvant être hébergé sur votre ordinateur, des serveurs privés ou des sites Web open source, tels que Github.
- Git permet également aux utilisateurs de créer de nouvelles "branches" du code, ce qui permet aux différentes versions du code de cohabiter.

LES CONCEPTS DU GIT

- Espace de travail : C'est l'endroit où vous travaillez sur vos fichiers. Il s'agit simplement des fichiers et répertoires que vous voyez et manipulez sur votre ordinateur. Ils n'ont rien de spécial par rapport à d'autres dossiers sur votre ordinateur.
- Dépôt local: Il s'agit de votre espace de travail associé à l'historique des modifications. C'est là que Git enregistre toutes les modifications que vous avez apportées au fil du temps.
- Commit : Un commit représente une version spécifique de votre projet à un moment donné. Chaque commit est comme une "capture d'écran" de l'état de vos fichiers à un moment précis. Il contient les modifications que vous avez apportées ainsi qu'un message descriptif.
- Historique : C'est la "chaîne" de tous les commits, du plus ancien au plus récent. Cet historique vous permet de voir l'évolution de votre projet au fil du temps.
- Dépôt distant : C'est un dépôt Git qui se trouve sur un serveur distant, comme GitHub. Il est souvent utilisé pour collaborer avec d'autres personnes sur un projet ou pour sauvegarder votre code.

LES ZONES DU GIT

Git a 3 "zones" différentes pour votre code:

- Répertoire de travail : zone dans laquelle vous allez travailler (création, modification, suppression et organisation des fichiers)
- Zone de transit : la zone dans laquelle vous listerez les modifications apportées au répertoire de travail
- Référentiel : où Git stocke en permanence les modifications que vous avez apportées sous différentes versions du projet

CONTRÔLE WORKFLOW

Le contrôle de version a un flux de travail général que la plupart des développeurs utilisent pour écrire du code et le partager avec l'équipe

- Obtenez une copie locale du code s'ils n'en ont pas encore.
- Apportez des modifications au code pour corriger les bogues ou ajouter de nouvelles fonctionnalités.
- Une fois le code prêt, rendez-le disponible pour examen par votre équipe (Pull Request).
- Une fois le code révisé, fusionnez-le dans la base de code partagée de l'équipe.

ACTIONS AVEC GIT

- **Créer** un dépôt sur GitHub.
- **Cloner** (faire une copie d') un dépôt de GitHub sur son PC: Git clone URL_Repo
- **Ajouter un fichier modifié :** il sera pris en compte dans le prochain commit: Git add nom_fichier / . / -a / *
- **Faire un commit :** créer une nouvelle version, qui contient tous les fichiers ajoutés. On y ajoute un commentaire (qui décrit les changements): Git commit –m « message du commit »

ACTIONS AVEC GIT

- Consulter un historique: Git log
- Push : envoyer ses nouveaux commits sur GitHub: Git push
- Pull : récupérer des changements (qui ont été envoyés par quelqu'un d'autre) depuis GitHub: Git pull
- Branches :
  - Créer une branche: Git branch nom_branche
  - Basculer entre les branches: Git checkout nom_branche
  - Lister les branches: Git branch –a
  - Supprimer une branche: Git branch –d nom_branch
- Merge : quand on Pull et qu'on a aussi des nouveaux commits sur son PC. Git essaye de fusionner automatiquement ; s'il ne sait pas le faire, il demande à l'utilisateur: Git merge

Bonnes pratiques pour le contrôle des sources

- Faites de petits changements, avec des commit régulières.
- Ne pas comité des fichiers personnels.
- Mettre à jour souvent et juste avant de pousser pour éviter les conflits de fusion.
- Vérifiez votre changement de code avant de le pousser dans un référentiel, assurez-vous qu'il compile et que les tests réussissent.
- Portez une attention particulière à la validation des messages car ceux-ci vous indiqueront pourquoi un changement a été effectué.
- Lier les modifications de code aux éléments de travail (WorkItem).

Gestion des Branches

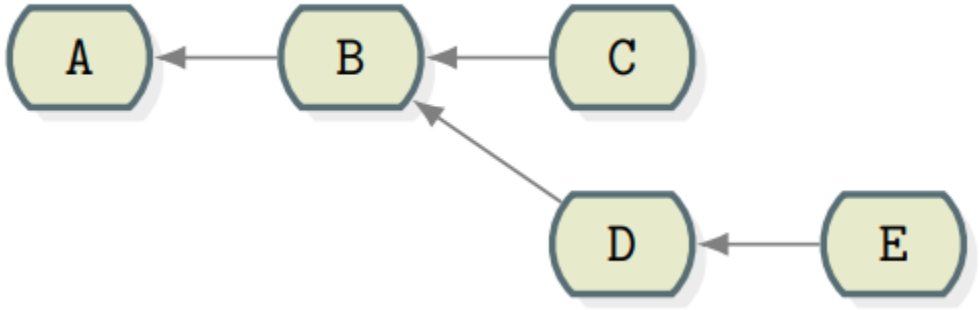
Motivations

Une équipe de développeurs participe à la réalisation d'une application:

- Comment conserver un historique?
- Comment revenir en arrière?
- Comment travailler à plusieurs en parallèle sur le même code?
- Comment gérer plusieurs versions du code à la fois?
- Comment savoir ce qui a modifié et par qui (et pourquoi)?

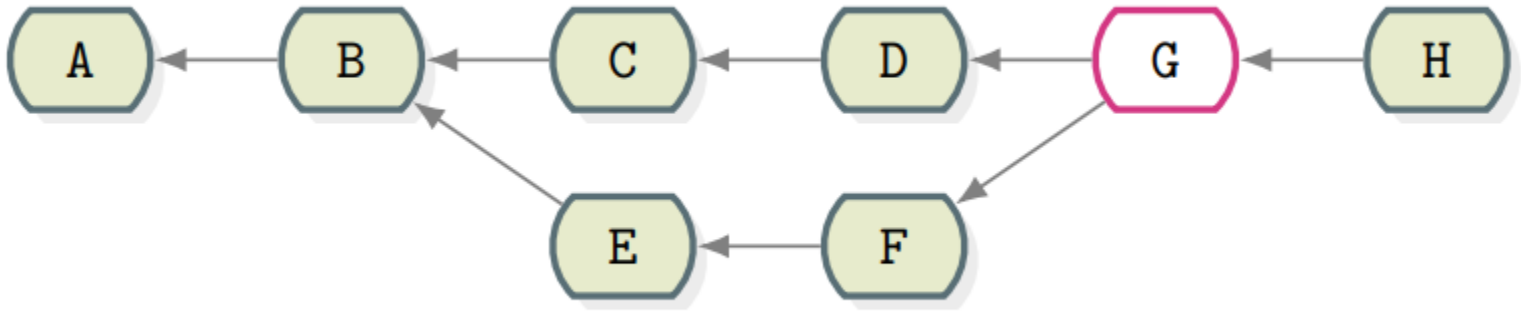
La notion d’historique

• L’historique est un graphe orienté composé d’un ensemble de versions pouvant être recalculées à partir des versions adjacentes en appliquant les patchs.
• L’historique peut inclure plusieurs branches, c’est-`a-dire des sous-graphes qui évoluent en parallèle.



Historique: les merges

On appelle merge toute version ayant un degré sortant strictement supérieur à 1.
Cette version correspond alors à la fusion des patchs de plusieurs branches.



Gestion des accès concurrents

Gestion pessimiste

- Un seul contributeur à accès en écriture à un fichier
- Pas de conflits
- Pas pratique

Gestion optimiste

- Chaque développeur peut modifier sa copie locale en parallèle
- Risques de conflits

► Modifications concurrentes de la même zone de texte

Les objets dans Git

- *Blobs*
- *Tree*
- *Commit*
- *Tag*

Blob « Binary Large Object »

On appelle Blob, l'élément de base qui permet de stocker le contenu d'un fichier.
C'est simplement le contenu d'un fichier, stocké de manière compressée dans la base de données Git. Les blobs sont identifiés par des clés SHA-1 qui sont calculées à partir de leur contenu.

- Chaque Blob est identifié de manière unique par sa clé
- A chaque révision du fichier correspond un nouveau Blob
  - Le blob stocke le contenu entier du fichier

Le Blob ne dépend pas du nom ou de l'emplacement

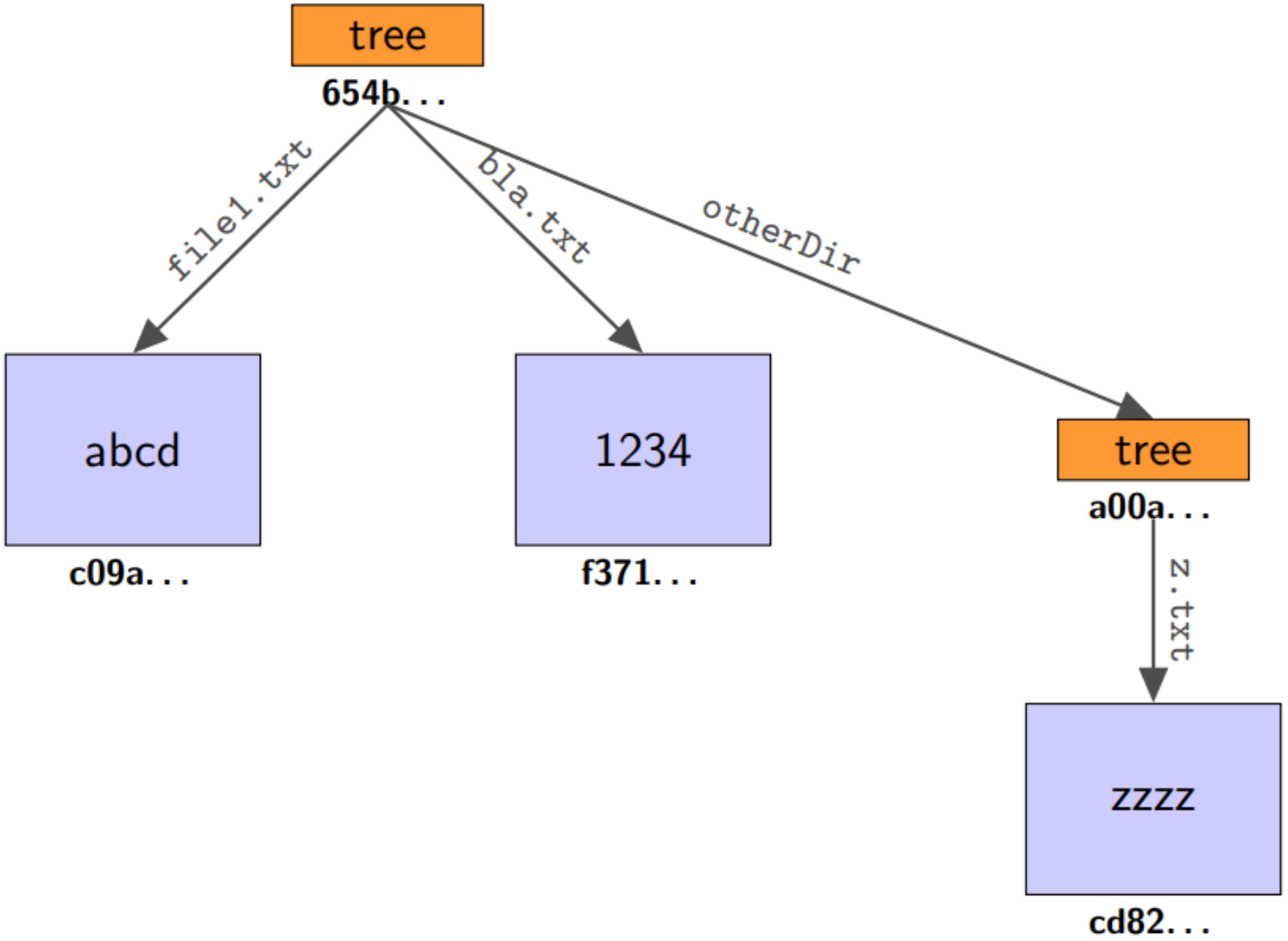
  - Si un fichier est renommé, pas de nouveau Blob
  - Si un fichier est déplacé, pas de nouveau Blob

- Le contenu du Blob est compressé avec zlib. Il contient:
  - Le type d'objet (blob)
  - La taille du fichier initial
  - Le contenu du fichier

Tree

- Un arbre représente un répertoire (ou dossier) dans un projet Git.
- Chaque arbre contient des entrées qui pointent vers des blobs correspondant aux fichiers présents dans ce répertoire, ainsi que vers d'autres sous-arbres pour les sous-répertoires.
- L'entrée d'un arbre contient des métadonnées telles que le nom du fichier ou du sous-répertoire, le mode (permissions), et l'identifiant SHA-1 du blob ou de l'arbre auquel elle pointe.

Tree et Blob: Exemple

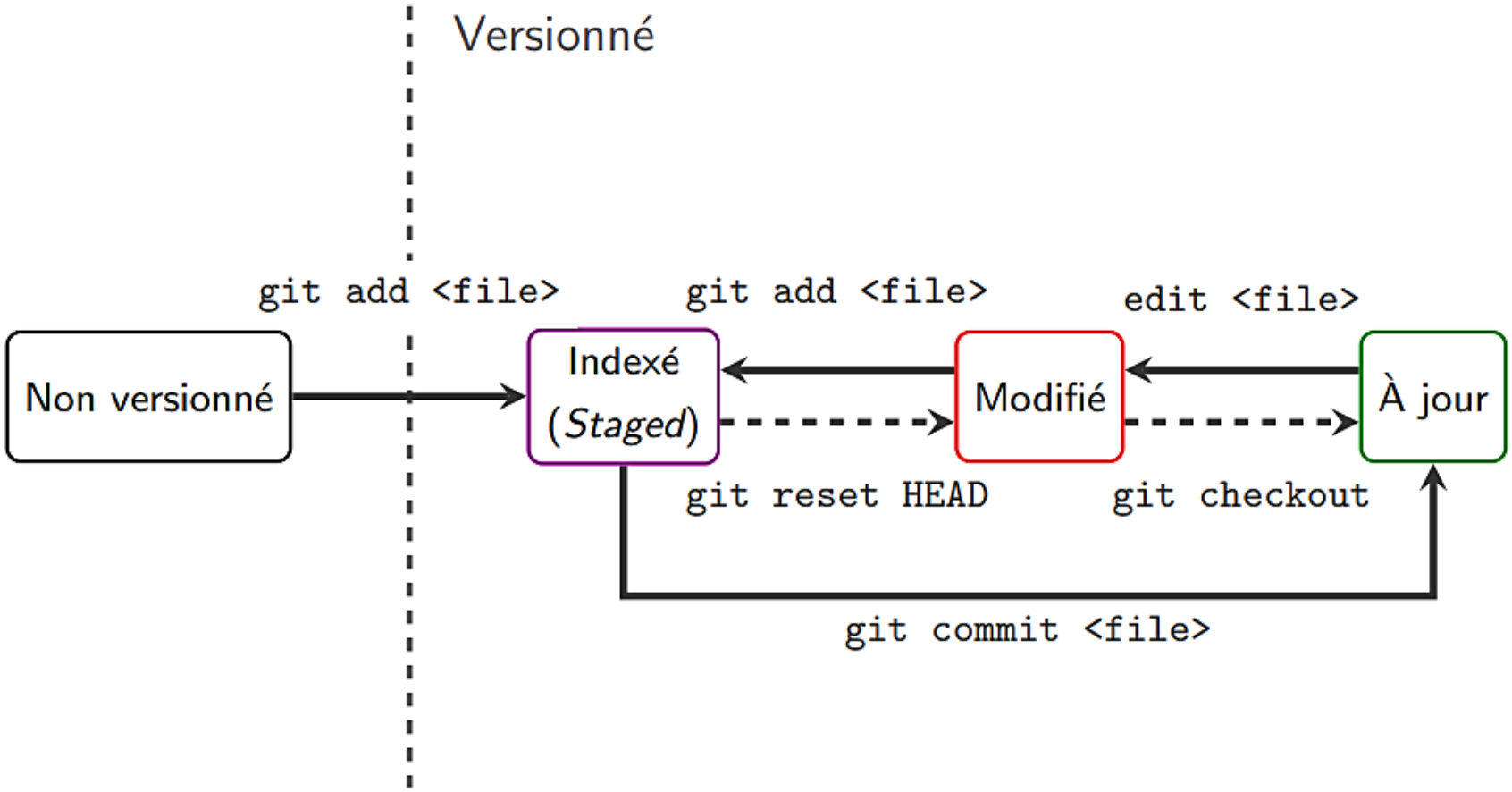


Commit

Un Commit stocke l'état d'une partie du dépôt à un instant donné.
Il contient :

- Un pointeur vers un Tree (arbre racine) dont on souhaite sauver l'état.
- Un pointeur vers un ou plusieurs autres Commits pour constituer un historique.
- Les informations sur l'auteur du Commit.
- Une description sous forme d'une chaîne de caractères.

Le cycle de vie d'un fichier



Les branches

Dans Git

- Une branche est un pointeur sur un commit
- Chaque commit pointe vers son prédécesseur
- La variable HEAD pointe sur la branche sur laquelle on travaille actuellement.

Les branches – Les commandes

branch : liste les branches avec une * pour la branche active.
branch <nom> : crée une nouvelle branche <nom>.
branch -m : permet de renommer une branche.
branch -d : permet de supprimer une branche.
checkout : change (ou/et crée) de branche active.
show-branch : affiche les branches et leurs commits.

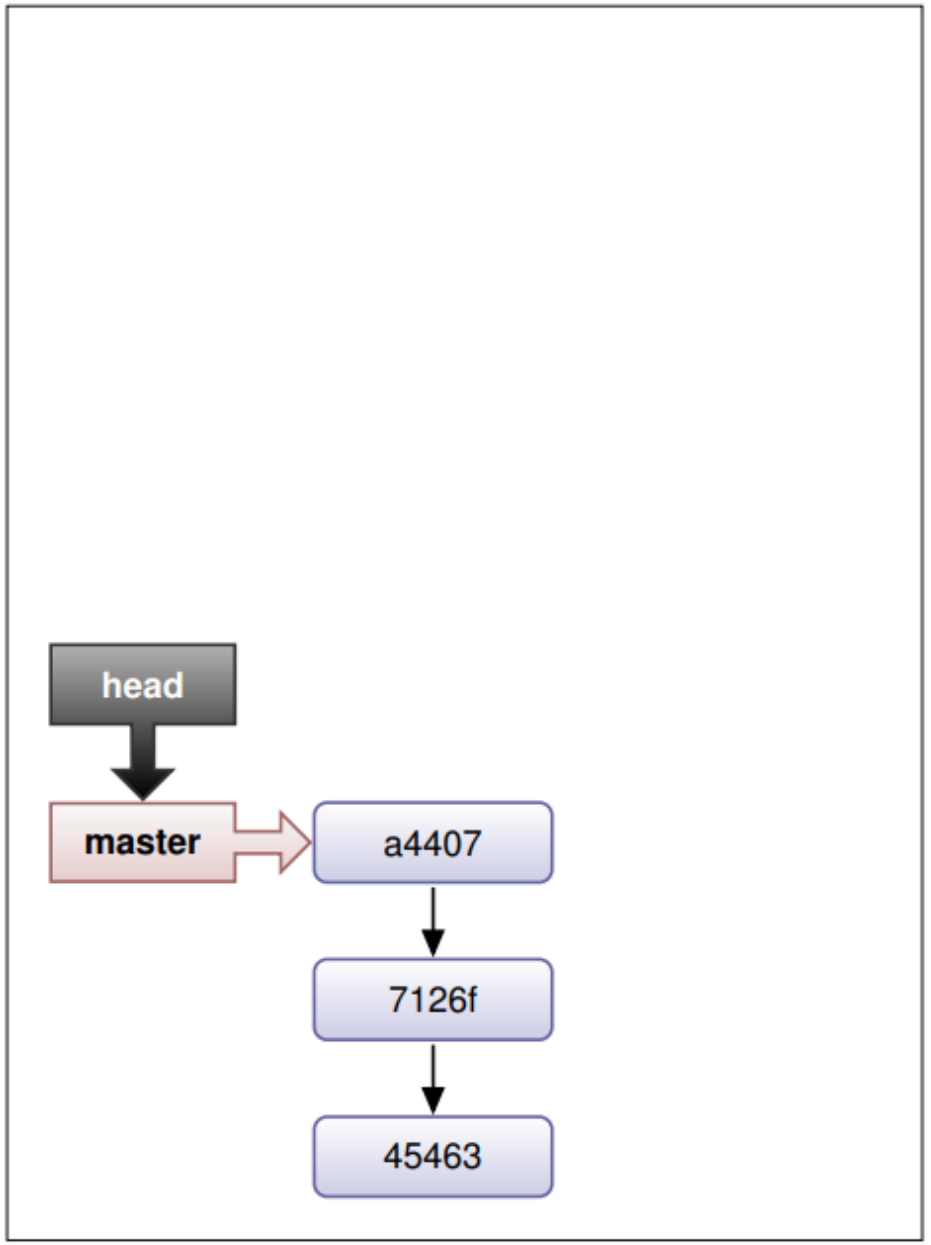
Exemple

```
$ git branch
    * master
$ git branch maBranche
$ git branch
      maBranche
    * master
$ git checkout maBranche
$ git branch
    * maBranche
      master
```

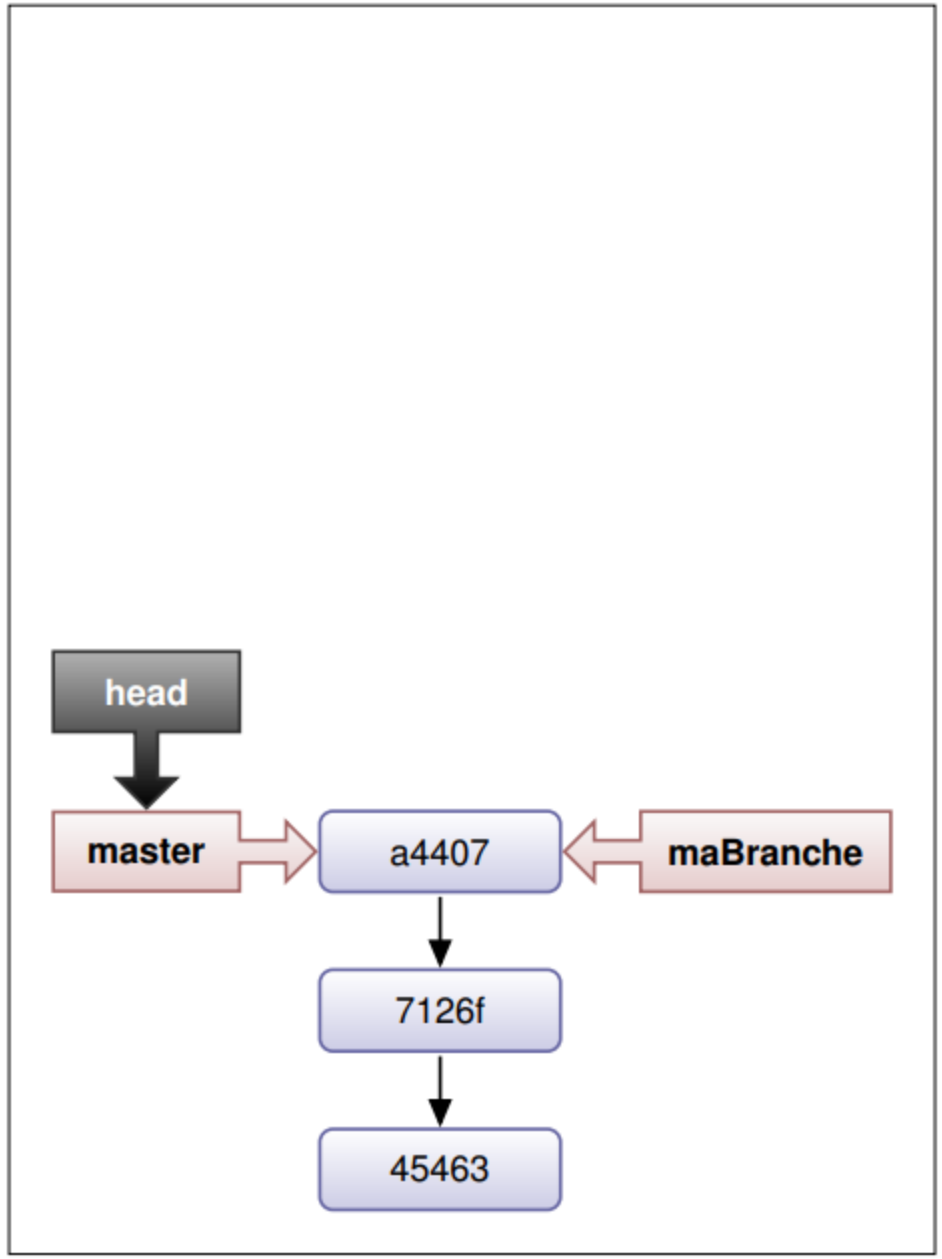
Les Merges

$ git checkout brancheDestination
$ git merge brangeSource
• Crée un commit qui a pour parent les deux branches
• La branche courante avance à ce commit
• La source ne bouge pas, mais devient un fils du nouveau commit

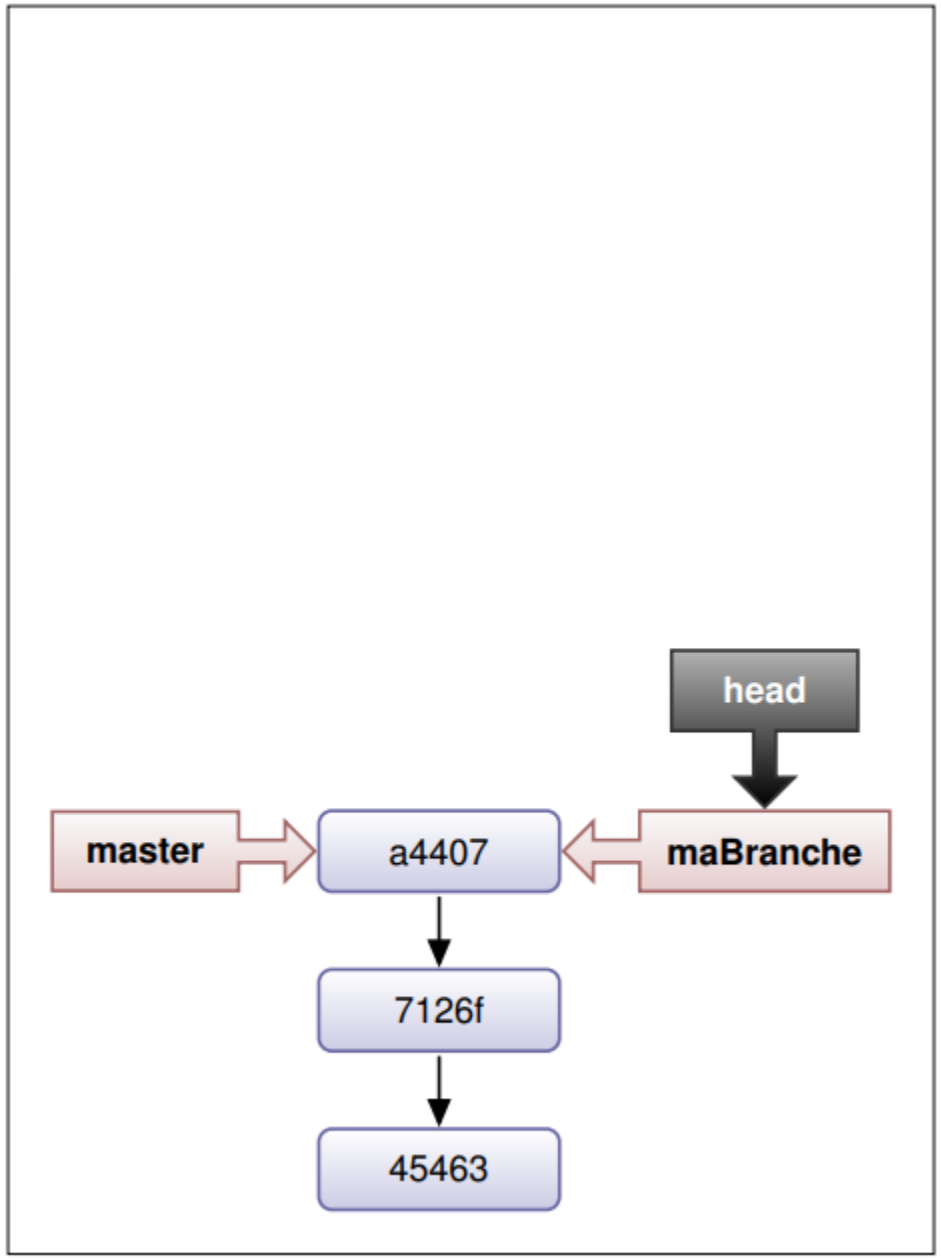
```
ls
 foo.txt dir
```



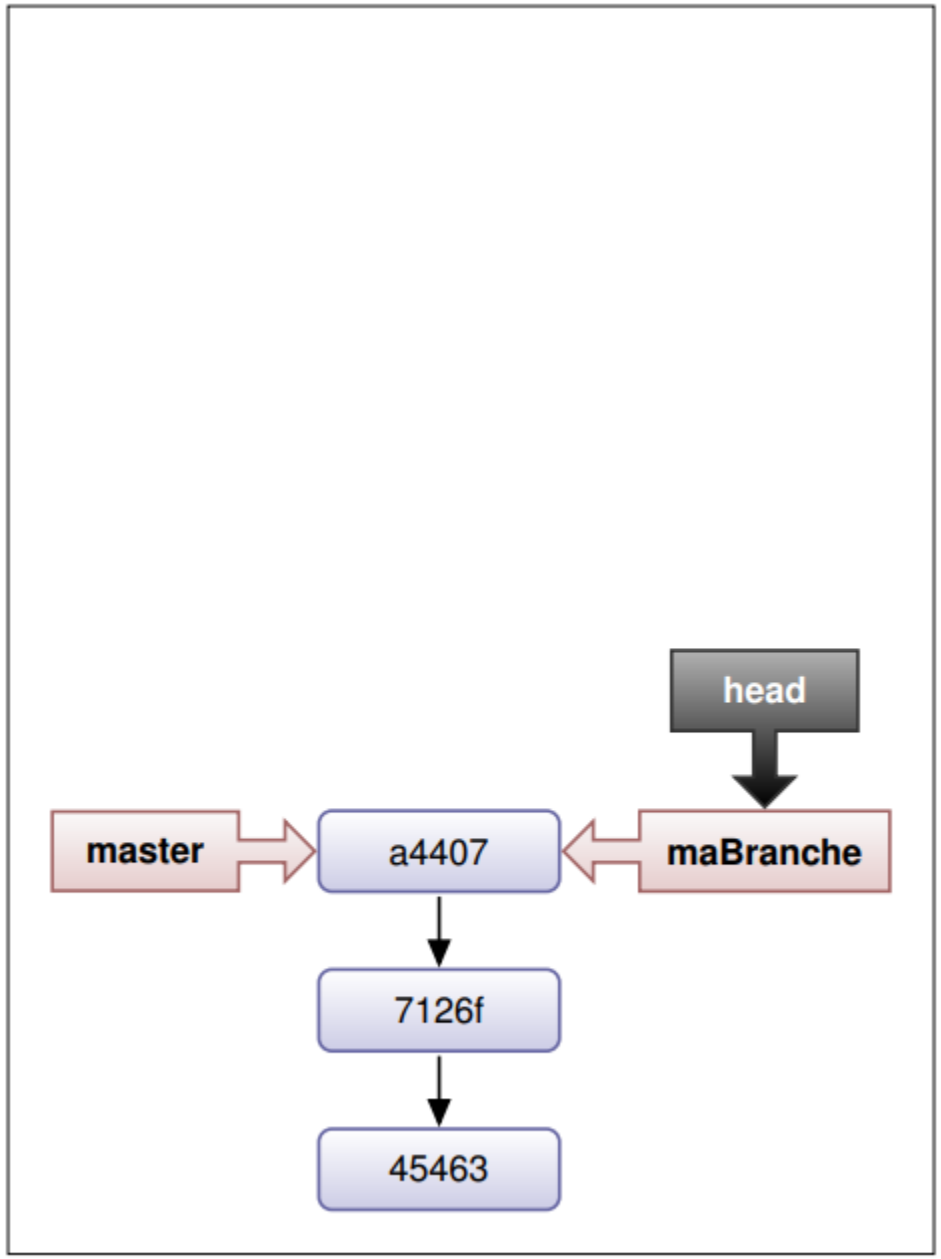
```
ls
 foo.txt dir

git branch maBranche
```



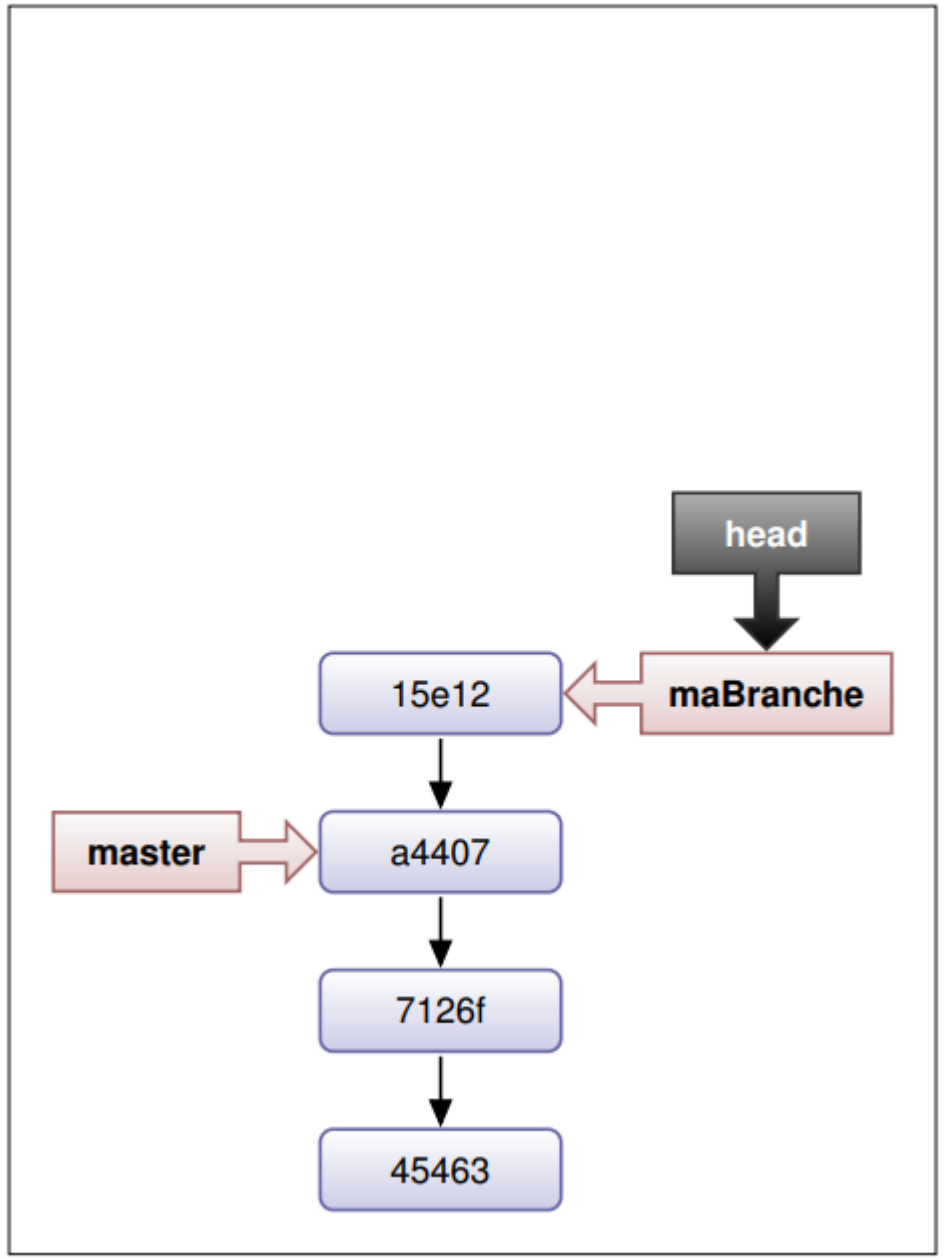
```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche
```



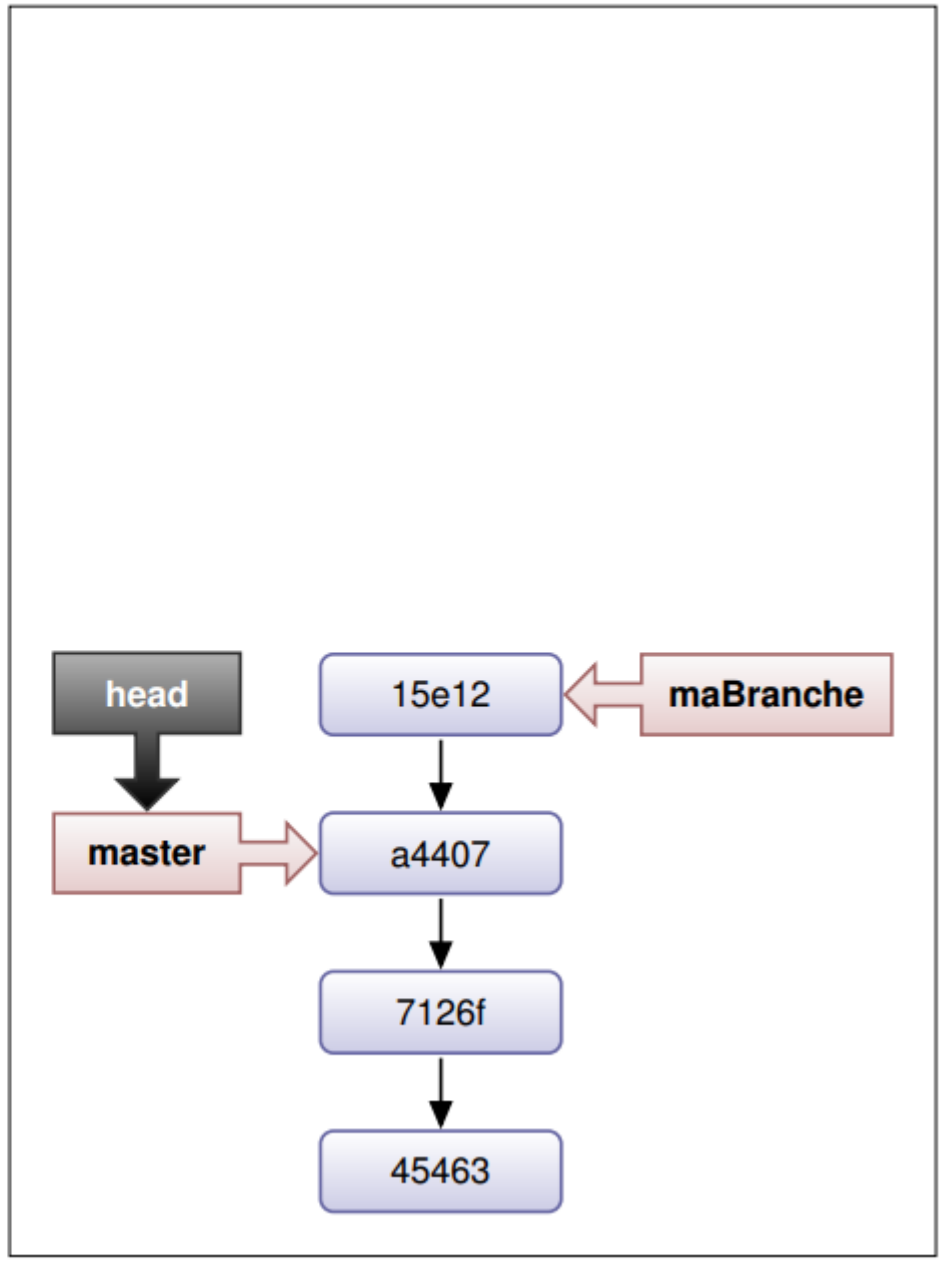
```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche
```



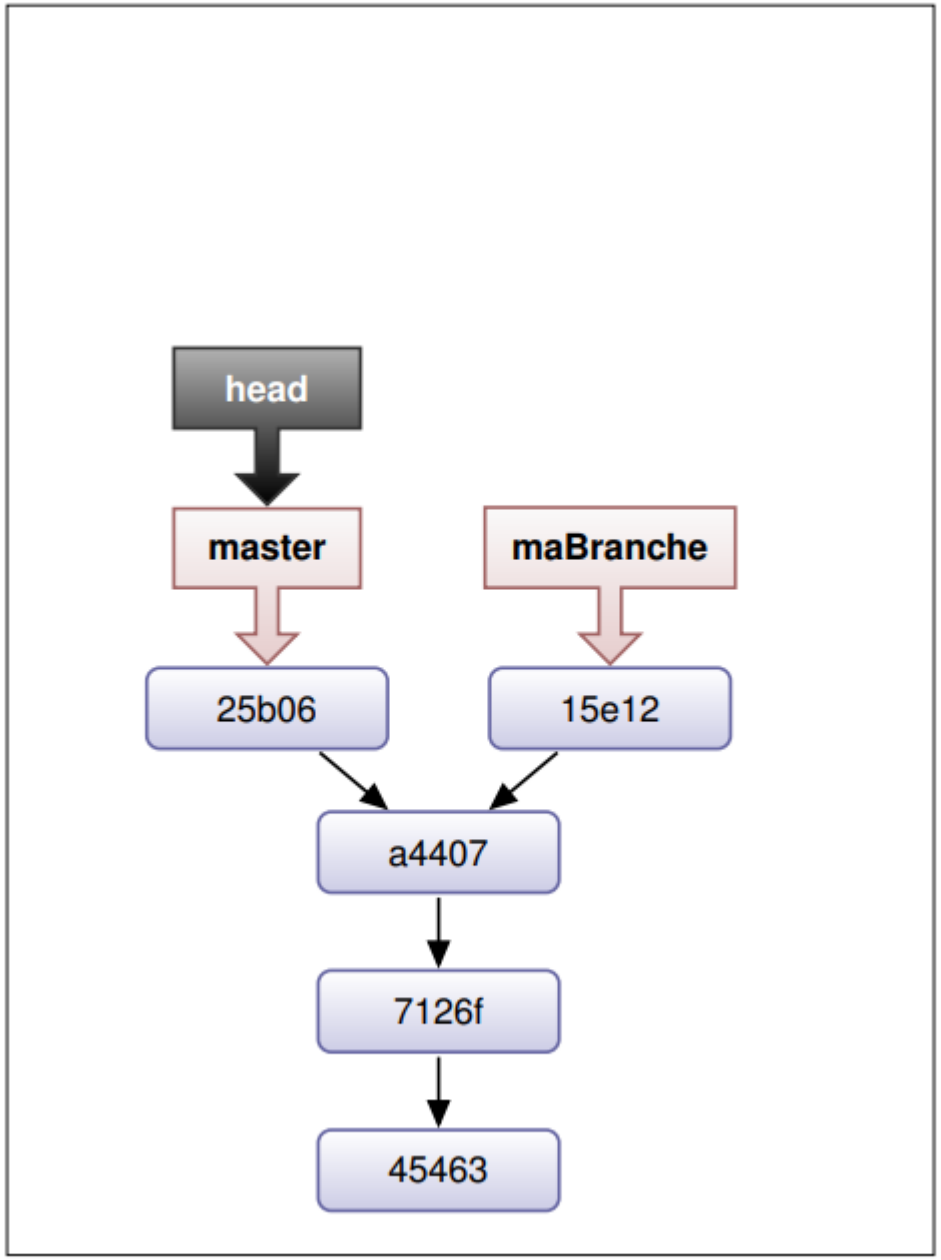
```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

touch fichier1.txt
ls
 dir fichier1.txt foo.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"
```



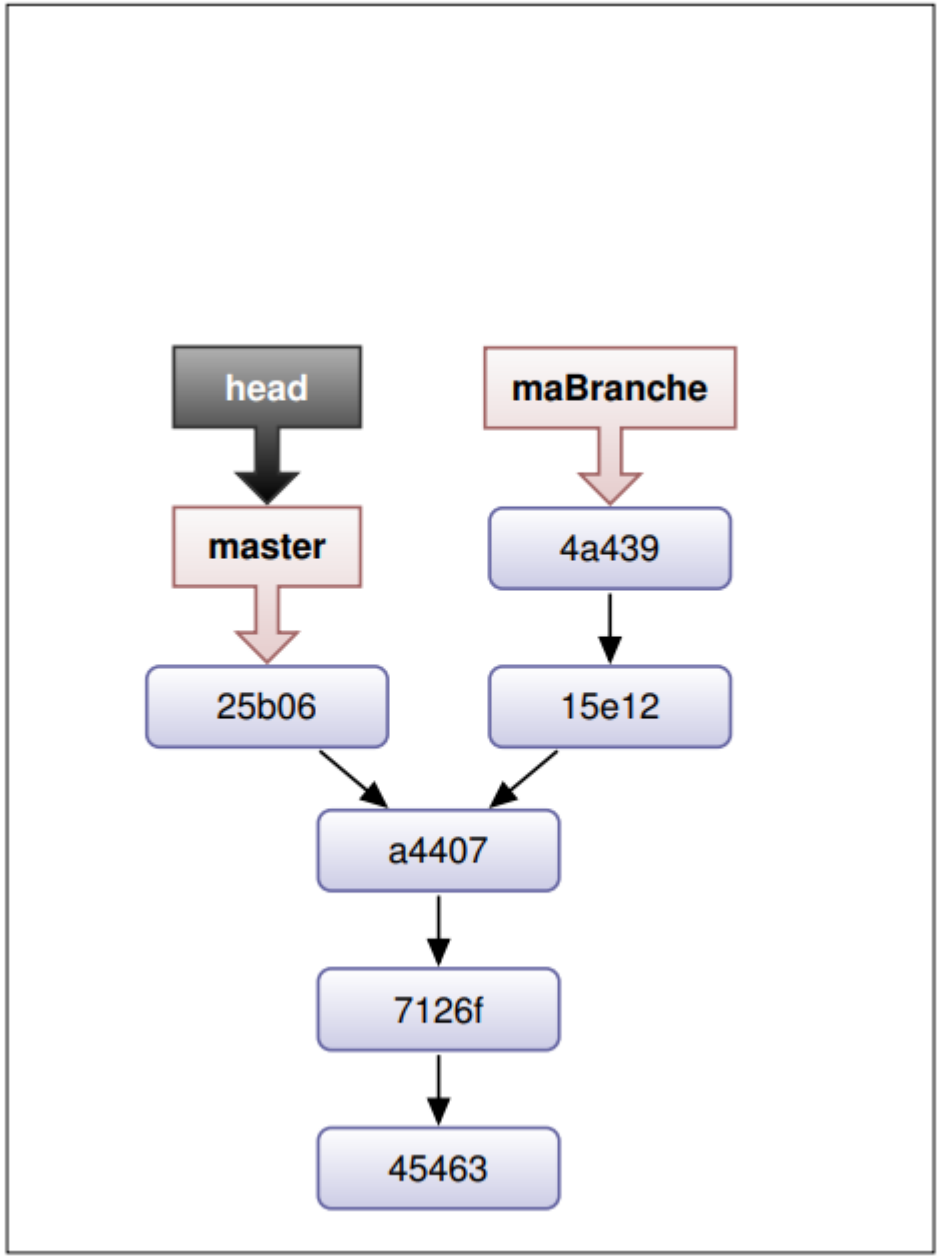
```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

touch fichier1.txt
ls
 dir fichier1.txt foo.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

git checkout master
ls
 dir foo.txt
```



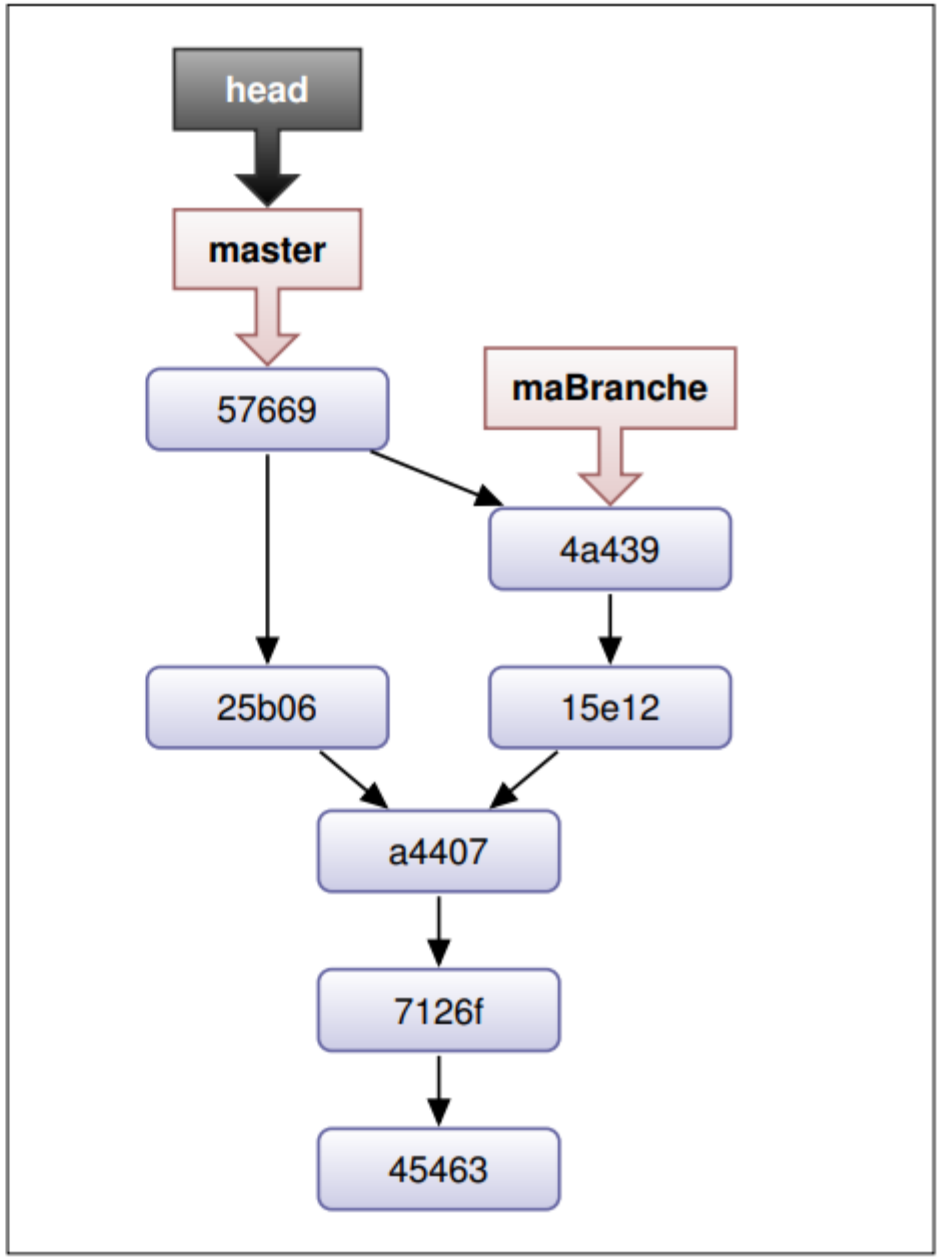
```
ls
 foo.txt dir

git branch maBranche

git checkout maBranche

touch fichier1.txt
ls
 dir fichier1.txt foo.txt
git add fichier1.txt
git commit -m "Add fichier1.txt"

git checkout master
ls
 dir foo.txt

touch fichier2.txt
git add fichier2.txt
git commit -m "Add fichier2.txt"
```



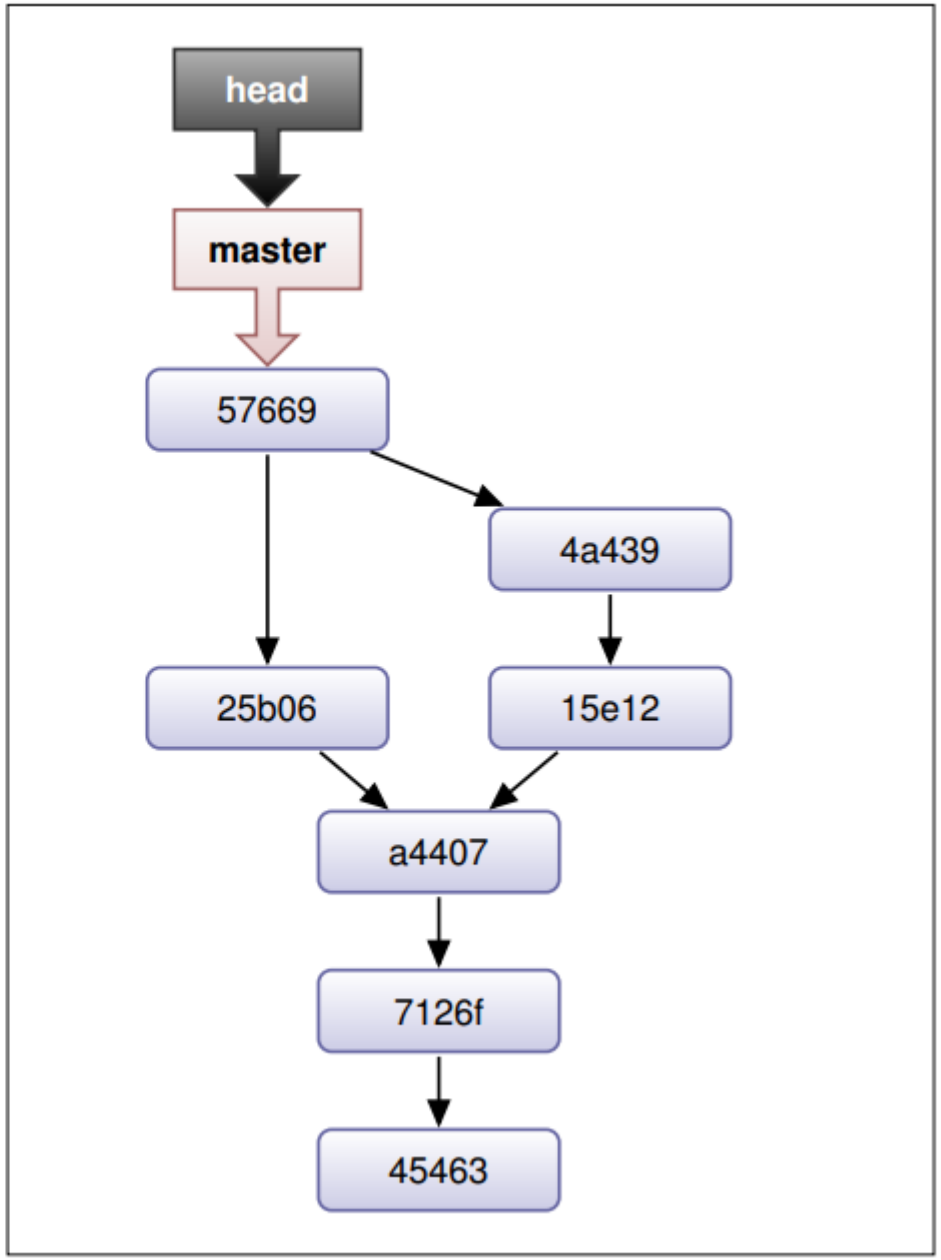
```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
```



```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
git merge maBranche
ls
 dir fichier1.txt fichier2.txt
foo.txt
```



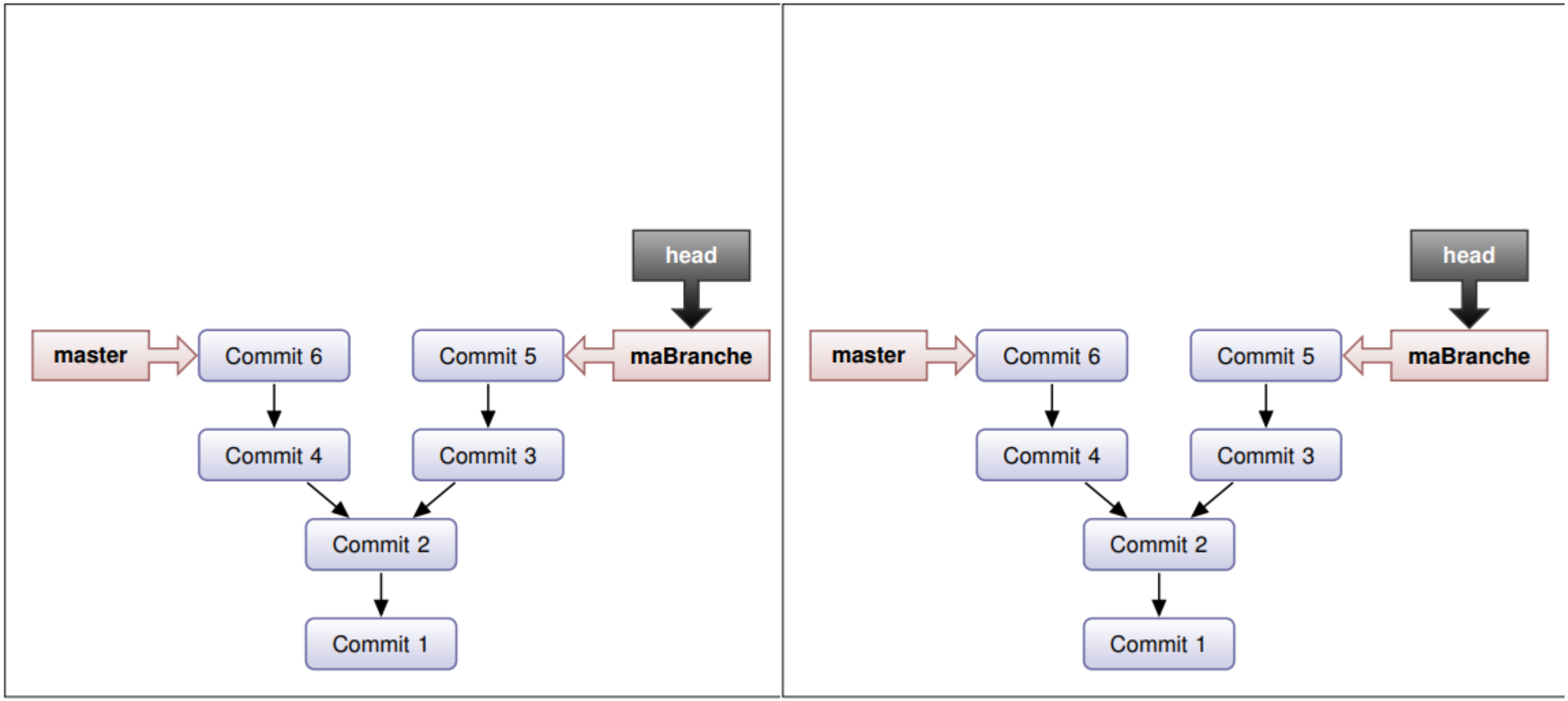
```
ls
 dir fichier2.txt foo.txt

git checkout maBranche
ls
 dir fichier1.txt foo.txt

echo "titi" > fichier1.txt
git commit -am "Modif
fichier1.txt"

git checkout master
git merge maBranche
ls
 dir fichier1.txt fichier2.txt
foo.txt

git branch -d maBranche
```



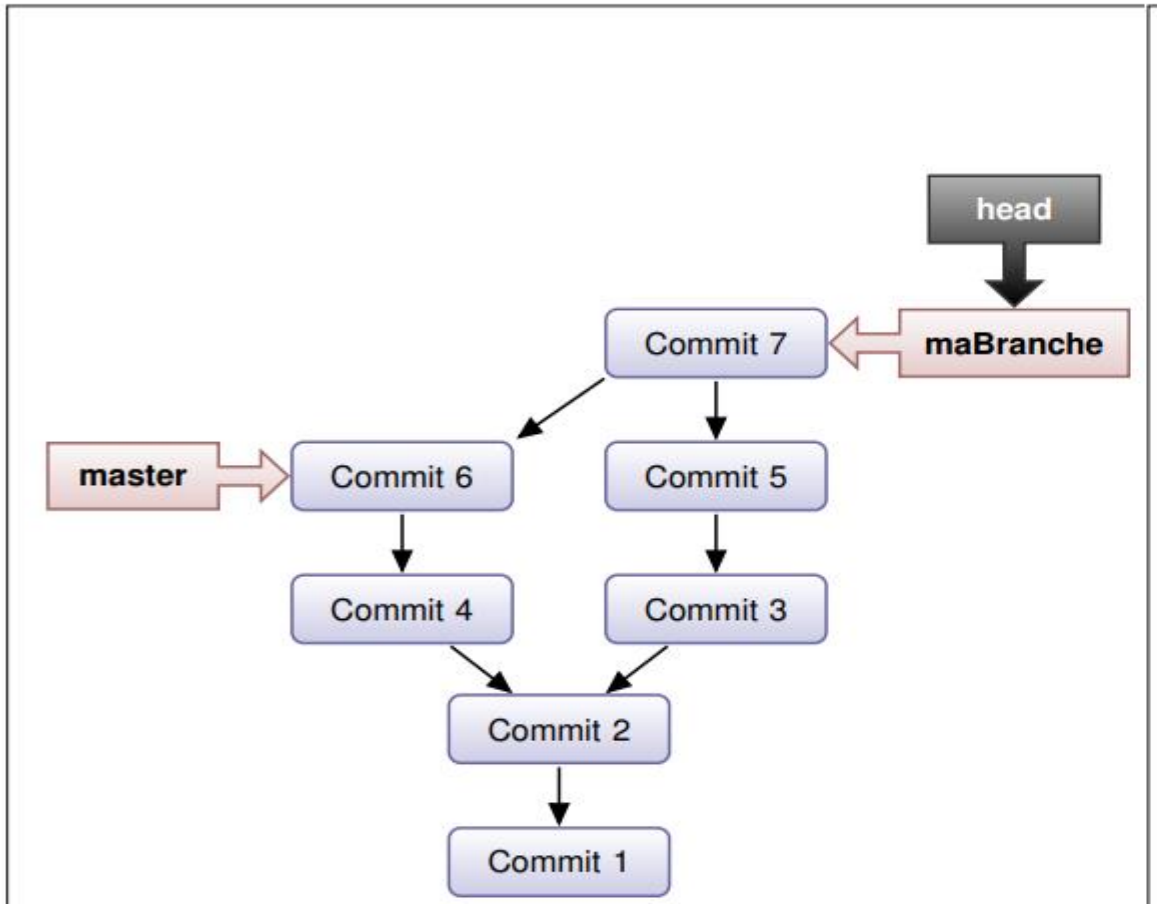
Rebase

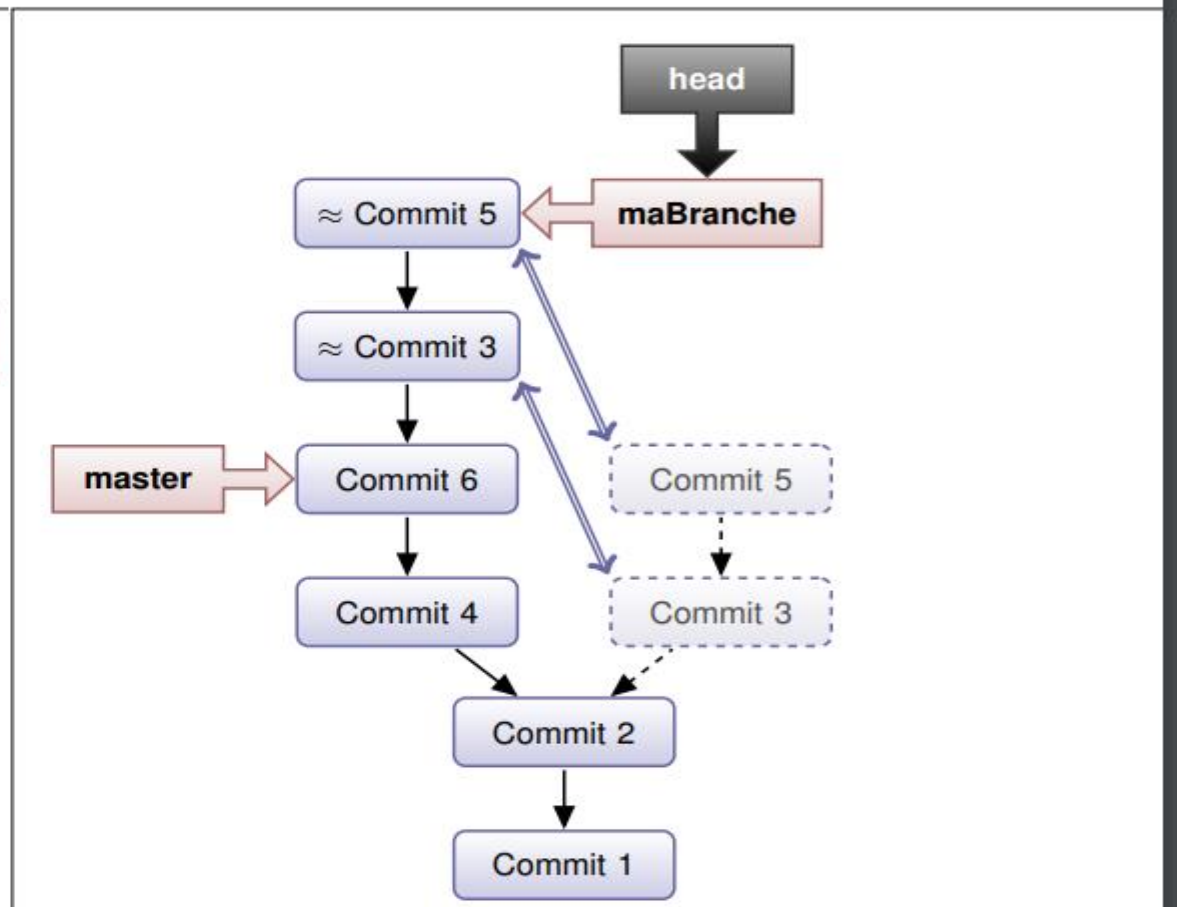
- Autre manière de fusionner 2 branches
- Permet de simplifier l’historique
- **Fusion (merge)** : Lorsque vous fusionnez deux branches avec **git merge**, Git combine l'historique des deux branches pour **créer un nouveau commit de fusion**. Ce nouveau commit de fusion a deux parents, représentant les commits les plus récents de chaque branche. En d'autres termes, Git prend tous les changements effectués dans les deux branches et les intègre dans un nouveau commit de fusion. Cela signifie que l'historique de développement de chaque branche est préservé.
- **Réalignement (rebase)** : Lorsque vous réalignez une branche avec une autre branche à l'aide de **git rebase**, **Git déplace les commits de la branche en cours de réalignement pour qu'ils apparaissent au-dessus des commits de l'autre branche**. Cela récrit essentiellement l'historique de la branche en cours sur le dessus de l'autre branche. Contrairement à la fusion, le réalignement n'ajoute pas de commit de fusion supplémentaire. Au lieu de cela, il réorganise l'historique de manière à ce que l'historique semble linéaire, ce qui peut rendre l'historique du projet plus clair.

GIT MERGE VS GIT REBASE



GIT MERGE VS GIT REBASE






```
git checkout maBranche
git merge master
```

```
git checkout maBranche
git rebase master
```

Rebase interactif

- Permet de réécrire l'historique en partant de CommitID
- Principales actions possibles
  - reword: editer le message du commit
  - squash: fondre le commit dans le commit précédent
  - drop: supprimer le commit

Message de commit

Le plus important

- Décrire quoi et pourquoi et pas comment
- Ne pas décrire les modifications qui sont faites (informations disponibles avec un diff)
- Décrire les fonctionnalités ajoutées

Exemple

- Bad: Modifie la fonction f pour tester la variable a
- Good: Vérifie les droits de l'utilisateur avant d'exécuter l'action X

Azure Boards

C’EST QUOI LA RELATION ENTRE AZURE ET DEVOPS?

Définition d'Azure Microsoft

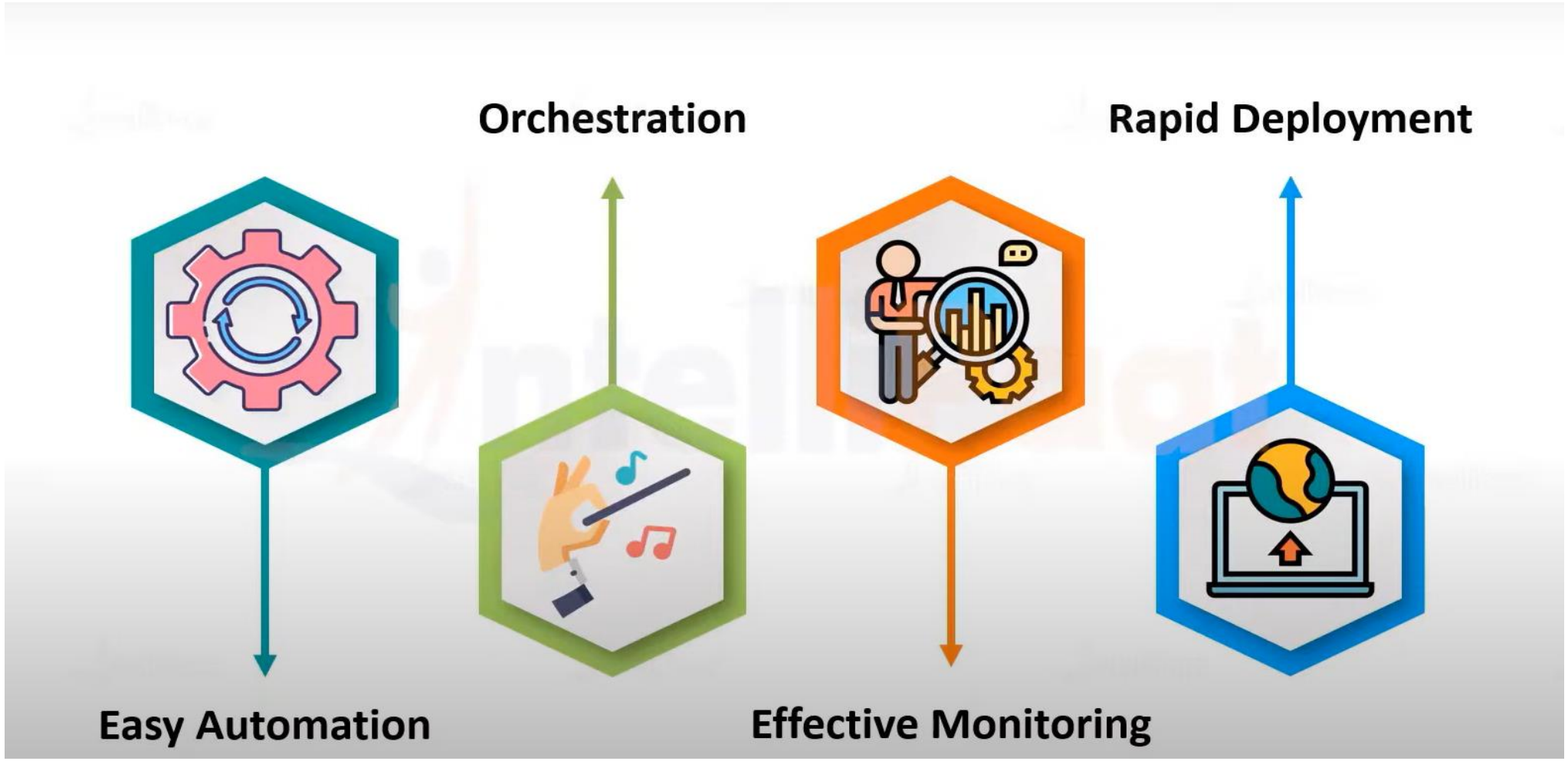
Azure est une plateforme de Cloud qui dispose de tout ce dont une entreprise a besoin pour gérer virtuellement une partie ou la totalité de ses opérations informatiques : serveurs, stockage, bases de données, réseaux, statistiques et bien plus encore.

Définition d'Azure Devops

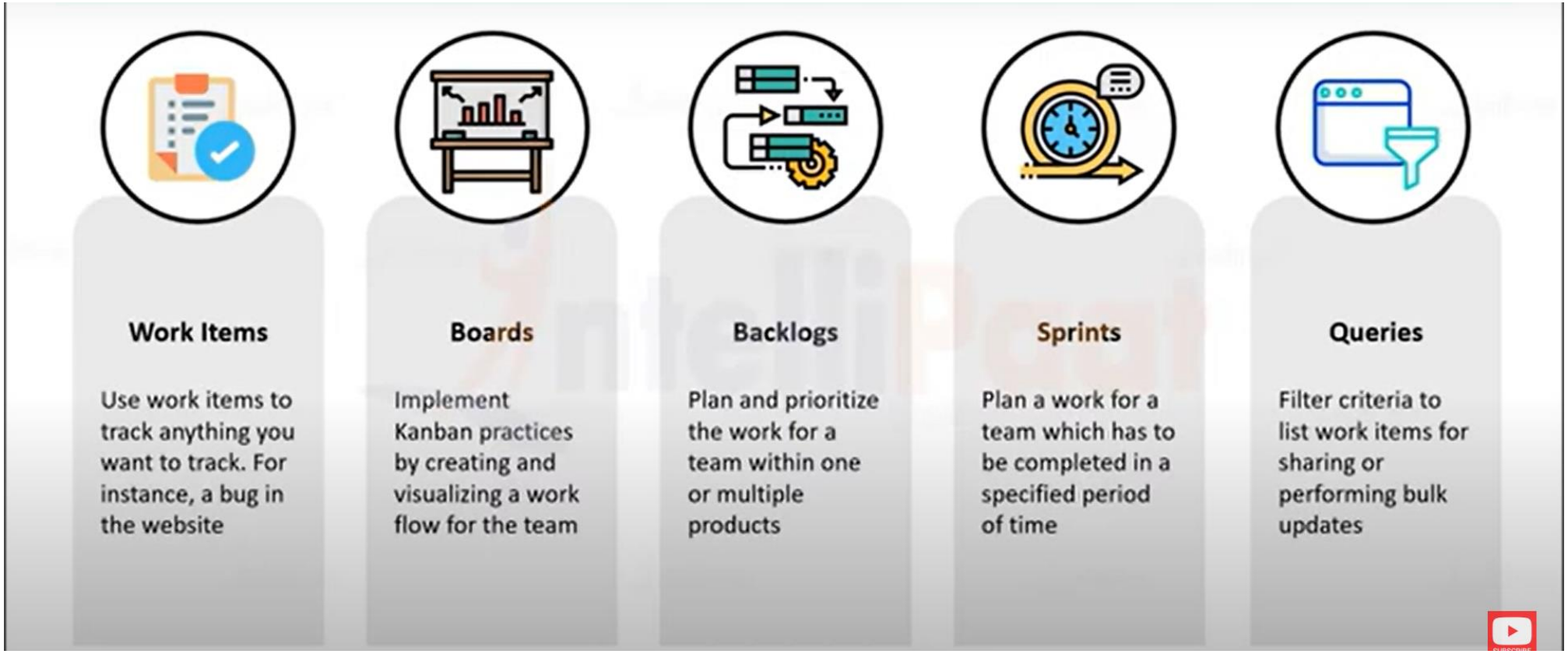
La plateforme Azure DevOps, facilite la collaboration entre deux métiers qui travaillaient auparavant séparément : le développeur de programmes et de logiciels (le dev) et le responsable des infrastructures informatiques chargé de la mise en production (l'Ops ou opérateur).

Azure DevOps dispose d'intégrations robustes avec Azure, ainsi que d'une suite complète de technologies qui vous permettent de fournir des logiciels rapidement et en toute sécurité.

AVANTAGES DE DEVOPS SUR AZURE



CONCEPTS DE DEVOPS



LES PRINCIPAUX CONCEPTS D'AZURE BOARDS

- **Boards ‘Tableau’**

Il s’agit des espaces de travail visuels où les équipes peuvent organiser leurs travaux en utilisant des colonnes personnalisables, souvent représentant l'état de progression des tâches (à faire, en cours, terminé, etc.).

- **Work item ‘Eléments du travail’**

C’est une unité de tâche, de problème, de fonctionnalité ou d'autres éléments à gérer.
Il s'agit de travaux individuels qui doivent être réalisés.

- **Backlog**

Le backlog peut être composé de différents types d'éléments, tels que des user stories, des bugs, des tâches, etc. Les éléments du backlog sont généralement triés par ordre de priorité et sont planifiés pour être réalisés au fil du temps

- **Sprint**

Un sprint est une période de temps définie, généralement de deux à quatre semaines, pendant laquelle une équipe de développement travaille sur un ensemble spécifique de tâches et de fonctionnalités.

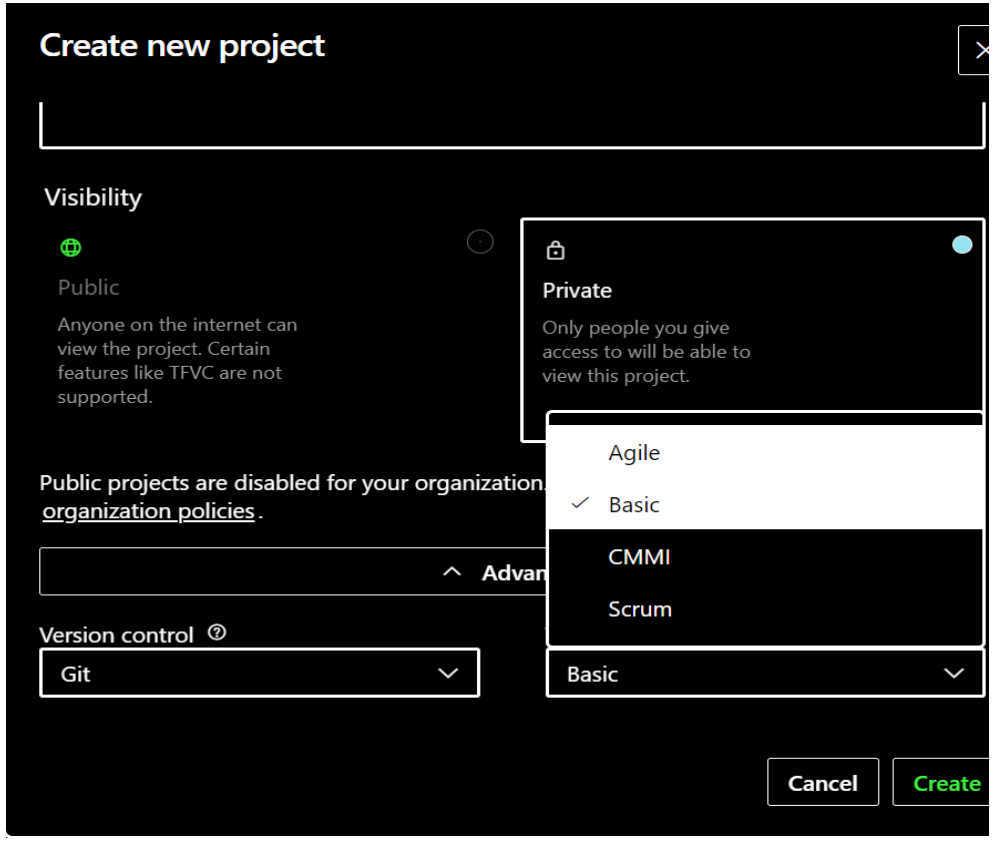
LES PRINCIPAUX CONCEPTS D'AZURE BOARDS

- **Queries**

Les requêtes (queries) dans Azure DevOps permettent de filtrer et d'interroger les données, y compris les éléments du backlog. Vous pouvez créer des requêtes pour afficher des sous-ensembles spécifiques du backlog, comme les éléments affectés à une personne particulière, les éléments qui n'ont pas encore été planifiés pour un sprint, etc.

LES DIFFÉRENTS WORK ITEM PROCESS

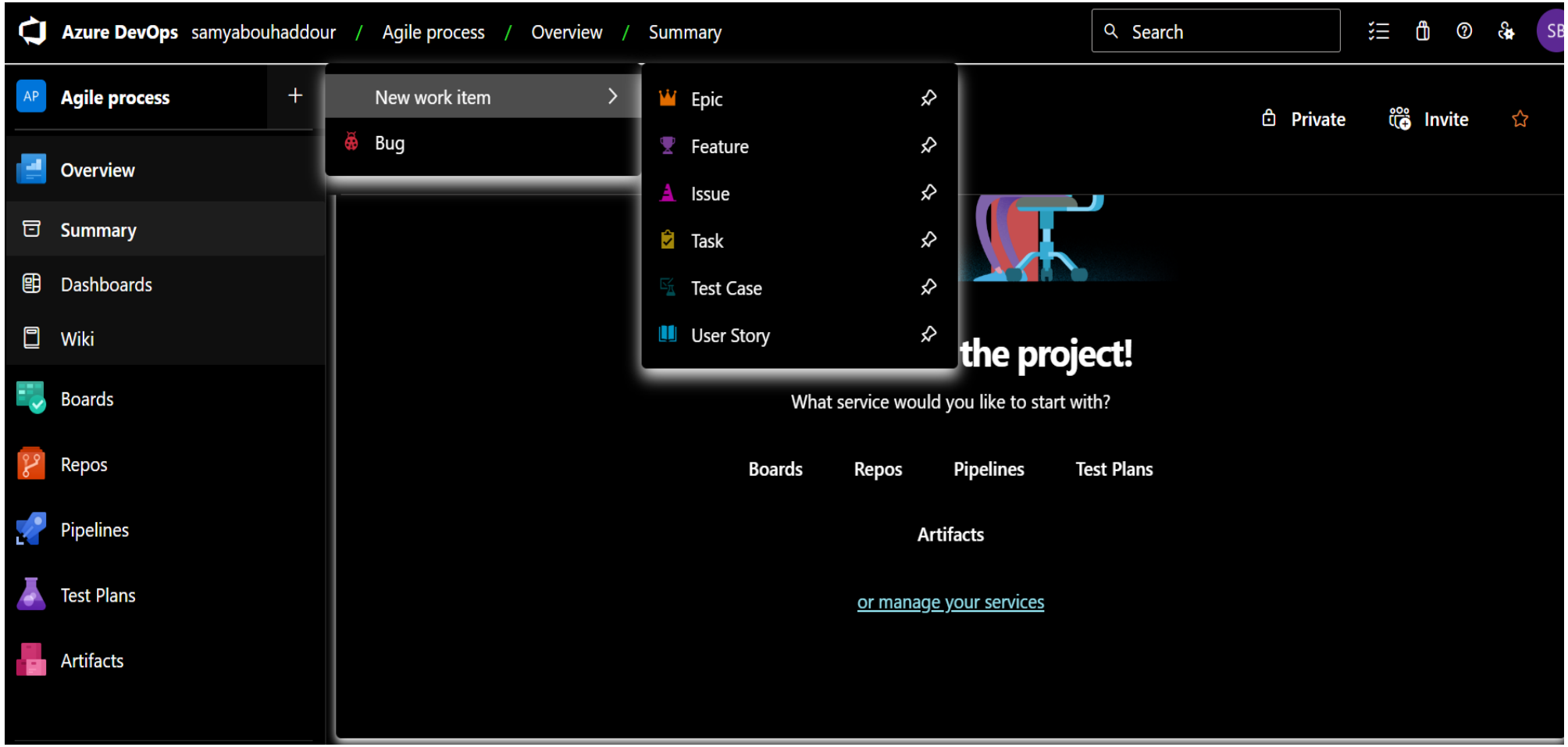
- **Basic** est le plus léger et se trouve dans un aperçu sélectif.
- **Scrum** est le deuxième plus léger.
- **Agile** prend en charge de nombreux termes de méthode Agile.
- **CMMI** fournit le plus grand support pour les processus formels et la gestion du changement.



WORK ITEM PROCESS : AGILE

Choisissez **Agile** lorsque votre équipe utilise des méthodes de planification Agile et suit séparément les activités de développement et de test.
Ce processus fonctionne très bien si vous souhaitez suivre les user stories et (éventuellement) les bugs et/ou les tâches.

Work item process : AGILE



Hiérarchie des éléments de travail dans Azure Boards - AGILE

- **Epic (Épopée)** :

Un "épic" représente un grand ensemble de fonctionnalités ou de tâches. C'est une unité de travail de haut niveau qui peut être divisée en fonctionnalités plus petites ou en user stories. Les épics fournissent une vue d'ensemble des objectifs à long terme du projet.

- **Feature (Fonctionnalité) :**

Une "fonctionnalité" est une unité de travail plus petite que l'épic, mais plus grande que l'user story. Elle représente une partie spécifique d'un épic. Les fonctionnalités sont souvent des ensembles cohérents de fonctionnalités liées entre elles.

- **Issue (Problème) :**

Une "Issue" est un élément de suivi qui représente un problème, un bogue, une amélioration ou toute autre tâche à résoudre dans le cadre d'un projet.
Il peut être utilisée pour signaler des problèmes techniques, des obstacles, des retards ou des demandes de fonctionnalités.
Les issues peuvent être assignées à des membres de l'équipe pour résolution et suivi.
Ils peuvent également être liés à d'autres éléments de travail pour montrer leur impact sur le projet en cours.

- **User Story :**

C’est une unité de travail centrée sur l'utilisateur. Elle décrit une fonctionnalité du point de vue de l'utilisateur final et fournit des détails sur ce que l'utilisateur souhaite accomplir.

- **Task (Tâche) :**

Une "tâche" est une unité de travail encore plus petite, souvent associée à une user story. Les tâches décrivent les actions spécifiques nécessaires pour réaliser une user story. Elles sont généralement assignées à des membres de l'équipe et sont suivies pour surveiller l'avancement du travail.

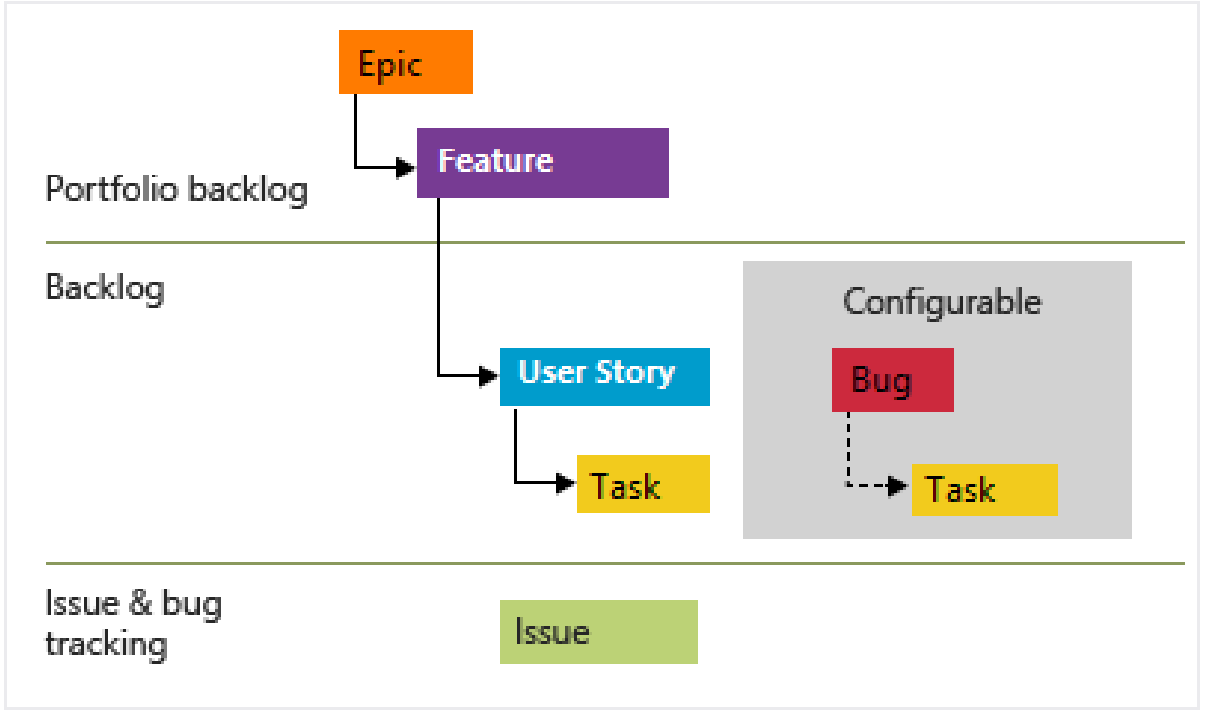
- **Test Case (Cas de Test) :**

Un test case est une documentation décrivant une série d'étapes spécifiques à suivre, ainsi que les conditions préalables et les résultats attendus, pour tester une fonctionnalité particulière d'un logiciel.

Work item process : AGILE

Le schéma suivant montre le processus Agile :

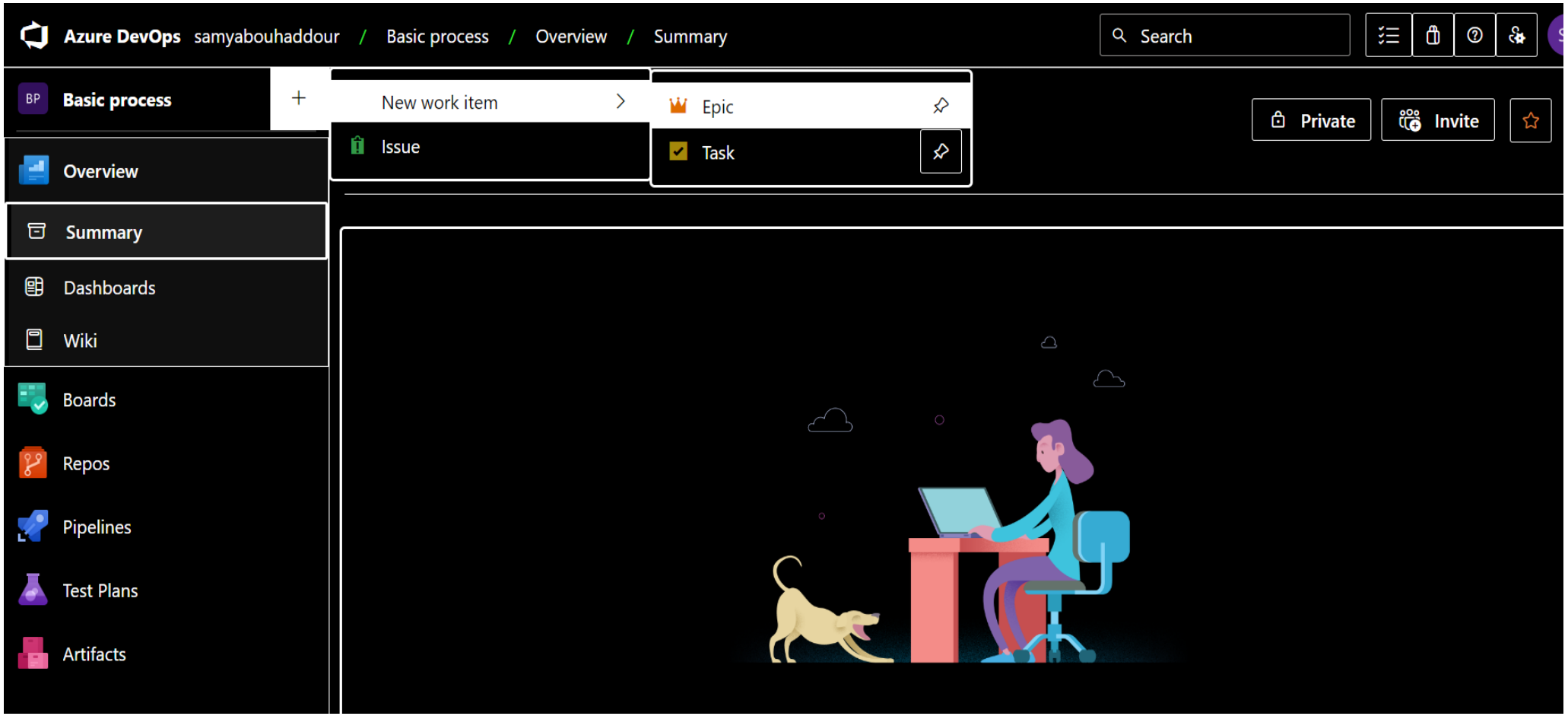
- ✓ Les Users Story et les Tasks sont utilisés pour le suivi du travail.
- ✓ Les Bugs sont utilisés pour les defects.
- ✓ Epics et Features sont utilisés pour suivre le travail des gros scénario



WORK ITEM PROCESS : BASIC

Choisissez **BASIC** lorsque votre équipe souhaite le modèle le plus simple qui utilise les problèmes, les tâches et les Epics pour suivre le travail.
Les tâches prennent en charge le suivi du travail restant.

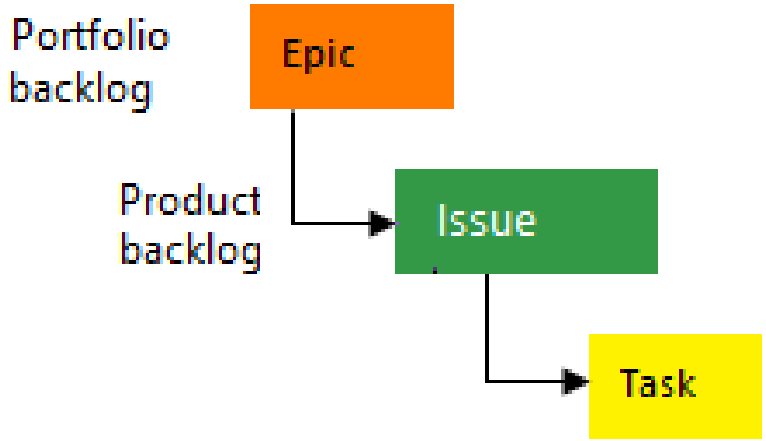
Work item process : BASIC



Work item process : BASIC

Le schéma suivant montre le processus Basic :

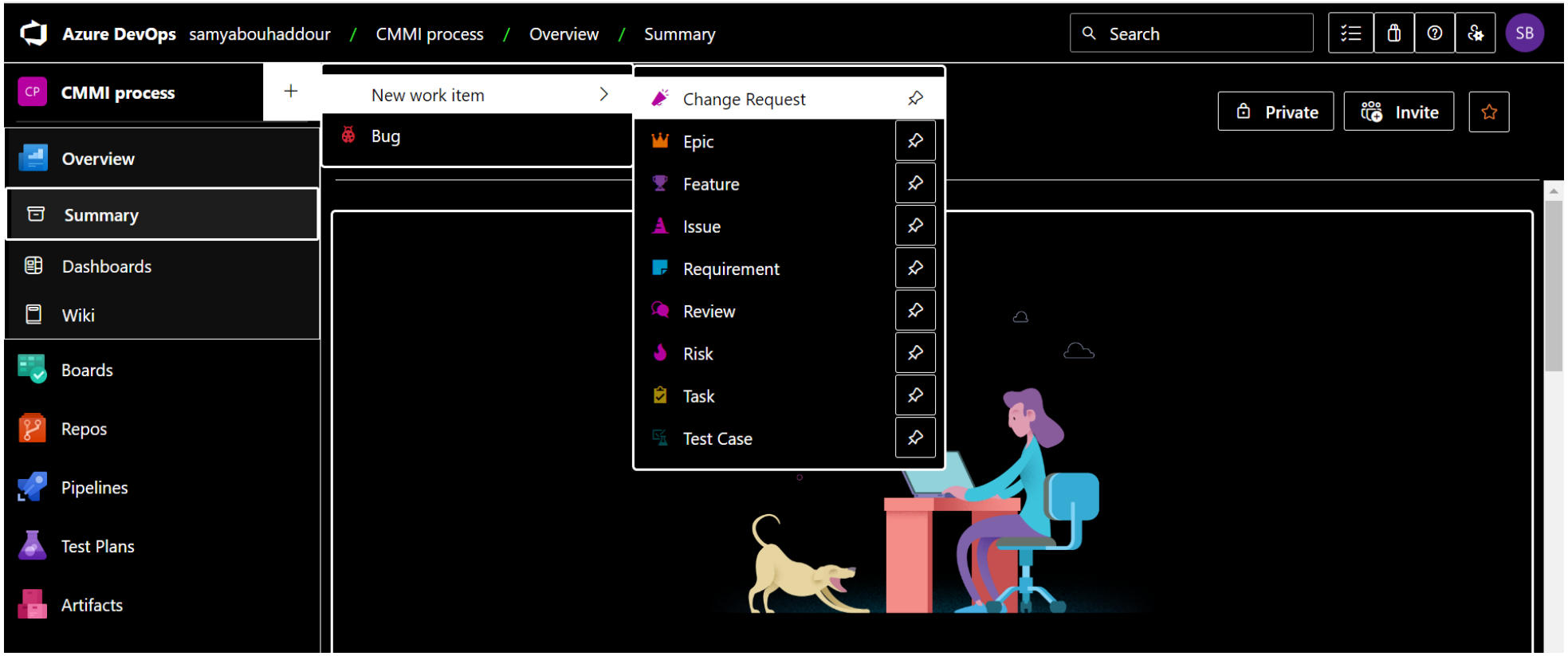
- ✓ Les Issues et les Tasks sont utilisés pour le suivi du travail.
- ✓ Epics sont utilisés pour suivre le travail des gros scénario



WORK ITEM PROCESS : CMMI

Choisissez **CMMI** lorsque votre équipe suit des méthodes de projet plus formelles qui nécessitent un cadre d'amélioration des processus et un enregistrement vérifiable des décisions. Avec ce processus, suivez les exigences, les demandes de modification, les risques et les avis.
Ce processus prend en charge les activités formelles de gestion du changement . Les tâches prennent en charge le suivi de l'estimation initiale, du travail restant et du travail terminé.

Work item process : CMMI



Hiérarchie des éléments de travail dans Azure Boards - CMMI

- **Epic (Épopée)**
- **Fonctionnalité (Feature)**
- **Exigence (Requirement)**

  Dans le contexte de CMMI, une exigence peut être considérée comme un synonyme d'une user story ou d'une fonctionnalité.
- **Revue (Review)**

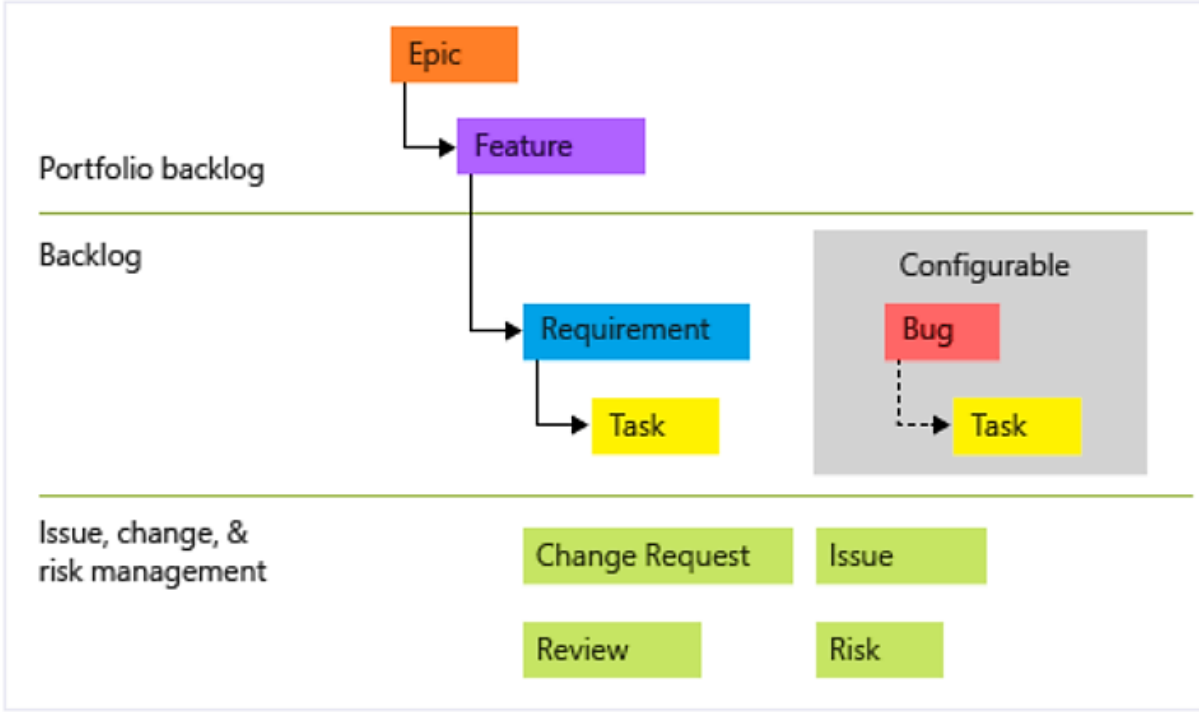
  Une activité de validation où le travail accompli est examiné et évalué par l'équipe et les parties prenantes.
- **Risque (Risk)**

  Un élément potentiellement négatif ou une incertitude qui pourrait affecter le succès du projet.
- **Tâche (Task)**
- **Cas de Test (Test Case) :**

Work item process : CMMI

Le schéma suivant montre le processus CMMI :

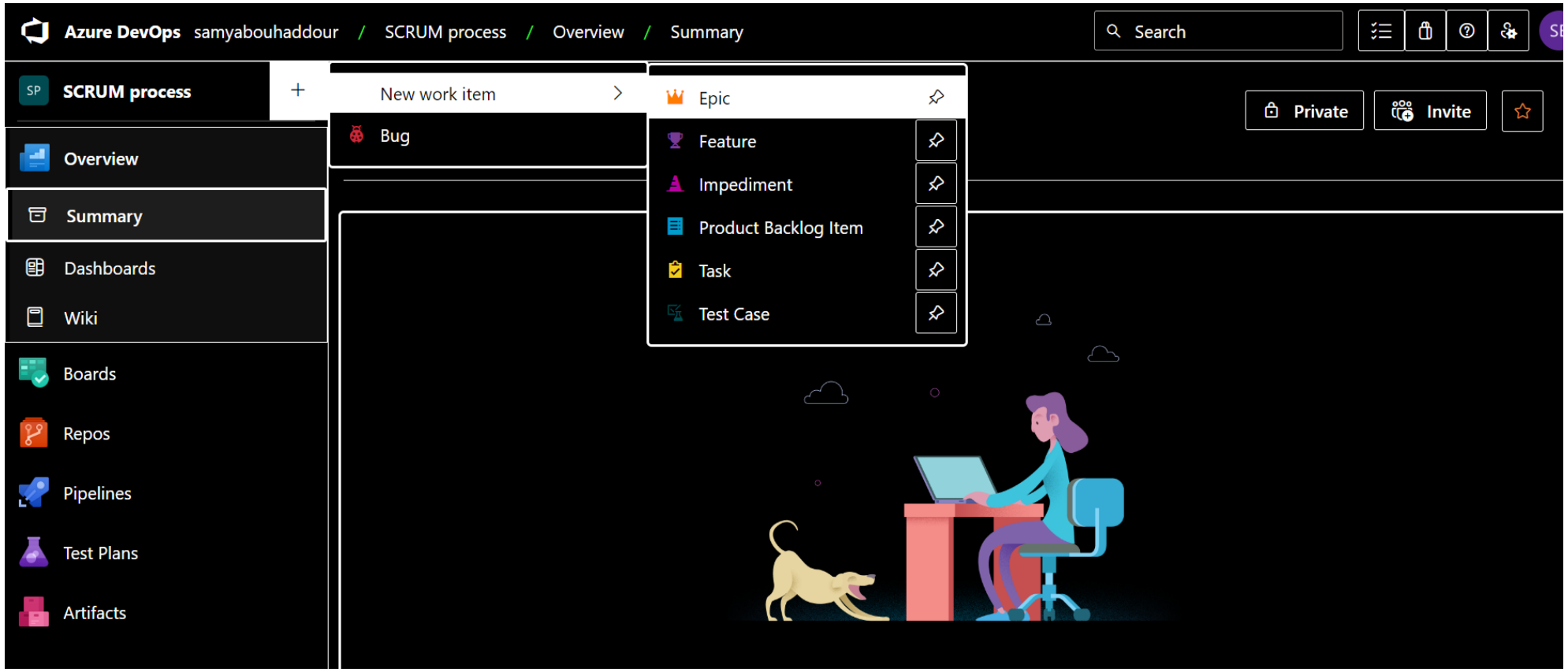
- ✓ Les Requirement et les Tasks sont utilisés pour le suivi du travail.
- ✓ Les Bugs pour le code defects (anomalie).
- ✓ Epics et Feature sont utilisés pour suivre le travail des gros scénario (Niveau Portfolio Backlog).



WORK ITEM PROCESS : SCRUM

Choisissez **Scrum** lorsque votre équipe pratique Scrum. Ce processus fonctionne très bien pour suivre les éléments du backlog de produit (PBI) et les bogues. Décomposez également les PBI et les bogues en tâches sur le tableau des tâches.

Work item process : SCRUM



Hiérarchie des éléments de travail dans Azure Boards - SCRUM

- **Épic (Epic)**
- **Fonctionnalité (Feature)**
- **Obstacle (Impediment)**

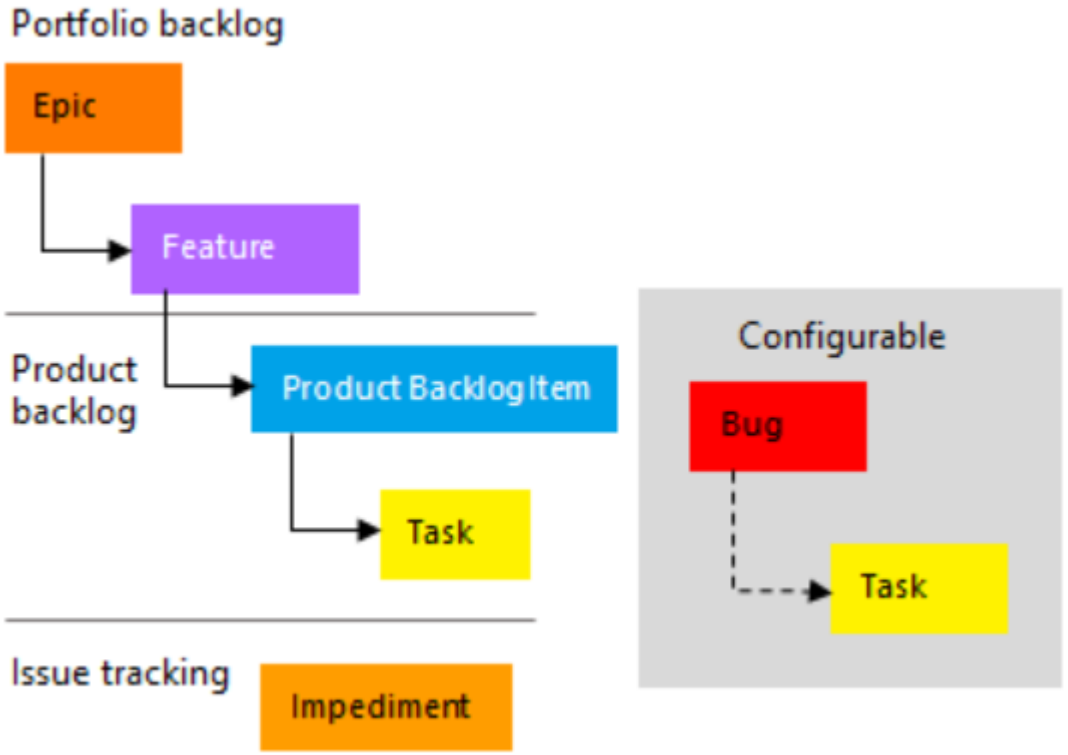
  Un élément qui entrave ou bloque le progrès de l'équipe. Les obstacles doivent être résolus rapidement pour assurer un flux de travail efficace.
- **Product Backlog**

  Une liste ordonnée de toutes les fonctionnalités, exigences, améliorations et bugs potentiels du produit. Il est priorisé par le Product Owner et est continuellement raffiné et mis à jour.
- **Tâche (Task)**
- **Cas de Test (Test Case)**

Work item process : SCRUM

Le schéma suivant montre le processus SCRUM :

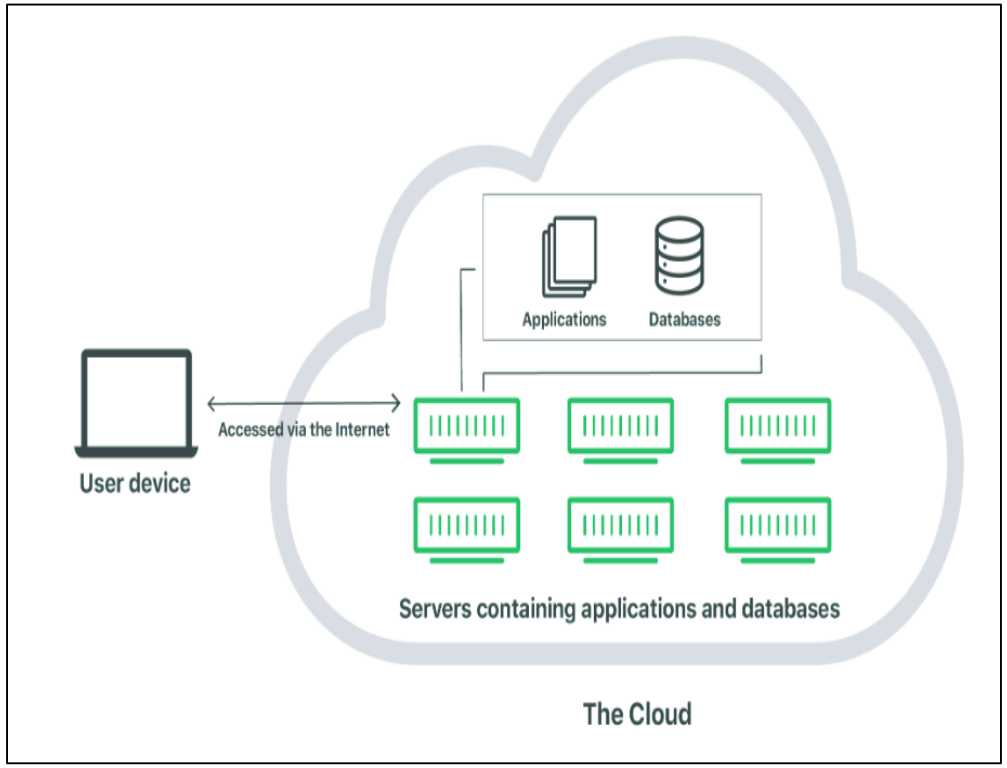
- ✓ Les Product Backlog Item et les Tasks sont utilisés pour le suivi du travail.
- ✓ Les Bugs pour le code defects (anomalie).
- ✓ Epics et Feature sont utilisés pour suivre le travail des gros scénario (Niveau Portfolio Backlog).



DOCKER

DÉFINITION DU CLOUD

Le terme « cloud » désigne les serveurs accessibles sur Internet, ainsi que les logiciels et bases de données qui fonctionnent sur ces serveurs. Les serveurs situés dans le cloud sont hébergés au sein de datacenters répartis dans le monde entier. L'utilisation du cloud computing permet aux utilisateurs et aux entreprises de s'affranchir de la nécessité de gérer des serveurs physiques eux-mêmes ou d'exécuter des applications logicielles sur leurs propres équipements.



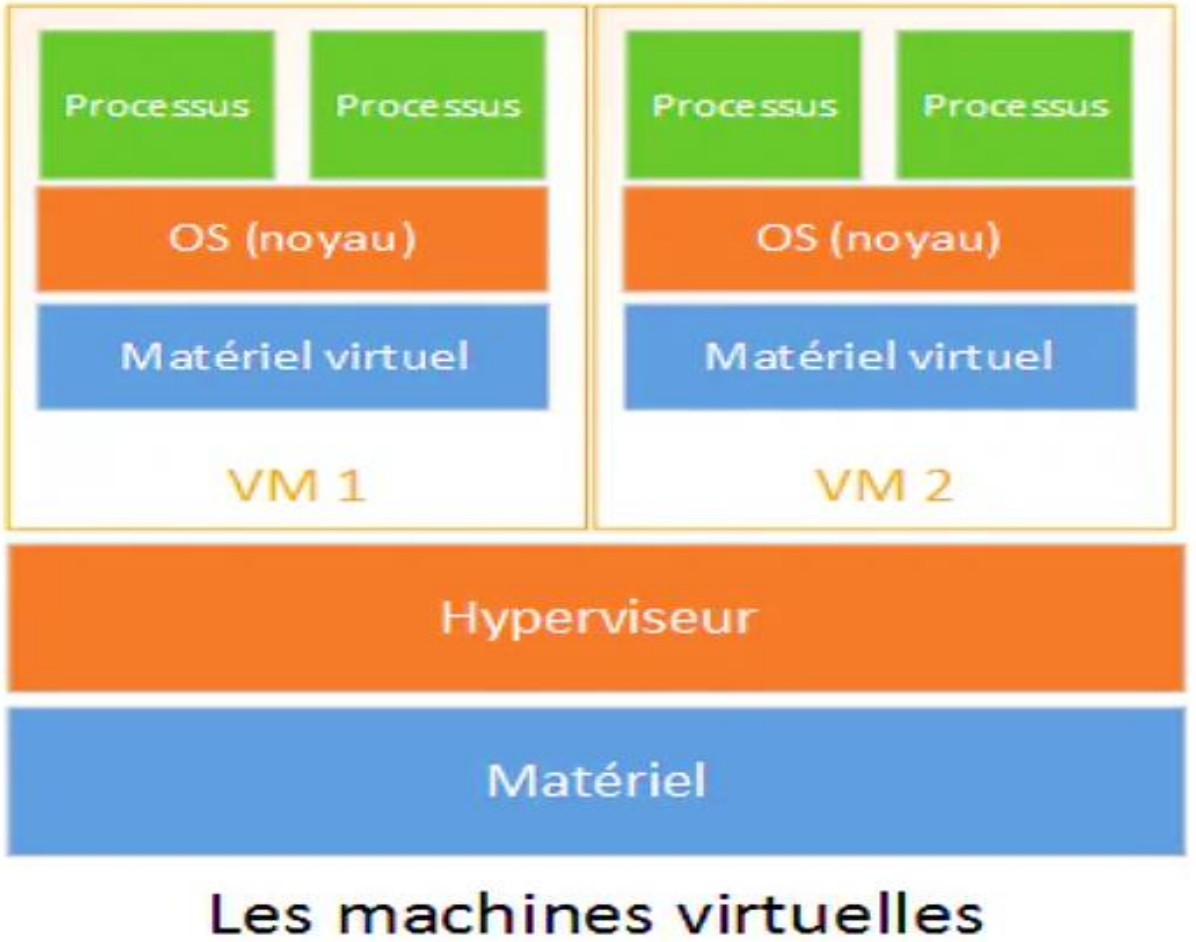
LA VIRTUALISATION

- La virtualisation est un mécanisme informatique qui consiste à faire fonctionner plusieurs systèmes, serveurs ou applications, sur un même serveur physique.
- La virtualisation est une technologie qui permet de créer et d’exécuter une version virtuelle des machines physiques, ainsi que des ressources associées.

LA VIRTUALISATION

La virtualisation est une technologie qui vous permet de ***créer plusieurs environnements simulés ou ressources dédiées à partir d'un seul système physique***.
Son logiciel, appelé ***hyperviseur***, est directement relié au matériel et vous permet de fragmenter ce système unique en plusieurs environnements sécurisés distincts. C'est ce que l'on appelle les machines virtuelles.

LA VIRTUALISATION



Les machines virtuelles

LA VIRTUALISATION ≠ CLOUD COMPUTING

Si le Cloud Computing s'appuie sur la virtualisation, les deux concepts diffèrent :

- la virtualisation permet de désolidariser les environnements informatiques de leurs machines. Comment? => *Chaque machine virtuelle dispose de ses propres ressources allouées, telles que la mémoire, le processeur, le stockage, etc*

- le Cloud Computing consiste à héberger et exploiter des données sur des serveurs distants, par le biais du réseau internet. Comment ? => *Cloud Computing vise à fournir des services informatiques à la demande sur Internet*

LE CLOUD COMPUTING

- Le cloud computing désigne un ensemble de principes et d'approches visant à ***mettre à la disposition d'utilisateurs qui en font la demande,*** quel que soit le réseau, des ressources de calcul, de réseau et de stockage, ainsi que des services, des plateformes et des applications.
- Ces ressources d'infrastructure, services et applications proviennent d'un cloud, c'est-à-dire ***d'un pool de ressources virtuelles orchestrées par des logiciels de gestion et d'automatisation et accessibles à la demande***.

LA VIRTUALISATION ≠ CLOUD COMPUTING

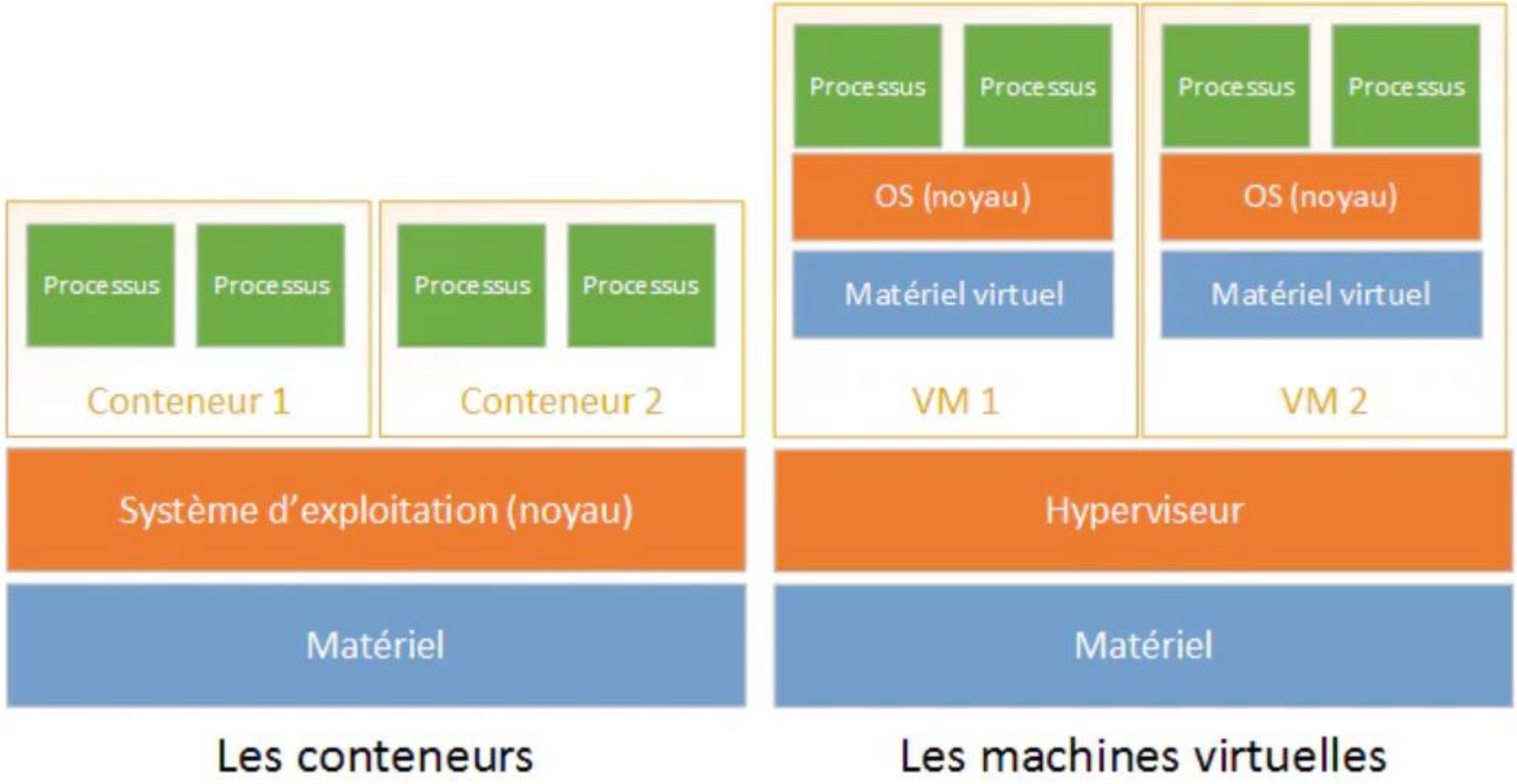
	Virtualisation	**Cloud Computing**
Objet	Créer plusieurs environnements simulés à partir d'un même système physique	Regrouper et automatiser des ressources virtuelles pour une utilisation à la demande
Utilisation	Fournir des ressources en paquets à des utilisateurs spécifiques pour une tâche spécifique	Fournir des ressources variables à des groupes d'utilisateurs pour diverses tâches
Configuration	À partir d'une image	A partir d'un modèle
Durée	Années (long terme)	Heure ou mois (court terme)

MACHINES VIRTUELLES ≠ CONTENEURS

- La notion de conteneur est aussi souvent rapprochée de celle de virtualisation. Mais une fois de plus, des différences existent.
- Comme son nom l'indique, ***une machine virtuelle est l'imitation virtuelle d'un appareil informatique créée, dans le cadre de la virtualisation, à l'aide d'un logiciel hyperviseur, et doté d'un système d'exploitation (ou OS) complet.***

- La virtualisation par conteneurisation, quant à elle, ***consiste à cloisonner directement au niveau du système d'exploitation.*** Ainsi, chaque conteneur exécute son environnement, ***mais partage le même OS hôte***. C'est pour cette raison que les conteneurs servent généralement à la virtualisation d'un programme, et non d'un serveur dans son intégralité

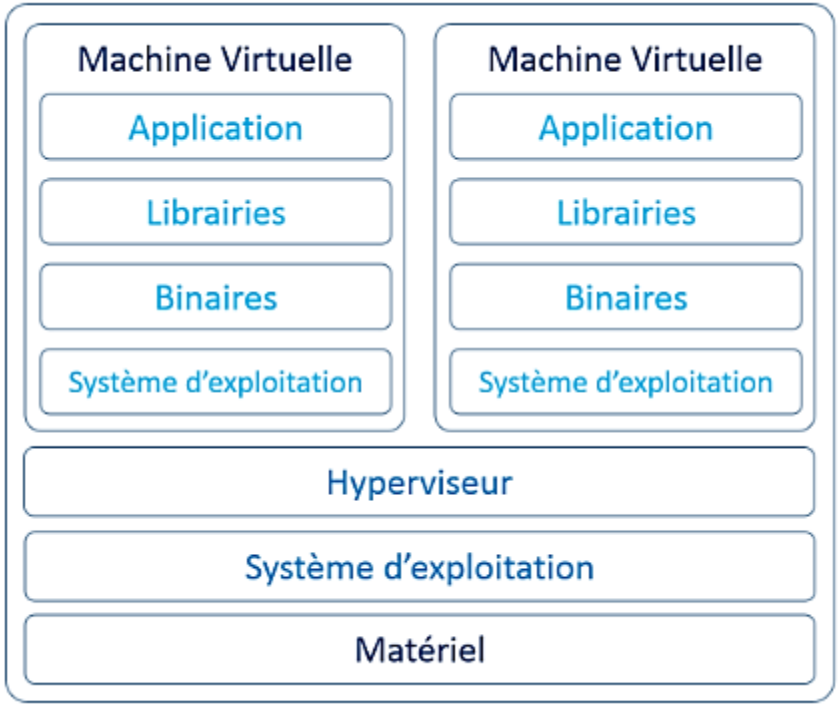
MACHINES VIRTUELLES ≠ CONTENEURS

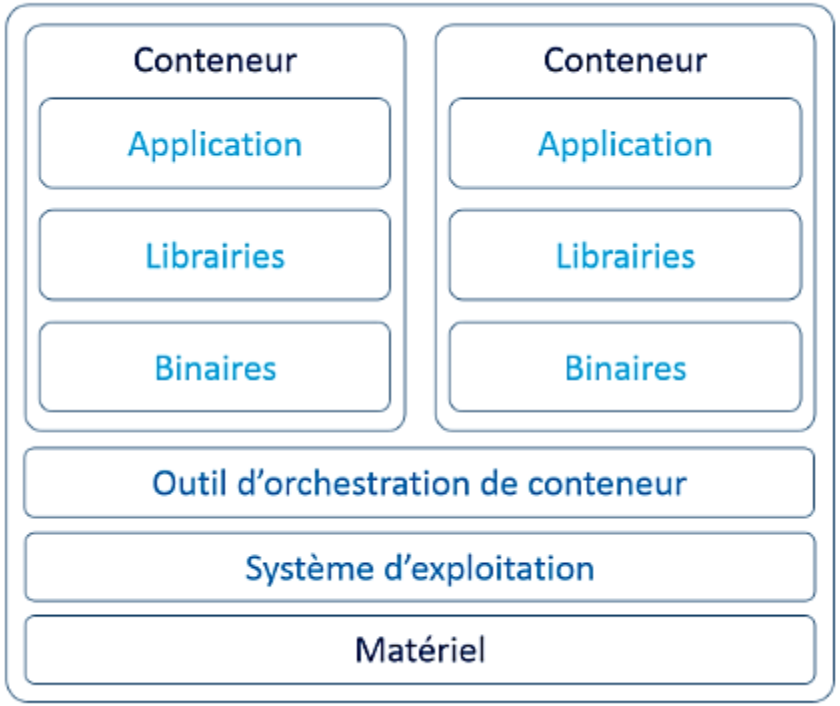


MACHINES VIRTUELLES ≠ CONTENEURS

Machine Virtuelle	Dans le cas de la virtualisation traditionnelle avec des machines virtuelles, chaque machine virtuelle dispose de son propre système d'exploitation. Ainsi, lors de l'exécution d'applications intégrées à des machines virtuelles, l'utilisation de la mémoire peut être supérieure à ce qui est nécessaire et les machines virtuelles peuvent commencer à utiliser les ressources requises par l'hôte.
Conteneur	Contrairement aux applications classiques, les applications conteneurisées partagent un environnement de système d'exploitation (noyau), elles utilisent donc moins de ressources que des machines virtuelles complètes et réduisent la pression sur la mémoire de l'hôte

MACHINES VIRTUELLES ≠ CONTENEURS

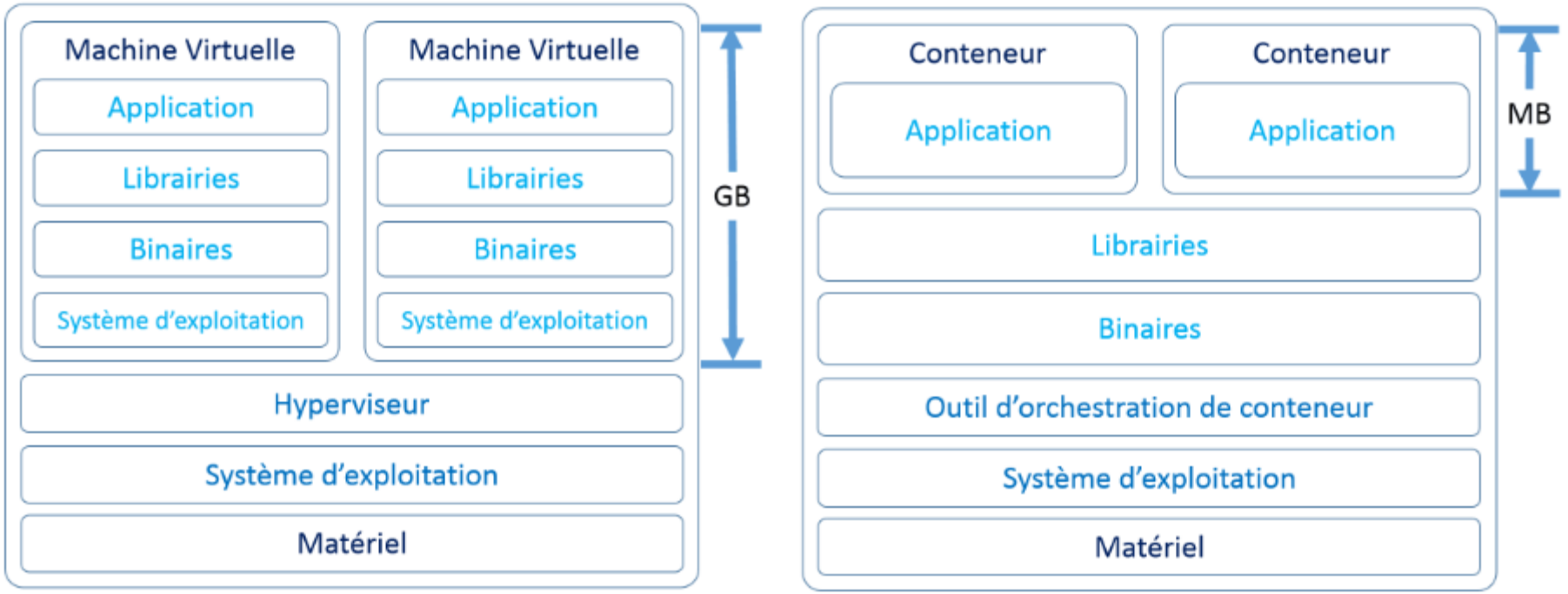




MACHINES VIRTUELLES ≠ CONTENEURS

Machine Virtuelle	Les machines virtuelles traditionnelles peuvent occuper beaucoup d'espace disque: elles contiennent un système d'exploitation complet et les outils associés, en plus de l'application hébergée par la machine virtuelle.
Conteneur	Les conteneurs sont relativement légers: ils ne contiennent que les bibliothèques et les outils nécessaires à l'exécution de l'application conteneurisée. Ils sont donc plus compacts que les machines virtuelles et démarrent plus rapidement.

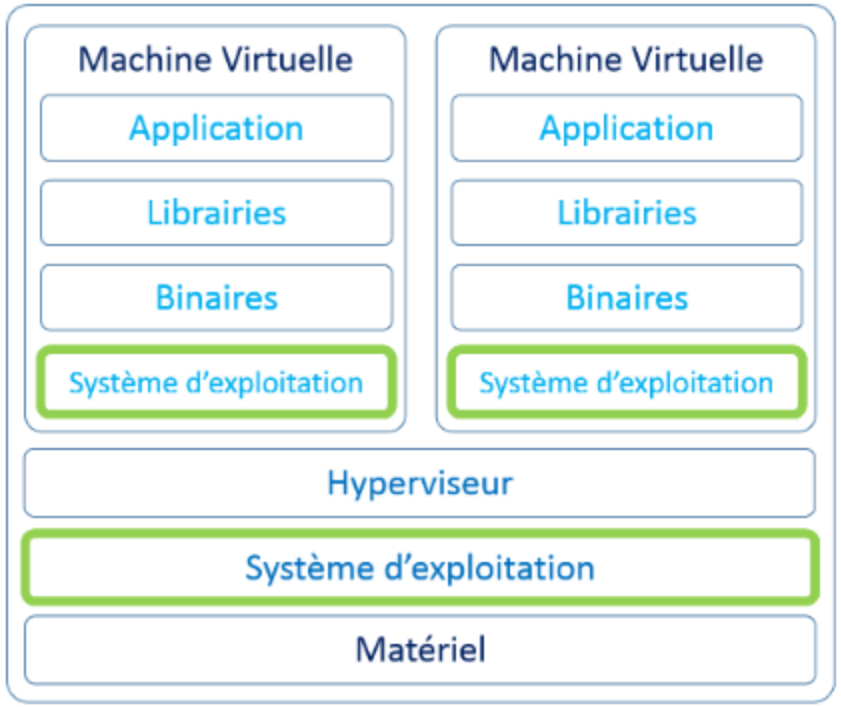
MACHINES VIRTUELLES ≠ CONTENEURS

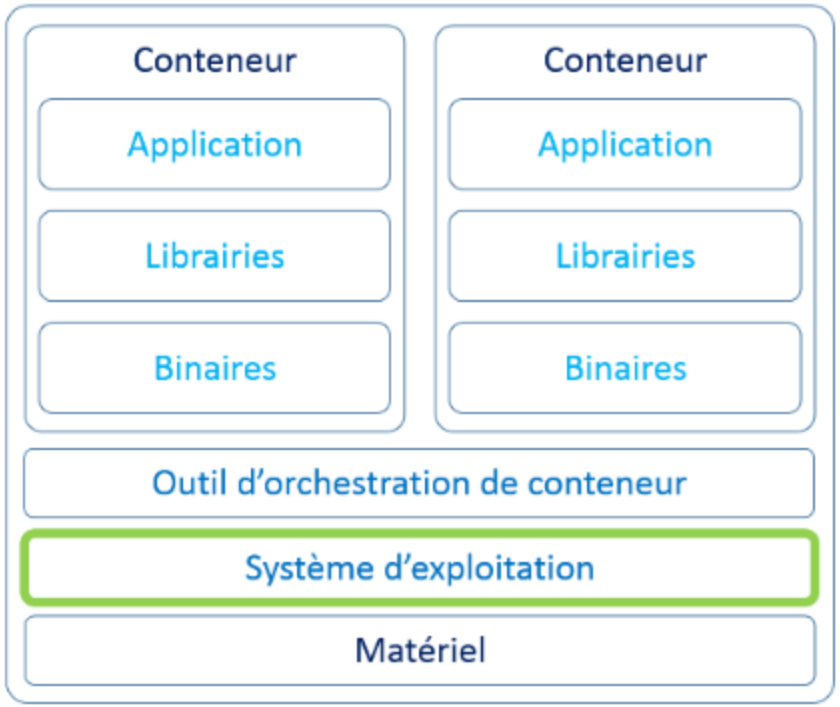


MACHINES VIRTUELLES ≠ CONTENEURS

Machine Virtuelle	En ce qui concerne la mise à jour ou la correction du système d'exploitation, les machines traditionnelles doivent être mises à jour une par une: chaque système d'exploitation invité doit être corrigé séparément.
Conteneur	Avec les conteneurs, seul le système d'exploitation de l'hôte du conteneur (la machine hébergeant les conteneurs) doit être mis à jour. Cela simplifie considérablement la maintenance.

MACHINES VIRTUELLES ≠ CONTENEURS





AVANTAGES DU CONTENEURS

- Meilleures performances
  - Accès direct au matériel
- Démarrage beaucoup plus rapide
  - Pas besoin de démarrer un système complet
- Images plus légères
  - Ne contient que les informations en lien avec l'application
  - Moins coûteux en terme d'espace de stockage
  - Plus rapide à transférer


docker

DEFINITION

Docker est une plateforme de conteneurs ayant largement contribué à la démocratisation de la conteneurisation.
Elle permet de créer facilement des conteneurs et des applications basées sur les conteneurs. Il en existe d'autres, mais celle-ci est la plus utilisée. Elle est par ailleurs plus facile à déployer et à utiliser que ses concurrentes.

Il s'agit d'une plateforme logicielle open source permettant de

- Créer,
- Déployer
- Gérer des containers d'applications virtualisées sur un système d'exploitation.

DEFINITION

Docker est composé de trois éléments :

- le daemon Docker qui s'exécute en arrière-plan et qui s'occupe de gérer les conteneurs (Containerd avec runC)
- une API de type REST qui permet de communiquer avec le daemon
- Le client en CLI (command line interface) : commande docker

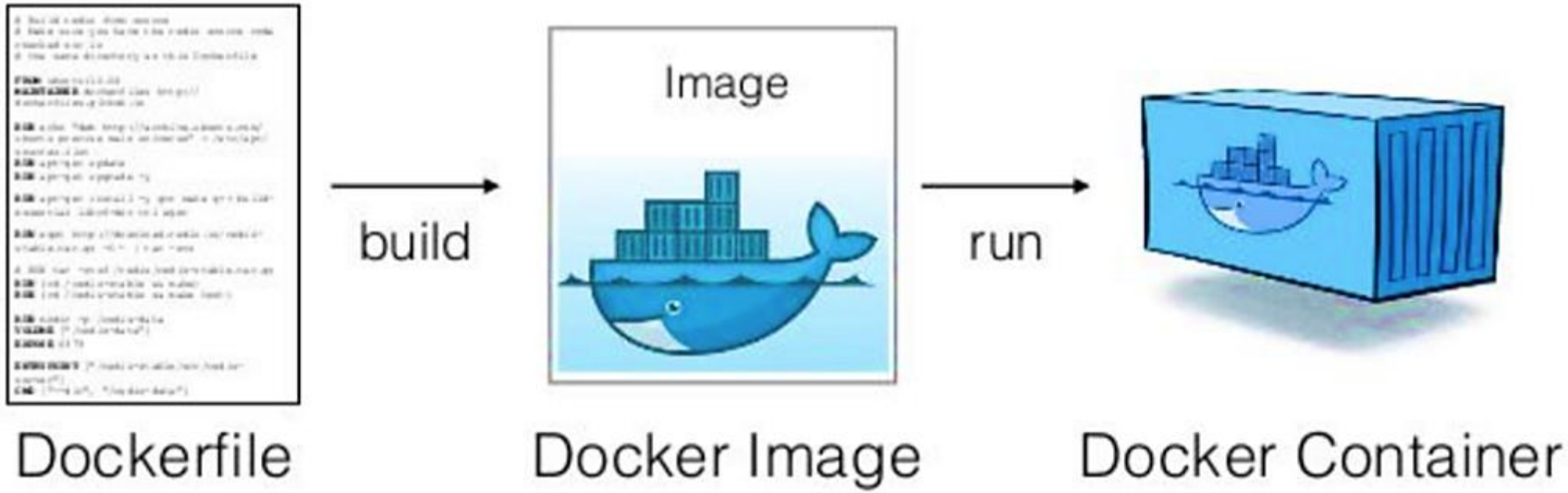


Image
build
run
Dockerfile
Docker Image
Docker Container

Fichier DOCKER

Il s'agit d'un fichier texte simple contenant une collection de commandes ou de procédures. Ces commandes et directives que nous exécutons agissent sur l'image de base configurée pour créer une nouvelle image Docker.

Un Dockerfile est le code source de l'image Docker. Un Dockerfile est un fichier texte contenant diverses instructions et configurations. La commande FROM dans un Dockerfile identifie l'image de base à partir de laquelle vous construisez.

Les images DOCKER

Les images sont des plans en lecture seule qui incluent des instructions de création de conteneurs. Une image Docker est un conteneur créé pour fonctionner sur le framework Docker. Considérez une image comme un plan ou une image de ce qu'il y aura dans un conteneur lorsqu'il sera opérationnel.

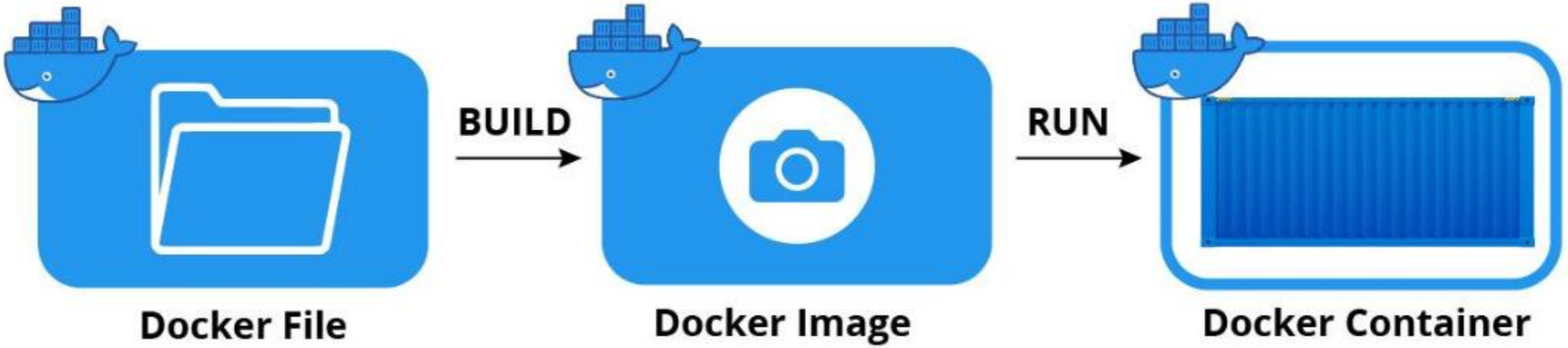
LES CONTENEURS

Les conteneurs sont des environnements d'exécution compacts et virtualisés utilisés pour exécuter des applications. Chaque conteneur est un progiciel* comprenant tous les fichiers de configuration, dépendances, outils système, bibliothèques et code source requis pour exécuter une certaine application. Ils sont distincts de l'hôte et de toute autre instance exécutée sur l'hôte.

*Ensemble de logiciels munis d'une documentation, conçus pour répondre à des besoins spécifiques et permettre une utilisation autonome.

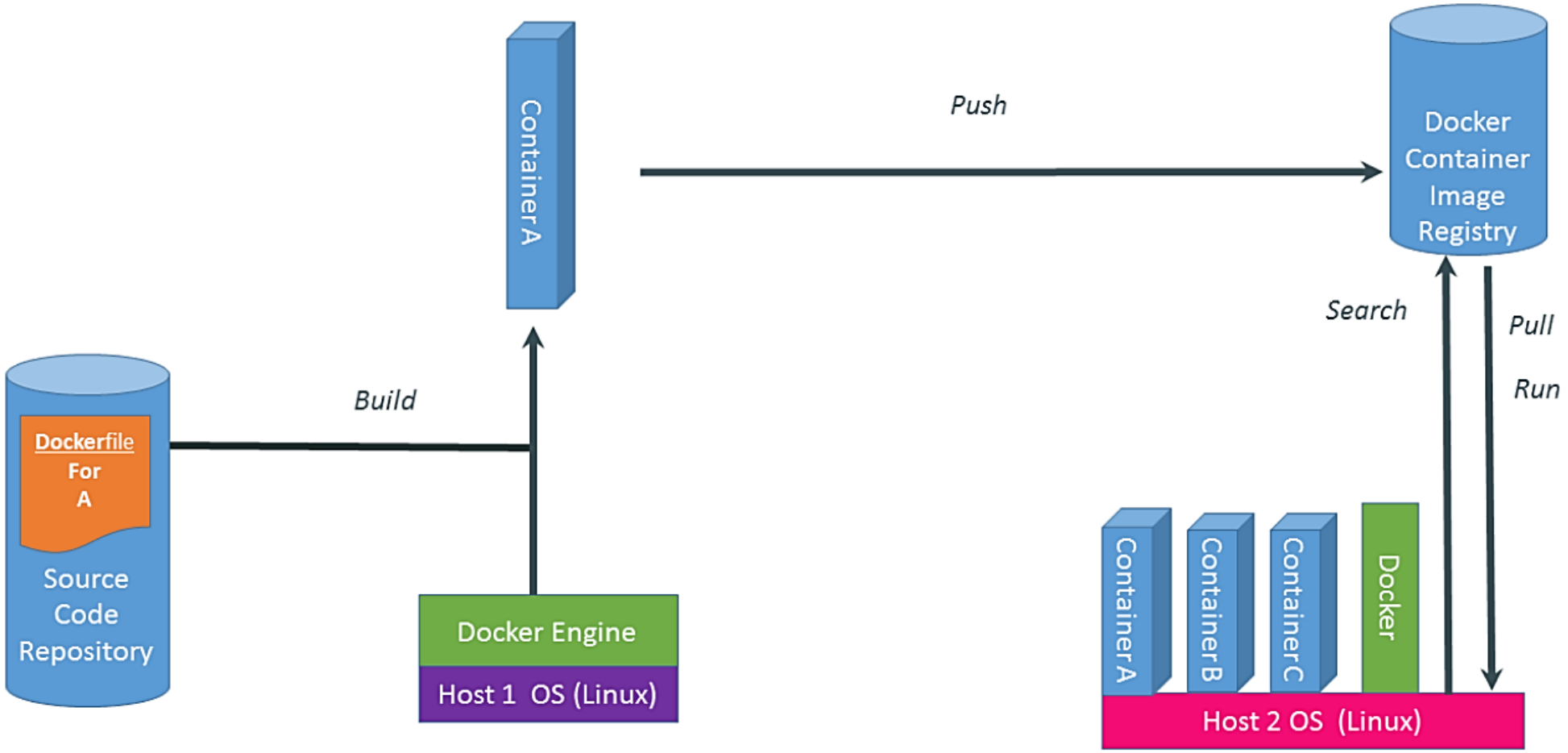
DOCKER HUB

Docker HUB : Dépôt public d’images mises à disposition par Docker



•Créez un fichier Docker et incluez les instructions pour créer votre image Docker.
•Exécutez la commande docker build pour créer votre image Docker.
•Utilisez la commande docker run pour créer des conteneurs maintenant que l'image docker est prête à être utilisée.

PRINCIPES DE FONCTIONNEMENT DE DOCKER



DOCKER: LES BRIQUES PRINCIPALES

Docker engine

Un environnement d'exécution et un ensemble de services pour manipuler des conteneurs docker sur une machine.

Une application client-serveur

Le serveur -- Un daemon (processus persistant) qui gère les conteneurs sur une machine

Le client -- Une interface en ligne de commande

Un/des registres d'images docker

Bibliothèque d'images disponibles

Docker Hub

PRINCIPES DES IMAGES DOCKER

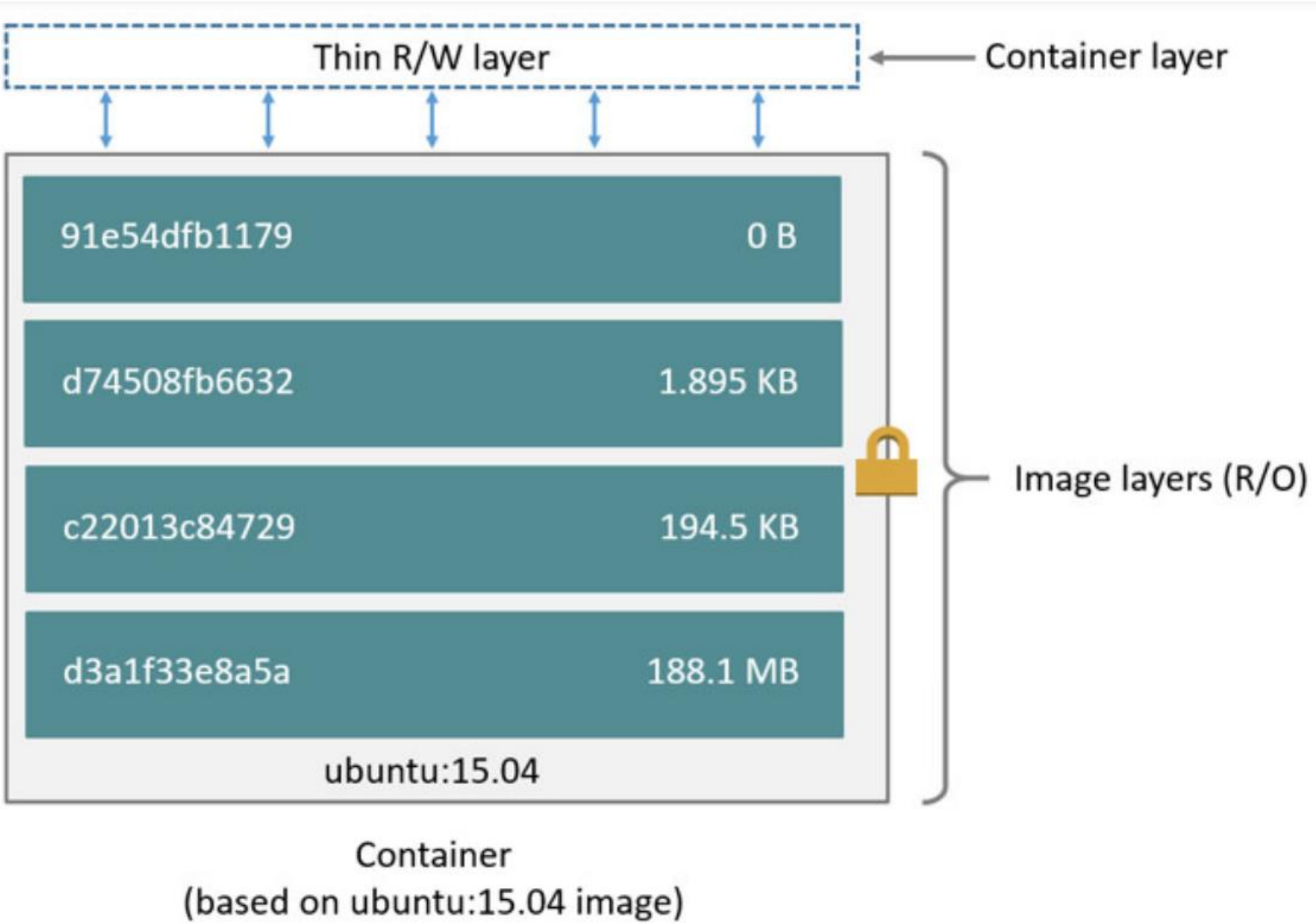
Fondé sur l'utilisation d'un Union File System

- Crée la vision d'un système de fichier cohérent à partir de fichiers/répertoires appartenant à des systèmes de fichiers différents

Un ensemble de couches

- Une image est composée d'un ensemble de couches (layers)
- L'Union File System est utilisé pour combiner ces couches
- Le file system utilisé par défaut s'appelle overlay2
- Chaque couche correspond à une instruction dans le fichier Dockerfile décrivant l'image.

Un système de fichiers Union, dans le contexte de Docker et de la conteneurisation, est une technologie qui permet de combiner plusieurs couches de systèmes de fichiers en une seule vue unifiée.


Thin R/W layer
Container layer
91e54dfb1179
0 B
d74508fb6632
1.895 KB
c22013c84729
194.5 KB
d3a1f33e8a5a
188.1 MB
ubuntu:15.04
Image layers (R/O)
Container
(based on ubuntu:15.04 image)

DOCKER IMAGE LAYERS

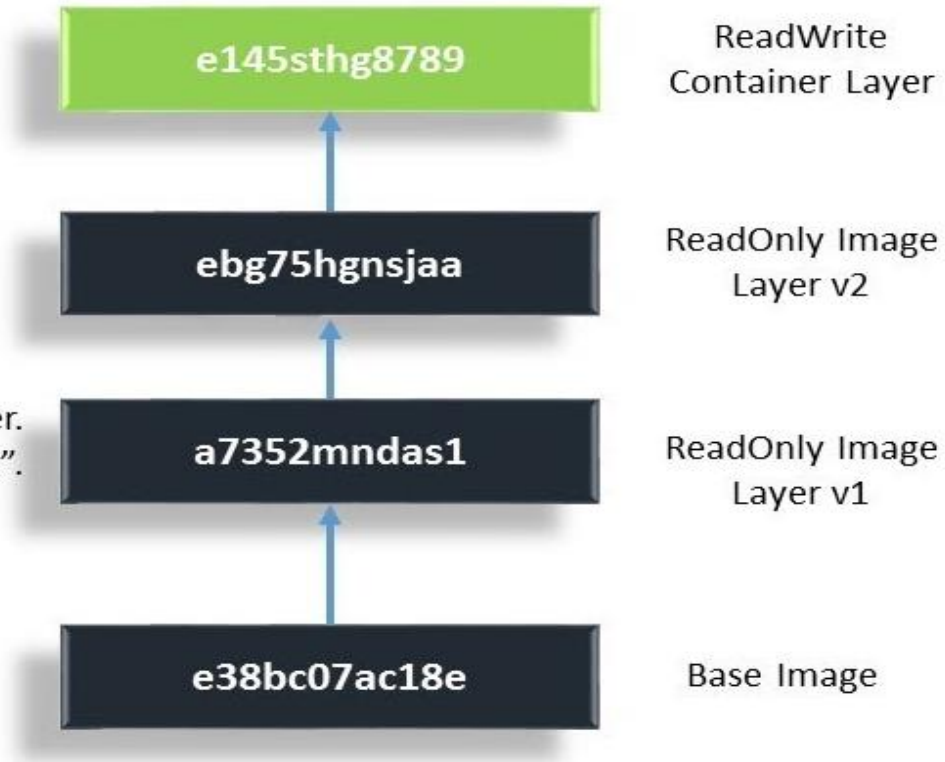
www.learnitguide.net

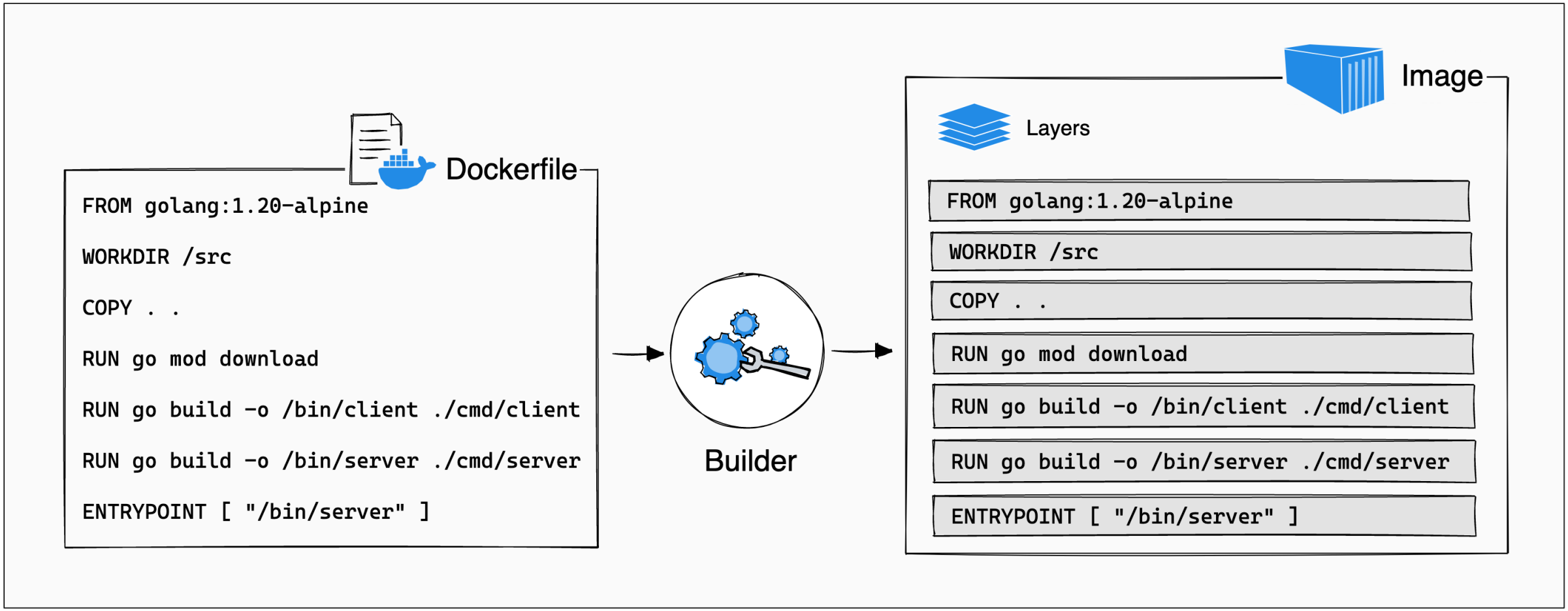
1. Started Image Layer 3 as a container and accessible by users

1. Started Image Layer v1 as a container.
2. Installed and Configured https web server.
3. Committed new layer v2

1. Started Base Image (**docker.io/centos**) as a container.
2. Package Updated on Base Image using “yum update”.
3. Committed new layer v1

Pulled CentOS image from Docker Hub using docker pull command. **Repo: docker.io/centos**




Dockerfile
FROM golang:1.20-alpine
WORKDIR /src
COPY . .
RUN go mod download
RUN go build -o /bin/client ./cmd/client
RUN go build -o /bin/server ./cmd/server
ENTRYPOINT ["/bin/server"]
Builder
Image
Layers
FROM golang:1.20-alpine
WORKDIR /src
COPY . .
RUN go mod download
RUN go build -o /bin/client ./cmd/client
RUN go build -o /bin/server ./cmd/server
ENTRYPOINT ["/bin/server"]

Exemple d'un Dockerfile

Les autres couches

- Les couches correspondent aux différentes modifications qui sont faites pour construire l'image à partir de l'image de base.
- Pour sauvegarder une nouvelle image, il suffit de sauvegarder les nouvelles couches qui ont été créées au dessus de l'image de base.
- Dans un conteneur en cours d'exécution, il existe une couche supplémentaire accessible en écriture
- Toutes les écritures vers le système de fichier faites à l'exécution du conteneur sont stockées dans cette couche.
- Les autres couches, définies au sein de l'image utilisée pour instancier le conteneur, ne sont accessibles qu'en lecture.
- Cette couche est supprimée à la suppression du conteneur

Optimisation des performances

- Partage des couches identiques entre conteneurs diminue l'espace de stockage utilisé par les conteneurs.

Affichage de l'ensemble des couches d'une image

```
$ docker history image_name
```

Avantages liés aux mécanismes de couches

Chaque couche est stockée une seule fois localement

Si certaines couches nécessaires pour une image à télécharger sont déjà présentes, pas besoin de les télécharger à nouveau.

Réduction de l'espace de stockage

Optimisations à l'exécution

Démarrage rapide

Démarrer un conteneur nécessite simplement de créer la couche accessible en écriture

Faible utilisation de l'espace disque

Si plusieurs containers sont instanciés à partir de la même image, ils partagent les couches en lecture seule.

Identification des couches/images

Dans la version courante de Docker :

- Chaque couche est identifiée par un **digest** qui est un hash du contenu de la couche
- Une image est définie par un ensemble de metadonnées (principalement la liste ordonnée des identifiants des couches constituant l'image)
- L'identifiant de l'image est créé en calculant un hash de ces metadonnées

CRÉER DES IMAGES

2 manières

- **Interactivement**
- **En utilisant un Dockerfile**

Avant de commencer: Premiers pas avec docker

L'outil à la ligne de commande pour exécuter des commandes Docker :

```
$ docker
```

```
docker run hello-world
```

Docker : Nous voulons exécuter une commande docker
Run : Commande pour créer et exécuter un conteneur docker
hello-world: Nom de l'image à partir de laquelle est construit le conteneur

docker pull: télécharge une image explicitement
docker run: Commande servant à créer un conteneur à partir d'une image. Télécharge l'image si elle n'est pas présente localement.

Que va-t-il se passer?

L'image à charger est hello-world:latest
Vérification: est ce que l'image est présente localement?
Sinon, télécharger l'image depuis Docker Hub
Charger l'image dans le conteneur et exécuter la commande par défaut définie pour ce conteneur

Construire une image interactivement

Objectifs

Créer une image à partir d'une image de base dans laquelle nous allons installer cowsay

Les étapes

1. Créer un conteneur à partir de l'image de base
2. Installer le logiciel manuellement dans le conteneur et en faire une nouvelle image
3. Jouer avec:
    docker commit
    docker tag
    docker diff

Configuration du conteneur

Démarrer un conteneur Ubuntu

```
$ docker run -it ubuntu
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
ea362f368469: Pull complete
Digest: sha256:b5a61709a9a44284d88fb12e5c48db0409cfad5b69d4ff8224077c57302df9cf
Status: Downloaded newer image for ubuntu:latest
root@f461e2e7afff:/#
```

f461e2e7afff est l'id du conteneur créé

Installer le programme dans le conteneur

```
root@f461e2e7afff:/# apt-get update
root@f461e2e7afff:/# apt-get install cowsay
```

Configuration du conteneur

Quitter la session interactive

```
root@f461e2e7afff:/# exit
```

Inspecter les changements

```
$ docker diff f461e2e7afff
C /var
C /var/log
C /var/log/apt
C /var/log/apt/history.log
A /var/log/apt/term.log
C /var/log/apt/eipp.log.xz
C /var/log/dpkg.log
...
```

C: fichier ou répertoire modifié
A: fichier ou répertoire ajouté

Sauvegarder les changements dans une nouvelle image

```
$ docker commit <yourContainerId>
<newImageId>
```

Exécution de la nouvelle image

```
$ docker run -it <newImageId>
root@7267696dc8c6:/# /usr/games/cowsay bonjour
... ca marche
```

Tagger une image

On peut tagger une image pour lui associer un nom (plus facile à manipuler qu'un identifiant)

```
$ docker tag <newImageId> cowsay
```

Il faut maintenant automatiser le processus!

Les Dockerfile

- Un Dockerfile est une recette décrivant comment construire une image
- Contient une suite d'instructions
- La commande docker build permet de créer une image à partir d'un Dockerfile

Éléments de syntaxe

- FROM: Définit l'image à partir de laquelle la nouvelle image est créée
- LABEL: Associe des meta-données à la nouvelle image (par exemple, l'auteur de l'image)
- RUN: Définie une commande exécutée dans la couche au dessus de l'image courante lors de la construction de l'image
- CMD: Définie la commande exécutée au démarrage du conteneur
- EXPOSE: Informe docker que le conteneur va écouter sur le port réseau défini
- COPY: Copier un fichier/répertoire depuis le contexte de construction de l'image vers la nouvelle couche

Les commandes de Dockerfile ne sont pas sensibles à la casse. On les note en majuscule par convention (facilite la lecture)

Notre premier Dockerfile

1- La création d'un Dockerfile doit se faire dans un nouveau répertoire vide

```
mkdir my_image
```

2- Créer le Dockerfile

```
FROM ubuntu
RUN apt-get update
RUN apt-get -y install cowsay
```

3- Construire notre image

```
$ docker build -t cowsay .
```

-t permet de tagger avec un nom l'image qui va être créée
. indique le contexte de construction de l'image (où se trouve le Dockerfile)

4- Que se passe-t-il?

```
$ docker build -t cowsay .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM ubuntu
 ---> d13c942271d6
Step 2/3 : RUN apt-get update
 ---> Running in 80f5510281d9
Removing intermediate container 80f5510281d9
 ---> cb1643c4393c
Step 3/3 : RUN apt-get -y install cowsay
 ---> Running in 01834650511b
Removing intermediate container 01834650511b
 ---> 9ca55c5ccc54
Successfully built 9ca55c5ccc54
Successfully tagged cowsay:latest
```

Le build context est envoyé vers le démon docker (contenu du répertoir .)
A chaque étape:

- Un conteneur est créé pour exécuter l'étape (Running in ...)
- Les modifications sont committées dans une nouvelle image (---> ...)
- Le conteneur est supprimé
- La nouvelle image est utilisée pour la prochaine étape

5- Construction de l'image

docker build
Crée une image à partir d'un ficher Dockerfile

```
docker build -t docker-whale .
```

Crée l'image docker-whale avec le contexte correspondant au répertoire courant (le '.' est nécessaire)
Par défaut, le Dockerfile est cherché à la racine du contexte

DOCKER-COMPOSE

- On sait créer des images
  - de manière manuelle
  - de manière automatisée
- On sait lancer des conteneurs

PROBLÈMES :

- On veut coordonner des conteneurs
- On veut simplifier la gestion multi-conteneurs

DOCKER-COMPOSE

COMMENT FAIRE POUR

- Gérer les deux conteneurs à la volée?
- Gérer des volumes?
- Gérer des ports différents?
- Me souvenir de ces commandes?

DOCKER-COMPOSE

SOLUTION

- Compose vous permet d'éviter de gérer individuellement des conteneurs qui forment les différents services de votre application.
- Outil qui définit et exécute des applications multi-conteneurs
- Utilise un fichier de configuration dans lequel vous définissez les services de l'application
- A l'aide d'une simple commande, vous contrôlez le cycle de vie de tous les conteneurs qui exécutent les différents services de l'application.

DOCKER-COMPOSE

UTILISATION

- Vous définissez l'environnement de votre application pour qu'il soit possible de la générer de n'importe où
    - à l'aide de Dockerfile
    - à l'aide d'image officielle
- Vous définissez vos services dans un fichier *dockercompose.yml* pour les exécuter et les isoler.
- Exécutez docker-compose qui se chargera d'exécuter l'ensemble de votre application

DOCKER-COMPOSE

Qu'est ce que Docker Compose?

- Un outil complémentaire du Docker Engine
  - présenté dans les versions récentes comme un plugin de Docker Engine
  - Ancienne syntaxe: docker-compose command
  - Nouvelle syntaxe: docker compose command
- Permet de décrire une application construite à base de conteneurs dans un fichier YAML
- Docker permet d'implémenter des micro-services, cela signifie que les conteneurs se limitent souvent à une tâche simple.

DOCKER-COMPOSE

Les principes du Docker-compose

- L'utilisateur décrit son application (multi) conteneurs dans un fichier YAML appelé *dockercompose.yml*
- Exécuter *docker-compose up* pour démarrer l'application

  Compose télécharge automatique les images, les build (si nécessaire), et démarre les conteneurs

  Compose peut configurer des volumes, le réseau, et toutes autres options liées à Docker

Exemple du fichier docker-compose.yml

```
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    depends_on:
      - redis
  redis:
    image: redis
volumes:
  logvolume01:
```

Exemple du fichier docker-compose.yml

- 2 services: web et redis
  - ✓ Par défaut, Compose créé un réseau bridge indépendant du réseau par défaut
  - ✓ La découverte de service fonctionne sur ce réseau

    web peut contacter redis en utilisant son nom
- Un conteneur pour chaque service est créé
  - ✓ Pour web, une image est d'abord recréé à partir du Dockerfile présent dans le répertoire courant
  - ✓ Pour redis, l'image redis est récupérée depuis Docker Hub
- Configuration du réseau
  - ✓ Le port 5000 de la machine hote est associé au port 5000 du conteneur web

Exemple 2 du fichier docker-compose.yml

```
version: '2'
services:
  db:
    image: mysql:5.7
    volumes:
      - "./.data/db:/var/lib/mysql"
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: wordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    links:
```

Exemple 2 du fichier docker-compose.yml :

Démarrage d'une application

```
$ docker-compose up -d
Creating wordpress_db_1
Creating wordpress_wordpress_1
```

Les différents services qui composent mon application ont été démarrés, avec la configuration et l'environnement qui va bien.

Information de mon application

On utilise la commande ps de Compose:

```
$ docker-compose ps
       Name                     Command               State          Ports
-------------------------------------------------------------------------------
wordpress_db_1          docker-entrypoint.sh mysqld      Up      3306/tcp
wordpress_wordpress_1   /entrypoint.sh apache2-for ...   Up      0.0.0.0:8000->80/tcp

$ docker-compose ps wordpress
       Name                     Command               State          Ports
-------------------------------------------------------------------------------
wordpress_wordpress_1   /entrypoint.sh apache2-for ...   Up      0.0.0.0:8000->80/tcp
```

Exemple 2 du fichier docker-compose.yml :

Conteneurs Classiques

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS
8d086aeba614        wordpress:latest    "/entrypoint.sh apach"   7 minutes ago       Up 7 minutes
689405bb755d        mysql:5.7           "docker-entrypoint.sh"   7 minutes ago       Up 7 minutes
```

Les services s'exécutent via des conteneurs sur l'hôte. Les commandes docker classiques sont toujours fonctionnelles.

Passage à l'échelle

On peut passer à l'échelle un service. Autrement dit, on peut augmenter/diminuer le nombre de conteneurs exécutant un service. Par défaut, Compose exécute chaque service avec un conteneur.
On utilise la commande scale pour changer le nombre de réplicas d'un service:

```
$ docker-compose scale db=3
$ docker-compose ps db
    Name                  Command            State    Ports
-------------------------------------------------------------
wordpress_db_1   docker-entrypoint.sh mysqld   Up      3306/tcp
wordpress_db_2   docker-entrypoint.sh mysqld   Up      3306/tcp
wordpress_db_3   docker-entrypoint.sh mysqld   Up      3306/tcp
$ docker-compose scale db=2
```

A RETENIR

- Compose est un outil pour définir, lancer et gérer des services qui sont définis comme une ou plusieurs instances d'un conteneur,
- Compose utilise un fichier de configuration YAML comme définition de l'environnement,
- Avec docker-compose on peut générer des images, lancer et gérer des services, …
- Certaines commandes de docker-compose sont équivalentes à l'outil docker, mais s'appliquent seulement aux conteneurs de la configuration de compose.

LES KUBERNETES

Problématique

Comment gérer des containers qui sont dispersés dans un grand nombre de machines ?
Dans le cas où il y a un container qui pose un problème, comment le détecter ?
Lequel aurait besoin de plus de réplications pour subvenir à une subite augmentation de la charge ?

Solution

L'équipe opérationnelle monitore les serveurs et si un de ces derniers tombe en panne, il devra créer de nouveaux containers avec les services déchus dans un autre serveur.

Solution automatisée

Kubernetes permet justement d'automatiser le déploiement, la gestion de demande de puissance et la gestion des applications containerisées.

LES KUBERNETES

- Kubernetes est un orchestrateur de multiples conteneurs notamment des conteneurs dockers.
- Sa responsabilité est de surveiller les différentes instances de micro containers qu'il supervise et d'en réguler la réplication sur différentes machines en fonction de la montée en charge des requêtes.
- Maintenir la haute disponibilité (maintenir les conteneurs up et assurer l'état des services qui ont été décrits).

LES KUBERNETES

- **Docker** est une technologie d'exécution de conteneurs qui vous permet de ***créer, tester et déployer des applications*** plus rapidement qu'avec des méthodes traditionnelles.
- **Kubernetes** est un outil d'orchestration de conteneurs qui vous permet ***de mettre à l'échelle vos systèmes de conteneurs*** afin que vous puissiez gérer, coordonner et planifier les conteneurs à grande échelle.

Architecture maître-esclave du Kubernetes

- Kubernetes suit l’architecture maître-esclave:
  - Le maître plus communément appelé master existe principalement pour gérer votre cluster Kubernetes.
  - Les esclaves sont quant à eux plus connus sous le nom de workers (on les appellent aussi minions) et ne sont là que pour fournir de la capacité et n'ont pas le pouvoir d'ordonner à une autre nœud ce qu'il peut ou ne peut pas faire.

L’architecture de Kubernetes

- **Volumes (persistent ou non persistent) :**
  - Ils constituent les lieux d’échange entre les pods
  - Persistent dans ce cas on va les stocker à l’extérieur des pods
  - Non persistent à l’intérieur des pods.
- **Déploiements :**
  - Gestion de création/suppression des pods
  - Gestion des nom de réplicas (les réplicas sets)
  - Assurer du respect des descriptions des relations entre conteneurs
- **Namespaces :**
  - Cluster virtuel (ensemble de services) à l’intérieur de KBS
  - Cloisonner à l’intérieur de notre cluster pour des services qui ne travaillent pas ensemble et avoir une cohérence des droits des utilisateurs pour accès aux services qui travaillent ensemble et gérer le cloisonnement si on a des services totalement différents.
  - Segmenter les pods

Architecture maître-esclave du Kubernetes

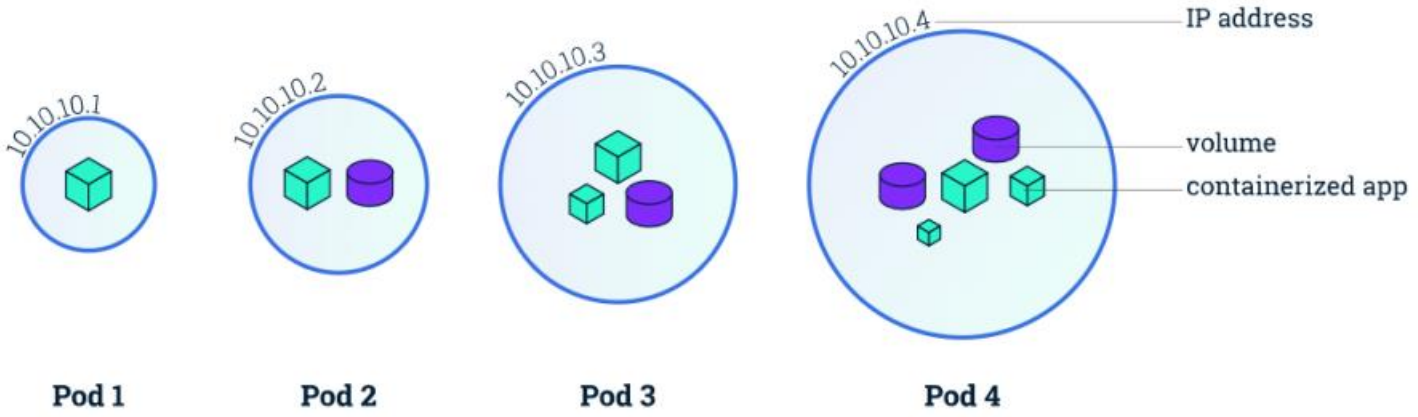
- Un cluster Kubernetes est une forme d'architecture de déploiement Kubernetes. L'architecture de base de Kubernetes existe en deux parties :

  Le plan de contrôle et les nœuds ou machines de calcul.

- Chaque nœud peut être une machine physique ou virtuelle et constitue son propre environnement Linux. Chaque nœud exécute également des pods, composés de conteneurs.

Les notions et concepts de Kubernetes

- **Pods: une instance de KBS : entité de référence de k8s :**
  - Chaque pod contient un fichier « descriptor » qui indique les paramètres d'installation du pod
  - Kubernetes ne fonctionne pas en interagissant directement avec les containers mais avec des pods.
  - Fournir un ensemble cohérent de conteneurs qui peut être un ou plusieurs conteneurs docker ou autre.



Les notions et concepts de Kubernetes

- **Services:**

  Le service se place au dessus des pods ce qui permet de faire une abstraction des pods.

  Le service évite la communication par IP comme dans le cas des docker (IP peut changer puisqu'on travaille avec les conteneurs).

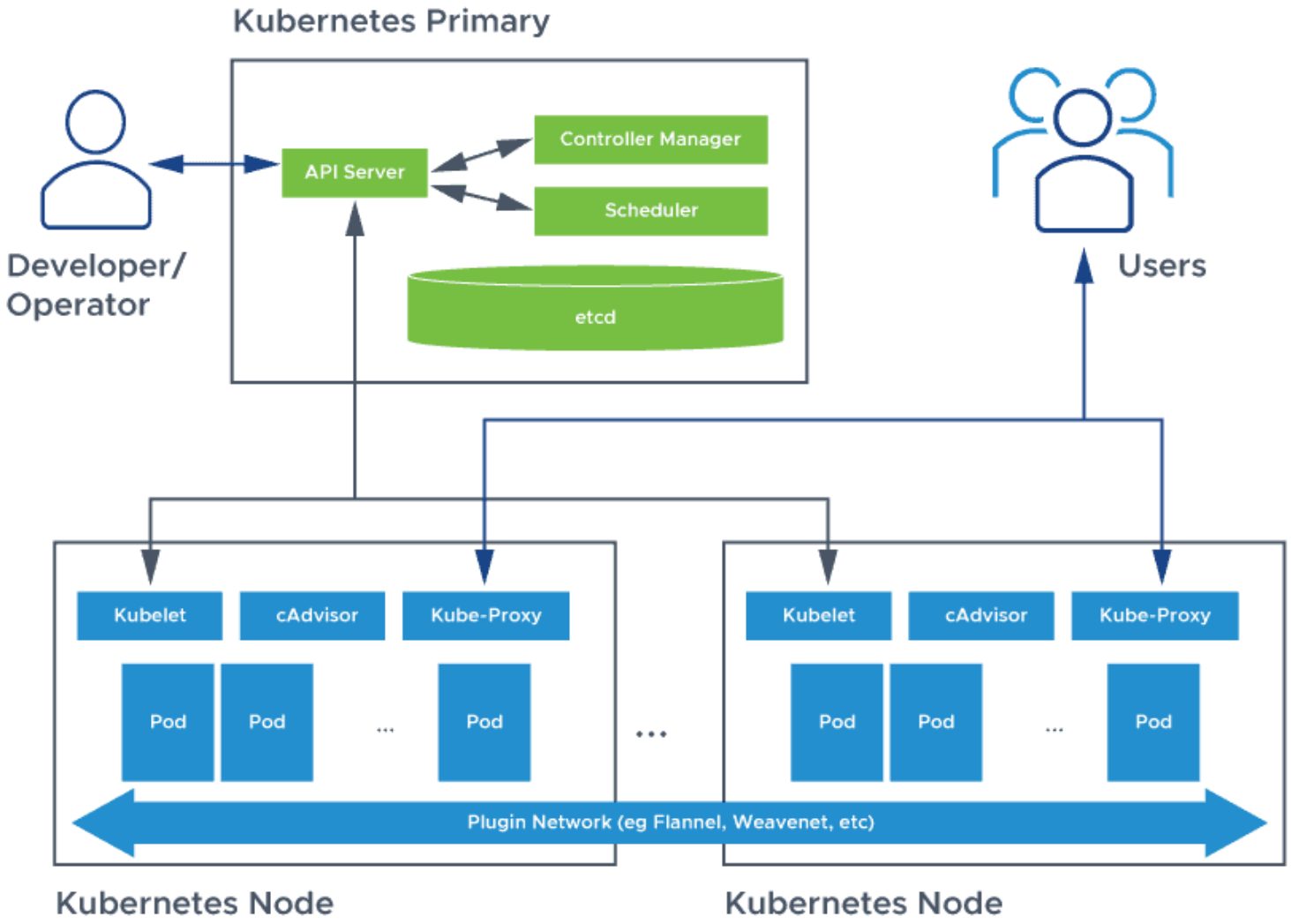
  Un service est compose d'un couple IP + Port fixe , qui permet de communiquer avec des conteneurs

Architecture de kubernetes

Kubernetes obéit à l'architecture maitre/esclave.

- Les composants de kubernetes sont divisés en composants de kubernetes master et ceux de kubenetes worker.
- Kubernetes master est une unité de contrôle qui charge la répartition de la charge de travail sur les pods, et dirige les communications dans le système et contrôle la santé des nœuds.
- Dès qu'il y a une panne dans pod il va arrêter et démarrer dans un autre nœud.
- Le kubernetes master comporte plusieurs composants et dans ce cas on a le choix d'installer ses composants sur une seul machine du cluster ou d'un ensemble de machines du cluster pour permettre la haute disponibilité.

Architecture du Kubernetes



Master Node

Le master node a la responsabilité d'administrer le cluster. Il coordonne les activités telles que la mise en échelle des applications, la maintenance des applications à l'état désiré et la propagation des mises à jour.

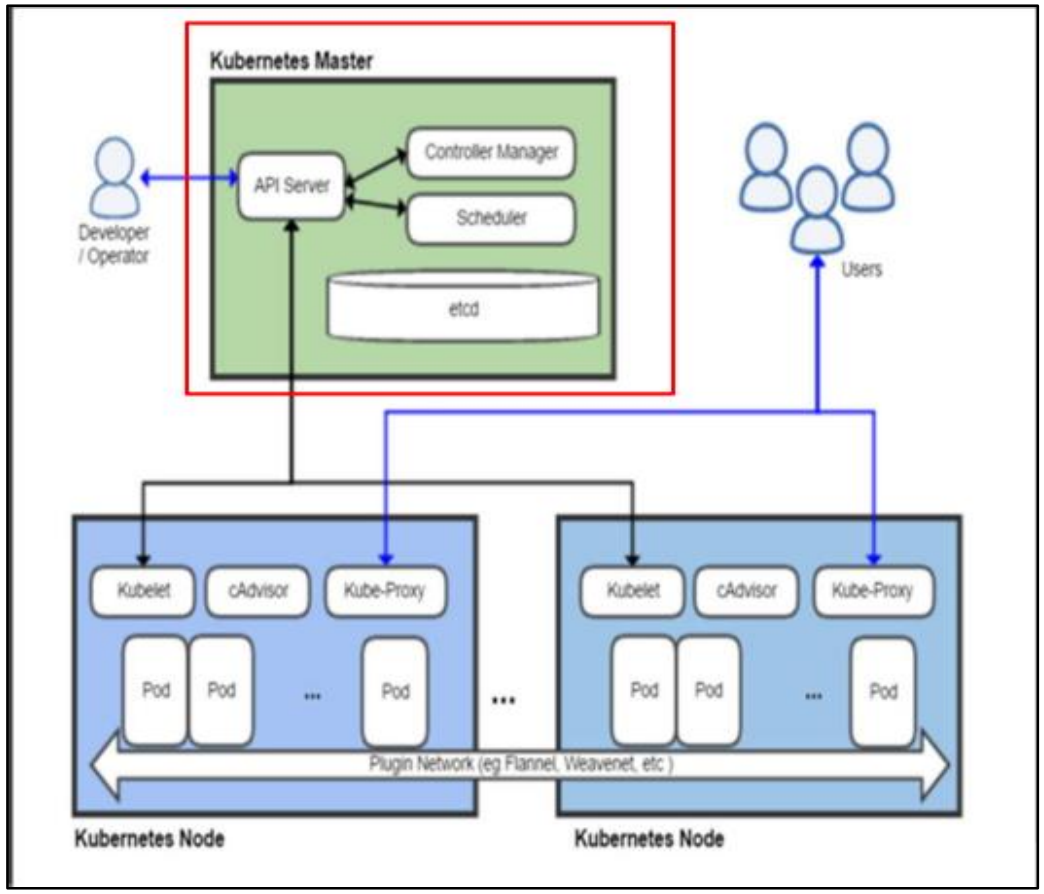
Kubernetes Master

Les composants déployés du plan de contrôle de kubernetes:

• Etcd: SGBD interne distribué et persistant qui stocke l'état du cluster (tous les évènement sui se produisent)

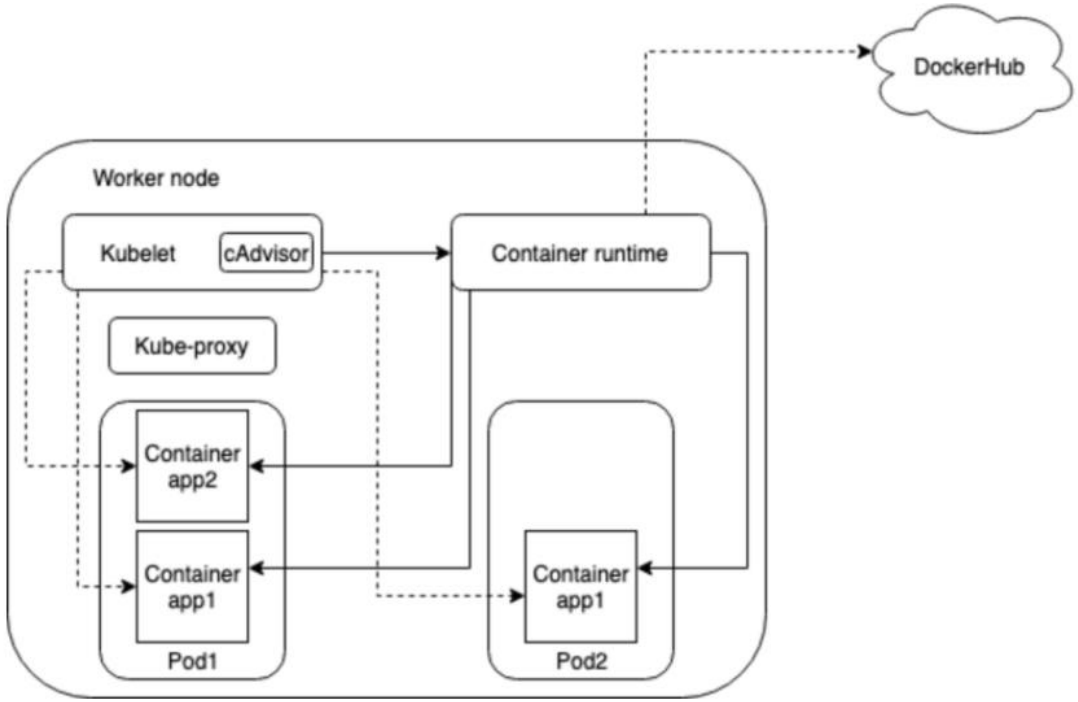
• API Server: conteneur web avec une API REST qui gère la communication avec les composants internes et externes

• Scheduler: l'ordonnanceur qui perme de déterminer la machine la moins chargé pour déployer le pod, donc il doit envoyer toutes les informations sur les nœuds (processus, mémoire…)



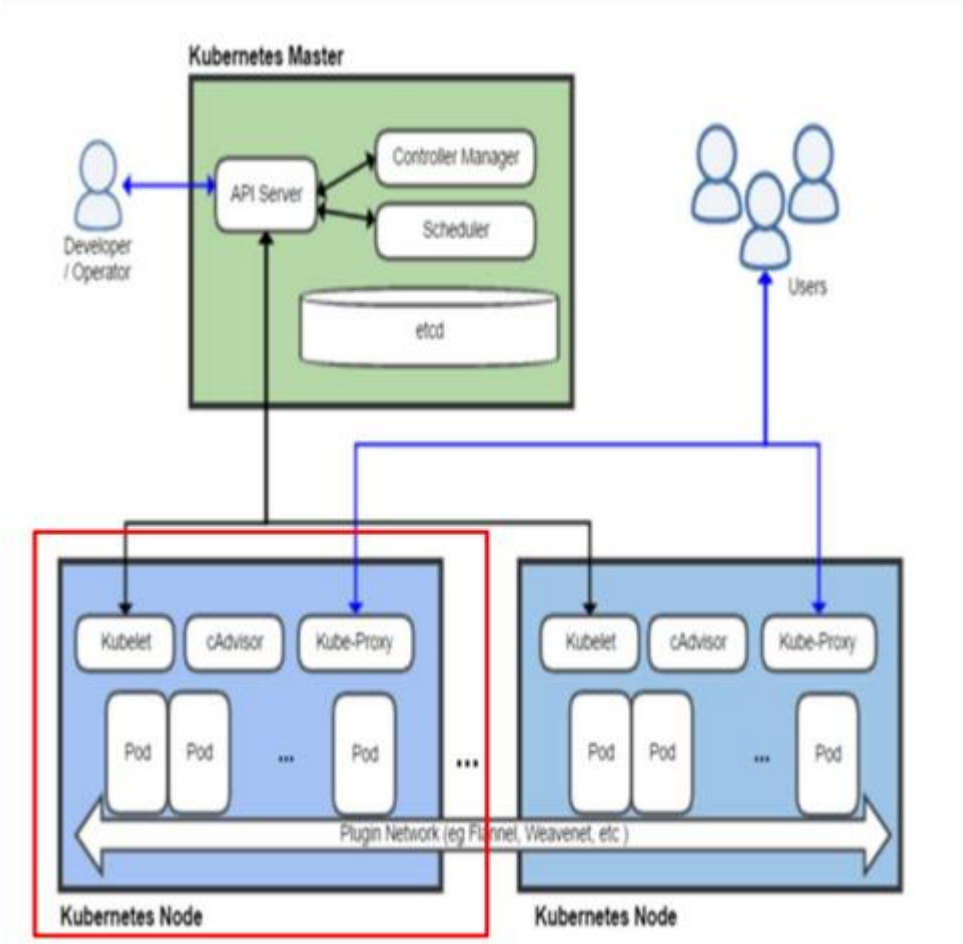
Worker Node

Un worker node (WN) est une machine physique ou une VM qui détient toutes les ressources nécessaires afin de garantir l'exécution d'un ou plusieurs pods. Cette entité va héberger tous les services qu'un développeur aura décidé de déployer.



Kubernetes Node

- Le Node appelé aussi Worker est une machine unique (ou une machine virtuelle) ou des conteneurs (les services sous forme de charges de travail) sont deployés sous forme de pod (les pods regroupent des conteneurs).
- Puisque les nodes contiennent des pods, donc chaque node du cluster doit exécuter le programme de conteneurisation de docker (Docker Engine)



Les Composants d'un Worker Node

Les composants d'un worker node sont responsables du déploiement des pods, et conteneurs, en plus, disposent des composants par default qui sont:

- Kubelet: Responsable de l'état d'exécution du nœud (c'est-à-dire, d'assurer que tous les conteneurs sur un nœud sont bien organisés en Pods et envoyer des informations en permanence au kubernetes master (plan de contrôle).
- Kube-proxy: Responsable d'effectuer le routage du trafic vers le conteneur approprié (sélectionné pour exécuter cette tâche) en se basant sur l'adresse IP et le numéro de port de la requête entrante (service demandé par l'utilisateur)
- cAdvisor: Agent qui surveille et récupère les données de consommation des ressources locales des performances comme le processeur, la mémoire, ainsi que l'utilisation disque et reseau des conteneurs du Node et envoyer ses données au composant scheduler (ordonnanceur) dans le composant master qui va se charger de repartir les charges selon l'état des ressources des nœuds.