

# DEVOPS

Labiba Vinson  
Labiba.vinson@gmail.com

Mars 2024

# Plan

- Concept Devops
  - Avant Devops
  - Devops c'est quoi ?
  - Les Principes
  - Le Fonctionnement
  -
- Cycle de vie
  - Développement continue
  - Integration continue
  - etc
- Avantages
- CI/CD
  - Gitlab CI
  - Jenkins
- Docker
- Cloud
  - Cloud public
  - Cloud privé
  - Cloud communautaire
  - Cloud hybride



# DEVOPS CONCEPT



# Développement sans DevOps

Avant 2000, les logiciels étaient créés, déployés et mis à jour à l'aide du modèle en cascade. Il s'agit d'une approche linéaire du développement à grande échelle

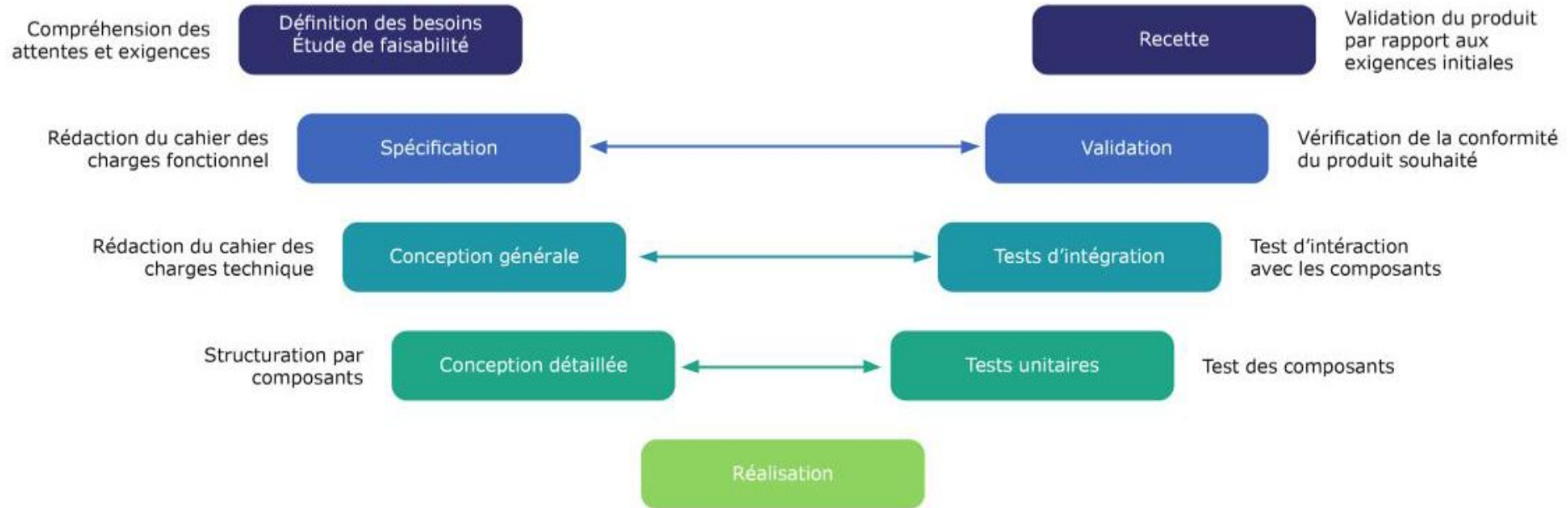
=> Nécessite beaucoup de temps pour la mise en service donc des risques de MEP (Mise En Production) et de commercialiser du produit

Pour accélérer le processus de développement et la qualité, les équipes de développement ont commencé à utiliser une approche itérative.

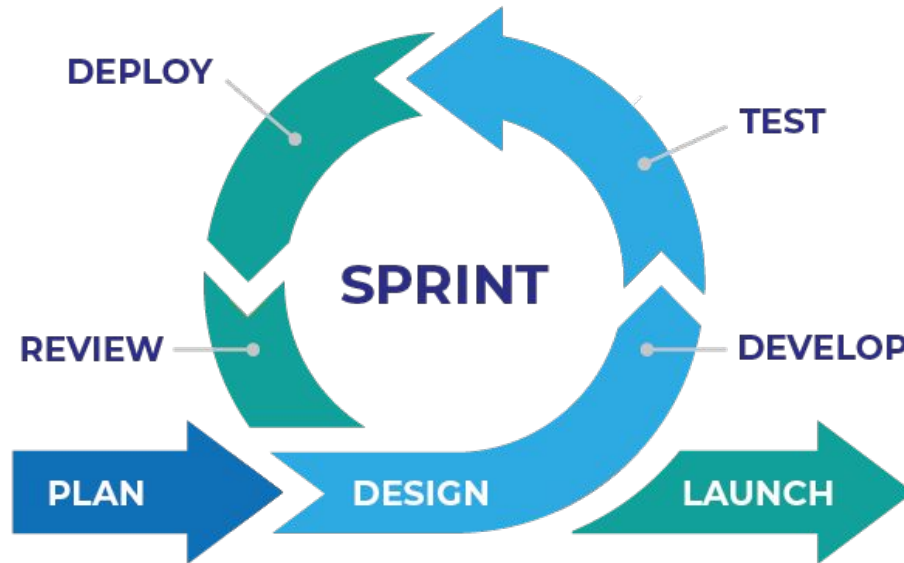
=> La méthode Agile du développement logiciel



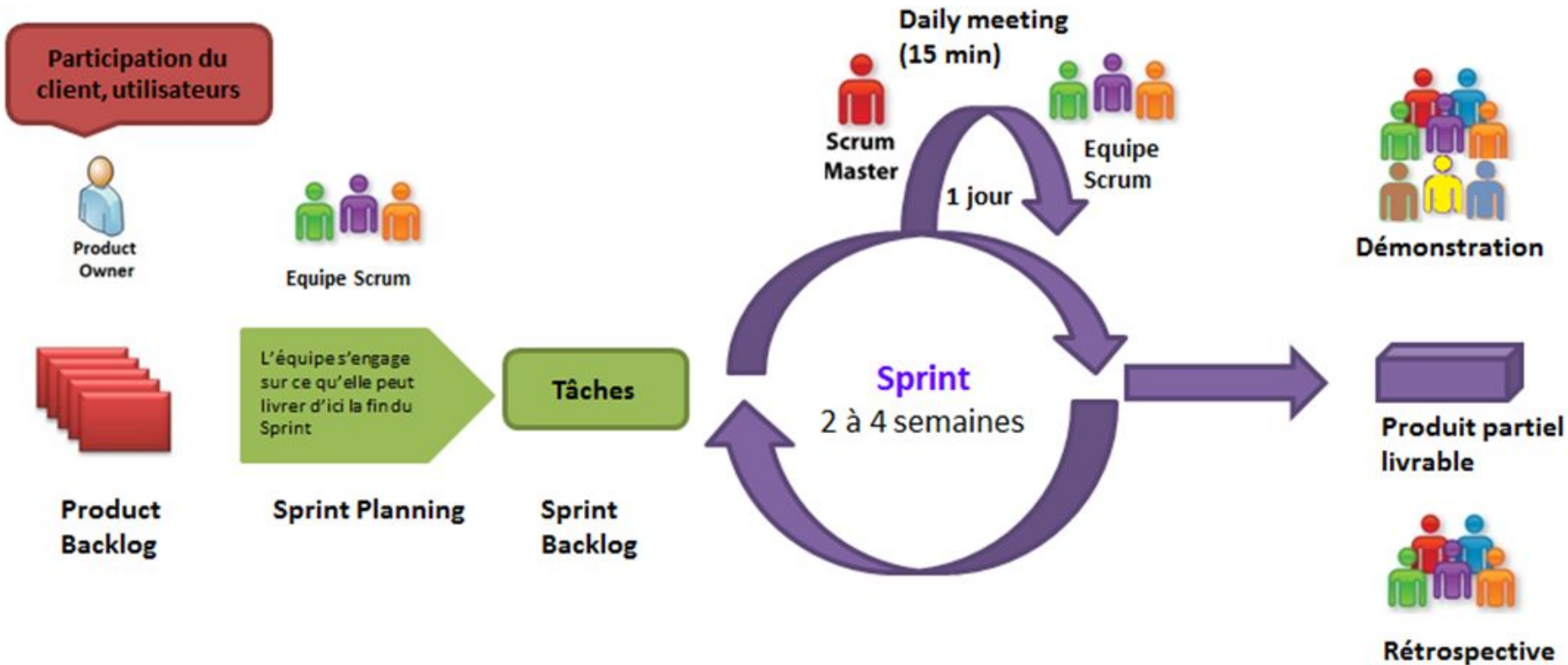
# Cycle en V



# MÉTHODE AGILE



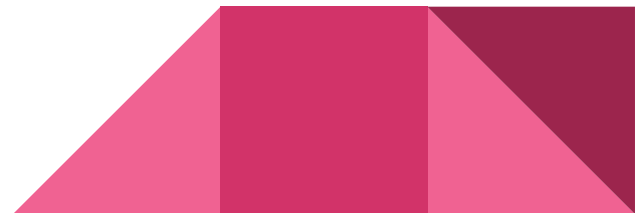
# Méthode Agile



Thème	Cycle en V	Scrum
Cycle de vie	Phases séquentielles	Processus itératif
Livraison	À la fin de la réalisation de toutes les fonctionnalités → livraison tardive	Utilisation partielle du produit suite à la priorisation des besoins → livraison plus rapide
Contrôle Qualité	À la livraison finale (fin du cycle de développement) → effet tunnel	À chaque livraison partielle au client → suivi à court terme
Spécification	Pas de changement possible sans revenir à la phase de spécifications et repasser par toutes les autres phases → délais et coûts supplémentaires	Spécifications plus souples en ajoutant/modifiant les fonctionnalités aux <i>sprints</i> suivants qui n'étaient pas prévues au départ → principal atout de la méthode Agile



Thème	Cycle en V	Scrum
Planification	Plans détaillés basés sur des exigences stables définies dès le début du projet	Planification adaptative et ajustements si nécessaires en fonction des nouvelles demandes
Équipe	Intervention uniquement dans la phase de développement, pas de vision globale du projet	Engagements, échanges et prises de décisions collectives par l'équipe
Documentation	Quantité importante	Strict nécessaire



# DevOps c'est quoi ? (1)

- Une approche combinée du développement logiciel (Dev) et des opérations/exploitations informatiques (Ops),
- Une DevOps culture, un DevOps environnement ou un modèle DevOps, plutôt qu'une méthode, une technologie de développement logiciel pour :
  - Favoriser une meilleure communication et collaboration entre les équipes (développement, opérations, qualité, sécurité) comme une seule équipe,
  - Fournir de façon continue les services/produits de meilleures qualités, plus performants, plus fiables aux clients,
  - Réaliser plus rapidement les nouvelles fonctionnalités et la réduction du « time to market »



# DevOps c'est quoi ? (2)

- DevOps est une combinaison de philosophie culturelle, d'ensemble de pratiques et d'outils permettant de s'intégrer et d'automatiser entre les équipes de développement de logiciels et des opérations informatiques. C'est axé sur le développement rapide de logiciels par des processus, la collaboration, la communication simplifiée et l'automatisation de la technologie.
- DevOps est un ensemble de pratiques visant à réduire le délai d'exécution entre la validation d'une modification d'un système et la mise en production avec une qualité élevée.
- L'implication de l'équipe de sécurité en amont dans le cycle de vie DevOps est appelée DevSecOps,



# DevOps – Principes

Le concept repose sur :

- La création de valeur pour mettre en place d'un cycle d'amélioration continue,
- La culture de collaboration permet de donner une vision globale à tous,
- L'automatisation des phases de provisionning, de test et de déploiement,
- La création des indicateurs de suivi dans tout le processus afin de mesurer la performance des équipes, la satisfaction cliente pour améliorer le produit,
- Le partage un objectif commun et le travail quotidien (problèmes, connaissances, expériences...) favorise une coopération efficace entre les équipes Dev, Sec, Ops



# DevOps - Fonctionnement

Le développement et l'évolution d'une application suit le cycle :

- **Build** correspond à la phase de conception de création de services/applications,
- **Run** correspond à la phase de déploiement et d'exploitation,
- **Change** correspond à la phase d'évolution,
- **Intégration continue** est la phase de vérification de la conformité et de déploiement à chaque modification,
- **Orchestration**, est un processus de gestion automatiquement d'un système,
- **Provisioning**, correspond à un processus d'allocation automatique des ressources



# DEVOPS CYCLE DE VIE



# DevOps – Cycle de vie (1)

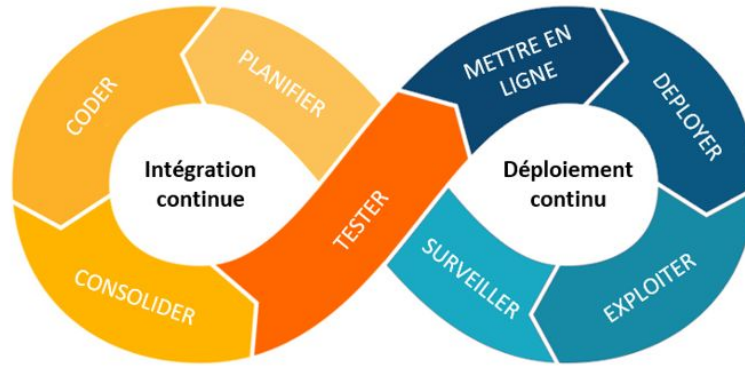
- DevOps Lifecycle (cycle de vie DevOps) est un ensemble de processus de développement itératifs, automatisés et continus conçus pour un processus de développement logiciel rapide avec un niveau de qualité élevée,
- Les outils dédiés sont associés à chaque phase du cycle de vie DevOps pour :
  - Automatiser les processus gérés manuellement,
  - Utiliser également des piles et des outils technologiques modernes pour faire fonctionner une application rapidement avec la fiabilité,



# DevOps – Cycle de vie (2)

Les phases du cycle de vie sont ordonnées, continues, itératives permettent une amélioration tout le long du processus de développement d'une application :

- Développement continu
- Intégration continue
- Tests continus
- Contrôle continu
- Retour terrain continue
- Déploiement continu
- Opérations continues






# DevOps – Cycle de vie (3)

Les étapes du cycle de vie DevOps constituent un cycle allant de l'intégration continue vers le déploiement continu. En automatisant ces étapes avec des outils appropriés, le cycle sera répétable à l'infini.

- **Planifier** (Plan) : cette étape permet de définir les spécifications fonctionnelles et opérationnelles suite à des retours clients ou tout simplement à l'ajout d'une nouvelle fonctionnalité. Il s'agit ensuite de planifier ses sprints.
- **Développer** (Code) : Ensuite, il faut développer la fonctionnalité en utilisant un gestionnaire de version.
- **Intégrer/Compiler** (Integrate/Build) : Le code est ensuite compilé pour obtenir un fichier exécutable à déployer sur différents environnements.
- **Tester** (Test) : Cette étape est cruciale pour détecter d'éventuels bug ou dysfonctionnement, mais aussi pour vous assurer que votre application fonctionne correctement face des pics d'activité (connexions simultanées)
- **Livrer** (Release) : lorsque tous les tests sont réalisés et que le logiciel est stable, le logiciel peut être livré.
- **Déployer** (Deploy) : installer le logiciel dans les différents environnements (pré-prod, prod etc.)
- **Superviser/Monitorer** (Operate & Monitor) : Enfin, cette dernière étape permet de s'assurer que logiciel et l'infrastructure soient opérationnels quelque soit l'environnement.

Le cycle se répète dès que le logiciel a besoin d'une modification ou d'une nouvelle fonctionnalité afin d'assurer de déployer un logiciel rapidement et surtout opérationnel,



# Cycle de vie - Développement continu (1)

Cette première phase correspond à :

- La planification qui ne nécessite aucun outil principal,
- Le codage du logiciel avec certains outils pour maintenir le code et le choix d'un langage de développement DevOps comme Python, C, C++, Ruby, JavaScript... qui font partie de langages DevOps les plus utilisés,
- La définition de la version du projet,
- Le fonctionnement initial basé sur une méthodologie de développement logiciel agile i.e. le développement d'un logiciel dans son ensemble avec les mises à jour sont effectuées en un flux continu dans un grand lot de traitement vs le développement « bloc par bloc » pour être livré dès que les tests de validation sont effectués.



# Cycle de vie - Développement continu (2)

Le développement continu offre des avantages tels que :

- Améliorer la qualité du logiciel final,
- Mettre à jour plus facile des modifications,
- Corriger rapidement les bogues et les erreurs,
- Gérer le risque global du projet,
- Améliorer la productivité,
- Économiser le temps et les ressources.

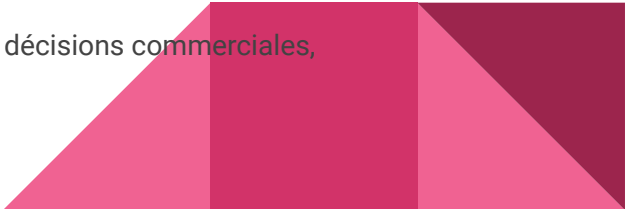


# Cycle de vie – Intégration continue (1)

Le processus d'intégration continue se compose des 4 étapes importantes :

- Contrôle de code source (« Commit Change »)
- Construire (Développer l'application et la vérifier avec les tests unitaires)
- Stagger (Déployer dans l'environnement de test)
- Mettre en production (Déployer dans l'environnement de production)

Des avantages de l'intégration continue :

- Identifier des bogues et des erreurs plus rapidement grâce à des tests fréquents,
  - Fournir des mises à jour à un rythme plus fréquent aux clients,
  - Réduire les risques du projet,
  - Améliorer la boucle de "Retour terrain" en fournissant des données rapidement pour les décisions commerciales,
  - Améliorer la responsabilité globale et la communication entre les équipes.
- 

# Cycle de vie – Intégration continue (2)

Des bonnes pratiques pour l'intégration continue :

- Maintenir le référentiel de code,
- Automatiser la construction,
- Développer axés sur les tests,
- Mettre en œuvre de la demande et de la révision du code,
- Optimiser de la vitesse du pipeline „
- Disposer un environnement de test représentative de la production,
- Maintenir une grande transparence dans le processus,
- Automatiser le déploiement



# Cycle de vie – Tests continus (1)

Le test continu est un processus de tests automatisés à chaque phase du cycle de vie. Il fait partie d'un point de livraison continue de logiciels pour obtenir un retour immédiat sur les risques éventuelles

=> L'automatisation des tests pour accélérer le développement et la livraison des logiciels.

Les tests continus peuvent être réalisés n'importe où (avant ou après) dans la phase d'intégration continue du cycle de vie DevOps

- Tester continuellement l'application pour détecter les bogues,
- Créer un environnement de test iso-fonctionnel à la production utilisant des technologies virtualisation, conteneurisation,
- Réaliser tests automatisés et exploiter des rapports générés destinés au processus d'évaluation des tests permettant d'économiser de temps et d'efforts,



# Cycle de vie – Tests continus (2)

Des avantages des tests continus :

- Obtenir les retours d'information continus, plus rapides et plus fiables,
- Faciliter les modifications mineures du code,
- Avoir un MTTR (Mean Time To Resolution) ou (temps moyen de résolution) plus rapide,
- Changer de version logicielle plus rapide,
- Anticiper la détection précoce des bogues et des erreurs,
- Optimiser le temps nécessaire pour la révision du code,
- Économiser les ressources lors de la mise en œuvre de CI
- Minimiser les risques aux premiers stades de développement.



# Cycle de vie – Surveillance continue

C'est une technologie et un processus mis en œuvre pour surveiller et détecter rapidement les problèmes de conformité et sécuritaire dans l'ensemble en temps réel à chaque phase,

- Surveiller les performances de l'application,
- Enregistrer les informations importantes et vitales liées au environnement de fonctionnement,
- Exploiter les informations collectées pour corriger les erreurs et les dysfonctionnements.





# Cycle de vie – "Retour terrain" continue

C'est une approche permettant de :

- Fournir de nouvelles applications directement au secteur d'activité,
- Ne pas limiter seulement un processus final mais une phase continue du cycle de vie,
- Gérer la coordination entre les équipes (développement, déploiement, le support dans les métiers informatiques et non-informatiques),
- Analyser les améliorations apportées à la surveillance continue,
- Evaluer le résultat de la version finale du logiciel par les développeurs,
- Prendre en compte des retours des testeurs et des clients finaux pour adapter et corriger,
- Mettre en production la nouvelle version du logiciel.



# Cycle de vie – Déploiement continu

C'est un processus de MEP de la version finale validée par des tests automatisés

- La gestion de la configuration est une étape clé pour déployer du code sur tous les serveurs ainsi que la planifications des MAJ,
- Les outils de déploiement peuvent automatiser les processus de déploiement et de libérer et de réallouer les ressources,



# **DEVOPS AVANTAGES**



# Avantages – DevOps

Le concept « DevOps » permet de :

- Réduire le temps de développement, de production,
- Se déployer rapidement et régulièrement,
- Développer des logiciels/services de haute qualité.

Les organisations qui adoptent la culture DevOps :

- Fonctionnent de manière efficace.
- Peuvent livrer avec rapidité, innovation et les dernières fonctionnalités,



# Avantages – Techniques


## **Livraison rapide :**

- Fournir des logiciels/applications/services plus rapidement avec une qualité élevée.
- Construire et de livrer rapidement grâce à des outils automatisés de la phase de livraison continue.

## **Fiabilité :**

- Garantir la qualité de la mise à jour des applications grâce à aux pratiques DevOps telles que CI & CD,
- S'appuyer sur les changements d'infrastructure et maintenir une expérience optimale pour les utilisateurs.

## **Sécurité :**

- Intégrer le niveau de sécurité souhaité dans le cycle de vie DevOps car DevSecOps fait partie intégrante du processus de développement,
  - Effectuer des audits de sécurité actifs avec les fonctionnalités de sécurité qui sont intégrées au logiciel et au produit,
  - Détectent les bogues et les problèmes dans les premières phases du développement à l'aide des tests continus et des commentaires continus
  - Assurer une sécurisation du code suivant les recommandations de sécurité.
- 

# Avantages – Culturels

- **Amélioration de la collaboration**

- Développer une culture collaborative entre les équipes de développement, de sécurité et d'exploitation
- Partager les responsabilités.
- Améliorer l'efficacité des équipes.

- **Anticipation des travaux non planifiés**

- Définir un processus continu afin que l'équipe de développement puisse gérer facilement les travaux non planifiés,
- Permettre de gérer le travail prévu avec le moins impact sur la productivité.

- **Environnement de travail plus heureux et productif**

- Augmenter la collaboration entre les équipes de développement et d'exploitation,
- Faire collaborer en amont dans chaque phase de développement
- Gérer efficacement les tâches complexes.



# Avantages – Commerciaux (1)

## Évolutivité

- Mettre en œuvre la culture DevOps et les meilleures pratiques dans l'organisation,
- Faire évoluer facilement l'organisation,
- Permettre d'utiliser de façon optimale les ressources.

## Libération plus rapide

- Permettre de créer plus de projets logiciels en moins de temps avec les principaux processus de DevOps (automatisation, intégration continue et livraison continue, retour d'information et surveillance ),
- Se concentrer sur la MEP plus rapide du logiciel. Plus de temps pour innover

## Plus de temps pour innover

- Optimiser le temps grâce à une MEP et à un déploiement plus rapides,
- Rationaliser le processus,
- Réaliser une veille technologie pour améliorer la productivité
- Permettre également de résoudre des problèmes réels en mettant en œuvre la combinaison des dernières technologies, frameworks et outils.



# Pipeline DevOps

C'est un ensemble de pratiques que les équipes de développement (Dev) et d'exploitation (Ops) mettent en œuvre pour créer, tester et déployer des logiciels plus rapidement et plus facilement.

L'un des principaux objectifs d'un pipeline est de maintenir le processus de développement logiciel organisé et ciblé. Autrement dit c'est une usine de production d'applications

Les composants d'un pipeline DevOps :

- Intégration continue
- Livraison continue
- Déploiement continu
- Tests continus





# Pipeline - Création

- Configurer un environnement de contrôle de code source
- Configurer un serveur de build
- Réaliser des tests automatisés
- Déployer/Mettre en production de façon automatique



# CI/CD INTRODUCTION



# Contexte

De nos jours, les besoins de développer d'une application notamment d'un service ou d'un produit avec qualité et rapidement en tenant compte des contraintes de délai, de budget, de ressources... sont une préoccupation des entreprises

=> Recherche des solutions.

L'émergence de culture, de concept tels que DevOps puis DevSecOps

Ces concepts apportent :

- Des nouveaux processus de collaboration plus efficace, plus performance entre les équipes

=> Cycle de vie DevOps et DevSecOps,

- Des nouveaux outils associés à chaque phase,
- Des solutions aux besoins des entreprises aujourd'hui.



# Cycle de vie – DevOps/DevSecOps (1)

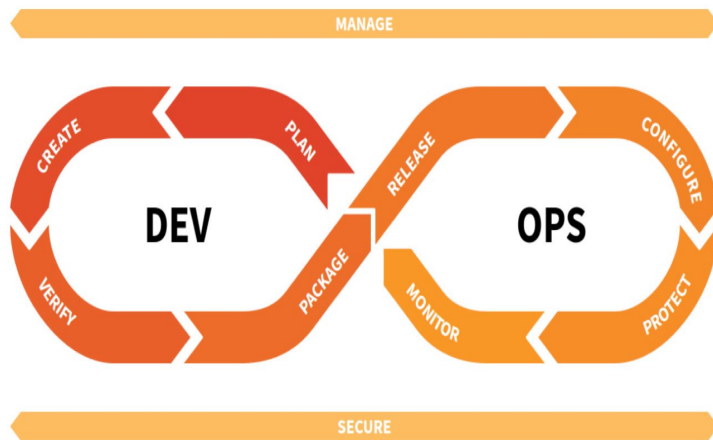
Le nom de chaque phase dans le cycle peut être varié mais le fonctionnement est identique



# Cycle de vie – DevOps/DevSecOps (2)

Le cycle de vie DevOps dans lequel suivent les développements des applications de façon infinie :

- Plan,
- Create,
- Verify,
- Package,
- Release,
- Configure,
- Protect,
- Monitor



Auquel s'ajoute des processus continus

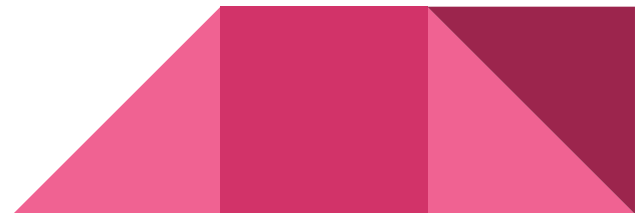
- Manage,
- Secure

=> Pour intégrer le processus sécurité Sec en amont dans le cycle de développement DevOps pour devenir DevSecOps.

# CI/CD

CI/CD (Continuous Intergration/Continous Delivery) font partie du cycle de vie de DevOps et DevSecOps

- Intégrer/Build,
- Tester (unitaires, intégrations, non-régressions...)
- Release,
- Deployer (staging, production...)



# CI

CI (Continuos Integration) ou Intégration Continue :

Un ensemble de processus utilisés pour vérifier le résultat à chaque modification de code source (corrections des dysfonctionnements ou ajout des fonctionnalités) ne perturbe pas le fonctionnement,

Le but est de :

- Détecter en amont les problèmes d'intégrations lors du développement,
- Automatiser le déroulement des tests,
- Observer le développement du logiciel.



# CD

CD (Continuous Delivery) ou Livraison Continue

Une approche pour raccourcir le développement des logiciels et leur mise en production,

Le but est de :

- Construire, tester et livrer rapidement un logiciel,
- Réduire le coût, le temps et les risques liés aux modifications,
- Simplifier et réutiliser les processus de déploiement







GITLAB CI/CD



# GitLab

GitLab est un gestionnaire de dépôt de codes basé sur « git » avec une interface web avec des fonctionnalités :

- Un suivi de code,
- Un suivi de problème,
- Une revue de code,
- Un wiki,
- Un CI/CD dans le processus DevOps,
- ...



# GitLab CI/CD

- GitLab CI/CD est une fonctionnalité de GitLab permet de mettre en œuvre de façon automatique des phases dans le cycle de vie DevOps (builds, tests, déploiements...),
- L'ensemble des tâches peut-être divisé en étapes,
- L'ensemble des tâches et étapes forme une pipeline CI/CD des projets utilisant Git,
- Une tâche est exécutée à l'aide de « runner »,
- Un seul « commit » pour lancer le pipeline GitLab permet de
  - Générer le build de production,
  - Lancer la suite des tests,
  - Déployer la nouvelle version en staging ou en production





**JENKINS**  
**CI/CD**

# Jenkins

Jenkins est un outil Open Source d'automatisation des chaînes de développement et existe depuis 2005 (originellement sous le nom de Hudson).

Il se présente sous la forme d'un serveur autonome et permet l'automatisation de presque toutes les phases décrites en introduction (à l'exception évidemment de la production de code). Cela comprend les phases suivantes :

- le build,
- l'exécution des tests,
- la génération de package - comprenant les phases de création de l'assistant d'installation (logiciel d'interface utilisé pour installer l'application) pour les logiciels qui en possèdent.

Il permet de créer des architectures de productions de gestion de CI/CD efficaces en permettant notamment d'avoir une machine 'maître' distribuant les tâches à plusieurs machines dites 'esclaves'.



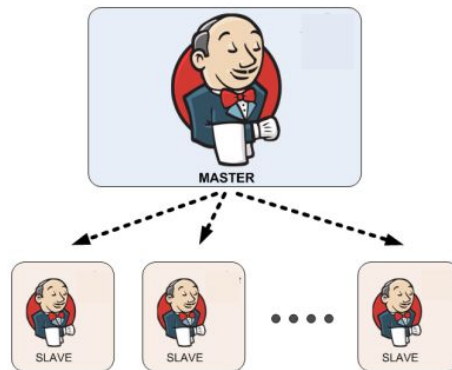
# Jenkins Architecture (1)

Pour un environnement distribué, Jenkins utilise une architecture maître-esclave pour gérer les builds. Ici, le protocole TCP / IP est utilisé pour gérer la communication entre les unités maître et esclave. Le maître et l'esclave Jenkins ont les responsabilités suivantes:

## Maître Jenkins

Le serveur principal Jenkins est le maître qui a les responsabilités suivantes:

- Pour planifier les builds de tâches
- Pour envoyer des builds aux esclaves pour exécution
- Pour surveiller les esclaves en ligne et hors ligne selon les besoins
- Pour enregistrer et présenter les résultats de la construction
- L'instance maître peut également exécuter des tâches de construction directement



# Jenkins Architecture (2)

## Esclave Jenkins

Jenkins Slave est une instance exécutable Java qui est essentiellement une machine distante. Il présente les caractéristiques suivantes:

- Pour entendre les requêtes de l'instance maître Jenkins
- Les esclaves peuvent fonctionner sur une variété de systèmes d'exploitation
- Les esclaves doivent faire le travail selon la direction qui leur est donnée
- Tout projet peut également être exécuté sur une machine particulière ou une machine esclave. Jenkins peut également choisir la prochaine machine esclave disponible pour l'exécution.



# Jenkins Pipeline

## Jenkins Build Pipelines

Les pipelines Jenkins sont utilisés pour vérifier quelle tâche est en cours d'exécution. Habituellement, plusieurs modifications sont effectuées par différents développeurs et il est nécessaire de savoir quelles modifications ont été testées et quelles modifications sont dans la file d'attente. Il existe trois options disponibles lors de la construction d'un projet à l'aide de Jenkins. Ceux-ci sont:

- **Projet Freestyle:** pour offrir une flexibilité maximale, cette option est utilisée pour les versions à usage général.
- **Tâche de multi-configuration:** si vous souhaitez exécuter la même tâche dans différents environnements de test, une option de tâche de multi-configuration est utilisée.
- **Surveiller le Job externe:** pour garder un œil sur les processus non interactifs, cette option est utilisée.





# PIPELINE CONCEPT (1)

## Pipeline

Il s'agit d'un bloc défini par l'utilisateur qui contient tous les processus tels que la construction, le test, le déploiement, etc. C'est le bloc clé pour une syntaxe de pipeline déclarative.

```
pipeline {  
}
```

## Node

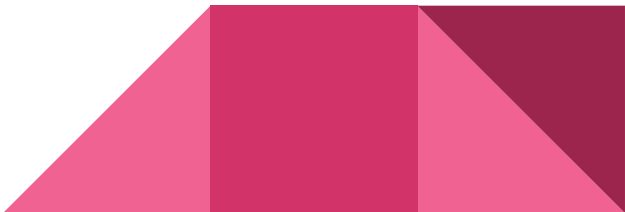
Un nœud est une machine qui exécute un flux de travail complet. Plusieurs sections obligatoires sont communes aux pipelines déclaratifs et scriptés, telles que les étapes et les agents, qui doivent être définies dans le pipeline :

```
node {  
}
```

## Agent

Un agent est une directive qui peut exécuter plusieurs builds avec une seule instance de Jenkins. Cette fonctionnalité permet de répartir la charge de travail entre différents agents et d'exécuter plusieurs projets au sein d'une seule instance Jenkins. Elle demande à Jenkins d'allouer un exécuteur pour les builds.


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



# PIPELINE CONCEPT (2)

Un seul agent peut être spécifié pour l'ensemble d'un pipeline ou des agents spécifiques peuvent être affectés à l'exécution de chaque étape d'un pipeline.

Voici quelques-uns des paramètres utilisés avec les agents :

- *Any* : Exécute le pipeline/la phase sur n'importe quel agent disponible.
  - *None*: Ce paramètre est appliqué à la racine du pipeline et indique qu'il n'y a pas d'agent global pour l'ensemble du pipeline et que chaque étape doit spécifier son propre agent.
  - *Label*: Exécute le pipeline/étape sur l'agent labellisé.
  - *Docker* : Ce paramètre utilise le conteneur Docker comme environnement d'exécution pour le pipeline ou une étape spécifique.
- 

# PIPELINE CONCEPT(2)

## Stages:

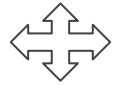
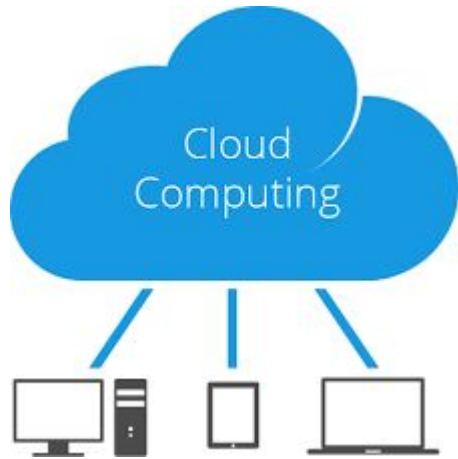
Ce bloc contient tous les travaux à effectuer. Le travail est spécifié sous forme d'étapes. Il peut y avoir plus d'une étape dans cette directive. Chaque étape exécute une tâche spécifique.

## Steps:

Une série d'étapes peut être définie dans un bloc d'étape. Ces étapes sont exécutées dans l'ordre pour réaliser une étape. Une directive steps doit contenir au moins une étape.

```
pipeline {
  agent any
  stages {
    stage ('Build') {
      steps {
        echo 'Running build phase...'
      }
    }
  }
}
```

```
pipeline {
  agent any
  stages {
    stage ('Build') {
      ...
    }
    stage ('Test') {
      ...
    }
    stage ('QA') {
      ...
    }
    stage ('Deploy') {
      ...
    }
    stage ('Monitor') {
      ...
    }
  }
}
```



**DOCKER  
CLOUD**

# Docker

Docker est un logiciel open source qui permet d'automatiser le déploiement d'applications en les empaquetant, ainsi que ses dépendances dans des conteneurs virtuels. La force de ce concept est que ses conteneurs peuvent ensuite être déployés sur n'importe quelle plateforme où peut être installé l'API de Docker.

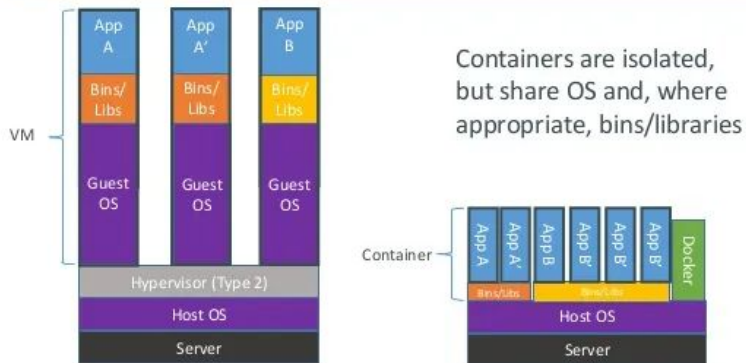
Docker permet non seulement d'exécuter des conteneurs, mais aussi de simplifier leur conception, la gestion des images...,

## Le Concept

Docker s'appuie à la base sur le format de conteneur Linux standard LXC auquel il a adossé une API qui permet d'exécuter des processus de manière isolée. En effet, contrairement aux machines virtuelles classiques qui utilisent un système hôte ("guest OS"), les conteneurs Docker ne contiennent que les applications/librairies en utilisant le système d'exploitation de l'infra.



## Containers vs. VMs



# Pourquoi utiliser Docker ?

Docker possède d'autres points forts, mais en résumé, voici les raisons pour lesquelles on l'utilise :

- On peut exécuter plusieurs applications en utilisant différents OS sur une même machine hôte, sans avoir à créer les systèmes en entier ;
- Les conteneurs peuvent être migrés n'importe où en générant leurs images une seule fois ;
- Docker est multiplateforme, on peut l'exécuter sur Windows, MacOS et Linux, bien sûr ;
- Les langages les plus populaires et les plus utilisés sont pris en charge par lui ;
- Il est léger, rapide, performant, facile à utiliser et moins gourmand en ressources ;
- Les applications conteneurisées sont faciles à déployer, à exécuter, à maintenir et à partager ;
- La gestion des ressources utilisées par les applications est moins fastidieuse ;
- Grâce à la superposition des couches des images de conteneurs Docker, on peut facilement gérer les versions d'une application et par conséquent revenir à une version antérieure lors d'un éventuel dysfonctionnement.



# Docker - Les composants(1)

## Docker Engine

Docker Engine est le système qui fait tourner l'ensemble de Docker. Il s'agit d'un moteur qui suit une architecture client-serveur comprenant principalement trois composants : Docker Daemon, Docker CLI et une API qui sert entre autres d'intermédiaire entre les deux composants précédents.

Ses principaux rôles sont de gérer chaque processus de création, d'exécution des applications dans les conteneurs. Il crée les processus Docker Daemon afin que tous les composants de Docker puissent fonctionner parfaitement.

Pour garantir l'état de Docker à un moment donné, il faut le paramétrer, c'est-à-dire déclarer les conditions requises et les paramètres nécessaires à votre plateforme.

Pour l'installation et la gestion de Docker Engine sur des hôtes virtuels ou sur les anciennes versions des OS supportés par Docker, il faut utiliser un autre composant à savoir Docker Machine.

Plusieurs plug-ins peuvent être installés sur Docker Engine, on retrouve ces derniers sur les Docker Registry.



# Docker - Les composants(2)

## Docker Daemon

Docker Daemon est la partie serveur de Docker Engine. Il agit comme un système d'exploitation, c'est-à-dire qu'il gère les processus et les objets Docker comme les images, les volumes, les conteneurs et les réseaux. Pour ce faire, il écoute les requêtes envoyées par l'API Docker et exécute ce qui est demandé par cette dernière.

## Docker Client

Comme son nom l'indique, il s'agit de la partie cliente de l'architecture globale de Docker, c'est-à-dire la partie où les utilisateurs interagissent avec le démon Docker. C'est lui qui envoie les requêtes de l'API en interprétant les commandes tapées par l'utilisateur afin que Docker Daemon puisse les exécuter. Un Docker Client peut communiquer avec plusieurs Docker Daemon.





# Docker - Les composants(3)

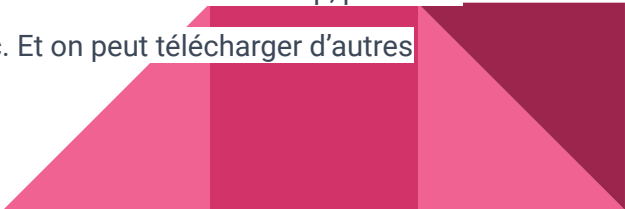
## Docker CLI

Docker CLI est l'abréviation de Docker Command Line Interface. Il s'agit des commandes à exécuter lors de chaque opération que l'on effectue sur Docker. Il existe trois types de CLI :

- Les Docker CLI sont les CLI principaux de Docker. Pour obtenir la liste de toutes les commandes disponibles, il suffit de taper `docker` ou `docker help` sur une invite de commande ;
- Les Compose CLI qui sont les CLI pour interagir avec Docker Compose ;
- Les Daemon CLI sont les CLI permettant de gérer les conteneurs et leurs processus.

## Docker Desktop

Docker Desktop est le logiciel Docker que l'on peut installer sur Windows et MacOS. Il nous permet de pouvoir utiliser Docker en passant par une interface intuitive, d'autant plus que son installation est très facile. Lorsque l'on installe Docker Desktop, plusieurs composants sont directement inclus tels que Docker Engine, Docker Compose, Docker CLI, etc. Et on peut télécharger d'autres composants en se rendant sur les Docker Registries.



# Docker - Les composants(4)


## Docker Registry

Docker Registry est l'emplacement où l'on peut stocker et télécharger des images Docker. Les images stockées sur ces registres disposent d'un système de versionning permettant aux utilisateurs de suivre l'évolution des images en question.

Il existe deux types de Docker Registry : les registres publics et les registres privés. Le registre par défaut de Docker est Docker Hub, qui permet d'accéder à des images, mais également de créer des registres privés. D'autres registres sont également accessibles, comme GitHub, par exemple. Vous pouvez même créer votre propre Docker Registry si vous le souhaitez.

## Docker Hub

Comme nous l'avons vu précédemment, Docker Hub est le registre par défaut de Docker. C'est un hébergeur d'images de conteneurs qui contient plus 100 000 images dont certaines ont été développées et est maintenu par Docker Inc. lui-même. On y retrouve également des images partagées par des développeurs tiers et on peut les télécharger pour nos propres besoins de conteneurisation.



# Docker - Les composants(5)

## Dockerfile

Dockerfile est la base sur laquelle tous les conteneurs sont construits. Il contient l'ensemble des commandes à exécuter afin de construire l'image à savoir le système d'exploitation de base du conteur, les langages utilisés, l'environnement et ses variables, l'emplacement des fichiers, etc. La création de Dockerfile se fait à l'aide des commandes CLI. Le but de ces déclarations est de pouvoir automatiser la création de l'image du conteneur par la suite.

## Docker Image

Une fois le Dockerfile écrit, on peut maintenant créer l'image d'un conteneur. Une image est un modèle à lecture seule pour les conteneurs. On y retrouve tout le code source de l'application à conteneurisé, ses dépendances, les bibliothèques à utiliser ainsi que d'autres outils nécessaires à la création du conteneur.

Elle est composée de plusieurs couches faisant référence aux versions de l'image. C'est-à-dire que lorsque l'on modifie l'image, une nouvelle couche se pose sur les anciennes versions. Donc au final, un Docker image comprend une couche de base à lecture seule et des couches supérieures sur lesquelles on peut écrire.

## Docker Container

Le conteneur, que nous ne cessons d'évoquer depuis le début de cet article, est en réalité une instance d'image sur laquelle on peut interagir. Lorsque l'on exécute une image, cela génère un conteneur. On peut également créer plusieurs conteneurs avec une seule image. Ces derniers peuvent être à leur tour lancés, modifiés, stoppés ou supprimés.

Les paramètres du conteneur ainsi que ses conditions d'exécution sont modifiables selon le besoin de l'utilisateur à l'aide des commandes CLI ou d'API.



# Docker - Les composants(6)

## Docker Volume

Le Docker Volume est l'emplacement où l'on stocke les données utilisées par les conteneurs et celles générées par Docker. Ces données persisteront sur les conteneurs qui y font appel. Cela veut dire que les volumes n'interfèrent pas dans le cycle de vie du conteneur. En plus, on peut les utiliser même si l'on exécute un conteneur Windows ou Linux.

## Docker Compose

Docker Compose est l'outil Docker qui sert à créer et gérer l'architecture d'une application multi conteneurisée, c'est-à-dire une application dont les composants tels que le code, la base de données, etc. sont éparpillés sur plusieurs conteneurs.

Pour effectuer cette tâche, Docker Compose utilise YAML pour spécifier les caractéristiques de l'architecture pour que l'on puisse par la suite exécuter tous les conteneurs avec une seule commande.



# Docker - Les composants(7)

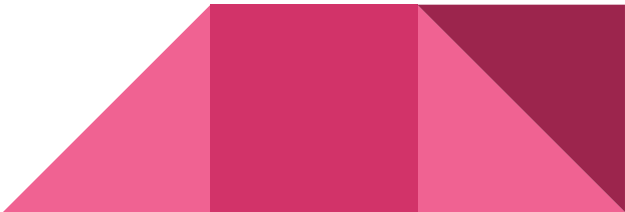
## Docker Swarm

Docker Swarm est un outil d'orchestration pour les conteneurs Docker. Grâce à lui, on peut gérer des conteneurs présents sur les machines hôtes qui composent une architecture distribuée. Ce qui offre une haute disponibilité des applications conteneurisées. Il sert entre autres à gérer les ressources utilisées par chaque nœud de cluster pour que ces derniers fonctionnent correctement.

Docker Swarm est préinstallé avec le logiciel Docker, c'est-à-dire que l'on peut directement l'utiliser une fois que ce dernier est installé sur tous les hôtes. Il agit également comme un équilibreur de charge sur l'ensemble de l'architecture.

## Kubernetes

Kubernetes n'est pas un composant Docker à proprement parler, mais se présente comme une alternative à Docker Compose et Docker Swarm. Il s'agit d'un logiciel open source permettant d'orchestrer des conteneurs. Il permet d'automatiser des tâches relatives aux conteneurs tels que leur déploiement, leur mise à jour, la surveillance d'état ou encore le rééquilibrage des charges.



# Docker - Les commandes

`docker -- version` : Connaître la version de Docker installé :

`docker run -d -p 80:80 docker/getting-started` : Lancer un conteneur

`docker ps -a` : Lister tous les conteneurs

`docker images` : Obtenir toutes les images présentes sur le système

`docker build` Construire une image à partir d'un Dockerfile

`docker rm ID du conteneur` Supprimer un conteneur

`docker rmi ID de l'image` Supprimer une image

`docker logs --details` Afficher les logs d'un conteneur avec leurs détails



# Cloud (1)

La définition proposée par le NIST (National Institute of Standards and Technology - [www.nist.org](http://www.nist.org)) :

- Modèle d'accès réseau, pratique et à la demande, à une réserve partagée de ressources informatiques configurables (réseaux, serveurs, stockage, applications, services...) mises à disposition et libérées rapidement ; nécessitant un effort de gestion et des interactions minimales avec le fournisseur de service

Sur le plan pratique :

- Une ressource (machine) mise à disposition sans savoir ce que le client va faire et le type d'OS



# Cloud (2)

- Le concept, le développement et le modèle économique du cloud est inspiré du modèle des services publics comme le gaz, l'électricité ou l'eau :
  - La mise à disposition des ressources pour les clients par le fournisseur,
  - L'envoi de facture à la fin du mois en fonction des consommations, le cout d'abonnement et de mise en service
- Le principe de services proposés
  - L'utilisation à la demande,
  - Le paiement à l'usage,





# Cloud (3)

- Des ressources évolutives en toute transparence...
  - L'infrastructure s'adapte de manière transparente aux variations de charge de travail en fonction des besoins de l'utilisateur,
- Des ressources informatiques matérielles et logicielles,
  - Les serveurs physiques, la capacité de stockage, les bases de données, les équipements réseaux, les équipements de sécurité, les applications...
- Une réserve de ressources a priori presque infinie...
  - Une évolutivité extrême qui semble sans limites,
  - Une optimisation des taux d'utilisation des ressources basée sur la mutualisation des ressources



# Cloud (4)

- Une offre de service optimisée
  - La mise à disposition des services correspond aux besoin réels de l'utilisateur,
  - La facturation est basée sur les services utilisés
- Une gestion autonome de l'utilisateur basé sur le principe du self-service
  - Les nouvelles ressources peuvent être immédiatement ajoutées ou supprimées directement par l'utilisateur



# Cloud – Principaux fournisseurs

AWS

Microsoft Azure

GCP

OVH,



# Clouds publics

- ❖ L'infrastructure et les services sont proposés à un grand nombre de clients : Offres souvent grand public
- ❖ Gestion assurée par un éditeur tiers qui : Héberge, exploite et gère les services et Gère la sécurité
- ❖ Avantages :
  - Mise à disposition dynamique des ressources, sur la base du self- service et d'une facturation selon la durée d'utilisation
  - Minimise les coûts grâce à une utilisation optimisée des ressources
    - Un ordinateur peut héberger les machines virtuelles de plusieurs entreprises
    - Concept de multitenancy (« mutualisation » en français)
- ❖ Inconvénients
  - Le client n'a plus le contrôle total de la sécurité, de la gouvernance et de la fiabilité
    - Obligation de faire confiance au fournisseur de services
  - Nécessite parfois des compromis pour avoir satisfaction

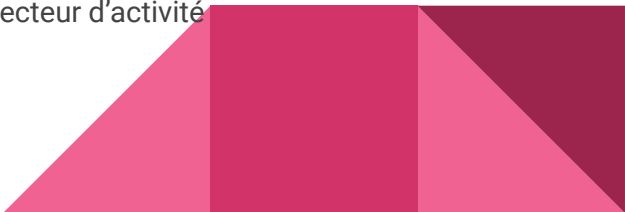
# Clouds privés

- ❖ L'infrastructure et les services sont réservés à une seule entreprise
  - Similaire à un cloud public, mais sur des réseaux privés
  - Pas de partage avec d'autres entreprises, comme sur un cloud public
- ❖ Gestion assurée par l'entreprise elle-même, ou par un tiers : Sur place ou hors site
- ❖ Avantages :
  - Minimise les inconvénients inhérents aux clouds publics
    - Sécurité des données
    - Gouvernance par l'entreprise et Fiabilité
  - Plus facile de respecter les normes de sécurité et les stratégies de l'entreprise, ainsi que les exigences en termes de réglementation
- ❖ Inconvénients :
  - Nécessite de disposer des compétences pour configurer, maintenir à jour et gérer le cloud
    - En interne ou en sous-traitance



# Clouds communautaires

- ❖ L'infrastructure et les services sont partagés par plusieurs entreprises
  - Les entreprises impliquées doivent avoir des besoins similaires
  - Déploiement souvent destiné à des secteurs d'activité particuliers
    - Service public, universités, etc.
- ❖ Gestion assurée par les entreprises ou par un tiers.
- ❖ Avantages:
  - Permet de profiter des ressources et des services requis par des activités similaires
  - Réalise des économies grâce au partage de l'infrastructure et des services
    - Alternative : chaque entreprise peut répliquer l'infrastructure en interne
  - Met en œuvre plus facile des normes en vigueur dans des entreprises ou le secteur d'activité
    - Déploiement des mises à jour centralisé



# Clouds hybrides

Mélange au moins deux modèles de clouds public, privé, communautaire)

- Permet d'exploiter les caractéristiques des différents types de clouds pour répondre plus efficacement aux besoins de l'entreprise
- Les applications cœur de métier et les données sensibles sont hébergées en interne, sur un cloud privé
- Le cloud public permet de traiter l'excédent de sollicitations du cloud privé
  - Technique du « cloud bursting »



Secure DevOps

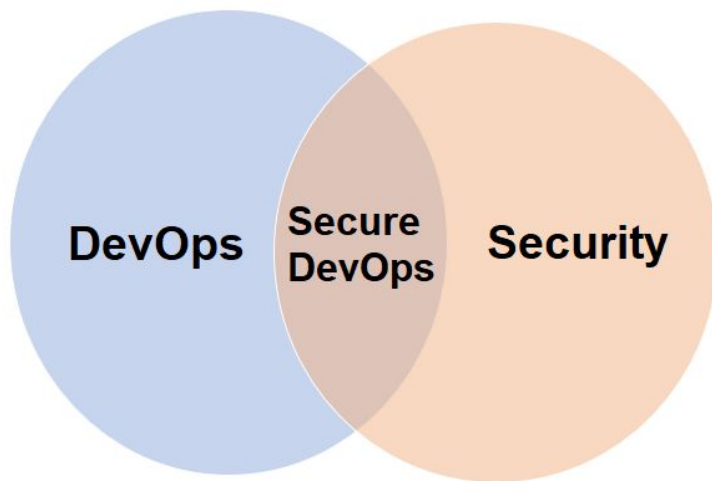
( DevSecOps)





# Introduction

- *Secure DevOps* intègre DevOps à la sécurité dans un ensemble de pratiques conçues pour atteindre efficacement les objectifs de DevOps et de sécurité.
- Un pipeline *Secure DevOps* permet aux équipes de développement de travailler rapidement sans compromettre leur projet en introduisant des failles de sécurité indésirables.

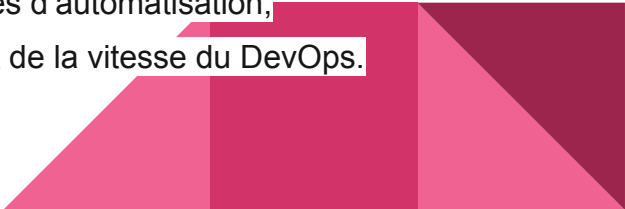


# La sécurité dans le contexte de Secure DevOps

La sécurité s'exerçait généralement sur un cycle plus lent et impliquait des méthodologies de sécurité traditionnelles, telles que :

- Contrôle d'accès.
- Renforcement de l'environnement.
- Protection du périmètre.

Secure DevOps :

- Inclut ces méthodologies de sécurité traditionnelles et bien plus encore. Avec Secure DevOps, la sécurité concerne la sécurisation du pipeline.
  - Consiste à déterminer où ajouter la protection des éléments qui se connectent à vos pipelines de build et de mise en production.
  - Peut vous montrer comment et où vous pouvez ajouter la sécurité à vos pratiques d'automatisation, environnements de production et autres éléments de pipeline tout en bénéficiant de la vitesse du DevOps.
- 

# La sécurité dans le contexte de Secure DevOps

Secure DevOps répond à des questions plus larges, telles que :

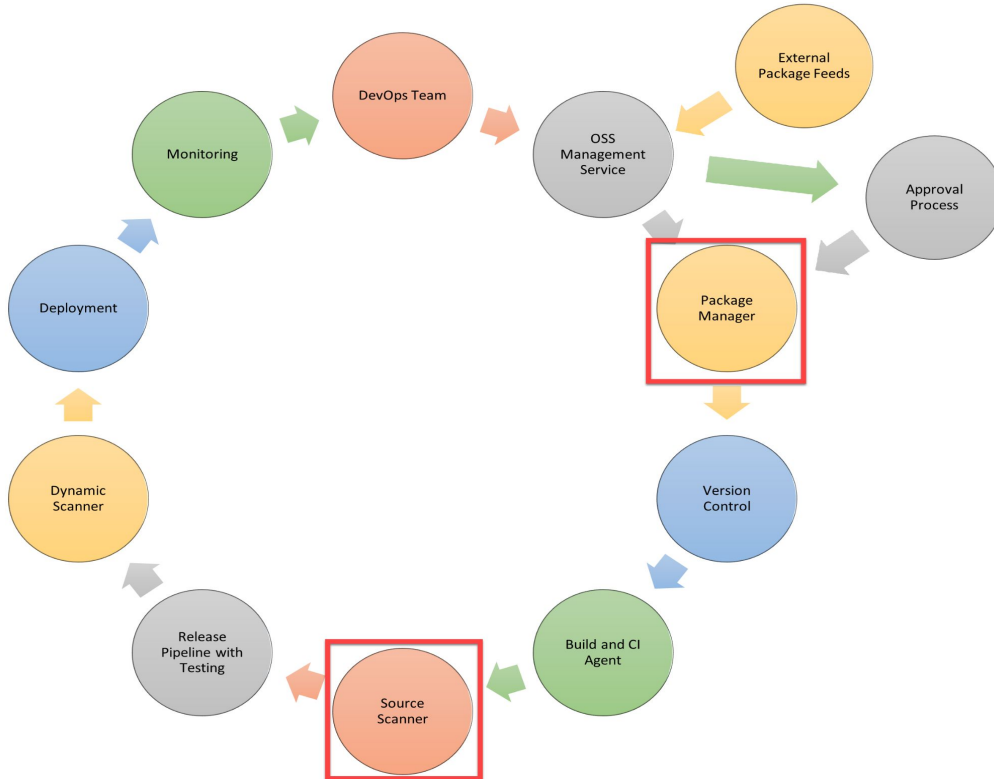
- Mon pipeline consomme-t-il des composants tiers et sont-ils sécurisés ?
- Existe-t-il des vulnérabilités connues dans l'un des logiciels tiers que nous utilisons ?
- À quelle vitesse puis-je détecter des vulnérabilités (également appelé *temps de détection*) ?
- À quelle vitesse puis-je corriger les vulnérabilités identifiées (également appelé *temps de correction*) ?

Les pratiques de sécurité pour la détection des anomalies de sécurité potentielles doivent être aussi robustes et rapides que les autres parties de votre pipeline DevOps. Cela inclut également l'automatisation de l'infrastructure et le développement de code.



# Exploration du pipeline Secure DevOps

l'objectif d'un pipeline *Secure DevOps* est de permettre aux équipes de développement de travailler rapidement sans introduire de vulnérabilités indésirables dans leur projet.



# Exploration du pipeline Secure DevOps

Les deux fonctionnalités essentielles des pipelines Secure DevOps que l'on ne retrouve pas dans les pipelines DevOps standard sont les suivantes :

- La gestion des packages et le processus d'approbation qui lui est associé. Le diagramme de workflow précédent décrit d'autres étapes d'ajout de packages logiciels au pipeline et les processus d'approbation par lesquels les packages doivent passer avant d'être utilisés. Ces étapes doivent être mises en œuvre tôt dans le pipeline pour identifier les problèmes de manière précoce dans le cycle.
- Analyseur source est également une autre étape supplémentaire pour l'analyse du code source. Cette étape permet d'analyser la sécurité et de rechercher des vulnérabilités qui ne sont pas présentes dans le code de l'application. L'analyse intervient *après* la génération de l'application, et *avant* le test de mise en production et de préversion. L'analyse de la source peut identifier les failles de sécurité plus tôt dans le cycle.

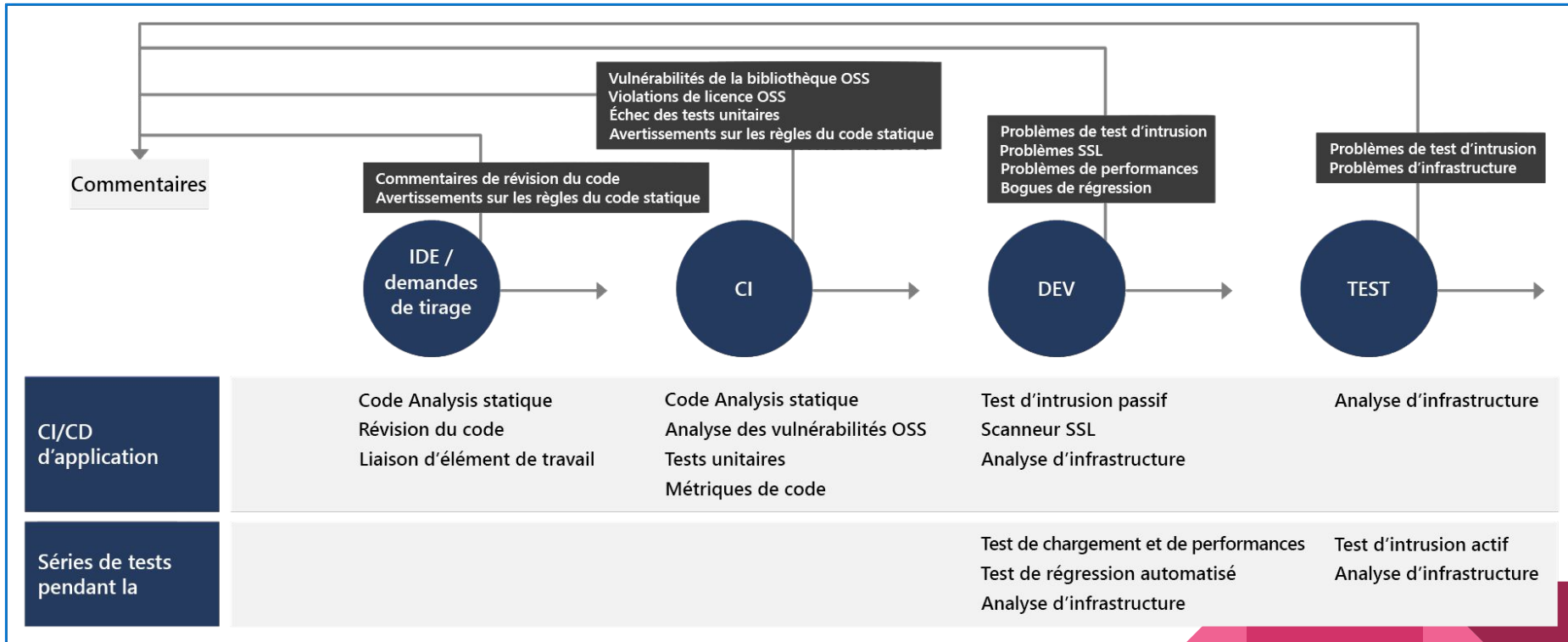


# Explorer les points de validation clés

- La validation continue de la sécurité doit être ajoutée à chaque étape du développement jusqu'à la production pour garantir que l'application est toujours sécurisée.
- Cette approche vise à faire passer la conversation avec l'équipe de sécurité de l'approbation de chaque mise en production au consentement du processus CI/CD, et à surveiller et auditer le processus à tout moment.
- Le diagramme ci-dessous met en évidence les points de validation critiques dans le pipeline CI/CD lors de la création d'applications entièrement nouvelles.
- La validation dans la CI/CD commence avant que le développeur ne commite son code.
- Les outils d'analyse du code statique de l'IDE fournissent la première ligne de défense pour s'assurer que des failles de sécurité ne sont pas introduites dans le processus CI/CD.
- Le processus de commit du code dans un dépôt central doit avoir des contrôles pour empêcher l'introduction de failles de sécurité.



# Explorer les points de validation clés



# Exploration de la validation de sécurité continue

- Les composants OSS (Open Source Software) sont encouragés dans de nombreuses organisations car ils assurent une livraison plus rapide et une meilleure productivité. Toutefois, étant donné que la dépendance envers ces composants d'OSS tiers augmente, le risque de failles de sécurité ou de spécifications de licence cachée augmente également les problèmes de conformité.
- Le fait d'identifier précocement ces problèmes dans le cycle de publication vous en informe suffisamment tôt pour les résoudre. Notamment, le coût de la rectification des problèmes est d'autant plus faible qu'ils sont détectés tôt.

⇒ De nombreux outils peuvent rechercher ces vulnérabilités dans les pipelines de build et de mise en production.. En règle générale, la principale différence entre les deux exécutions est que le processus PR-CI n'a pas besoin d'effectuer un empaquetage/une mise en lot dans le build CI.





# Exploration de la validation de sécurité continue

Ces builds CI doivent exécuter des tests d'analyse du code statique pour s'assurer que le code respecte toutes les règles de maintenance et de sécurité.

Plusieurs outils peuvent être utilisés pour cela :

- SonarQube.
- Analyse Visual Studio Code et Analyseurs de sécurité Roslyn.
- Checkmarx - Outil de test de sécurité d'application statique (SAST).
- BinSkim - Outil d'analyse statique binaire qui fournit des résultats sur la sécurité et sur sa bonne utilisation dans les exécutables portables Windows et plus encore.

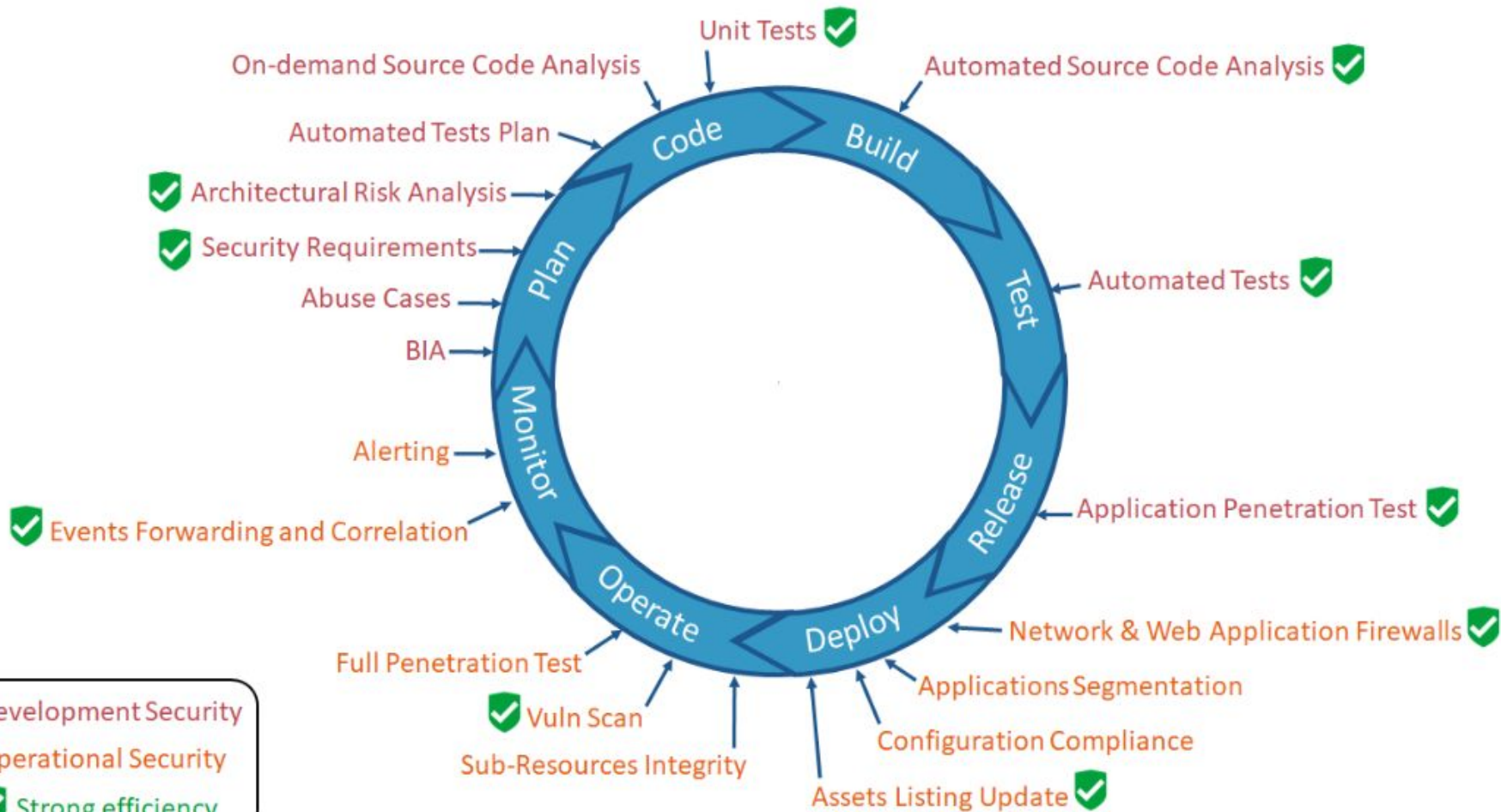


# Conclusion

En termes de sécurité applicative, les mesures les plus efficaces sont à prioriser dans cet ordre :

- De procéder à tests unitaires au niveau de l'équipe de développement, afin d'obtenir un premier niveau de vérification que les différentes fonctionnalités ne sont bien accessibles que dans le contexte initialement prévu [par exemple en testant l'accès à une API après suppression du token associé].
- L'analyse automatisée du code source (avec une solution comme Checkmarx ou Fortify).
- L'analyse itérative des risques relatifs à l'architecture.
- Des tests d'intrusion applicatifs, en mettant l'application au cœur de l'analyse.
- Des tests automatiques de sécurité au niveau de l'équipe qualité.
- L'extension des *Uses Cases* conventionnels aux *Abuses Cases*.







**Avez-vous  
des  
questions ?**

