CI/CD

Continuous Integration / Continuous Delivery/ Deployment

-

Intégration Continue / Livraison/ Déploiement continu

Philippe GUEMKAM SIMO

OBJECTIFS

- Fondamentaux et enjeux de la mise en place de l'intégration continue / Déploiement /livraison continue

- Améliorer la communication et la collaboration entre les différentes équipes IT / Métier

- Connaître les principes et les bonnes pratiques dans la mise en place d'un processus d’intégration continue

OBJECTIFS (Suite)

- Implémenter un processus CI/CD dans un projet existant
- Appréhender les outils indispensables

PLAN - MODULE 1
Préparer un projet à la CI/CD

Chapitre 1 : L’importance des tests pour la CICD

Chapitre 2 : L’importance de la qualimétrie de code pour la CICD

Chapitre 3 : Concepts et enjeux de la livraison logicielle

PLAN - MODULE 2
Intégration / Déploiement / Livraison continue

Chapitre 4 : Continuous Integration, première étape du Continuous delivery

Chapitre 5 : Continuous delivery vs Continuous Deployment

Chapitre 6: Gestion de données

Chapitre 7 : Gestion des infrastructures et environnement de déploiement

1- L’importance des tests pour la CICD

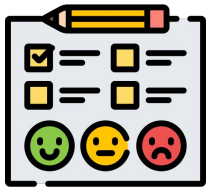
1- L’importance des tests pour la CICD

Les Tests :

Éléments indispensables de l’ “**Extreme Programming”/Agilité/CICD**

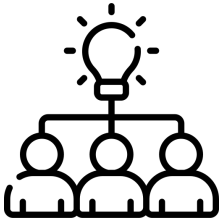
1-1 Extreme Programming / CICD / DevOps

Rapid fine feedback



Continuous Process

Shared understanding



Developer welfare

1-2 La place des tests

Rapid, Fine Feedback

Tests
“Working Software”

Continuous Process

Livrables courts
Intégration continue

Xtrem Prog / Agilité / DevOps

Améliorer/développer

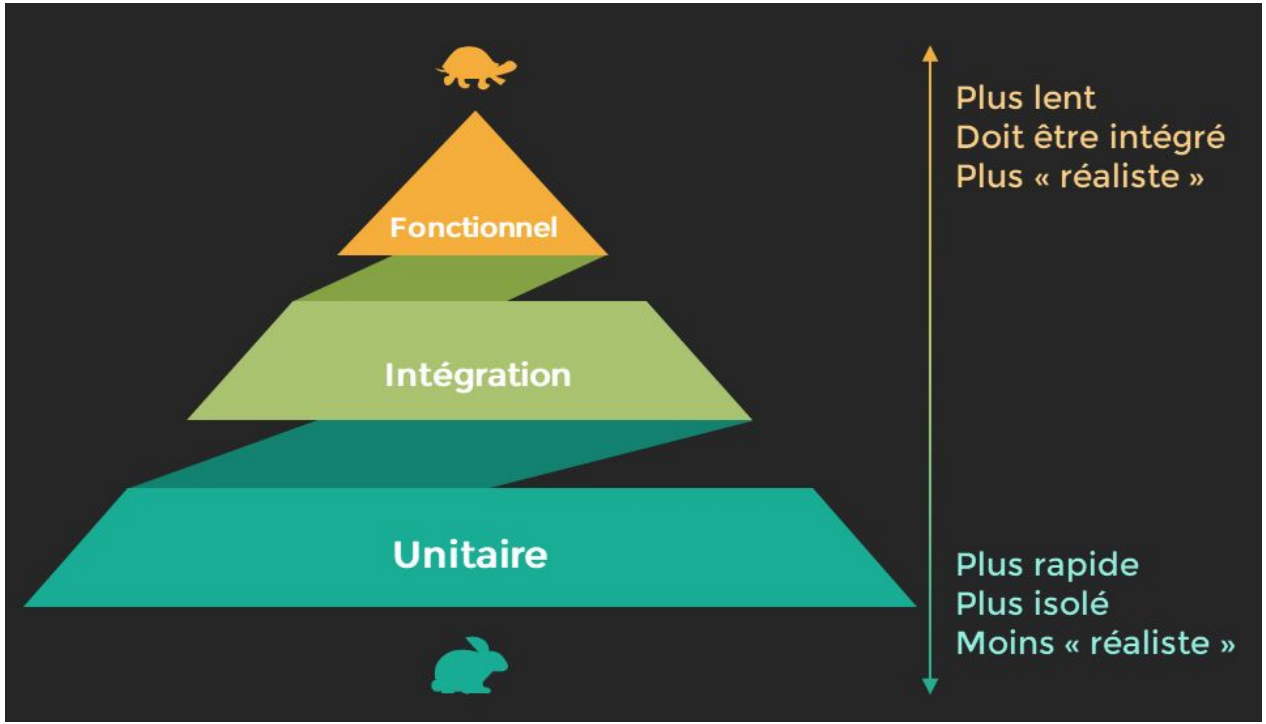
Tester

Intégrer

Livrer un incrément qui marche !

Rebelote ...

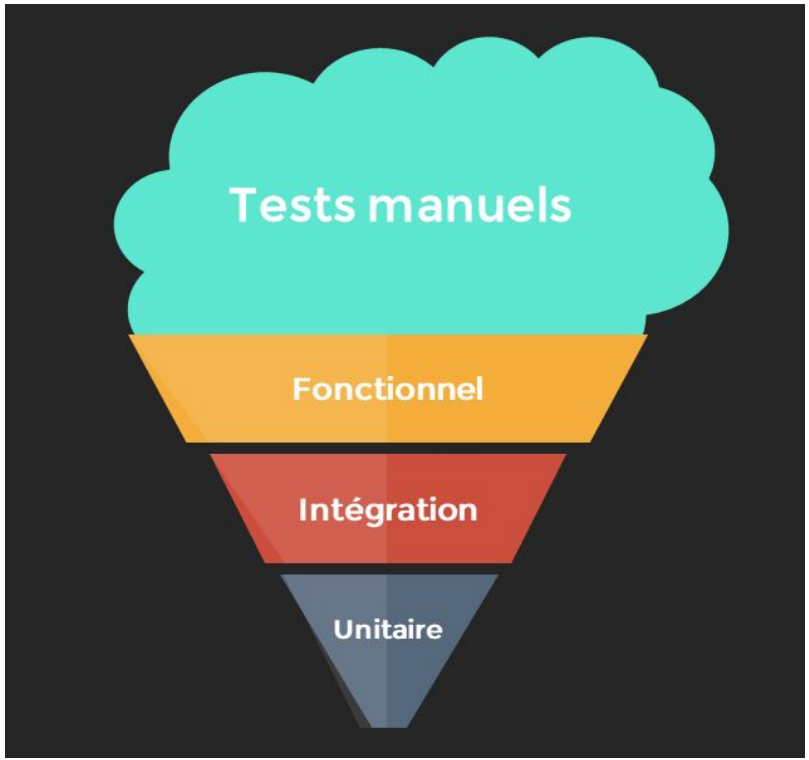
1-3 Les types de test



$$$
Fragiles

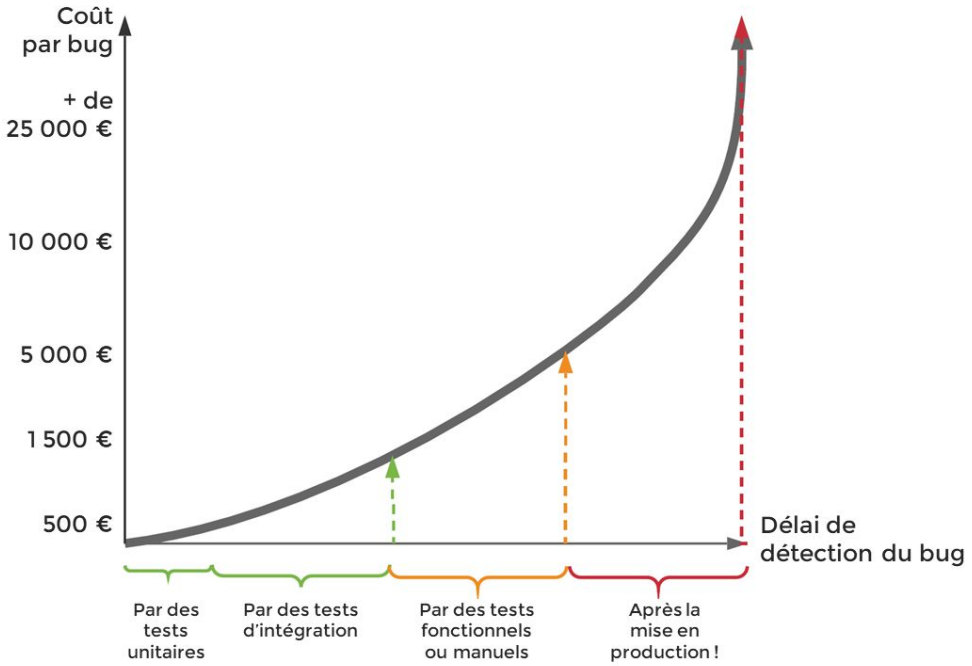
$
Stables

1-3 Les types de test



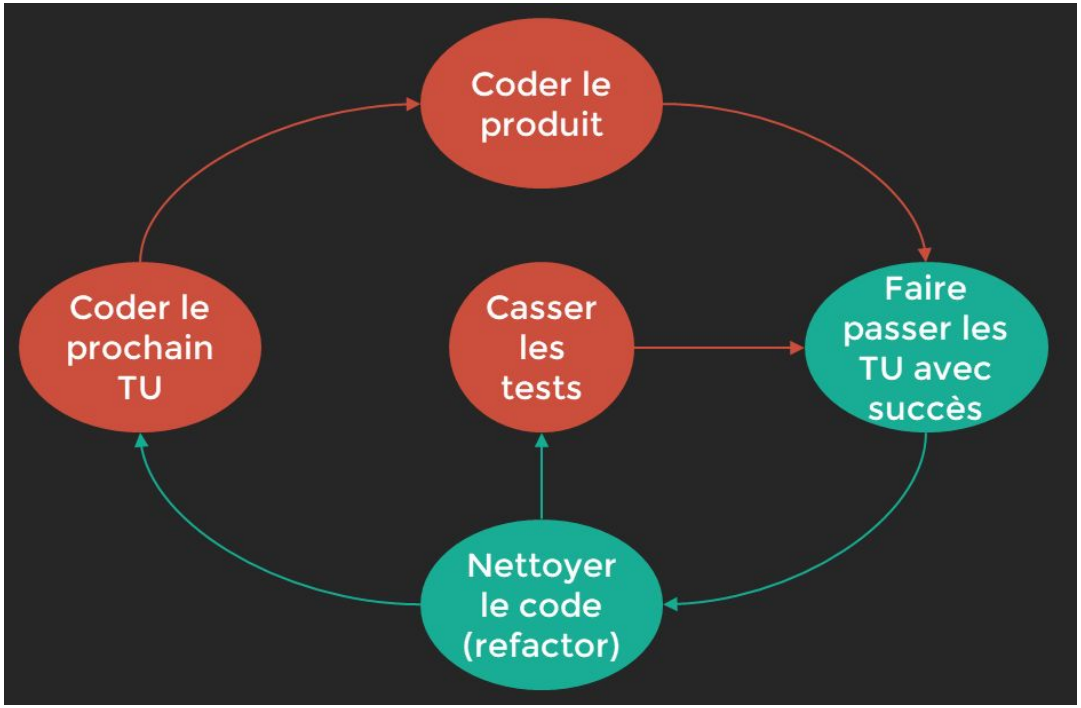
Evitez le cône de glace

1-4 Tester le plus tôt possible



Testez tôt et économisez

1-5 Principe du Test Driven Development



Avantages:

Code précis et concis

Respect des fonctionnalités

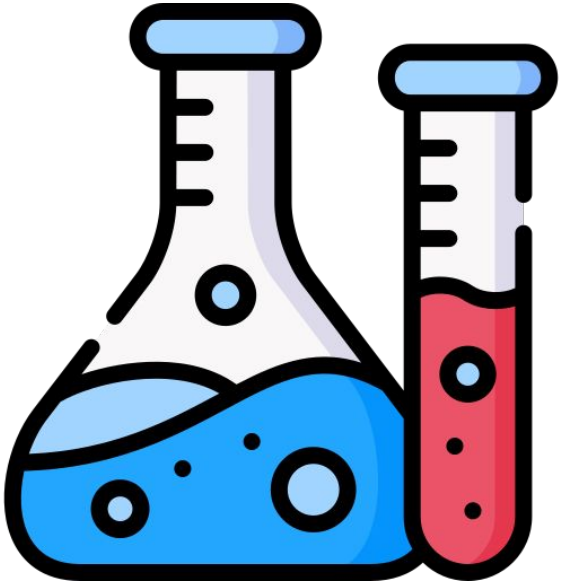
Testable unitairement

Stable

1- 6 Test auto / Tests non auto

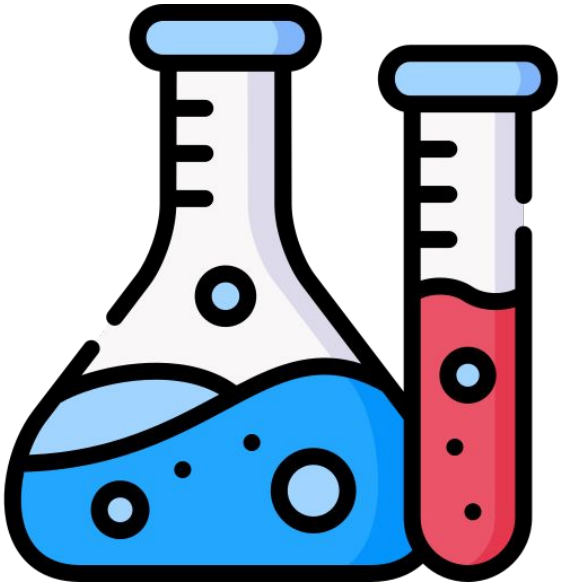
Test automatisés	**Test manuels**
Non chronophages	Chronophage
Auto-validation indispensable pour la CI	Efficacité relative (non profonds)
Feedback rapide	Feedback Lent
Difficile à écrire / coder	Pas de code nécessaire
Nécessite des jeux de données	Léger jeux de données

1- 7 LAB



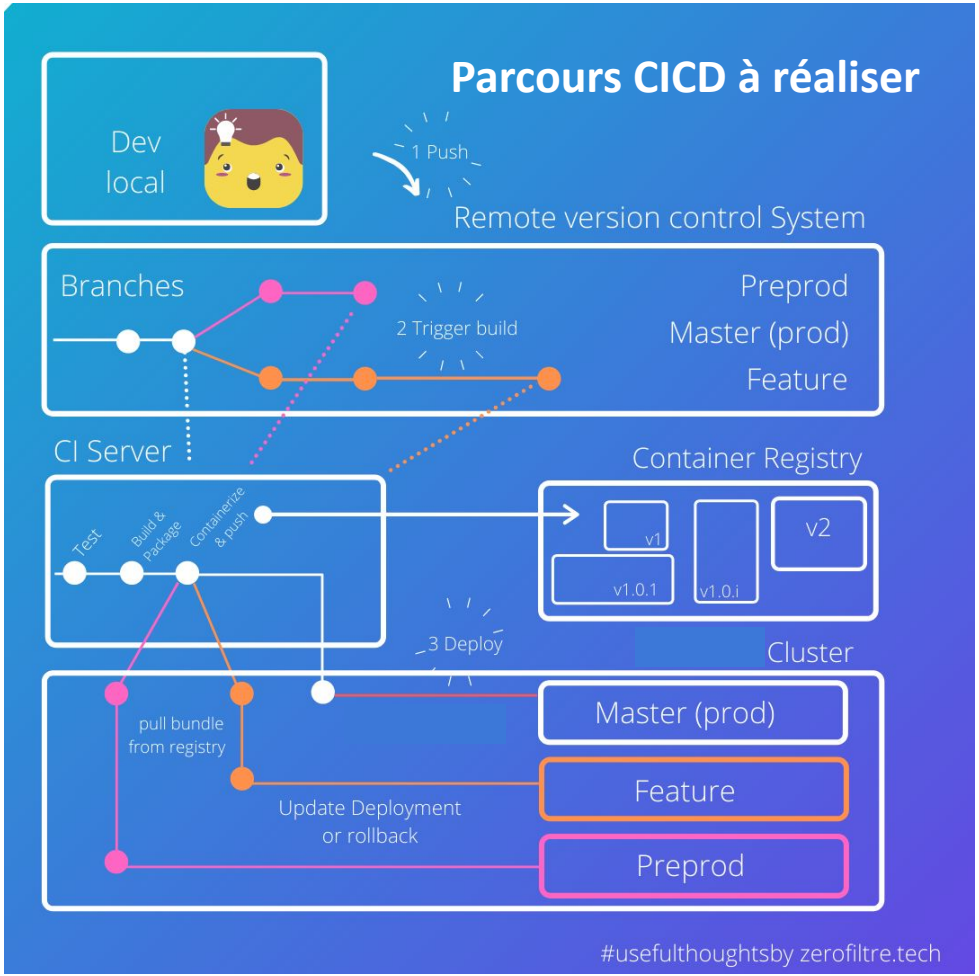
- Un exemple de test manuel / auto
- Notion de branching
- Initialisation des branches

1- 7 Prérequis



- Java 8 ou + : https://adoptium.net
  Maven 3 : https://maven.apache.org/download.cgi
- Git for windows : https://git-scm.com/download/win
  (Éventuellement créer un compte sur github.com)
- Intellij IDEA Community Edition :
  https://www.jetbrains.com/idea/download/#section=windows
- Docker Desktop : https://docs.docker.com/desktop/
- Pour Windows :
  https://docs.docker.com/desktop/windows/install/
- Créer un compte sur Docker Hub : https://hub.docker.com/

Définition des branches liées aux environnements



1- 9 Conclusion

3 types de tests: unitaires, intégration et fonctionnels,

Les test unitaires sont les plus utilisés => moins douloureux,

Les tests permettent d'obtenir un feedback rapide sur les nouvelles fonctionnalités,

Les tests protègent contre la régression,

Ajouter les tests automatisés au processus de CI assure la non régression, le feedback rapide et favorise produit fonctionnel

2- L’importance de la qualimétrie de code pour la CICD

2- 1 Les métriques de qualité de code

Métriques qui mesurent de la qualité du code

Taux de couverture de tests

Nombre de duplications

Complexité Cognitive

Nombre de bugs

Quantité de code défectueux (Code Smell)

Dette Technique

...

2- 1 Les métriques de qualité de code

Ces métriques sont gages de:

- Maintenabilité
- Robustesse
- Non régression de l’application

D’où l’importance de s’assurer de :

- leur maintien
- leur non-dégradation

Comment ?

2 - 2 Processus de gestion de qualimétrie

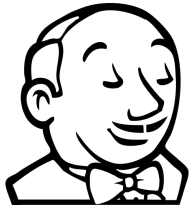
Monitoring constant du code:

- Définir les métriques de qualité
- Définir les seuils
- Collecter
- Publier

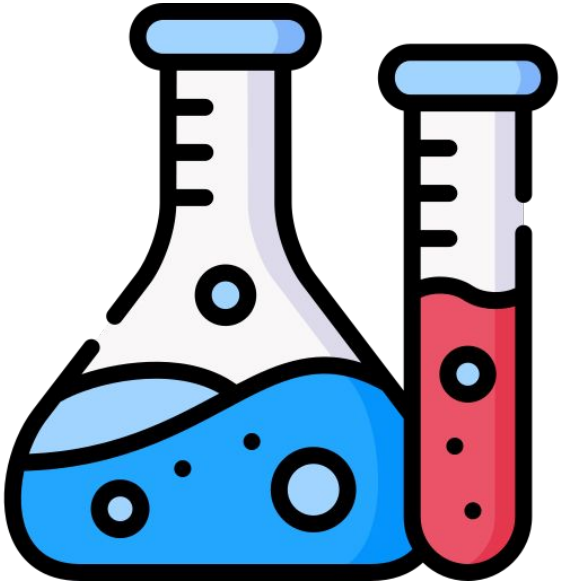
sonarqube

Vérifier en continu:

- Automatiser le refus de dégradation de qualité dans les nouvelles itérations
- **Automatisation à inclure comme étape d'un processus CICD**

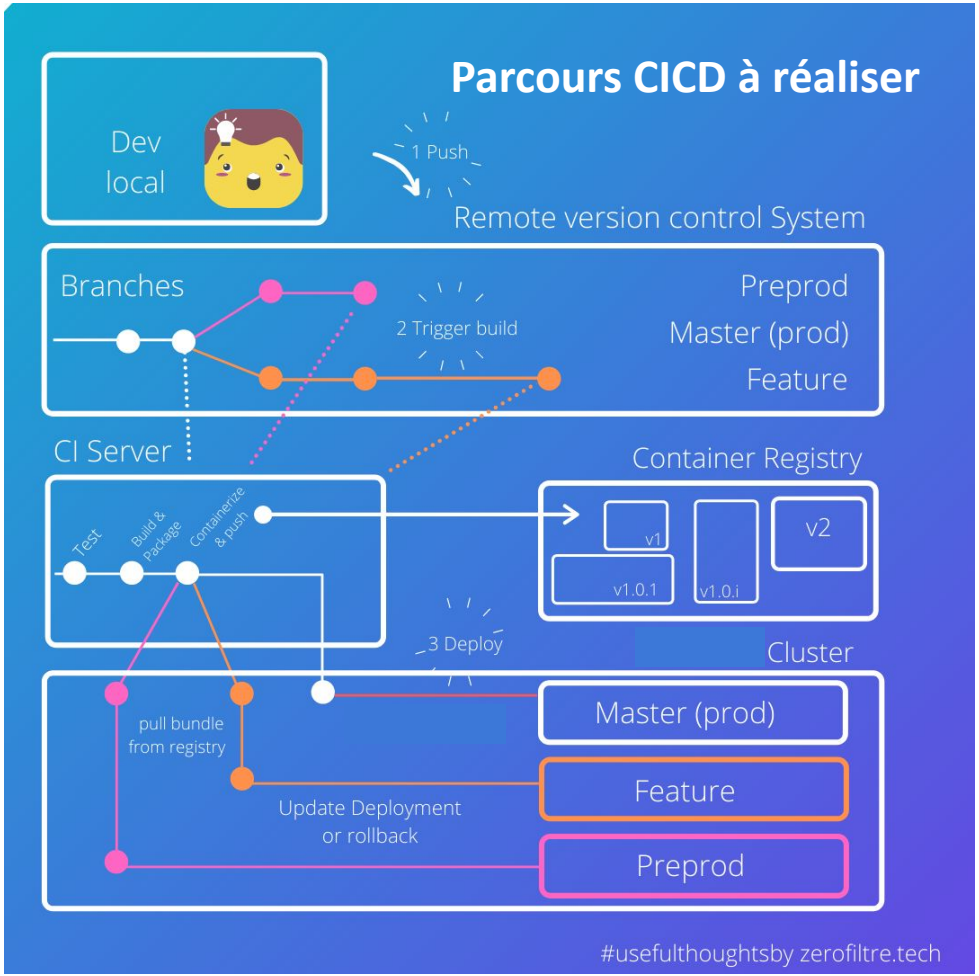


2- 3 LAB : Monitoring de code avec SONAR



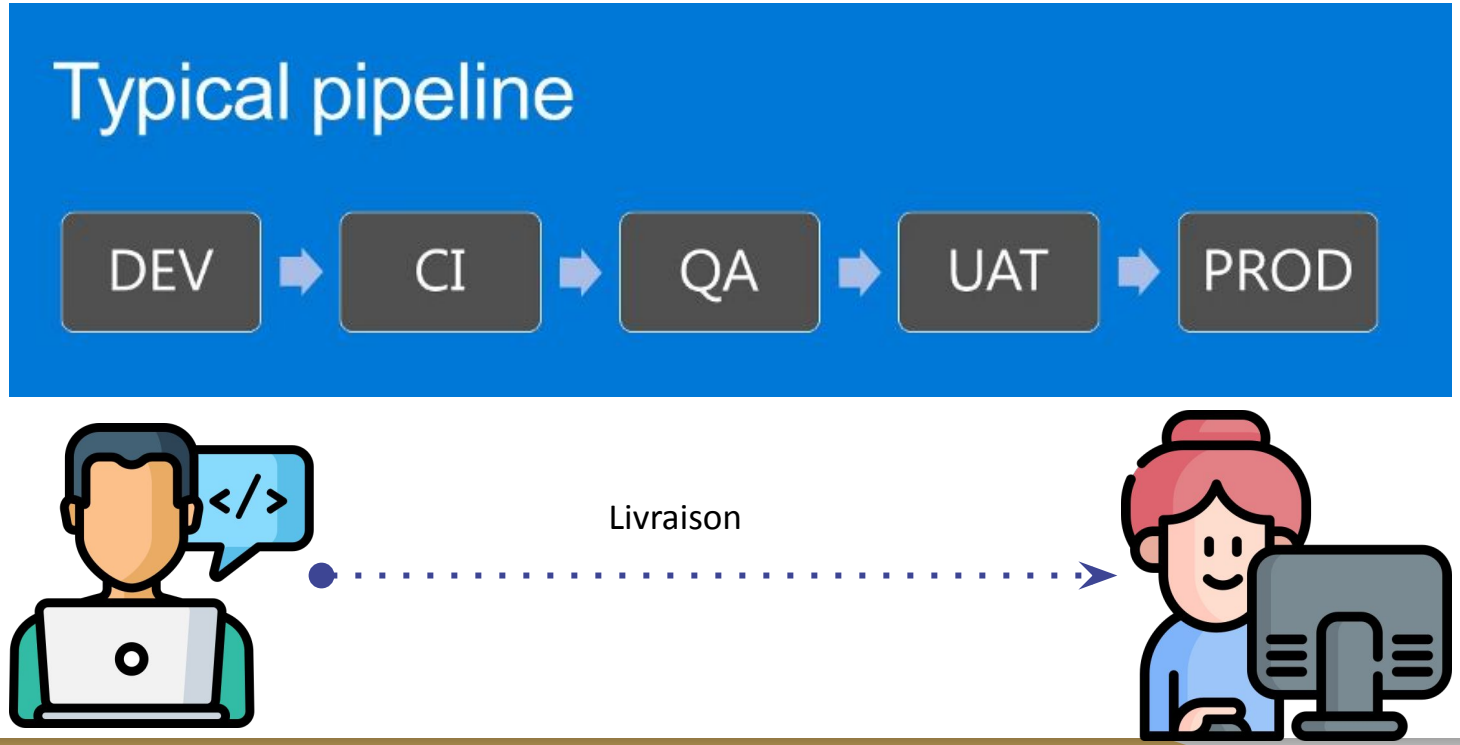
- Installation
- Définition des seuils
  Quality Gate / Quality Profile
- Configuration du taux de couverture de tests

Serveur de qualimétrie à intégrer au serveur CI



3 - Concepts et enjeux de la livraison logicielle

3 - 1 La livraison logicielle



3 - 2 Problématiques

Plusieurs environnements différents: DEV , CI, QA, UAT, PROD ...

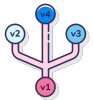
Maintenir l’équivalence
“Ça marchait pourtant sur mon PC” ⚠️

Configurations adaptées au contexte
(Capacité PROD > Capacité DEV)

Configuration manuelle

Configuration automatisée

Montée de version
Code/environnement

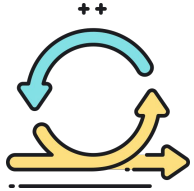


3 - 3 Stratégie de livraison - Définition

Un seul objectif: ⚠️ **Livrer pour améliorer la chaîne de valeur** ⚠️

i/ Identifier les opportunités d'amélioration

ii/ Mesurer l'impact des améliorations sur le business



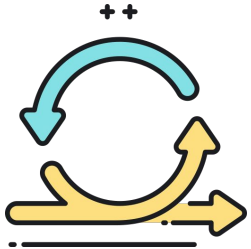
iii/ Implémenter les modifications

iv/ Déployer les résultats

3 - 3 Stratégies de livraison - Acteurs

Product Owner

DevOps Team



Quality Manager

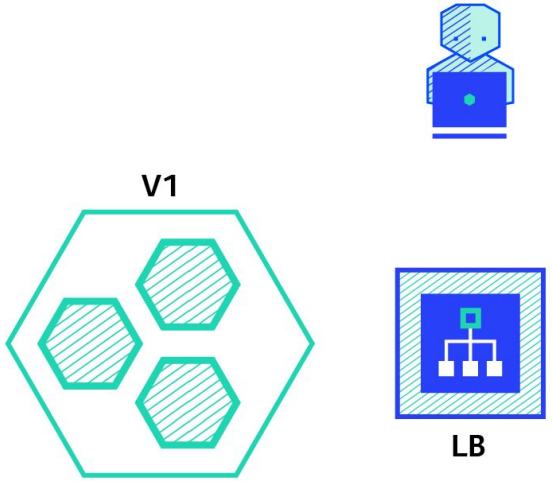
3 - 4 Stratégies de déploiement

Livraison => Déploiement de résultat

- **Recreate**
- **Ramped**
- **Blue/Green**
- **Canary**
- **A/B testing**
- **Shadow**

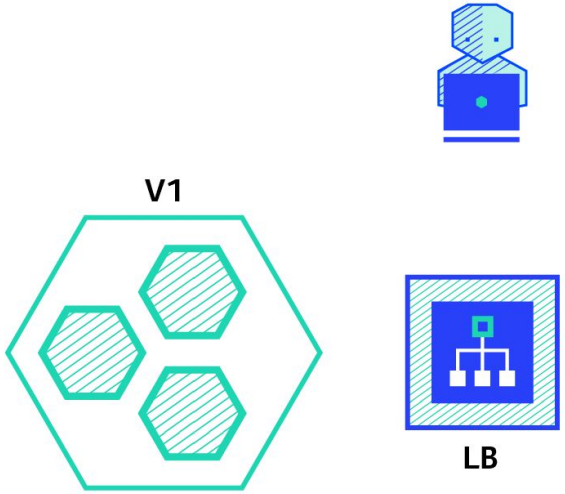
3 - 4 Stratégies de déploiement

Recreate



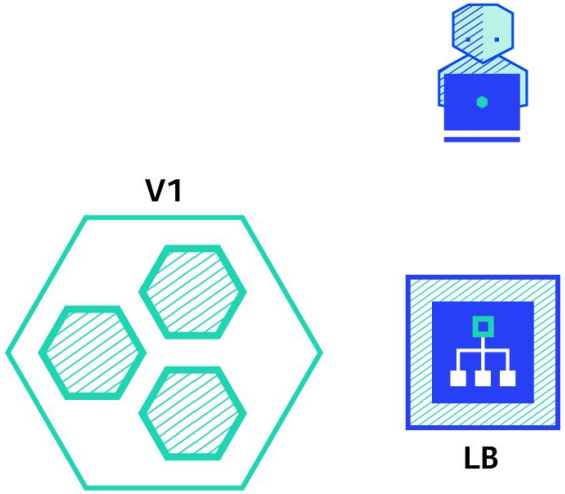
3 - 4 Stratégies de déploiement

Ramped



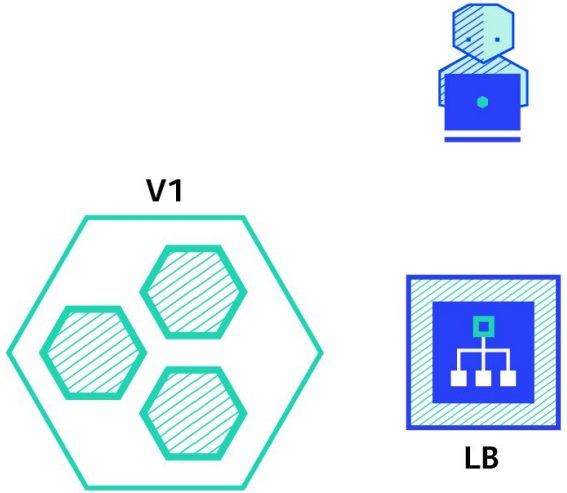
3 - 4 Stratégies de déploiement

Blue / Green



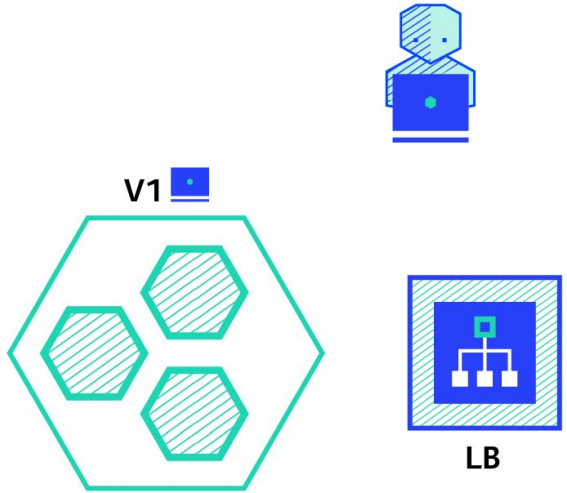
3 - 4 Stratégies de déploiement

Canary



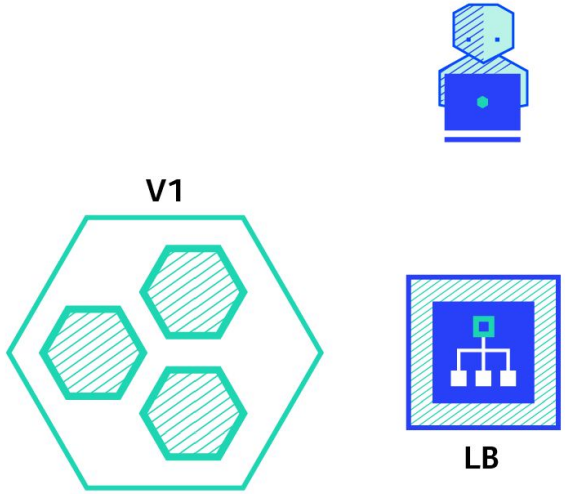
3 - 4 Stratégies de déploiement

A/B Testing



3 - 4 Stratégies de déploiement

Shadow



DEPLOYMENT STRATEGIES

When it comes to production, a ramped or blue/green deployment is usually a good fit, but proper testing of the new platform is necessary.

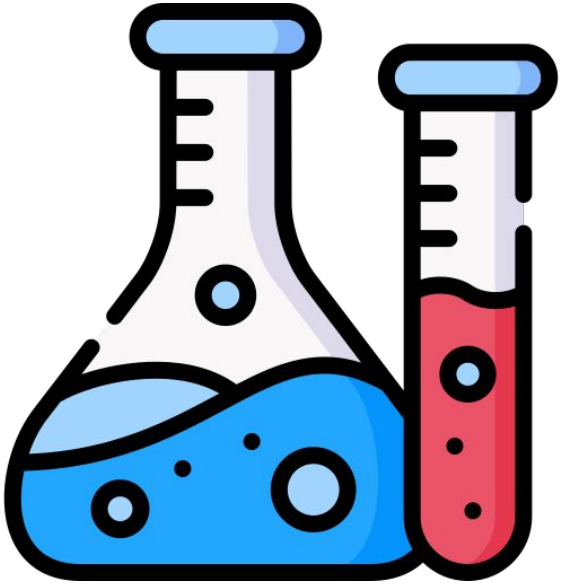
Blue/green and shadow strategies have more impact on the budget as it requires double resource capacity. If the application lacks in tests or if there is little confidence about the impact/stability of the software, then a canary, a/b testing or shadow release can be used.

If your business requires testing of a new feature amongst a specific pool of users that can be filtered depending on some parameters like geolocation, language, operating system or browser features, then you may want to use the a/b testing technique.

Container Solutions

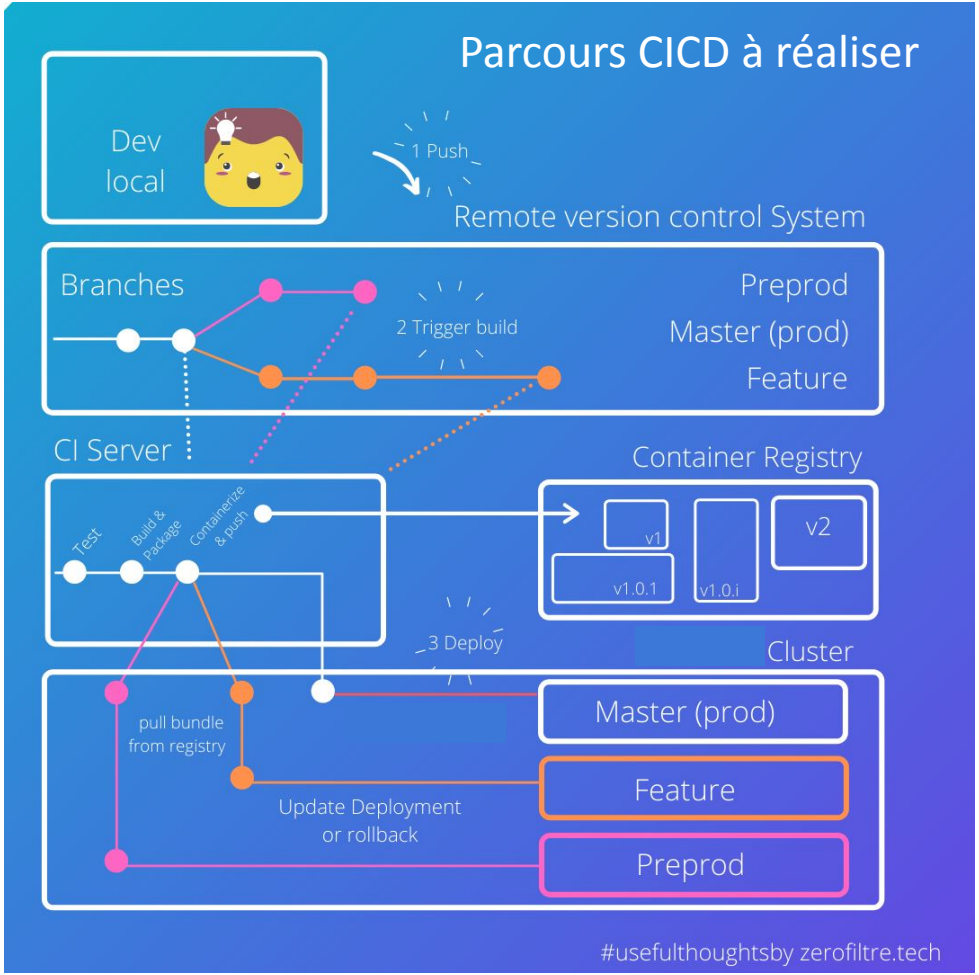
Strategy	ZERO DOWNTIME	REAL TRAFFIC TESTING	TARGETED USERS	CLOUD COST	ROLLBACK DURATION	NEGATIVE IMPACT ON USER	COMPLEXITY OF SETUP
RECREATE version A is terminated then version B is rolled out	✗	✗	✗	■□□	■■■	■■■	□□□
RAMPED version B is slowly rolled out and replacing version A	✓	✗	✗	■□□	■■■	■□□	■□□
BLUE/GREEN version B is released alongside version A, then the traffic is switched to version B	✓	✗	✗	■■■	□□□	■■□	■■□
CANARY version B is released to a subset of users, then proceed to a full rollout	✓	✓	✗	■□□	■□□	■□□	■■□
A/B TESTING version B is released to a subset of users under specific condition	✓	✓	✓	■□□	■□□	■□□	■■■
SHADOW version B receives real world traffic alongside version A and doesn't impact the response	✓	✓	✗	■■■	□□□	□□□	■■■

3 - 5 LAB : Déploiement manuel



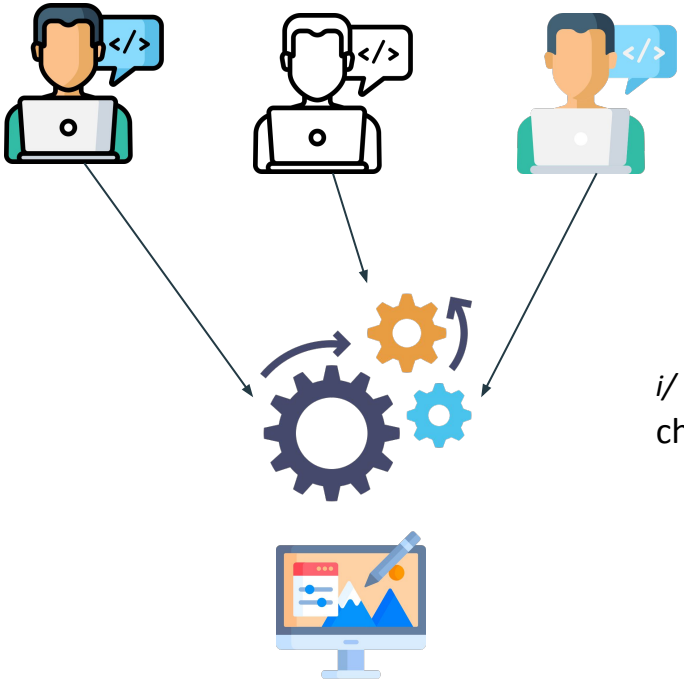
- Configuration des environnements
- Déploiement manuel
- Stratégie **Recreate**

Initialisation des profils d'exécution liées aux environnements



4- CI, première étape du Continuous Delivery

4- 1 Intégration Continue - Définition

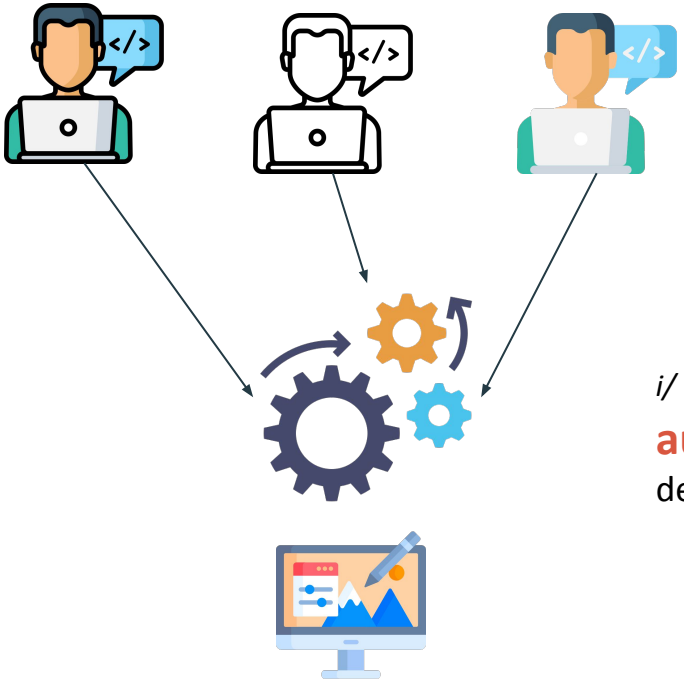


ii/ … venant de plusieurs dev

i/ **Automatiser l'intégration** des changements de code …

iii/ … dans **un seul projet Logiciel**

4- 1 Intégration Continue - Définition



ii/… venant de plusieurs dev

i/ **Fusionner et tester automatiquement** des changements de code ...

iii/… dans **un seul projet Logiciel**

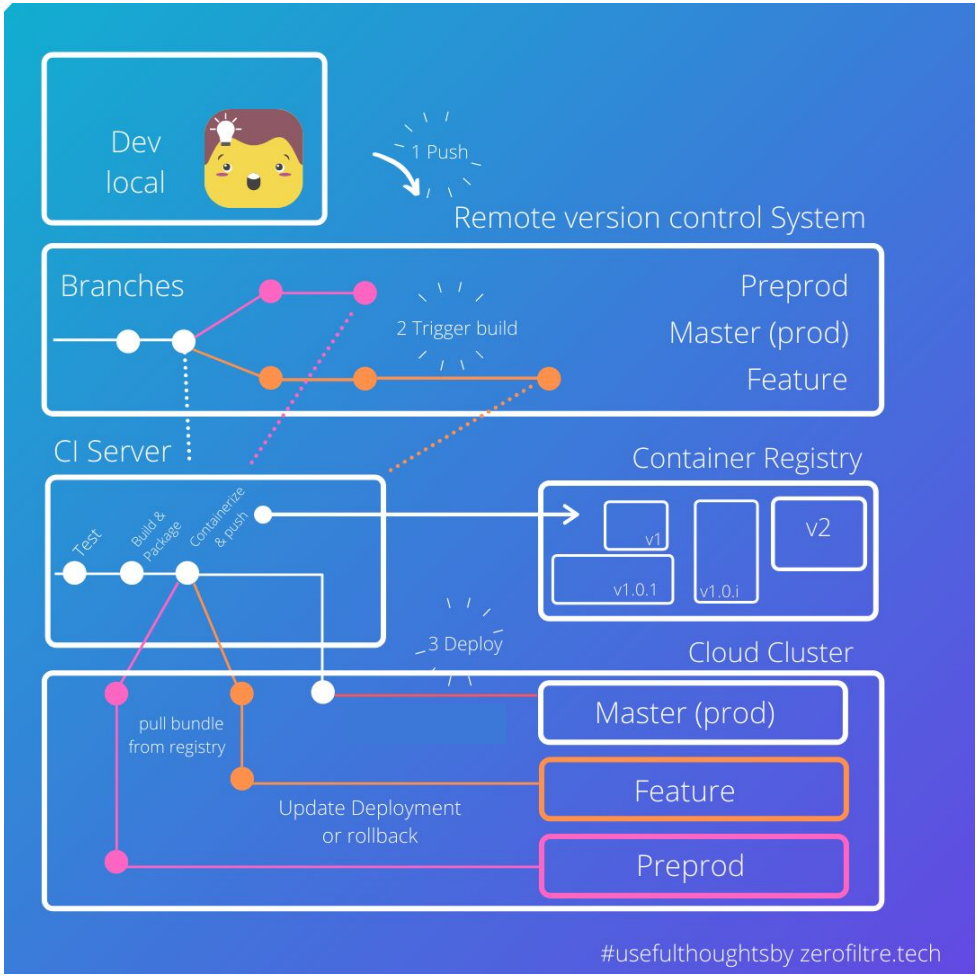
4- 3 Intégration continue : Importance

Sans CI	Avec CI
Coordination manuelle entre DEV, OPS , testeurs ...	Travail indépendant des DEV sur des fonctionnalités en parallèle
Communication manuelle	Livraison des tâches indépendamment des autres tâches en cours
Bureaucratie supplémentaire liée à la coordination	
Travail des DEV pas/mal connu des autres parties prenantes	Transparence dans le processus de DEV

4- 2 Processus

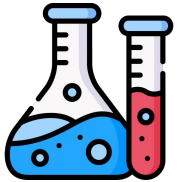
Intégration Continue

Déploiement Continu

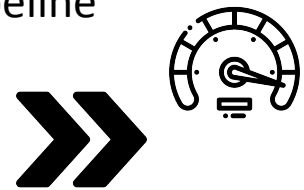


4- 4 Bonnes pratiques d’intégration continue

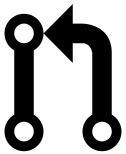
Test Driven Development



Optimiser la vitesse du pipeline



Pull Requests & Code Reviews



4-5 Intégration continue et Communication

Indicateurs de visibilité du résultat construit :

Tableau de bord Sonar

Statut, Santé et tendance :

Tableau de bord Sonar

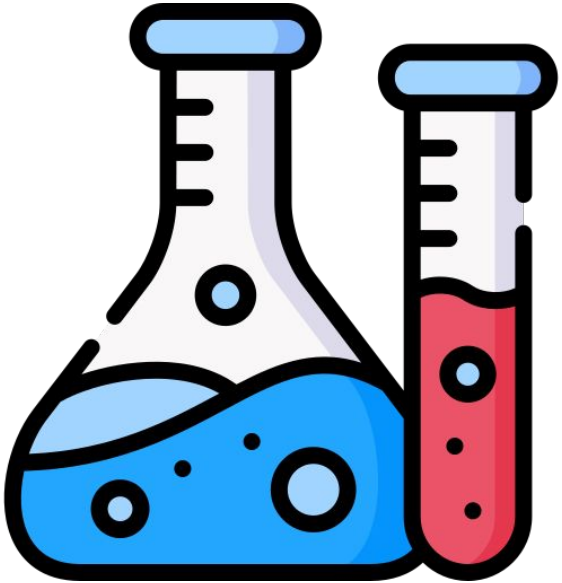
Notes de release

Rédiger des notes de release

Notification par email :

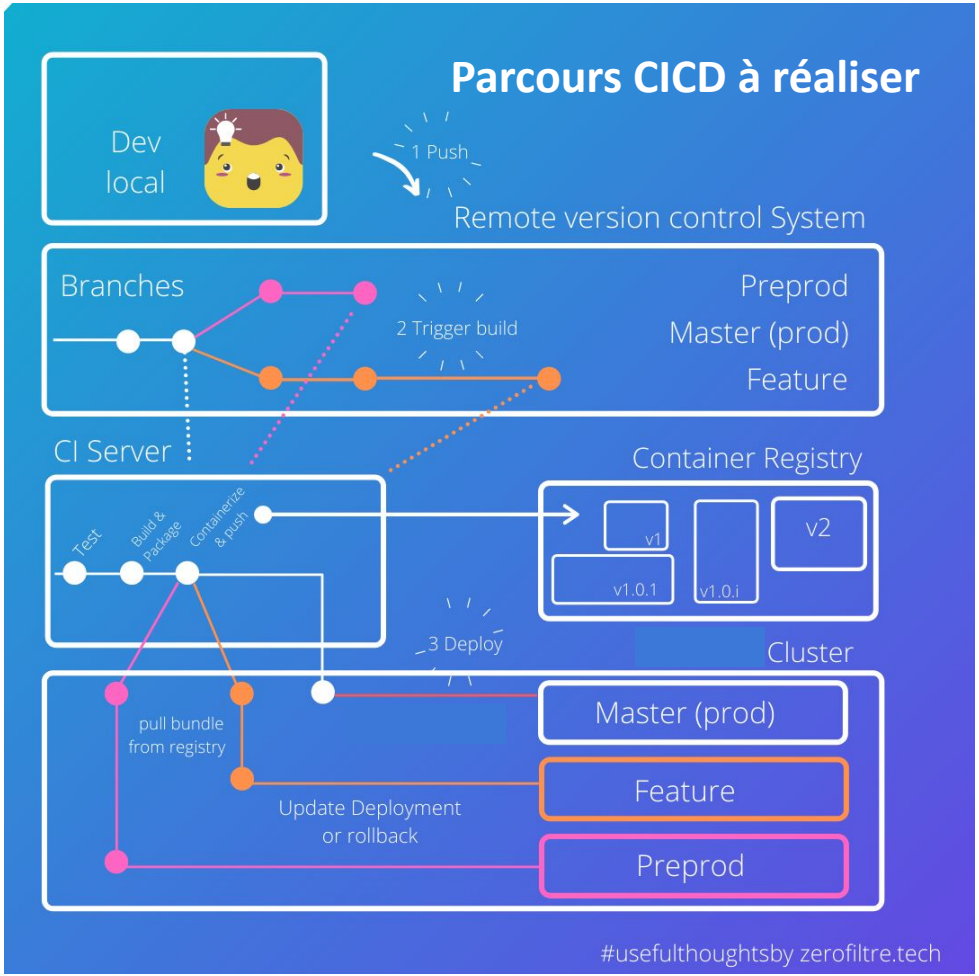
Jenkins

4 - 6 Mise en place de l'intégration continue pour un projet existant



- Installation Jenkins & Configuration
- Ecriture du pipeline
- Intégration d'une nouvelle feature
- Envoi de mail

Mise en place du serveur CI

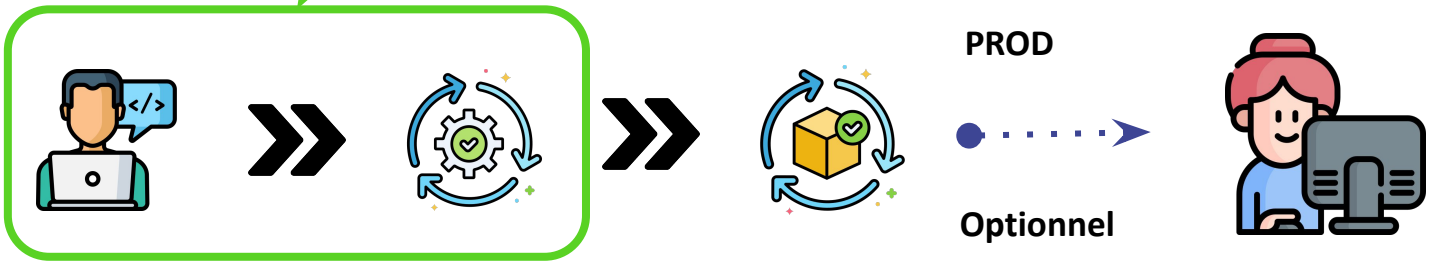


5 - Continuous Delivery vs C. Deployment

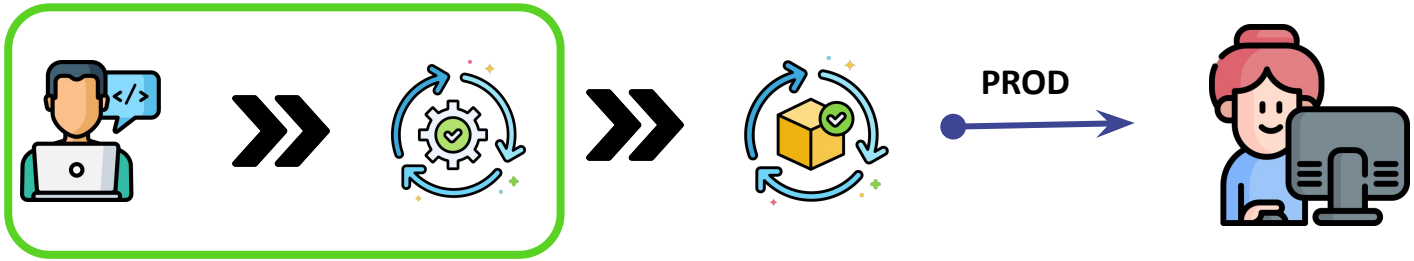
5 - Continuous Delivery vs Deployment

Livraison continue

CI

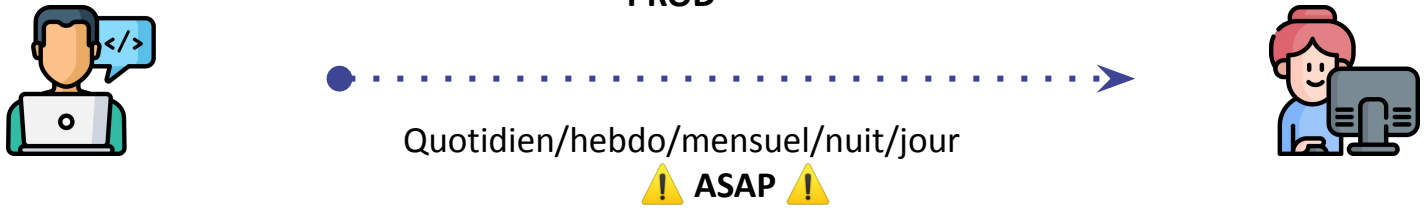


Déploiement continu

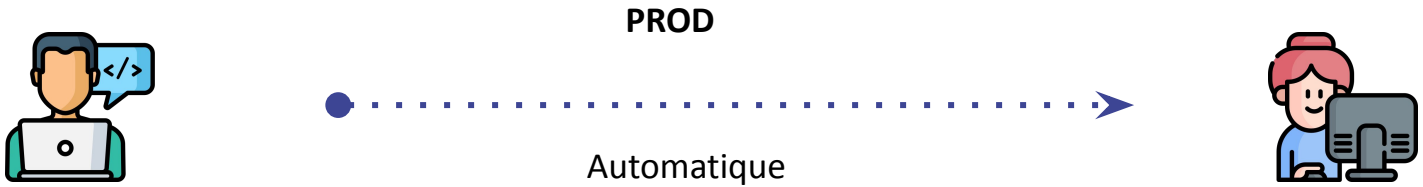


5 - Continuous Delivery vs Deployment

Livraison continue : Fréquence de mise en PROD adaptée au besoin métier



Déploiement continu



5 - 1 Delivery vs Deployment : Impacts

Coût

Livraison continue
Ecriture des tests automatisés pour chaque nouvelle fonctionnalité /bug/amélioration
Bon taux de couverture
Mise en place d’un serveur d’intégration continu
Fusion des changement autant que possible (au moins 1/jour)
Maîtrise de la notion et flag et de versioning
Avoir un **Git flow** clairement défini

5 - 1 Delivery vs Deployment : Impacts

Coût

Déploiement Continu
Qualité de tests impeccable
La documentation doit être constamment mis à jour en fonction des nouvelles releases
Notion de flag et de versioning impeccable
Communication Marketing / RH / Support autour de chaque nouvelle release

5 - 1 Delivery vs Deployment : Impacts

Bénéfices

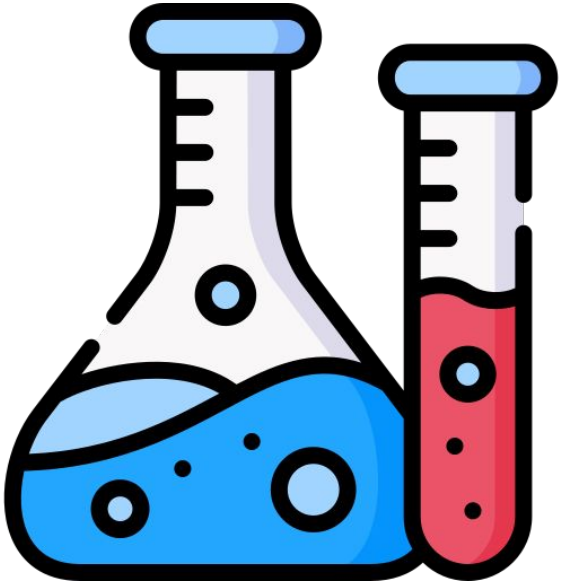
Livraison continue
Moins de bugs en production - Régressions détectées au plus tôt
Moins de changements de contexte pour les dev: Les bugs détectés sont corrigés aussitôt, avant les tâches suivantes
Coûts de tests réduits % tests manuels: Les serveurs CI exécutent des centaines de tests en secondes
Les tests manuels sont réservés à la stricte politique QUALITE de l'entreprise
Moins de perte de temps pour la préparation des livraisons - automatisation

5 - 1 Delivery vs Deployment : Impacts

Bénéfices

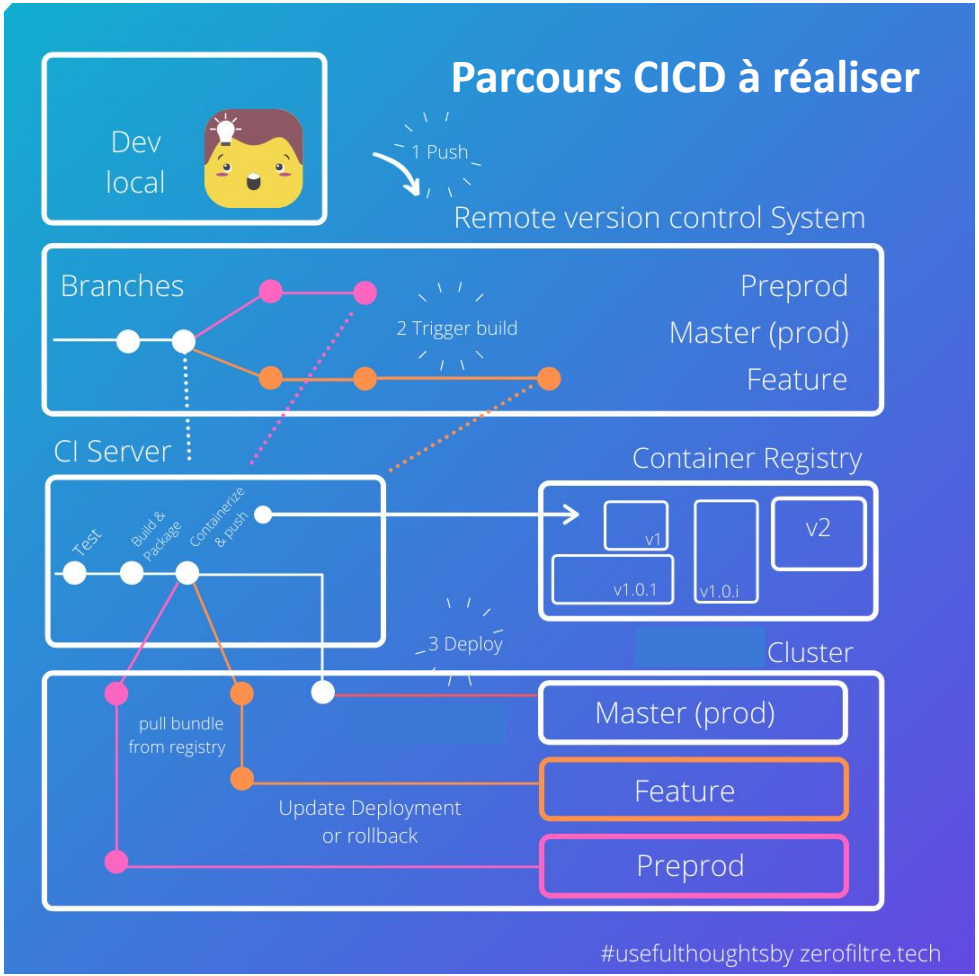
Déploiement Continu
Feedback rapide, itérations rapides, développement rapide: Pas besoin de pauses pour préparer un release
Les releases sont moins risquées et faciles à corriger car les changements sont minimes
Apport de valeur continue pour le consommateur

5 - 2 Déploiement continu

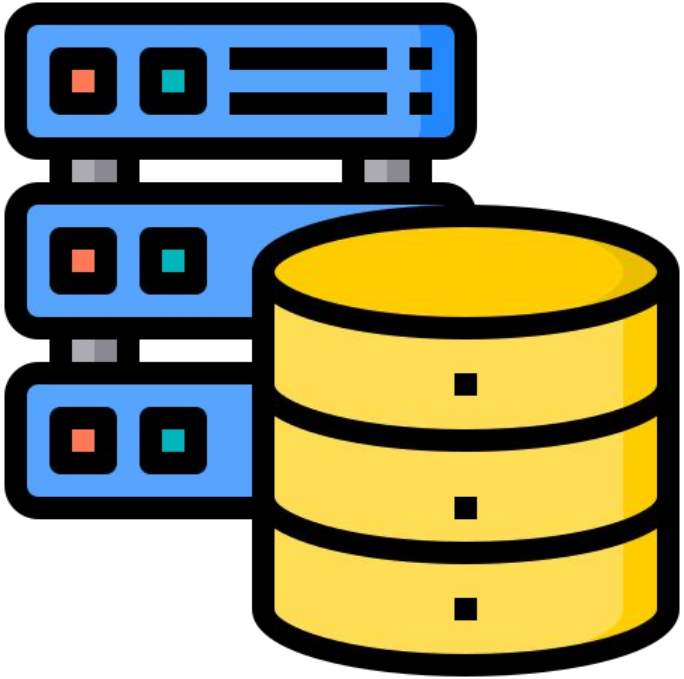


- Exemple de politique de branching et de versionning
- Livraison en PROD avec tags
- Correction d’un bug en prod: hotfix

Orchestration de déploiement



6 - Gestion de données



6 - 1 Problématiques

Plusieurs environnements différents: DEV , CI, QA, UAT, PROD ...

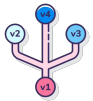
Maintenir l'équivalence
"Ça marchait pourtant sur mon PC" ⚠️

Configurations adaptées au contexte
(Disque PROD > Disque DEV)

Configuration manuelle

Configuration automatisée

Montée de version code/environnements



6 - 2 Montée en version du schéma de BD

Problématique :

- BD déjà fournie en données et utilisée
- Les besoins évoluent - le schéma de la BD doit s'adapter aussi bien que le code
- Faire évoluer un schéma nécessite l'intervention/validation des DBA - perte de temps
- La philosophie CICD est vaine si tout le processus n'est pas fluide

6 - 2 Montée en version du schéma de BD

Solution:

Il faut traiter les changements évolutions de la BD tel que l'on traite ceux du code

Modification | Commit | Validation | Push

[Tracage] | [Rollback]

6 - 2 Montée en version du schéma de BD

Liquibase

Définir les changements à faire à la BD à travers des *changesets*

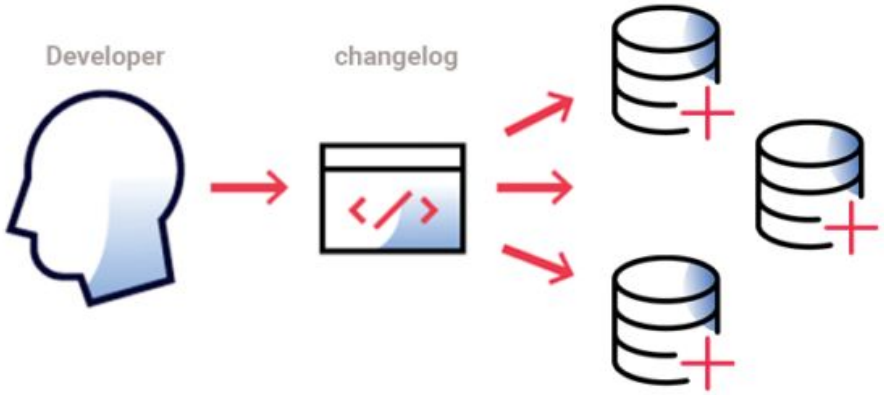
Ex: Création d'une table

```
1.  <changeSet author="Bob" id="157">
2.    <createTable tableName="person">
3.      <column autoIncrement="true" name ="id"
type="INTEGER">
4.        <constraints> nullable="false"
primaryKey="true"
primaryKeyName="person_pkey"/>
5.      </column>
6.      <column name="firstname"
type="VARCHAR(50)"/>
7.      <column name="lastname"
type="VARCHAR(50)">
8.        <constraints nullable="false"/>
9.      </column>
10.     <column name="state" type="CHAR(2)"/>
11.     <column name="username"
type="VARCHAR(8)"/>
12.     <column name="column1"
type="VARCHAR(8)"/>
13.     <column name="column2"
type="VARCHAR(8)"/>
14.   </createTable>
15.  </changeSet>
```

6 - 2 Montée en version du schéma de BD

Liquibase

Gérer le schéma de la BD pour plusieurs types de BD



Formats agnostiques aux BD:
XML, YAML, JSON

Rollback possible !

6 - 2 Montée en version du schéma de BD

Liquibase

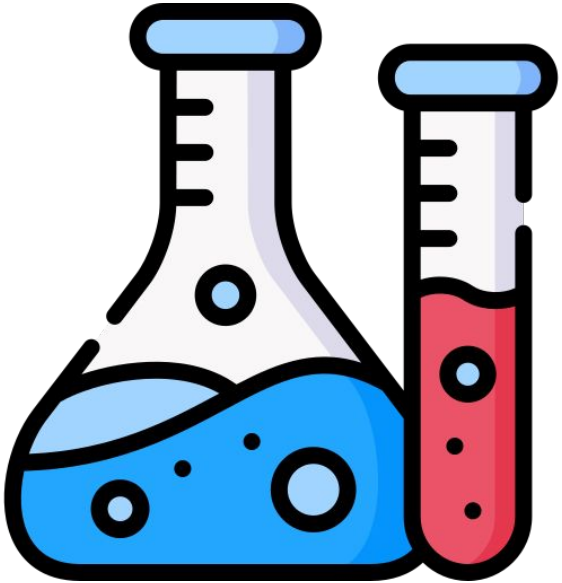
Ajouter et ordonner les *changesets* dans un registre commun : Le *changelog*

C’est le journal qui trace tous les changements et l’ordre dans lequel ils doivent être appliqués

Une table spéciale sera créée pour sauvegarder ces changements

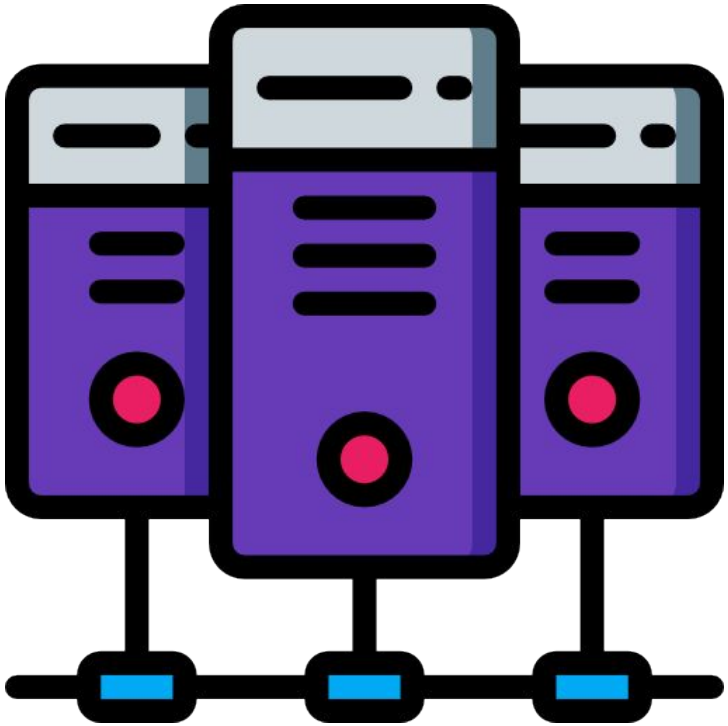
```xml
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
    xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"
    xmlns:pro="http://www.liquibase.org/xml/ns/pro"

xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-3.8.xsd
    http://www.liquibase.org/xml/ns/dbchangelog-ext http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd
    http://www.liquibase.org/xml/ns/pro http://www.liquibase.org/xml/ns/pro/liquibase-pro-3.8.xsd ">
</databaseChangeLog>
```

6 - 3 Lab: Liquibase



- Génération de différence entre schémas
- Mise à jour de BD
- Intégration au pipeline CICD

7 - Gestion des INFRA et ENV de déploiement



7 - 1 Problématiques

Plusieurs environnements différents: DEV , CI, QA, UAT, PROD ...

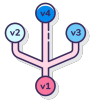
Maintenir l'équivalence
"Ça marchait pourtant sur mon PC" ⚠️

Configurations adaptées au contexte
(Disque PROD > Disque DEV)

Configuration manuelle

Configuration automatisée

Montée de version code/environnements



7 - 1 Problématiques

Problématique :

- Configurer manuellement un serveur peut prendre des jours:
  Contraintes administrative
  Complexité technique

- Changer de configuration , gérer les services/serveurs devient impossible avec le parc s'aggrandissant

- En cas de panne, réparer les serveurs un à un est inefficace

- Montée en échelle difficile et épuisant

7 - 2 Automatiser la configuration des INFRA

Solution:

Fournir, et configurer et mettre à jours les serveurs et l'INFRA de façon automatique

Provisioning

Configuration Management

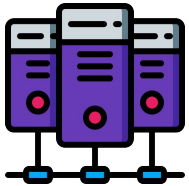
7 - 3 ANSIBLE

Open source

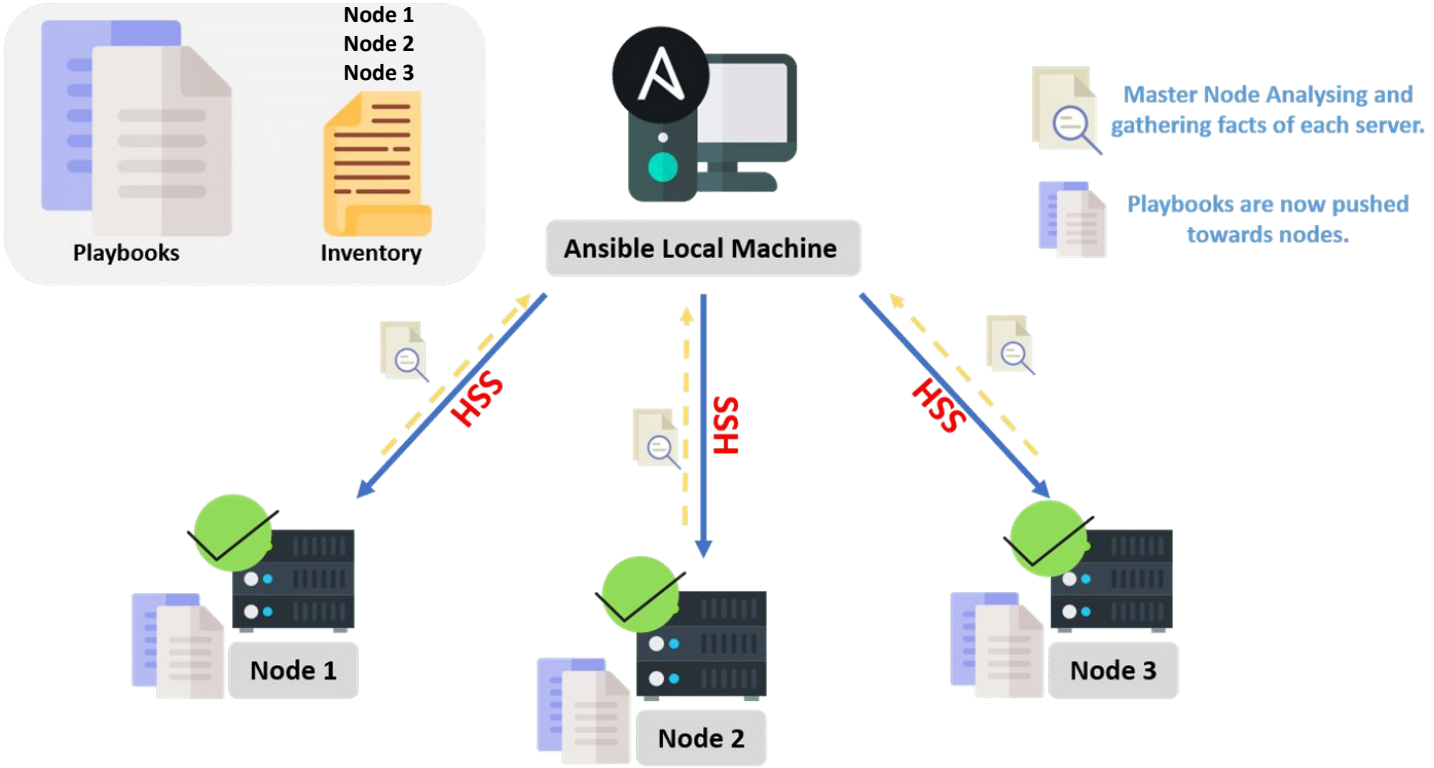
Automatisation :

- configurations
- déploiement

Gestion de la configuration de plusieurs noeuds (machines)



7 - 3 - a ANSIBLE - Comment ça marche ?



7 - 3 - a ANSIBLE - Comment ça marche ?

Control machine:
Machine de contrôle

C’est celle où réside
l’Installation Ansible

Besoins:
Python
Client SSH
OS non Windows
Paire de clés

Remote machine:
Machine distante contrôlée

Besoins:
Python
Serveur SSH

7 - 3 - b ANSIBLE - Playbook

Contient la liste des tâches à exécuter sur les machines à contrôler

Ensemble d'étapes exécutées de façon séquentielle

Écrit au format YAML

L'exécution d'un playbook => un ***'play'***

7 - 3 - b ANSIBLE - Playbook

```
---
- name: install db servers play
  hosts: databases
  tasks:
  - name: Install server and client
    command: apk add mysql mysql-client

  - name: Pause 10 seconds for installation process to finish
    pause: seconds=10

  - name: Setup database
    command: /etc/init.d/mariadb setup

  - name: Start the server
    command: /etc/init.d/mariadb start
```

Nom du playbook

Machines où exécuter les tâches

Liste des tâches à exécuter

Nom de la tâche

instruction à exécuter

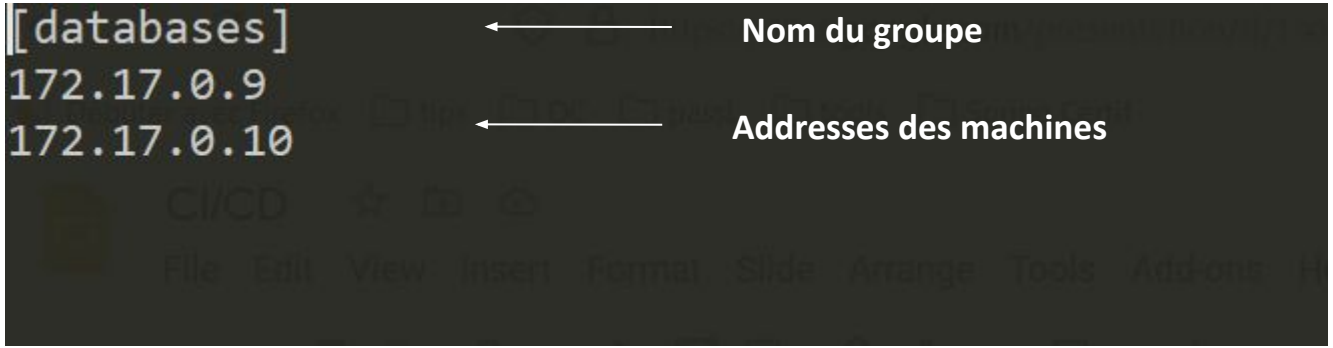
7 - 3 - b ANSIBLE - Inventory

Contient la liste des machines à controller

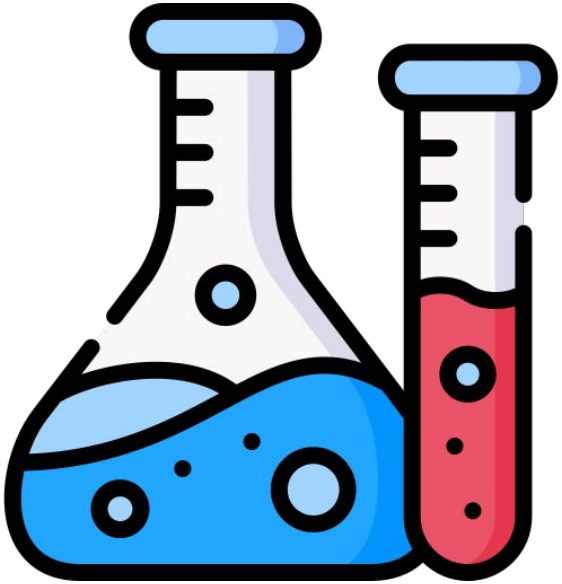
Possibilité de créer des groupes afin d'en faire la cible d'une tâche dans le playbook

Simple format texte

7 - 3 - b ANSIBLE - Inventory



7 - 4 Lab: Ansible



- Configuration des machines de contrôle
- Configuration des serveurs
- Installation Ansible
- Définition d’un inventory
- Définition d’un Playbook
- Exécution