

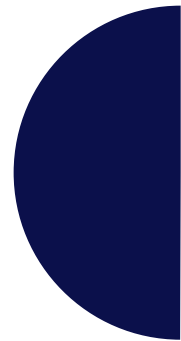


# MODULE :

## Le Cloud Computing & DevOps

Pr. F. Benabbou  
Master DSBD  
2024-2025

Faculté des Sciences Ben M'Sik Casablanca





# TABLE OF CONTENTS

## 01 CLOUD COMPUTING

- Introduction générale
- La Virtualisation
- Les concepts de base du Cloud Computing
- Technologies émergentes du CC : Edge, Fog, ...
- Étude de cas et projet pratique

## 02 DevOps & Cloud

- La philosophie DeVops
- Version control systems (git)
- **Intégration Continue CI**
- **Déploiement Continu CD**
- **Tests automatisés**
- Infrastructure en tant que Code (IaC)
- Surveillance et Journalisation
- Étude de cas et projet pratique



02

CI/CD/CT



# C'est quoi Jenkins?



- ✓ Jenkins est une plateforme, écrite en java, open source d'automatisation de l'intégration continue et du déploiement continu.
- ✓ Il permet aux équipes de développement de construire, tester et déployer automatiquement leur code de manière efficace.

# Histoire de Jenkins



Kohsuke Kawaguchi

2004



**Hudson:** un outil interne pour automatiser les tâches de construction et de test

2011



Lancement  
D'un nouveau  
projet : **Jenkis**

2019



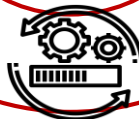
**Jenkins X** a été  
accepté par la  
foundation Cloud  
Native Computing  
Foundation  
(CNCF)

2010



Conflit entre l'équipe  
de développement  
d'Hudson et Oracle

2016



Dernières mises à  
jour officielles de  
**Hudson**

2025



plus de 10 millions  
de téléchargement par  
an Jenkins à travers  
le monde

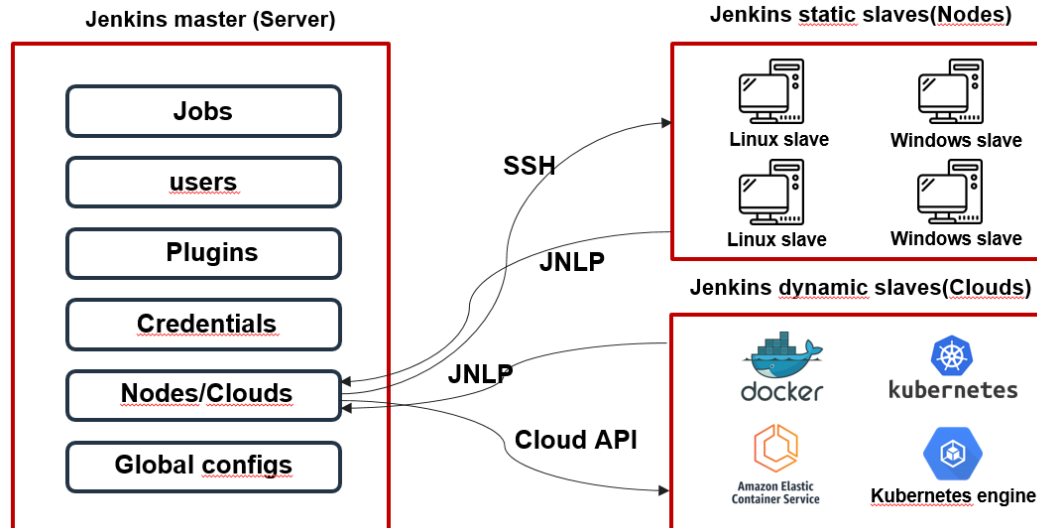


# Architecture de Jenkins

- L'architecture de Jenkins, repose sur le modèle maître-esclave et comprend deux principaux composants Jenkins Master (ou Controller) et Jenkins Agents (ou Nodes/Slaves).
- Ce modèle permet de distribuer la charge de travail en exécutant des tâches sur des machines distinctes, appelées nœuds ou agents.
- Jenkins Master
  - Le serveur principal qui orchestre les tâches et gère les interactions utilisateur faites via une interface web, il permet de configurer les objets, de gérer les utilisateurs et les autorisations, et de visualiser l'état des tâches en cours.
  - Gère les plugins pour étendre les fonctionnalités de Jenkins (par exemple, Git, Docker, Kubernetes).
- Jenkins Agents
  - Des machines (physiques ou virtuelles) qui exécutent les tâches assignées par le master, telles que la compilation du code, les tests et le déploiement.
  - Chaque esclave peut être configuré avec un environnement spécifique (système d'exploitation, outils, etc.) pour exécuter des tâches requérant des configurations particulières.

# Architecture de Jenkins

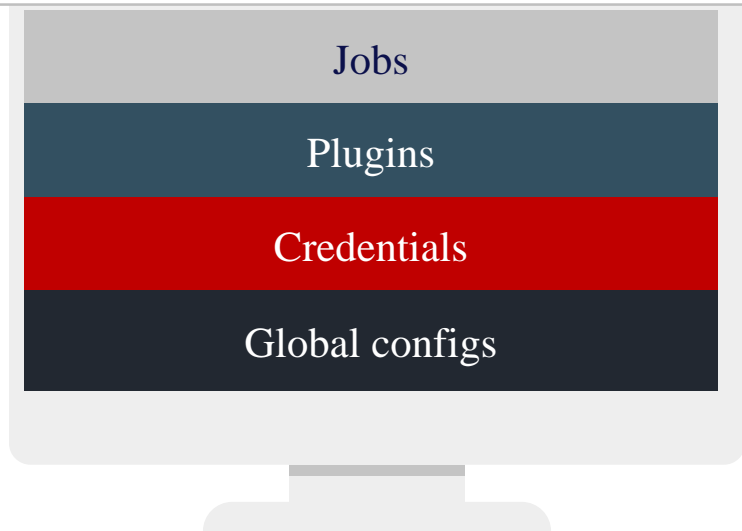
- La communication entre le maître et les agents se fait généralement via le protocole TCP/IP.
- Deux méthodes principales sont utilisées :
  - SSH : Le maître initie une connexion SSH vers l'agent pour contrôler son exécution.
  - JNLP (Java Network Launch Protocol) : L'agent se connecte au maître en utilisant JNLP, utile lorsque l'agent se trouve derrière un pare-feu ou un NAT.



# Architecture de Jenkins

- Exemple : Build, test, staging, deploy, notify, etc.
- Les Jobs sont isolés les uns des autres pour garantir la cohérence et la sécurité des opérations.
- Chaque job s'exécute dans son propre espace de travail isolé, appelé Workspace,

Un job en Jenkins est une série de tâches qui sont configurées pour être exécutées automatiquement sur un nœud esclave Jenkins





# Architecture de Jenkins

## Catégories de plugins :

- **SCM** : Git, Subversion, Mercurial.
- **Build Tools** : Maven, Gradle, Ant.
- **Testing** : JUnit, TestNG, Selenium.
- **Deployment** : Docker, Kubernetes, AWS, Azure.
- **Notification** : Slack, Email, Microsoft Teams.
- **Monitoring** : Grafana, Prometheus.

Global configs

Un job en Jenkins est une série de tâches qui sont configurées pour être exécutées automatiquement sur un nœud esclave Jenkins

Les plugins sont des modules complémentaires que peuvent être installer sur Jenkins et le permettent de communiquer avec d'autres outils.

# Architecture de Jenkins

## Exemples :

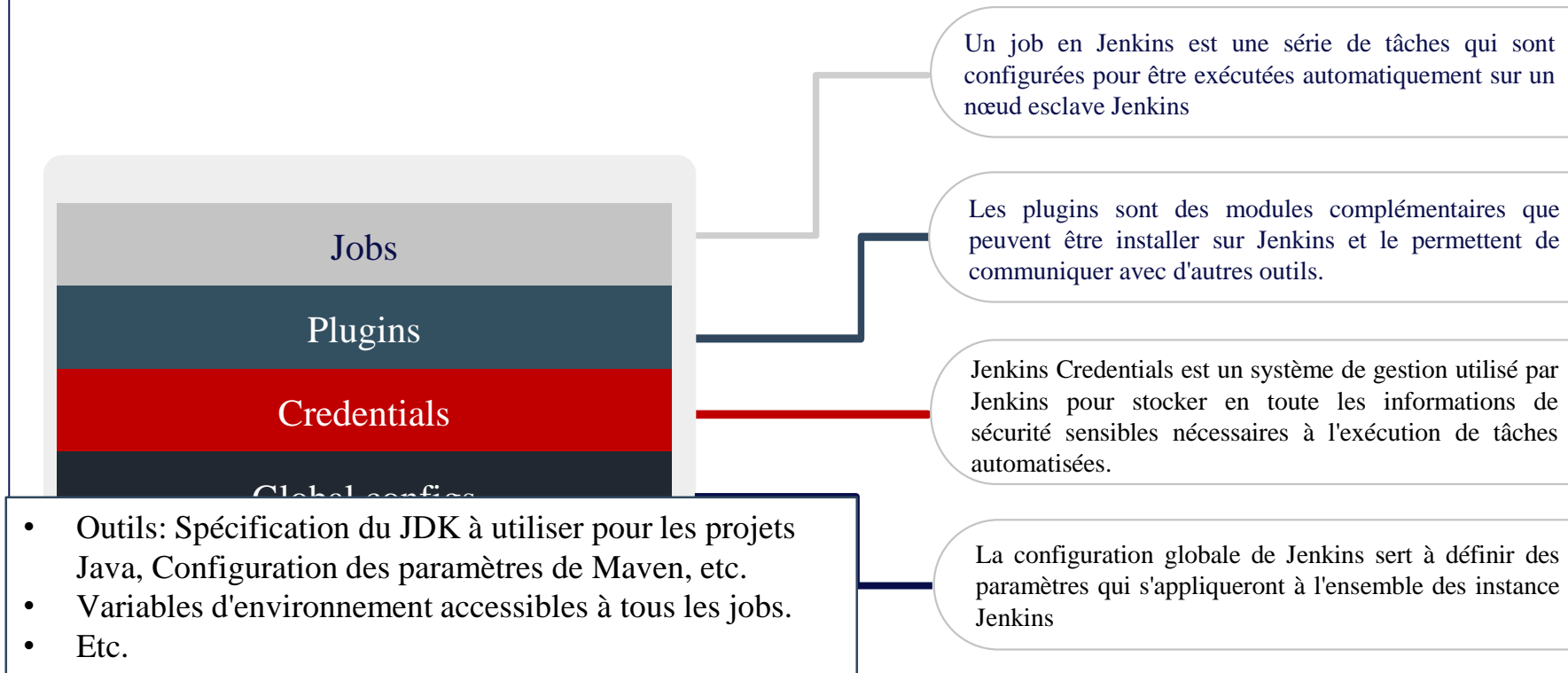
- Référentiels Git: pour accéder à des dépôts privés sur GitHub, GitLab, Bitbucket, etc.
- Serveurs de déploiement: pour se connecter à des serveurs d'application, des serveurs cloud (AWS, Azure, GCP), etc.
- Registres de conteneurs: pour accéder à des registres privés comme Docker Hub.
- etc.

Un job en Jenkins est une série de tâches qui sont configurées pour être exécutées automatiquement sur un nœud esclave Jenkins

Les plugins sont des modules complémentaires que peuvent être installer sur Jenkins et le permettent de communiquer avec d'autres outils.

Jenkins Credentials est un système de gestion utilisé par Jenkins pour stocker en toute les informations de sécurité sensibles nécessaires à l'exécution de tâches automatisées.

# Architecture de Jenkins



# Fichier de configuration : JenkinsFile

- ✓ JenkinsFile est un fichier texte utilisé pour créer un pipeline sous forme de code dans Jenkins.
- ✓ Il écrit en langage **Groovy**, qui est simple à écrire et lisible par l'homme.

Dans le Jenkinsfile on définit un pipeline Jenkins.  
Ce fichier est généralement stocké dans le répertoire racine du projet

Un Jenkinsfile est généralement structuré en plusieurs étapes.  
Chaque étape représente une phase du pipeline, comme la compilation, les tests, le déploiement, etc.

# Exemple de JenkinsFile

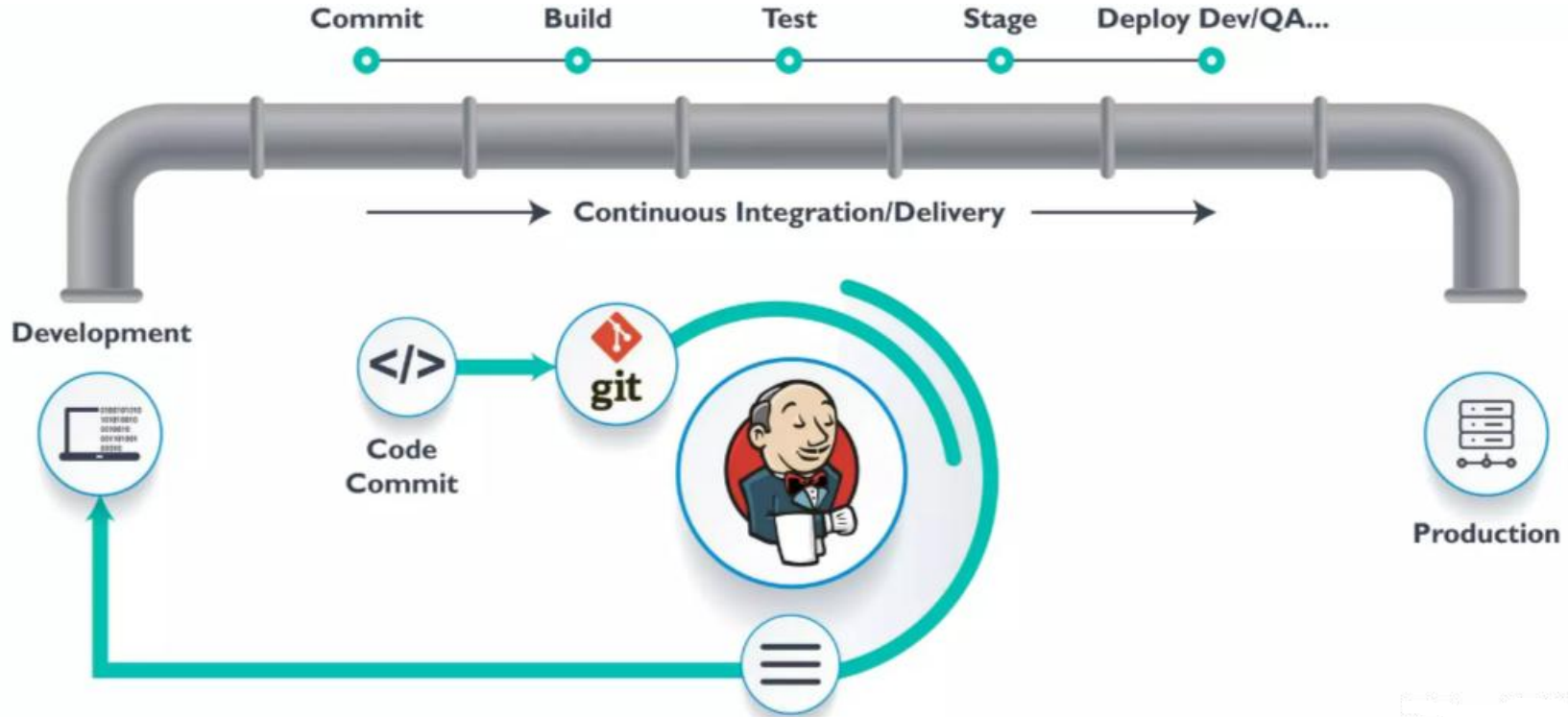
Le nom d'agent où on va exécuter le pipeline

```
pipeline {  
  agent any  
  stages {  
    stage('Build') {  
      steps {  
        sh 'mvn clean package'  
      }  
    }  
    stage('Test') {  
      steps {  
        sh 'mvn test'  
      }  
    }  
    stage('Deploy') {  
      steps {  
        sh 'docker build -t my-image .'  
        sh 'docker push my-image'  
      }  
    }  
  }  
}
```



Le job et ses étapes,  
sur un agent linux

# Fonctionnement de Jenkins

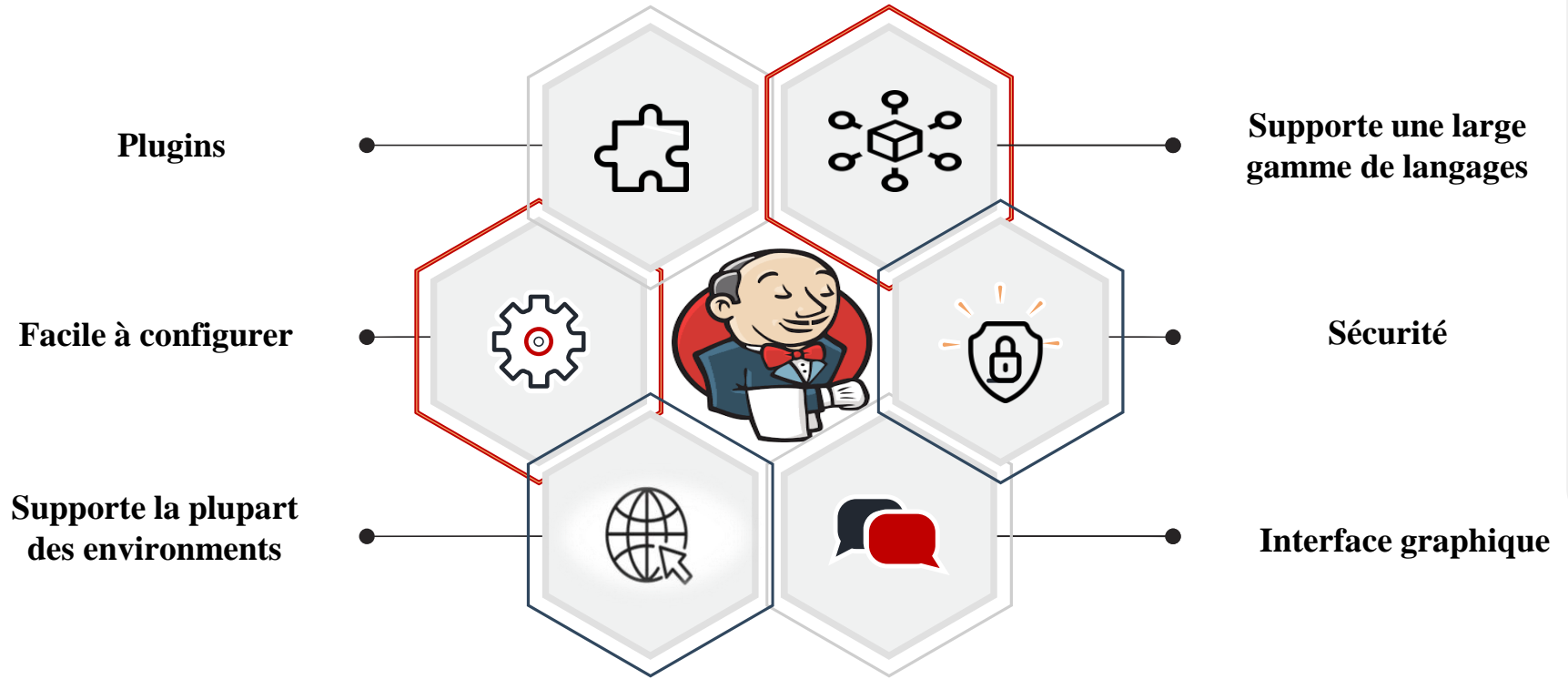


# Interface Jenkins

The screenshot displays the Jenkins web interface in a browser window. The address bar shows 'localhost:8080'. The Jenkins logo and name are at the top left, with a search bar labeled 'rechercher (CTRL+K)' to the right. Below the header, the 'Tableau de bord' (Dashboard) is visible, featuring a sidebar with links: '+ Nouveau Item', 'Historique des constructions', 'Administrer Jenkins', and 'Mes vues'. The main content area shows two expandable sections: 'File d'attente des constructions' and 'État du lanceur de compilations' (indicating 0 of 2 executors busy). At the bottom, a table header is partially visible with columns: 'S', 'M', 'Nom du projet ↓', 'Dernier succès', 'Dernier échec', and 'Dernière durée'.

S	M	Nom du projet ↓	Dernier succès	Dernier échec	Dernière durée
---	---	-----------------	----------------	---------------	----------------




# Avantages



En plus, Jenkins peut être déployé sur des plateformes cloud



# Jenkins VS autres outils de CI/CD

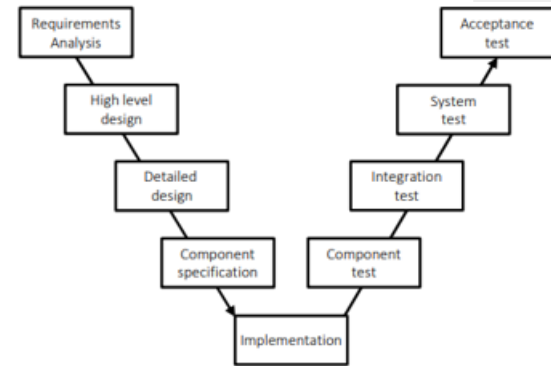
	 <b>Jenkins</b>	 <b>TeamCity</b>	 <b>circleci</b>	 <b>Bamboo</b>	 <b>GitLab</b>
Open source	Yes	No	No	No	No
Ease of use & setup	Medium	Medium	Medium	Medium	Medium
Built-in features	3/5	4/5	4/5	4/5	4/5
Integration	★★★★★	★★★	★★★★★	★★★★	★★★★★
Hosting	On premise & Cloud	On premise & Cloud	On premise	On premise & Bitbucket as Cloud	On premise & Cloud
Free version	Yes	Yes	Yes	Yes	Yes
Build agent license pricing	Free	From \$59 per month	From \$15 per month	From \$10 one-off payment	From \$19 per month per user
Supported OSs	Windows, Linux, macOS, Unix-like OS	Linux or MacOS	Windows, Linux, macOS, Solaris, FreeBSD and more	Windows, Linux, macOS, Solaris	Linux distributions: Ubuntu, Debian, CentOS, Oracle Linux



# Test Continue

# Tests Logiciels

- Dans les approches non agiles, les tests constituent la dernière phase du cycle de vie de développement Logiciel.
- Si un bug était détecté lors de la phase de test, il était difficile et coûteux de revenir en arrière et d'apporter les modifications nécessaires.
- Les testeurs de logiciels faisaient partie d'une équipe distincte, isolée de l'équipe de développement.



Le modèle en V de  
Boehm B.

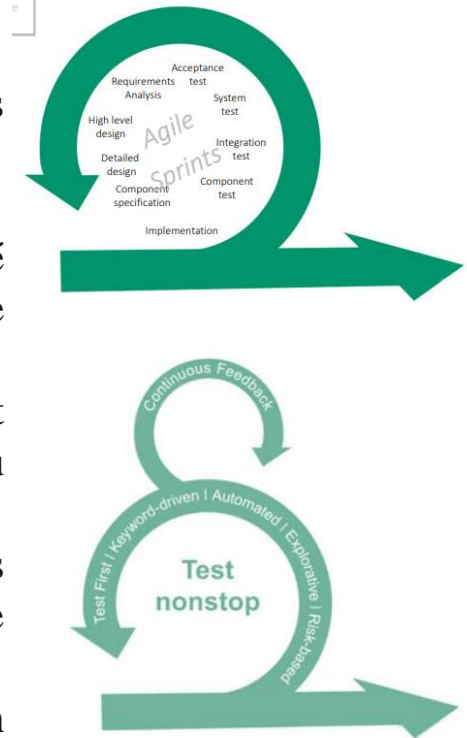


# L'approche de test « Shift Left »

- La culture DevOps adopte l'approche de test « Shift Left », qui contraste avec les environnements traditionnels où les tests ont lieu à la fin du développement.
- Cette approche pousse les tests vers la gauche, c'est-à-dire vers les premières étapes du processus de développement logiciel.
- Là aussi, les tests sont lancés dès le début du développement.
- Cette approche permet d'identifier les bugs le plus tôt possible et contribue de manière significative à l'amélioration de la qualité des logiciels.

# Test Automatisés dans CI/CD

- La livraison continue (CD) vise à fournir aux clients de nouvelles versions de code dans les plus brefs délais.
- Pour y parvenir, les tests automatisés jouent un rôle crucial.
- Ils permettent de vérifier de manière systématique et répétée la qualité du code à chaque modification, garantissant ainsi une livraison continue et fiable.
- Les tests couvrent chaque étape du cycle de vie du développement logiciel (SDLC), ce qui minimise les retours en arrière dans le cas où une erreur est détectée.
- De plus, les tests ne sont plus la responsabilité d'une seule équipe mais partagé entre l'ensemble de l'équipe ce qui permet à chacun de comprendre l'impact de chaque modification dans le code.
- Le test n'est pas une phase mais une activité, qui doit être exécutée en continu pendant le développement et les opérations !→ Tests continus



# Types de Tests Logiciels

- Dans un pipeline CI/CD, les tests automatisés sont intégrés dans le processus de développement.
- Chaque fois qu'une modification de code est poussée vers un dépôt, une série de tests est exécutée automatiquement.
- Ces tests couvrent différents niveaux :
  - Tests unitaires: Ils vérifient le fonctionnement correct de chaque unité de code indépendamment (fonctions, méthodes).
  - Tests d'intégration: Ils vérifient que les différentes parties du système interagissent correctement entre elles.
  - Tests fonctionnels: Ils vérifient que le logiciel répond aux exigences fonctionnelles spécifiées.
  - Tests d'acceptation: Ils valident que le logiciel répond aux besoins des utilisateurs finaux.
  - Tests de performance: ils évaluent la capacité d'une application à répondre aux exigences de performance sous une contraintes donnée : tests de charge, tests de stress, tests de résistance dans la durée, et tests de pic.

# Avantages des tests automatisés dans CI/CD

- **Gain de temps:** Les tests sont exécutés automatiquement, ce qui accélère le processus de validation du code.
- **Gain d'efforts:** Une fois les scripts de test créés, ils peuvent être réutilisés plusieurs fois sans effort supplémentaire, réduisant considérablement la charge de travail.
- **Augmentation de la fiabilité:** Les tests automatisés sont moins sujets aux erreurs humaines, garantissant des résultats cohérents et fiables.
- **Détection rapide des bugs:** Les bugs sont identifiés dès les premières étapes du développement, réduisant les coûts de correction.
- **Élargissement de la couverture des tests :** Ils permettent de tester un plus grand nombre de cas et scénarios.
- **Amélioration de la qualité Logiciel:** En détectant et corrigeant les problèmes plus tôt, les tests automatisés contribuent à améliorer la stabilité et la qualité globale des applications.
- **Support multi-environnements :** On peut tester facilement le code sur différents environnements, systèmes d'exploitation ou configurations, augmentant la portabilité et la compatibilité.

# Suivi des performances à l'aide d'indicateurs

- L'utilisation de mesures pour évaluer le succès ou l'échec des tests est l'une des bonnes pratiques.
- Exemples d'indicateurs clés :
  - Nombre de cas de test réussis ou échoués
  - Nombre de bugs identifiés
  - Fréquence des cas de test qui échouent
  - Temps d'exécution de la suite d'automatisation
- Ces mesures fournissent des informations sur les domaines les plus vulnérables aux défaillances, et les tests continus permettent d'obtenir des valeurs immédiates pour ces domaines.
- Les mesures du temps d'exécution des tests aident les ingénieurs en automatisation à identifier de meilleures façons d'écrire des cas de test qui augmentent les performances.



# Outils d'automatisation des tests

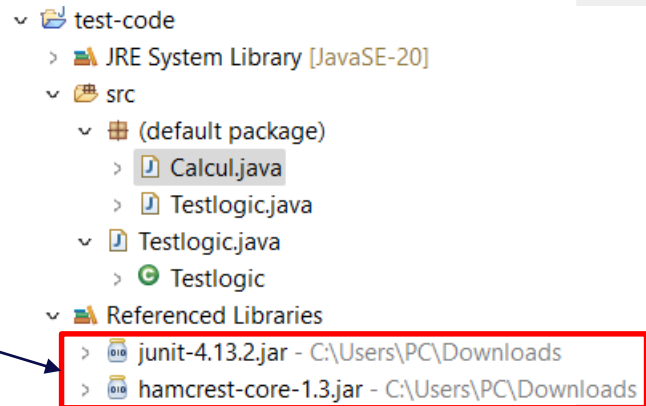
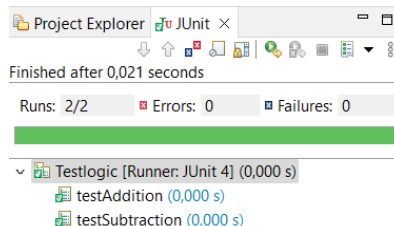
- **Selenium** est un framework open-source qui automatise principalement les tests d'applications web. Il s'agit d'une suite de logiciels destinés à répondre à différents besoins en matière de tests. Les scripts Selenium peuvent être écrits dans plusieurs langages de programmation.
- **Katalon Studio**, classé parmi les meilleurs logiciels de tests automatisés, possède des versions gratuites et payantes. Il peut être utilisé pour des tests automatisés sur le web, les API, les ordinateurs de bureau et les téléphones portables. Il prend en charge de nombreuses plateformes.
- **JMeter** est un logiciel open-source basé sur Java et utilisé pour les tests de performance et de charge, Il peut être utilisé pour tester de nombreux types de protocoles.
- **SoapUI** aide à tester les API REST, SOAP et GraphQL.
- Il s'agit d'un autre outil d'automatisation open-source et multiplateforme doté d'une interface utilisateur graphique (GUI) pratique.

# Exemple avec JUNIT

- Commencer par ajouter les deux fichiers jar au projet
- Ajouter une classe (Calcul) au projet qui fait l'addition et la soustraction

```
*Calcul.java x Testlogic.java
1 public class Calcul {
2
3     public int add(int a, int b) {
4         return a + b;
5     }
6
7     public int subtract(int a, int b) {
8         return a - b;
9     }
10 }
```

- Créer une autre classe pour y mettre les test automatisés (Testlogic).



```
Calcul.java x Testlogic.java x
1 import static org.junit.Assert.*;
2 import static org.junit.Assert.assertEquals;
3 import org.junit.Before;
4 import org.junit.Test;
5
6 public class Testlogic {
7     @Before
8     public void setUp() throws Exception {
9         System.out.println("before Calcul");
10    }
11    @Test
12    public void testAddition() {
13        Calcul calculator = new Calcul();
14        int result = calculator.add(2, 3);
15        assertEquals("Addition of 2 and 3 should be 5", 5, result);
16    }
17
18    @Test
19    public void testSubtraction() {
20        Calcul calculator = new Calcul();
21        int result = calculator.subtract(5, 3);
22        assertEquals("Subtraction of 3 from 5 should be 2", 2, result);
23    }
24 }
```