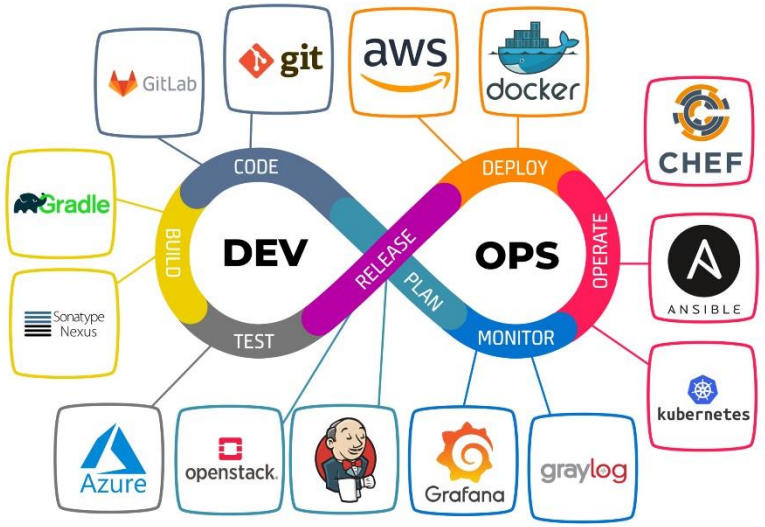
INITIATION AU DEVOPS

CODE :B3COM0105 **Durée** : 20H



Ing. BOGNI-DANCHI T.

OBJECTIFS PEDAGOGIQUES

Prérequis : ***Connaissances de base en administration système et développement logiciel***

Ce cours a pour objectif d'initier les étudiants aux concepts et pratiques de DevOps. Les étudiants apprendront à combiner le développement logiciel et l'administration système pour améliorer la collaboration entre les équipes et automatiser les processus de livraison continue. Le cours couvre les outils, les processus et les méthodologies utilisés dans un environnement DevOps.

À l'issue de ce cours, l'étudiant doit être capable de :

- Comprendre les principes fondamentaux de DevOps et leur rôle dans le développement logiciel.
- Utiliser des outils de gestion de versions, d'intégration continue et de déploiement continu.
- Automatiser les tests et les déploiements à l'aide d'outils comme Jenkins, Docker, et Kubernetes.
- Mettre en place des pipelines CI/CD (Continuous Integration/Continuous Delivery) pour optimiser les processus de développement.

PLAN

SECTION1: Introduction à DevOps

- Historique
- Définition et Contexte
- Principes Fondamentaux
- Cycle de vie du DEVOPS
- Les Outils du Devops
- TP : Préparation de l'environnement
  - ✓ Installation de WSL
  - ✓ Installation de la VM et du système

SECTION2: Le versioning et Conteneurisation

- 1) Le versioning
  - Notion sur les SCM et SVC
  - Workflow de versionning
  - Introduction Git
- 2) La conteneurisation

- TP (3h)
  - ✓ Installation de GIT
  - ✓ Création de dépôts
  - ✓ Branches, merge, pull requests
  - ✓ Exercices sur GitHub/GitLab
  - ✓ Gestion collaborative de code source

PLAN

SECTION3:

- Principes des conteneurs
- Docker : architecture et concepts
- Comparaison machines virtuelles vs conteneurs
- Travaux Pratiques (3h)
  - ✓ Installation de Docker
  - ✓ Création de Dockerfile
  - ✓ Construction et gestion d'images
  - ✓ Docker Compose
  - ✓ Déploiement d'applications conteneurisées

SECTION4: Intégration continue

- Gestion des erreurs et notifications
- Concepts d'intégration continue
- Outils : Jenkins, GitLab CI
- Principes des pipelines CI/CD
- Travaux Pratiques (3h)
  - ✓ Installation de Jenkins
  - ✓ Configuration de Jenkins
  - ✓ Création de premier pipeline
  - ✓ Tests automatiques
  - ✓ Déploiement automatisé

PLAN

- SECTION5: Orchestration et gestion des configuration
  - Architecture Kubernetes
  - Orchestration de conteneurs
  - Concepts : pods, deployments, services
  - Notion sur Ansible
  - Travaux Pratiques (3h)
    - ✓ Installation d'un cluster Kubernetes
    - ✓ Installation et configuration de Ansible
    - ✓ Déploiement d'applications
    - ✓ Mise à l'échelle
    - ✓ Gestion des mises à jour
    - ✓ Configuration de services
    - ✓ Gestion des configurations

- SECTION6: PROJET FINAL (Par l'étudiant)

SECTION 3 : INTÉGRATION CONTINUE

- 1- Introduction

- Définition

L'intégration continue (Continuous Integration, **CI**) est une pratique de développement logiciel qui consiste à intégrer fréquemment (souvent plusieurs fois par jour) le code produit par les développeurs dans un dépôt central partagé. Chaque intégration est automatiquement testée pour détecter rapidement les erreurs et maintenir une base de code stable.

Les objectifs de l'intégration continue

- **Détection rapide des erreurs :** Identifier les conflits ou les bugs dès qu'ils se produisent, avant qu'ils ne deviennent coûteux à corriger.
- **Amélioration de la qualité du code :** Grâce à des tests automatisés, on garantit que les modifications respectent les normes et ne cassent pas les fonctionnalités existantes.
- **Collaboration accumulée :** Les développeurs travaillent ensemble sur un dépôt partagé, ce qui évite les conflits importants de fusion (merge).
- **Automatisation :** Les tâches répétitives comme la compilation, les tests, et l'analyse du code sont automatisées.

SECTION 3 : INTÉGRATION CONTINUE

1- Introduction

- **Étapes du processus d'intégration continue**

1.Commit des changements : Les développeurs intègrent leur code dans un dépôt central (par exemple, Git).
2.Déclenchement d'un pipeline CI : Chaque nouveau commit déclenche automatiquement une série d'étapes dans un système CI.
3.Compilation du code : Le code est compilé pour vérifier qu'il n'y a pas d'erreurs syntaxiques ou de dépendances manquantes.
4.Exécution des tests automatisés :

1. Tests unitaires : Vérification des fonctions et classes individuellement.
2. Tests d'intégration : Validation de l'interaction entre plusieurs composants.
3. Tests fonctionnels : Vérification des fonctionnalités globales.

5.Analyse statique du code : Outils comme SonarQube analysent la qualité du code pour identifier les problèmes de sécurité, de performance, ou de lisibilité.
6.Génération d'un rapport : Le système CI génère un rapport détaillé indiquant le statut de chaque étape (succès/échec).

SECTION 3 : INTÉGRATION CONTINUE

2- Intégration Continue et la Compilation(Build)

Une fois que la compilation est configurée comme un script exécutable avec une seule commande, l'intégration continue peut être réalisée comme suit :

- Le serveur CI est notifié des nouveaux commits ou les vérifie périodiquement.
- Lorsqu'un nouveau commit est détecté, le serveur CI le récupère.
- Le serveur CI exécute les scripts de compilation.
- Si la compilation est réussie, le serveur CI exécute les tests automatisés, comme décrit précédemment et dans la section suivante.
- Le serveur CI fournit les résultats de ses activités à l'équipe de développement (par exemple, via une page web interne ou un e-mail). Un concept important dans l'IC est appelé "casser la compilation".

Un commit est considéré comme cassant la compilation si la procédure de compilation/compilation échoue, ou si les tests automatisés déclenchent par celui-ci violent une gamme de valeurs acceptables définies pour certaines métriques. Par exemple, oublier d'ajouter un nouveau fichier dans un commit mais modifier d'autres fichiers qui supposent la présence d'un nouveau fichier cassera la compilation. Les tests peuvent être grossièrement classés en critiques (une seule défaillance de test entraînerait la cassure de la compilation) et moins critiques (seulement un pourcentage de tests échoués supérieur à un seuil fixé entraînerait la cassure de la compilation). Toutes les métriques peuvent être résumées en un résultat binaire : Est-ce que votre compilation est bonne (suffisamment bonne) ? (c'est-à-dire une compilation non cassée ou verte) ; ou est-ce que votre compilation n'est pas bonne (suffisamment bonne) ? (c'est-à-dire une compilation cassée ou rouge).

SECTION 3 : INTÉGRATION CONTINUE

- **2- Intégration Continue et la Compilation(Build)**

Casser la compilation signifie que d'autres membres de l'équipe sur la même branche ne peuvent pas non plus compiler. Ainsi, les tests d'intégration continuent sont effectivement arrêtés pour une grande partie de l'équipe. Corriger la compilation devient un élément de haute priorité. Certaines équipes ont un "chapeau que j'ai cassé la compilation" qui doit porter un membre de l'équipe jusqu'à ce que la compilation soit corrigée afin de souligner l'importance de ne pas casser la compilation. Le statut des tests peut être affiché de différentes manières. Certaines équipes utilisent des widgets électroniques (comme des lampes à laver), ou ont de grands écrans visibles affichant des feux rouges/verts pour chaque composant. D'autres équipes utilisent des notifications de bureau, particulièrement lorsqu'elles sont situées sur le site d'un client, où le client pourrait devenir nerveux si une grande lumière rouge apparaît. Enfin, si votre projet est divisé en plusieurs composants, ces composants peuvent être compilés séparément. En gestion de version, ils peuvent être conservés comme un seul code source.

SECTION 3 : INTÉGRATION CONTINUE

3- Introduction au CI avec jenkins

Jenkins est un programme open-source écrit en Java avec une multitude de plugins créés pour l'intégration continue. Jenkins est utilisé pour construire et tester votre projet de manière continue. Cette pratique permet aux développeurs de tester et de modifier rapidement leur logiciel. Quant aux utilisateurs, ca leur permet d'avoir rapidement la version la plus à jour de votre logiciel. Ainsi Jenkins vous permet de délivrer votre logiciel de manière continue en intégrant une variété d'outils de déploiement et de tests.

Avec l'aide de Jenkins, les entreprises peuvent produire des logiciels à une plus grande fréquence grâce aux merveilles de l'automatisation. Jenkins est capable d'intégrer plusieurs processus du cycle de vie d'une application, que ce soit la compilation, la documentation, les tests, les analyses, le déploiement et beaucoup plus.

C'est grâce aux plugins que Jenkins réussit à faire de l'intégration continue. Ces derniers permettent d'intégrer différentes étapes de DevOps en fonction des besoins spécifiques de chaque projet. Ainsi si vous voulez intégrer un outil en particulier, il vous suffit d'installer son plugin dans Jenkins.

Parmis les avantages de Jenkins, on peut citer :

- Une très grande communauté d'utilisateurs.
- Plus de 1000 plugins pour faciliter votre vie
- Vous pouvez créer vos propres plugins et les partager avec la communauté.
- La facilité d'installation.
- C'est écrit en Java, donc peut fonctionner dans la plupart des plateformes (Windows, Mac, Linux).
- C'est gratuit.

SECTION 3 : INTÉGRATION CONTINUE

4- Intégration Continue avec jenkins

- Architecture de Jenkins

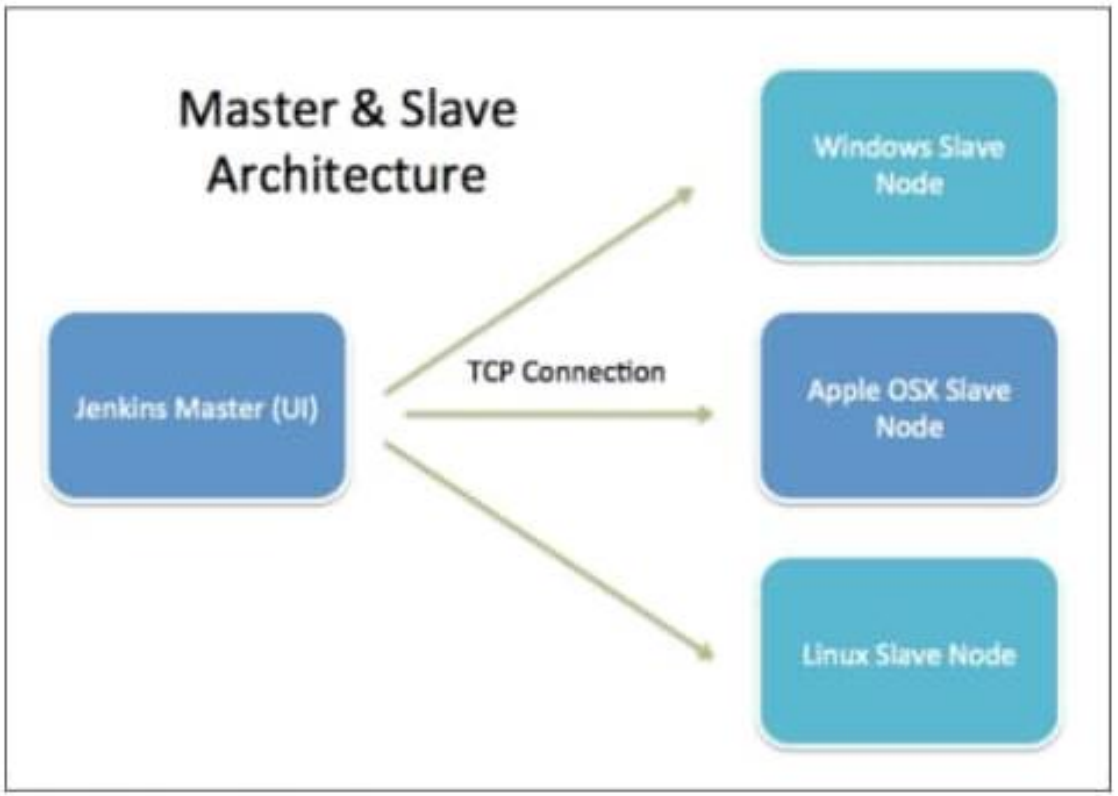
Jenkins suit une architecture maître-esclave pour gérer les builds distribués. Dans cette architecture, le maître et les esclaves communiquent via le protocole TCP/IP.

L'architecture de Jenkins comprend deux composants :

- **Maître/Serveur Jenkins**
- **Esclave/Node/Serveur de Build Jenkins**

Maître Jenkins

Le serveur principal de Jenkins est le Maître Jenkins. C'est un tableau de bord web alimenté par un fichier WAR. Par défaut, il fonctionne sur le port 8080. À l'aide du tableau de bord, nous pouvons configurer les tâches/projets, mais les builds sont exécutés sur les nœuds/esclaves. Par défaut, un nœud (esclave) est configuré et fonctionne sur le serveur Jenkins. Nous pouvons ajouter d'autres nœuds en utilisant une adresse IP, un nom d'utilisateur et un mot de passe via les méthodes SSH, JNLP ou WebStart.



SECTION 3 : INTÉGRATION CONTINUE

4- Intégration Continue avec jenkins

Les principales responsabilités du serveur ou maître sont :

- Planifier les tâches de build.
- Attribuer les builds aux nœuds/esclaves pour leur exécution réelle.
- Surveiller les nœuds/esclaves (les mettre en ligne ou hors ligne si nécessaire).
- Enregistrer et présenter les résultats des builds.
- Une instance Maître/Serveur de Jenkins peut également exécuter directement les tâches de build.

Esclave Jenkins

Un esclave Jenkins est utilisé pour exécuter les tâches de build attribuées par le maître. Nous pouvons configurer un projet pour qu'il s'exécute toujours sur une machine esclave particulière, sur un type spécifique de machine esclave, ou laisser Jenkins choisir le prochain esclave/nœud disponible.

Comme Jenkins est développé en Java et est indépendant de la plateforme, les Maîtres/Serveurs et les Esclaves/nœuds Jenkins peuvent être configurés sur n'importe quel serveur, y compris Linux, Windows et Mac.

Le diagramme ci-dessous est explicite. Il montre un Maître Jenkins gérant trois Esclaves Jenkins.

SECTION 3 : INTÉGRATION CONTINUE

- **4- Intégration Continue avec jenkins**

- **5- Le pipeline jenkins**

Le pipeline Jenkins est une fonctionnalité puissante qui permet d'automatiser l'ensemble du processus de développement logiciel, de la gestion du code source jusqu'à la mise en production. Il fournit une représentation visuelle des étapes du cycle de vie d'un build et permet de gérer des workflows complexes sous forme de scripts.

SECTION 3 : INTÉGRATION CONTINUE

5- Le pipeline jenkins

Un pipeline Jenkins est une suite de tâches qui s'exécutent de manière séquentielle ou parallèle pour atteindre un objectif spécifique, tel que :

- Construire un projet.
- Exécuter des tests.
- Déployer une application sur un environnement cible.

Il est défini dans un fichier texte appelé **Jenkinsfile** , qui est stocké dans le contrôle de version avec le code source du projet. Ce fichier suit une syntaxe DSL (Domain-Specific Language) basée sur Groovy.

Déclaration de pipeline :
Syntaxe simplifiée et structurée pour décrire les étapes d'un pipelin

SECTION 3 : INTÉGRATION CONTINUE

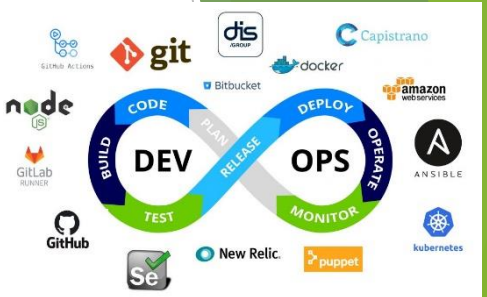
- **5- Le pipeline jenkins**

Déclaration de pipeline :
Syntaxe simplifiée et structurée pour décrire les étapes d'un pipeline

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Construire le projet'
            }
        }
        stage('Test') {
            steps {
                echo 'Exécuter les tests'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Déployer l'application'
            }
        }
    }
}
```

SECTION 3 : INTÉGRATION CONTINUE

6- TP



- TP 3.1 Installation

Installation de jenkins avec Docker-compose

- TP 3.2 : webapp

Création un projet free style dont le 'build' de notre application webapp pour le redéployer sur un autre port exemple: 8180

TP 3.3 : Build

Création d'un job pour un pour un projet free style dont le 'build' contient les paramètres pour s'exécuter un projet github

TP 3.4 : Test d'acceptance

Installer le plugin `**HTTP request**` pour sous jenkins faire le test du TP 3.3

TP 3.5 : Artefact

Execution du Build avec l'artefact deposé sur docker hub

TP 3.6 : Jenkins file

Pipeline depuis le jenkins file

Fin

Par Thomas BOGNI-DANCHI