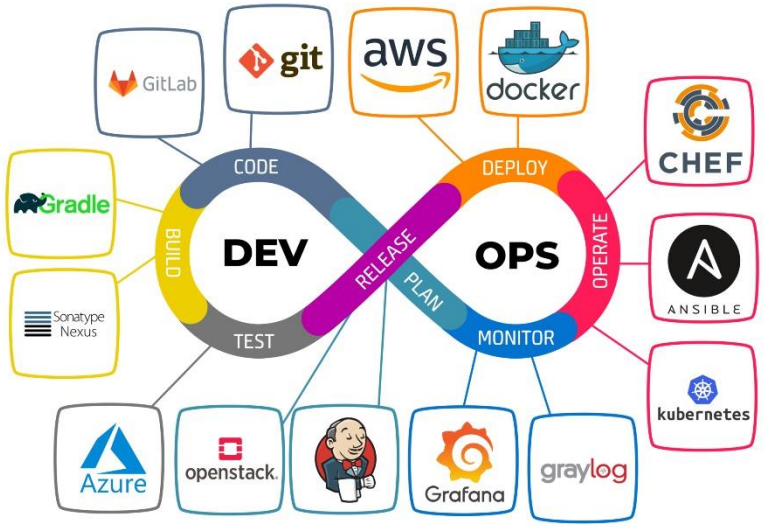
INITIATION AU DEVOPS

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 2 : VERSIONNING ET CONTENUERISATION

I- LE VERSIONING

- 1- Notion sur les SCM et SVC

Définition :
SCM (Source Control Management) définit les outils et pratiques pour gérer et suivre les modifications dans le code source d'un projet. Cela inclut le contrôle des versions, la gestion des branches et la collaboration entre développeurs.

SVC **(Source Version Control)** est une sous-catégorie du SCM qui se concentre spécifiquement sur le contrôle des versions, permettant de suivre les changements apportés à un projet, de revenir à des états antérieurs et de gérer des branches.

Fonctionnalités principales :
•Gestion des versions de fichiers.; Comparaison entre les versions.
•Marquage de versions stables (tags , Fusion (merge) et résolution de conflits.

Types de SVC :
1.Systèmes Centralisés :

1. Exemples : SVN, CVS.
2. Les fichiers sont stockés sur un serveur central, et les développeurs travaillent à partir de ce serveur.

2.Systèmes distribués :

1. Exemples : Git, Mercurial.
2. Chaque développeur possède une copie complète du dépôt, ce qui permet de travailler hors ligne.

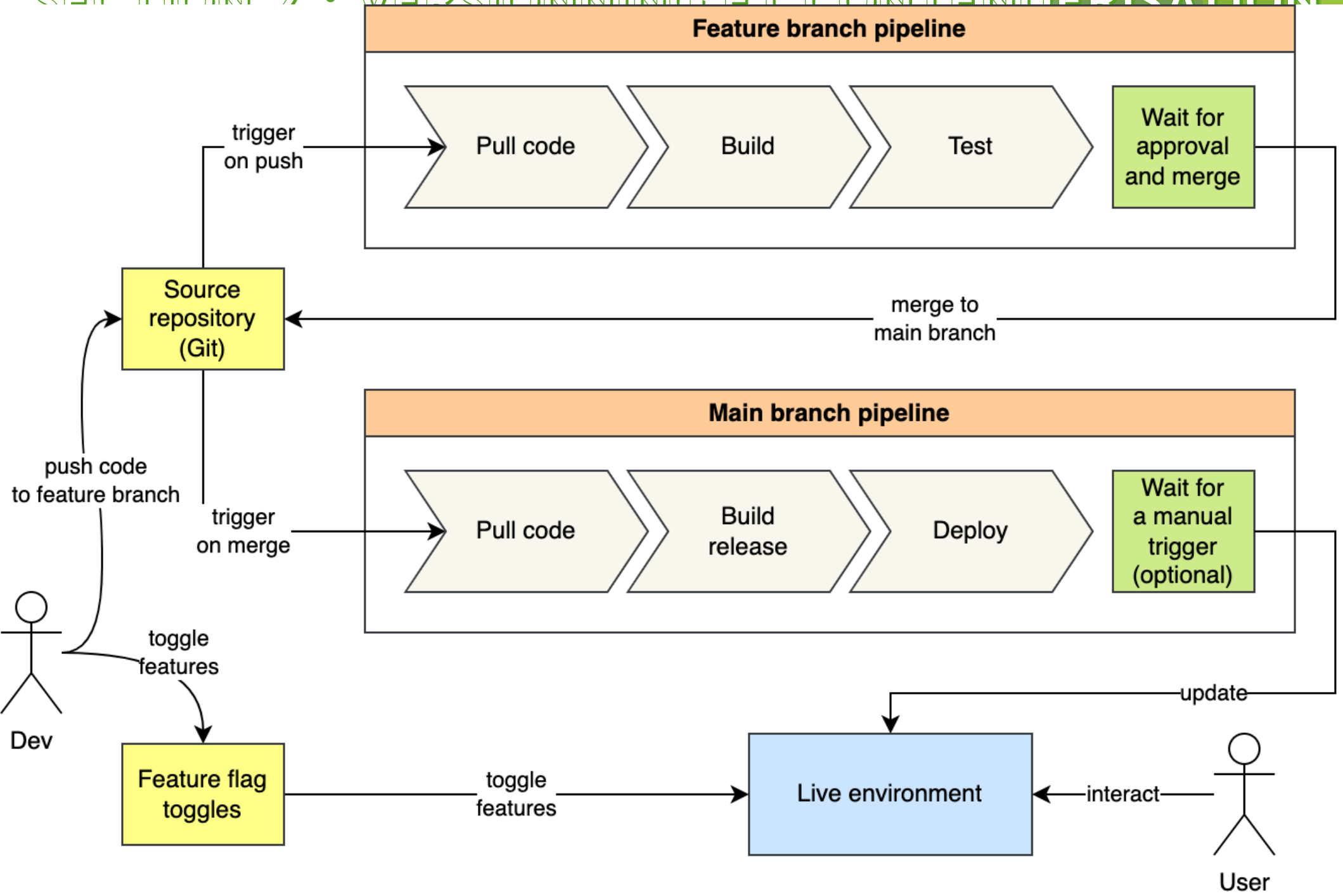
SECTION 2 : VERSIONNING ET CONTENUERISATION

I-2- Workflow du Versioning

Le **versioning** , ou **gestion des versions** , désigne le processus de gestion des différentes versions d'un fichier, d'un logiciel ou d'un projet au fil du temps. Il permet de suivre, d'enregistrer et de gérer les modifications effectuées sur des fichiers ou un ensemble de fichiers, tout en conservant un historique détaillé de ces changements.

Objectifs principaux du versioning :

1. **Suivi des modifications** : Conserver un historique de toutes les modifications aux fichiers.
2. **Collaboration efficace** : Permettre à plusieurs personnes de travailler simultanément sur les mêmes fichiers sans conflit.
3. **Revenir à une version précédente** : Faciliter la restauration d'une version antérieure si une modification cause des problèmes.
4. **Audit et traçabilité** : Identifiant qui a effectué des modifications, à quel moment et pourquoi.
5. **Gestion des branches** : Travailler sur différentes fonctionnalités ou correctifs déterminant les uns des autres avant de fusionner les modifications.


Feature branch pipeline
Pull code
Build
Test
Wait for approval and merge
trigger on push
Source repository (Git)
merge to main branch
Main branch pipeline
Pull code
Build release
Deploy
Wait for a manual trigger (optional)
push code to feature branch
trigger on merge
Dev
toggle features
update
Feature flag toggles
toggle features
Live environment
interact
User

SECTION 2 : VERSIONNING ET CONTENUERISATION

I-3- Introduction à Git

- Configuration de GIT

- Définir votre nom d'utilisateur : *git config --global user.name "Mon Nom"`*

- Définir votre email : *git config --global user.email "mon@email.com"`*

- Configurer l'éditeur par défaut pour les commit messages : *git config --global core.editor nano*

- Ignorer les fichiers temporaires globalement : *git config --global core.excludesfile ~/.gitignore`*

- Activer la colorisation de la sortie : *git config --global color.ui auto`*

- Mémoriser les identifiants de manière permanente : *git config --global credential.helper store`*

- Les commande de base de GIT

Voici quelques commandes Git fondamentales :

- **git init** : Initialiser un nouveau dépôt Git
- **git clone** : Cloner un dépôt Git existant
- **git add** : Ajouter des fichiers à l'index
- **git commit** : Valider les changements dans le dépôt local

I-3- Introduction à Git

- Les commande de base de GIT

- **git push** : Pousser les commits vers le dépôt distant
- **git pull** : Récupérer les nouveaux commits depuis le dépôt distant
- **git status** : Voir l'état des fichiers et des commits
- **git log** : Voir l'historique des commits
- **git branch** : Lister, créer ou supprimer des branches
- **git checkout** : Changer de branche ou restaurer des fichiers
- **git merge** : Fusionner des branches
- **git tag** : Marquer des commits avec des tags
- **git reset** : Annuler des commits locaux
- **git revert** : Annuler un commit via un nouveau commit
- **git stash** : Mettre de côté temporairement des modifications
- **git diff** : Différences entre commits, branches, working tree

- Afficher toutes les configurations globales : *git config --global -l`*

Les configurations globales s'appliquent à tous les dépôts Git pour l'utilisateur actuel.

II- LA CONTENEURISATION

II-1- Introduction

Docker est un moteur open-source qui automatise le déploiement d'applications dans des conteneurs. Il a été développé par l'équipe de Docker, Inc (anciennement dotCloud Inc, un acteur précoce du marché de la Plateforme-as-a-Service ou PaaS), et publié par leurs soins sous la licence Apache 2.0.
Alors, qu'est-ce qui rend Docker si spécial ? Docker ajoute un moteur de déploiement d'applications par-dessus un environnement d'exécution de conteneurs virtualisés. Il est conçu pour fournir un environnement léger et rapide pour exécuter votre code, ainsi qu'un workflow efficace pour faire passer ce code de votre ordinateur portable à votre environnement de test, puis en production. Docker est incroyablement simple. En effet, vous pouvez commencer à utiliser Docker sur un hôte minimal ne faisant tourner qu'un noyau Linux compatible et un binaire Docker. La mission de Docker est de fournir :

Une façon simple et légère de modéliser la réalité
Docker est rapide. Vous pouvez "Dockeriser" votre application en quelques minutes. Docker s'appuie sur un modèle de copie à l'écriture, de sorte que modifier votre application est également incroyablement rapide : seul ce que vous voulez changer est modifié.
Vous pouvez ensuite créer des conteneurs exécutant vos applications. La plupart des conteneurs Docker se lancent en moins d'une seconde.

SECTION 2 : VERSIONNING ET CONTENUERISATION

II-1- Introduction

Selon **Amazon ,La conteneurisation** est un processus de déploiement logiciel qui regroupe le code d'une application avec tous les fichiers et bibliothèques dont elle a besoin pour s'exécuter sur n'importe quelle infrastructure. Traditionnellement, pour exécuter n'importe quelle application sur votre ordinateur, vous deviez installer la version correspondant au système d'exploitation de votre machine. Par exemple, vous deviez installer la version Windows d'un progiciel sur un ordinateur Windows. Toutefois, avec la conteneurisation, vous pouvez créer un progiciel unique, ou conteneur, qui s'exécute sur tous les types d'appareils et de systèmes d'exploitation.

Docker est un moteur open-source qui automatise le déploiement d'applications dans des conteneurs. Il a été développé par l'équipe de Docker, Inc (anciennement dotCloud Inc, un acteur précoce du marché de la Plateforme-as-a-Service ou PaaS), et publié par leurs soins sous la licence Apache 2.0.
Alors, qu'est-ce qui rend Docker si spécial ? Docker ajoute un moteur de déploiement d'applications par-dessus un environnement d'exécution de conteneurs virtualisés. Il est conçu pour fournir un environnement léger et rapide pour exécuter votre code, ainsi qu'un workflow efficace pour faire passer ce code de votre ordinateur portable à votre environnement de test, puis en production. Docker est incroyablement simple. En effet, vous pouvez commencer à utiliser Docker sur un hôte minimal ne faisant tourner qu'un noyau Linux compatible et un binaire Docker. La mission de Docker est de fournir :

SECTION 2 : VERSIONNING ET CONTENUERISATION

II-1- Introduction

Une façon simple et légère de modéliser la réalité

Docker est rapide. Vous pouvez "Dockeriser" votre application en quelques minutes. Docker s'appuie sur un modèle de copie à l'écriture, de sorte que modifier votre application est également incroyablement rapide : seul ce que vous voulez changer est modifié. Vous pouvez ensuite créer des conteneurs exécutant vos applications. La plupart des conteneurs Docker se lancent en moins d'une seconde. Supprimer la surcharge de l'hyperviseur signifie également que les conteneurs sont hautement performants et que vous pouvez en intégrer davantage sur vos hôtes et faire le meilleur usage possible de vos ressources.

Une ségrégation logique des tâches

Avec Docker, les développeurs se concentrent sur leurs applications à l'intérieur des conteneurs, et les équipes opérationnelles se préoccupent de la gestion des conteneurs. Docker est conçu pour améliorer la cohérence en garantissant que l'environnement dans lequel vos développeurs écrivent du code correspond aux environnements dans lesquels vos applications sont déployées. Cela réduit le risque de problèmes de la phase de développement à la phase opérationnelle.

Un cycle de développement rapide et efficace

Docker vise à réduire le temps entre l'écriture du code et son test, son déploiement et son utilisation. Il cherche à rendre vos applications portables, faciles à construire et à collaborer.

SECTION 2 : VERSIONNING ET CONTENUERISATION

II-1- Introduction

Une façon simple et légère de modéliser la réalité

Docker est rapide. Vous pouvez "Dockeriser" votre application en quelques minutes. Docker s'appuie sur un modèle de copie à l'écriture, de sorte que modifier votre application est également incroyablement rapide : seul ce que vous voulez changer est modifié. Vous pouvez ensuite créer des conteneurs exécutant vos applications. La plupart des conteneurs Docker se lancent en moins d'une seconde. Supprimer la surcharge de l'hyperviseur signifie également que les conteneurs sont hautement performants et que vous pouvez en intégrer davantage sur vos hôtes et faire le meilleur usage possible de vos ressources.

Une ségrégation logique des tâches

Avec Docker, les développeurs se concentrent sur leurs applications à l'intérieur des conteneurs, et les équipes opérationnelles se préoccupent de la gestion des conteneurs. Docker est conçu pour améliorer la cohérence en garantissant que l'environnement dans lequel vos développeurs écrivent du code correspond aux environnements dans lesquels vos applications sont déployées. Cela réduit le risque de problèmes de la phase de développement à la phase opérationnelle.

Un cycle de développement rapide et efficace

Docker vise à réduire le temps entre l'écriture du code et son test, son déploiement et son utilisation. Il cherche à rendre vos applications portables, faciles à construire et à collaborer.

SECTION 2 : VERSIONNING ET CONTENUERISATION

II-1- Introduction

- **Conteneurisation & virtualisation**

Virtualisation

Principes

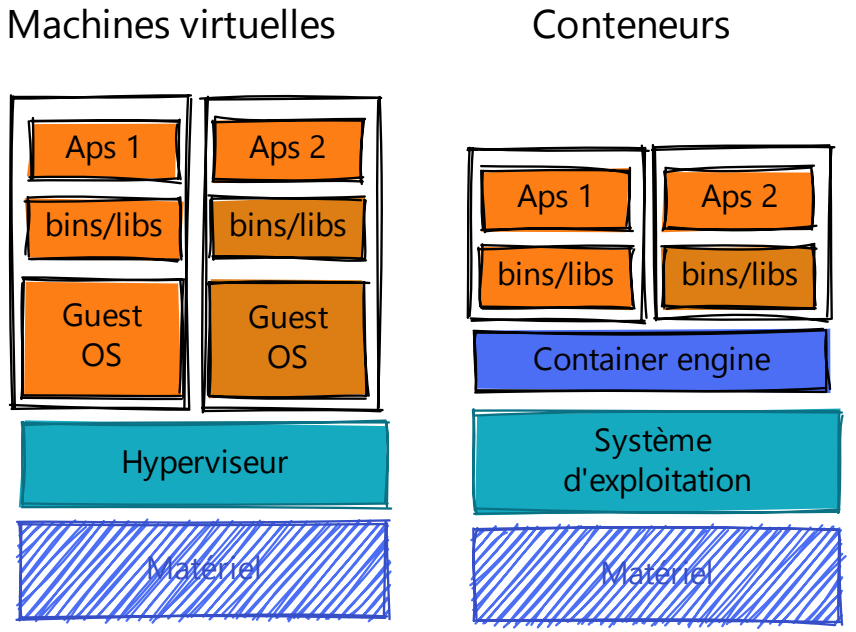
- Émulation complète d'un système informatique
- Chaque machine virtuelle (VM) possède son propre système d'exploitation complet
- Isolation totale entre les machines virtuelles

Caractéristiques techniques

- Utilise un hyperviseur (VMware, Hyper-V, VirtualBox)
- Charge système d'exploitation complète pour chaque VM
- Consommation importante de ressources
- Démarrage relativement lent (quelques minutes)

Avantages

- Isolation complète
- Possibilité d'exécuter différents systèmes d'exploitation
- Sécurité renforcée
- Migration facilitée des systèmes



Inconvénients

- Surcharge importante en ressources
- Performance réduite
- Consommation élevée d'espace disque
- Temps de démarrage long

SECTION 2 : VERSIONNING ET CONTENUERISATION

II-1- Introduction

- **Conteneurisation & virtualisation**

Conteneurisation

Principes

- Virtualisation légère au niveau du système d'exploitation
- Partage du noyau hôte entre les conteneurs
- Isolation des processus et des ressources applicatives

Caractéristiques techniques

- Utilise Docker, Kubernetes comme technologies principales
- Léger et rapide à déployer
- Démarrage quasi-instantané
- Portabilité maximale

Avantages

- Très faible consommation de ressources
- Démarrage rapide (secondes)
- Haute densité de déploiement
- Cohérence entre environnements
- Scalabilité simplifiée

Inconvénients

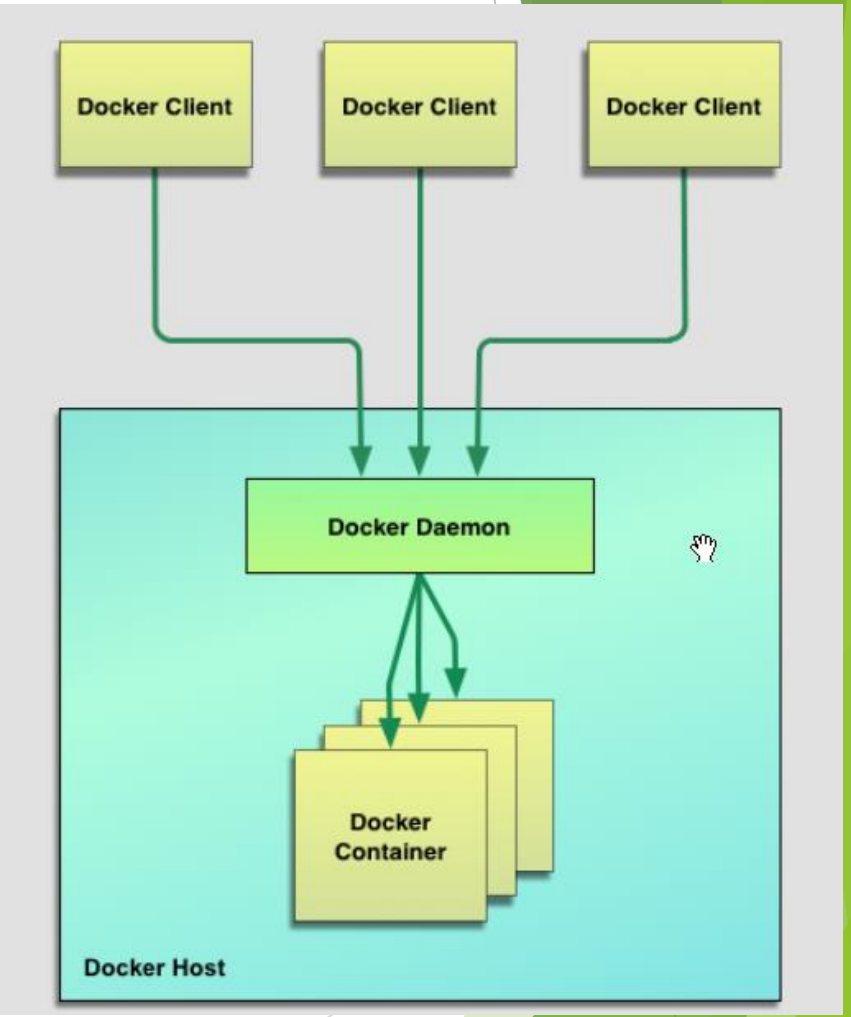
- Moins d'isolation que la virtualisation
- Limité aux systèmes d'exploitation similaires
- Sécurité potentiellement réduite

Critère	Virtualisation	Conteneurisation
Isolation	Complète	Processus
Ressources	Élevées	Minimales
Démarrage	Lent	Instantané
Portabilité	Limitée	Excellente
Overhead	Important	Minimal
Use cases	Infrastructure complexe	Microservices, DevOps

II-2-Les Composants principaux de Docker

Docker se compose de plusieurs éléments fondamentaux qui travaillent ensemble pour permettre la conteneurisation :

- Le client et le serveur Docker
- Images Docker
- Registres
- Conteneurs Docker



Client et serveur Docker

Docker est une application basée sur une architecture client-serveur. Le client Docker communique avec le serveur ou démonter Docker, qui se charge d'effectuer toutes les opérations. Docker est fourni avec un binaire de ligne de commande appelé **docker**, ainsi qu'une API RESTful complète.

Vous pouvez exécuter le démon Docker et le client sur la même machine, ou connecter votre client Docker local à un démon fonctionnant à distance sur une autre machine. L'architecture de Docker peut être représentée comme suit :

II-2-Les Composants principaux de Docker

Images Docker

Les images sont les éléments fondamentaux du monde Docker. Vous lancez vos conteneurs à partir d'images. Les images représentent la phase de **« construction »** dans le cycle de vie de Docker. Par exemple :

- Ajouter un fichier.
- Exécuter une commande.
- Ouvrir un port.

Vous pouvez considérer les images comme le **« code source »** de vos conteneurs. Elles sont hautement portables, peuvent être partagées, stockées et mises à jour. Dans ce livre, nous apprendrons à utiliser des images existantes ainsi qu'à créer nos propres images.

Les Registres

Docker stocke les images que vous avez créées dans des registres. Il existe deux types de registres : publics et privés. Docker, Inc. exploite le registre public des images, appelé **Docker Hub** . Vous pouvez créer un compte sur le Docker Hub et l'utiliser pour partager et stocker vos propres images. Le Docker Hub contient également, à ce jour, plus de 10 000 images créées et partagées par d'autres utilisateurs. Vous recherchez une image Docker pour un serveur web Nginx, le système PABX open source Asterisk ou une base de données MySQL ? Toutes ces images sont disponibles, ainsi qu'une multitude d'autres.

II-2-Les Composants principaux de Docker

Conteneurs

Docker vous permet de créer et de déployer des conteneurs à l'intérieur desquels vous pouvez emballer vos applications et services. Comme nous venons de l'apprendre, les conteneurs sont lancés à partir d'images et peuvent contenir un ou plusieurs processus en cours d'exécution. On peut considérer les images comme représentant l'aspect **construction ou emballage** de Docker, tandis que les conteneurs incarnent l'aspect **exécution ou fonctionnement** .

Un conteneur Docker est :

- Un format d'image.
- Un ensemble d'opérations standards.
- Un environnement d'exécution.

Docker s'inspire du concept du conteneur d'expédition standard, utilisé pour transporter des marchandises à travers le monde, pour modéliser ses propres conteneurs. Mais au lieu de transporter des biens physiques, les conteneurs Docker transportent des logiciels.

Chaque conteneur contient une image logicielle – son « contenu » – et, comme son équivalent physique, permet d'effectuer un ensemble d'opérations. Par exemple, un conteneur peut être créé, démarré, arrêté, redémarré ou supprimé.

Comme un conteneur d'expédition, Docker ne s'intéresse pas au contenu du conteneur lorsqu'il exécute ces actions. Par exemple, qu'il s'agisse d'un serveur web, d'une base de données ou d'un serveur applicatif, chaque conteneur est traité de la même manière.

SECTION 2 : VERSIONNING ET CONTENUERISATION

II-3-Dockerfile et les Commandes Docker

- **Le Dockerfile**

Un Dockerfile est un fichier texte qui contient un ensemble d'instructions permettant de construire une image Docker. C'est une sorte de "recette" pour créer un environnement logiciel standardisé et reproductible.

Instructions de Base

1.FROM

- •Définit l'image de base
- •Première instruction obligatoire
- •Exemple : *FROM ubuntu:20.04*

2.RUN

- •Exécute des commandes pendant la construction de l'image
- •Peut installer des packages, créer des répertoires
- •Exemple : *RUN apt-get update && apt-get install -y python3*

3.WORKDIR

- •Définit le répertoire de travail pour les instructions suivantes
- •Équivalent à cd dans un terminal
- •Exemple : *WORKDIR /app*

4.COPY

- •Copie des fichiers depuis l'hôte vers l'image
- •Exemple : *COPY . /app*

5.ADD

- •Similaire à COPY mais avec des fonctionnalités supplémentaires
- •Peut télécharger des fichiers distants
- •Décompresse automatiquement des archives

6.ENV

- •Définit des variables d'environnement
- •Exemple : *ENV PORT=8080*

7.EXPOSE

- •Indique les ports sur lesquels le conteneur écoutera
- •Exemple : *EXPOSE 80*

8.CMD

- •Définit la commande par défaut lors du lancement du conteneur
- •Une seule instruction CMD est autorisée
- •Exemple : *CMD ["python3", "app.py"]*

9.ENTRYPOINT

- •Configure le conteneur pour s'exécuter comme un exécutable
- •Peut être combiné avec CMD

SECTION 2 : VERSIONNING ET CONTENUERISATION

➢ II-3-Dockerfile et les Commandes Docker

- Les Commandes de base de Docker

Téléchargement et Recherche

- docker pull <image> : Télécharger une image
- docker search <terme> : Rechercher une image sur Docker Hub

Gestion des Images

- docker images : Lister les images locales
- docker rmi <image> : Supprimer une image
- docker build -t <nom>:<tag> . : Construire une image à partir d'un Dockerfile
- docker tag <image> <nouveau_nom> : Renommer/Étiqueter une image

Commandes de Gestion des Conteneurs

Création et Exécution

- docker run <options> <image> : Créer et démarrer un conteneur
  - -d : Mode détaché
  - -it : Mode interactif
  - --name : Nommer le conteneur
  - -p : Mapper les ports
  - -v : Monter des volumes

Listing et Inspection

- docker ps : Lister les conteneurs en cours d'exécution
- docker ps -a : Lister tous les conteneurs
- docker inspect <conteneur> : Détails d'un conteneur

Contrôle des Conteneurs

- docker start <conteneur> : Démarrer un conteneur arrêté
- docker stop <conteneur> : Arrêter un conteneur
- docker restart <conteneur> : Redémarrer un conteneur
- docker rm <conteneur> : Supprimer un conteneur
- docker pause <conteneur> : Mettre en pause un conteneur
- docker unpause <conteneur> : Reprendre un conteneur mis en pause

Interaction

- docker exec -it <conteneur> <commande> : Exécuter une commande dans un conteneur en cours
- docker logs <conteneur> : Afficher les logs
- docker attach <conteneur> : Se connecter au processus principal

Par Thomas BOGNI-DANCHI

II-3-Dockerfile et les Commandes Docker

- Les Commandes de base de Docker

Commandes Réseau

- docker network ls : Lister les réseaux
- docker network create <nom> : Créer un réseau
- docker network connect <réseau> <conteneur> : Connecter un conteneur à un réseau

Commandes de Volume

- docker volume create <nom> : Créer un volume
- docker volume ls : Lister les volumes
- docker volume rm <nom> : Supprimer un volume

Commandes de Nettoyage

- docker system prune : Supprimer les ressources inutilisées
- docker image prune : Supprimer les images non utilisées
- docker container prune : Supprimer les conteneurs arrêtés

II-4- Les images Docker

Les **images Docker** sont les éléments de base qui permettent de créer et de lancer des conteneurs Docker. Une image peut être considérée comme un **modèle prêt à l'emploi** qui contient tout le nécessaire pour exécuter une application, notamment :

- Le système d'exploitation de base (souvent une version minimale comme Alpine ou Ubuntu).
- Les dépendances logicielles.
- Les fichiers et configurations de l'application.

Cycle de vie des images

- **Construction :** Les images sont créées à l'aide d'un fichier de configuration appelé **Dockerfile** , qui contient des instructions pour assembler l'image.
- **Stockage :** Les images sont sauvegardées dans des registres (publics ou privés), comme **Docker Hub** ou des registres internes.
- **Exécution :** Les conteneurs sont créés à partir d'une image grâce à la commande *docker run*.

II-4- Les images Docker

Caractéristiques des images Docker

1. Format en couches :

Les images sont construites sous forme de **couches superposées** , en utilisant un système de fichiers Union. Chaque instruction dans le Dockerfile ajoute une couche. Par exemple :

- Ajouter un fichier.
- Installer une dépendance.
- Configurer un environnement.

Ces couches permettent de :

- **Réutiliser les éléments existants** pour réduire la taille et accélérer la construction.
- **Partager des canapés communes** entre plusieurs images.

2. Portabilité :

Une fois créée, une image est indépendante de la plateforme sous-jacente, ce qui signifie qu'elle peut être exécutée sur n'importe quel hôte Docker (Linux, Windows, macOS).

3. Source de conteneurs :

Les conteneurs Docker sont des instances en cours d'exécution d'images. Une seule image peut servir à créer plusieurs conteneurs.

Création d'images avec un Dockerfile

```
# Utilisation de l'image de base Ubuntu
FROM ubuntu:20.04
# Installation d'Apache
RUN apt-get update && apt-get install -y apache2
# Ajout d'un fichier HTML
 COPY index.html /var/www/html/index.html
# Exposition du port 80
EXPOSE 80
# Commande à exécuter au démarrage
CMD ["apache2ctl", "-D", "FOREGROUND"]
```

II-4- Les registres Docker

- **Définition**

Un **registre Docker** est un dépôt (repository) où les **images Docker** sont stockées, partagées et gérées. Les registres servent de point central pour :

- Sauvegarder vos images Docker après leur création.
- Télécharger des images existantes pour créer des conteneurs.
- Partager des images avec d'autres utilisateurs ou équipes.

Les registres permettent d'organiser et de distribuer les images en garantissant leur versionnement et leur disponibilité.

- **Types de registres Docker**

1.Registres publics :

•Le **Docker Hub** est le registre public officiel tenu par Docker. Il contient des milliers d'images publiques, comme Nginx, MySQL, Redis, etc.

•Ces registres sont accessibles à tous et permettent le partage de projets open source.

•Exemple d'utilisation :

frapper

docker pull nginx # Télécharger l'image Nginx depuis Docker Hub docker push mon_image # Pousser une image vers un registre public

➢ II-4- Les registres Docker

- **Types de registres Docker**

2.Registres privés :

•Les registres privés sont hébergés en interne ou sur une infrastructure cloud pour une utilisation limitée à une organisation ou à un projet.
•Ils offrent plus de contrôle et de sécurité pour les images sensibles.
•Exemple :
•Registre de conteneurs élastiques AWS (ECR)
•Registre de conteneurs Google (GCR)
•Registre de conteneurs Azure

Docker Hub : Le registre public officiel

•Docker Hub propose :
•**Images officielles :** Des images vérifiées et exécutées par Docker ou des organisations partenaires. (Ex. : nginx, mysql, redis)
•**Dépôts communautaires :** Des images partagées par des utilisateurs.
•**Gestion des dépôts privés :** Vous pouvez stocker vos images dans des dépôts privés, limités au nombre de dépôts gratuits pour un compte Docker gratuit.

II-4- Les registres Docker

- Commandes des registres :

- **Se connecter à un registre :** docker login REGISTRY_URL
- **Télécharger une image :** docker pull REGISTRY_URL/IMAGE_NAME
- **Pousser une image :** docker push REGISTRY_URL/IMAGE_NAME
- **Lister les tags disponibles pour une image :** curl REGISTRY_URL/v2/IMAGE_NAME/tags/list

Création d'un registre privé

Si vous souhaitez héberger un registre privé, Docker propose un conteneur pré-construit pour cela.

1. **Lancer un registre privé avec Docker :**

   docker run -d -p 5000:5000 --name registry registry:2

   Cela lance un registre local accessible via localhost:5000.

2. **Pousser une image vers le registre privé :**

   docker tag mon_image localhost:5000/mon_image

   docker push localhost:5000/mon_image

3. **Télécharger une image depuis le registre privé :**

   docker pull localhost:5000/mon_image

II-5- Docker compose

Docker Compose est un outil permettant de définir et de gérer plusieurs conteneurs Docker comme une seule application. Grâce à un fichier de configuration YAML, vous pouvez décrire les services (conteneurs), leurs réseaux, leurs volumes et leurs dépendances, puis les gérer avec des commandes simples.

Pourquoi utiliser Docker Compose ?

- **Faciliter la gestion multi-conteneurs :** Parfait pour les applications complexes avec plusieurs services (exemple : une API, une base de données, un frontend).
- **Automatisation :** Permet de démarrer, arrêter et configurer tous les conteneurs d'une application avec une seule commande.
- **Portabilité :** Avec un fichier docker-compose.yml, toute l'application est portable et facile à déployer.

- Structure de Docker Compose

SECTION 2 : VERSIONNING ET CONTENUERISATION

II-5- Docker compose

Fichier docker-compose.yml

Ce fichier YAML définit les conteneurs, les volumes, les réseaux et leurs configurations. Voici un exemple

```
version: "3.9"  # Version de la syntaxe Compose
services:
  web:
    image: nginx:latest  # Utilise l'image officielle Nginx
    ports:
      - "8080:80"        # Mappe le port 8080 de l'hôte au port 80 du conteneur
    volumes:
      - ./html:/usr/share/nginx/html  # Monte un répertoire local dans le conteneur
  database:
    image: mysql:8       # Utilise l'image officielle MySQL
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: mydb
    volumes:
      - db_data:/var/lib/mysql
volumes:
  db_data:               # Déclare un volume pour stocker les données MySQL
```

SECTION 2 : VERSIONNING ET CONTENUERISATION

II-5- Docker compose

- Les Commandes Docker Compose

Voici les principales commandes utilisées avec Docker Compose :

1. **Créer et démarrer les conteneurs définis dans docker-compose.yml:** *docker-compose up*
   - Avec l'option -d pour exécuter en arrière-plan : *docker-compose up -d*
2. **Arrêter et supprimer les conteneurs, réseaux, et volumes créés :** *docker-compose down*
3. **Afficher les logs des conteneurs :** *docker-compose logs*
4. **Redémarrer un ou plusieurs services :** *docker-compose restart SERVICE_NAME*
5. **Afficher les conteneurs en cours d'exécution :** *docker-compose ps*
6. **Construire ou reconstruire les images :** *docker-compose build*
7. **Exécuter une commande dans un conteneur :** docker-compose exec SERVICE_NAME COMMAND
   Exemple *: docker-compose exec web bash*
8. **Afficher les configurations actuelles :** *docker-compose config*

- Avantages de Docker Compose

- **Simplicité :** Réduction des commandes manuelles grâce au fichier de configuration YAML.
- **Portabilité :** Une application complète peut être reproduite à partir du fichier docker-compose.yml.
- **Gestion des dépendances :** Compose démarre automatiquement les services dans le bon ordre (par exemple, une base de données avant le backend).
- **Support des environnements multi-conteneurs :** Idéal pour des environnements de développement et de tests.

SECTION 2 : VERSIONNING ET CONTENUERISATION

➢ II-6- TP

- TP 2.1 GIT

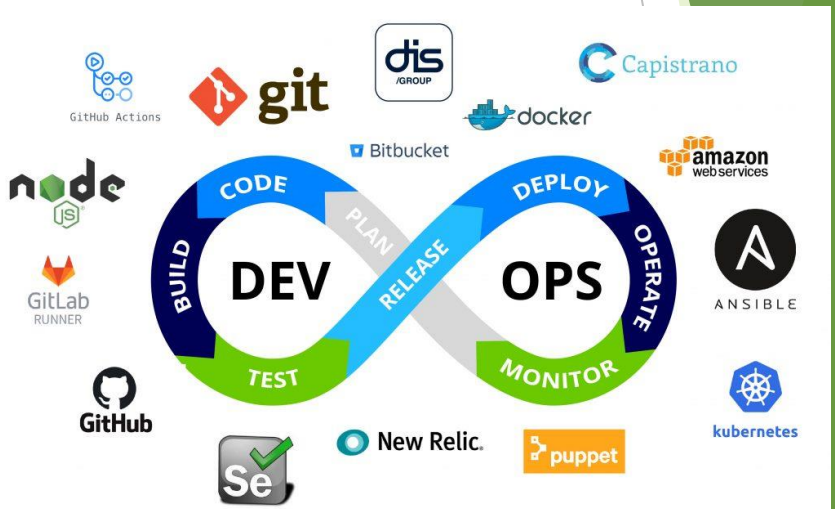
Apprendre les bases de l'utilisation de Git en local et avec un dépôt distant. Ce TP comprend la création de tags et d'alias pour optimiser le travail avec Git.

- TP 2.2 Docker

Installation de Docker, Docker compose

- TP 2.3 Image, Conteneur et Registre Docker

Création du Dockerfile
Création de l'image d'une application
Création d'un conteneur pour exécuté l'application
Sauvegarde de l'image sur Docker hub



- TP 2.4 Image, Docker Compose

Déploiement de l 'image précédente avec docker compose

- TP 2.5 Docker Compose

Déploiement de WordPress avec docker compose

Fin

Par Thomas BOGNI-DANCHI