DevOps

Dev -> Développement  
Ops -> Opération, déploiement

Partie dev, test, ops (déploiement), surveillance, monitoring (logs…)

Après déploiement, si l’appli à des défaillances, on retourne dans le dev

Toutes les étapes de développement sans que l’humain n’intervienne trop pour éviter toute erreur dans le processus.

Avant le DevOps -> un modèle traditionnel qui freinait l’innovation et créait des tensions constantes entre équipes (pas de liens dev/testeurs, pas de retours, Waterfall)  
-> cycles très lents  
-> ça peut marcher chez qq et pas chez l’autre, comment faire ?  
-> les devs ne font qu’ajouter des fonctionnalités  
-> les opérateurs veulent que le produit marche toujours -> réticent aux changements

Défis avant DevOps  
-> équipes en opposition (développeurs VS opérationnels)  
-> déploiements problématiques (mises en production risquées et retours arrière complexes)  
-> automatisation limitée (scripts maison non standardisés et difficiles à maintenir)

DevOps ?  
-> culture collaborative (réunir développeurs et opérations dans une même équipe)  
-> automatisation (réduire les tâches manuelles et répétitives)  
-> cycle continu (retour d’expérience)  
-> agilité (livraisons plus fréquentes et plus fiables)

Cycle de vie DevOps  
-> planification  
-> développement  
-> intégration (transformer le prototype en réel produit)  
->déploiement  
-> surveillance

Pourquoi versionner le code ?

Traçabilité/Sécurité/Collaboration

Révolution GIT -> système distribué ; travail hors ligne ; branches efficaces  
Conflit -> modifier le même code au même endroit

Problématiques avant Docker  
-> incohérences entre environnements (« ça marche sur ma machine ! » et pas chez le collègue)  
-> installation complexes (dépendances et conflits de versions)  
-> VMs lourdes (ressources excessives, démarrage lents  
-> Déploiement difficile (différences Dev/Test/Prod)

Docker -> système de gestion de conteneurs (léger, démarrage instantané, partage du noyau OS)  
-> mini machine virtuelle sans le noyau (OS léger)  
-> sans superflu  
-> facilement construire des images déployables en prod -> convertir une app en docker, le passer au collègue et ça marche  
  
VS

Machine Virtuelle (lourde, lancement lent, OS complet par VM)

Docker Compose (ensembles) -> composer plusieurs containeurs ensembles  
-> construire un environnement de travail facilement  
-> éviter d’avoir toutes les dépendances sur toutes les machines (Apache, MySQL…)  
-> on a tout sur une machine

Docker utile pour la production  
-> Kubernetes (orchestrateur d’images, les organiser) ou Nomad

Pourquoi Docker ?  
-> standardisation (assure des environnements identiques)  
-> rapidité (déploiements rapides)  
-> Isolation (gère efficacement les dépendances)  
-> Intégration (améliore l’intégration)