Pilotage des développements par les tests

Approches

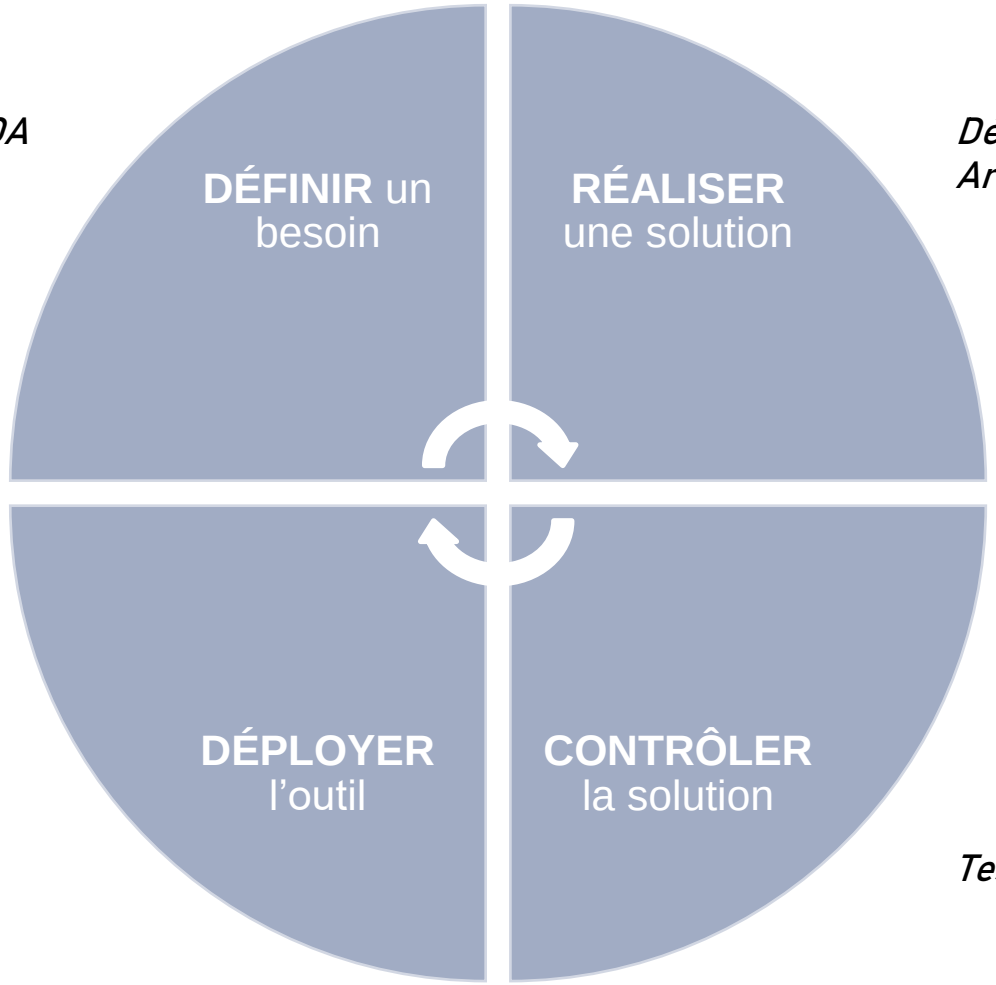
henix

Sommaire

Cycle de vie d'un logiciel

Analyste métier / AMOA / Product Owner

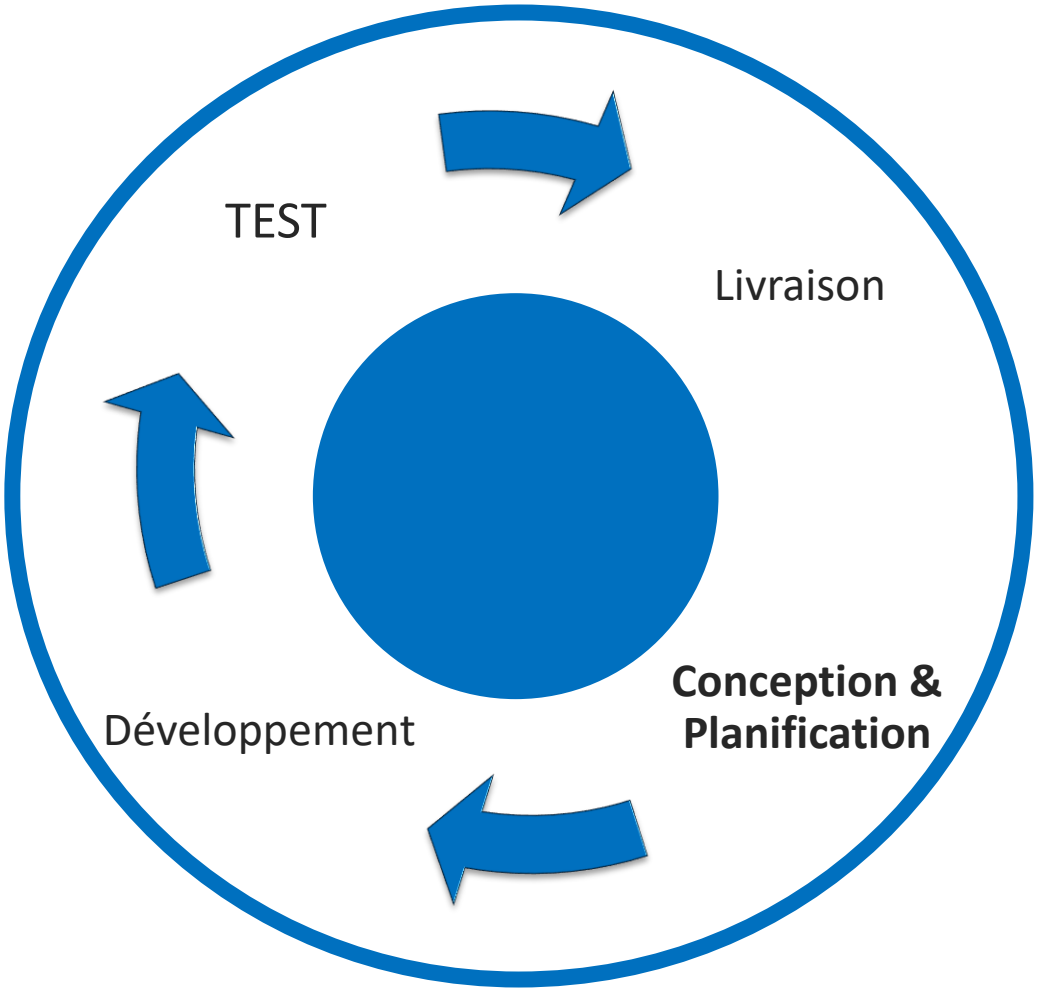
Développeur / Architecte Logiciel



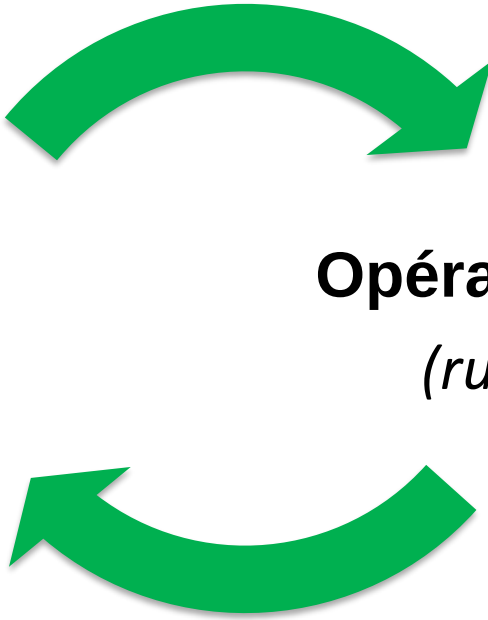
Chargé de déploiement / de production

Testeur

Développement

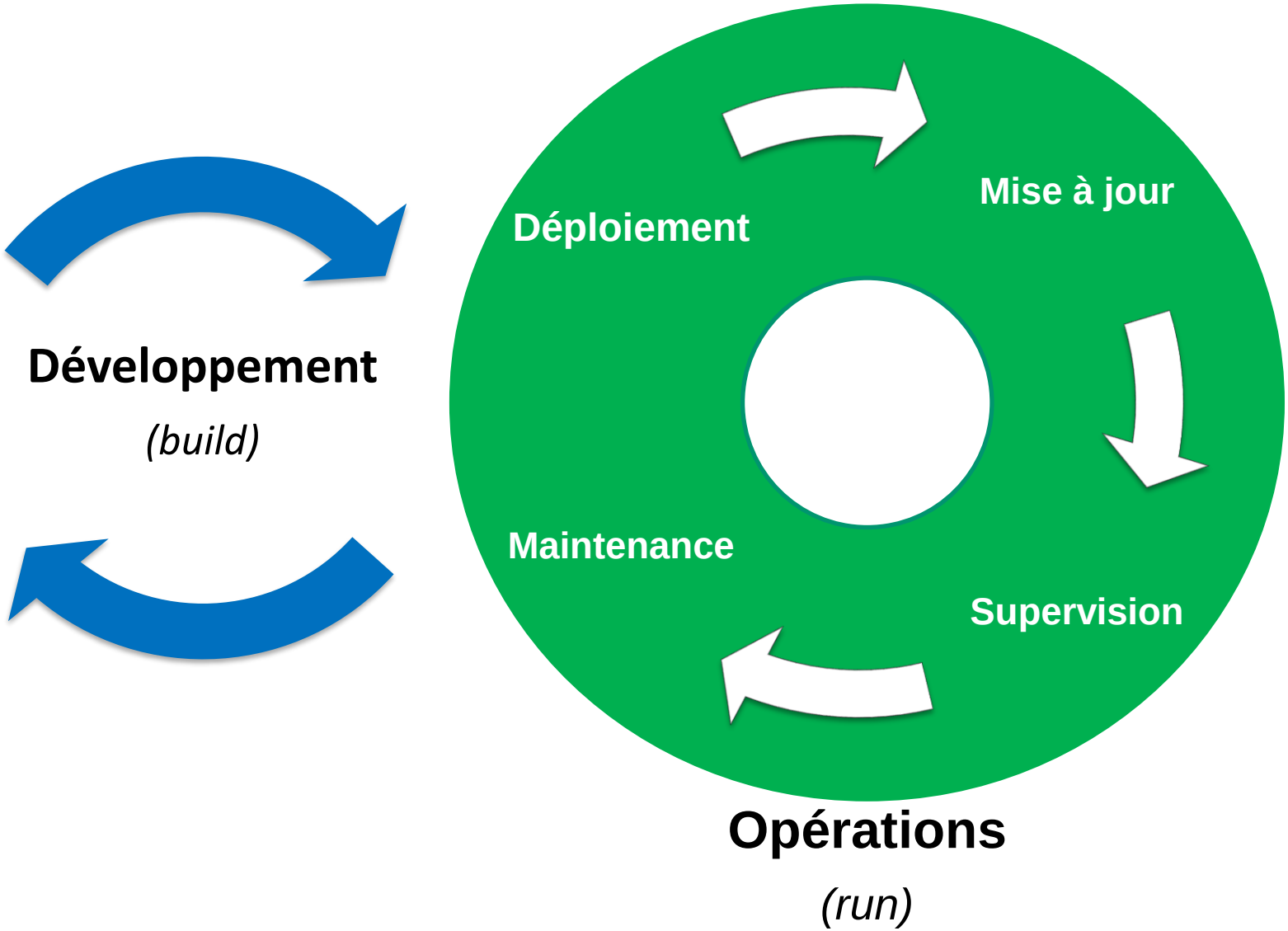


Développement
(build)



Opérations
(run)

Opérations


Développement
(build)
Déploiement
Mise à jour
Supervision
Maintenance
Opérations
(run)

Problématique

L’enjeu

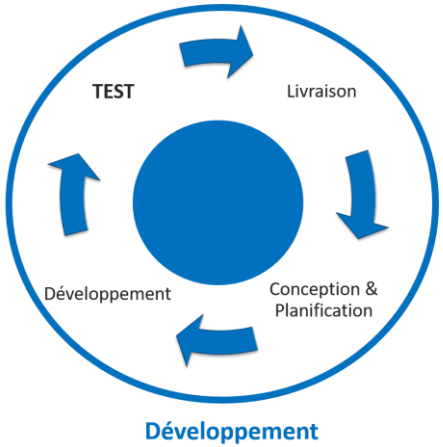
Quelle relation entretiennent ces deux familles d’activités ?

Comment se sont elles adaptées aux nouveaux modèles de gestion de projet (Agile, DevOps) ?

Sommaire

Approches classiques VS Agile

Les activités liées au développement d'une solution informatique :



- Recueil des besoins fonctionnels
- Ecriture des spécifications fonctionnelles et techniques
- Programmation et écriture du code
- Conception et exécution des tests
- Livraison de l'application

Le testeur fonctionnel et l'automaticien de tests interviennent tous les deux lors de la phase de développement

Approches classiques VS Agile

Présentation comparée des différentes approches de développement

Les méthodes et pratiques de développement se calquent sur la gestion de projet dans lesquelles elles s'inscrivent.

On distingue deux grands types d'approches :

prédictive et séquentielle **itérative et adaptative**

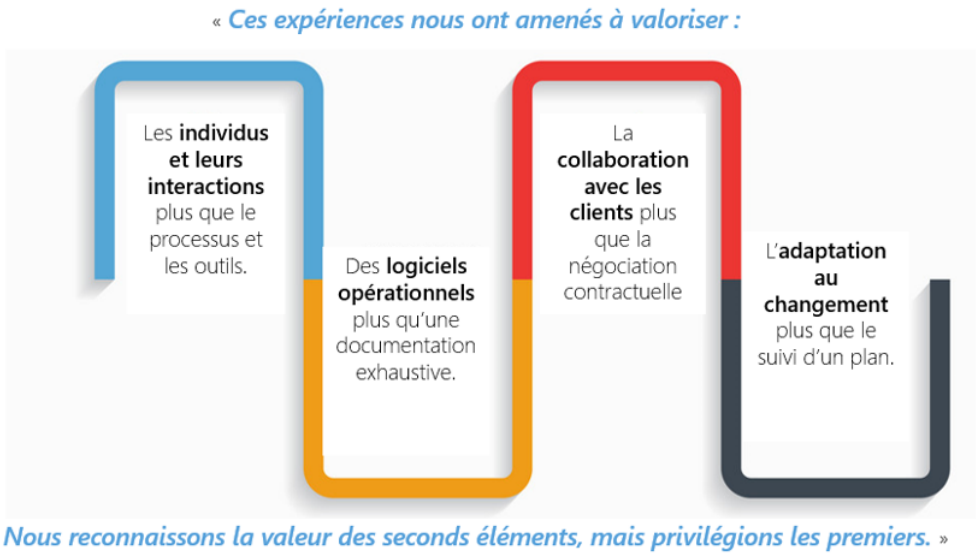
L'approche traditionnelle	L'approche agile
Elle suppose de *spécifier et planifier dans les détails* l'intégralité d'un produit avant de le développer.	Elle consiste à *définir un objectif*, puis apprendre en faisant, en procédant par étape
Les activités (spécification, conception, implémentation, vérification) **sont séquentielles**	Spécification, conception et vérification **s'enchaînent** au cours de **phases itératives de courte durée**
Les testeurs interviennent à la fin du cycle de développement	Les testeurs sont parties prenantes des développements
Méthodes de gestion de projet : **Cycle en V**, **Cascade**...	Méthodes de gestion de projet : **Scurm**, **Kanban**, **XP**...

Approches classiques VS Agile

Les principes généraux de l'Agile

Les pratiques de développement itératives et incrémentales ne sont pas récentes (les premières apparaissent dans les années 1970), mais c'est le Manifeste Agile, rédigé en 2001, qui a consacré le terme « Agile » pour qualifier ces approches.

Ce manifeste comporte uniquement une déclaration de 4 valeurs et 12 principes. On parle de méthode**s** agile**s** pour les méthodes fondées sur ces principes.

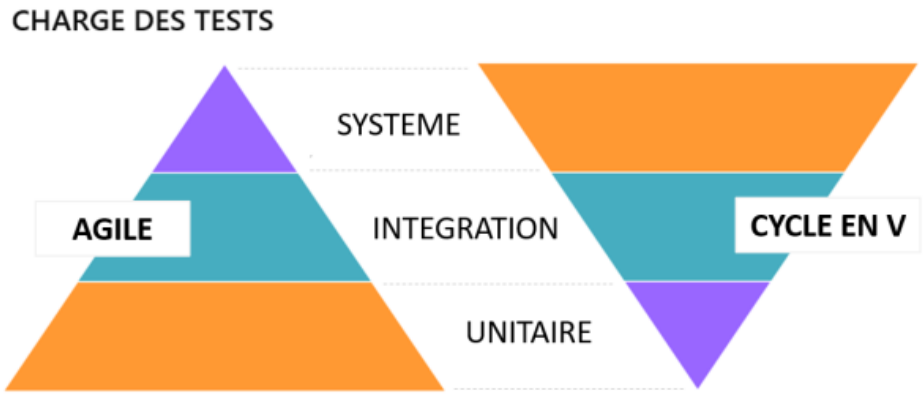


Les trois méthodes agiles les plus utilisées sont : **Kanban**, **Scrum**, et **XP Extreme programming.**

Approches classiques VS Agile

En contexte agile, l'activité de test est présente à tous les niveaux **: unitaire, intégration, système, acceptation**.

Mais, la logique adoptée diffère des cycles de développement traditionnel.



« plus une anomalie est découverte tard, plus sa correction coûte cher »

Les mots d'ordre :

- **Responsabiliser les développeurs sur la qualité des productions**
- **Trouver les anomalies au plus tôt**
- **Moins de spécifications écrites : les tests deviennent des spécifications actives**

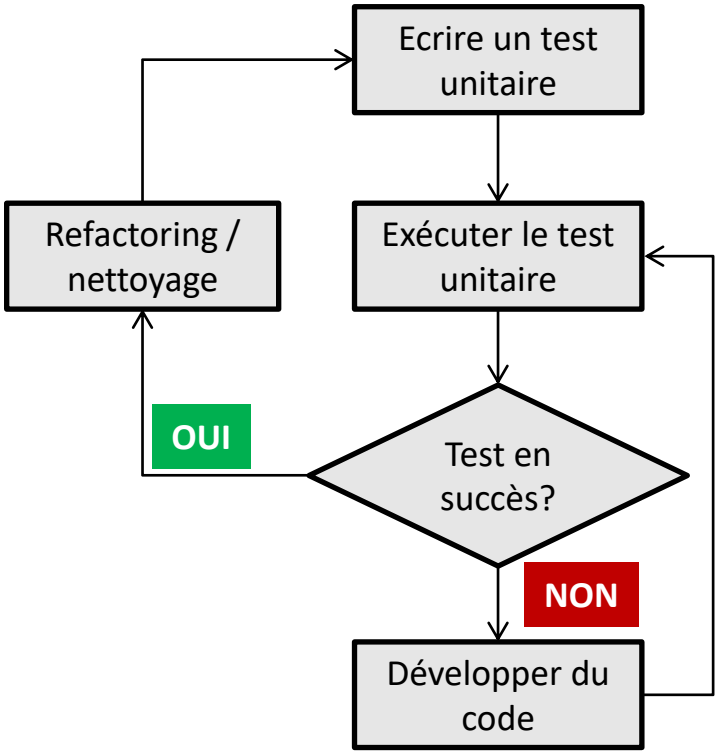
Sommaire

Des développements pilotés par les tests

Testing first !

Le TDD, qu'est-ce que c'est ?

- « *Test Driven Development* »
- Une pratique de développeur
- Concerne les tests unitaires
- les TU sont écrits avant le développement, pour définir les règles de gestion d'un composant (méthode)
- Puis on développe les fonctions avec pour objectif de faire passer les tests en succès.



Des développements pilotés par les tests

Testing first !

Avantages de la TDD

- Les tests constituent une spécification « active » / « vivante » du composant
- L'objectif de 100% de couverture des tests est forcément atteint
- Une fois les tests écrits, la validation est très rapide et ne coûte virtuellement rien – les T.U. constituent de vrais tests de non-régression !

➔ **Tests rapides et exécutables aussi souvent que nécessaire**

OUTILLAGE

Les frameworks de tests unitaires *(ex: JUnit)*

JUnit

Des développements pilotés par les tests

Testing first !

Extension de la TDD, le BDD

- Même approche que le TDD : écriture des tests avant le code

  MAIS …

  - Orienté « fonctionnalité » et non « composant »
  - Méthode adressée à un public **fonctionnel** et pas uniquement aux **développeurs**

- Faire collaborer les équipes à la réalisation des tests (Agile)
  - Les équipes fonctionnelles rédigent les tests d'un point de vue de l'utilisateur final
  - Les développeurs écrivent le code validant le test

- Cette approche est appelée **BDD** (Behaviour Driven Development)

Le besoin fonctionnel guide le développement de l'application

OUTILLAGE

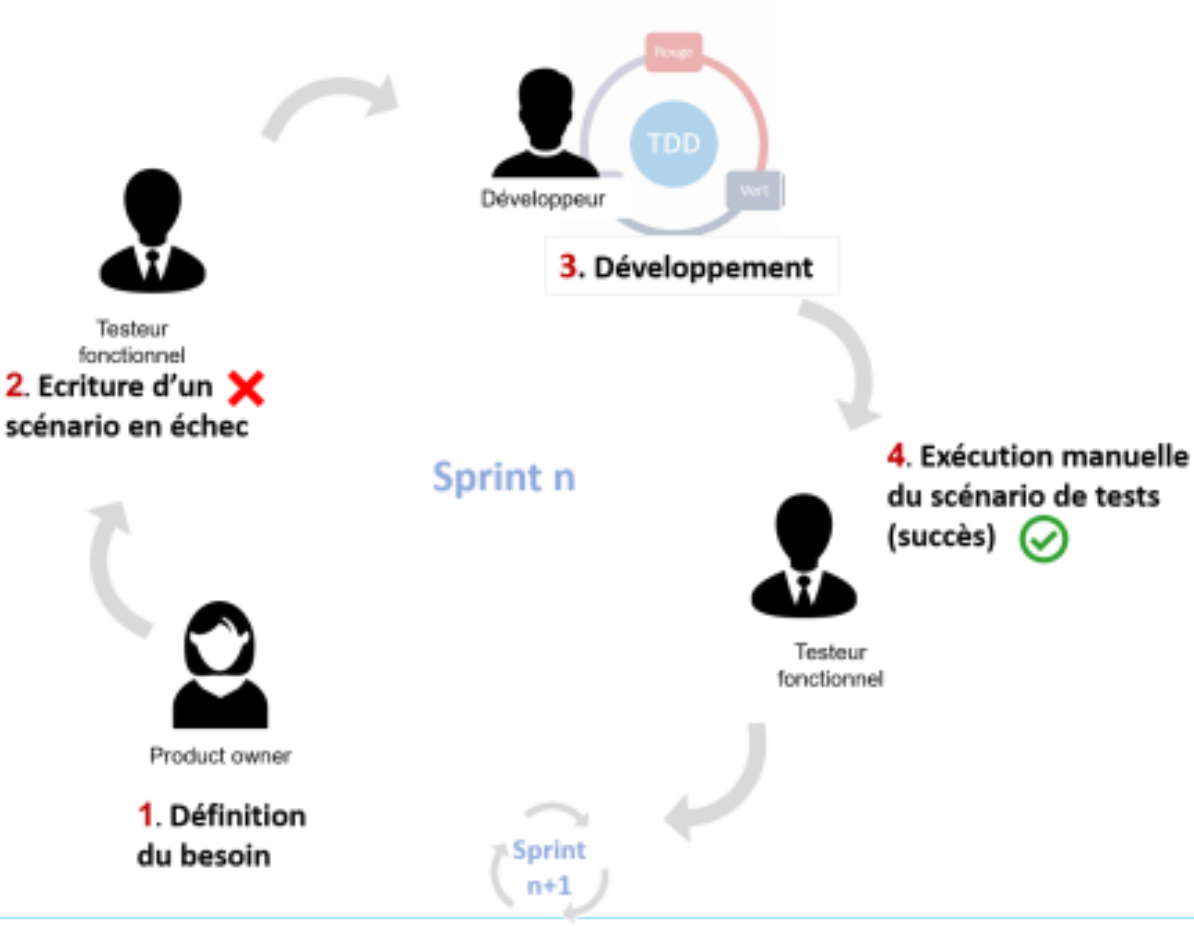
Les frameworks d'implémentation BDD *(Gherkin/Cucumber)*

cucumber

Des développements pilotés par les tests

Le BDD : le workflow

Dans la logique Agile BDD, le scénario est spécifié avant le développement. A la fin du sprint, si tout s'est passé comme convenu, les *scenarii* sont en succès



Des développements pilotés par les tests

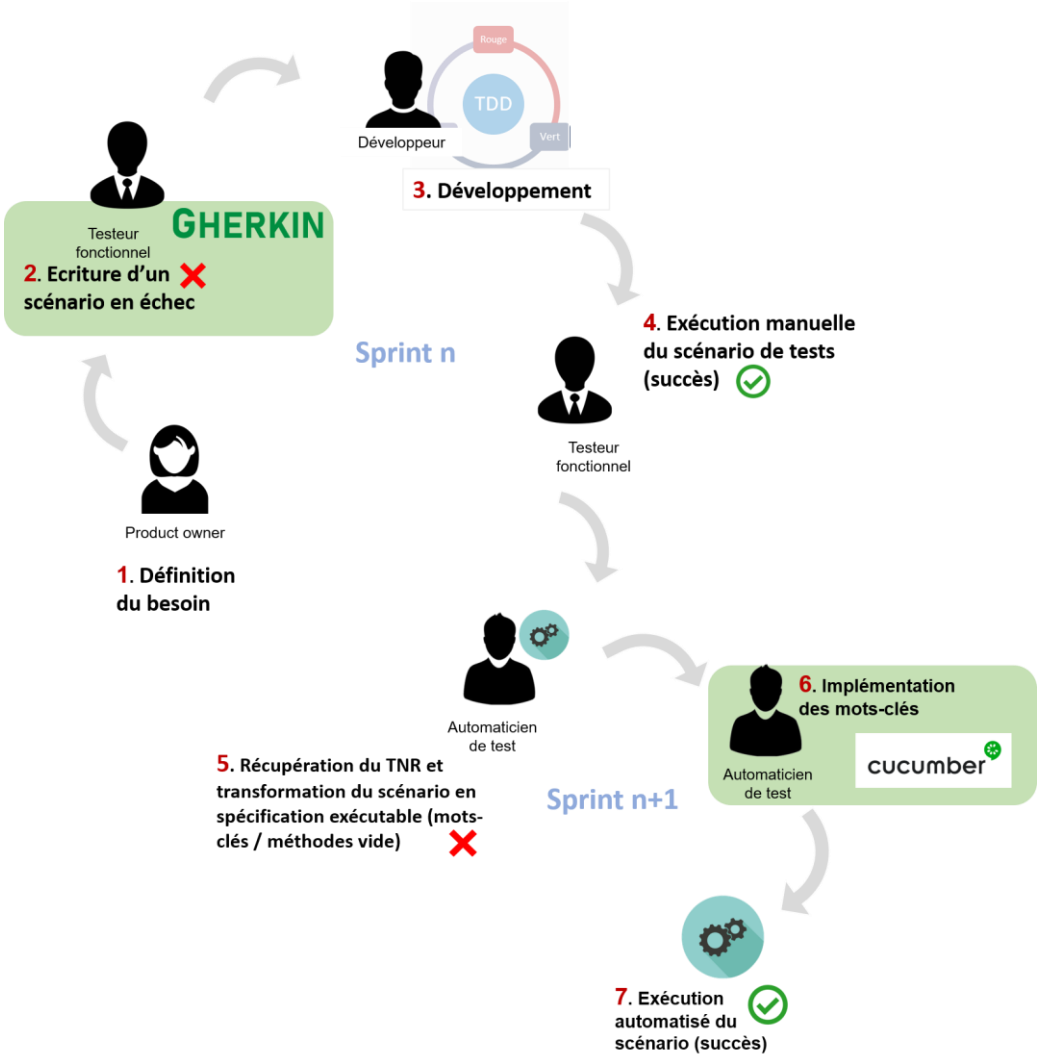
Testing first !

Le BDD : le workflow

Automatiser les TNR en BDD

Une fois le scénario en succès, l'équipe aura pour tâche d'automatiser son exécution.

Le scénario, s'il est bien spécifié, deviendra le script de test automatisé, à condition qu'un automaticien implémente les mots-clés.

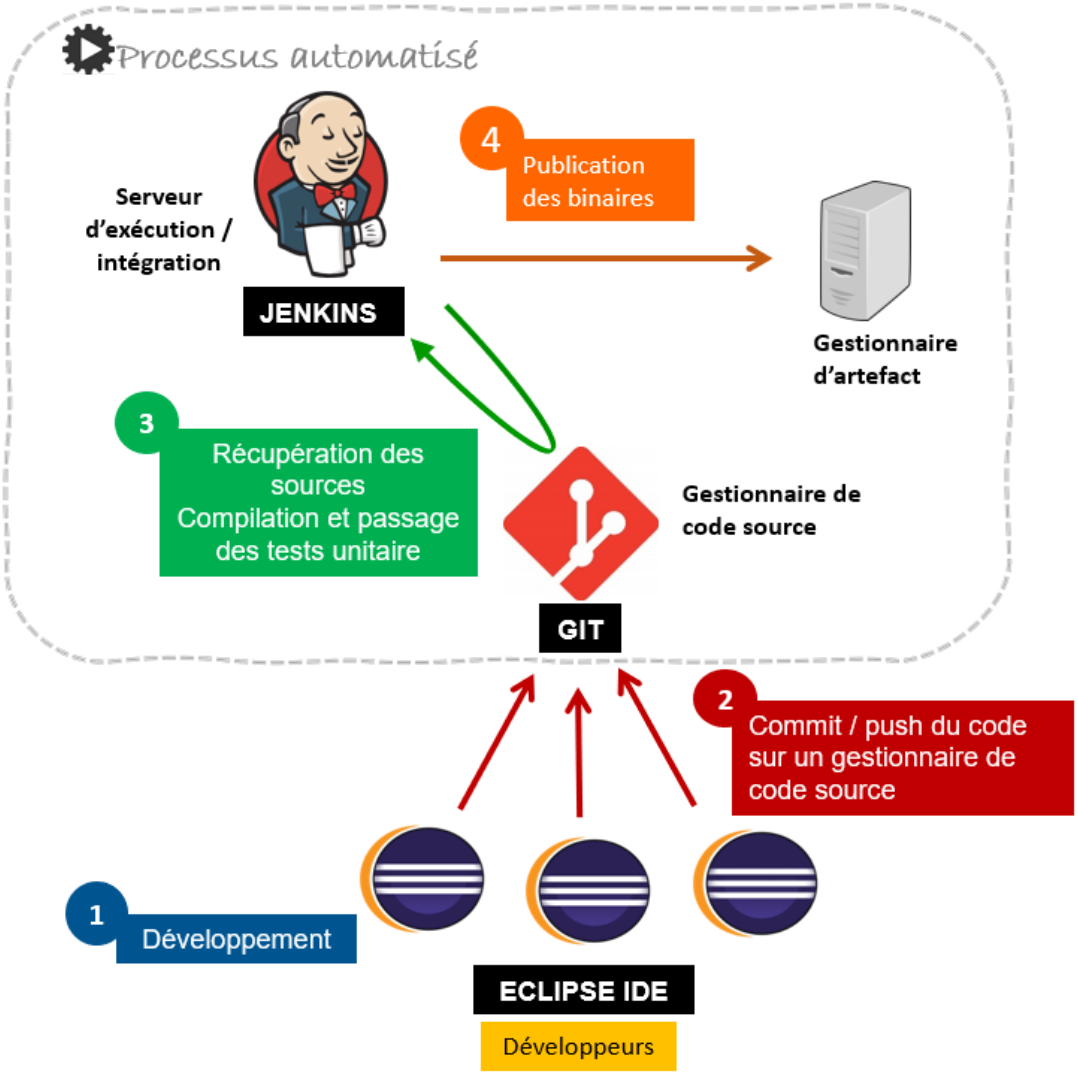


Sommaire

Adoption de l'intégration continue

Continious testing

L'intégration continue… industrialiser l'intégration du code source



Adoption de l'intégration continue

Les avantages de l'intégration continue

L'intégration continue… industrialiser l'intégration du code source

- Assure la qualité du code au plus tôt
- Rend la production d'une application plus « fluide »
- Permet de générer du reporting de qualité
- Rend possible la mise en place d'une méthode de travail DevOps

Adoption de l'intégration continue

Continious testing

Les outils de l'intégration continue

Gestionnaire de code source

- Gérer l'historique du code source
- Gérer le statut des différentes révisions de ce code source
- Simplifier les mécanismes de travail à plusieurs

Outillage : Git, SVN …

Le serveur d'intégration continue

- Déclencher des tâches
- Enchaîner des tâches (constructions, tests)
- Informer sur le statut de ces tâches

Outillage : Jenkins, CircleCI …

Le gestionnaire d'artefact

- Gérer un patrimoine d'artéfact
- Mettre des artéfacts à disposition des équipes « opérations » (ex : une release d'application)

Outillage : Nexus, Artifactory…

Sommaire

Production, Exploitation et Opérations

Englobent toutes les activités liées au déploiement et au suivi de l'application en production :

- Montage et provisionnement des environnements (production et recette)
- Déploiement des applications
- Mise-à-jour et montée de version des systèmes
- Supervision et surveillance des environnements
- Opérations de maintenance
- Sauvegarde et back-up

Ces métiers utilisent aussi plusieurs formes de test (maintenance, récupération…) ainsi que l'industrialisation de certains processus : scripts de déploiement, sauvegarde reporting…

Sommaire

Pourquoi le DevOps ?

Etymologie

- ***Dév****eloppement* ou *Etudes*
  - Recueillir le besoin en nouvelles fonctionnalités ou corrections
  - Concevoir et développer les nouvelles fonctionnalités
  - Livrer l'application terminée à la production
- *Production* ou *Exploitation ou* ***Op****érations****s***
  - Déployer l'application sur un serveur ou sur les postes utilisateur
  - S'assurer que le service est disponible
  - S'assurer que le service est de qualité
  - En première ligne en cas de problème sur l'application

Pourquoi le DevOps ?

	Etudes / Développement	Exploitation / Production
Rôle :	Fabriquer un produit	Fournir un service basé sur un produit
Objectifs :	Répondre aux besoins client en terme de fonctionnalités (évolutions / corrections) Respecter des délais client (livraisons)	Garantir la disponibilité du service Garantir la qualité du service Respecter des normes et procédures d'exploitation
Caractéristiques:		
Répétabilité d'une approche	50%	99%
Approche	Adaptive	Planifiée
Amélioration	Savoir-faire	Contrôle

Objectifs différents … Cultures différentes…

Pourquoi le DevOps ?

Développement vs. Production, quelle relation?

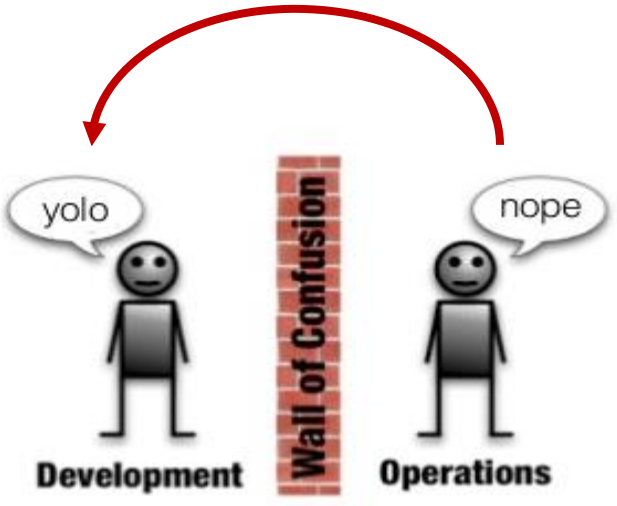
Objectifs différents … Cultures différentes…

Pourquoi le DevOps ?

Développement vs. Production, quelle relation?

Application finie
avec documentation d'exploitation

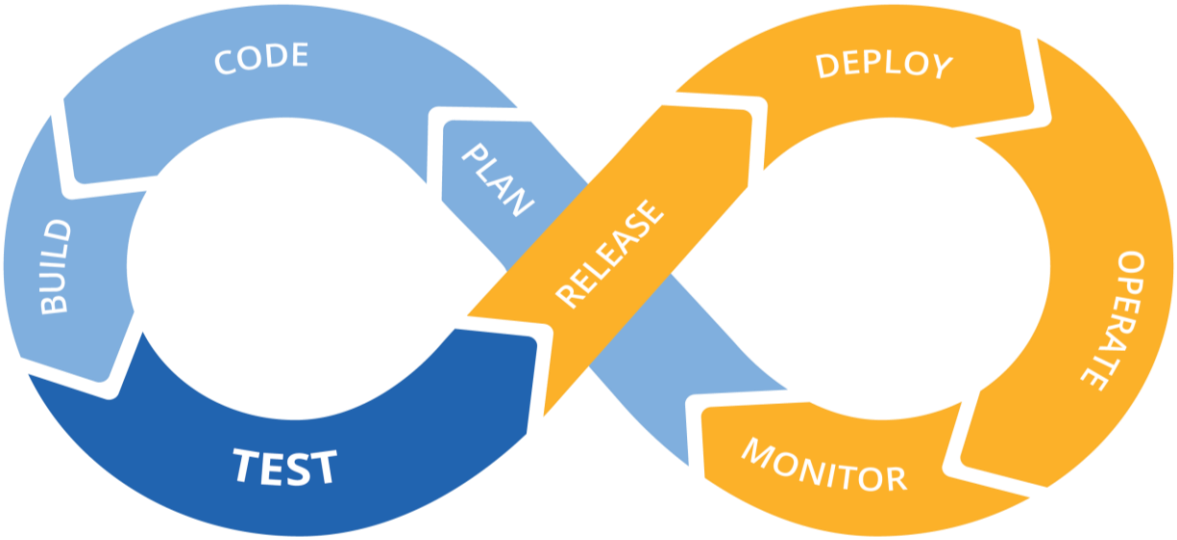
Spécifications de critères pour l'exploitabilité
Demandes de corrections suite à anomalies



- Rigidité
- Perte d'information lors du passage du « mur »
- Perte de temps liés aux allers-retours

Pourquoi le DevOps ?

Rapprocher le Développement et les Opérations (DevOps)

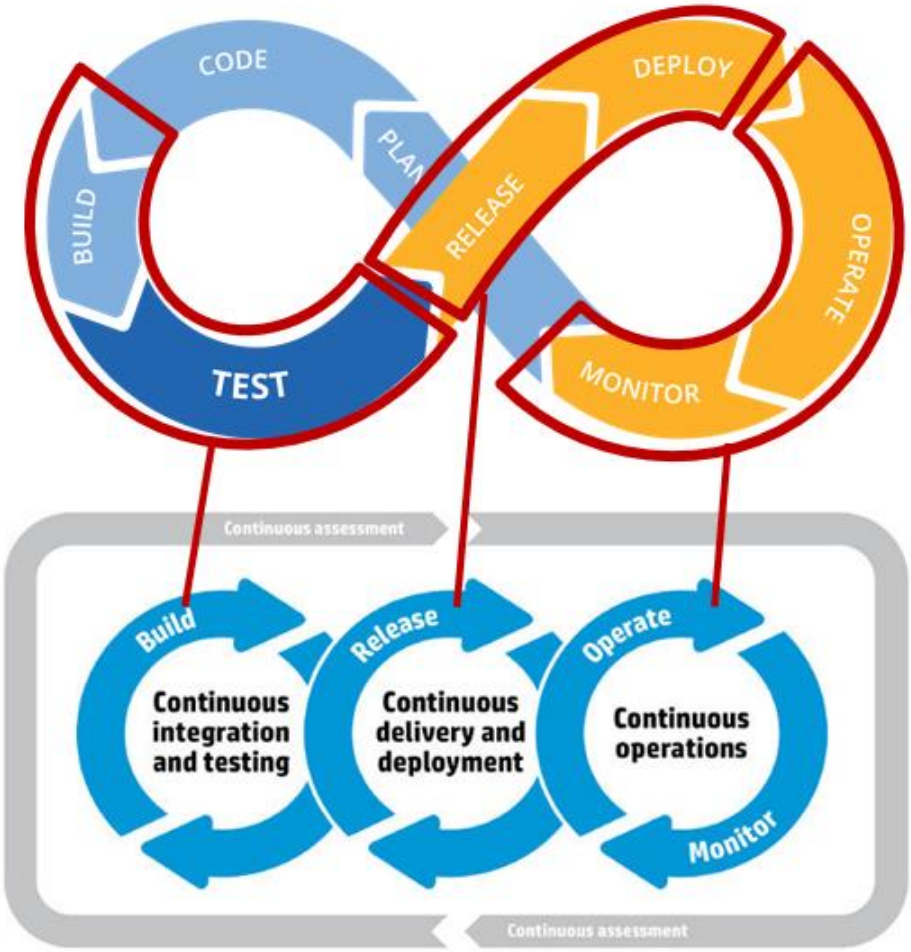


Réduire le *time-to-market* c'est-à-dire le temps entre une demande (évolution ou correction) et sa livraison en production tout en s'assurant de la qualité.

Pourquoi le DevOps ?

Les méthodes et concepts du DevOps

- *Intégration et tests continus*
- *Livraison / déploiement continu*
- *Opérations continues*

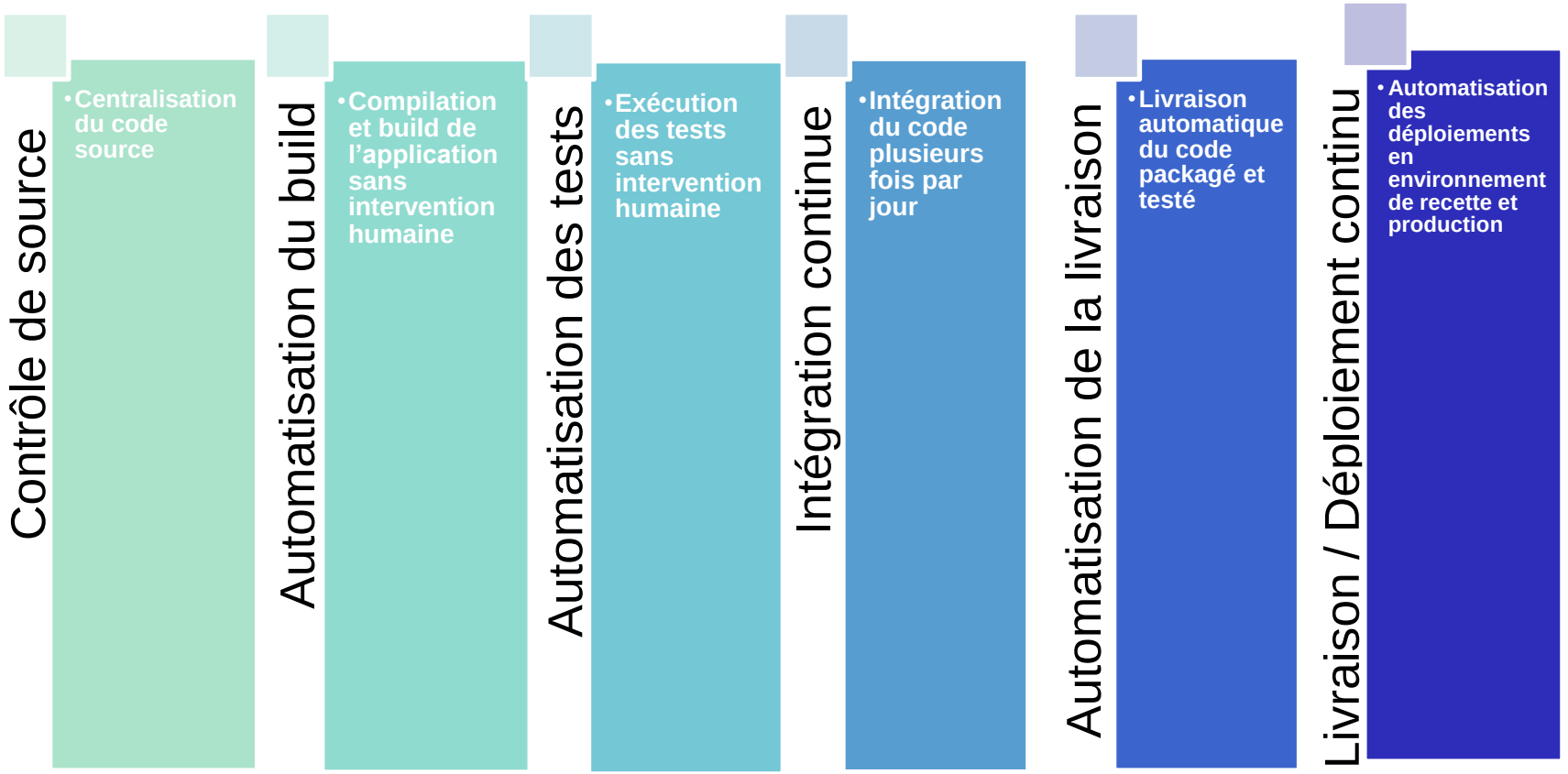


Sommaire

La mise en œuvre DevOps

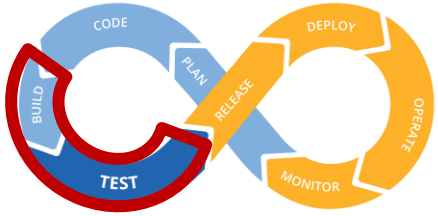
L'approche DevOps se définit par l'industrialisation des phases d'intégration, de livraison et de déploiement.

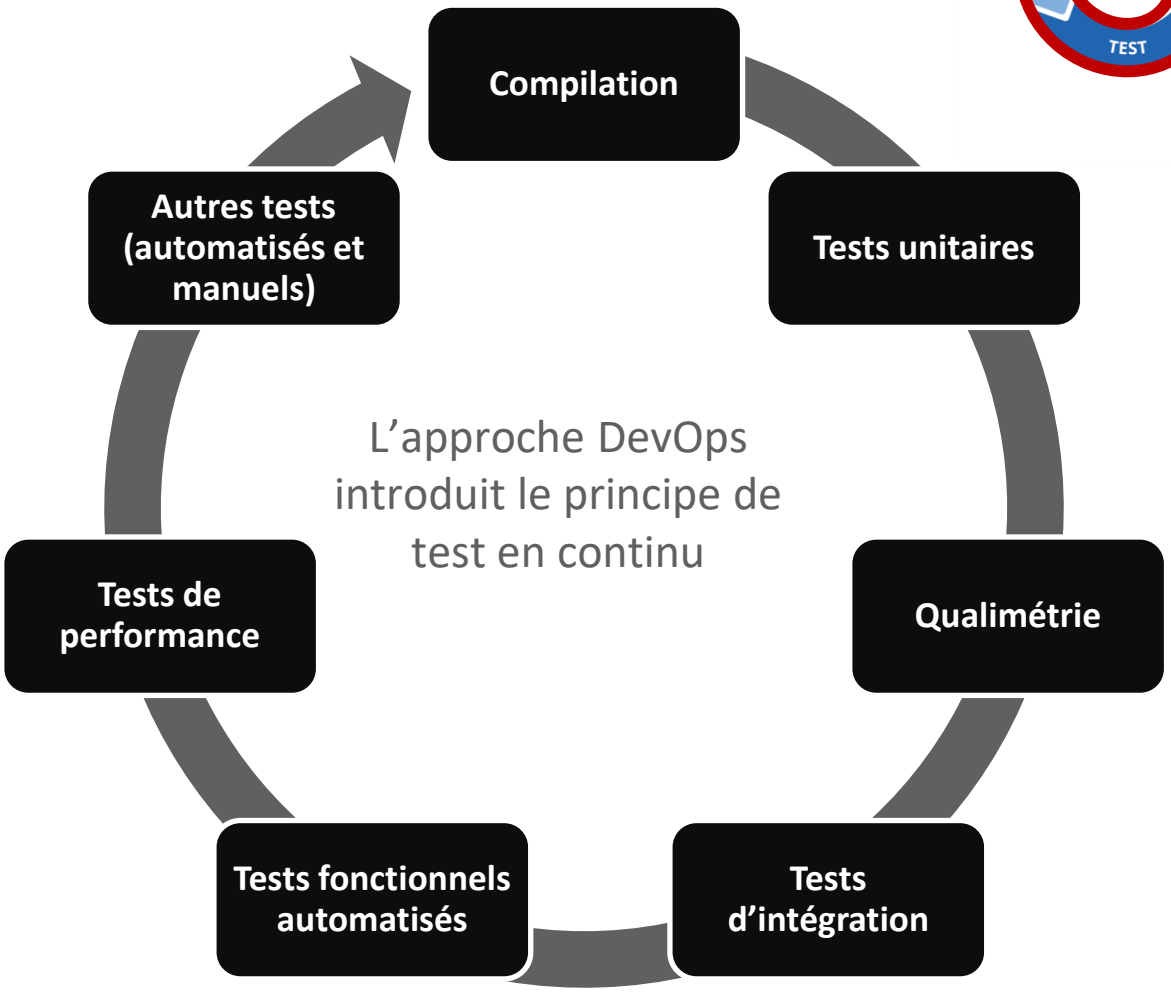
Pour beaucoup d'entreprise, le DevOps est un but vers lequel progresser… par **étape** :



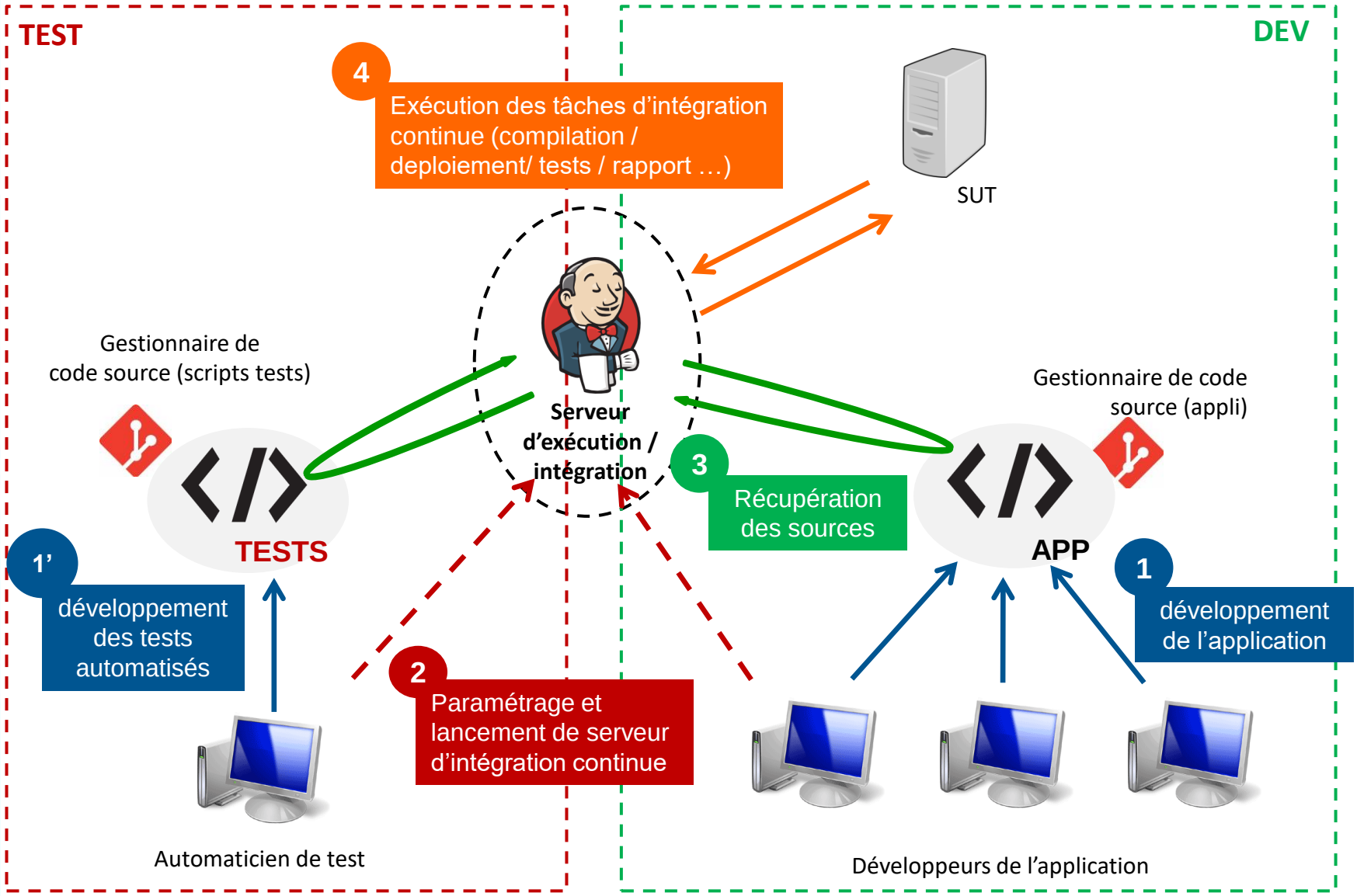
La mise en œuvre DevOps

Continuous integration and testing



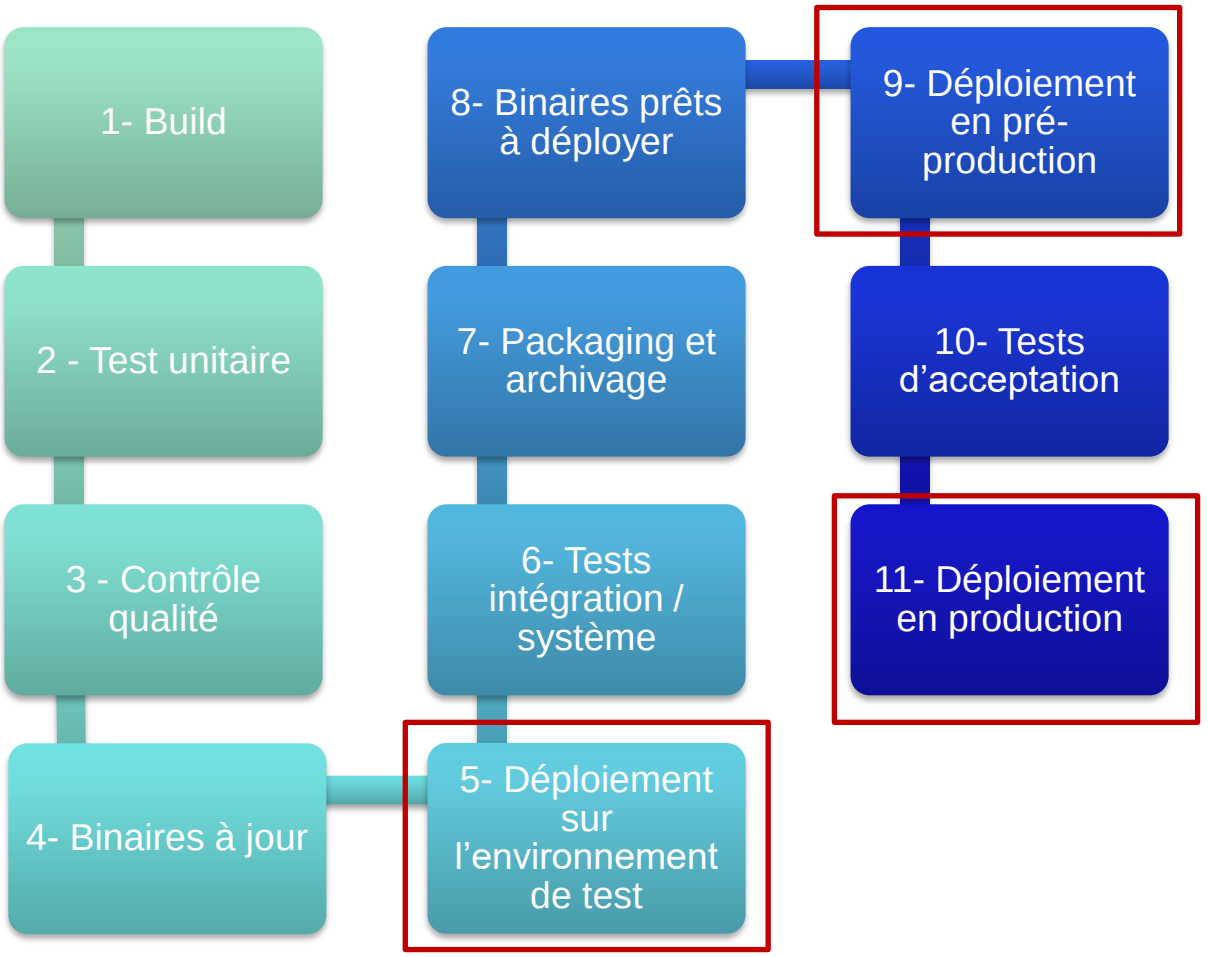


La mise en œuvre DevOps



La mise en œuvre DevOps

Du l'intégration continue au déploiement continue



La gestion des environnements de test

Sommaire

Provisionner les environnements

Les environnements de test

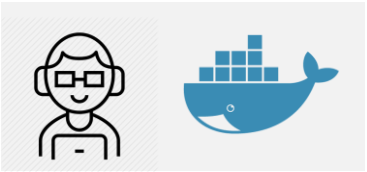
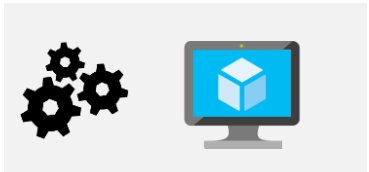
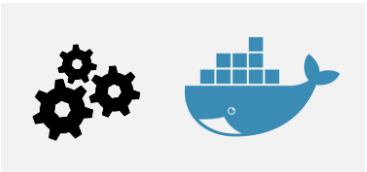
- Différents types de tests requièrent différents types d'environnements pour être exécutés.

- Certains ne requièrent pas un déploiement complet de l'application voire ne demandent pas de déploiement du tout.
  - Tests unitaires
  - Certains tests d'intégration
  - Qualimétrie/Analyse de code
  - Certains tests de sécurité qui analysent le code

- D'autres, au contraire, requièrent un déploiement complet de l'application
  - Tests d'IHM
  - Tests de sécurité (robots)
  - Tests de performance
  - Tests d'accessibilité

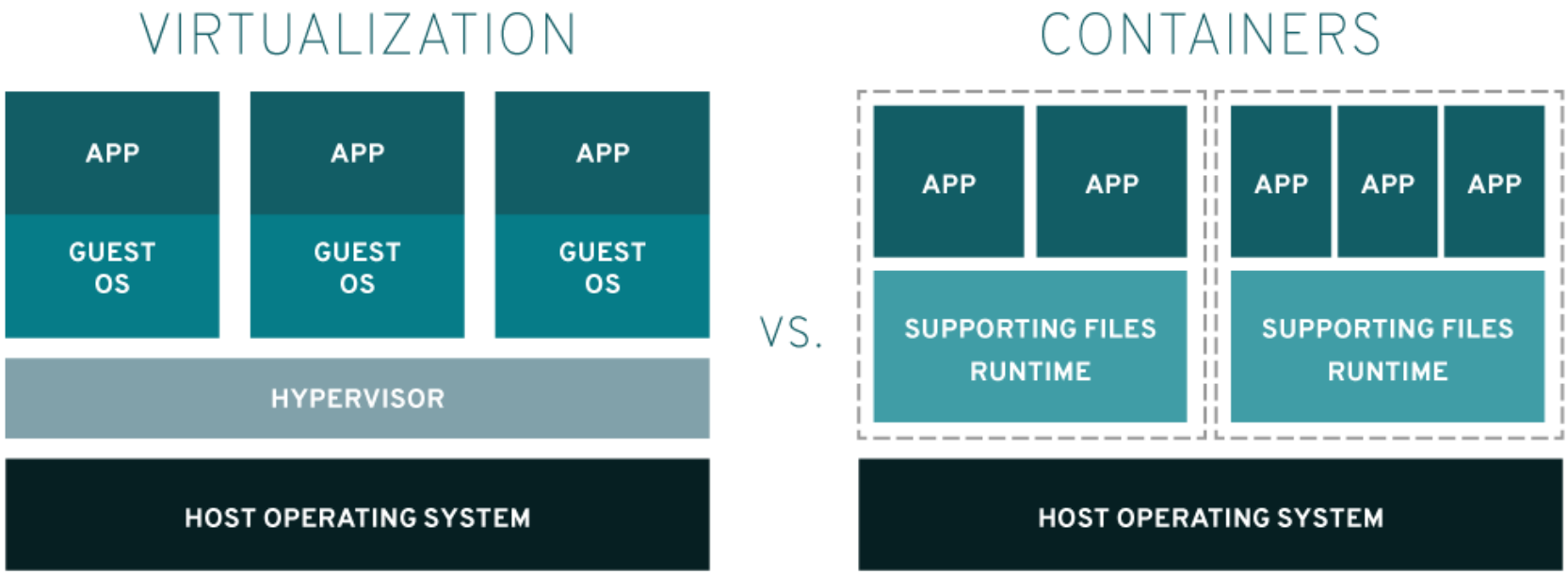
Provisionner les environnements

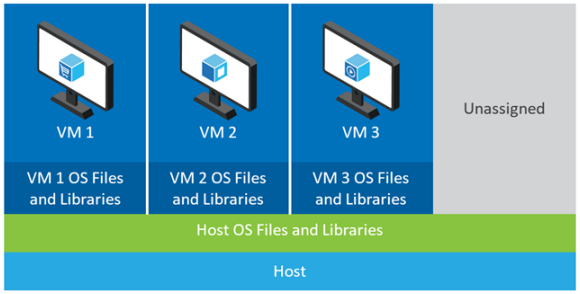
Comment provisionne-t-on des environnements ?

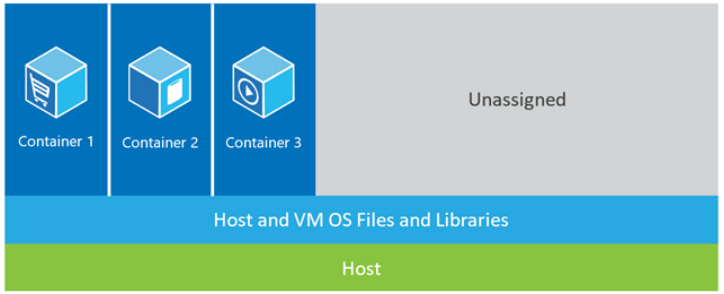
	Serveur Physique	Machine Virtuelle	Conteneurs
Manuel			
Automatisé			

Provisionner les environnements

Virtualisation classique vs. Conteneurs

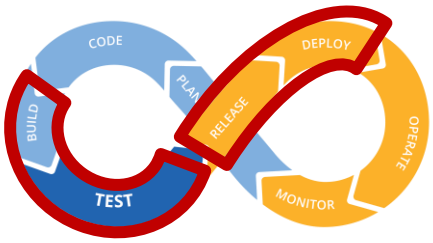




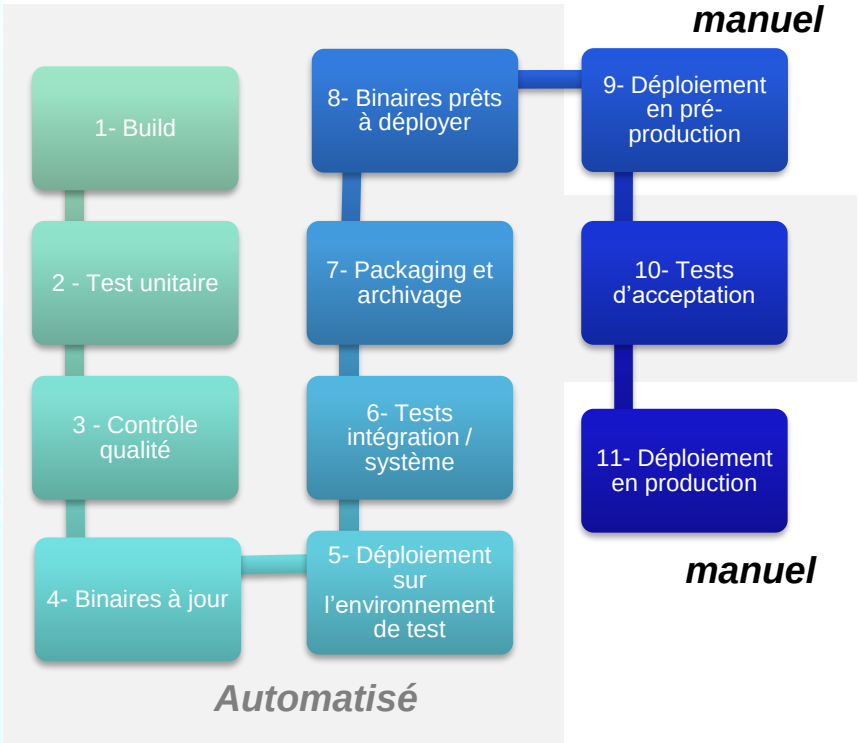


Provisionner les environnements

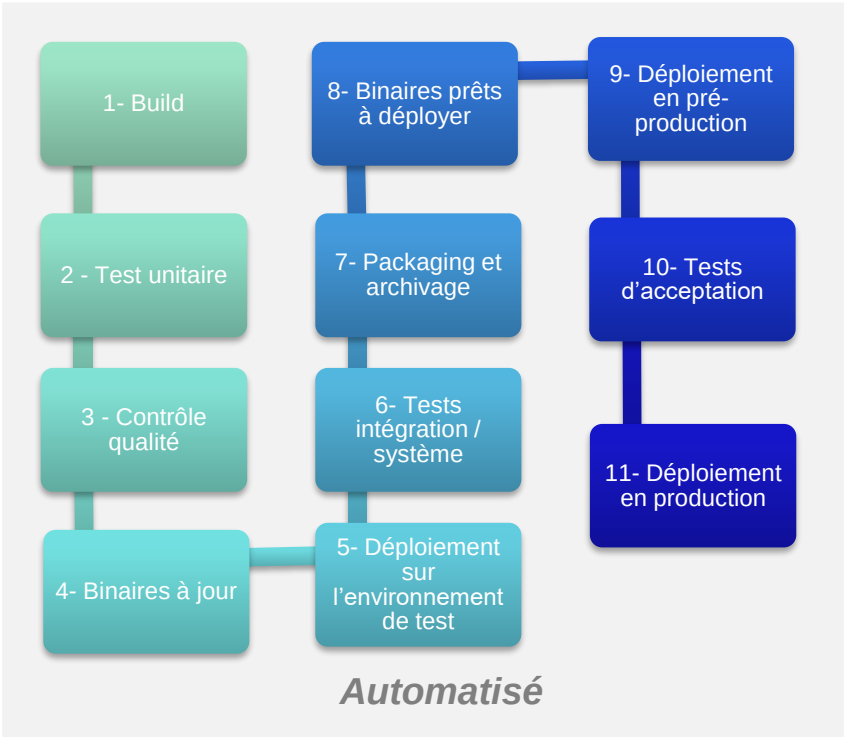
Quelle différence entre intégration continue et déploiement continu ?



Intégration/test en continu (dans sa forme complexe)



Déploiement continu



Integration, test, déploiement continu – les outils

GitHub

Maven

JUnit

sonarqube

1- Build

2 - Test unitaire

3 - Contrôle qualité

4- Binaires à jour

5- Déploiement sur l'environnement de test

6- Tests intégration et système

7- Packaging et archivage

8- Binaires prêts à déployer

Sonatype Nexus

Selenium

appium

SoapUI

docker

ANSIBLE

9- Déploiement en pré-production

10- Tests d'acceptation

11- Déploiement en production

ANSIBLE

docker

cucumber

NeoLoad

ANSIBLE

docker

Installation automatisée

	Build & Validate	Quality Scan	Package & Publish	Deploy to Qualif	Automated Tests	Tear down Qualif	Deploy to UAT	Deploy to Preprod
Average stage times: (Average full run time: ~16min 32s)	7min 7s	19s	1ms	6min 55s	27s	1min 43s	16ms	12ms
#18 Jul 27 09:49 — No Changes	7min 21s	20s	2ms	7min 46s	25s (paused for 18min 45s)	1min 54s	13ms (paused for 5min 44s)	15ms (paused for 19s)
#17 Jul 26 18:18 — No Changes	7min 4s	18s	1ms	6min 31s	34s (paused for 1min 20s)	1min 50s	15ms (paused for 40s)	12ms (paused for 42s)
#16 Jul 26 17:53 — No Changes	7min 18s	20s	2ms	6min 45s	24s (paused for 7min 32s)	1min 52s	11ms (paused for 30s)	18ms (paused for 9s)
#15 Jul 26 17:32 — No Changes	7min 3s	18s	2ms	6min 31s	24s (paused for 1min 6s)	1min 1s	23ms (paused for 2min 18s)	9ms (paused for 2s)
#14 Jul 26 17:01 — No Changes	7min 5s	21s	1ms	7min 1s	23s (paused for 32s)	1min 53s	21ms (paused for 2min 41s)	7ms (paused for 6s)
#13 Jul 26 16:33 — No Changes	6min 56s	20s	2ms	7min 17s	24s failed			
#12 Jul 26 15:51 — No Changes	7min 0s	20s	1ms	6min 32s	34s (paused for 5min 55s)	1min 50s	16ms (paused for 6s)	12ms (paused for 2s)