INTRODUCTION DEVOPS

2023-2024

ESPRIT thouraya.louati@esprit.tn UP ASI (Bureau E204)

- Thouraya Louati
- Docteur en informatique
- Courriel: thouraya.louati@esprit.tn
- Ressources DevOps:

  Classroom

  https://gitlab.com/ThourayaLouati/

Module et Objectif

Module : DevOps.

Objectif : L'étudiant sera capable de mettre en œuvre les maillons d'une chaîne DevOps avec Jenkins, Docker, Nexus, Sonar, Git et Grafana.

Plan :

Séance	Contenu
S1	Introduction à DevOps + Environnement
S2	Le serveur d'intégration continue Jenkins
S3	Introduction Docker
S4	Le gestionnaire du code source Git
S5	La gestion du dépôt de livrables Nexus et JUNIT
S6	La vérification de la qualité du code Sonar
S7	La livraison continue (Docker avancé: compose + volume)
S8	Grafana & Prometheus
S9&S10	Validation du projet

Répartition et modalité d'évaluation

- Répartition de la charge horaire:
  - 30h Cours Intégré (3h par semaine (2h en synchrone + 1h en asynchrone)).
    - 9h cours
    - 21h TPs
    - 10 séances
- Modalité d'évaluation
  - Examen 100% (Pratique: Projet, groupes de 5 ou 6 étudiants)
- Courriel de groupe / Mailing List ?

Acquis d'apprentissage

- **Acquis d'apprentissage :**
  - Créer un pipeline Jenkins.
  - Conteneurisation (Sonar, Nexus, Grafana, Prometheus)
  - Interpréter les rapports des normes de développement Sonar.
  - Construire un dépôt pour la livraison des artéfacts (Nexus)
  - Évaluer le fonctionnement et les résultats de l'exécution du pipeline.
  - Créer un projet en collaboration avec les membres de l'équipe (Spring, Angular, Maven).
  - Appliquer la journalisation des actions des utilisateurs d'une application : (LOG4J)
  - Tester les fonctionnalités implémentées (JUnit)
  - Livraison en continue (Docker)
  - Monitoring en continue (Grafana & Prometheus)

Prérequis

- Java

https://www.coursera.org/learn/init-prog-java

https://www.coursera.org/learn/programmation-orientee-objet-java

- Linux
  - **https://www.server-world.info/en/**

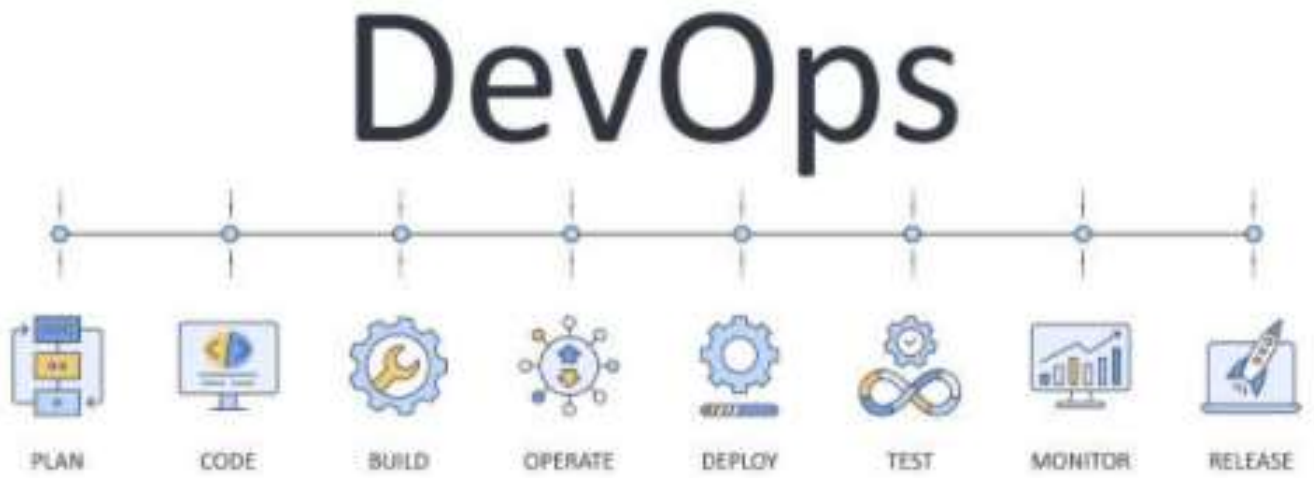
Séance 1: Introduction DevOps

Séance 1

- **Séquence 1: Introduction DevOps**
  - Charge : 3 heures
    - Savoir les origines et les motivations derrière l'apparition du concept de DevOps.
    - Savoir les enjeux de DevOps
    - Connaître les pratiques DevOps
    - Connaître les outils DevOps

Objectifs

- La culture DevOps.
- Pourquoi DevOps ?
- Connaître les plus importantes terminologies.
- Pratique : Environnement de configuration.



Résoudre les problèmes sans DevOps

Projets sans DevOps

- Le syndrome du « Je ne comprends pas, ça marche sur mon poste et ne fonctionne pas ailleurs ! ».
- Les **symptômes**:
  - Commit partiels (chaque développeur commit sa partie)
  - Fichiers de configuration dépendants du poste de travail.
- **Résultat:** équipe régulièrement bloquée une demi-journée sur la correction.

Projets sans DevOps

Exemples

- **Projet Java, 60 développeurs.**
- Après 6 mois de développement
  - Créer une **release (livrable: .jar,.war)** pour pouvoir **tester.**
- Après 3 tentatives pour faire la release se soldant par des échecs.
- Constituer une équipe de 6 personnes pour effectuer cette tâche.
- **Résultat:** 2 ans de retard!

Projets sans DevOps

Projet .NET avec 25 personnes

- Sur au moins 3 **releases**: impossible de **build** pour l'équipe de test.
  - Perte d'1journée avec 6 personnes en attente et 2 développeurs pour reprendre les commits un par un.
- Sur au moins 2 **releases**: build construit sans problème. Livraison à l'équipe de test.
  - Après 2 jours de tests par l'équipe de 6 personnes, découverte d'un bug bloquant. Livraison en urgence d'un patch et d'un commit.
- Release après une semaine de tests, mais le build ne passe plus.
  - 2 jours de reprise du patch par deux développeurs.
- Donc, ensemble des tests de la semaine à repasser, car la confiance dans la constitution de la version est trop faible.
- **Résultat:** 42 jours-hommes perdus.

Projets sans DevOps

Motivation au niveau entreprise

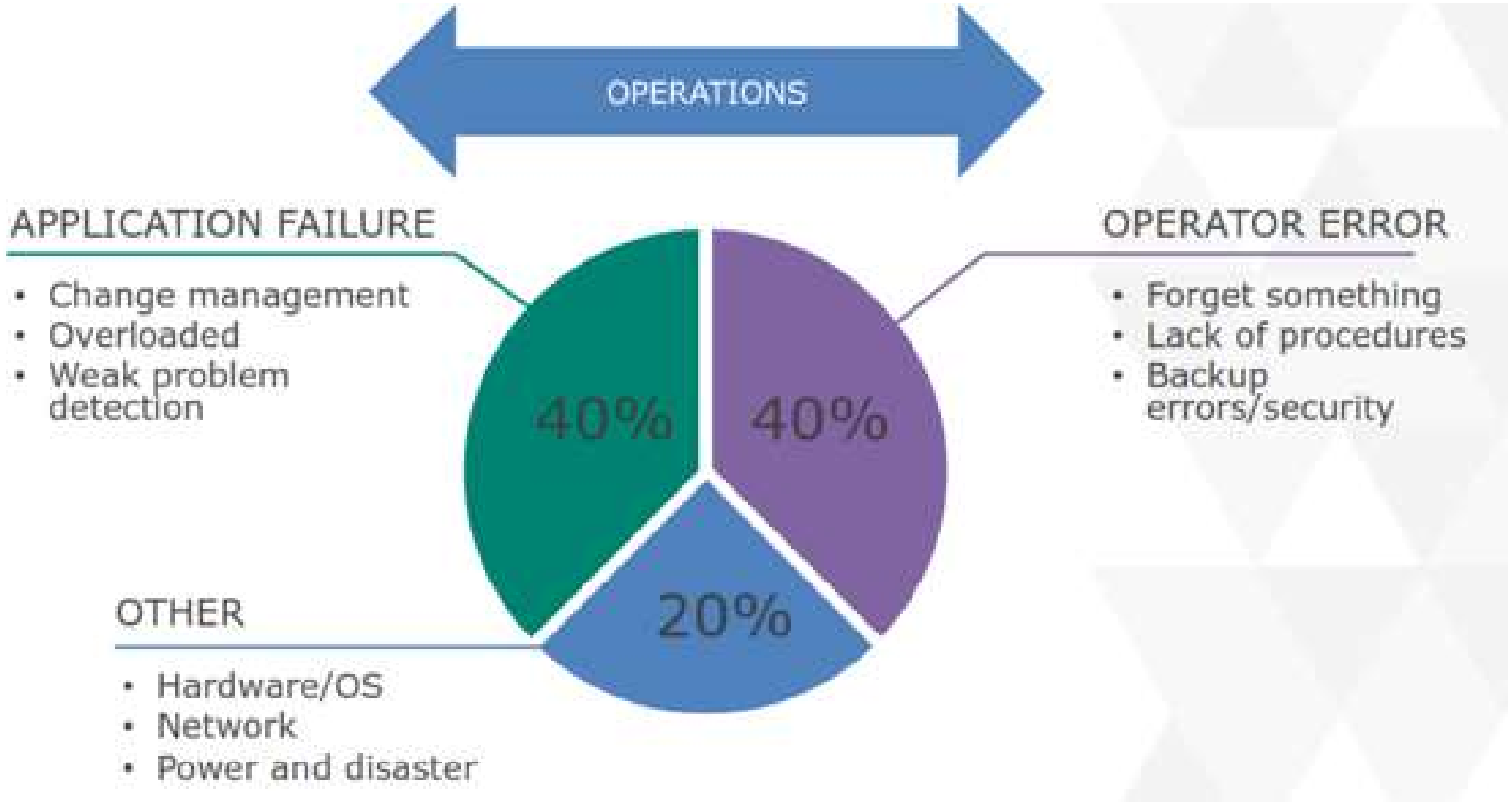
- **Marketing**
  - Demande de démonstrations non planifiées.
- **Budgets**
  - Démontrer rapidement l'avancement d'un projet (Projets gérés par tranches, par lots conditionnels: focus sur le fonctionnel important!).
- **Ressources, équipes**
  - Il faut partager les mêmes éléments d'évaluation
- => Besoin d'intégrer les évolutions d'un projet en continu

Projets sans DevOps

Motivation au niveau projet

- **Nécessité d'améliorer**
  - La qualité des livrables
    - Réduire la complexité (→ meilleure maintenabilité)
    - Adéquation
- **La traçabilité**
  - Des changements
  - Des déploiements
- **La productivité**
  - Se focaliser sur le métier, pas sur la technique
- **Principes « agiles »:** Fabriquer souvent, Tester souvent, Intégrer souvent dans le SI.

People/Process is what % of downtime ?



DevOps: Conversation en 3 étapes

Collaboration
Apporter de la valeur ajoutée au client et la valeur à l'entreprise. Responsabilité partagée

Equipe motivée pour le travail en groupe. Si des processus métier qui bloquent l'innovation ? Difficulté pour faire des changements. Toute l'équipe doit avoir une idée sur Test, Build,deploy, etc.

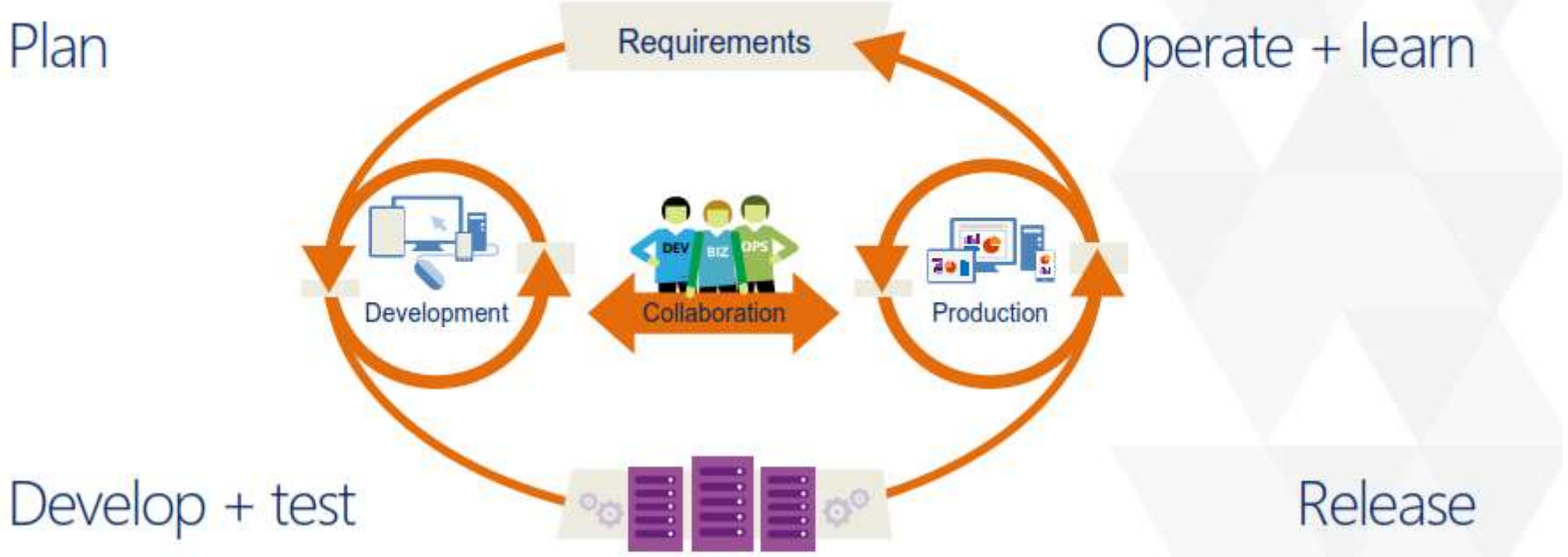
DevOps n'est pas un produit. C'est une méthodologie, culture d'entreprise.

1 People

2 Process

3 Products

Cycle de vie logiciel



L'évolution

Dev process	App architecture	Deployment pack	Infrastructure
Waterfall	Monolithic	Physical server	Data center
Agile	N-tiers	Virtual server	Hosted
Devops	Micro-services	Container	Cloud

DevOps ?

- DevOps et une **pratique,** « ***operations and development*** » participant ensemble dans tout le cycle de vie du logiciel, de la conception au développement, jusqu'à la production.
- DevOps est la combinaison de philosophies (**cultures**), de pratiques et d'outils qui augmentent la capacité d'une organisation à fournir des applications et des services à grande vitesse :
  - Faire évoluer et améliorer les produits à un rythme plus rapide que les processus de gestion de l'organisation et de l'infrastructure.
  - Cette vitesse permet aux organisations de mieux servir leurs clients et d'être plus compétitives sur le marché

(Source: définition AWS).

https://aws.amazon.com/fr/devops/what-is-devops/

Pourquoi DevOps ?

- Les avantages offerts par l'utilisation de l'approche DevOps tout au long du cycle de vie du développement logiciel incluent :
  - Développer et tester par rapport à des systèmes de type production.
  - Versions/déploiements plus fréquents, plus agile.
  - L'infrastructure est facilement disponible et peut être fournie à la demande.
  - Moins de risques de défaillance du produit une fois déployé (stabilité).
  - Temps de récupération plus rapide après des événements inattendus
  - Efficacité accrue grâce à l'automatisation
  - Maintenabilité (et évolutivité) des processus opérationnels Ops.

Avantages DevOps

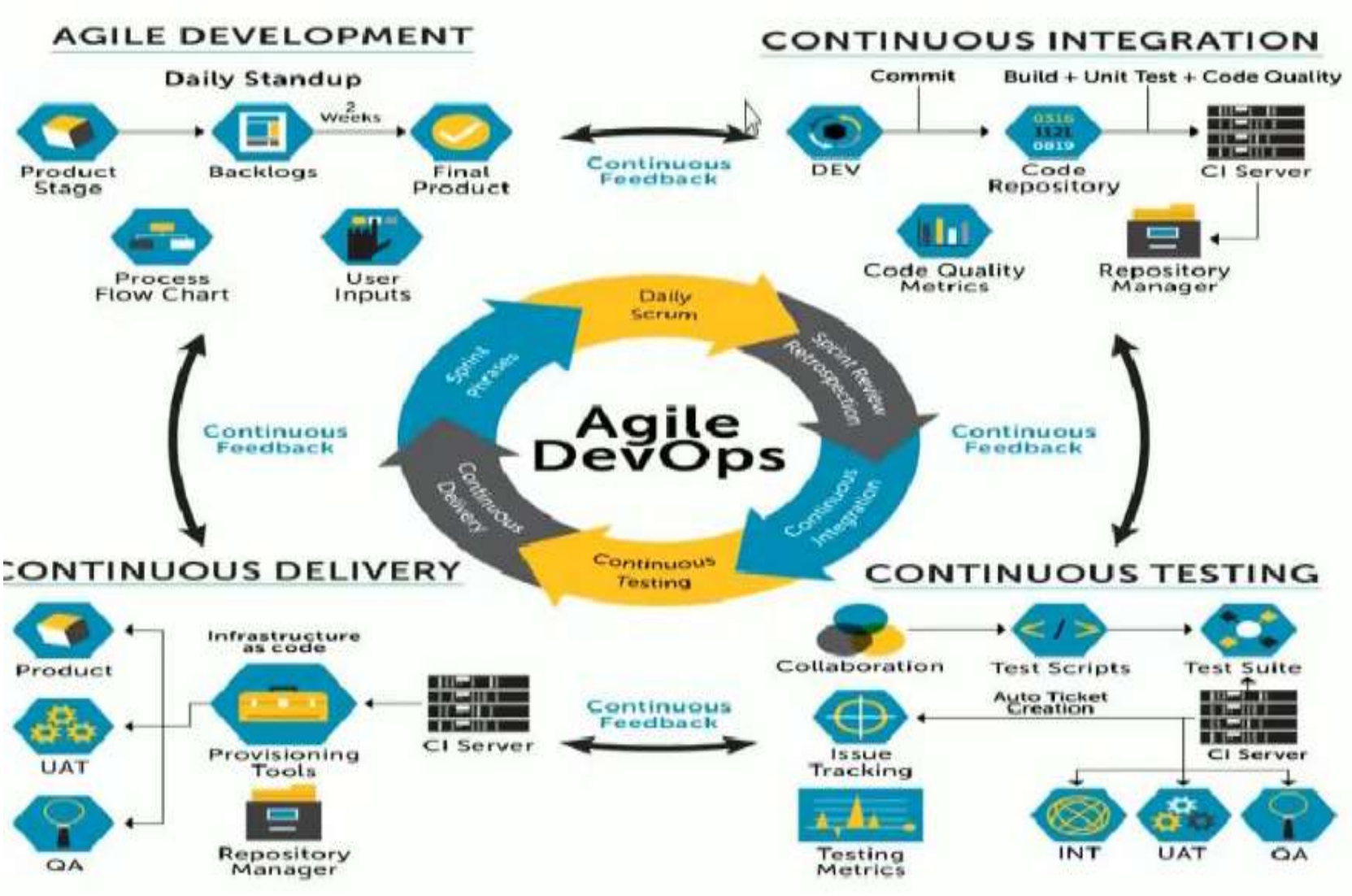
- Les principaux résultats:
  - Amélioration de la qualité des déploiements logiciels.
  - Hautes performances et fiabilité.
- Les organisations qui ont mis en œuvre DevOps ont vu ces avantages:
  - Amélioration de la qualité des déploiements logiciels
  - Versions logicielles plus fréquentes.
  - Meilleure visibilité sur le processus et les exigences informatiques
  - Le changement culturel entreprise « Collaboration/Coopération »
  - Plus de réactivité aux besoins de l'entreprise
  - Développement plus agile
  - Processus de gestion du changement plus agile
  - Amélioration de la qualité du code.

https://puppet.com/resources/report/2021-state-of-devops-report/

Avantages DevOps

- TOP 5 outils utilisés:
  - Systèmes de contrôle de version
  - Gestion de la configuration
  - Système de Ticketing
  - Surveillances des ressources (Monitoring)
  - Approvisionnement

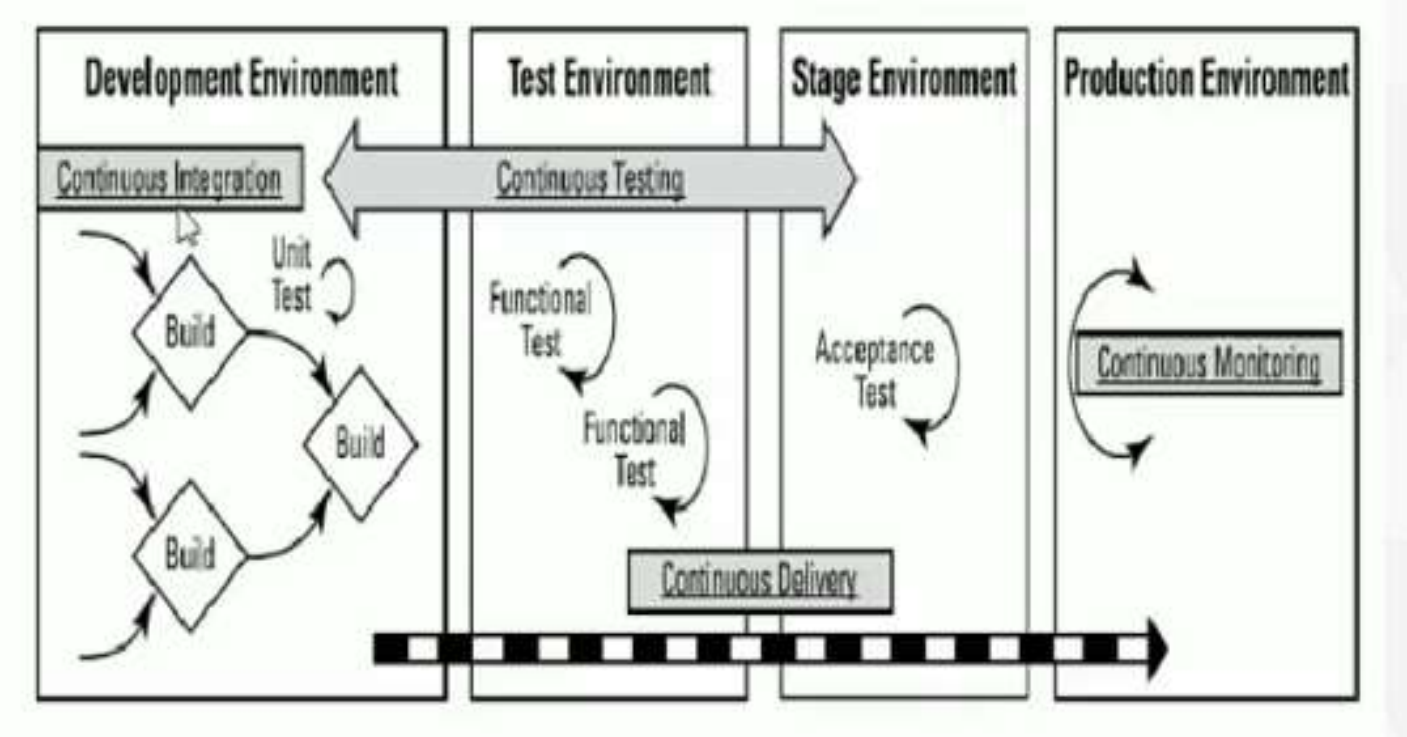
Terminologies DevOps



Terminologies DevOps

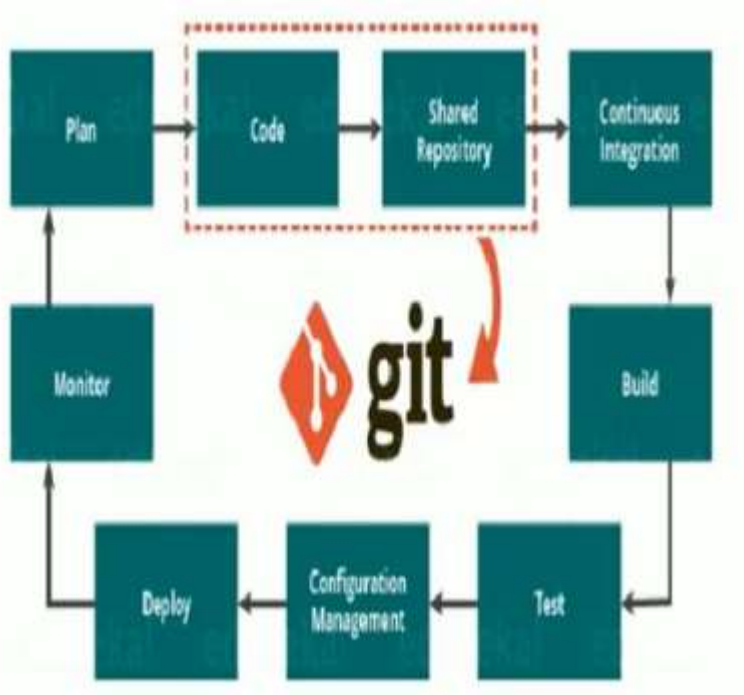
- DevOps dépend principalement de la continuité pour livrer plus rapidement avec une meilleure qualité.
- Les terminologies DevOps sont :
  - **Développement continu** (Continous Development)
  - **Intégration continue** (Continous Integration)
  - **Tests continus** (Continious Testing)
  - **Infrastructure en tant que code** (Infrastructure as Code)
  - **Livraison continue** (Continous Delivery)
  - **Déploiement continu** (Continous Deployment)
  - **Surveillance continue** (Continous Monitoring)

Terminologies DevOps



Développement continu

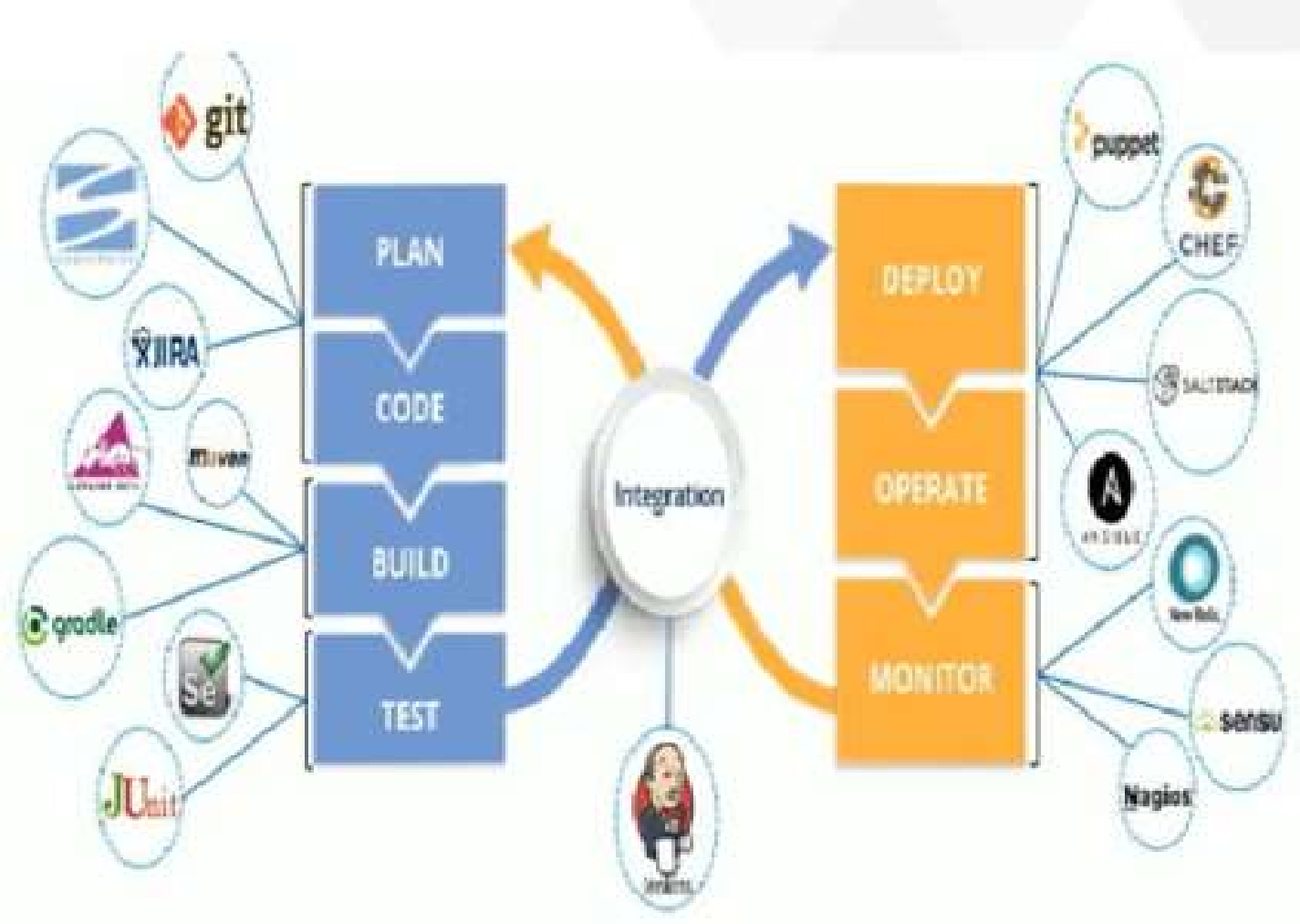
- Le logiciel est développé en continu. Cette étape implique les phases de codage et de construction et utilise des outils tels que Git et SVN pour maintenir différentes versions de code.



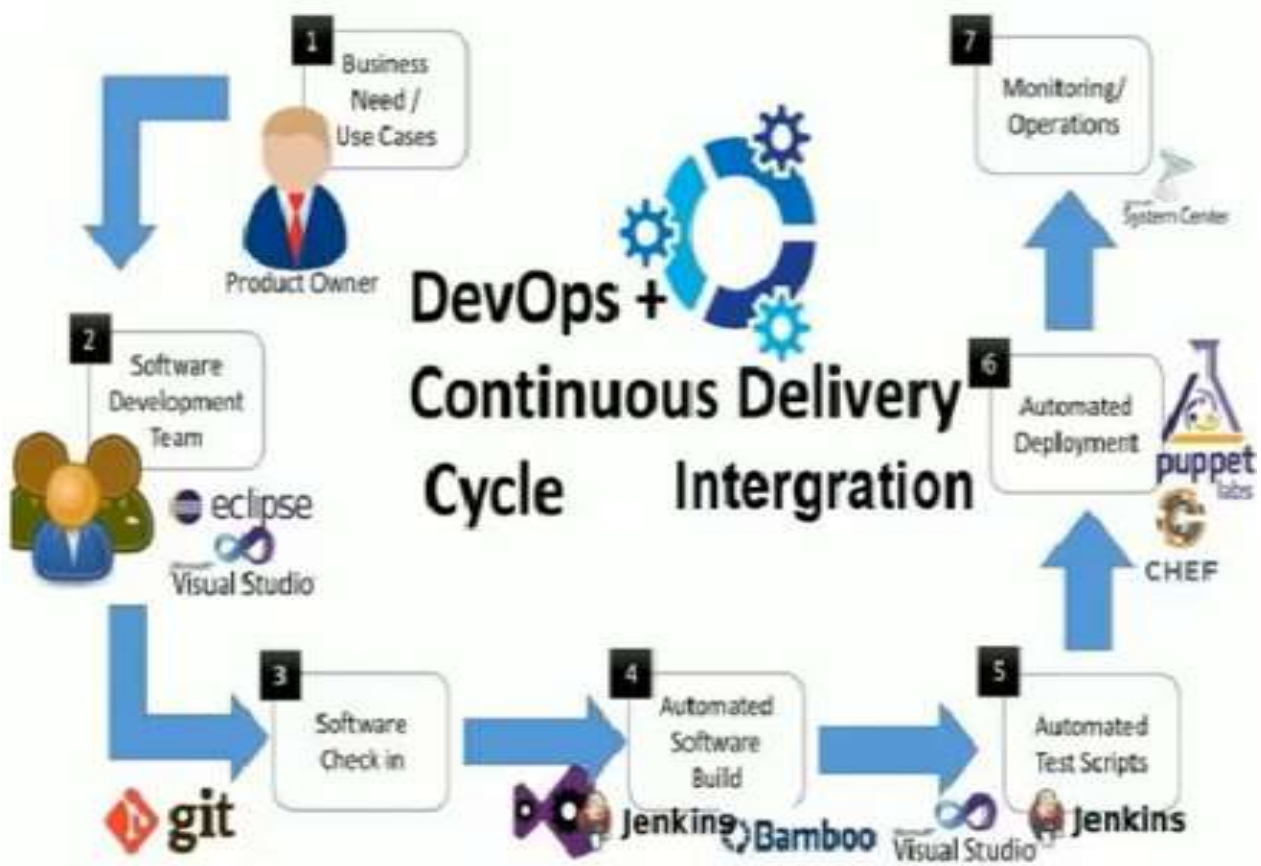
Intégration continue CI

- **L'intégration continue (CI)** est le processus d'automatisation de la construction « build » et du test du code chaque fois qu'un membre de l'équipe valide des modifications.
- CI encourage les développeurs à partager leur code et leurs tests unitaires en fusionnant leurs modifications dans un référentiel de contrôle de version partagé après chaque achèvement de petite tâche.
- La validation du code déclenche un système de génération automatisé pour récupérer le dernier code du référentiel partagé et pour construire « build », tester et valider la branche principale complète.
- Remarque : La construction du logiciel « build » est le processus de compilation, où les fichiers de code source sont convertis en code exécutable.

Intégration continue CI



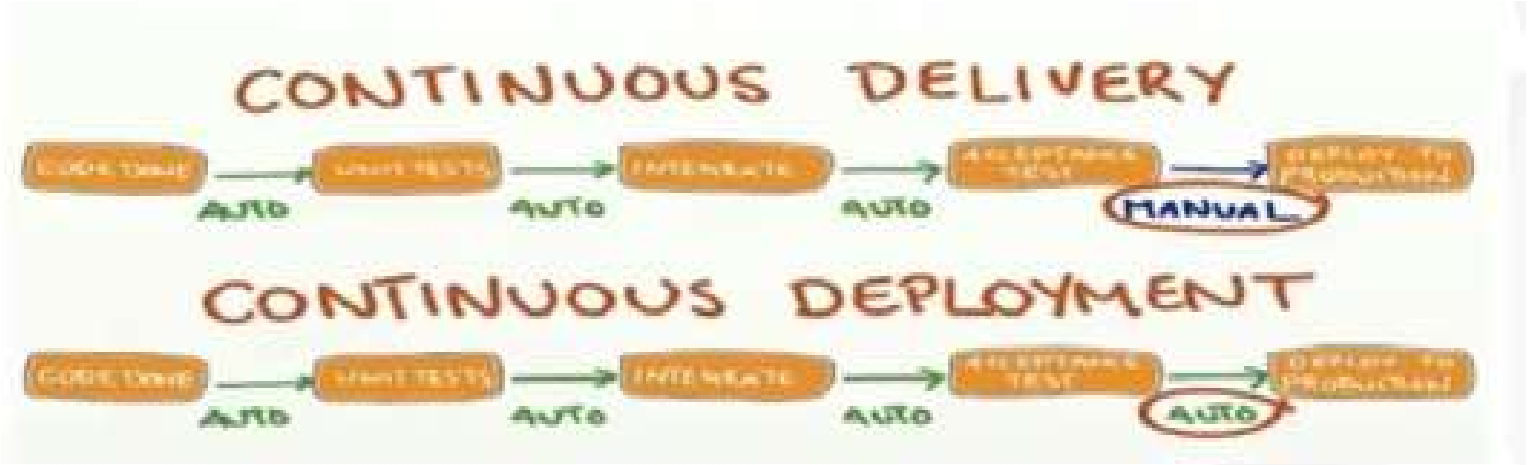
Intégration continue CI



Livraison continue vs Déploiement continu

- **Livraison continue CD (Continous Delivery):**
  - La capacité de fournir le logiciel à n'importe quel environnement donné à tout moment selon les besoins.
  - Non seulement pour être déployé à tout moment mais aussi pour avoir toute la configuration nécessaire pour pousser en production.
- **Déploiement continu (Continous Deployment)**
  - Pour que votre application reste déployée à tout moment ou même qu'elle soit automatiquement publiée dans un environnement de production si la dernière version réussit tous les tests automatisés.
  - C'est du développement **à la production directement**. C'est le déploiement du code dans l'environnement de production.

Livraison continue vs déploiement continu

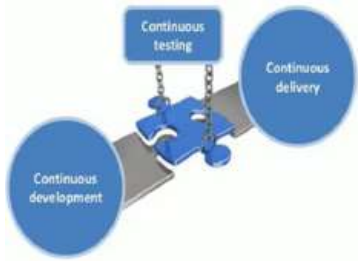


Infrastructure en tant que code (Infrastructure as Code)

Infrastructure en tant que code (gestion de configuration)

- Il s'agit d'écrire du code (qui peut être fait à l'aide d'un langage de haut niveau ou de n'importe quel langage descriptif) pour **gérer les configurations et automatiser** la mise à disposition de **l'infrastructure** en plus des déploiements.
- C'est l'approche de définition de l'infrastructure informatique et réseau à travers un code source qui peut ensuite être traité comme n'importe quel logiciel.
- Vous pouvez déployer plus rapidement, avec une plus grande fiabilité, car vous n'avez plus besoin de tracer et de déployer manuellement chaque étape.

Tests Continus



- Les **tests continus** sont le processus d'exécution de tests automatisés dans le cadre du pipeline de livraison de logiciels afin d'obtenir le plus rapidement possible des commentaires sur les risques commerciaux associés à une version logicielle.
- **Types de tests automatisés :**
  - Environnement de développement : vérification du code
  - Environnement d'intégration continue : permet un test et une vérification continus de la version
  - Environnement de test de bout en bout,
  - Environnement de test de pré-production et de performance QA.

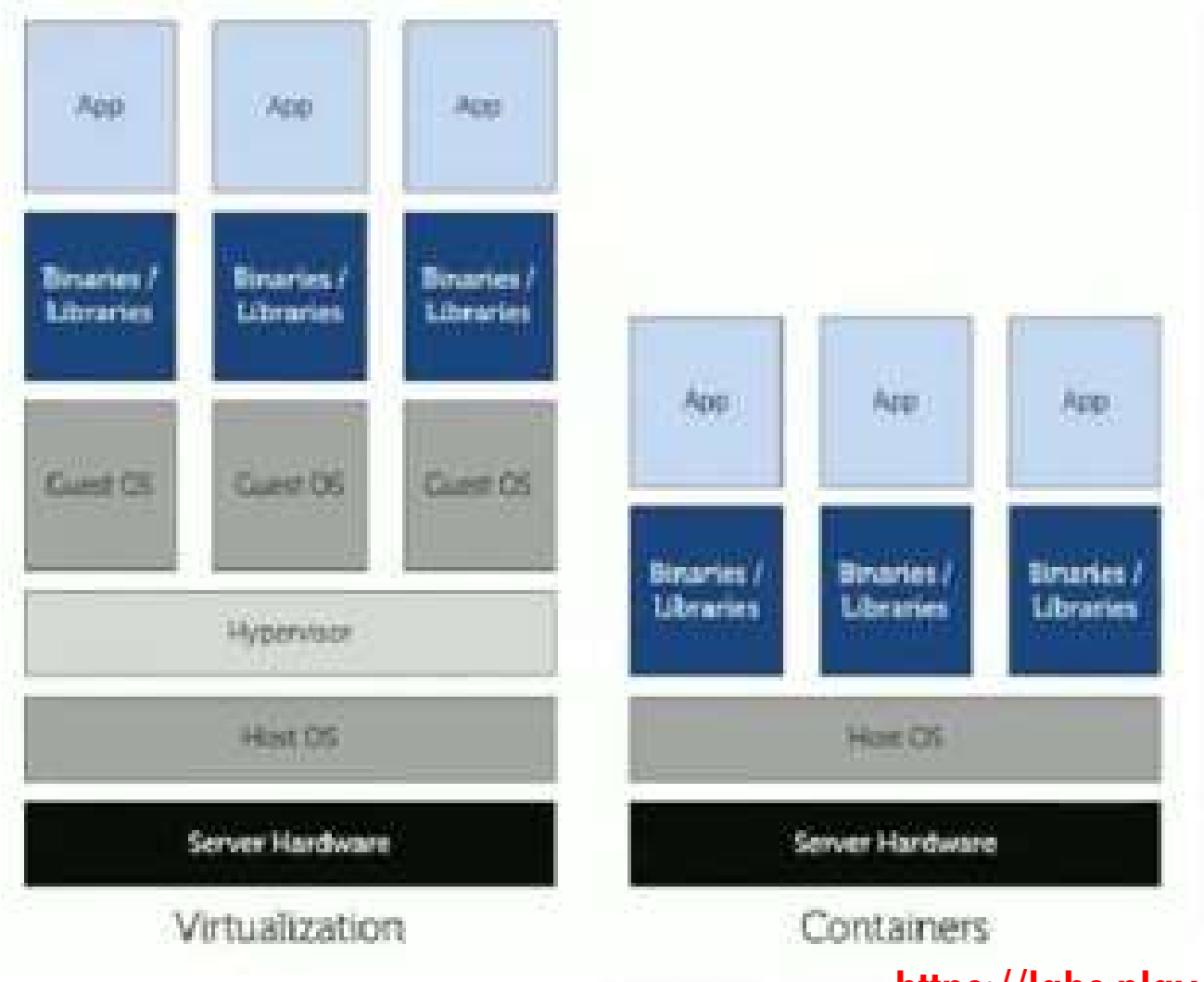
Surveillance continue (Continous Monitoring)

- Les équipes opérationnelles utilisent des outils capables de surveiller les performances et les problèmes des applications. Cela peut également nécessiter qu'ils travaillent avec l'équipe des développeurs pour créer une auto-surveillance ou une collecte d'analyses.
- **Surveiller**
  - Les logs de l'application.
  - Serveur « Server Health »
  - Détection d'intrusion
  - Déploiements

Conteneurisation

- La **conteneurisation** d'applications est une méthode de virtualisation au niveau du système d'exploitation pour déployer et exécuter des applications distribuées sans lancer une machine virtuelle (VM) entière pour chaque application. Au lieu de cela, plusieurs systèmes isolés sont exécutés sur une seule hôte de contrôle et accèdent à un seul noyau.
- Les conteneurs d'application contiennent les composants tels que les fichiers, les variables d'environnement et les bibliothèques nécessaires pour exécuter le logiciel souhaité.
- Si la virtualisation (c'est-à-dire les machines virtuelles) a été conçue pour traiter la consolidation des serveurs et des ressources, la conteneurisation a été conçue pour résoudre un problème plus moderne : les problèmes de gestion.

Conteneurisation

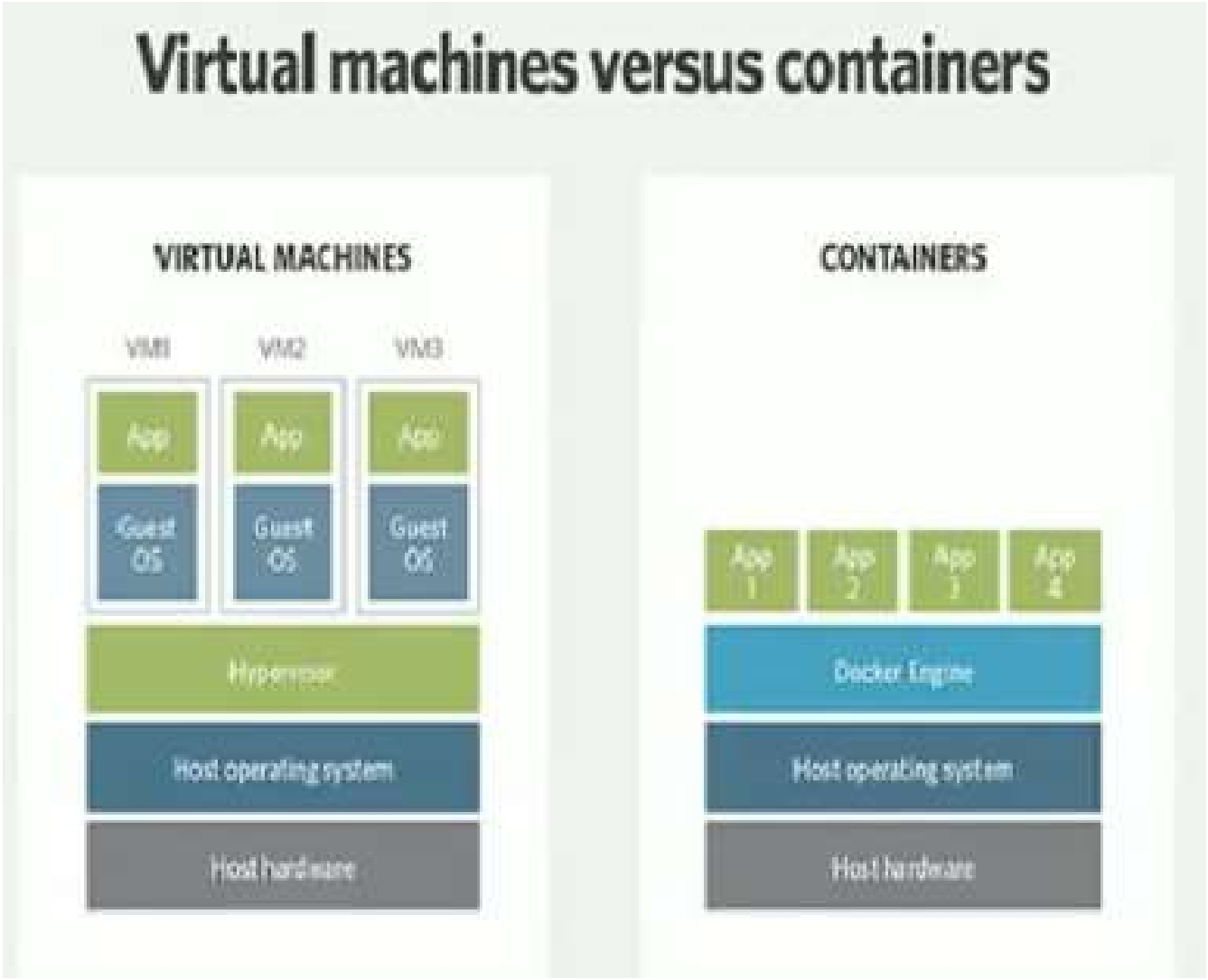


https://labs.play-with-docker.com/

Conteneurisation

Machines virtuelles (VM)	Conteneurs
Représente la virtualisation au niveau matériel	Représente la virtualisation du système d'exploitation
Poids lourd	Poids léger
Performances limitées	Performances natives
Démarrage plus lent	Démarrage en quelques secondes
Entièrement isolé et donc plus sécurisé	Isolation au niveau du processus et donc moins sécurisée

Conteneurisation



Résumé

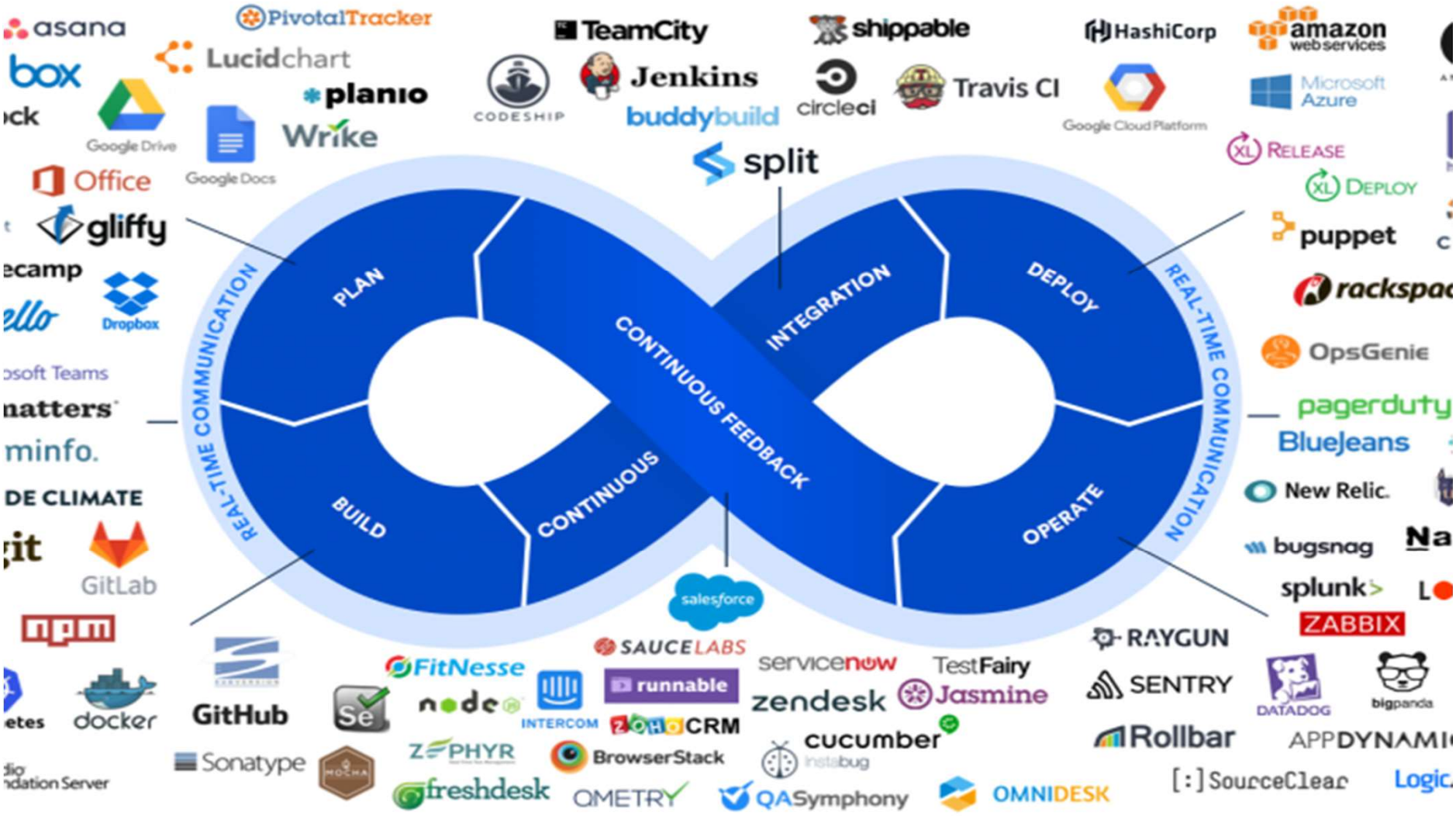
- Actuellement, DevOps s'apparente plus à un mouvement philosophique, pas encore à un recueil précis de pratiques, descriptives ou prescriptives. **Gene Kim.**
- DevOps dépend de l'**agilité** et de la tendance des **microservices**
- DevOp est la tendance future du développement des logiciels
- Il existe de nombreux outils sur le marché, mais nous avons juste besoin d'apprendre quelques outils pour produire une culture DevOps efficace.

Résumé


04 TEST
08 MONITOR
03 BUILD
07 OPERATE
DEV
OPS
02 CODE
06 DEPLOY
01 PLAN
05 RELEASE

Outils DevOps

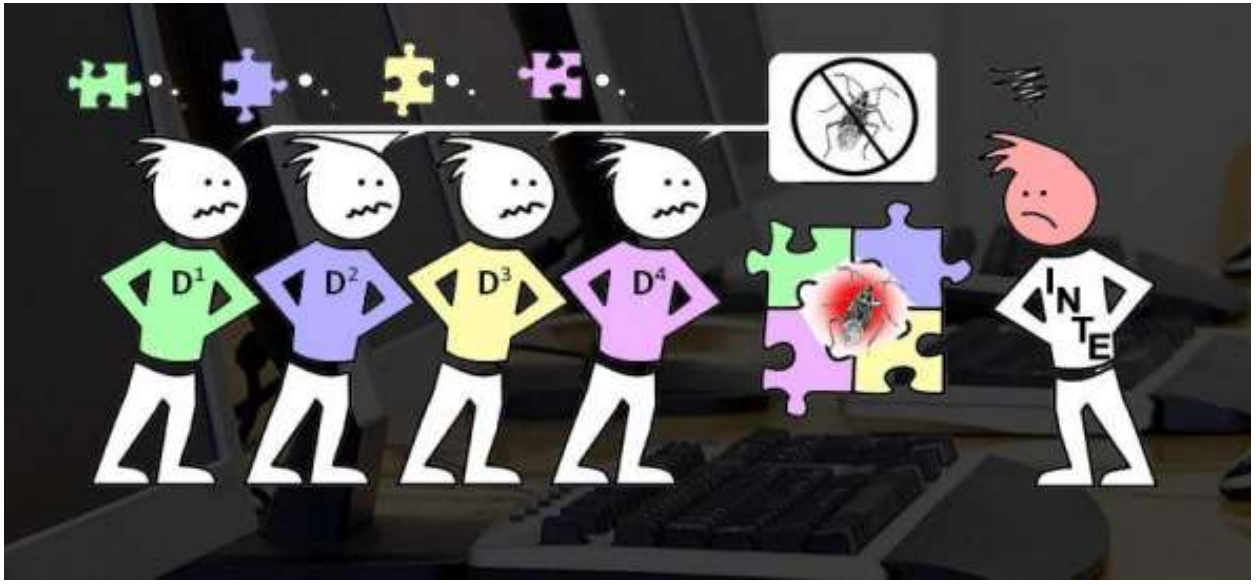
Outils DevOps



Mots clés de perspectives

- **MLOps**
- **GitLab CI/CD** https://docs.gitlab.com/ee/ci/
- **DevOps** .NET https://www.devopsschool.com/path/
- **SRE DevOps**

Intégration continue CI



Projets sans Intégration Continue (IC)

- Le syndrome du « Je ne comprends pas, ça marche sur mon poste et ne fonctionne pas ailleurs ! ».
- Les **symptômes**:
  - Commit partiels (chaque développeur commit sa partie)
  - Fichiers de configuration dépendants du poste de travail.
- **Résultat:** équipe régulièrement bloquée une demi-journée sur la correction.

Projets sans Intégration Continue (IC)

Exemples

- **Projet Java, 60 développeurs.**
- Après 6 mois de développement
  - Créer une **release (livrable: .jar,.war)** pour pouvoir **tester.**
- Après 3 tentatives pour faire la release se soldant par des échecs.
- Constituer une équipe de 6 personnes pour effectuer cette tâche.
- **Résultat:** 2 ans de retard!

Projets sans Intégration Continue (IC)

Projet .NET avec 25 personnes

- Sur au moins 3 **releases**: impossible de **build** pour l'équipe de test.
  - Perte d'1journée avec 6 personnes en attente et 2 développeurs pour reprendre les commits un par un.
- Sur au moins 2 **releases**: build construit sans problème. Livraison à l'équipe de test.
  - Après 2 jours de tests par l'équipe de 6 personnes, découverte d'un bug bloquant. Livraison en urgence d'un patch et d'un commit.
- Release après une semaine de tests, mais le build ne passe plus.
  - 2 jours de reprise du patch par deux développeurs.
- Donc, ensemble des tests de la semaine à repasser, car la confiance dans la constitution de la version est trop faible.
- **Résultat:** 42 jours-hommes perdus.

Projets sans Intégration Continue (IC)

Motivation au niveau entreprise

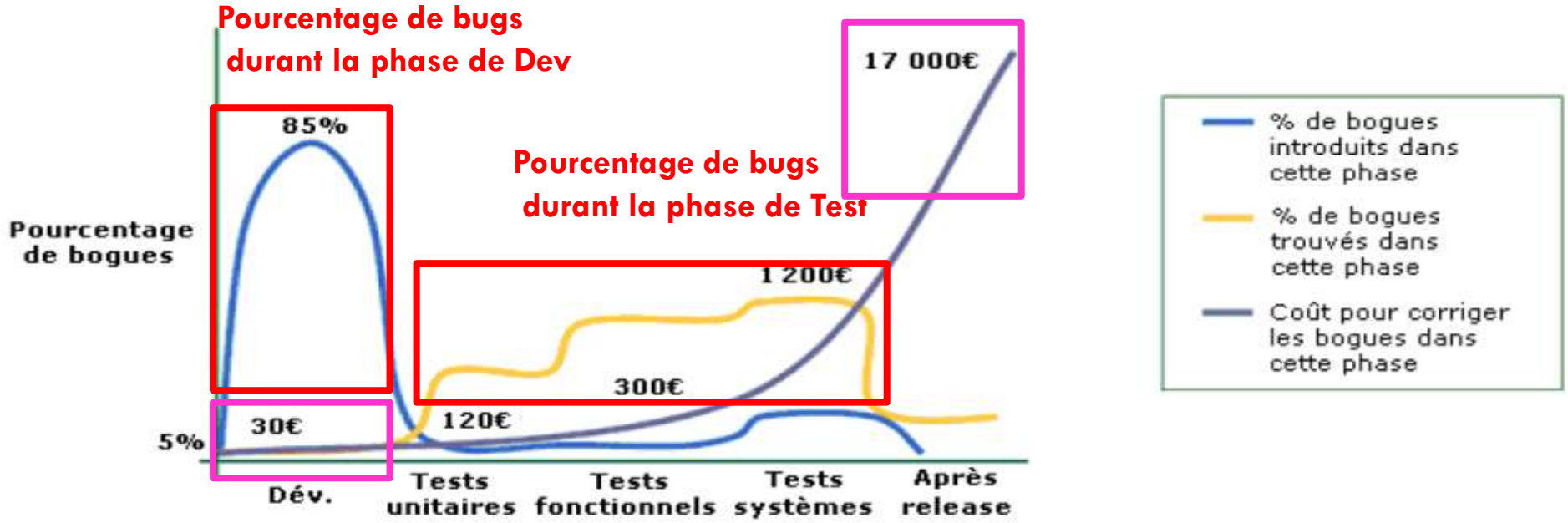
- **Marketing**
  - Demande de démonstrations non planifiées.
- **Budgets**
  - Démontrer rapidement l'avancement d'un projet (Projets gérés par tranches, par lots conditionnels: focus sur le fonctionnel important!).
- **Ressources, équipes**
  - Il faut partager les mêmes éléments d'évaluation
- => Besoin d'intégrer les évolutions d'un projet en continu

Projets sans Intégration Continue (IC)

Motivation au niveau projet

- **Nécessité d'améliorer**
  - La qualité des livrables
    - Réduire la complexité (→ meilleure maintenabilité)
    - Adéquation
- **La traçabilité**
  - Des changements
  - Des déploiements
- **La productivité**
  - Se focaliser sur le métier, pas sur la technique
- **Principes « agiles »:** Fabriquer souvent, Tester souvent, Intégrer souvent dans le SI.

Projets sans Intégration Continue (IC)

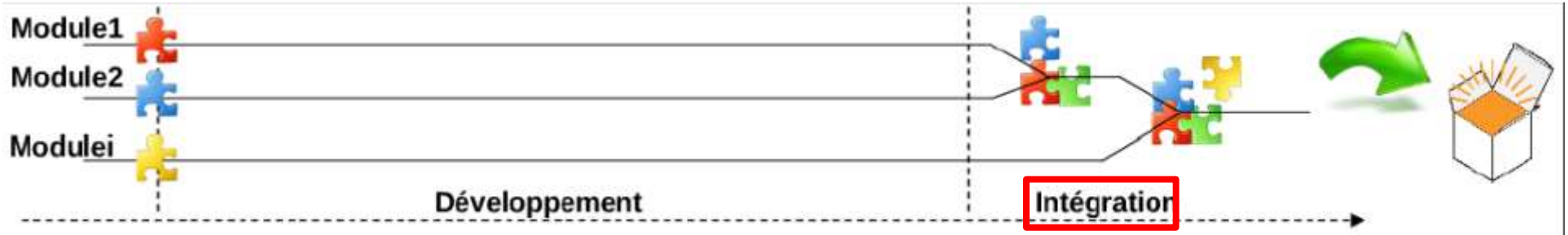


- **Plus une erreur est détectée tard et plus le coût de la correction sera élevé !**

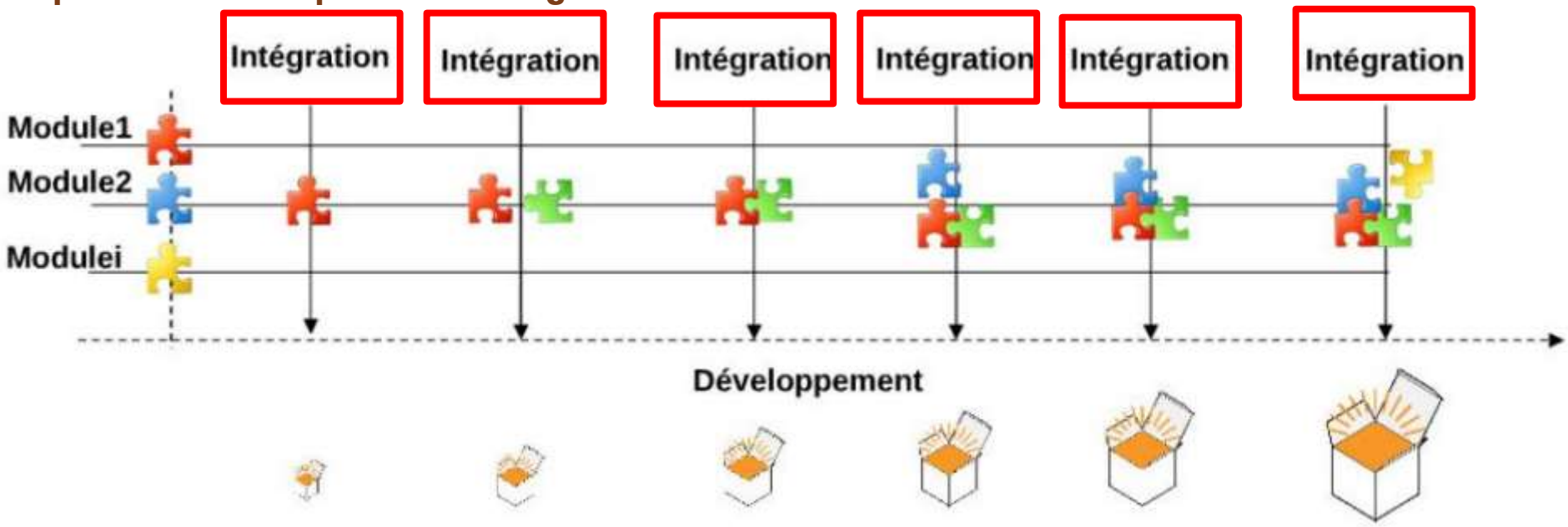
Les 5% de bugs découverts après release représentent 95% des coûts de correction.

Comparaison avant et après IC

Avant la mise en place de l'intégration continue

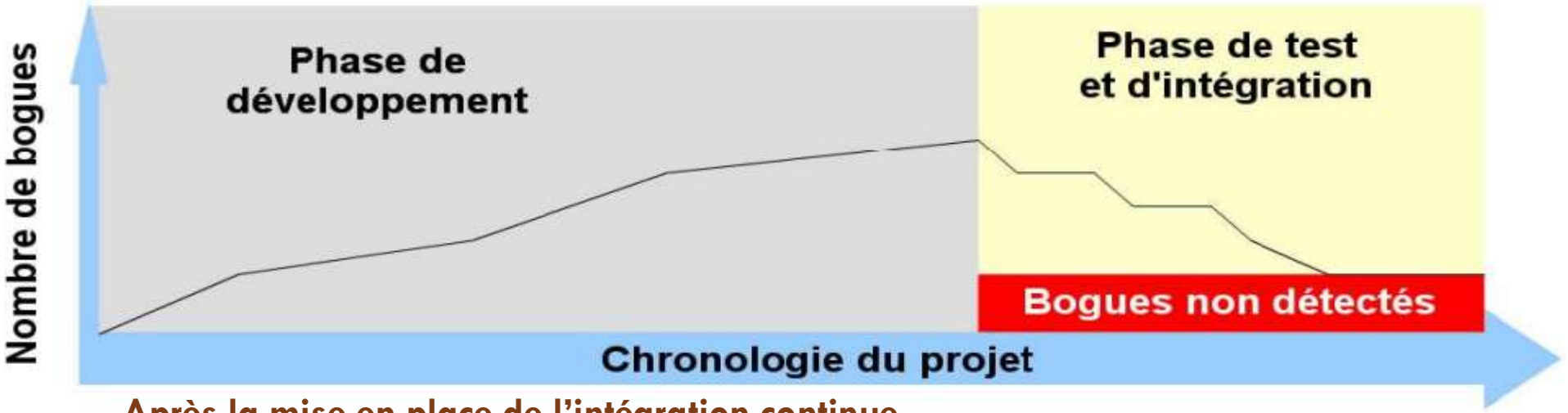


Après la mise en place de l'intégration continue

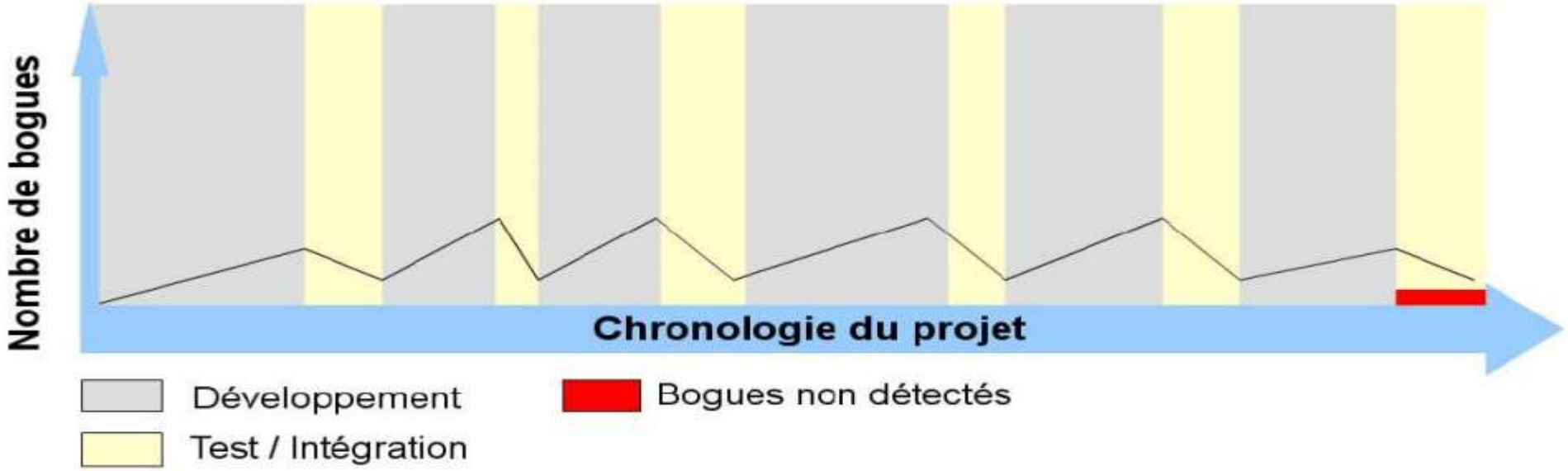


Comparaison avant et après IC

53 **Avant la mise en place de l'intégration continue**



Après la mise en place de l'intégration continue



Intégration Continue (IC)

- **Martin Flower**

... UNE PRATIQUE DE DEVELOPPEMENT LOGICIEL OU LES MEMBRES D'UNE EQUIPE INTEGRENT LEUR TRAVAIL FREQUEMMENT, HABITUELLEMENT CHACUN AU MOINS UNE FOIS PAR JOUR – CE QUI ENTRAINE PLUSIEURS INTEGRATIONS PAR JOUR. CHAQUE INTEGRATION EST VALIDEE PAR UN 'BUILD' AUTOMATIQUE (CE QUI INCLUT LES TESTS) POUR DETECTER LES ERREURS D'INTEGRATION AUSSI VITE QUE POSSIBLE.

CETTE APPROCHE PERMET DE REDUIRE SIGNIFICATIVEMENT LES PROBLEMES D'INTEGRATION ET PERMET A UNE EQUIPE DE DEVELOPPER DES LOGICIELS DE QUALITE PLUS RAPIDEMENT...

"L'intégration continue est le principe de faire d'un processus d'intégration logiciel un « non-évènement»."

Martin Fowler

Intégration Continue (IC)

- Intégration continue: méthode agile:
  - **Fabriquer** souvent (« **build** »)
  - **Tester** souvent(« **test** »)
  - **Intégrer** souvent (« **Integrate** »).
- Bonnes pratiques:
  - Maintenir un dépôt unique de code source versionné (**GIT**)
  - Automatiser les compilations (**Maven**)
  - Rendre les compilations auto-testantes (**Junit**)
  - Tout le monde **commit** tous les jours
  - Tout commit doit amener à une compilation du tronc sur **une machine d'intégration**
  - Maintenir une construction de build courte

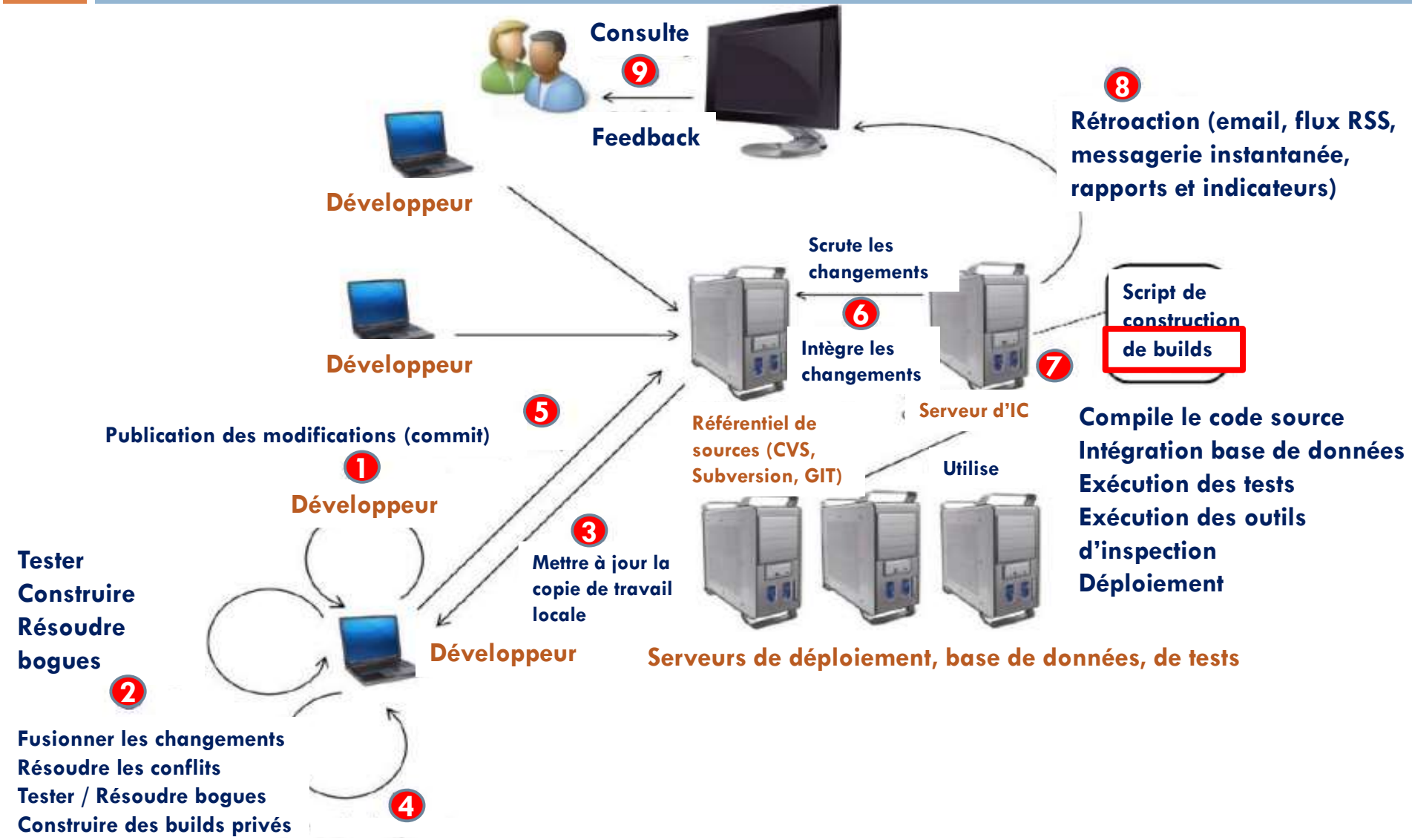
Intégration Continue (IC)

- Tester dans un environnement de production cloné
- Rendre disponible facilement **le dernier exécutable (Nexus)**
- Tout le monde doit voir ce qui se passe **(Jenkins Email, Sonar)**
- Automatiser le déploiement

Intégration Continue (IC)

- Intégration continue industrialise (automatise):
  - La génération des packages **(.jar, .war), Maven;**
  - Les tests unitaires **JUnit;**
  - La qualimétrie **Sonar;**
  - Le reporting **Jenkins.**
- **Objectif: vérifier de façon automatique et à chaque modification de code source, que le résultat des modifications ne produit pas de régression de l'application en cours de développement.**

Le processus d'intégration Continue (IC)



Le processus d'intégration Continue (IC)

- **Build**, pas seulement une compilation :
  - **Chargement** de la dernière version du projet depuis le GIT
  - **Compilation et Exécution** des tests unitaires
  - **Inspection** du code (en vue de générer les métriques de qualité);
  - **Construction** des releases (JAR, WAR, EAR, etc)
  - **Déploiement** de l'application sur l'environnement de développement (copie des librairies, configuration et redémarrage + sur le remote repository dans le cadre de Maven)
  - **Exécution** des tests d'intégration;
  - **Génération** de la documentation, des rapports, des notes de release (par exemple Javadoc ou les rapports Maven).
- **Un build, une fois construit, est en fait un package de livrables pour le client qui ont été pleinement testés. Il contient tous les fichiers nécessaires (exécutables, documentation, etc,).**

Le processus d'intégration Continue (IC)

1 2 3 4 5

- Les développeurs publient leurs modifications de code sur GIT(commit). Ils auront pris soin d'écrire des tests sur leurs nouveaux codes et de lancer des **builds privés** (=locaux) sur leurs postes. **C'est l'étape la plus importante du processus.**

6 7

- Le serveur d'intégration scrute les changements sur GIT. À chaque fois qu'il analyse un changement, il réalise un **build d'intégration**. C'est une méthode d'automatisation « on stimulus ». Parce que le serveur d'intégration continue utilise un système de build déjà existant, il est important pour l'équipe de développement de consacrer du temps et des efforts à:
  - Diminuer, optimiser au maximum le temps de création d'un build;
  - Écrire des tests;
  - Inclure ces tests dans la construction du build

Le processus d'intégration Continue (IC)

8 9

- A chaque fois qu'un **build est réalisé,** un **courriel de notification est envoyé à tous les membres de l'équipe.** Ce courriel contient notamment le statut du dernier build. En cas d'échec de build, les développeurs pourront travailler directement sur sa correction.

Le processus d'intégration Continue (IC)

- **L'intégration locale (le développeur):**
  - Le développeur récupère une copie du dernier code source en date à partir du GIT.
  - Il fait toutes les modifications au code dont il a besoin pour accomplir sa tâche.
  - Une fois qu'il a terminé, le développeur lance un build privé sur sa machine.
  - Si le build réussi, le développeur peut commencer à songer à commiter
  - Le problème est que d'autres développeurs auront peut-être commité leur codes avant lui.
  - Pour vérifier, le développeur doit donc mettre à jour sa copie locale pour récupérer les dernières modifications.
  - Ensuite, il doit refaire un build privé et ..
  - …Si le build privé rate, le développeur doit résoudre le problème.
  - Une fois que le build réussit avec le code synchronisé, le développeur pourra enfin commiter ses changements.

Le processus d'intégration Continue (IC)

- **L'intégration locale (le développeur):**
  - Une fois le code commité, le serveur d'intégration lance un build d'intégration (dans le cas d'une automatisation « on stimulus »)
  - Le développeur reçoit un courriel quelques secondes plus tard lui indiquant le statut du build.

Les 7 bonnes pratiques

- Les 7 **bonnes pratiques du développeur:**
  - **Commiter** le code fréquemment [« Commit early, commit often »]
  - **Commiter** du code bon [« Never commit broken code »]
  - **Résoudre** les builds ratés rapidement [« Fix build failures immediately »]
  - **Tous les tests et les rapports d'inspection doivent passer** [« All tests and inspections must pass »]
  - **Créer des builds privés** [« Run private builds »]
  - Garder des builds dans le vert tout au long de la journée [« Keep builds in the green »]
  - **Éviter de récupérer du mauvais code** [« Avoid getting broken code »]

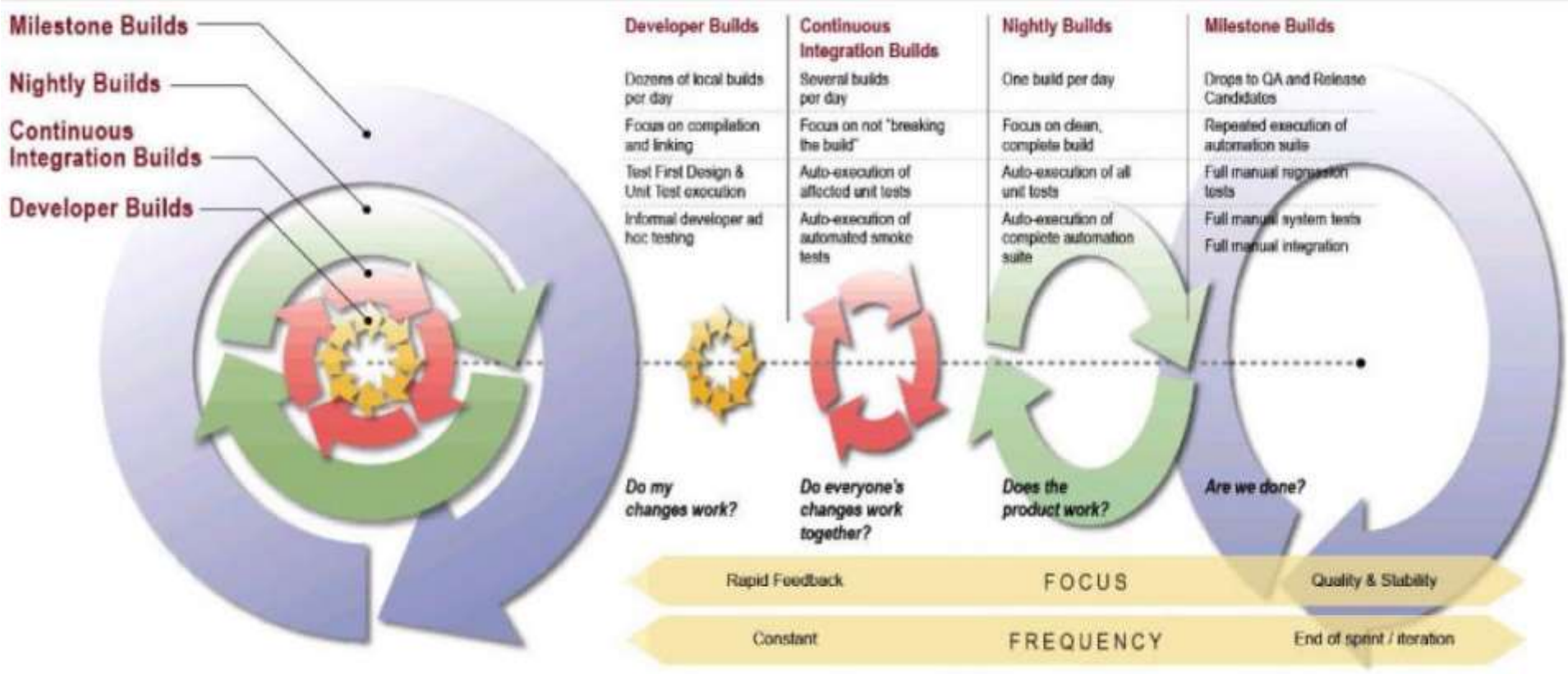
Les différents types de build

- Le **build** est vraiment l'un des mots les plus importants. Il est au **cœur du concept d'intégration continue**. Il existe de multiples sorte de builds.
- Le **build** n'est pas qu'une simple compilation, c'est:
  - **Chargement** de la dernière version du projet depuis le gestionnaire de sources;
  - **Compilation**;
  - **Exécution** des tests unitaires;
  - **Inspection** du code;
  - **Construction** des releases;
  - **Déploiement** de l'application sur l'environnement de dév;
  - **Exécution** des tests d'intégration;
  - **Génération** de la documentation, des rapports, etc.

Les différents types de build

- **Private build, Integration build, Release build**

Build gros mais moins fréquent



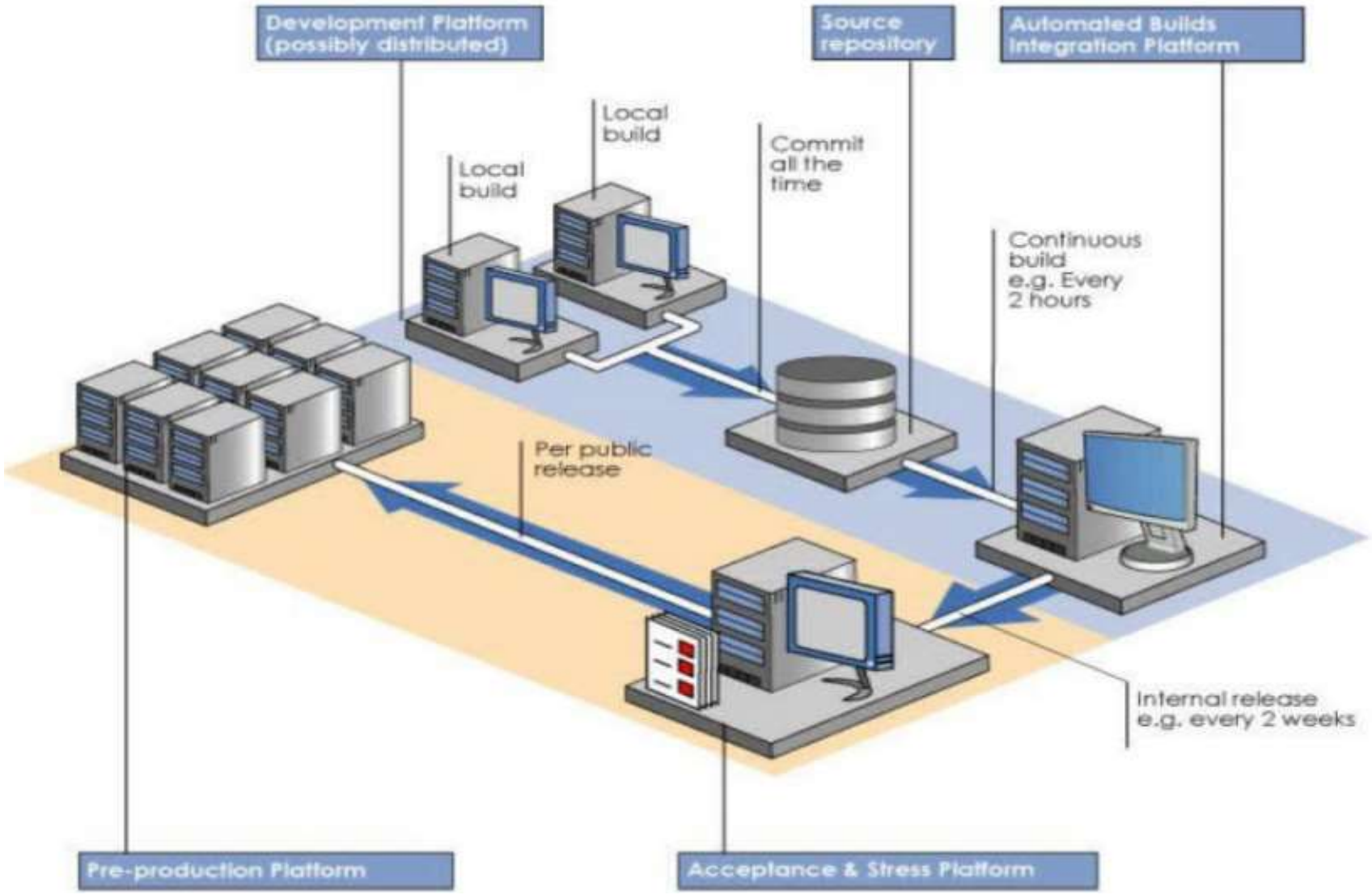
Les différents types de build

- **Builds privés**
  - Le build privé désigne le build du développeur ou **build local** qui de l'ordre d'une dizaine par jour. Ce build sert à se rendre compte rapidement si les changements que l'on a effectué marchent et commits corrects.
- **Build d'intégration de jour**
  - Le **build d'intégration** de jour est effectué par le serveur d'intégration continue. Ce build est réalisé plusieurs fois par jour et de manière automatique. Il sert à vérifier que les changements de tout le monde marchent. Tous les jours, à chaque commit des développeurs ou à intervalles réguliers, les projets sont compilés, testés, déployés et les résultats sont générés.

Les différents types de build

- **Build d'intégration de nuit**
  - Le build d'intégration de nuit est lui aussi effectué par le serveur d'intégration continue mais qu'une fois par jour. Il sert à vérifier que le produit marche. **Ce build est plus complet que le précédent, tous les tests sont réalisés.** Par exemple, les tests de performance qui sont ignorés lors des builds de jour. Les builds d'intégration de jour sont plus lights que ceux de nuit pour éviter un temps d'attente trop long pour les développeurs.
- **Conseils pour les builds d'intégration**
  - Il faut garder en tête que les builds de jour ne doivent pas excé 10 à 15 minutes. Pour les builds de nuit, la durée importe moins mais il faut éviter que cela dépasse une heure.

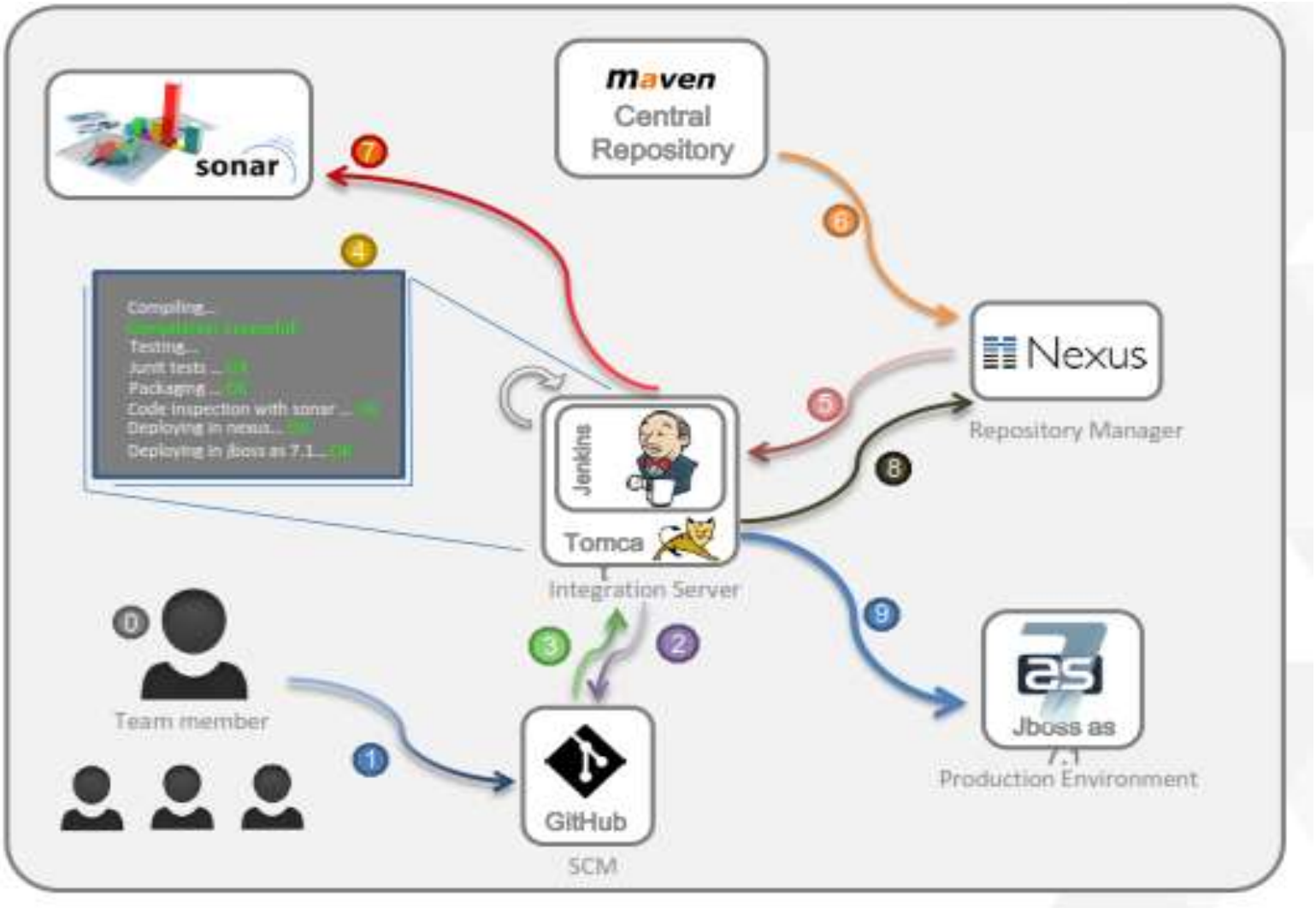
Les différents types de build



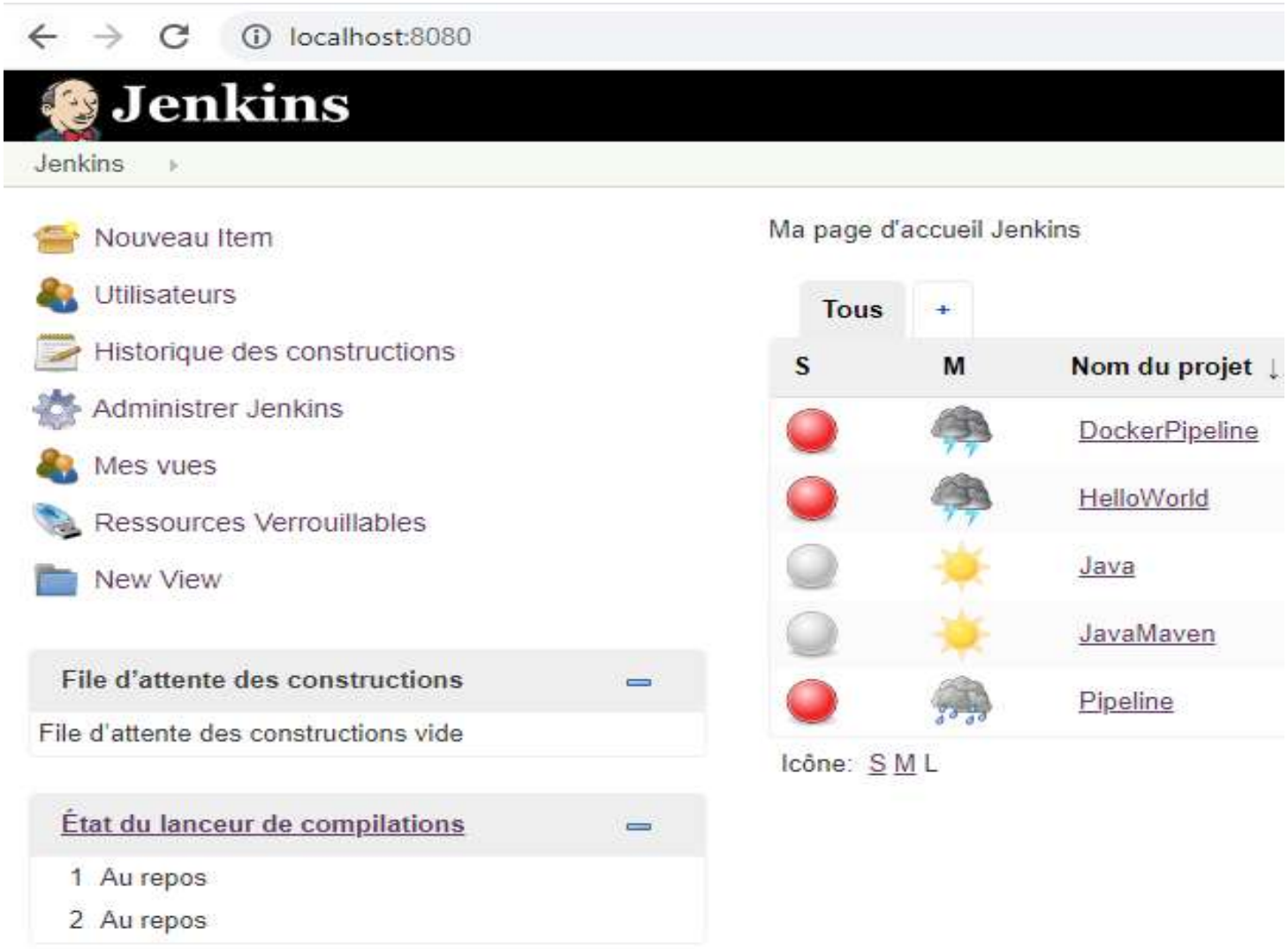
Outils Intégration Continue

- **Gestionnaires de versions:**
  - **GIT,** SVN(subversion); CVS; ClearCase; SourceSafe, etc.
- **Serveurs d'intégration continue**
  - Cruise Control; Banboo; **Jenkins**(Hudson);Continuum, etc.
- **Gestionnaires de dépendances**
  - **Maven**; Ivy, etc
- **Assemblage et building:**
  - **Maven**; Ant; Buildr; etc.
- **Outils de test**
  - XUnit; **JUnit**, Mock object; Selenium, etc.
- **Outils de qualimétrie (génération de métriques)**
  - **Sonar**; PMD; CheckStyle, etc.
- **Outils de Repository Manager**
  - **Nexus**

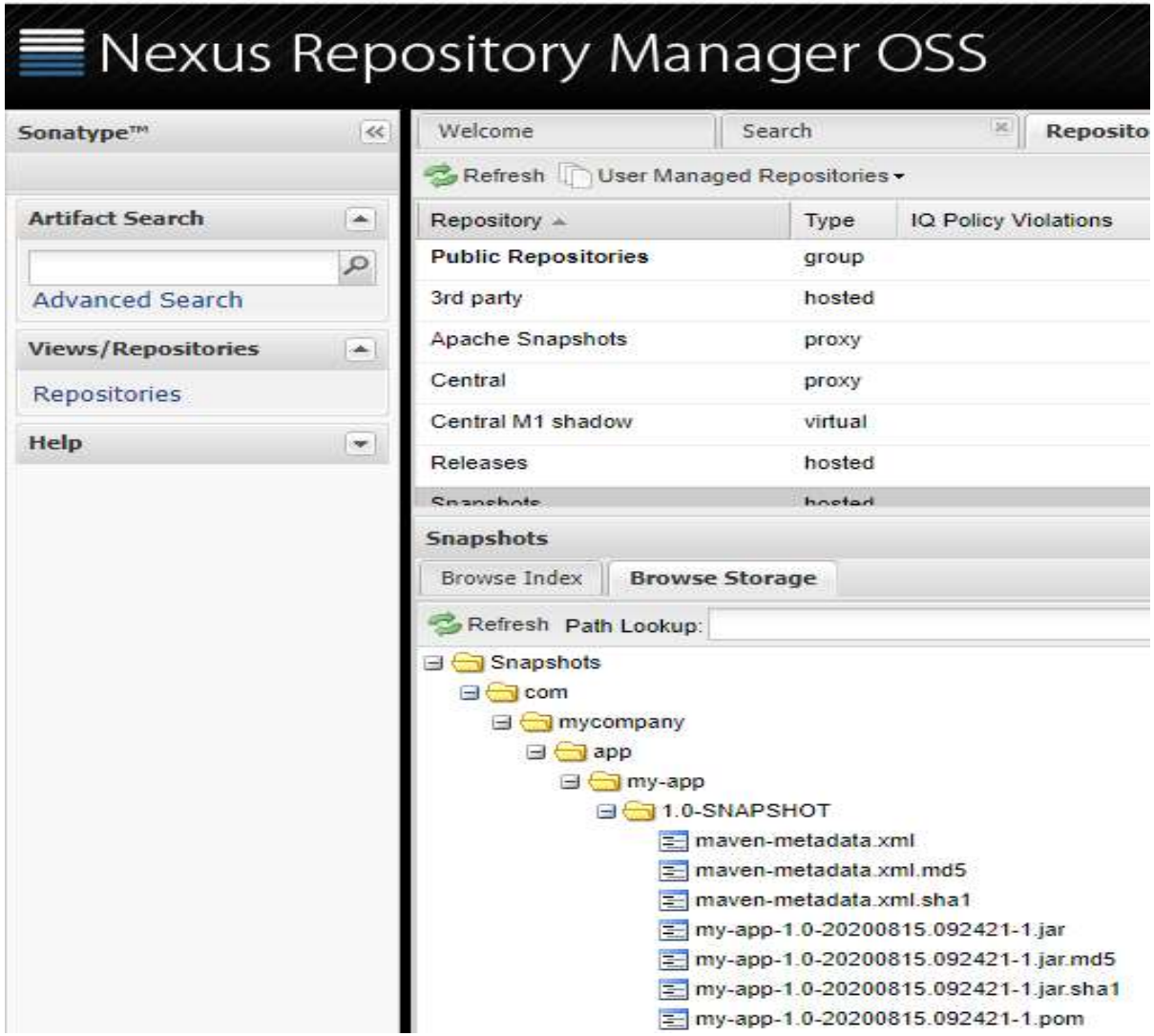
Outils


sonar
maven
Central
Repository
Nexus
Repository Manager
Compiling...
Testing...
Junit tests ...
Packaging ...
Code inspection with sonar ...
Deploying in nexus...
Deploying in jboss as 7.1...
Jenkins
Tomca
Integration Server
Team member
GitHub
SCM
Jboss as
Production Environment
0
1
2
3
4
5
6
7
8
9

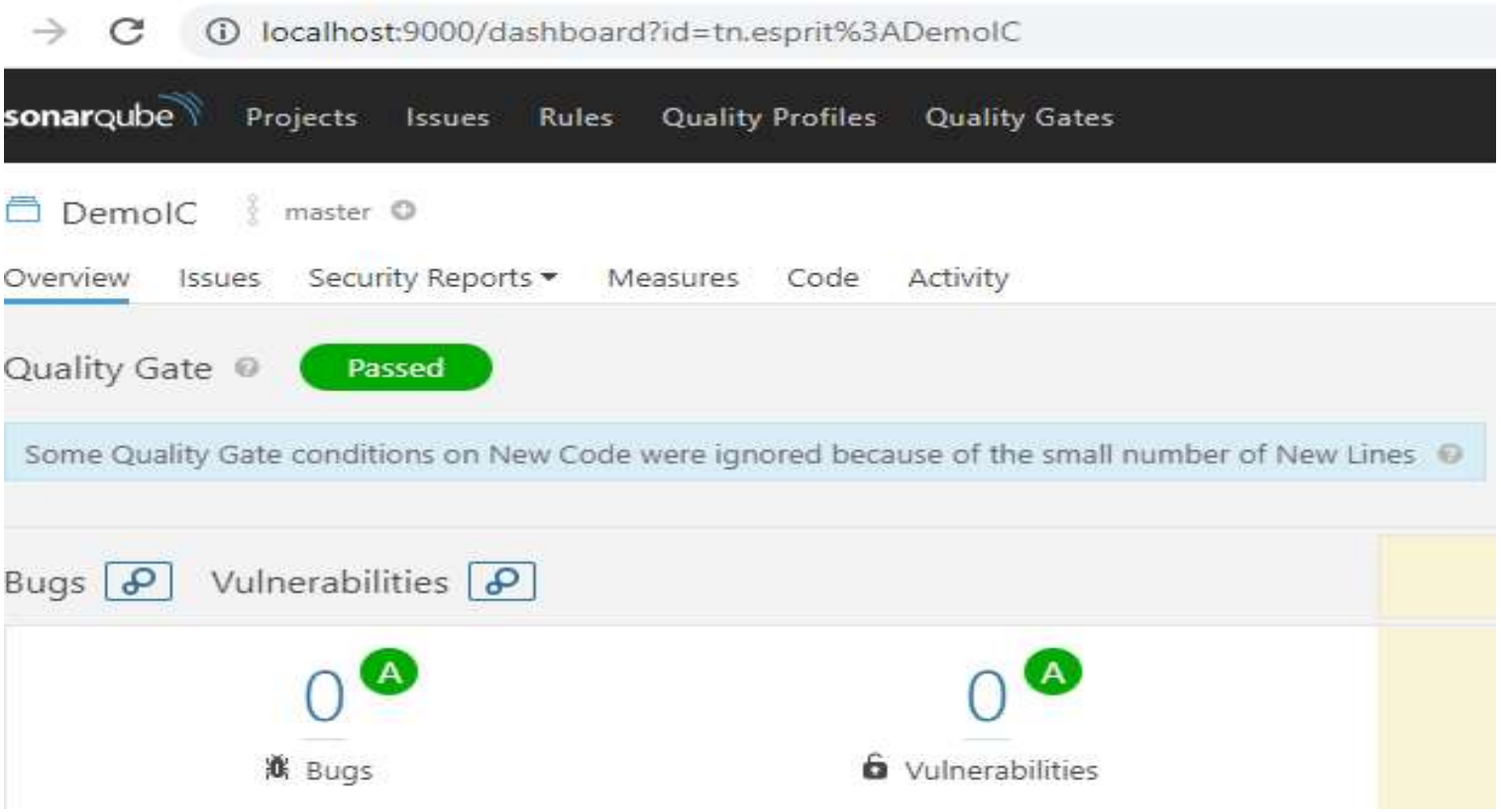
Outils


localhost:8080
Jenkins
Jenkins
Nouveau Item
Utilisateurs
Historique des constructions
Administrer Jenkins
Mes vues
Ressources Verrouillables
New View
File d'attente des constructions
File d'attente des constructions vide
État du lanceur de compilations
1 Au repos
2 Au repos
Ma page d'accueil Jenkins
Tous
+
S
M
Nom du projet
DockerPipeline
HelloWorld
Java
JavaMaven
Pipeline
Icône: S M L

Outils


Nexus Repository Manager OSS
Sonatype™
Artifact Search
Advanced Search
Views/Repositories
Repositories
Help
Welcome
Search
Reposito
Refresh
User Managed Repositories
Repository
Type
IQ Policy Violations
Public Repositories
group
3rd party
hosted
Apache Snapshots
proxy
Central
proxy
Central M1 shadow
virtual
Releases
hosted
Snapshots
Browse Index
Browse Storage
Refresh
Path Lookup:
Snapshots
com
mycompany
app
my-app
1.0-SNAPSHOT
maven-metadata.xml
maven-metadata.xml.md5
maven-metadata.xml.sha1
my-app-1.0-20200815.092421-1.jar
my-app-1.0-20200815.092421-1.jar.md5
my-app-1.0-20200815.092421-1.jar.sha1
my-app-1.0-20200815.092421-1.pom

Outils


localhost:9000/dashboard?id=tn.esprit%3ADemoIC
sonarqube
Projects
Issues
Rules
Quality Profiles
Quality Gates
DemoIC
master
Overview
Issues
Security Reports
Measures
Code
Activity
Quality Gate
Passed
Some Quality Gate conditions on New Code were ignored because of the small number of New Lines
Bugs
Vulnerabilities
0
A
Bugs
0
A
Vulnerabilities

Les avantages de l'IC

- **Réduction** du temps de correction des bogues
- **Réduction** des risques
- **Meilleure** visibilité du projet
- **Amélioration** du travail collaboratif
- **Déployer** le logiciel n'importe quand, n'importe où
- **Gain de temps** au niveau du déploiement et du développement
- **La qualité du code** est améliorée

Les inconvénients de l'IC

- Méthode de mise en place: nouveau projet/projet en cours
- Détection du besoin d'intégration: **fréquence de builds ?**
- L'intégration continue vue comme **une perte de temps:** moins de temps sur l'intégration, donc plus de temps sur le développement
- La peur de l'intégration continue
- Le piège de la compilation continue

Conclusion

- Bonne pratique mais demande de l'implication
- Mettre en place sous forme de jeu
- Apporte la qualité, la vitesse et le moindre coût

Installation des Outils

- Les outils suivants vont nous permettre de développer des applications avec Spring et Angular (vous pouvez utiliser d'autres versions) :
  - **JDK 8**
  - **Intellij**
  - **Maven 3.5.0**
  - **MySQL 5.6.17 (WAMP 3.1)**
  - **Visual Studio Code.**

Installation STS (**S**pring **T**ool **S**uite)

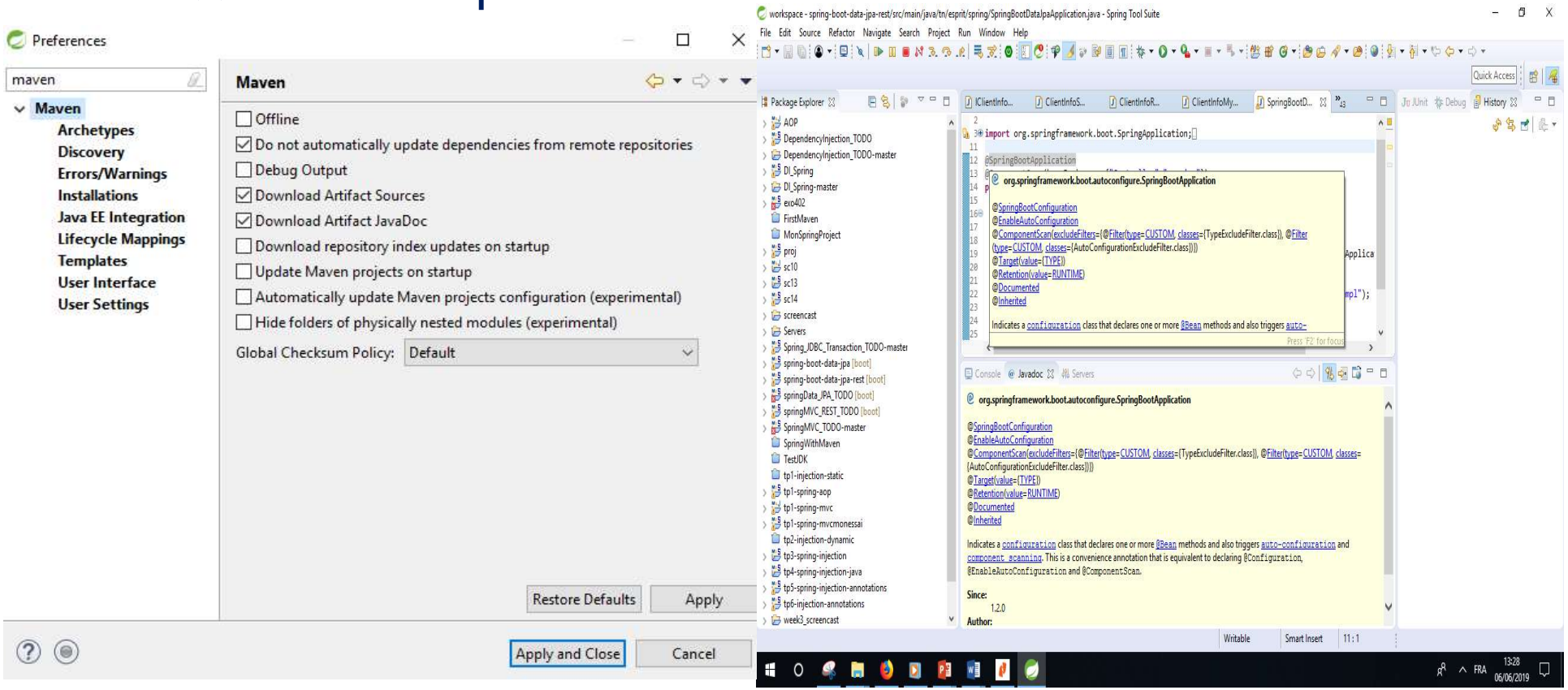
- Il suffit de dézipper le fichier **spring-tool-suite-3.8.4.RELEASE-e4.6.3-win32-x86_64.zip**
- Si vous n'avez pas le fichier, vous pouvez le télécharger à partir :
- Téléchargement STS 3.8.4 : http://spring.io/tools/sts/all

Spring Tool Suite™ est un eclipse personnalisé pour le développement d'application Spring

Fonctionnalités : https://spring.io/tools/sts

Sources et JavaDoc

- Pour visualiser les sources et JavaDoc des différents jars utilisés:
  - Windows → preferences



Prochain Cours

- **Télécharger tous les outils à partir du Drive**
- **VirtualBox & Vagrant**
  https://drive.google.com/drive/folders/1j6DHSrvohknqM0Rhs9PGjm2GJwp0V_zm

- **MySQL, JDK, GIT**
  https://drive.google.com/drive/u/1/folders/1SCURKPL7yyWnlDqsS-dZxCJM2llsD89B

Références

- Cours 01: Intégration continue, Hayri ACAR
- Continuous Integration: Improving Software Quality and Reducing Risk, Paul M. Duvall
- http://skalp.developpez.com/traductions/martin-fowler-integration-continue//
- http://www.slideshare.net/xwarzee/Principes-de-lintgration-continue
- http://blog.pascal-martin.fr/post/integration-continue-jenkins-installation-configuration
- http://www.slideshare.net/JM.Pascal/nuxeo5-installation-integration-continue
- DevOps, workshop Wevioo

Exemple Jenkins Pipeline


```
pipeline {
    agent any
    stages{
       stage('clone and clean repo'){
            steps {
                 bat "git clone https://gitlab.com/ThouravaLouati/demoic"
                  bat "mvn clean -f DemoIC"
                 }
       }
       stage('Test'){
             steps{ bat "mvn test -f DemoIC"
                  }}
                 stage('Deploy'){
                 steps  {
                        bat "mvn package -f DemoIC"
                        bat "mvn deploy -f DemoIC"
                        bat "mvn sonar:sonar -f DemoIC"

                        }
}
}
}
```

INTRODUCTION DEVOPS

2022-2023

ESPRIT thouraya.louati@esprit.tn UP ASI (Bureau E204)