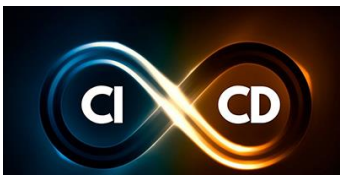
CI/CD, mise en production en continu... la solution ?

5 Avril 2024

claude@lemarson.com
https://www.lemarson.com

SOMMAIRE

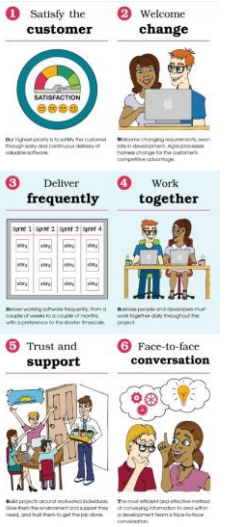
Intégration, distribution et déploiement continus



Le marché mondial du CI/CD est évalué à 4,52 milliards $ pour 2024
(The Business Research Company). Il devrait atteindre 9,33 milliards $ en 2033.
Ce n'est pas un raz de marée, qui témoigne de la difficulté du sujet.

Les 12 principes des méthodes agiles

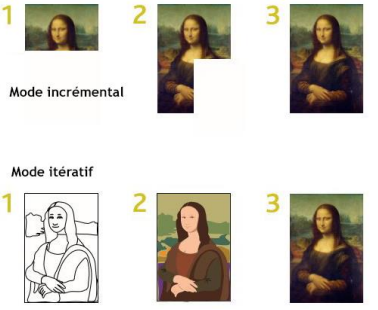
01 - La priorité est de satisfaire le client en lui livrant rapidement et régulièrement des fonctionnalités à forte valeur ajoutée.
02 - Le changement est accepté, même tardivement, car les processus agiles exploitent le changement comme un avantage compétitif pour le client.
03 - Livraisons le plus fréquemment possible, de fonctions qui « marchent » : toutes les deux semaines et au plus tard tous les deux mois
04 - Le métier et les développeurs doivent collaborer régulièrement, de préférence quotidiennement et si possible se partager les mêmes bureaux.
05 - Le projet doit impliquer des personnes motivées. Il faut leur donner l'environnement et le soutien dont elles ont besoin et leur faire confiance quant au respect des objectifs.
06 - La méthode la plus efficace de transmettre l'information est une conversation en face à face, plutôt qu'un travail à distance ou par courriel.
07 - L'unité de mesure de la progression du projet est la fonction livrée, à l'exclusion des fonctions non achevées.
08 - Les processus agiles promeuvent un rythme de développement soutenu…mais soutenable, afin d'éviter la non qualité découlant de la fatigue.
09 - Les processus agiles recommandent une attention continue à l'excellence technique et à la qualité de la conception.
10 - La simplicité et l'art de minimiser les tâches parasites, sont appliqués comme des principes essentiels. Ne produire que ce qui est nécessaire.
11 - Les équipes s'auto-organisent afin de faire émerger les meilleures architectures, spécifications et conceptions.
12 - À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son processus de travail en conséquence.

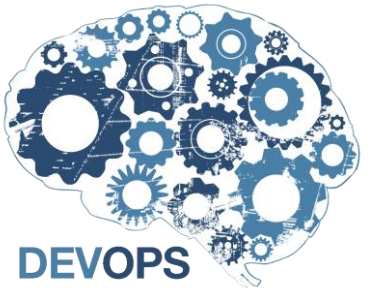


Pas de méthode agile…

Sans mise en production optimisée

- Les méthodes agiles sont fondées sur le mode incrémental : les éléments d'un même projet sont réalisés indépendamment les uns des autres et chaque incrément, testé complètement est mis en production en fin de développement, mais l'agilité peut aussi s'appliquer sur un développement itératif.
- Un incrément est un cycle court ou très court durant lequel une partie de l'application doit être développée. Elle aboutit à la fourniture d'un livrable applicatif qui se suffit normalement à lui-même.
- Le mode itératif par opposition correspond au développement d'un tout, amélioré à chaque itération.





On n'a rien inventé
(ou presque…)

Les pipelines actuels sont les héritiers directs des JCL IBM (entre autres)

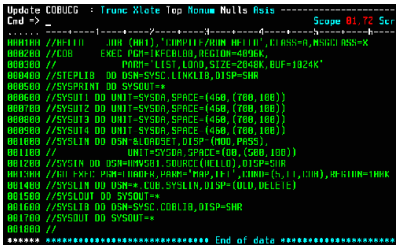
```
//JOB1    JOB  (034D),' XXXX ',CLASS='A',PRTY=6
//STEP01  EXEC PGM=COBPROG.
//INFILE  DD   DSN=SED.GLOB.DES.INFILE,DISP=SHR
//OUTFILE DD   DSN=SED.GLOB.DES.OUTFILE,
//            DISP=(NEW,CATLG,DELETE),
//            UNIT=DISK,
//            SPACE=(CYL,(1,5),RLSE),
//            DCB=(RECFM=FB,LERECL=70,BLKSIZE=700)
```

Lancement du Job

Nom du programme Cobol à exécuter

Nom du fichier Input utilisé dans SELECT INPUT-FILE ASSIGN TO INFILE.

Nom du fichier Output utilisé dans SELECT OUTPUT-FILE ASSIGN TO OUTFILE.



```
//MYJOB  JOB  (3434E),'   XXXX XXXX   ',CLASS=A,MSGCLASS=1,MSGLEVEL=(1,1),PRTY=6,TYPRUN=SCAN
```

Nom du job 1 à 8 car

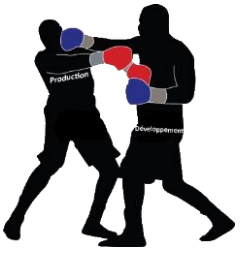
Information comptable

Nom du programmeur 1 à 20 car

Paramètres

Paramètres

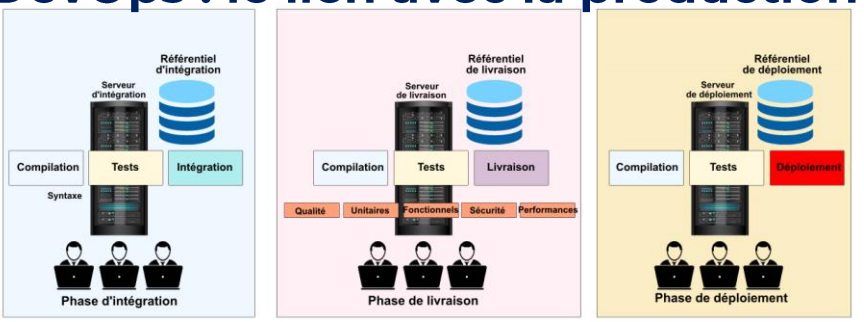
CI/CD inséparable de DevOps



Patrick Debois, à l'origine du concept…maintenant utilisé par Nokia, Google, American Express, Linkedin, Amazon

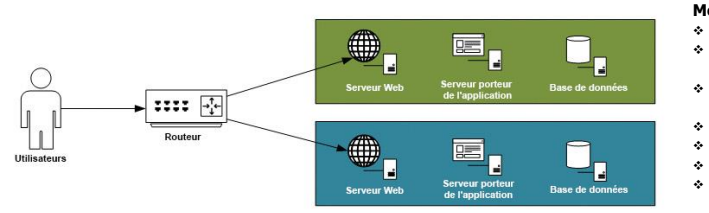
- Les mises en production rapprochées des méthodes agiles induisent d'automatiser le cycle opérationnel… Tout le monde est d'accord.
- Dans la pratique ce n'est pas évident :
  - DevOps n'est pas un succès planétaire et de nombreuses entreprises l'abandonnent.
  - Les équipes de développement et de production sont différentes et difficiles à rapprocher.
  - Le management a trop longtemps négligé la production… qui pouvait être sous-traitée.
  - Toutes les applications ne se prêtent pas au mode CI/CD.
  - On va parfois trop loin dans l'automatisation. Elle peut être une bombe à retardement.

DevOps : le lien avec la production



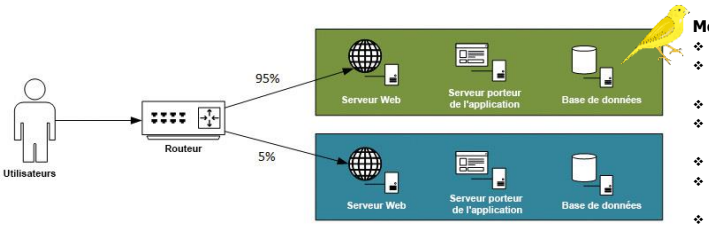
- Intégration continue :
  - Compilation, tests, mise à jour du référentiel sur une plate-forme d'intégration. Exécutée très souvent pour détecter les bugs rapidement. La mise à jour sur la plate-forme d'intégration est simple et accessible par les études.
- Livraison continue :
  - Compilation, tests et livraison de l'application à chaque étape de son cycle de vie (recette, pré-production, production). Il s'agit de valider les besoins métiers (fonctionnels). Le passage d'un état à l'autre est entièrement automatisé. Situation de "staging".
- Déploiement continu :
  - Compilation, tests et déploiement d'applications en production. Le « Continuous Deployment » nécessite que les processus d'intégration continue et de livraison continue aient été réalisés avec succès. Le déploiement est réalisé par un simple « press button ». En cas de problème, un processus automatisé de retour arrière peut être exécuté.
- Il n'y a pas qu'une seule méthode de mise en œuvre CI/CD.
- L'objectif est d'automatiser au maximum… en conservant le contrôle des opérations.
- Pour des applications sensibles, il faut être très prudent et le recours à des méthodes traditionnelles n'est pas une honte absolue…
- On n'est pas là pour réaliser des prouesses techniques, mais pour faciliter le travail des usagers.
- On évitera de faire trop de prosélytisme… avant d'être sûr que tout fonctionne correctement.

Les formes classiques de déploiement



Mode Bleu – Vert

- Switch entre 2 plates-formes de production, dites Bleu et Verte
- La nouvelle version est installée et testée sur la 2 ème plate-forme. Les usagers ne sont pas concernés.
- Quand les tests sont passés, les routeurs sont reconfigurés et les usagers sont routés vers la nouvelle version.
- En cas de problème, on revient à la dernière version stable.
- Problème : il faut maintenir 2 plates-formes, ce qui est coûteux.
- Pas adapté aux "petites" mises à jour.
- Pas toujours simple de garantir l'égalité système des plates-formes.

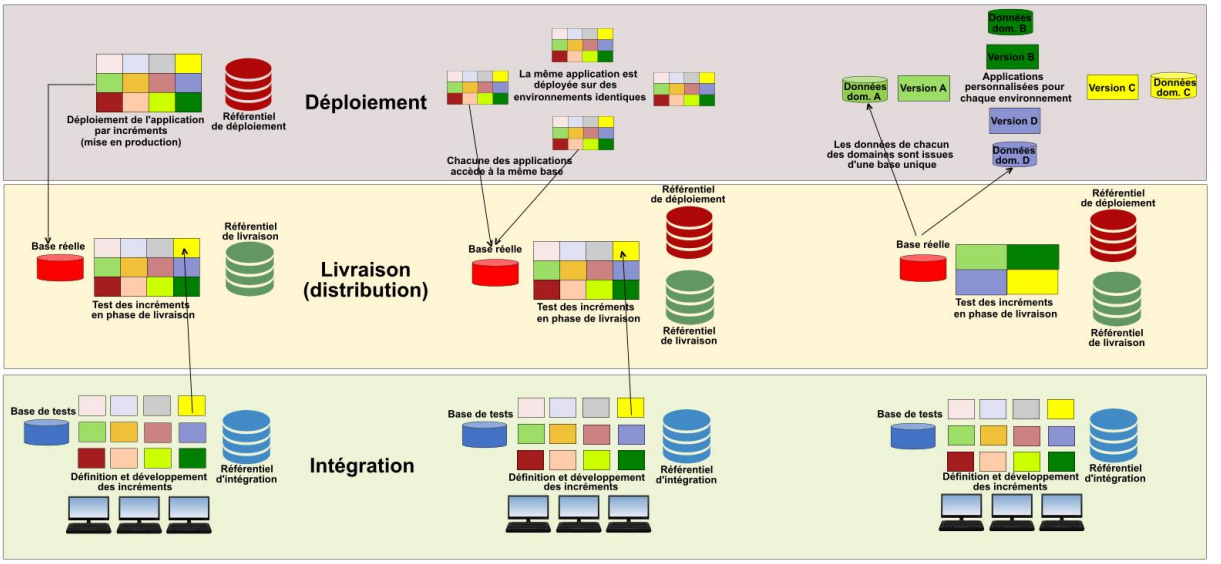


Mode Canari

- Induit 2 plates-formes.
- Quand une application est développée, elle est testée sur un sous-ensemble d'usagers (5 ou 10 %)
- Les autres restent sur la précédente version.
- Quand la nouvelle version est validée, la permière infrastructure est décommissionnée.
- Il faut choisir les usagers…
- On ne gagne rien en infra, on ne peut pas basculer en permanence entre 1 et 2 plates-formes.
- Les performances ne sont pas validées.

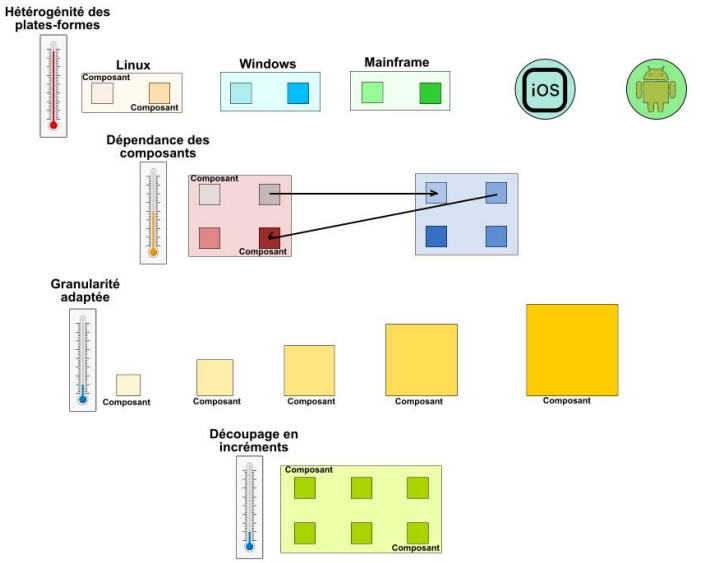
- On peut se limiter aux 2 premières phases : intégration et livraison et effectuer le déploiement manuellement.

Les différents contextes CI/CD



Les critères de sensibilité

Dépendances, hétérogénéité, granularité, découpage)



12 règles pratiques

Règle 1 : Avant toute chose, il faut se poser la question de la **faisabilité du projet**, en tenant compte de nos qualités et faiblesses, tant techniques qu'organisationnelles. Le mode SWOT est très adapté à cette réflexion.

Règle 3 : Retour arrière ("rollback"). Quelle que soit l'architecture cible, il est indispensable de pouvoir revenir en arrière, au plan applicatif, de manière automatique et sans avoir à résoudre des problèmes techiques complexes. La souplesse doit pouvoir se faire dans les 2 sens.

Règle 4 : Gestion des versions. La présence d'un utilitaire de gestion des versions est indispensable, liée au foisonnement des versions, qui augmente au fur et à mesure où l'on se rapproche du temps réel. Là encore, tout doit être automatique et simple.

Règle 5 : La coordination des tiers externes actifs. Il est très rare qu'un projet CI/CD soit exécuté sans avoir recours à des ressources externes, des tiers qu'il faut savoir gérer et intégrer dans le processus mis en oeuvre.

Règle 6 : Une cartographie à jour des ressources disponibles. Le mode CI/CD est aussi une manière d'adapter les ressources TI disponibles, aux objectifs d'efficacité. Mais encore faut-il les connaître..

Règle 7 : Le monitoring. Le mode CI/CD est un mode en constante évolution. D'autant plus difficile à gérer que l'on développe des incréments courts et que l'on travaille à "flux tendus".

Règle 12 : Les coûts vs avantages. L'objectif du processus est de faire bénéficier l'entreprise d'avantages, en termes de productivité, d'image, etc. Il faut opposer les coûts à consentir à ceux induits par les avantages. Ce calcul n'est pas simple...mais il est décisif...

Règle 11 : La bnécessité de convaincre. Le processus CI/CD appartient au modèle agile. Il faudra donc convaincre des intervenants... ce qui ne sera pas facile, compte tenu des changements profonds apportés au développement et à la mise en production.

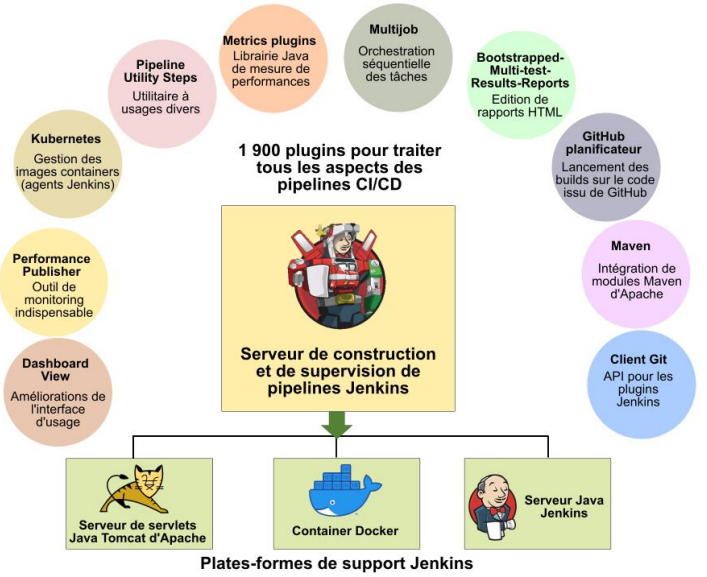
Règle 8 : L'intégrité des référentiels. Le mode CI/CD implique la gestion de 3 référentiels, pour chacune de ses étapes, consacrés aux 3 phases d'intégration, livraison et déploiement. Le processus repose donc largement sur la qualité et l'intégrité des données enregistrées.

Règle 10 : La gestion des logs. Par nature, la suite CI/CD est très évolutive et sensible aux évènements de tous ordres. En cas d'anomalies, les fichiers logs apportent une source d'informations inépuisable, que l'on complètera avec des outils d'introspection avancés.

Règle 2 : S'appuyer sur un plan de continuité et reprise d'activité, **PCA/PRA**, testé le plus souvent en vraie grandeur. Sans cette protection, le projet peut présenter des risques trop lourds.

Règle 9 : la gestion des droits. Le mode CI/CD est un processus qui met en ouvre divers intervenants, caractérisés par des "credentials" spécifiques. Ces droits et rôles seront décrits dans un annuaire LDAP, dont on s'assurera qu'il est compatible avec l'annuaire de l'entreprise.

L'exemple de Jenkins (Open Source)



- L'application Jenkins est une servlet Java qui permet d'automatiser toutes les opérations CI/CD.
- On peut la considérer comme un squelette sur lequel sont greffés des plugins (1 900 disponibles en plus de ceux que l'on peut développer spécifiquement).
- Le processus n'est pas complexe, mais nécessite une bonne expérience.
- Plutôt destiné à des environnements simples : une application, une base de données.
- Comme les autres gestionnaires CI/CD, la maintenance des opérations devient difficile avec l'augmentation de la complexité des architectures applicatives.
- Situation difficile, qui nécessiterait la présence d'un ingénieur système… malheureusement disparu !

L'exemple de Jenkins (Open Source)

- Le cœur du mécanisme Jenkins est un fichier texte Jenkinsfile, qui décrit les différentes phases du processus.
- 2 types de pipelines :
  - Steps : opérations simples à exécuter (script…)
  - Stage : regroupement de Steps.
- Le pipeline est structuré en sections , qui contiennent un ou plusieurs Directives et Steps.
- Les Directives permettent de configurer les opérations, avec les conditions, les triggers, les fichiers, la sécurité.
- A chaque Commit sur le repository central, le Commit pipeline est déclenché, avec 3 Stages :
  - Checkout : chargement du code source
  - Compile
  - Unit Test : exécution des tests unitaires et génération d'un rapport HTML.

```
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                sh 'make'
                archiveArtifacts artifacts: '**/target/*.jar', fingerprint: true
                echo 'Building…'
            }
        }
        stage('Test') {
            steps {
                /* 'make check' retourne un code non-zéro en cas de
                * problèmes ou 'true" pour que le pipeline se poursuive
                */
                sh 'make check || true'
                junit '**/target/*.xml'
                echo 'Testing…'
            }
        }
        stage('Deploy') {
            when {
                expression {
                    currentBuild.result == null || currentBuild.result == 'success'
                }
            }
            steps {
                sh 'make publish'
                echo 'Deploying…'
            }
        }
    }
}
```

La première tâche est de faire un Build. Il s'agit d'aller chercher le code là où il est assemblé, compilé ou packagé. Le Jenkinsfile ne remplace pas le générateur de Build (Maven, Gradle, GNU/Make).
archiveArtifacts récupère les fichiers de la cible **/target/*.jar, qu'il sauvegarde dans le contrôleur Jenkins.

Jenkins comporte de nombreux modules de tests. L'exemple utilise un step junit fourni par le plugin Junit.

Un script Pipeline peut comporter des tests conditionnels, des boucles, des blocs try/catch/finally et des fonctions.

Platforms

Interface utilisateur

Le "phénomène" BitBucket

- 20 millions de développeurs se servent de BitBucket.
- Service d'hébergement (dépôts), qui utilise Git pour la gestion des versions (préféré à GitHub).
- Pipeline d'intégration complet.
- Compatible Mac, iOS, Android, Linux…
- Dépôts privés ou publics.
- Suivi des bugs avec Jira, qui assure la traçabilité du code.
- Gestion de projet Trello.
- Gestionnaire Confluence pour les Wikis.
- Plutôt destiné à de petites équipes.
- Gratuit jusqu'à 5 membres.
- N'est pas conçu pour les gros projets : performances dégradées.

Ex de pipeline sur une application PHP Laravel

```
# Deploy Laravel application with bitbucket pipelines
# author : XXX
# version : N.N.N
image: php:8.1-fpm

definitions:
  caches:
    composer: $HOME/.composer/cache
  services:
    mysql:
      image: mysql:8.0
      environment:
        MYSQL_DATABASE: 'backend-pipeline'
        MYSQL_RANDOM_ROOT_PASSWORD: 'yes'
        MYSQL_USER: 'homestead'
        MYSQL_PASSWORD: 'secret'
    redis:
      image: redis

# Dependencencies Installation and Deployement to server
build: &build
  name: Install PHP dependencies and Deploy to linux server
  caches:
    - composer
  script:
    - ./bin/install.sh
    - ./bin/deploy.sh

# Testing Application
testing: &testing
  name: Testing application
  caches:
    - composer
  script:
    - ./bin/install.sh
    - ./bin/testing.sh
```

```
pipelines:
  custom:
    test-runner:
      - parallel:
        - step:
          name: PHPUnit tests
          trigger: manual
          <<: *testing
        - step:
          name: Manual deployment on test server
          deployment: test
          <<: *build
```

```
dev-deploy:
  - step:
    name: Manual deployment on development server
    deployment: development
    <<: *build

staging-deploy:
  - step:
      name: Manual deployment on uat server
      deployment: staging
      <<: *build

production-deploy:
  - step:
    name: Manual deployment on production server
    deployment: production
    <<: *build
```

composer : gestionnaire de dépendances écrit en PHP. Déclare et installe les bibliothèques dont le programme a besoin.
UAT : User Acceptance Test

Les outils CI/CD les plus utilisés

Jenkins
Serveur d'automatisation CI/CD Open Source. Fonctionne sous Linux/unix, MacOS et Windows. S'appuie sur une importante bibliothèque de plugins. Gratuit.

circleci
Mécanisme de pipeline complet. Intégré à Bitbucket, GitHub et GitHub Enterprise. Fortement personnalisable. Version gratuite limitée.

Bamboo
Couvre le spectre complet du pipeline. Supporte jusqu'à 100 agents de build. Compatible avec les repository Git, Mercurial et SVN. Parallélisation. Tarification basée sur les agents (pas les utilisateurs).

GitLab
Builds, tests et déploiements. Pour une MC, container Docker ou un autre serveur. Contrôle d'infrastructures distribuées. Bénéficie de l'environnement GitHub.

TeamCity
Outil de JetBrains. Environnement Java, intégré à Visual Studio et divers IDE. Linux et Windows et support .NET. Compatible GitLab et BitBucket. Facile à personnaliser. Version gratuite personnalisée.

Travis CI
Pipeline CI avec repository GitHub. Facile à installer, déploiement sur des clouds multiples. Services de bases de données pré-installés. MacOS, Linux et iOS. Service hébergé. Projets Open Source gratuits. Facturation pour les projets privés.

Buddy
Pipeline complet avec GitHub, BitBucket et GitLab. Utilise les containers Docker avec des langages et API pré-installés. Personnalisation simple. Services connectables à Elastic, MariaDB, Memcached, Mongo, PostGreSQL. Gratuit.

Buildbot
Framework CI basé sur Python. Open Source. Distribution et exécution sur plates-formes multiples.

semaphore
Service hébergé, intégré à GitHub. Facturation à l'usage. Support Docker intégré.

GoCD
Open Source. Nombreux plugins. Gratuit. Produit complet, facile à utiliser.

CODESHIP
Plate-forme hébergée, s'intègre à de nombreux services, outils et environnements. Simple à utiliser. Gratuit jusqu'à 100 builds/mois et illimité à partir de 49 $/mois.

Spinnaker
Pipeline multi-channel, avec Git, Jenkins, Travis CI, Docker... Services de monitoring Datadog, Prometheus, Stackdriver ou SignalFX. Open Source.

wercker
Filiale d'Oracle depuis 2017. Orienté Docker et micro-services. Intégration Git ou GitHub, BitBucket, GitLab. Réplication d'environnements SaaS en local avec Wercker CLI.

NEVERCODE
Destiné aux applications mobiles. Configurations et tests automatiques, analyse de code, tests UI, parallélisation. Publication automatique iTunes Connect, Google Play, Crashlytics, TestFairy... Conditions contractuelles adaptables.

Avantages et inconvénients du mode CI/CD

Avantages

- On colle au plus près des besoins de l'entreprise.
- Les retours utilisateurs sont plus rapides (méthode agile).
- On constate plus rapidement l'inadéquation éventuelle des ressources. matérielles, logicielles et réseaux nécessaires pour satisfaire un besoin.
- Très bonne perception des utilisateurs, qui se sentent mieux compris.

Inconvénients

- Plus on se rapproche du temps réel, plus le processus est risqué et fragile.
- Les changements permanents peuvent ne pas être appréciés.
- Dans un contexte de micro-services, l'effet domino est fréquent.
- Grande fragilité, il est nécessaire de se doter d'outils de monitoring et de reporting.
- Les coûts sont élevés et… difficiles à maîtriser.
- La rapidité d'écriture peut entraîner un manque de sécurité du code.
- On aboutit à des systèmes complexes, difficiles à maintenir.
- Une preuve de concept n'est pas significative sur le long terme.

CI/CD, mise en production en continu... la solution ?

5 Avril 2024

Nos prochains webinaires

19 avril 2024 :	**Une nouvelle composante du TI : capteurs et IoT**
3 mai 2024 :	**Le monde glaçant du "deep web" et du "dark web"**
17 mai 2024 :	**Comprendre les consensus de la Blockchain**
31 mai 2024 :	**IBN : La programmation du comportement des réseaux**
14 juin 2024 :	**L'impossible protection des données personnelles**
28 juin 2024 :	**Au cœur des technologies LLM et transformers**
13 septembre 2024 :	**Nomophobie : un pied chez les fous**
11 octobre 2024 :	**Le monde nouveau de l'argent**

claude@lemarson.com
https://www.lemarson.com