



Partie 5 : Mettre en œuvre les outils de la chaîne du DevOps

Filière : Développement digital – option web full stack

Module : Approche agile

Rédigé par : M. Haij Oussama & M. Goumih Mohamed

Révisé par: Mme Laouija Soukaina

Adapté par : Mme YOUALA Asmae



PARTIE 5

Mettre en œuvre les outils de la chaîne du DevOps

Dans ce module, vous allez :

- Introduire la chaîne DevOps
- Mettre en place la CI/CD avec Gitlab



CHAPTRE 1

Introduire la chaîne DevOps

Ce que vous allez apprendre dans ce chapitre :

- Introduction aux concepts DevOps (définition, avantages, outils)
- Lien entre l'agilité et DevOps
- Définition des notions (CI/CD)



CHAPITRE 1

Introduire la chaîne DevOps

1. Introduction aux concepts DevOps (definition, avantages, outils)
2. Lien entre l'agilité et DevOps
3. Définition des notions (CI/CD)



01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Cycles de vie d'un projet digital (Rappel)

Le cycle de vie de développement d'un système informatique comprend six phases :

Planification: Lorsque les développeurs décrivent la portée et les exigences du projet

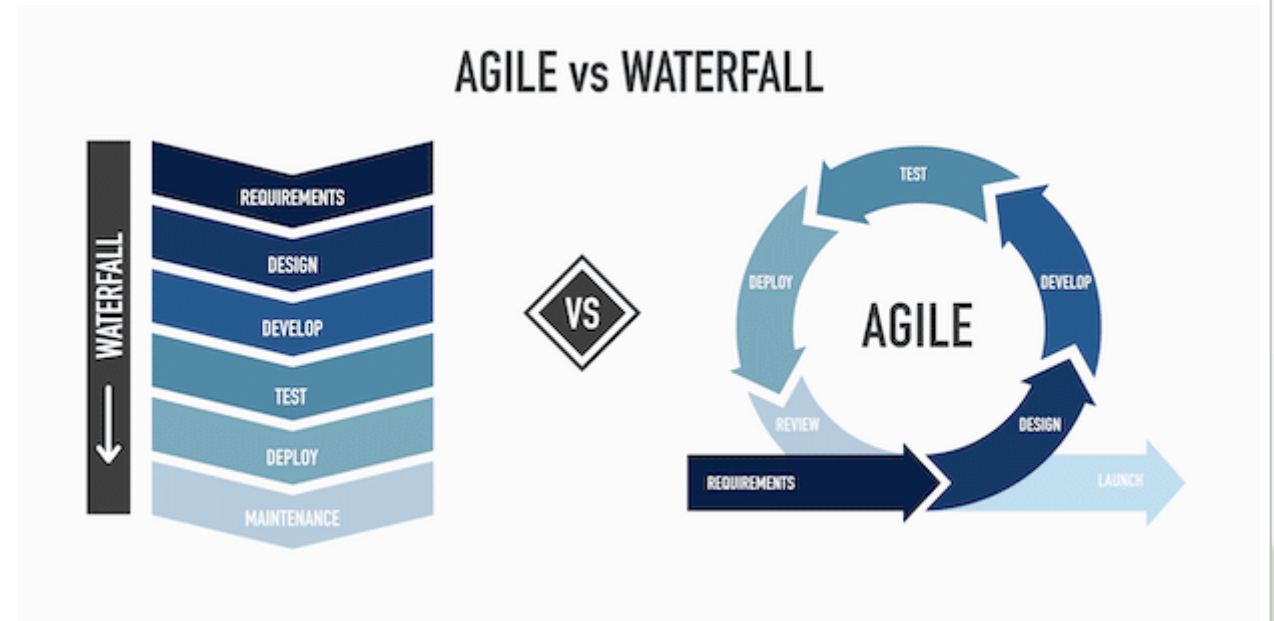
Analyse : Lorsque les développeurs recherchent et analysent les exigences pour les mettre en œuvre dans le système

Conception : Lorsque les développeurs conçoivent l'architecture du projet

Mise en œuvre: Lorsque les développeurs construisent le système

Test : Les développeurs testent le code du système et résolvent les erreurs

Déploiement et maintenance: Lorsque les développeurs mettent le système en service et effectuent la maintenance pour en assurer le bon fonctionnement



01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Pourquoi Devops?

**« Le DevOps a permis à 60 % des développeurs de publier du code 2 fois plus vite pendant la période de la pandémie »
(zdnet)**

- De nos jours, pour rester dans la course et ne pas se voir dépasser par un concurrent, une entreprise doit s'adapter, voire se transformer, continuellement.
- Cela implique de faire évoluer en permanence ses applications web, internet, intranet/extranet et eCommerce.
- L'objectif est évident: s'adapter aux nouveaux usages et aux nouvelles tendances qui émergent chaque mois.

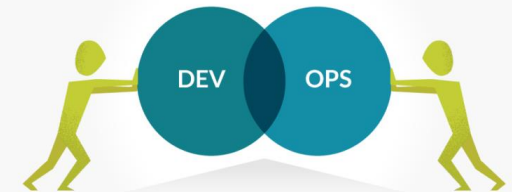
➔ On doit AUTOMATISER au maximum la chaîne de production pour pouvoir suivre le rythme.

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Définition

- Le **Devops** permet d'unifier le **développement** d'une application et la **production** de celle-ci.
- Ce mouvement se caractérise principalement par **l'automatisation** et le **suivi** (la supervision) **de toutes les étapes** de la création d'un logiciel, depuis le développement (Dev), l'intégration, les tests, la livraison en production pour déploiement, l'exploitation et la maintenance des infrastructures (Ops).
- Le terme **DevOps** est né de l'union du «**development**» et des «**operations**» dont l'objectif est favoriser une meilleure communication entre les deux équipes.
- **DevOps** vise à créer une culture et un environnement dans lesquels la conception, les tests et la diffusion de logiciels peuvent être réalisés rapidement, fréquemment et efficacement.
- **DevOps** n'est pas seulement une méthodologie, c'est une véritable philosophie de travail.



01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Définition

Ce schéma résume le concept du DevOps sous un symbole de l'infini puisque le DevOps représente une logique qui se répète en continu avec les grandes étapes suivantes :

- Un développement constant
- Des tests en continu
- Une intégration continue
- Une mise en œuvre continue
- Un monitoring permanent

Les deux activités de Dev et d'Ops représentées sur le logo finissent par se confondre étant donné que chaque étape est liée à l'autre.



Figure 17 : La chaîne DevOps

01-Introduire la chaîne DevOps

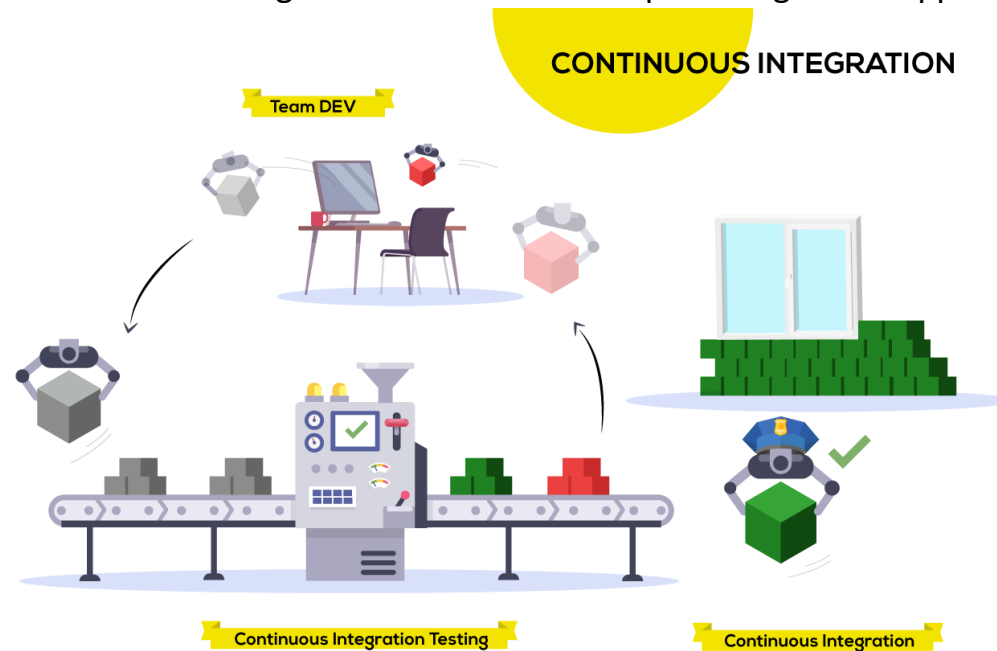
Introduction aux concepts DevOps (définition, avantages, outils)

Axes de DEVOPS

Devops se caractérise par 3 axes stratégiques permettant d'automatiser la chaîne de développement, de déploiement et de provisionning des environnements :

1. L'intégration continue « Continuous Integration » (CI)

L'équipe DevOps met en place l'automatisation d'une intégration continue de chaque changement apporté par l'équipe de développement.



Source : <https://www.aodb.com/offres/devsecfinops/audit-et-optimisation-devops>

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)



Axes de DEVOPS

1. L'intégration continue « Continuous Integration » (CI)

Dans ce cas, l'équipe de développement livre des petits paquets de modifications testés unitairement sur son propre environnement de développement : modification du code source, nouvelles fonctionnalités, changement de configuration, modification de la charte graphique.. Chaque paquet est testé de façon automatisée afin de garantir que les critères de qualité ont été atteints et qu'il n'y a pas de régression injectée par erreur: il s'agit du Continuous Integration Testing (CIT). En cas de refus d'un seul test, le livrable est rejeté par le système avec une explication claire pour le développeur. En cas de validation, le livrable est ajouté automatiquement à l'application.

Les tests peuvent être très variés et reposent sur de nombreux logiciels à mettre en œuvre :

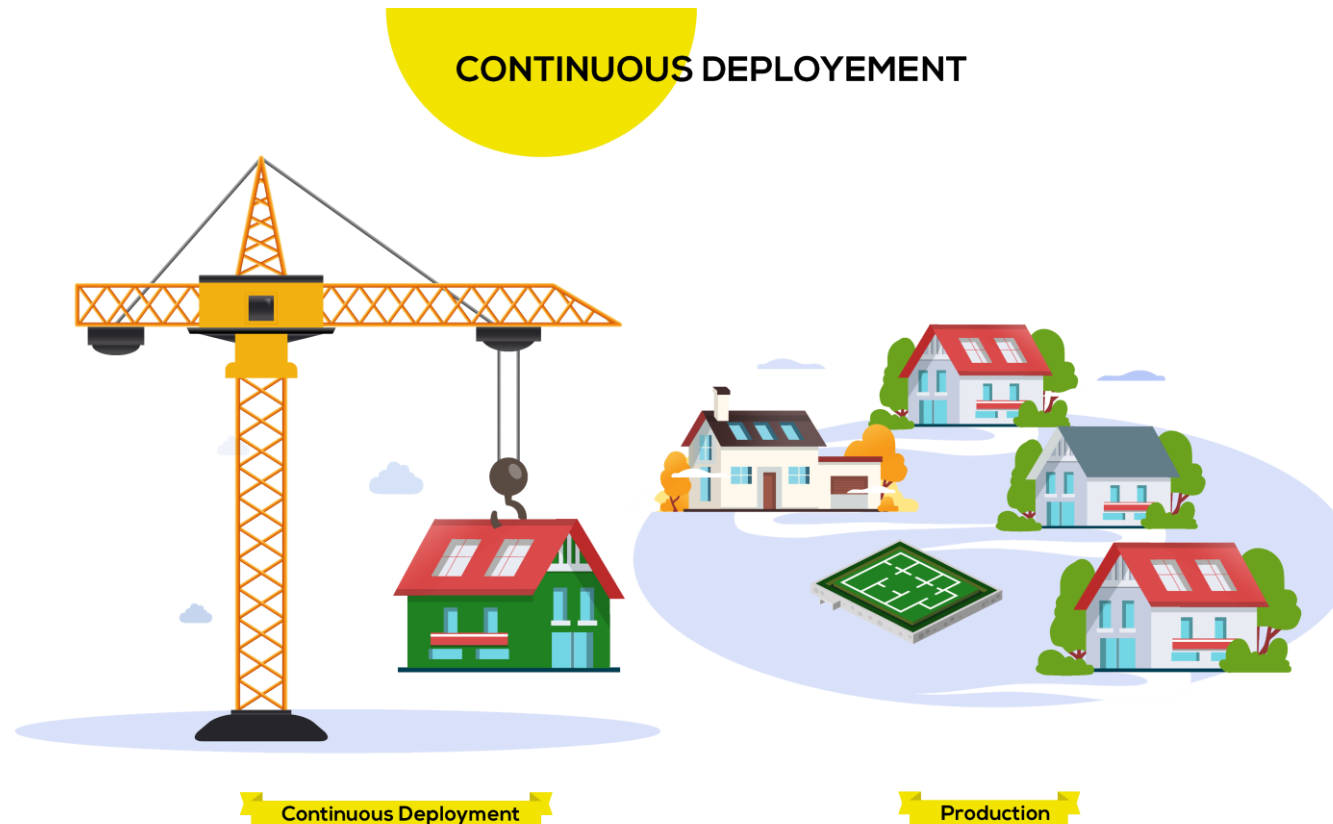
- **Test fonctionnel** : simulation de scénarios utilisateurs complets
- **Test technique** : vérifier le bon fonctionnement des API/Web Services
- **Test de qualité du code** : vérifier par exemple que le code est bien commenté
- **Test de sécurité** : vérification automatique de la vulnérabilité du code
- **Test de performance** : test de montée en charge automatique sur l'ensemble de l'application
- **Test ergonomique** : vérification de l'écart graphique avec la version précédente.

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Axes de DEVOPS

2. Le déploiement continu « Continuous Deployment » (CD)



Source : <https://www.aodb.com/offres/devsecfinops/audit-et-optimisation-devops>

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)



Axes de DEVOPS

2. Le déploiement continu « Continuous Deployment » (CD)

L'équipe DevOps met en place l'automatisation des processus de déploiement à partir de la version de l'application validée lors du processus CI.

Dans ce cas, un script de déploiement sur mesure est conçu afin de veiller à ce que l'installation du logiciel en production respecte tous les critères d'infrastructure, de sécurité et de performance.

Les déploiements automatiques doivent gérer des milliers de paramètres, c'est pourquoi il est préférable de confier l'opération à une machine afin de garantir le succès de l'opération :

- **La gestion de la sécurité** : permissions du système de fichier, des End-points, des configurations
- **La gestion du contenu** : migration automatique du contenu dans la nouvelle version
- **La gestion des backup** : la mise en place d'un backup de déploiement automatique permettant un retour arrière très rapide
- **La gestion de la performance** : l'activation des systèmes de caches, reverse proxy, CDN, etc..

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)



Etapas du cycle de vie DEVOPS

3. L'Infrastructure-as-Code (IaC)

- Basée sur le **Cloud**, la **virtualisation** et la **gestion de conteneurs**, l'objectif de l'Infrastructure as Code (IaC) est de rendre les infrastructures reproductibles simplement et de façon illimitée.
- Plus besoin d'aller installer de serveurs, de lourdes procédures d'installation logiciels et de configurations manuelles, l'approche IaC permet de scripter la création automatique des infrastructures de vos applications, permettant de créer des environnement iso-production, de versionner votre infrastructure, et bien plus encore.
- Les bénéfices sont multiples :
 - Pour les équipes de développement : ils bénéficient d'instances iso-production à la demande pour réaliser leurs travaux, ils n'ont plus de contrainte pour avancer rapidement.
 - Pour les utilisateurs qui exploitent les applications en production : La performance des applications n'est plus dégradée par le trafic.
 - L'infrastructure en ligne s'adapte automatiquement selon les besoins : allocation de puissance de calcul supplémentaire et/ou création automatique de Cluster, etc..
 - Pour la sécurité : Les changements réalisés sur l'infrastructure en dehors du IaC sont détectés et remontés. Rien n'est laissé au hasard.
 - En automatisant l'infrastructure, vous éliminez les problématiques d'approvisionnement d'environnements et de serveurs.

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Avantages

1) La collaboration

Travailler en collaboration a ses avantages. Cela motive les personnes de différents départements à s'assembler et à réfléchir à la meilleure façon d'améliorer le flux de travail opérationnel d'un produit.

2) La vitesse

- L'un des avantages inhérents du **DevOps** est qu'il accélère la fréquence et la vitesse à laquelle les entreprises peuvent introduire des nouveaux produits sur le marché.
- Cette réduction de temps est liée à ce qu'on appelle le **TTM (Time-to-market)**, **DevOps** accélère ce délais **TTM** grâce aux **tests continus** et à **l'automatisation**.

→ **Time to market (TTM)** : est le délai de mise sur le marché, c'est le temps qu'il faut entre la conception d'un produit et sa mise en production.

3) L'agilité

les pratiques **DevOps** permettent à une organisation d'être plus flexible lorsqu'il s'agit d'équilibrer sa capacité en conséquence des fluctuations de la demande . De plus, l'adoption du **DevOps** améliore la façon dont la gestion des changements est effectuée et garantit qu'elle ne ralentit pas et n'interrompt pas le processus en cour.

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)



Avantages

4) La satisfaction du client

Heureusement, l'un des principaux avantages du **DevOps** est l'amélioration continue de l'expérience client et sa satisfaction car, au bout du compte, l'objectif principal du **DevOps** est de fournir aux utilisateurs finaux des logiciels plus utiles et de meilleure qualité.

5) L'innovation

Le **DevOps** nourrit l'innovation en permettant aux équipes d'en savoir plus et de mieux comprendre les attentes des clients.


6) La sécurité

Le **DevSecOps** suit la philosophie des améliorations itératives constantes, ce qui facilite grandement le processus de gestion de la sécurité. Il accélère également la vitesse de récupération si et quand des incidents de sécurité se produisent.

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Avantages

 Pour résum , le **DevOps** permet d'am liorer la collaboration entre toutes les parties prenantes, de la planification   la livraison et   l'automatisation du processus de livraison afin de :

- Am liorer la fr quence de d ploiement
- Acc l rer la mise sur le march 
- R duire le taux d' chec des nouvelles livraisons
- Raccourcir le d lai entre les correctifs
- Am liorer le temps moyen de r cup ration
- S'adapter aux changements des besoins client avec l'agilit 
- Poss der un avantage concurrentiel
- Satisfaire les clients
- Accro tre l'innovation
- Am liorer la s curit 



01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Outils

Les équipes de **Devops** utilisent quotidiennement des outils divers pour des tâches et missions variées. Nous avons préparé ici une liste (**non exhaustive**) de ces outils.

1) Les outils de gestion de code source

- La première étape d'une collaboration **Devops** est d'aligner les équipes de développement et les ops sur un même outil de gestion de code source.
- Il y a deux types de gestions de code :
 - Les outils comme **Git** et **Subversion**, qui servent à créer un historique de ses fichiers : à tel moment, tel changement a été fait dans tes fichier. **Subversion** est un outil **plus ancien** et **moins efficace** que **Git**.
 - Les outils comme **Github**, **Gitlab** et **Bitbucket** qui servent à partager son code, et donc l'historique qui va avec. Ils sont basés sur **Git** et il est possible d'avoir l'historique du code et de travailler à plusieurs dessus. Si **Github** a le monopole historiquement, **Gitlab** devient de plus en plus populaire, notamment grâce à **Gitlab CI** qui est efficace.



01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Outils

2) Les tests d'intégration continue / déploiement continu

- **Les outils d'intégration continue et de déploiement continu**, ou **CI/CD** permettent l'automatisation des tests des modifications de code source. Concrètement, les outils de **CI/CD** permettent la modernisation des applications en réduisant le temps nécessaire pour créer de nouvelles fonctions.
- Il existe de nombreux outils de **CI/CD**. L'une des plateformes la plus utilisée est **Jenkins**, un outil open source (qui peut cependant être difficile à prendre en main).
- Il existe aussi des solutions payantes comme **GitlabCI**, Bamboo, TeamCity, Concourse, CircleCI ou Travis CI.
- Les **cloud providers**, Google et **AWS** notamment.



Jenkins



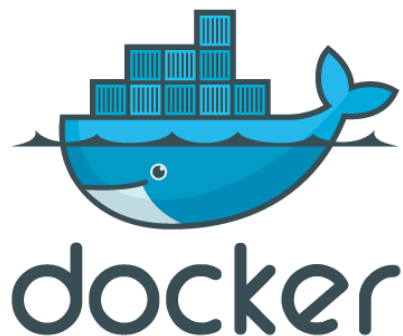
01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Outils

3) Conteneurs

- Les **conteneurs** permettent d'isoler une application avec l'ensemble des éléments dont elle a besoin pour fonctionner. L'utilisation de conteneurs permet d'être le plus "isolé" possible depuis le code des développeurs jusqu'à la production et de ne pas avoir de surprise au moment de la production.
- **Docker** automatise et standardise le déploiement d'application dans ces conteneurs virtuels et s'impose comme le leader de ce segment d'outils.
- Lorsque l'on utilise des **conteneurs**, le besoin d'un orchestrateur se fait très rapidement sentir.
- L'orchestration de conteneurs permet de simplifier leur déploiement et leur gestion. L'orchestrateur le plus utilisé sur le marché est **Kubernetes**, mais il en existe d'autres comme **MesOs** et **Docker-Swarm**.



kubernetes

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Outils

4) Cloud providers

- Les **Cloud providers** proposent aux entreprises et particuliers des solutions de stockage distant. Aujourd'hui, trois importants **players** se partagent le marché du service **cloud** : **Google Cloud Platform**, **Azure** et **AWS**. En proposant le plus large éventail de services, **AWS** est sans conteste le leader mondial de ce marché.
- Quand on parle de **Cloud providers**, on pense aux services de **load balancing**. Les services de **load balancing** ont pour mission de répartir les charges sur différents appareils, permettant une amélioration du temps de réponse.



01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Outils

5) Automatisation et gestion de configuration

- L'automatisation permet d'éliminer les tâches répétitives des équipes **Devops**.
- Plusieurs types d'automatisation en **Devops** existent :
 - Mettre en place des configurations automatiques sur les serveurs,
 - Automatiser les actions des serveurs.
- Plusieurs outils existent en fonction de l'infrastructure existante et des besoins de l'entreprise : **Terraform, Ansible, Puppet, SaltStack**



Terraform



puppet



SALTSTACK

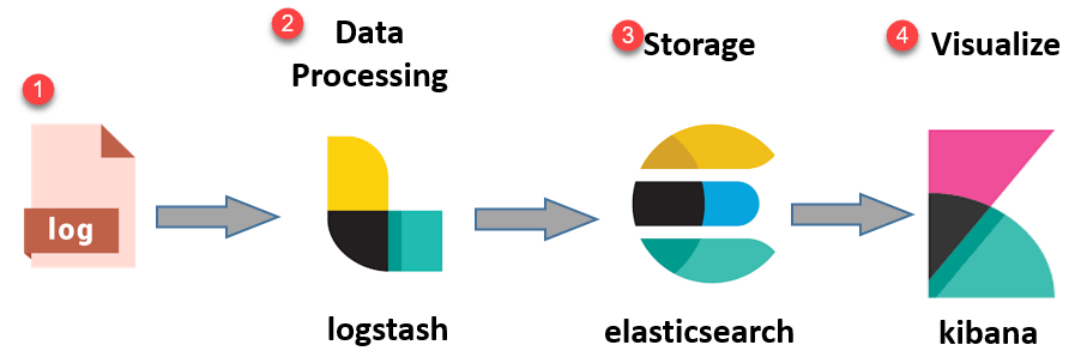
01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)

Outils

6) Monitoring et alerting

- Les outils de **monitoring** et **alerting** permettent d'avoir une vue d'ensemble sur son infrastructure, de résoudre les problèmes qui surviennent et d'améliorer les performances.
- L'application open source **Prometheus** et le service **Grafana** permettent de **monitorer** les clusters **Kubernetes**.
- En couplant trois outils, **ELK (Elasticsearch, Logstash et Kibana)** est une solution d'analyse de logs performante.



© guru99.com

01-Introduire la chaîne DevOps

Introduction aux concepts DevOps (définition, avantages, outils)



Outils

7) Outils de gestion de projet

- Pour mener à bien le développement d'un logiciel, miser sur un outil de management de projet commun dans l'équipe **Devops** paraît indispensable.
- **Jira** est un outil de gestion de projet Agile qui permet de planifier, suivre et gérer les projets de développement logiciel. **Avec Jira**, chaque membre de l'équipe de développement peut suivre l'avancée des projets et définir les priorités du sprint.
- D'un autre côté, **Trello** se démarque par son intuitivité et sa simplicité pour gérer les différentes tâches du projet.



CHAPITRE 1

Introduire la chaîne DevOps

1. Introduction aux concepts DevOps (definition, avantages, outils)
- 2. Lien entre l'agilité et DevOps**
3. Définition des notions (CI/CD)

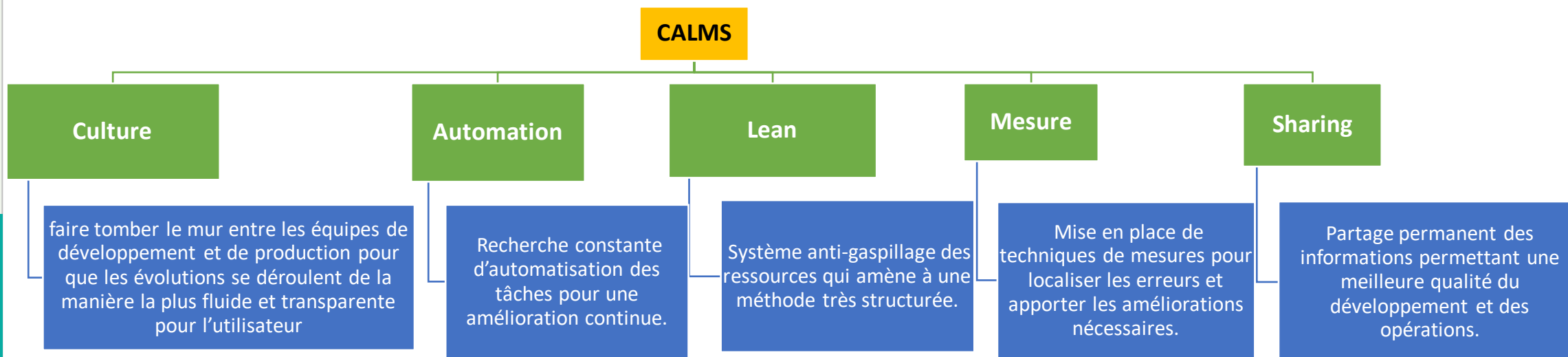


01-Introduire la chaîne DevOps

Lien entre l'agilité et DevOps

La philosophie DevOps

- Née aux alentours de 2008, la démarche **DevOps** prend sa source directement de **l'agilité**. Elle fait suite à de nouveaux besoins d'organisation de projets. Si l'agilité avait permis aux développeurs de livrer les applications plus rapidement, les opérateurs, eux, avaient conservé un fonctionnement plus classique. Le **DevOps** fut donc créé pour réduire cet écart qui faisait tant défaut à **l'agilité**.
- Les piliers de la structure **DevOps** peuvent se résumer sous un simple acronyme : **CALMS**. Ce cadre de référence est la base de la méthode, car sans lui, le **DevOps** serait voué à l'échec :



01-Introduire la chaîne DevOps

Lien entre l'agilité et DevOps

- **Agile** s'applique à mettre en place un prototype fonctionnel qui évolue en fonction des besoins du client.
- Le processus de développement est géré par un Scrum master ou un chef de projet qui le décompose en étapes ou « **sprints** ».
- Les principes agiles sont axés sur le dialogue et la collaboration. Ils comblent l'écart entre les développeurs et les utilisateurs finaux.
- L'avantage de cette méthodologie agile, c'est sa souplesse. On collabore, on discute et on corrige. Cette méthode s'attache beaucoup à la gestion globale du projet, ce qui la rend adaptable à différents corps de métiers.



01-Introduire la chaîne DevOps

Lien entre l'agilité et DevOps



Tableau de comparaison : DevOps vs méthodes Agiles

	DevOps	Développement agile
Objectifs	<ul style="list-style-type: none">• Gestion intégrale des processus d'ingénierie informatique• Rapprochement entre les développeurs et les opérateurs	<ul style="list-style-type: none">• Gestion des projets complexes• Rapprochement entre le client et les développeurs
Focus	<ul style="list-style-type: none">• Déploiement fiable et sécurisé• Processus de livraison accéléré	<ul style="list-style-type: none">• Dialogue permanent• Adaptation face au changement• Collaboration avec le client
Tâches	<ul style="list-style-type: none">• Intégration continue (CI)• Tests continus (CT)• Livraison continue (CD)	<ul style="list-style-type: none">• Développement logiciel exclusif
Rétroaction	<ul style="list-style-type: none">• Avec l'équipe interne et les gens d'affaires	<ul style="list-style-type: none">• Entre l'équipe et le client
Automatisation	<ul style="list-style-type: none">• Automatisation au cœur de la pratique	<ul style="list-style-type: none">• Peu ou pas d'automatisation

CHAPITRE 1

Introduire la chaîne DevOps

1. Introduction aux concepts DevOps (definition, avantages, outils)
2. Lien entre l'agilité et DevOps
3. **Définition des notions (CI/CD)**



01-Introduire la chaîne DevOps

Définition des notions : Intégration continue /Livraison continue /Déploiement continu



Intégration continue

- **L'intégration continue** est une méthode de développement de logiciel **DevOps** avec laquelle les développeurs intègrent régulièrement leurs modifications de code à un référentiel centralisé, suite à quoi des opérations de création et de test sont automatiquement menées.
- **Les principaux objectifs de l'intégration continue** sont de :
 - **Trouver et de corriger plus rapidement les bogues**
 - **Améliorer la qualité des logiciels**
 - **Réduire le temps nécessaire pour valider**
 - **Publier de nouvelles mises à jour de logiciels.**



01-Introduire la chaîne DevOps

Définition des notions : Intégration continue /Livraison continue /Déploiement continu



Intégration continue



Pourquoi l'intégration continue est-elle nécessaire ?

- Autrefois, les développeurs au sein d'une même équipe avaient tendance à travailler séparément pendant de longues périodes et à n'intégrer leurs modifications au référentiel centralisé qu'après avoir fini de travailler.
- Cela a rendu la fusion de changement de codes difficile et chronophage, et a également entraîné des bogues pendant une longue période, sans correction.
- La combinaison de ces différents facteurs empêchait de livrer rapidement des mises à jour aux clients.



Comment fonctionne l'intégration continue ?

- **Avec l'intégration continue**, les développeurs appliquent régulièrement leurs modifications sur un référentiel partagé (ex. Gitlab), avec un système de contrôle des versions comme **Git**.
- Avant d'envoyer leur code, les développeurs peuvent choisir d'exécuter des tests sur des unités locales pour le vérifier davantage avant son intégration.
- Un service d'intégration continue crée et exécute **automatiquement** des tests unitaires sur les nouveaux changements de codes pour détecter immédiatement n'importe quelle erreur.

01-Introduire la chaîne DevOps

Définition des notions : Intégration continue /Livraison continue /Déploiement continu

Livraison continue

- **La livraison continue** est une méthode de développement de logiciels dans le cadre de laquelle les modifications de code sont automatiquement préparées en vue de leur publication dans un environnement de production.
- Véritable pilier du développement moderne d'applications, **la livraison continue** étend le principe de l'intégration continue **en déployant tous les changements de code dans un environnement de test et/ou de production après l'étape de création.**
- Lorsque **la livraison continue** est correctement implémentée, les développeurs disposent en permanence d'un artefact de génération prêt pour le déploiement qui a été soumis avec succès à un processus de test standardisé.



01-Introduire la chaîne DevOps

Définition des notions : Intégration continue /Livraison continue /Déploiement continu

Livraison continue

- La **livraison continue** permet aux développeurs d'**automatiser** les tests au-delà des simples tests d'unité, afin de vérifier différents aspects d'une mise à jour d'application avant de la déployer auprès des clients.
- Il peut s'agir de tests d'interface, de charge, d'intégration, de fiabilité de l'API, etc. De cette manière, les développeurs peuvent vérifier de façon plus complète les mises à jour et détecter les éventuels problèmes à corriger avant le déploiement.
- La **livraison continue** automatise tout le processus de publication de logiciel. Chaque révision apportée déclenche un flux automatique qui crée, teste et planifie la mise à jour. C'est le développeur qui prend la décision finale du déploiement vers un environnement de production en ligne.



01-Introduire la chaîne DevOps

Définition des notions : Intégration continue /Livraison continue /Déploiement continu

Déploiement continu

- Pour rappel **l'intégration continue(CI)** consiste en **une phase de tests automatisés intégrée au flux de déploiement**.
- **Le déploiement continu (CD)** est donc la suite de **l'intégration continue**. Une fois que les **tests sont validés sur l'environnement de dev**, il faut **mettre en production**. Le déploiement continu consiste donc à **automatiser les actions de déploiements** qui étaient auparavant réalisées manuellement. C'est pour cette raison que l'on parle souvent de **CI/CD ensemble**, l'un va difficilement sans l'autre.

La différence entre déploiement continu et livraison continue

- Le **déploiement continu** c'est un idéal que peu d'entreprises ont réellement mis en place. La plupart du temps les équipes IT préfèrent avoir la main sur la dernière étape du déploiement. Dans ce cas on parle donc **de livraison continue**, toutes les étapes du déploiement sont automatisées sauf la dernière : **la mise en production**.
- Le déploiement continu **inclut donc la livraison continue**. Le déploiement continu va plus loin que la livraison continue en orchestrant automatiquement le déploiement des nouvelles fonctionnalités.

CHAPITRE 2

Mettre en place la CI/CD avec Gitlab

Ce que vous allez apprendre dans ce chapitre :

- Présentation de GitLab CI/CD
- Configuration du pipeline CI/CD sur gitlab



10 heures



CHAPITRE 2

Mettre en place la CI/CD avec Gitlab

1. Définition de Gitlab CI/CD
2. Définition du pipeline CI/CD: intérêt et étapes
3. Architecture du pipeline
4. Configuration du pipeline : stages ,jobs ,fichier .gtilab-ci.yml
5. Manipulation du pipeline avec Gitlab

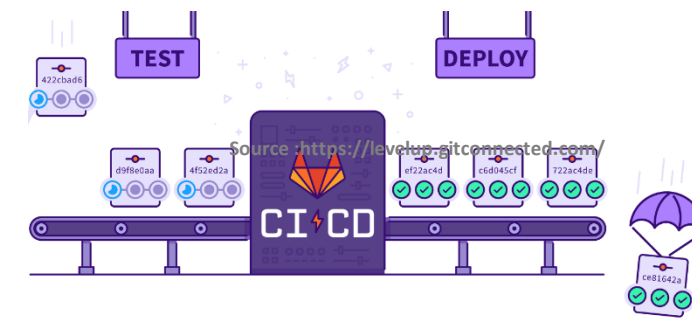


02. Mettre en place la CI/CD avec Gitlab :

Définition de Gitlab CI/CD

Définition de Gitlab CI/CD

- **GitLab CI** est un système très puissant **d'intégration continue**, intégrant de **nombreuses fonctionnalités**, et évoluant rapidement.
- **GitLab CI** va vous permettre d'automatiser les **builds**, les **tests**, les **livraisons** et les **déploiements** des applications.
- Mettre en place la **CI/CD** avec **GitLab** vous permet d'automatiser les étapes :
 - **D'intégration continue : Build** et les **Tests (unitaires, d'intégration, de non-régression...)**
 - **Déploiement continu : Déploiement** et **review (staging, production...)**



Principes de fonctionnement de Gitlab CI/CD :

Les pipelines:

- Les pipelines sont le composant de niveau supérieur de **l'intégration**, de la **livraison** et du **déploiement continue** de **Gitlab**.
- Les pipelines gèrent des **étapes (Stages)** qui contiennent des **tâches (Jobs)** qui sont exécutés sur des **runners**.

Le Runner :

- Est une application qui fonctionne avec **GitLab CI / CD** pour exécuter des **tâches (Jobs)** dans un pipeline.

02. Mettre en place la CI/CD avec Gitlab :

Définition de Gitlab CI/CD

Principes de fonctionnement de Gitlab CI/CD :

Les tâches (Jobs) :

- Une tâche est un ensemble d'instructions qu'un **runner** doit exécuter et peut produire un artefact.
- Les jobs sont un élément fondamental dans un pipeline Gitlab, il définit ce que le pipeline doit effectuer par exemple : la compilation, le test du code ...
- Chaque job a un nom et contient de script qui définit ce qui doit être fait.
- Si tous les jobs d'une étape (Stage) se terminent avec succès, le pipeline passe l'étape suivante.

Les étapes (stages) :

- Les étapes déterminent quand exécuter les tâches.
- Toutes les tâches d'une même étape sont exécutées en parallèle et l'étape suivante ne commence que lorsque toutes les tâches de l'étape précédente sont terminées sans erreurs.
- Il suffit que l'une des tâches échoue pour que l'ensemble du pipeline échoue à part quelques cas particuliers.

Les artefacts:

- Les artefacts de pipeline sont des fichiers créés par **GitLab** après la fin d'un pipeline.
- Les artefacts de pipeline sont utilisés par la **fonctionnalité de visualisation de la couverture de test** pour collecter des informations sur la couverture.

Les tags :

- Utilisez tags pour sélectionner un **Runner** spécifique dans la liste de tous les **Runners** disponibles pour le projet.

CHAPITRE 2

Mettre en place la CI/CD avec Gitlab

1. Définition de gitlab CI/CD
2. **Définition de pipeline CI/CD: intérêt et étapes**
3. Architecture de pipeline
4. Configuration de pipeline : stages ,jobs ,fichier .gitlab-ci.yml
5. Manipulation de pipeline avec gitlab

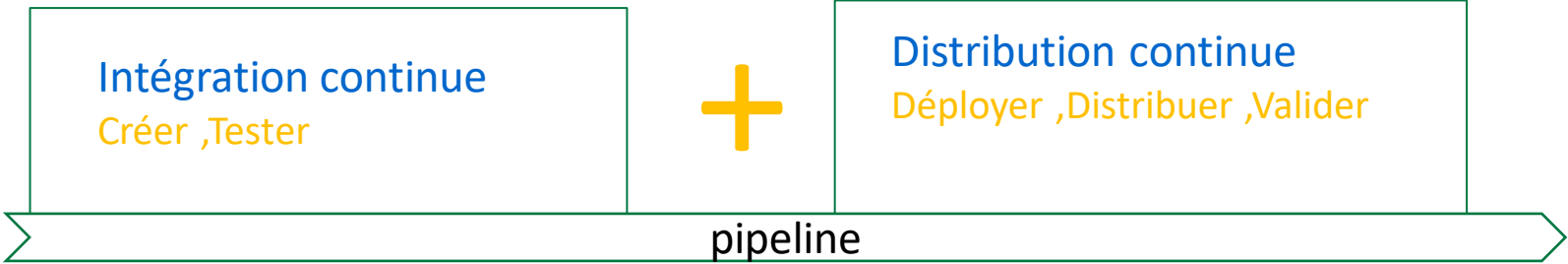


02. Mettre en place la CI/CD avec Gitlab :

Définition du pipeline CI/CD

Pipeline CI/CD : définition

- Un **pipeline CI/CD** est une série d'étapes à réaliser en vue de distribuer une nouvelle version d'un logiciel.
- Les pipelines **d'intégration et de distribution continues (CI/CD)** désignent une pratique qui consiste à améliorer la distribution de logiciels à l'aide de l'approche **DevOps**.
- Un **pipeline CI/CD utilise la surveillance et l'automatisation** pour améliorer le processus de développement des applications, en particulier lors des phases d'intégration et de tests ainsi que pendant la distribution et le déploiement.
- Le **pipeline** est généralement déclenché lors de l'envoi de code vers le dépôt.



Intérêt du pipeline

- Avoir une séquence reproductible de préparation du logiciel avant sa mise en production,
- Automatiser les étapes de livraison logicielle et ainsi gagner en temps et en fiabilité dans le déploiement,
- Eliminer les erreurs manuelles et normaliser les boucles de rétroaction des développeurs,
- Augmenter la vitesse d'itération des produits.

02. Mettre en place la CI/CD avec Gitlab :

Définition du pipeline CI/CD

Éléments d'un pipeline CI/CD (workflow du pipeline):

- Les étapes qui constituent **un pipeline CI/CD** sont des sous-ensembles distincts de tâches regroupés dans ce que nous appelons **une phase de pipeline**.
- Voici les phases de pipeline les plus courantes :
 - **Création** : compilation de l'application.
 - **Test** : test du code. L'automatisation permet ici d'épargner du temps et des efforts.
 - **Lancement (distribution)** : distribution de l'application au référentiel.
 - **Déploiement** : déploiement du code en production.
 - **Validation et conformité** : ces étapes de validation sont à adapter en fonction des besoins.

CHAPITRE 2

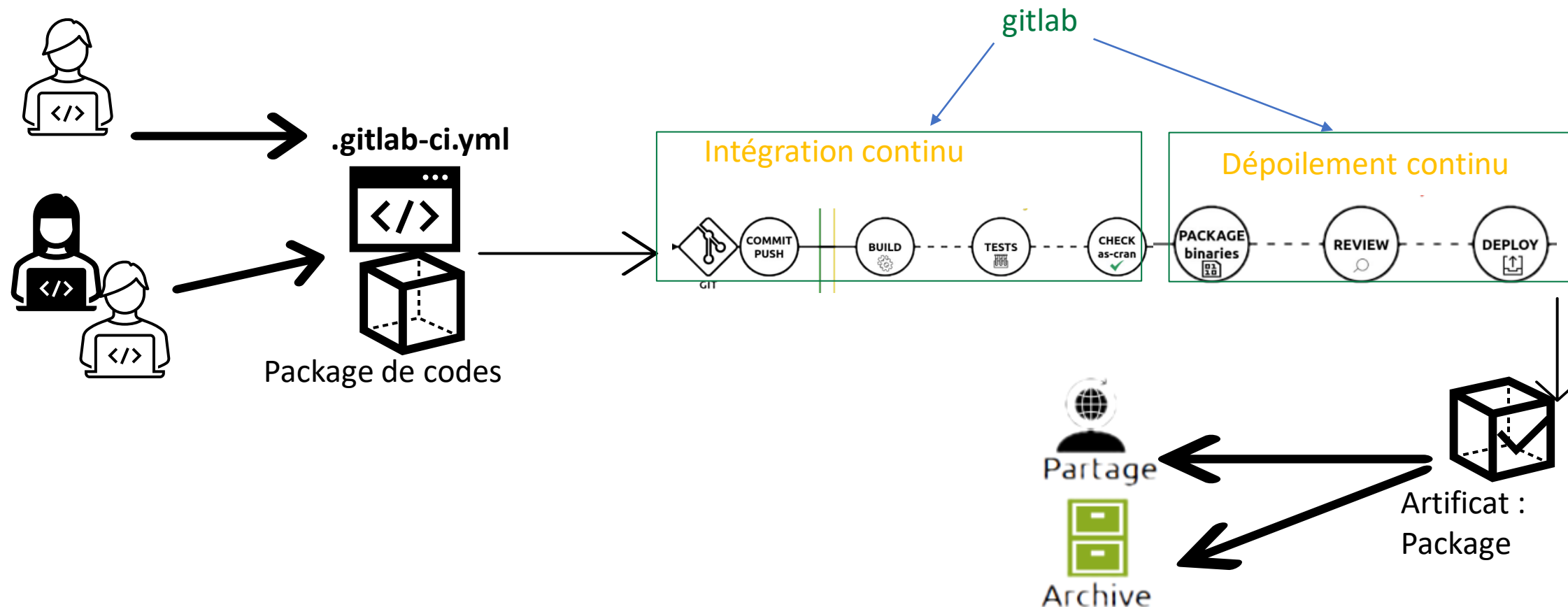
Mettre en place la CI/CD avec Gitlab

1. Définition de gitlab CI/CD
2. Définition de pipeline CI/CD: intérêt et étapes
- 3. Architecture de pipeline**
4. Configuration de pipeline : stages ,jobs ,fichier .gitlab-ci.yml
5. Manipulation de pipeline avec gitlab



02. Mettre en place la CI/CD avec Gitlab : Architecture du pipeline

Architecture basique du pipeline



CHAPITRE 2

Mettre en place la CI/CD avec Gitlab

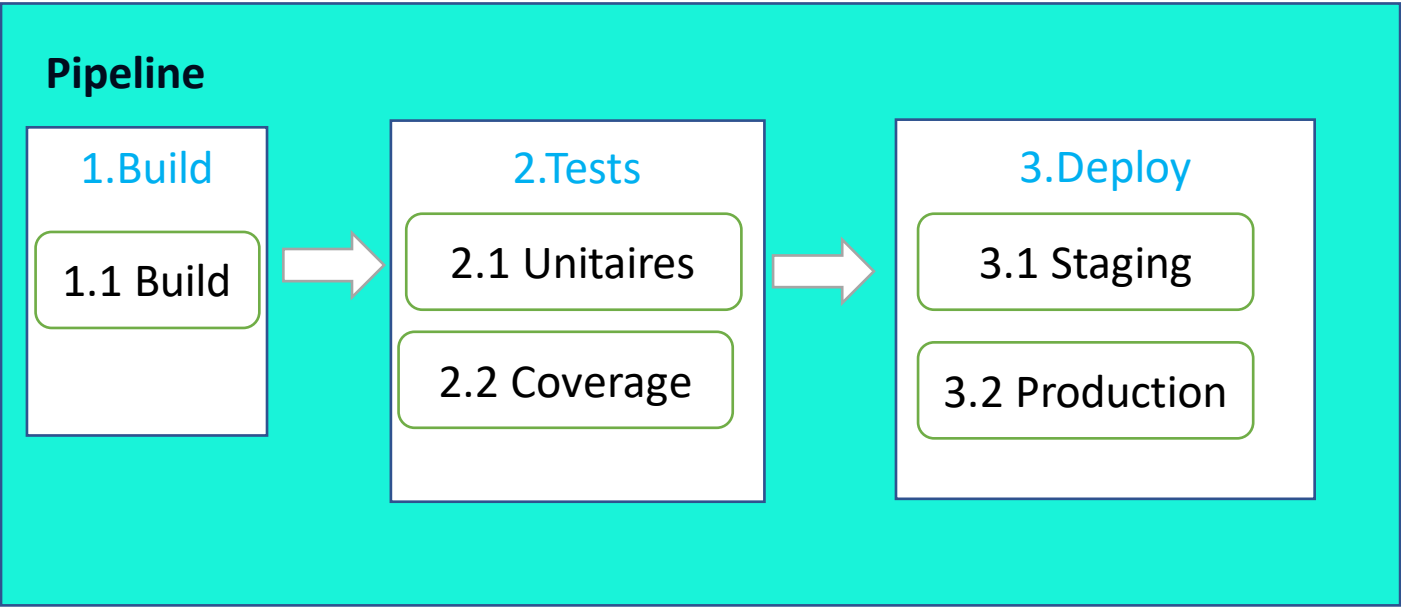
1. Définition de gitlab CI/CD
2. Définition de pipeline CI/CD: intérêt et étapes
3. Architecture de pipeline
- 4. Configuration de pipeline : stages ,jobs ,fichier .gitlab-ci.yml**
5. Manipulation de pipeline avec gitlab



02. Mettre en place la CI/CD avec Gitlab : Configuration du pipeline

Configuration pipeline: stages et jobs

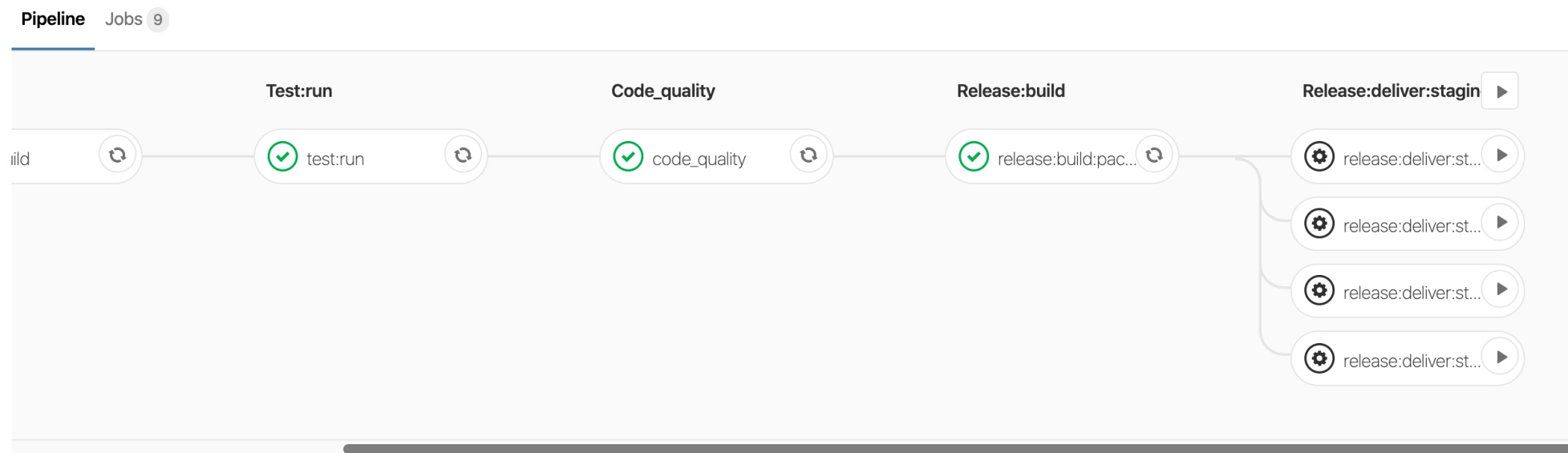
- Il faut définir les différents **stages** du pipeline CI/CD que l'on souhaite créer; En général ,un découpage en trois stages : **Build**, **Tests** et **Deploy**.
- Ces **stages** sont lancés séquentiellement et sont composées de **jobs**.
- Une étape doit contenir au moins un job, ces derniers étant exécutés en parallèle par défaut.
- Voici une structure de pipeline avec trois **stages** : **Build**, **Tests** et **Deploy**.
- Sur ce schéma, les stages **Tests** et **Deploy** possèdent chacun deux **jobs**.



02. Mettre en place la CI/CD avec Gitlab :

Configuration du pipeline

Exemple de pipeline avec gitlab



- Dans cet exemple, on observe une **étape** durant laquelle les tests sont joués, puis une seconde durant laquelle la qualité du code est analysée, etc.
- Lorsqu'une étape est **validée (coche verte)**, l'étape suivante peut alors être jouée ; si elle était rouge, les étapes suivantes n'auraient pu être jouées et notre pipeline aurait été considéré comme **en échec**
- On observe que les dernières étapes (**deliver**) n'ont pas été lancées : cela est dû au fait que ces étapes sont manuelles. Elles pourraient être activées en cliquant sur le bouton « **play** » et lancées en séquence : chaque job permettant d'envoyer vers un environnement différent.

02. Mettre en place la CI/CD avec Gitlab :

Configuration du pipeline

Configuration pipeline avec CI/CD Gitlab

- Il existe de nombreux outils pour faciliter la mise en place **du CI/CD**. Nous utilisons les outils **de CI/CD de GitLab**. Cette fonctionnalité intégrée à **GitLab** nous permet d'appliquer de la **CI/CD** sans installer d'application tierce.
- Pour appliquer cette **CI/CD dans Gitlab**, tout passe par un fichier **YAML**: **gitlab-ci.yml**. Une fois créé à la racine du projet, **Gitlab** détecte sa présence et va exécuter à chaque commit **un pipeline de CI/CD**. En fonction de sa configuration, ce pipeline va exécuter différentes instructions de **build**, de **test** et / ou de **déploiement**.
- Pour commencer, nous avons besoin de :
 - Un compte sur **GitLab.com**
 - Un repository **GitLab (projet)**
 - Déclaration d'un pipeline **CI/CD Gitlab** dans le repository :Il suffit d'ajouter un fichier manifeste portant le nom : **gitlab-ci.yml** contenant les stages et jobs et d'autres déclarations de configuration.
- **En voici un exemple:**

```
+ stages:                # List of stages for jobs, and their order of execution
+   - build
+   - test
+   - deploy
+
+ build-job:              # This job runs in the build stage, which runs first.
+   stage: build
+   script:
+     - echo "Compiling the code..."
+     - echo "Compile complete."
+
+ unit-test-job:          # This job runs in the test stage.
+   stage: test           # It only starts when the job in the build stage completes successfully.
+   script:
+     - echo "Running unit tests... This will take about 60 seconds."
+     - sleep 60
+     - echo "Code coverage is 90%"
+
+ 
```

CHAPITRE 2

Mettre en place la CI/CD avec Gitlab

1. Définition de gitlab CI/CD
2. Définition de pipeline CI/CD: intérêt et étapes
3. Architecture de pipeline
4. Configuration de pipeline : stages ,jobs ,fichier .gitlab-ci.yml
5. **Manipulation de pipeline avec gitlab**

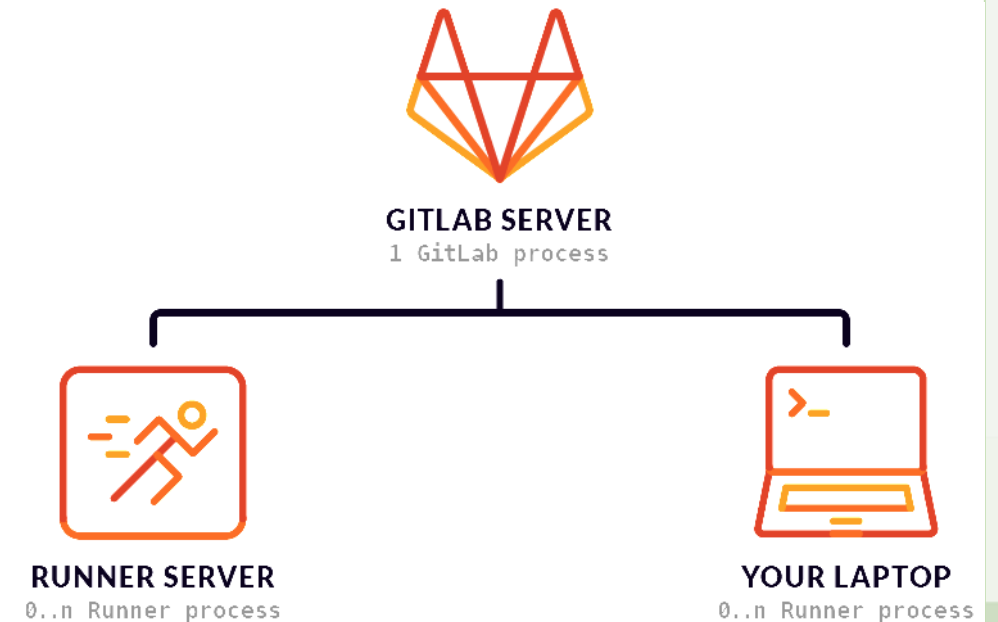


02. Mettre en place la CI/CD avec Gitlab :

Manipulation du pipeline

Runner sous Gitlab

- Afin d'exécuter les tâches figurant dans le pipeline CI/CD, Gitlab a besoin des applications appelées **Runner**;
- Gitlab Runner un composant de l'application GitLab qui permet d'effectuer les tâches d'intégration continue, de livraison continue et de déploiement continue. Par exemple: il peut lancer les tests unitaires;
- Le GitLab Runner est un programme développé en Go que vous pouvez installer sur un serveur, un portable, une VM, un container, un cluster Kube. Ce runner va rester en « écoute » à l'instance GitLab pour "découvrir" si il y a de tâche exécuter.
- On peut utiliser les Runners de Gitlab (version payante) pour exécuter les tâches du pipeline CI/CD, comme on peut les installer et les configurer manuellement dans notre infrastructure.
- Dans ce cours, on va installer Gitlab Runner sous windows, voici le lien officiel décrivant les étapes nécessaires à sa mise en place : <https://docs.gitlab.com/runner/install/windows.html>

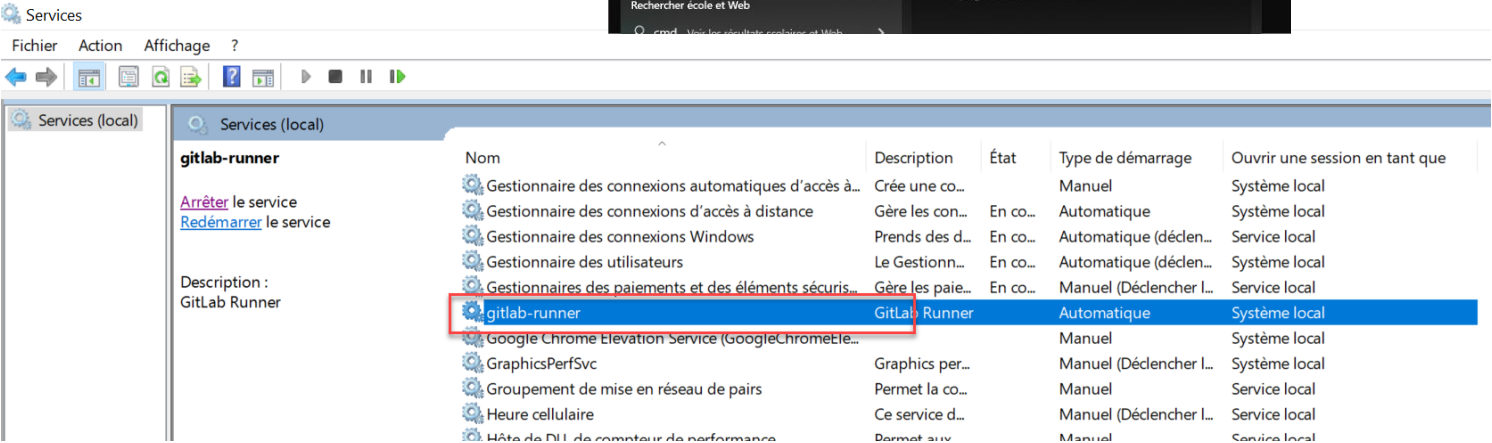
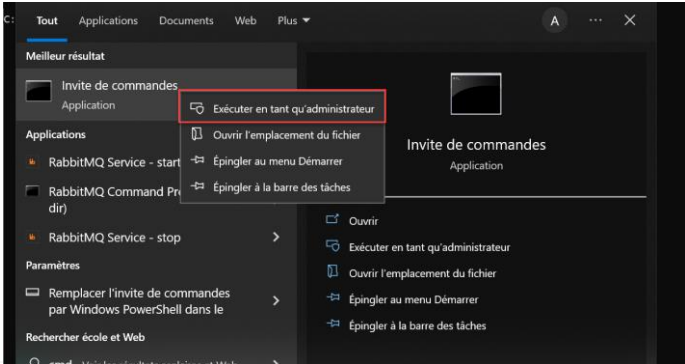


02. Mettre en place la CI/CD avec Gitlab :

Manipulation du pipeline

Installation de Gitlab Runner sous Windows

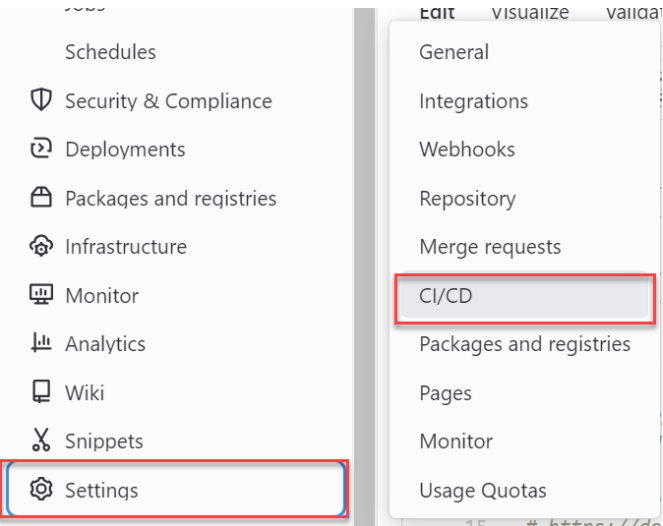
1. Télécharger Gitlab-Runner depuis ce lien : <https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-windows-amd64.exe>
2. Renommez le fichier téléchargé en: gitlab-runner.exe
3. Placez le fichier dans un répertoire de votre choix, de préférence appelé « Gitlab-Runner » . Ex.: « C:/Gitlab-Runner »
4. Démarrez l'« invite de commande windows » en tant qu'Administrateur;
5. Placez vous sous le dossier de gitlab runner: ex. : `cd C:/Gitlab-Runner`
6. Pour l'installer, tapez la commande : `.\gitlab-runner.exe install`
7. Pour le démarrer, tapez la commande : `.\gitlab-runner.exe start`



02. Mettre en place la CI/CD avec Gitlab : Manipulation du pipeline

Enregistrement de Gitlab Runner

1. Sous le projet Gitlab, Naviguez à Settings>CI/CD:
2. Désactivez les runners partagés en décochant : « Enable shared for this project »



Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. [What is GitLab Runner?](#)

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine.

How do runners pick up jobs?

Runners are either:

- **active** - Available to run jobs.
- **paused** - Not available to run jobs.

Tags control which type of jobs a runner can handle. By tagging a runner, you make sure shared runners only handle the jobs they are equipped to run. [Learn more.](#)

Specific runners

These runners are specific to this project.

Set up a specific runner for a project

1. [Install GitLab Runner and ensure it's running.](#)
2. Register the runner with this URL:

<https://gitlab.com/>

And this registration token:

[Reset registration token](#)

[Show runner installation instructions](#)

Shared runners

[These runners](#) are available to all groups and projects.

Each CI/CD job runs on a separate, isolated virtual machine.

Enable shared runners for this project



Available shared runners: 50

• #17142247 (ZQ74L_ri)
blue-2.saas-linux-medium-amd64.runners-
manager.gitlab.com/default
[saas-linux-medium-amd64](#)

• #11574045 (8cwZ3F43) 
[4-blue-shared.gitlab.org.runners-manager.gitlab.com](#)

02. Mettre en place la CI/CD avec Gitlab : Manipulation du pipeline

Enregistrement de Gitlab Runner

3. Démarrez l'« invite de commande windows » en tant qu'Administrateur;
4. Tapez la commande : `gitlab-runner.exe register`

Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab.

Register as many runners as you want. You can register runners as:

How do runners pick up jobs?

Runners are either:

- **active** - Available to run jobs.
- **paused** - Not available to run jobs.

Tags control which type of jobs a runner can handle. By tagging a runner, you can equip it to run. [Learn more](#).

Specific runners

These runners are specific to this project.

Set up a specific runner for a project

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:

`https://gitlab.com/`

And this registration token:

`GR1348941hyjwkZ5r`

Reset registration token

Show runner installation instructions

```
C:\Windows\system32>gitlab-runner.exe register
Runtime platform arch=amd64 os=windows pid=23960 revision=133d7e76 version=15.6.1
WARNING: The 'register' command has been deprecated in GitLab Runner 15.6 and will be replaced with a 'deploy' command. For more information, see https://gitlab.com/gitlab-org/gitlab/-/issues/380872
Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
GR1348941hyjwkZ5r
Enter a description for the runner:
[DESKTOP-PGQ50JJ]: dev_pc
Enter tags for the runner (comma-separated):
shell
Enter optional maintenance note for the runner:

Registering runner... succeeded runner=GR1348941hyjwkZ5r
Enter an executor: docker, docker-ssh, virtualbox, docker+machine, custom, parallels, shell, ssh, docker-ssh+machine, instance, kubernetes, docker-windows:
shell
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!

Configuration (with the authentication token) was saved in "C:\\Windows\\system32\\config.toml"
```

Dans le cas de l'executor **shell**, le runner va utiliser les ressources logicielles installées sur la machine où lui-même a été installé (ce peut-être une machine physique ou virtuelle ou même un container) pour exécuter les jobs du pipeline.

02. Mettre en place la CI/CD avec Gitlab :

Manipulation du pipeline

Enregistrement de Gitlab Runner


On peut remarquer que le runner a été bien ajouté au projet Gitlab

Specific runners


These runners are specific to this project.

Set up a specific runner for a project

1. [Install GitLab Runner and ensure it's running.](#)
2. Register the runner with this URL:

`https://gitlab.com/` 

And this registration token:

`[token]` 

[Reset registration token](#)

[Show runner installation instructions](#)

Available specific runners

 `#19381926 (abgayzVV)` 

`dev_pc`

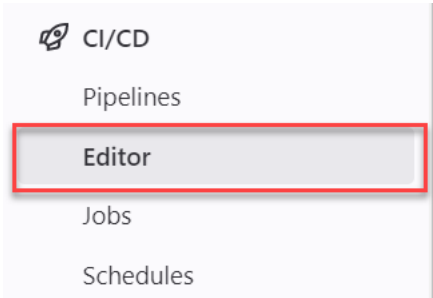
  [Remove runner](#)

02. Mettre en place la CI/CD avec Gitlab :

Manipulation du pipeline

Créer un pipeline

Sous le menu Gitlab de votre projet, naviguez vers CI/CD -> Editor



Optimize your workflow with CI/CD Pipelines

Create a new `.gitlab-ci.yml` file at the root of the repository to get started.

Configure pipeline

02. Mettre en place la CI/CD avec Gitlab :

Manipulation du pipeline

Créer un pipeline

Pour cet pipeline, on va configurer deux étapes (stages) : build et test

- **build-job** : l'exécution de cette tâche consiste à afficher deux messages
- **unit-test-job**: Cette tâche lancera phpunit pour executer les tests qui sont placés sous le dossier tests et affichera par suite le message : tests terminus.

```
stages:           # List of stages for jobs, and their order of execution
- build
- test
```

```
build-job:        # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
    - echo "Compile complete."
```

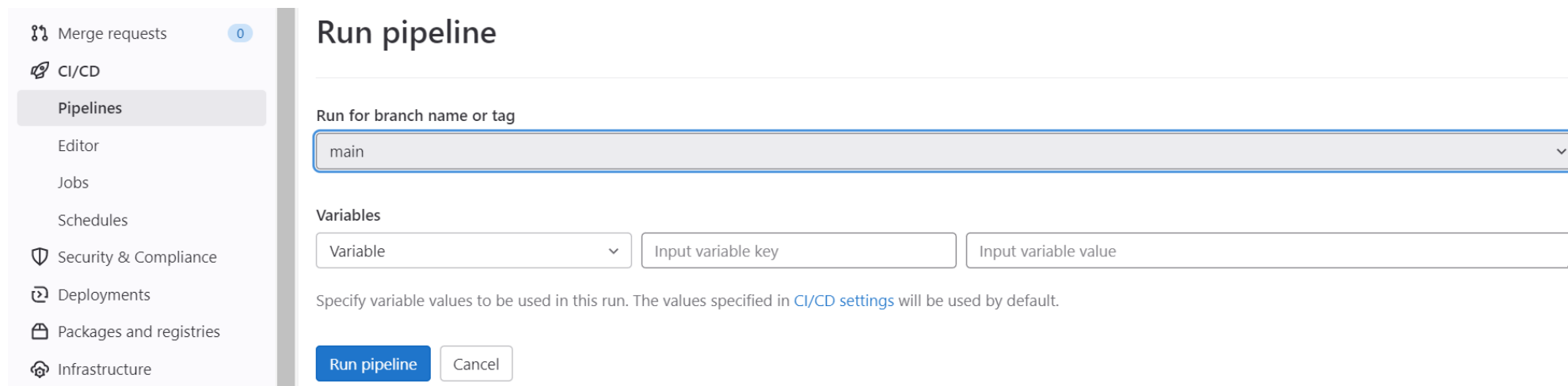
```
unit-test-job:    # This job runs in the test stage.
  stage: test      # It only starts when the job in the build stage completes successfully.
  script:
    - vendor/bin/phpunit tests
    - echo "tests terminés"
```

02. Mettre en place la CI/CD avec Gitlab :

Manipulation du pipeline

Démarrer le pipeline

- Sous l'invite de commande windows, tapez : **gitlab-runner.exe run**
- En Gitlab; Sous le menu CI/CD->Pipelines , cliquez « Run pipeline »:



The screenshot shows the GitLab web interface for running a pipeline. On the left is a sidebar with navigation links: Merge requests (0), CI/CD, Pipelines (selected), Editor, Jobs, Schedules, Security & Compliance, Deployments, Packages and registries, and Infrastructure. The main content area is titled 'Run pipeline'. It features a text input field labeled 'Run for branch name or tag' with 'main' entered. Below this is a 'Variables' section with a table header: 'Variable', 'Input variable key', and 'Input variable value'. A note states: 'Specify variable values to be used in this run. The values specified in [CI/CD settings](#) will be used by default.' At the bottom are two buttons: 'Run pipeline' (blue) and 'Cancel' (grey).

02. Mettre en place la CI/CD avec Gitlab :

Manipulation du pipeline

Etat du pipeline

On peut consulter l'état de l'exécution des tâches sous le menu **pipeline**:

- Project information
- Repository
- Issues 0
- Merge requests 0
- CI/CD
- Pipelines**
- Editor
- Jobs
- Schedules
- Security & Compliance
- Deployments
- Packages and registries
- Infrastructure
- Monitor

🕒 4 jobs for `main` in 36 seconds (queued for 2 minutes and 24 seconds)

🚩 `latest`

🔗 `32a488ac` 📄

🔗 No related merge requests found.

Pipeline Needs Jobs 4 Tests 0

build

✓ build-job

🔄

test

✓ unit-test-job

🔄

02. Mettre en place la CI/CD avec Gitlab :

Manipulation du pipeline

Etat du pipeline

On peut, également, consulter le détails d'une tâche en cliquant dessous :
- Ci-après, le détail de l'exécution de la tâche **build-job** :

```
24 Checking out 32a488ac as main...
25 git-lfs/3.2.0 (GitHub; windows amd64; go 1.18.2)
26 Skipping Git submodules setup
27 ✓ Executing "step_script" stage of the job script 00:03
28 $ echo "Compiling the code..."
29 Compiling the code...
30 $ echo "Compile complete."
31 Compile complete.
32 ✓ Cleaning up project directory and file based variables 00:02
33 Job succeeded
```

02. Mettre en place la CI/CD avec Gitlab :

Manipulation du pipeline

Etat du pipeline

- Ci-après, le détail de l'exécution de la tâche **unit-test-job** :

```
24 Checking out 32a488ac as main...
25 git-lfs/3.2.0 (GitHub; windows amd64; go 1.18.2)
26 Skipping Git submodules setup
27 ✓ Executing "step_script" stage of the job script 00:03
28 $ vendor/bin/phpunit tests
29 PHPUnit 9.5.26 by Sebastian Bergmann and contributors.
30 ..... 15 / 15 (100%)
31 Time: 00:00.024, Memory: 4.00 MB
32 OK (15 tests, 15 assertions)
33 $ echo "tests terminés"
34 tests terminés
35 ✓ Cleaning up project directory and file based variables 00:03
36 Job succeeded
```

02. Mettre en place la CI/CD avec Gitlab :

Exemple de pipeline

Exemple d'un pipeline :

```

stages:          # Liste des étapes en ordre d'exécution
- creation
- test
- code_quality

build-job:       # Cette tâche (job) appartient à l'étape « creation » et s'exécutera en premier
  stage: creation
  script:
    - echo "Démarrage ..."

unit-test-job:   # Cette tâche (job) appartient à l'étape « test », lancera les tests unitaires et générera deux fichiers d'analyses
  stage: test
  variables:
    XDEBUG_MODE: "coverage"
  artifacts:
    reports:
      junit: phpunit-execution-result.xml
    paths:
      - phpunit-execution-result.xml
      - phpunit-coverage-result.xml
  script:
    - vendor/bin/phpunit --coverage-clover=phpunit-coverage-result.xml --log-junit=phpunit-execution-result.xml

code-quality-job: # Cette tâche (job) appartient à l'étape « code_quality » et lancera le scan avec sonar-scanner
  stage: code_quality
  variables:
    GIT_CLEAN_FLAGS: none
  script:
    - E:\sonar-scanner_204\bin\sonar-scanner.bat -D"sonar.qualitygate.wait=true" -D"sonar.projectKey=demo2_pipeline" -D"sonar.sources=." -
D"sonar.host.url=http://localhost:9000" -D"sonar.login=sqp_68d2bda07c5d5bd88d33993c9dd8a8ad15d99b82"

```

Références :

- <https://www.atlassian.com/fr/software/Jira>
- <https://www.twybee.com/blog/>
- https://www.youtube.com/watch?v=P_8Zav29rJs
- Jira Agile Essentials ,Patrick Li
- Jira Quick Start Guide ,Sagar Ravi
- <http://adrienjoly.com/cours-git/tutos/conflit-de-fusion.html>
- <https://ensc.gitbook.io/>
- <https://documentation-snds.health-data-hub.fr/>
- <https://kinsta.com/>
- MÉTRIQUES ET CRITÈRES D'ÉVALUATION DE LA QUALITÉ DU CODE SOURCE D'UN LOGICIEL , Pierre Mengal
- M. Contensin – SonarQube
- <https://docs.sonarqube.org>
- <https://docs.gitlab.com/>
- <https://gitlab-ci.goffinet.org/>
- <https://docs.gitlab.com/ee/ci/examples/index.html>