CI/CD

I/ Définitions :

- **Que veut dire intégration continue**

L'intégration continue, ou CI comme on l'appelle souvent, est la pratique qui consiste à faire en sorte que toutes les personnes travaillant sur le même projet logiciel transmettent régulièrement leurs modifications à la base de code et vérifient ensuite que le code fonctionne toujours comme il le devrait après chaque modification. L'intégration continue est un élément clé de l'approche DevOps en matière de création et de publication de logiciels, qui favorise la collaboration, l'automatisation et les cycles de feedback courts.

La pratique de l'intégration continue commence par la soumission régulière des modifications à un système de contrôle de code source/version, afin que tout le monde puisse travailler à partir de la même base. Chaque commit déclenche un build et une série de tests automatisés pour vérifier le comportement et s'assurer que la modification n'a rien cassé. Bien que l'intégration continue soit bénéfique en soi, c'est aussi la première étape vers l'implémentation d'un pipeline de CI/CD.

Les ingrédients clés de l'intégration continue sont les suivants :

- un système de contrôle des sources ou des versions contenant l'ensemble de la base de code, y compris les fichiers de code source, les bibliothèques, les fichiers de configuration et les scripts
- des scripts de build automatisés
- des tests automatisés
- une infrastructure sur laquelle il est possible d'exécuter des builds et des tests.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**			
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD

- **Que veut dire livraison continue**

La livraison continue reprend là où l'intégration continue s'est arrêtée. Étant la prochaine étape dans la construction d'un pipeline de CI/CD, la livraison continue automatise les étapes nécessaires à la mise en production de votre logiciel, mais ne permet pas un processus de publication entièrement automatisé.

L'objectif de la livraison continue est de rendre le processus de publication des logiciels plus rapide et plus fiable, de réduire le temps nécessaire pour obtenir des retours et d'apporter de la valeur aux utilisateurs plus rapidement que ne le permet un processus manuel.

Une fois que la publication est fiable et reproductible, il devient facile d'en faire plus souvent et vous pouvez commencer à livrer de petites améliorations sur une base hebdomadaire, quotidienne ou toutes les heures.

- **Que veut dire déploiement continue**

Le déploiement continu pousse la pratique DevOps consistant à automatiser les étapes de build, de test et de déploiement à son extrême logique. Si une modification du code passe avec succès toutes les étapes précédentes du pipeline, cette modification est automatiquement déployée en production sans aucune intervention manuelle. L'adoption du déploiement continu signifie que vous pouvez offrir de nouvelles fonctionnalités à vos utilisateurs aussi rapidement que possible, sans compromis sur la qualité.

Le déploiement continu est fondé sur des étapes éprouvées d'intégration continue et de livraison continue. De petites modifications du code sont régulièrement commit sur le master, soumises à un processus automatisé de build et de test avec promotion dans divers environnements de pré-production, et, si aucun problème n'est découvert, finalement déployées. La construction d'un pipeline de déploiement automatisé robuste et fiable signifie que la publication peut devenir un non-événement ayant lieu plusieurs fois par jour.

Bien que l'automatisation de la publication finale ne convienne pas à tous les projets logiciels, vous pouvez tout de même bénéficier de certains des éléments individuels impliqués dans l'implémentation efficace du déploiement continu.

- **En quoi les conteneurs favorisent l'approche CI/CD**

Pour que les résultats de vos tests automatisés soient fiables, vous devez vous assurer qu'ils s'exécutent de manière cohérente.

<table>
<tr><th>Auteur(s)</th><th colspan="2">Relu, validé et visé par :</th><th>Date de création :</th><th>Date dernière MAJ :</th></tr>
<tr><td><u>Maxime PRZYBYLO</u></td><td>Jérôme CHRETIENNE :
Resp. Secteur NUMERIQUE OCCITANIE</td><td></td><td></td><td></td></tr>
<tr><td></td><td>Florence CALMETTES :
Coordinatrice Filière SYST. & RESEAUX</td><td></td><td colspan="2" rowspan="2">Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.</td></tr>
<tr><td></td><td>Sophie POULAKOS :
Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT</td><td></td></tr>
</table>

CI/CD

Idéalement, vos environnements de test doivent être configurés de manière à être aussi similaires que possible à l'environnement de production, et ils doivent être réinitialisés entre les tests pour éviter que les incohérences environnementales ne perturbent les résultats de vos tests.

Les machines virtuelles (VM) sont depuis longtemps un choix populaire pour l'exécution des environnements de test, car vous pouvez programmer l'environnement pour qu'il se réinitialise à chaque nouveau build.

Cependant, arrêter et lancer de nouvelles machines virtuelles prend du temps, et vos scripts doivent inclure une configuration pour chaque environnement virtuel afin de fournir toutes les dépendances nécessaires à l'exécution du logiciel. Lorsque de nouvelles dépendances sont ajoutées, les scripts d'environnement doivent être mis à jour. Attention à ne pas oublier ce point, car cela peut empêcher l'exécution du build.

Vous pouvez éviter ces problèmes en intégrant votre code dans un conteneur lors de l'étape initiale de build. Un conteneur comprend toutes les dépendances dont le logiciel a besoin pour fonctionner, ce qui lui confère une grande portabilité et facilite son déploiement dans différents environnements.

Si vous hébergez votre pipeline sur votre propre infrastructure, des VM seront toujours nécessaires pour déployer les conteneurs, mais la préparation de l'environnement de test nécessite moins d'efforts, résultant ainsi à un pipeline plus efficace. Si vous exécutez votre pipeline dans le cloud, vous pouvez tirer parti des services gérés en utilisant des conteneurs et externaliser l'ensemble de l'infrastructure à votre fournisseur cloud.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD

- **Qu'est-ce qu'un pipeline CI/CD**

Un pipeline de CI/CD désigne une série d'étapes par lesquelles votre code passe pour aller de votre système de développement à votre communauté d'utilisateurs, en passant par des tests et des étapes intermédiaires.

Comme la stratégie de CI/CD consiste à exécuter ce processus de manière très régulière (généralement plusieurs fois par jour), il est essentiel d'en automatiser la plus grande partie possible, chaque étape déclenchant la suivante ou un signal si un problème est rencontré.

L'automatisation permet non seulement d'accélérer le processus global, et donc les boucles de feedback, mais aussi de garantir que chaque étape est exécutée de manière cohérente et fiable.

- **Qu'est-ce qu'un environnement de pré-production**

Le nombre d'environnements de test et intermédiaires dans votre pipeline dépend de votre produit et des exigences des parties prenantes de votre organisation. Par exemple, il peut s'agir de tests exploratoires, d'analyses de sécurité, de recherches relatives aux utilisateurs, de démonstrations de vente, d'environnements de formation et de sandboxes permettant au personnel chargé de l'assistance de reproduire les problèmes des clients.

La création et le déploiement automatisés de ces environnements sont plus efficaces que leur actualisation manuelle, et vous pouvez configurer différents déclencheurs de pipeline pour différents environnements.

Par exemple, alors que vos environnements de test peuvent être mis à jour à chaque build, vous pouvez décider d'actualiser les environnements intermédiaires moins fréquemment (éventuellement une fois par jour ou par semaine avec le dernier build réussi).

- **Quels sont les outils de CI/CD**
  - Jenkins => CI/CD
  - Gitlab-ci => CI/CD
  - Gerrit => Revue de code
  - Jira => gestion d'incident
  - Artifactory => gestionnaire de dépôt d'artefacts
  - Docker => environnement
  - Kubernetes => orchestration de containeurs

<table>
<tr><th>Auteur(s)</th><th colspan="2">Relu, validé et visé par :</th><th>Date de création :</th><th>Date dernière MAJ :</th></tr>
<tr><td><u>Maxime PRZYBYLO</u></td><td>Jérôme CHRETIENNE :
Resp. Secteur NUMERIQUE OCCITANIE</td><td></td><td></td><td></td></tr>
<tr><td></td><td>Florence CALMETTES :
Coordinatrice Filière SYST. & RESEAUX</td><td></td><td colspan="2" rowspan="2">Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.</td></tr>
<tr><td></td><td>Sophie POULAKOS :
Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT</td><td></td></tr>
</table>

CI/CD

- **Qu'est-ce qu'un runner**

Gitlab runner est un service qui va exécuter les jobs d'un pipeline de CI d'un dépôt gitlab. On peut utiliser les runner gratuit et mutualisé du cloud Gitlab ou bien installer son propre runner dans son infrastructure.

- **Qu'est-ce qu'un manifeste gitlab-ci**

Un manifeste gitlab-ci est le document indiquant les différentes étapes d'un pipeline CI/CD. Il est écrit en YAML et doit être nommé .gitlab-ci.yml pour être interprété par Gitlab comme un manifeste. Il se compose en différents jobs classifiant les différentes étapes de votre pipeline

- **Qu'est-ce que le yaml**

Avant : Yet Another Markup Language
Maintenant : YAML Ain't a Markup Language.
Langage de sérialisation de donnée fonctionnant sur le principe de clefs valeurs comme le XML ou le JSON. Sa structure se base comme python sur l'indentation des éléments pour définir des sections.

- **Qu'est-ce qu'un artefact**

Les artefacts de build sont des fichiers créés par le processus de build, tels que les paquets de distribution, les fichiers WAR, les fichiers journaux et les rapports. Les artefacts peuvent être stockés soit dans un référentiel sur votre serveur CI, soit dans un emplacement externe accessible à votre serveur CI.

- **Qu'est-ce qu'un job**

Phase d'un pipeline

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD

II/Bonnes pratiques

Mettre en place des outils de CI/CD n'est pas suffisant pour "faire du DevOps". Le DevOps s'appuie avant tout sur idéologie, un état d'esprit, une manière de penser. Il s'agit avant tout de favoriser le retour sur expérience à chaque étape du projet au lieu d'en avoir que 1 à 2 fois par mois. Il s'agit aussi d'automatiser les tâches répétitives pour gagner en productivité.

Pour que vos pipelines soient bien structurés et en adéquation avec les principes DevOps il faut suivre certaines bonnes pratiques :

Valider tôt, valider souvent

Afin de profiter des avantages de l'intégration continue, il est essentiel pour chacun de partager ses modifications avec le reste de votre équipe en effectuant un push sur la branche principale (master) et d'actualiser la version sur laquelle il travaille afin de recevoir les modifications des autres. En règle générale, il faut faire un commit vers la branche principale (master) au moins une fois par jour.

Maintenir les builds dans le vert

L'objectif est d'éviter de bâtir sur de mauvaises fondations et de maintenir le code dans un état constamment publiable. Non seulement il est beaucoup plus efficace de traiter les problèmes dès qu'ils surviennent, mais cela permet également de déployer rapidement un correctif si quelque chose ne va pas en production.

Si un build échoue pour une raison quelconque, la priorité de l'équipe doit être de le faire fonctionner. Il peut être tentant de blâmer celui qui a effectué la dernière modification et de lui laisser la tâche de régler le problème. Cependant, chercher à blâmer les membres de votre équipe produit rarement une culture d'équipe constructive et a moins de chances de révéler la cause sous-jacente d'un problème. En confiant à toute l'équipe la responsabilité de réparer un build défaillant et en essayant de comprendre ce qui a conduit à l'échec, vous pouvez améliorer l'ensemble du processus de CI/CD.

Ne faire d'un build

Une erreur courante consiste à créer un nouveau build pour chaque étape. Reconstruire le logiciel pour différents environnements risque d'introduire des incohérences et signifie que vous ne pouvez pas être sûr que tous les tests précédents ont réussi. Au lieu de cela, le même artefact de build doit être promu à travers chaque étape du pipeline de CI/CD et publié en production.

Pour mettre cela en pratique, il faut que le build ne dépende pas de l'environnement. Les variables, les paramètres d'authentification, les fichiers de configuration ou les scripts doivent être appelés par

CI/CD

le script de déploiement plutôt que d'être incorporés dans le build lui-même. Cela permet de déployer le même build dans chaque environnement pour le tester, chaque étape augmentant la confiance de l'équipe dans cet artefact de build spécifique.

Rationaliser vos tests

Il faut trouver un équilibre entre la couverture des tests et les performances. Si cela prend trop de temps pour obtenir les résultats des tests, les gens chercheront des raisons et des moyens de contourner le processus.

Faites d'abord les tests qui se terminent le plus rapidement afin d'obtenir un retour d'information le plus tôt possible, et n'investissez dans des tests plus longs que lorsque vous avez un certain degré de confiance dans le build. Compte tenu du temps nécessaire aux tests manuels et du fait que votre équipe doit être disponible pour effectuer ces tests, il est préférable de limiter cette phase jusqu'à ce que tous les tests automatisés aient abouti avec succès.

En faire le seul moyen de déployer en production

Vous avez investi dans la construction d'un pipeline de CI/CD fiable, rapide et sécurisé qui vous donne confiance en la qualité de vos builds. Vous ne voulez pas saper cet effort en permettant que le processus soit contourné pour une raison quelconque. Généralement, la demande de contourner le processus de publication est faite parce que le changement est mineur ou urgent (ou les deux), mais céder à ces demandes est une fausse économie.

Sauter les étapes des tests automatisés risque d'introduire des problèmes évitables, et la reproduction et le débogage des problèmes est beaucoup plus difficile car le build ne peut pas être immédiatement déployé dans un environnement de test.

En faire un travail d'équipe

La mise en place d'un pipeline efficace de CI/CD est autant une question de culture d'équipe et d'organisation qu'une question de processus et d'outils. L'intégration, la prestation et le déploiement en continu sont toutes des pratiques du DevOps. Ils s'appuient sur la suppression du cloisonnement traditionnel entre les développeurs, les testeurs et les opérations, et encouragent la collaboration entre les disciplines.

Le décloisonnement donne aux équipes une plus grande visibilité du processus de bout en bout et la possibilité de collaborer et de bénéficier de différents domaines d'expertise. L'entretien du pipeline ne devrait jamais être le travail d'une seule personne.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

CI/CD

III/Gitlab-ci :

Gitlab est une plateforme de développement collaborative open source reprenant le fonctionnement de Github. Ce que gitlab apporte de plus par rapport à github est l'aspect CI/CD. En effet, gitlab permet, grâce aux gitlab runners, de mettre en place des pipeline d'intégration, livraison et déploiement continue. Pour se faire il faut rédiger le manifeste, un fichier YAML portant le nom .gitlab-ci.yml et y déclarer les étapes de notre pipeline.

Le simple fait de rédiger ce document et de l'inclure à la racine de votre projet permet de lancer automatiquement le pipeline lors d'un commit sur la branche. Dans la version gratuite de gitlab, 400 minutes de CI/CD par mois sont offertes. Des minutes de CI/CD représentent du temps accordé au runners pour exécuter les pipelines.

Chaque job du pipeline est exécuté dans un container éphémère qui est détruit une fois le pipeline terminé. Les données créées lors de ce pipeline sont détruites, sauf pour les artefacts qui sont conservés.

a) Le manifeste

Dans le manifeste on va définir des stages dans lequel on va pouvoir définir des jobs. Tous les jobs s'exécutant dans le même stage vont démarrer en même temps ce qui veut dire que si vous mettez plusieurs jobs dans un même stage il faut que tous soient indépendants par rapport aux autres.

Dans un job on va retrouver

- l'image docker sur laquelle il se base
- le stage dans lequel il s'éxecute
- les potentiels services supplémentaires nécessaires
- les variables d'environnement propre au job
- une étape before_script indiquant les commandes à lancer avant de démarrer les étapes du job
- une étape script indiquant ce qu'il faut faire pour réaliser ce job
- des règles conditionnant l'exécution du job
- des artefacts à sauvegarder en sortie de job

Cette liste est non exhaustive, on peut retrouver d'autre informations dans le manifeste. Quelques précisions sur certains éléments :

Les services potentiels dont on aurait besoin peuvent être une base de donnée, un module supplémentaire pour le serveur web, comme Php, etc. Ils se définissent en indiquant une image à

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

utiliser et on peut indiquer des éléments supplémentaires de configuration, comme un alias, un script d'entrypoint ou une commande de démarrage.

Les variables d'environnement peuvent être définies à plusieurs endroits. Soit c'est une variable qui n'est utilisé que dans ce job et dans ce cas-là on met la définition de cette variable dans le job. Soit c'est une variable non sensible qui doit être utilisé dans plusieurs jobs et dans ce cas-là on met la définition à l'extérieur des jobs, au même niveau qu'un job en termes d'indentation. Soit il s'agit d'une variable sensible et dans ce cas-là on va la définir dans les paramètres CI/CD du projet.
En plus des variables que l'on définit nous-même, gitlab nous met à disposition toute une série de références à des informations du commit (comme l'id ou la référence du commit, le nom du registry mis à disposition pour le projet, etc. La liste complète des variables se trouve ici : https://docs.gitlab.com/ee/ci/variables/predefined_variables.html).

On peut dire à notre pipeline d'omettre un job en faisant commencer son nom par un point. Le point au début du nom d'un job permet également de définir un modèle de configuration de job qu'on pourra inclure ailleurs grâce à l'instruction extend.

b) Gitlab GUI

Sur la page d'administration de votre dépôt Gitlab vous pouvez visualiser toutes les étapes de vos pipelines, en cours ou terminés.

Lorsque vous faites un commit d'une branche contenant un fichier .gitlab-ci.yml un pipeline CI/CD se déclenche automatiquement. Sur la page principale de votre branche vous retrouvez, au-dessus des fichiers de votre branche, un indicateur de pipeline vous indiquant à quelle étape le runner en est.

Vous avez dans le menu de gauche un onglet CI/CD dans lequel vous pouvez retrouver vos pipelines, les jobs passés ou en cours, l'éditeur de manifeste, un planificateur pour prévoir le déclenchement de vos pipelines.

Lorsque vous cliquez sur un job en cours vous voyez ce qu'il se passe dans le container. Cela vous permet de vérifier pourquoi un job ne fonctionnerai pas. Si le job s'est bien déroulé et qu'il a généré un artefact vous avez la possibilité de télécharger cet artefact sur votre ordinateur pour vérifier son contenu.