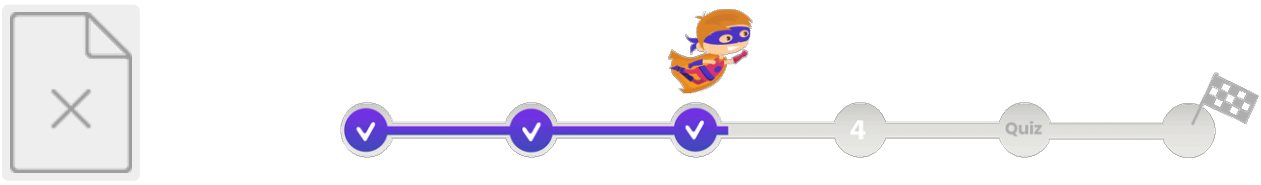
4 - Déploiement Continu



4. Déploiement Continu

Le **Déploiement Continu** (***CD en anglais pour Continuous Delivery***) est l'**étape ultime d'automatisation d'une chaîne de production**.

Le déploiement continu reprend les principes de l'intégration continue, mais en ajoutant des fonctionnalités permettant de modifier l'environnement de production automatiquement.

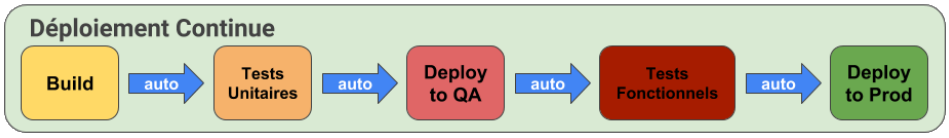


Figure 1 : Principe du déploiement continu

Le déploiement continu est le Saint Graal vers lesquels de plus en plus d'entreprise tentent de se rendre. Malheureusement il nécessite de posséder un écosystème basé sur des outils automatisables.

Vous trouverez bon nombre d'outils qui vous permettront de piloter toutes les étapes de votre pipeline de production. En anglais, on parle de CI/CD tools alors qu'en français même si on limite leur utilisation par le terme de PIC (Plateforme d'Intégration Continue) on parle bien d'outils permettant de faire du Déploiement Continu.

4.1 Jenkins

Jenkins est l'un des premiers outils d'intégration continue open-source et reste l'un de ceux le plus utilisé aujourd'hui.
Il a vu le jour sous le projet open source java nommé Hudson de Sun Microsystem en 2005. Le nom de Jenkins est né suite aux conflits après le rachat de Sun Microsystems par Oracle. Les développeurs originaux du projet Hudson et la communauté ont décidé de créer un nouveau projet basé sur le code d'Hudson (en informatique on parle de fork) pour donner la première version de Jenkins en 2011.

Figure 2 : Logo de Jenkins

Au fil des ans, Jenkins s'est transformé en un **système puissant et flexible d'automatisation des tâches** liées aux logiciels. Jenkins lui-même sert principalement de moteur d'automatisation, mais une grande partie de la logique est mise en œuvre par le biais d'une bibliothèque de plugins. Bien que cette approche offre une grande flexibilité, elle peut finir par fragiliser votre processus de CI si ce dernier vient à s'appuyer sur de trop nombreux plugins.

C'est la société Cloudbees qui, à partir de 2014, est devenu le principal contributeur du projet Jenkins. L'entreprise a embauché Kohsuke Kawaguchi le développeur en chef et fondateur du projet Jenkins afin de le nommer CTO (Chief Technology Officer). De grosses modifications ont été apportées par la suite grâce à Cloodbees.

Figure 3 : Exemple d'utilisation du plugin pipeline de Jenkins 2

En 2016, le principe de **pipeline**, succession de tâche qui s'enchaînent en fonction du résultat de la tâche précédente, arrive par le biais d'un nouveau plugin.

Ce processus peut ainsi être défini de deux manières :

- En utilisant le langage Groovy qui permet de versionner les scripts d'intégrations.
- A travers des zones de texte dans l'interface web de Jenkins.

Un nouveau plugin apparait en 2017, baptisé Blue Ocean ce dernier permet de **faciliter la création de pipeline** en générant du code groovy à l'aide d'une interface simplifiée. Cet outil permet également d'améliorer la visibilité du pipeline en **remaniant complètement l'interface graphique** de Jenkins.

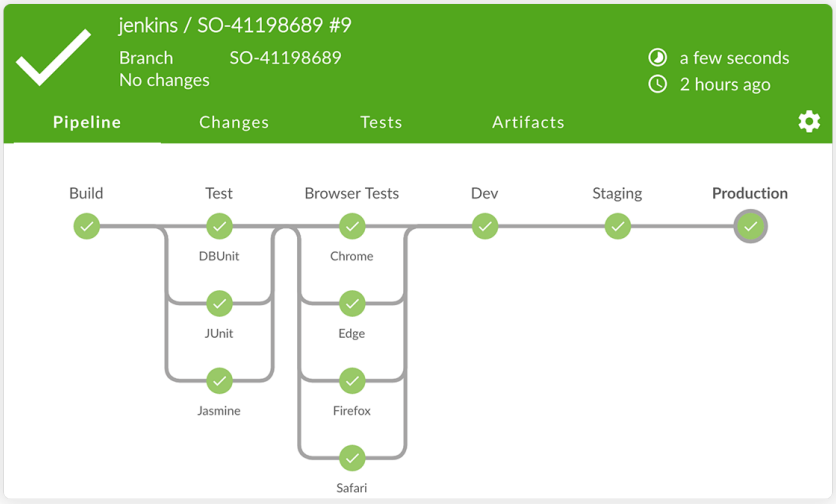


Figure 4 : Exemple d'utilisation du plugin Blue Ocean de Jenkins

Début 2018, alors que beaucoup commençaient à reprocher à Jenkins sa complexité d'utilisation avec les conteneurs Docker, une nouvelle version voit le jour sous le nom de Jenkins X. Cette version est dédiée au déploiement vers l'orchestrateur de conteneurs Kubernetes permettant ainsi à Jenkins de rattraper son retard concernant les écosystèmes Docker.

Une critique récurrente de Jenkins vient de son modèle de configuration centré sur l'utilisation de plugin. Sa capacité à définir des processus de pipeline en dehors de ces plugins peuvent parfois rendre difficile la réplication d'une configuration sur une autre instance Jenkins.

4.2 Gitlab CI

GitLab CI est un **outil d'intégration continue intégré à GitLab**, une plateforme d'outils de développement et d'hébergement de référentiel git écrits en Ruby et en Go et publiés sous licence MIT.
Initialement publié en tant que projet autonome, GitLab CI a été intégré dans le logiciel principal de GitLab avec la sortie de GitLab 8.0 en septembre 2015.

Figure 5 : Logo de Gitlab Runner

Le processus de CI/CD dans GitLab CI est défini dans un fichier, utilisant une syntaxe de configuration YAML, dans le dépôt git lui-même. Le travail est ensuite acheminé vers des machines appelées runners, qui sont faciles à mettre en place et peuvent être provisionnées sur de nombreux systèmes d'exploitation différents.
Lors de la configuration des runners, vous pouvez choisir entre différents exécuteurs comme Docker, shell, VirtualBox, ou Kubernetes pour déterminer comment les tâches sont exécutées.

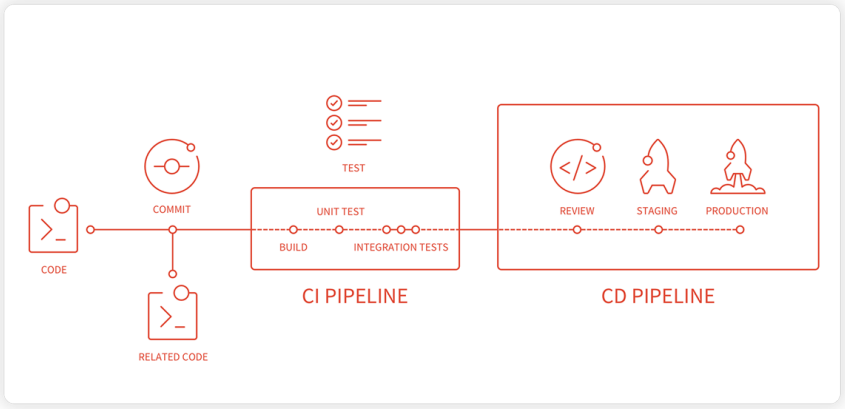


Figure 6 : Gitlab + Gitlab CI, une plateforme tout-en-un

Le couplage étroit de GitLab CI avec la plate-forme de dépôt GitLab a de fortes implications sur la manière dont le logiciel peut être utilisé :

- GitLab CI n'est donc pas une option viable pour les développeurs qui utilisent d'autres plates-formes d'hébergement git comme github, bitbucket ou même un serveur git natif.
- Le côté positif, c'est que cette intégration permet aux utilisateurs de GitLab de configurer un environnement CI/CD sans avoir à installer et à configurer un outil supplémentaire.
- Les runners peuvent être partagés entre plusieurs projets.
- Voir automatiquement l'état actuel de compilation depuis git.
- Conserver les artefacts de compilation avec le code qui les a produits.

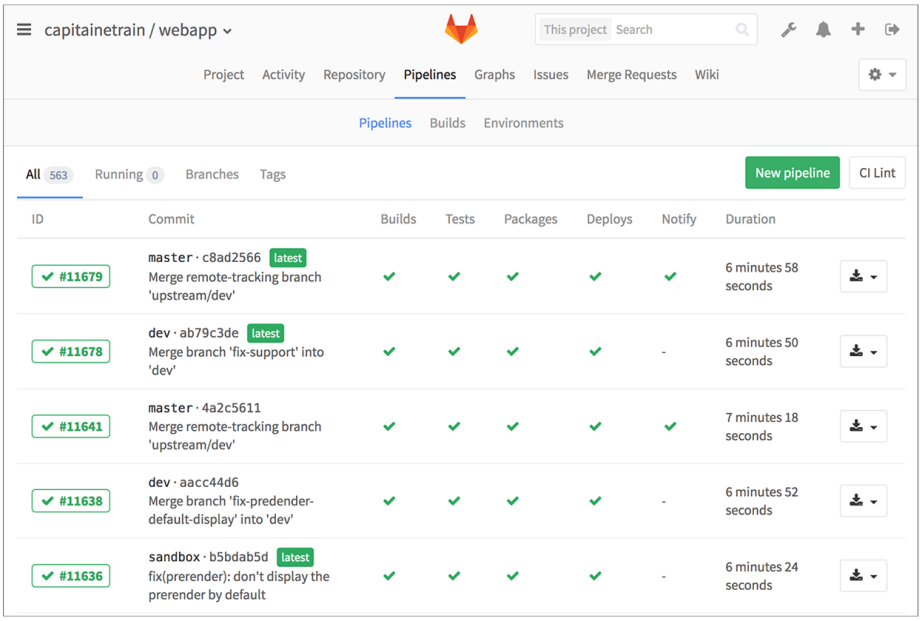


Figure 7 : Exemple de pipeline avec Gitlab CI

Les tests automatisés sont **activables depuis l'interface Web**, l'**enregistrement d'un runner** et l'**ajout d'un fichier de définition de pipeline** se fait directement dans le **repository gi**t.

Pour voir la fiche complète et les documents attachés, rendez-vous sur
https://elearning.26academy.com/course/play/5aa265f759a5ca3c65d5487b