

## TD8 : Intégration continue

Compétences

- Découvrir le principe d'intégration continue en génie logiciel
- Mise en œuvre de l'intégration continue sur GitLab
- Savoir lire les instructions élémentaires d'un fichier yaml

### Introduction

Un peu de lecture avant de commencer :

- [https://fr.wikipedia.org/wiki/Int%C3%A9gration\\_continue](https://fr.wikipedia.org/wiki/Int%C3%A9gration_continue)

Référence pour GitLab :

- <https://docs.gitlab.com/ee/ci/>

### Avant l'intégration continue

Autrefois, les développeurs 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 rendait la fusion des différents codes difficile et chronophage. Cette méthode entraînait également des bugs difficiles à détecter. La combinaison de ces différents facteurs empêchait de livrer rapidement des mises à jour au propriétaire.

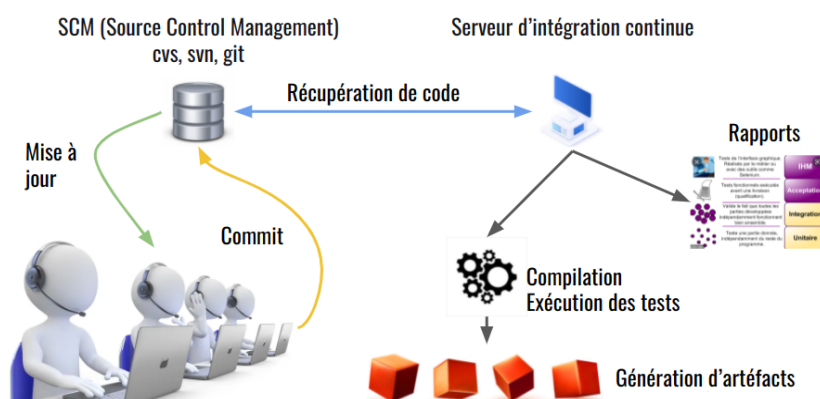


### Avec l'intégration continue

L'intégration continue a pour but de tester l'ensemble du code déjà écrit. À mesure que le développement logiciel se poursuit, à chaque publication de code, des tests sont réalisés en parallèle du développement et ce de manière continue.



### Comment fonctionne l'intégration continue?

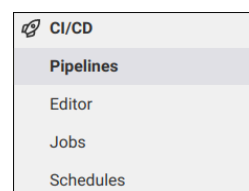


Explication du fonctionnement de l'intégration continue :

- Le développeur code les tâches qui lui ont été attribuées. Il réalise ensuite les tests unitaires correspondants afin de s'assurer que tout fonctionne correctement dans son environnement. Il fusionne son code avec celui du dépôt distant et résout les éventuels conflits, teste de nouveau son code sur sa machine et apporte des corrections si nécessaire. Si tout est ok, il publie alors son code sur le dépôt distant.
- Le serveur d'intégration continue associé au dépôt distant détecte le commit et lance les opérations de compilation, de tests et de couverture. Ces opérations sont décrites sous forme de **jobs**. Les jobs sont associés à des **stages** : build, test, coverage.
- Une fois les jobs terminés, une notification est envoyée à tous les membres de l'équipe. Elle contient notamment le statut du dernier *build and test*.
- L'équipe peut alors consulter les rapports émis lors de ce dernier build, les analyser, traiter des bugs s'il y en a, puis continuer à développer les autres phases du projet.

## Intégration continue avec GitLab CREMI

Sur le bandeau gauche de votre projet GitLab, vous avez un menu CI/CD. Ce menu va nous permettre de mettre en place un ensemble de tests unitaires et d'intégration pour s'assurer que le code ajouté au projet répond bien à toutes les attentes des différents scénarios conçus pour évaluer le code.



1. Téléchargez l'archive `src-sudent.zip`,
2. Créez un nouveau projet GitLab pour ce TD (par exemple projetTD8\_<login>),
3. Clonez le projet sur ordinateur et copiez-y le contenu de l'archive `src-student.zip`,
4. Décrivez précisément chacune des lignes du fichier `.gitlab-ci.yml`,
5. Lisez le code `Parser.py` contenu dans l'archive et indiquez en quelques lignes ce qu'il est censé faire.
6. Faites votre premier commit et mettez à jour le dépôt distant.
7. Cliquez sur le menu d'intégration continue **CI/CD** puis choisissez *Pipelines*. L'image ci-dessous montre le résultat d'un commit sur le projet. On peut voir que tous les jobs ont été réalisés avec succès.

Christophe Casseau > Video Tracker > Pipelines

All 1 Finished Branches Tags			
Filter pipelines			
Status	Pipeline	Triggerer	Stages
<div>passed</div> <div>00:00:11</div> <div>1 hour ago</div>	<div>parser_test</div> <div>#1839 main 920e04c2</div> <div>latest</div>		<div>✓ ✓ ✓</div>

Pour accéder aux rapports des différents jobs, cliquez sur *passed*. Vous devriez obtenir la fenêtre suivante. On y retrouve les noms des jobs que nous avons déclarés dans le fichier `.gitlab-ci.yml`. Il n'y a plus qu'à choisir votre job pour obtenir le rapport. Chaque job peut également être relancé de manière indépendante.

Pipeline Needs Jobs 3 Tests 0		
Build	Test	Coverage
<div>✓ build-job1</div> <div>↺</div>	<div>✓ test-job1</div> <div>↺</div>	<div>✓ coverage-job1</div> <div>↺</div>

8. À l'aide des rapports, indiquez :
  - Le nom du serveur utilisé pour l'intégration continue.
  - La version de Python utilisée ainsi que celle du package coverage.
  - Le(s) fichier(s) concerné(s) par la couverture de code.

- Le pourcentage de couverture de code par fichier.
  - Le rapport du **test-job1** indique deux warnings, lesquels? À partir du dépôt local, corriger le premier. Puis faites un nouveau commit.
9. Dans le fichier `test_parser` de votre dépôt local, décommentez le dernier test. Ce test n'est pas complètement fonctionnel. Ne corrigez pas le code dans un premier temps. Faites un commit puis un push sur GitLab. Vérifiez l'état de votre *Pipelines* dans le menu **CI/CD** et relevez la ou les erreurs consignée(s) dans les rapports.
  10. Sur votre dépôt local, modifier le code du dernier test pour que celui-ci s'effectue correctement. Puis faire un nouveau commit.
  11. Mettez en œuvre le principe d'intégration continue sur votre projet VideoTracker.