

Formation GitLab : Manipulations pratiques

El Hadji Gaye

Auteur El Hadji Gaye

Pour Formation

Date 30/09/2024

Objet Formation GitLab : Manipulations pratiques

I)	Introduction à Gitlab	3
II)	Les fonctionnalités Principales de Gitlab	4
III)	Création d'un compte GitLab	5
IV)	Quelques généralités sur GitLab CI/CD	13
V)	Configuration sur GitLab CI/CD	14
VI)	Un PipeLine en pratique	20
1.	Définir un premier PipeLine	20
2.	Barre de recherche	25
3.	L'onglet Jobs	27
4.	L'onglet Artifacts	28
5.	Dernier PipeLine executé sur une branche	29
6.	Declanchement manuel d'un Job	30
VII)	Quelques notions importantes de GitLab CI/CD	33
1.	Les étapes (stages)	33
a)	Définition	33
b)	Pratique	36
2.	Les scripts	50
a)	Définition	50
b)	Pratique	51
3.	Les variables	55
a)	Définition	55
b)	Pratique	58
4.	Les environnements	65
a)	Définition	65
b)	Pratique	70
5.	Les images docker	78
a)	Définition	78
b)	Pratique	80
6.	Contrôle du flux d'exécution	91
a)	Définition	91
b)	Pratique	96
7.	Filtrage et conditions	97
a)	Définition	97
8.	La notion de default	100
a)	Définition	100
9.	Gestion des artifacts et dependencies	102
a)	Définition	102

I) Introduction à Gitlab

GitLab est une plateforme web qui fournit un service de gestion de dépôts Git, ainsi qu'un ensemble de fonctionnalités de DevOps, d'intégration continue et de livraison continue (CI/CD), de gestion de projet et bien plus encore.

Il propose globalement les mêmes fonctionnalités que Github et c'est l'une des plateformes les plus connues.

II) Les fonctionnalités Principales de Gitlab

Gestion de versions avec Git

- Dépôts Git : créez, consultez et gérez vos dépôts de code.
- Merge requests : proposez, discutez et gérez les changements de code.
- Diff viewer : comparez les changements dans le code facilement.

CI/ CD (Intégration Continue et Livraison Continue)

- Pipelines : automatisez les tests et le déploiement de votre code.
- Runners : exécutez les tâches sur des machines spécifiques ou sous des conditions spécifiques.
- Artifacts : stockez les fichiers générés lors des pipelines pour une utilisation ultérieure.

Gestion de projet

- Issues : suivez les bugs, les tâches et les fonctionnalités à venir.
- Milestones : définissez et suivez des objectifs à court et à long terme.
- Boards : utilisez des tableaux Kanban pour la gestion agile des projets.

Collaboration et Code Review

- Commentaires : discutez des changements dans les merge requests ou les issues.
- Snippets : partagez des morceaux de code ou de texte avec d'autres.
- Wiki : documentez votre projet directement dans GitLab.

Sécurité et Conformité

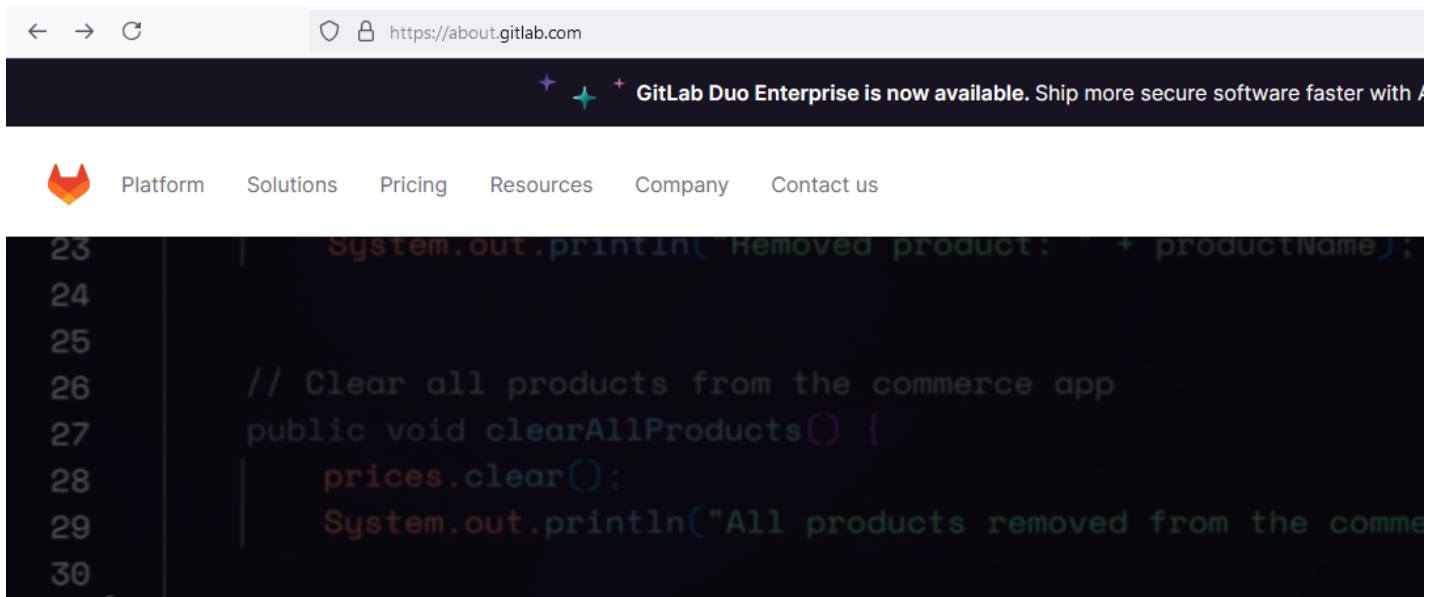
- Scanner de sécurité : détectez des vulnérabilités dans le code.
- Audit Trail : suivez qui a fait quoi et quand.
- Gestion des accès : contrôlez qui peut accéder à quoi avec des règles d'accès détaillées.

Déploiement et Monitoring

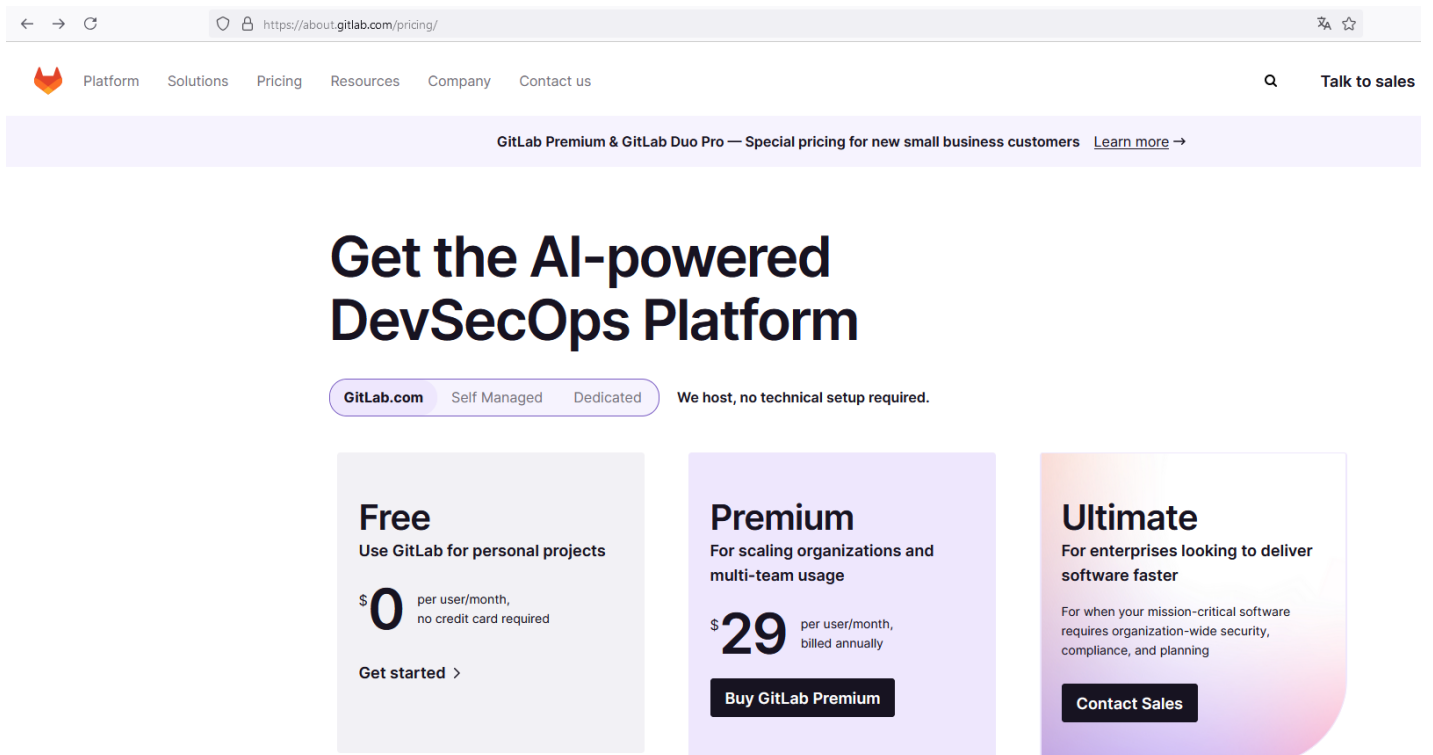
- Environnements : gérez vos environnements de déploiement.
- Auto DevOps : automatisez tout le pipeline DevOps avec des pratiques préconçues.
- Monitoring : surveillez la performance de votre application.

III) Création d'un compte GitLab

Se rendre sur le site <https://about.gitlab.com/>



Cliquer sur Pricing ce qui nous renvoie sur <https://about.gitlab.com/pricing/>



GitLab.com

Self Managed

Dedicated

Free

Use GitLab for personal projects

\$**0** per user/month,
no credit card required

Get started >

Essai gratuit de 30 jours de GitLab Ultimate

Prénom

Nom

Nom d'utilisateur

Courriel

Nous recommandons une adresse de courriel professionnelle.

Mot de passe



La longueur minimale est de 8 caractères.

SignUp|En cliquant sur Continuer ou en vous inscrivant via un service tiers, vous acceptez les [Conditions d'utilisation](#) et reconnaissez avoir pris connaissance de la [Politique de confidentialité](#) et de la [Politique en matière de cookies](#) de GitLab.

Continuer

ou


Continue with:



Google




GitHub

 Se connecter avec Google



Connectez-vous à GitLab

 elhadji.gaye83@gmail.com ▼

Annuler

Continuer

Si vous continuez, Google partagera votre nom, votre adresse e-mail, vos préférences linguistiques et votre photo de profil avec GitLab. Consultez les [Règles de confidentialité](#) et les conditions d'utilisation de GitLab.


Vous pouvez gérer Se connecter avec Google dans votre [compte Google](#).

Français (France) ▼

[Aide](#)

[Confidentialité](#)

[Conditions](#)

 Vous devez confirmer votre adresse e-mail dans les 3 jours suivant votre inscription. Si vous ne confirmez pas votre adresse e-mail dans ce délai, votre compte sera supprimé et vous devrez vous réinscrire à GitLab.

Aidez-nous à assurer la sécurité de GitLab

You are signed in as elhadji.gaye83. For added security, you'll need to verify your identity in a few quick steps.


Étape 1 : vérifiez l'adresse de courriel

Un code de vérification vous a été envoyé à
el*****@g****.com

Code de vérification

 Having trouble? [Send a new code](#) or [contact support](#).


Vérifier l'adresse de courriel

 Vous devez confirmer votre adresse e-mail dans les 3 jours suivant votre inscription. Si vous ne confirmez pas votre adresse e-mail dans ce délai, votre compte sera supprimé et vous devrez vous réinscrire à GitLab.

Aidez-nous à assurer la sécurité de GitLab

You are signed in as elhadji.gaye83. For added security, you'll need to verify your identity in a few quick steps.

Étape 1 : vérifiez l'adresse de courriel

 Processus terminé

Suivant

Welcome to GitLab, El!

To personalize your GitLab experience, we'd like to know a bit more about you. We won't share this information with anyone.

Role

Development Team Lead



I'm signing up for GitLab because:

I want to store my code



Who will be using this GitLab trial?



Just me



My company or team

Email updates (optional)



I'd like to receive updates about GitLab via email

Continue

About your company

To activate your trial, we need additional details from you.

First name

El Hadji

Last name

Gaye

Company name

El Hadji Gaye

Number of employees

1 - 99

Country or region

France

Telephone number (optional)

Allowed characters: +, 0-9, -, and spaces.

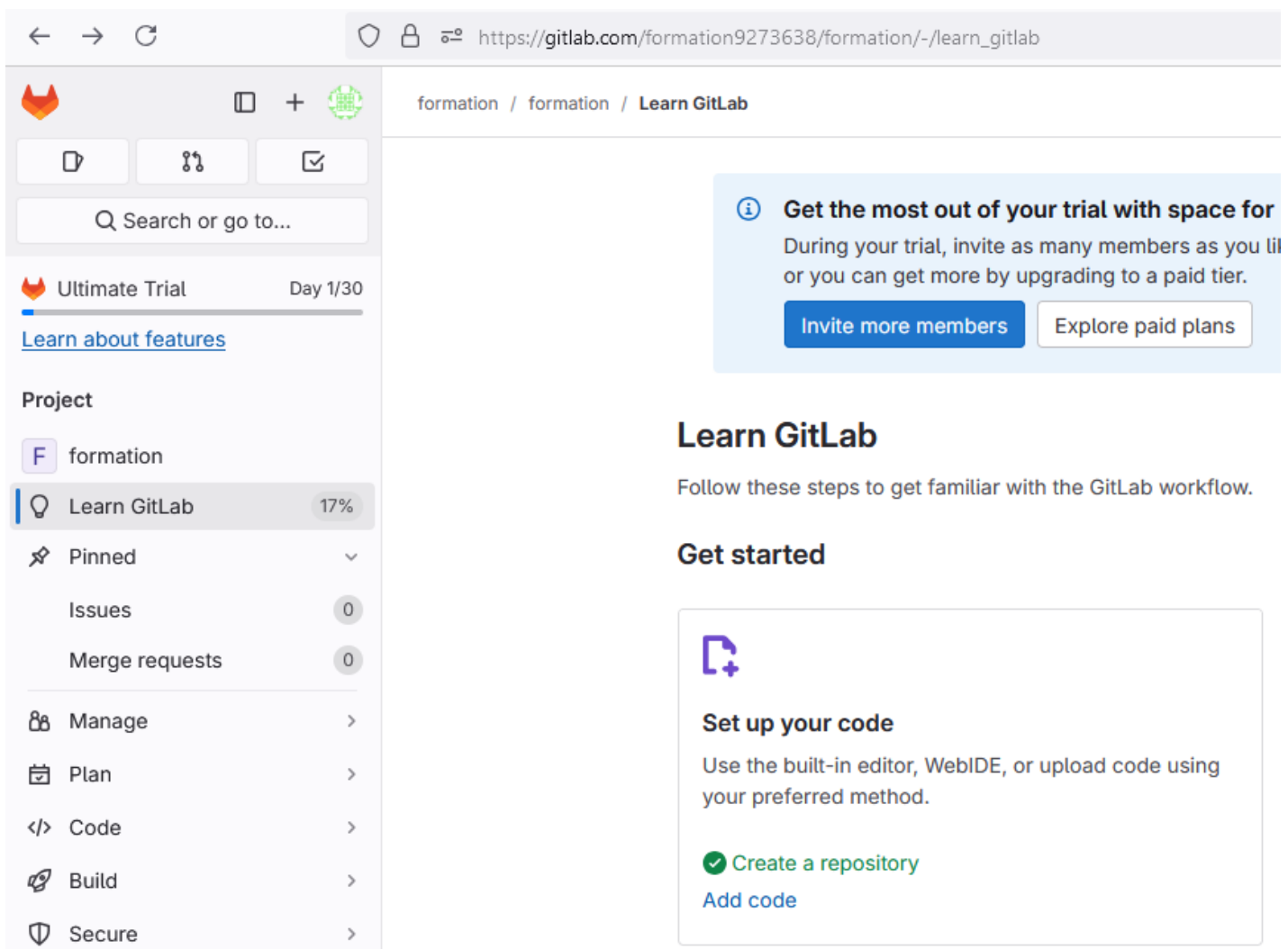
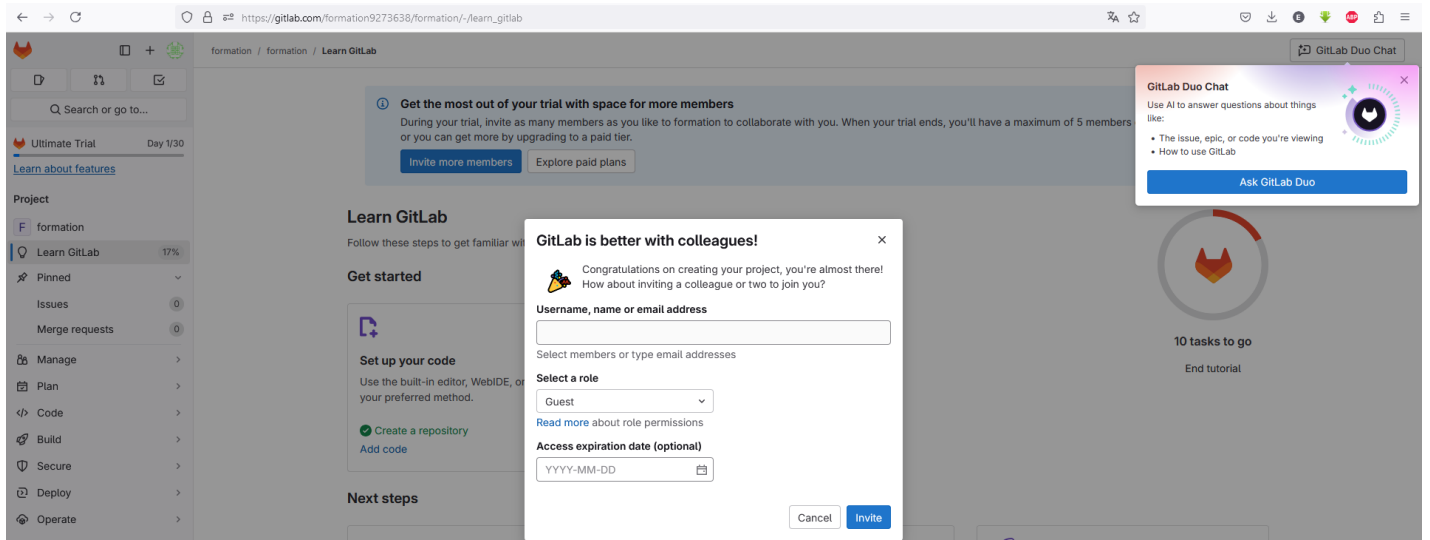
Website (optional)

Continue

Create or import your first project

Projects help you organize your work. They contain your file repository, issues, merge requests, and so much more.

Create	Import
Group name <input type="text" value="formation"/>	
Project name <input type="text" value="formation"/>	
Select a template (optional) Get started with one of our popular project templates. ? <div><div>Select</div><div>▼</div></div>	
Your project will be created at: <code>https://gitlab.com/formation9273638/formation</code> You can always change your URL later	
<input checked="" type="checkbox"/> Include a Getting Started README Recommended if you're new to GitLab	
<div>Create project</div>	



IV) Quelques généralités sur GitLab CI/CD

Il est important de vous familiariser avec certains termes couramment utilisés pour mieux comprendre le processus.

Le fichier .gitlab-ci.yml

Le point de départ pour utiliser GitLab CI/CD est un fichier .gitlab-ci.yml à la racine de votre projet. Ce fichier en format YAML définit la liste des tâches que vous souhaitez effectuer, telles que les tests et le déploiement de votre application. Bien que ce fichier puisse être nommé comme vous le souhaitez, .gitlab-ci.yml est le nom le plus couramment utilisé et recommandé. GitLab fournit également un éditeur de pipeline pour vous aider à valider la syntaxe de ce fichier avant d'effectuer des changements.

Runners

Les runners sont les agents qui exécutent les tâches (jobs) spécifiés dans votre fichier .gitlab-ci.yml. Ces agents peuvent être des machines physiques ou des instances virtuelles. Vous pouvez également spécifier une image (Docker) à utiliser pour l'exécution de la tâche. Si vous utilisez GitLab.com, des runners partagés gratuits sont déjà disponibles, mais vous pouvez également utiliser vos propres runners.

Pipelines

Un pipeline est constitué de tâches (jobs) et d'étapes (stages) :

- Les jobs définissent ce que vous souhaitez accomplir, par exemple, tester des modifications de code ou déployer dans un environnement de préproduction.
- Les jobs sont groupés en stages, comme build, test, ou deploy, et chaque stage contient au moins un job.

Voici à quoi peut ressembler la vue dans **Build > pipelines**.

All1,000+

Finished

Branches

Tags

Clear runner caches




CI lint

Run pipeline

Filter pipelines

Q

Show Pipeline ID ▾

Status	Pipeline	Created by	Stages	
<div><div>Warning</div><div>02:24:47</div><div>2 days ago</div></div>	<div>Scheduled Ruby 3.2 ruby3_2 branch</div> <div>#1234145533</div> <div>P ruby3_2 008a47ea</div> <div>scheduled</div>	<div></div>	<div><div><div><div>✓</div><div>✓</div><div>✓</div><div>✓</div><div>✓</div><div>✓</div><div>!</div><div>✓</div><div>✓</div></div><div><div>✓</div><div>»</div><div>→</div><div>✓</div><div>✓</div><div>✓</div><div>+30</div></div></div></div>	<div><div>▶ ▾</div><div>↺</div><div>⬇ ▾</div></div>
<div><div>Passed</div><div>00:00:59</div><div>2 days ago</div></div>	<div>Merge branch 'generalize-ruby-sync' into 'rub...</div> <div>#1234144111</div> <div>P ruby-sync 6dc82a4d</div> <div>scheduled latest</div>	<div></div>	<div><div><div><div>✓</div><div>»</div></div></div></div>	<div><div>⬇ ▾</div></div>
<div><div>Failed</div><div>00:35:06</div><div>2 days ago</div></div>	<div>Ruby 3.1 MR [types: qa_code_spec-predictive]</div> <div>#1234128996</div> <div>1 147325 0bd7ba8a</div> <div>merged results</div>	<div></div>	<div><div><div><div>✓</div><div>✓</div><div>✓</div><div>✓</div><div>✓</div><div>✓</div><div>✗</div><div>✓</div><div>»</div></div><div><div>»</div><div>»</div><div>→</div><div>✓</div><div>✓</div><div>✓</div></div></div></div>	<div><div>▶ ▾</div><div>↺</div><div>⬇ ▾</div></div>

V) Configuration sur GitLab CI/CD

Il est important de vous familiariser avec certains termes couramment utilisés pour mieux comprendre le processus.

Nous allons configurer GitLab CI/CD par l'intermédiaire du fichier **.gitlab-ci.yml**



GitLab

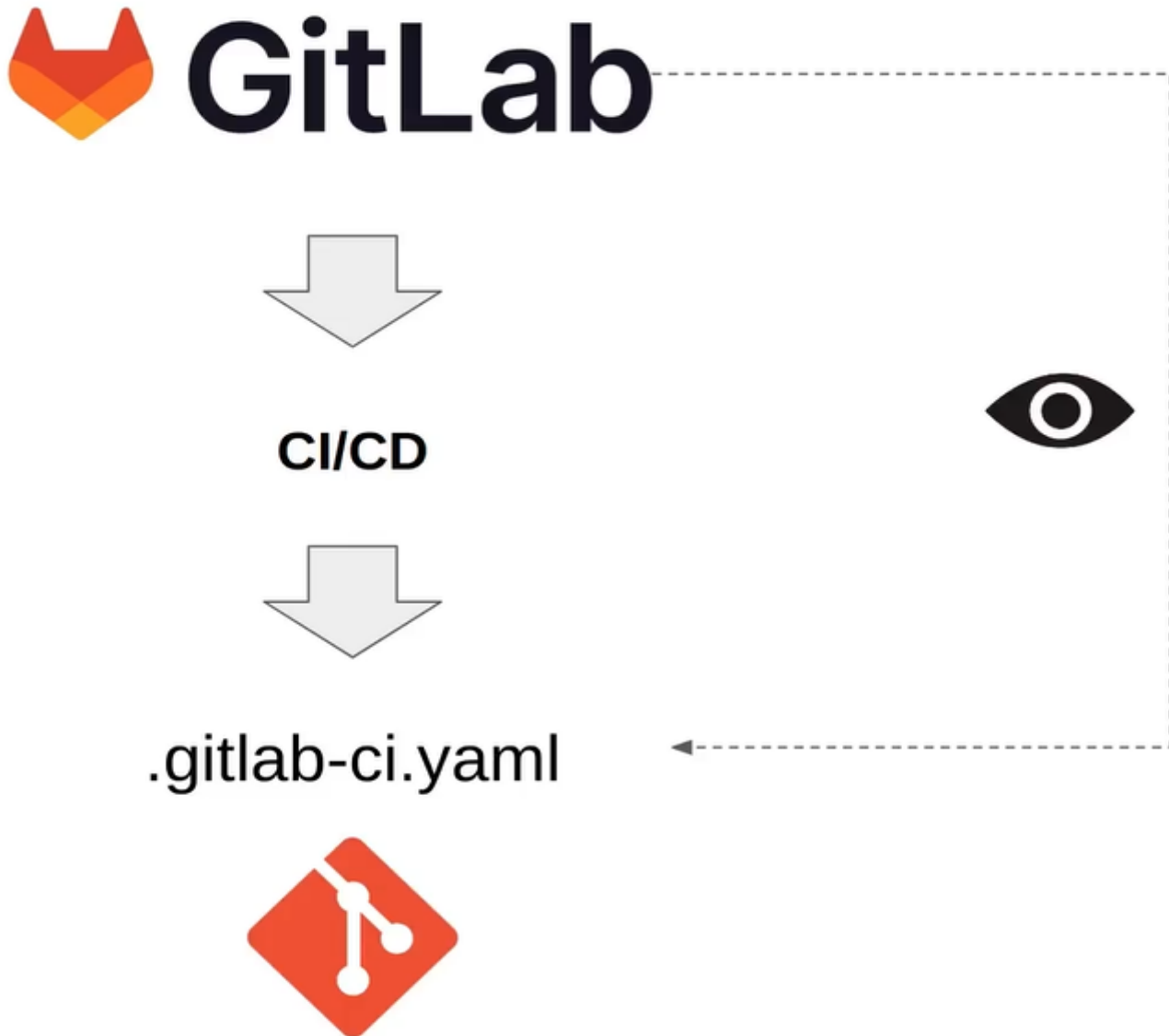


CI/CD



.gitlab-ci.yml

Nous allons mettre notre fichier **.gitlab-ci.yml** à la racine de notre projet ce qui va permettre à Gitlab de pouvoir detecter les processus **CI/CD**.

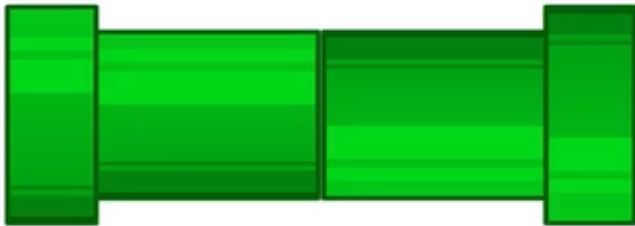


Grâce au fichier **.gitlab-ci.yml** nous allons configurer notre pipeline.

.gitlab-ci.yml



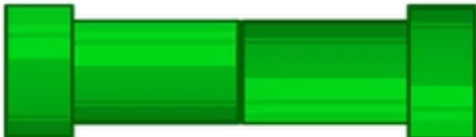
Pipeline



On peut imaginer le pipeline comme étant une tube dans lequel vont transiter des choses. Le but en définitif est de réaliser des actions avec ces choses.

Il va falloir faire tourner les pipelines sur des machines physiques.

Pipeline



Pour faire tourner le Pipeline nous allons avoir besoin dans la machine physique d'un agent. L'ensemble Machine physique + Agent va former ce qu'on appelle un Runner.

Pipeline



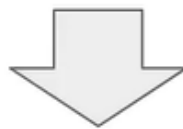
runner



agent

Gitlab CI/CD nous met à disposition un certain nombre de Runner mais il est tout à fait possible de créer ses propres Runners.

Pipeline



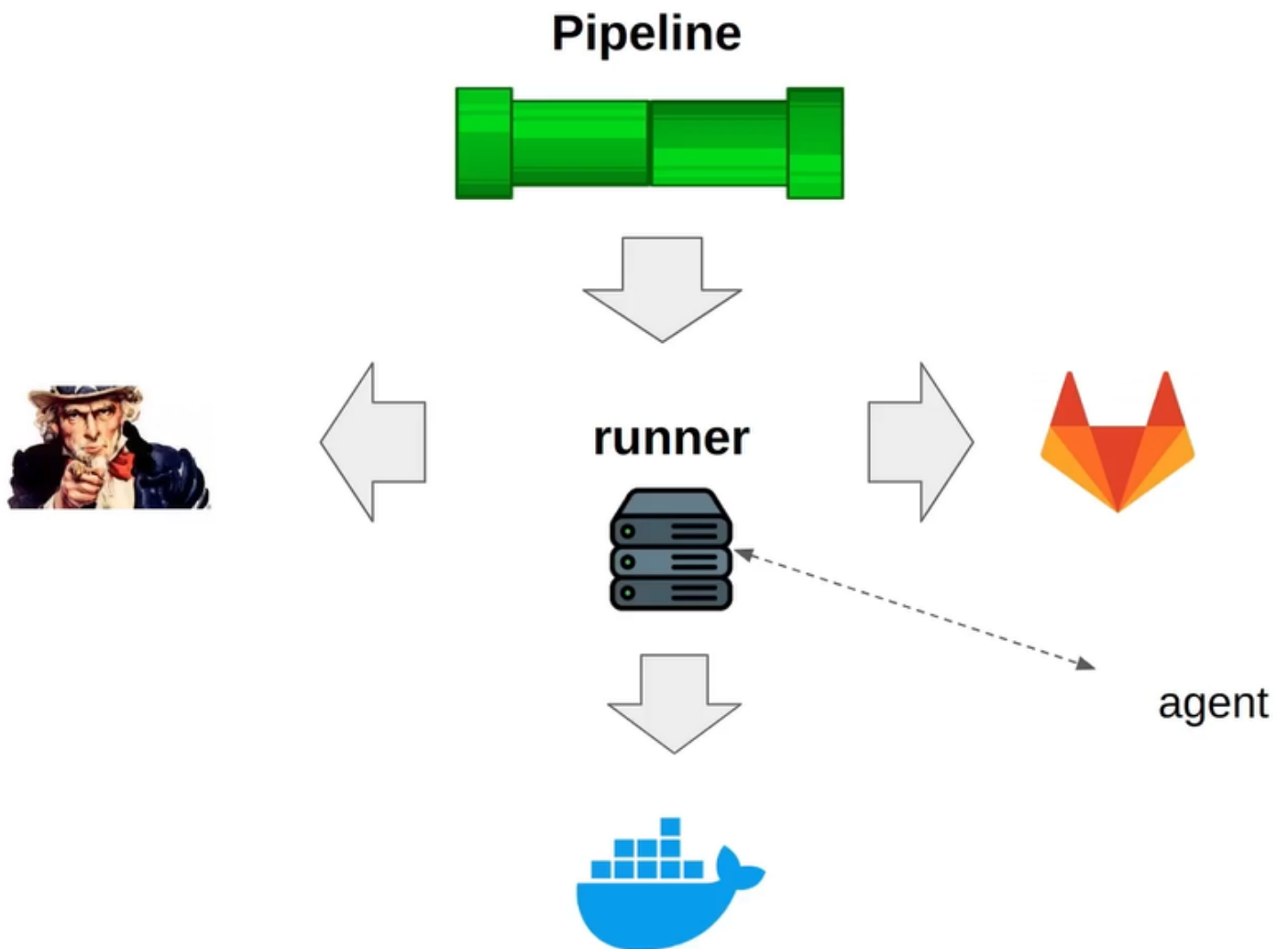
runner



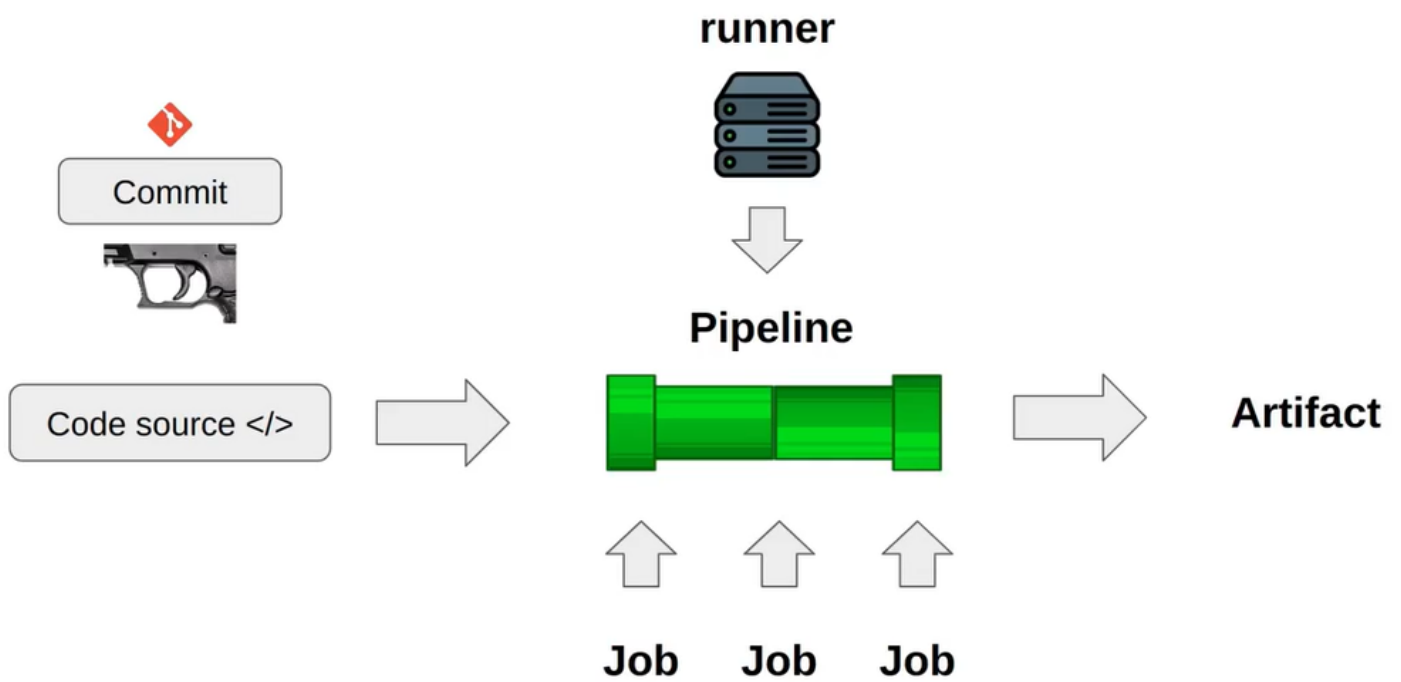
agent



Pour le runner il y a une certaine abstraction sur le système d'exploitation on peut même utiliser Docker.



Attention en fonction de la technologie utilisée il va falloir initialiser les bonnes images pour avoir accès aux outils du langage en question. C'est-à-dire si vous décidez d'exécuter du Java ou du Python il faut qu'il soit installé dans votre image Docker.



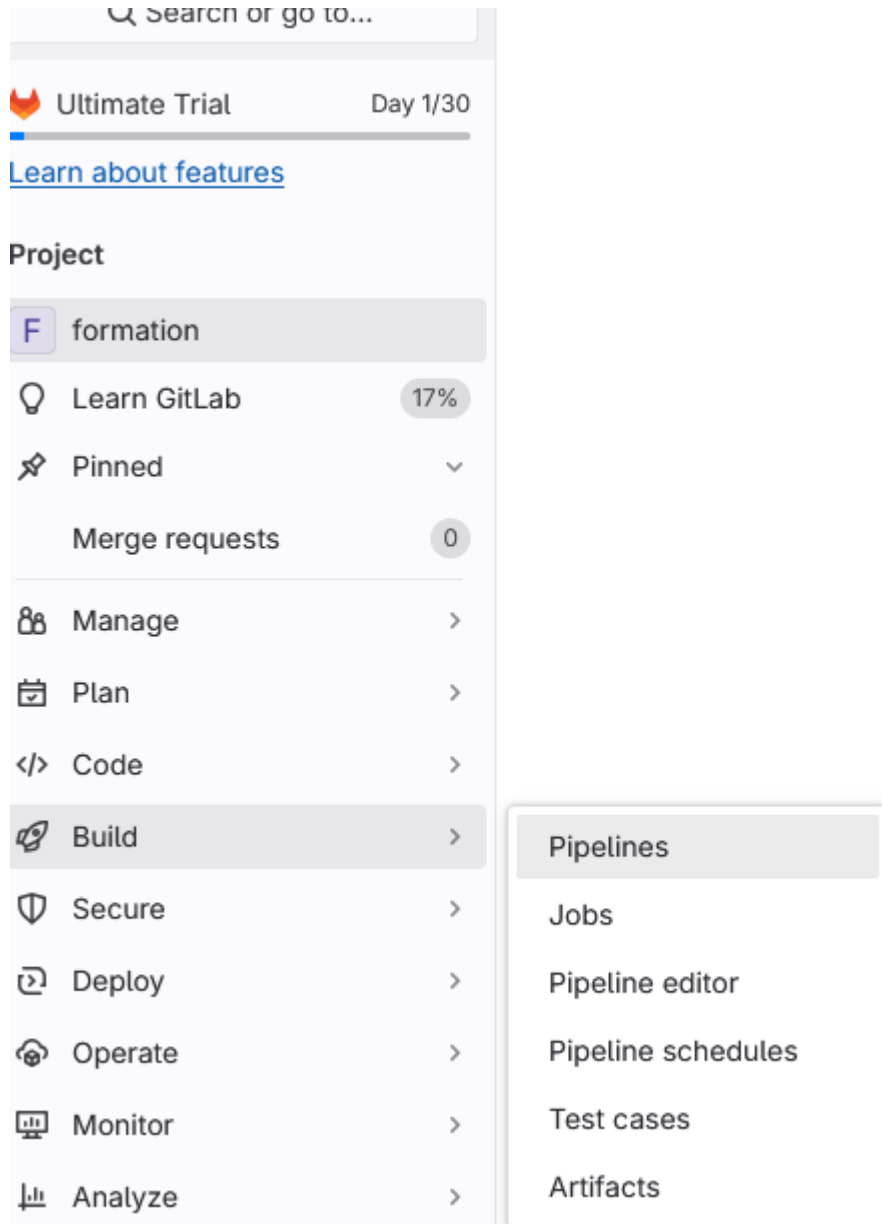
Le code source du développeur va passer à travers du Pipeline par l'intermédiaire du commit Git. Ce dernier est donc le déclencheur. Les actions ou Jobs du Pipeline sont par la suite exécuté par l'intermédiaire du Runner. En sortie de l'exécution du Pipeline nous allons avoir des artefacts.

VI) Un PipeLine en pratique

1. Définir un premier PipeLine

Se rendre sur votre repository GitLab (pour moi ce sera

<https://gitlab.com/formation9273638/formation>) puis cliquer sur Build → PipeLine



Get started with GitLab CI/CD

Learn the basics of pipelines and .yaml files

Use a sample `.gitlab-ci.yml` template file to explore how CI/CD works.



"Hello world" with GitLab CI

Get familiar with GitLab CI syntax by setting up a simple pipeline running a "Hello world" script to see how it runs, explore how CI/CD works.

[Try test template](#)

Cliquer sur **Try test template**.

On obtient pour le contenu par défaut de **.gitlab-ci.yml**

```
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%"

lint-test-job: # This job also runs in the test stage.
  stage: test  # It can run at the same time as unit-test-job (in parallel).
  script:
    - echo "Linting code... This will take about 10 seconds."
    - sleep 10
    - echo "No lint issues found."

deploy-job:  # This job runs in the deploy stage.
  stage: deploy # It only runs when *both* jobs in the test stage complete successfully.
  environment: production
  script:
    - echo "Deploying application..."
    - echo "Application successfully deployed."
```

Quand vous avez fini votre modification adapter votre message de commit puis lancer votre commit sur votre branche (ici la branche main).

Commit message

Update .gitlab-ci.yml file



Branch

main


Commit changes


Reset

Update .gitlab-ci.yml file

 **Failed** El Hadji Gaye created pipeline for commit 61af5da9  1 minute ago, finished 1 minute ago

For `main`








`latest`  **error** 0 jobs

 **Before you can run pipelines, we need to verify your account.**
We won't ask you for this information again. It will never be used for marketing purposes.



[Verify my account](#)

Pipeline Jobs 0 Tests 0


Notre Pipeline ne se declanche pas correctement car notre compte GitLab n'est pas verifié donc suivre la procedure pour verifier votre compte puis relancer votre PipeLine à nouveau.

All 1	Finished	Branches	Tags	Clear runner caches	New pipeline
<input type="text" value="Filter pipelines"/>				<input type="button" value="Q"/>	Show Pipeline ID ▾
Status	Pipeline	Created by	Stages		
 Failed 11 minutes ago	Update .gitlab-ci.yml file #1463317425  main  61af5da9 			 ▾	
	<code>latest</code>  error				

Update .gitlab-ci.yml file

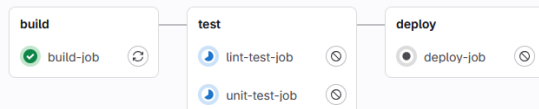
 **Running** El Hadji Gaye created pipeline for commit 61af5da9  just now

For `main`

`latest` 4 jobs  In progress, queued for 0 seconds

[Cancel pipeline](#) [Delete](#)

Pipeline Jobs 4 Tests 0



Quelques secondes plus tard on voit que tous les job du pipeLine ont été executé :

Update .gitlab-ci.yml file

✓ Passed

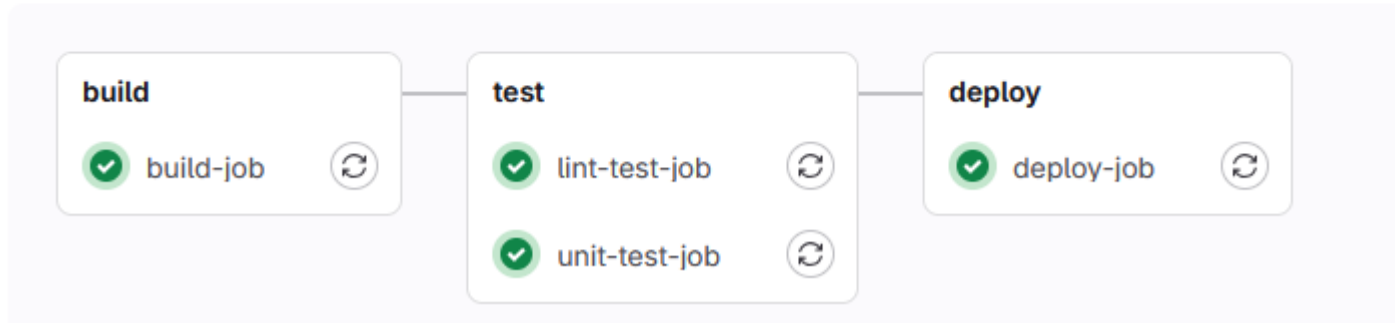
El Hadji Gaye created pipeline for commit 61af5da9

2 minutes ago, finished just now

For main

latest 4 jobs 3.14 2 minutes 27 seconds, queued for 0 seconds

Pipeline Jobs 4 Tests 0



All2

Finished

Branches

Tags

Clear runner caches

New pipeline

Filter pipelines

Q

Show Pipeline ID ▾

Status	Pipeline	Created by	Stages
<div>✓ Passed</div> <div>00:02:27</div> <div>7 minutes ago</div>	<div>Update .gitlab-ci.yml file</div> <div>#1463320313 61af5da9 </div> <div>latest</div>		<div><div>✓</div><div>✓</div><div>✓</div></div> <div></div>

Vous pouvez cliquer sur un Job en particulier pour voir toute l’exécution du Job. A titre d’exemple cliquer sur **build-job** :

build-job

✓ Passed Started 18 minutes ago by El Hadji Gaye

Search visible log output

```
1 Running with gitlab-runner 17.4.0~pre.110.g27400594 (27400594)
2   on blue-6.saas-linux-small-amd64.runners-manager.gitlab.com/default n8vMRS9Z, system ID: s_a899fcd611a3
3 Resolving secrets
4 Preparing the "docker+machine" executor
5 Using Docker executor with image ruby:3.1 ...
6 Pulling docker image ruby:3.1 ...
7 Using docker image sha256:baf5a3575bb23602cc4df3f00d45cea103d1ce276ef56080051c42ed8f96dd76 for ruby:3.1 with digest ruby@sha256:22c659ad4fd97f61d4c986bfd7b
86155d28fc0d6396fda4d5b17c454c8e9a8ca ...
8 Preparing environment
9 Running on runner-nn8vmrs9z-project-61863328-concurrent-0 via runner-nn8vmrs9z-s-l-s-amd64-1726914412-eff74211...
10 Getting source from Git repository
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/formation9273638/formation/.git/
13 Created fresh repository.
14 Checking out 61af5da9 as detached HEAD (ref is main)...
15 Skipping Git submodules setup
16 $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 Executing "step_script" stage of the job script
18 Using docker image sha256:baf5a3575bb23602cc4df3f00d45cea103d1ce276ef56080051c42ed8f96dd76 for ruby:3.1 with digest ruby@sha256:22c659ad4fd97f61d4c986bfd7b
86155d28fc0d6396fda4d5b17c454c8e9a8ca ...
19 $ echo "Compiling the code..."
20 Compiling the code...
21 $ echo "Compile complete."
22 Compile complete.
23 Cleaning up project directory and file based variables
24 Job succeeded
```


2. Barre de recherche

Se rendre à gauche de votre menu cliquer sur Pipelines pour voir la liste de vos pipeline.

Status	Pipeline	Created by	Stages
Passed 00:02:27 1 hour ago	Update .gitlab-ci.yml file #1463320313 main 61af5da9 latest		✓ ✓ ✓
Failed 1 hour ago	Update .gitlab-ci.yml file #1463317425 main 61af5da9 latest error		

Il est donc possible de rechercher :

- Par Auteur :

Status	Pipeline	Created by	Stages
Passed 00:02:27 1 hour ago	Update .gitlab-ci.yml file #1463320313 main 61af5da9 latest		✓ ✓ ✓
Failed 1 hour ago	Update .gitlab-ci.yml file #1463317425 main 61af5da9 latest error		

- Par Branche :













Status	Pipeline	Created by	Stages
Passed 00:02:27 1 hour ago	Update .gitlab-ci.yml file #1463320313 main 61af5da9 latest		✓ ✓ ✓
Failed 1 hour ago	Update .gitlab-ci.yml file #1463317425 main 61af5da9 latest error		

- Par Status :

All 2 Finished Branches Tags Clear runner c			
<div>Status = Passed X</div>			
Status	Pipeline	Created by	Stages
<div>✓ Passed</div> <div>⌚ 00:02:27</div> <div>📅 1 hour ago</div>	<div>Update .gitlab-ci.yml file</div> <div>#1463320313 main 61af5da9 </div> <div>latest</div>		<div>✓✓✓</div>
<div>✗ Failed</div> <div>📅 1 hour ago</div>	<div>Update .gitlab-ci.yml file</div> <div>#1463317425 main 61af5da9 </div> <div>latest error</div>		

3. L'onglet Jobs

Toujours sur le menu de gauche cliquer sur l'onglet Jobs pour voir la liste des Jobs contenu dans les pipelines :





















All 4 Finished		Search or filter jobs...			
Status	Job	Pipeline	Coverage		
 Passed ⌚ 00:00:29 🕒 1 hour ago	#7884336156: deploy-job 📁 main → 61af5da9	#1463320313 created by  Stage: deploy			
 Passed ⌚ 00:00:40 🕒 1 hour ago	#7884336155: lint-test-job 📁 main → 61af5da9	#1463320313 created by  Stage: test			
 Passed ⌚ 00:01:28 🕒 1 hour ago	#7884336154: unit-test-job 📁 main → 61af5da9	#1463320313 created by  Stage: test			
 Passed ⌚ 00:00:29 🕒 1 hour ago	#7884336153: build-job 📁 main → 61af5da9	#1463320313 created by  Stage: build			

4. L'onglet Artifacts

Cliquer sur l'onglet Artifact pour voir la liste des artifacts qui ont été produit par l'exécution des pipeline :

Artifacts

Total artifacts size 0 B

<input type="checkbox"/>	Artifacts	Job	Size	Created	
<input type="checkbox"/>	> 1 file	 deploy-job GO #1463320313  61af5da9 P main	2.07 KiB	1 hour ago	  
<input type="checkbox"/>	> 1 file	 lint-test-job GO #1463320313  61af5da9 P main	2.12 KiB	1 hour ago	  
<input type="checkbox"/>	> 1 file	 unit-test-job GO #1463320313  61af5da9 P main	2.12 KiB	1 hour ago	  
<input type="checkbox"/>	> 1 file	 build-job GO #1463320313  61af5da9 P main	2.04 KiB	1 hour ago	  

5. Dernier Pipeline exécuté sur une branche

Il suffit d'aller votre repository et l'information se trouvera en dessous de la liste des branches :

F

formation

Star 0 Fork 0

main

formation /

History

Find file

Edit

Code

Update .gitlab-ci.yml file
El Hadji Gaye authored 1 hour ago

61af5da9

Name	Last commit	Last update
.gitlab-ci.yml	Update .gitlab-ci.yml file	1 hour ago
README.md	Initial commit	21 hours ago

README.md

formation

2 Commits

1 Branch

0 Tags

4 KiB Project Storage

1 Environment

README

CI/CD configuration

[+ Add LICENSE](#)

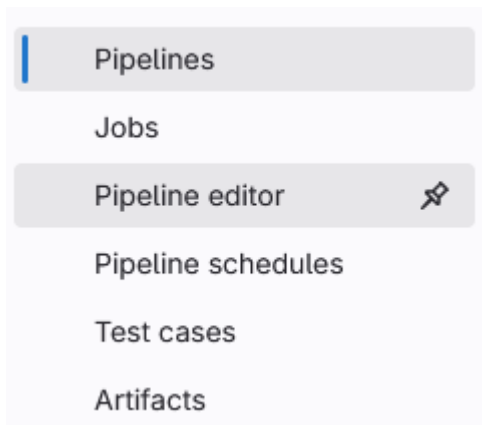
[+ Add CHANGELOG](#)

[+ Add CONTRIBUTING](#)

[+ Add Kubernetes cluster](#)

6. Declanchement manuel d'un Job

Pour tester le declanchement manuel il suffit juste de modifier le fichier `.gitlab-ci.yml` pour cela cliquer sur l'onglet pipeline editor.



```
44  deploy-job:      # This job runs in the deploy stage.
45      stage: deploy # It only runs when *both* jobs in the test stage complete successfully.
46      environment: production
47      when: manual
48      script:
49      - echo "Deploying application..."
50      - echo "Application successfully deployed."
51
```

Commit message

Update .gitlab-ci.yml file

Branch

main

```
deploy-job:  # This job runs in the deploy stage.
stage: deploy # It only runs when *both* jobs in the test stage complete successfully.
environment: production
when: manual
script:
- echo "Deploying application..."
- echo "Application successfully deployed."
```

Une fois que le commit est validé on obtient :

All5FinishedBranchesTags

Clear runner cachesNew pipeline

Filter pipelines

Show Pipeline ID

Status	Pipeline	Created by	Stages
<div>Passed</div> <div>00:01:59</div> <div>19 minutes ago</div>	<div>Update .gitlab-ci.yml file</div> <div>#1463436986</div> <div>main</div> <div>b2a04e25</div> <div>latest</div>		<div></div> <div></div> <div></div>

For main

latest 4 jobs 3.12 2 minutes 27 seconds, queued for 4 seconds

Pipeline

Jobs4

Tests0

build

build-job

test

lint-test-job

unit-test-job

deploy

deploy-job

Run

Update .gitlab-ci.yml file

Running El Hadji Gaye created pipeline for commit b2a04e25 23 minutes ago

For main

latest 5 jobs In progress, queued for 4 seconds

Pipeline

Jobs4

Tests0

build

build-job

test

lint-test-job

unit-test-job

deploy

deploy-job

Cancel

Update .gitlab-ci.yml file

✓ Passed El Hadji Gaye created pipeline for commit `b2a04e25` 25 minutes ago, finished just now

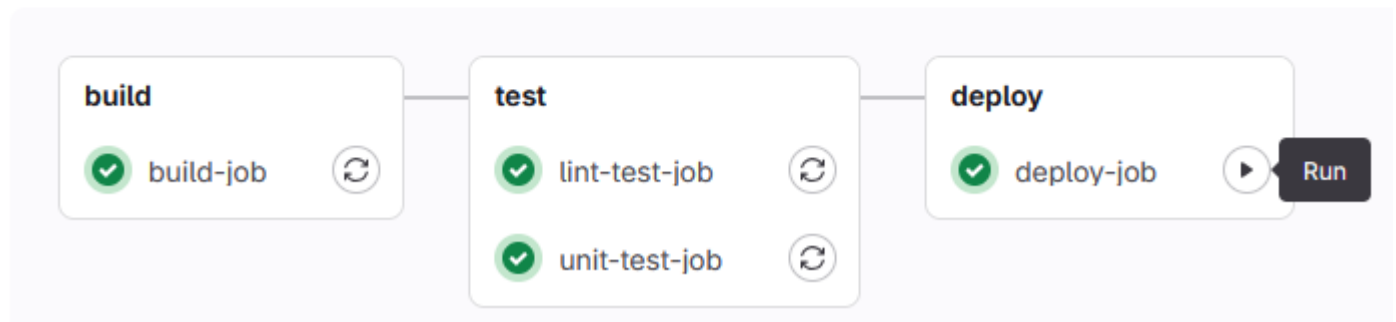
For `main`

latest 5 jobs 3.64 2 minutes 30 seconds, queued for 4 seconds

Pipeline

Jobs 4

Tests 0



VII) Quelques notions importantes de GitLab CI/CD

1. Les étapes (stages)

a) Définition

Le mot-clé stages est un élément crucial pour organiser votre pipeline CI/CD en groupes logiques de tâches (jobs). Dans cette leçon, nous plongerons dans le fonctionnement des étapes et comment elles affectent l'exécution du pipeline.

Déclaration et syntaxe

La syntaxe pour déclarer une étape ou stage est :

```
stages:  
- build  
- test  
- deploy
```

Si stages n'est pas défini dans **.gitlab-ci.yml**, les étapes par défaut sont .pre, build, test, deploy et .post.

Ordre d'exécution

L'ordre des éléments dans stages définit l'ordre d'exécution des jobs.

Les jobs dans la même étape sont exécutés en parallèle.

Les jobs de l'étape suivante ne s'exécutent que si tous les jobs de l'étape précédente réussissent.

Notion d'étape par défaut et étapes affichées

Si un job ne spécifie pas une étape, il est assigné à l'étape test.

Si une étape est définie mais qu'aucun job ne l'utilise, elle n'apparaîtra pas dans le pipeline.

Utilisation de .pre et .post

L'utilisation de .pre et .post peut être bénéfique pour les configurations de conformité.

Si un pipeline contient uniquement des jobs dans les étapes .pre ou .post, il ne s'exécutera pas.

```
stages:  
- .pre  
- build  
- .post
```

Exemple :

```
stages:
- .pre
- build
- deploy
- .post

prepare_environment:
  stage: .pre
  script:
    - echo "Mise en place de l'environnement..."
    - mkdir build/

build_project:
  stage: build
  script:
    - echo "Construction du projet..."
    - make build

deploy_project:
  stage: deploy
  script:
    - echo "Déploiement du projet..."
    - make deploy

cleanup:
  stage: .post
  script:
    - echo "Nettoyage..."
    - rm -rf build/
```

D'autres cas d'utilisation peuvent être par exemple la validation de configurations dans l'étape .pre et l'envoi de notifications (par exemple sur slack) lorsque le pipeline est fini dans l'étape .post.

Ignorer l'ordre des étapes avec needs

Le mot-clé `needs` permet de faire démarrer un job plus tôt en ignorant l'ordre des étapes.

```
job_1:
  stage: build
  script: echo "Building du front..."

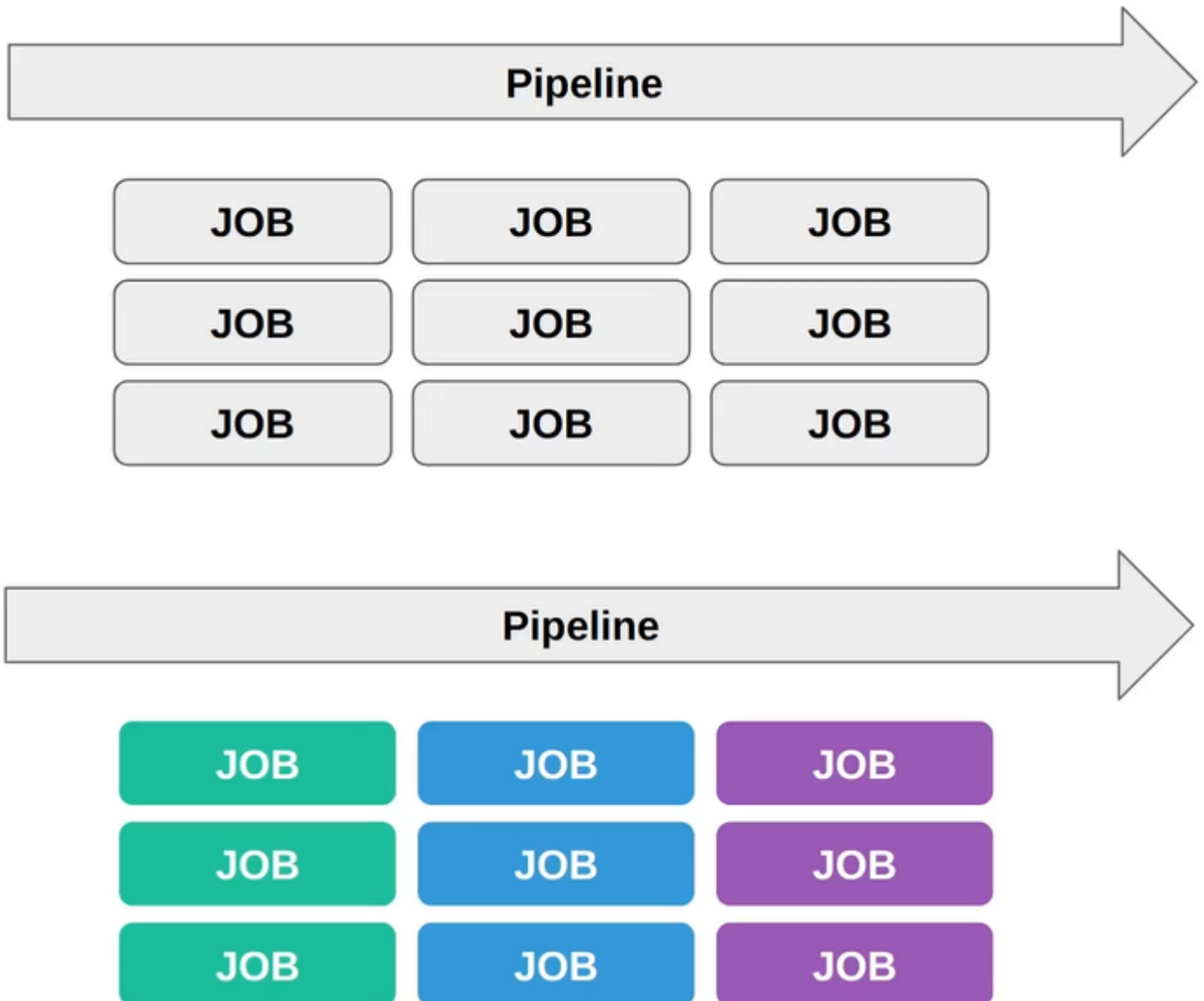
job_2:
  stage: build
  script:
    - echo "Building du back..."
    - sleep 20

job_3:
  stage: test
  script: echo "Testing du front..."
  needs: ["job_1"]
```

Dans cet exemple, `job_3` peut démarrer dès que `job_1` a terminé, sans attendre que tous les jobs de l'étape `build` se terminent (ici `job_2`).

b) Pratique

Les stages vont nous de définir l'ordre d'exécution des jobs mais aussi la façons dont ils vont s'exécuté.

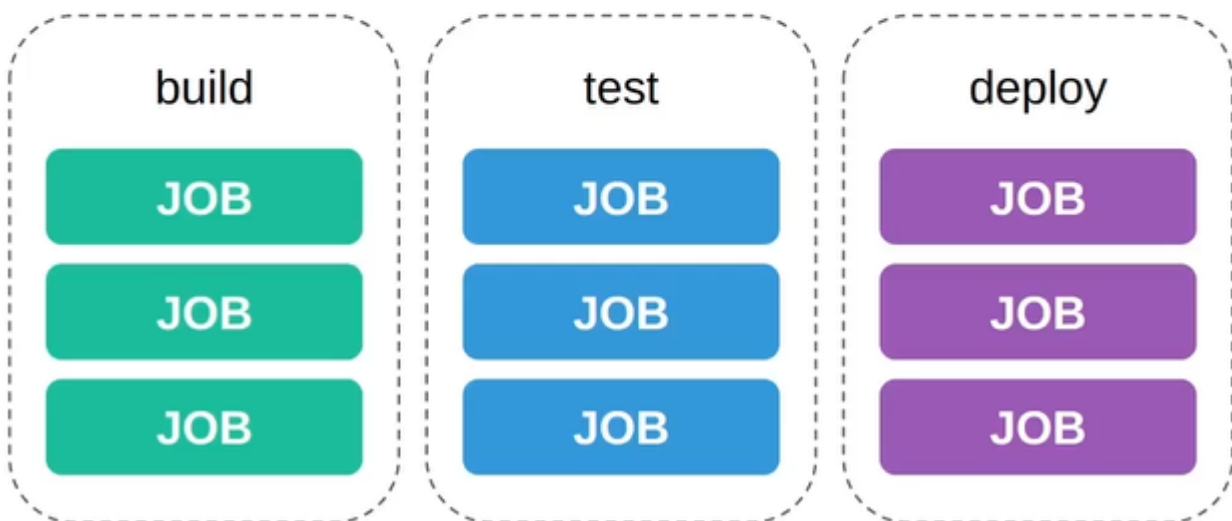




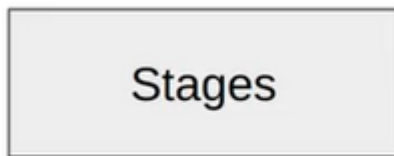
.gitlab-ci.yml

Stages

- **build (1)**
- **test (2)**
- **deploy (3)**
- **... (+1)**

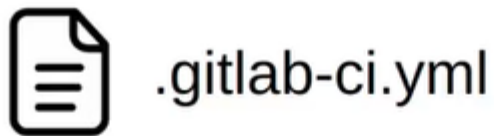


Dans la clés stage comme le stage **build** a été déclaré en premier alors tous les jobs de **build** seront executé en premier, puis on passe à **test** puis ce sera **deploy**. Pour plus d'efficacité tous les jobs d'un stage sont exécutés en parallèle.



- **build (1)**
- **test (2)**
- **deploy (3)**
- **... (+1)**

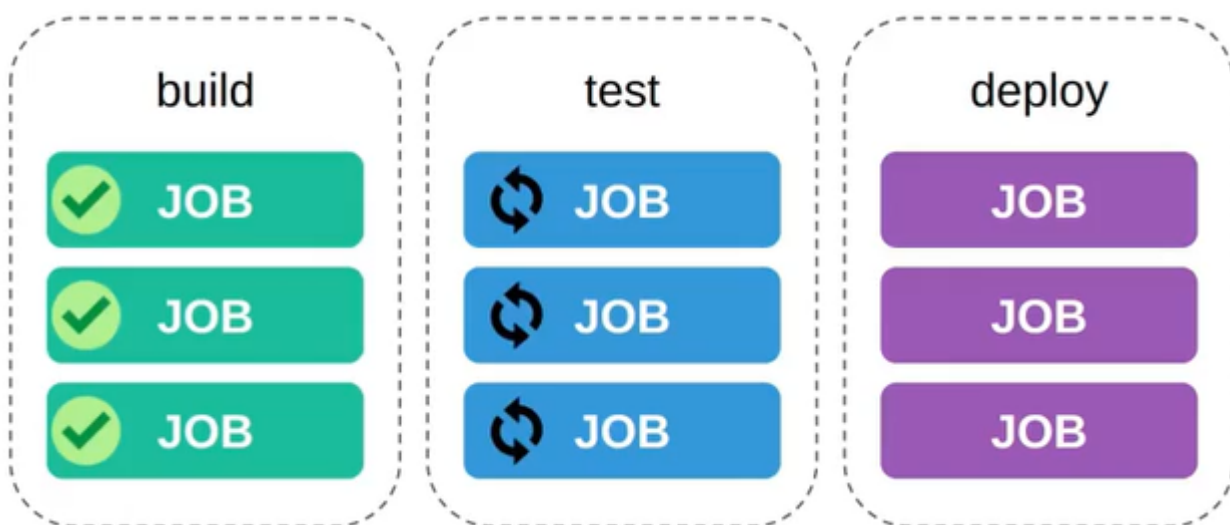
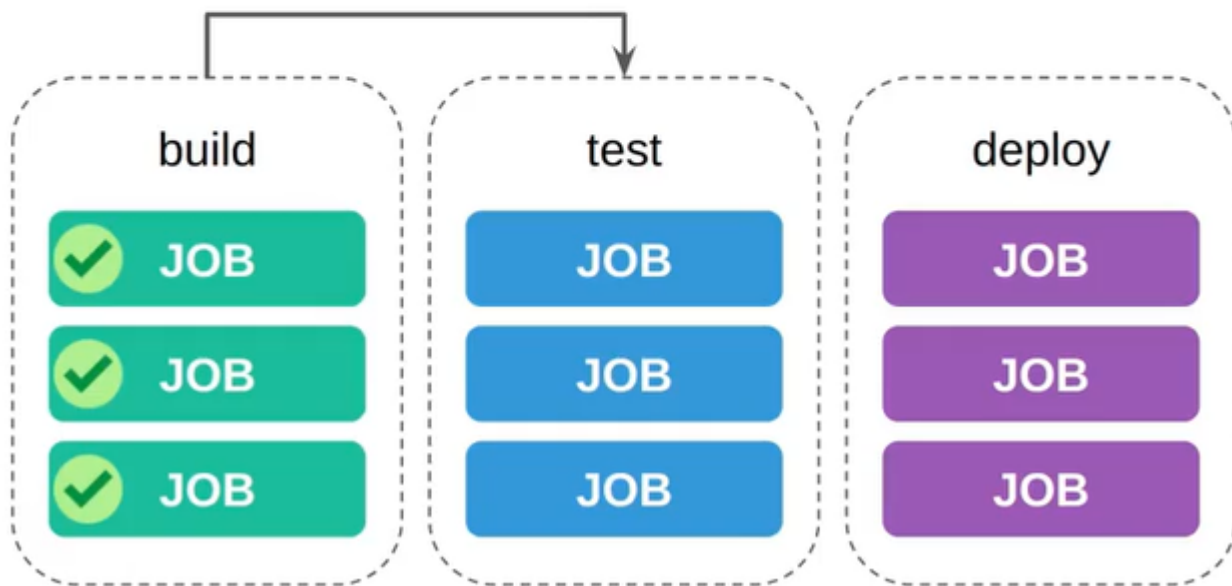


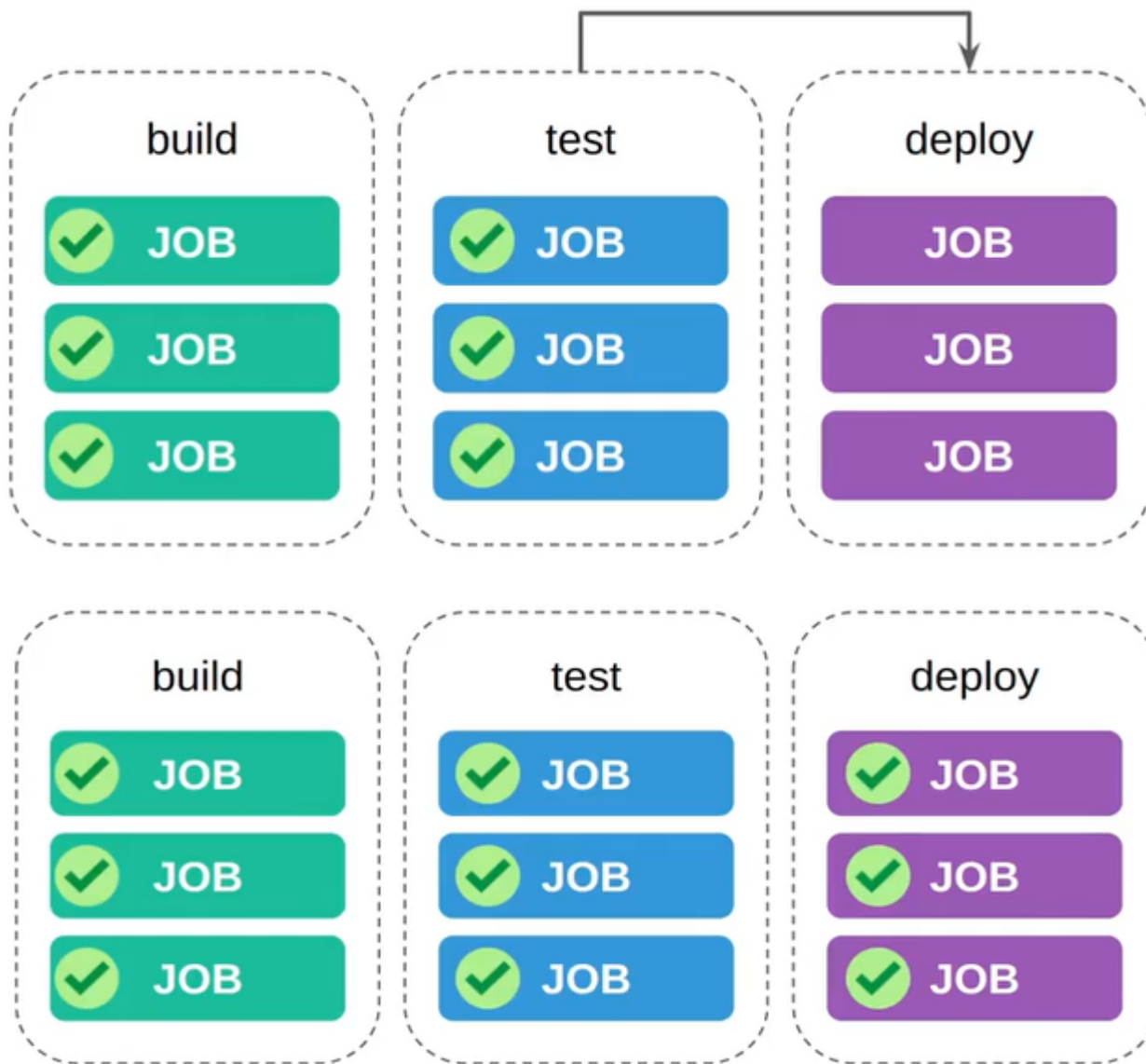


.gitlab-ci.yml

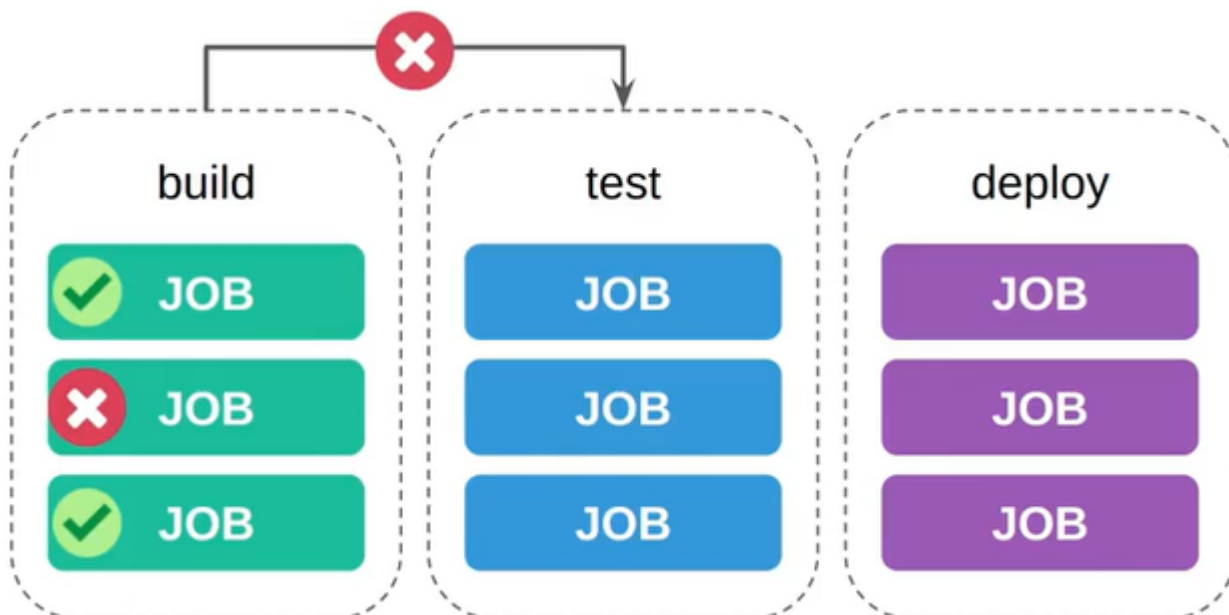
Stages

- build (1)
- test (2)
- deploy (3)
- ... (+1)





Si un job appartenant à un stage échoue alors tout est arrêté on ne passe pas au stage suivant et tout le pipeline s'arrête.



Il est important de signaler que même si vous ne déclarez pas de stages il y a des stage par défaut :



.gitlab-ci.yml

Stages

- .pre
- build
- test
- deploy
- .post

Si vous ne précisez pas pour un Job à quel stage il appartient alors par default il sera exécuté dans le stage **test**.

test

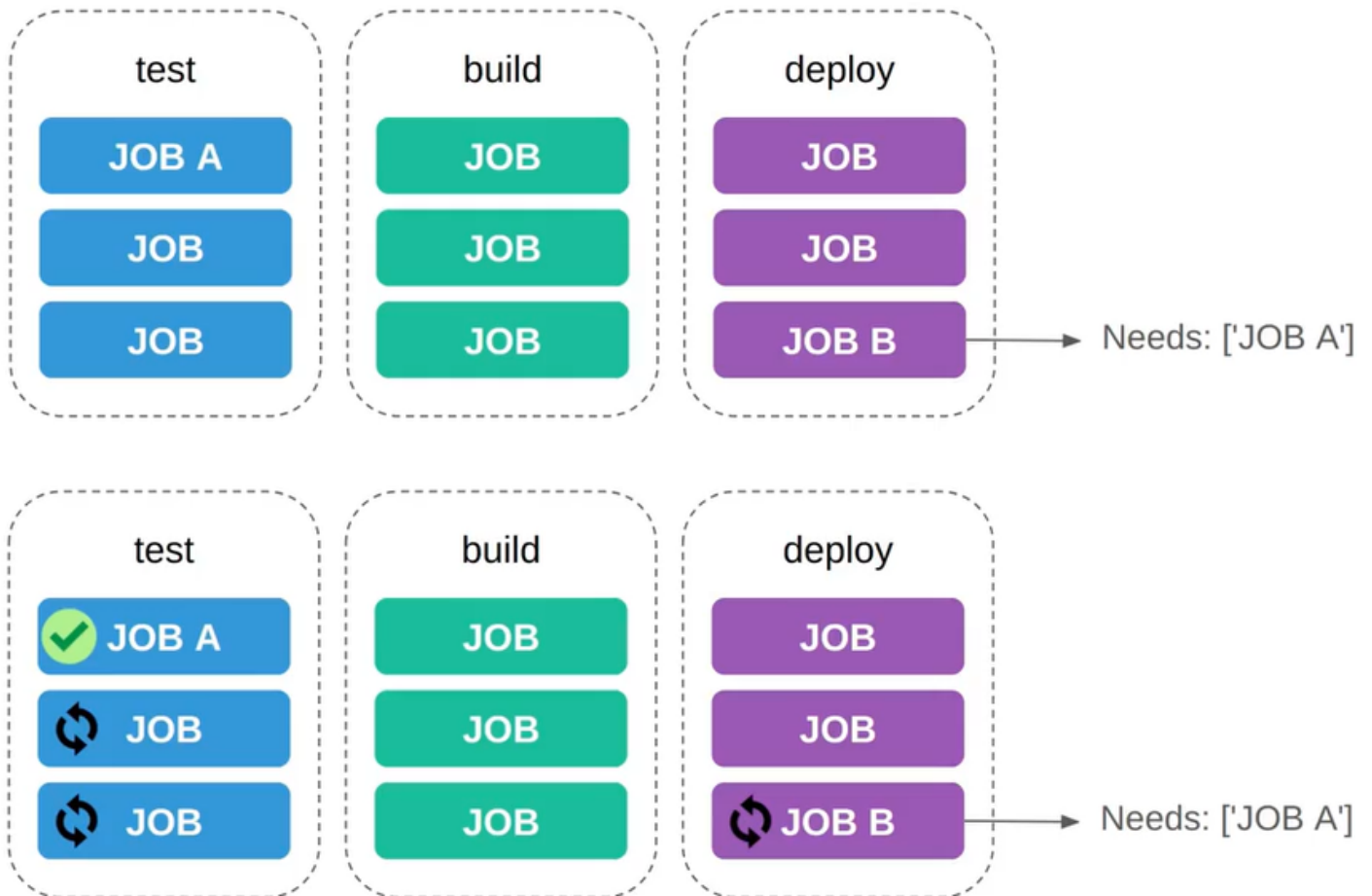
JOB

JOB

JOB

La clé needs sur le Job B permet de lancer le Job B dès que le Job A est terminé.

needs



Retournons à notre projet sur <https://gitlab.com/formation9273638/formation>

formation / formation

F formation

main

formation /

+

History

Find file

Edit

Code

Update .gitlab-ci.yml file

El Hadji Gaye authored 6 days ago

b2a04e25

Name	Last commit	Last update
.gitlab-ci.yml	Update .gitlab-ci.yml file	6 days ago

</> Code

Build

Secure

Deploy

Operate

Monitor

Analyze

Pipelines

Jobs

Pipeline editor

Pipeline schedules

Test cases

Artifacts

Edit

Visualize

Validate

NEW

Full configuration

CI/CD Catalog

Help

14

To contribute improvements to CI/CD templates, please follow the Development guide at:

15

https://docs.gitlab.com/ee/development/cicd/templates.html

16

This specific template is located at:

17

https://gitlab.com/gitlab-org/gitlab/-/blob/master/lib/gitlab/ci/templates/Getting-Started.gitlab-ci.yml

18

19

stages:

20

- build

21

- test

22

- deploy

23

24

build-job:

25

stage: build

26

script:

27

- echo "Compiling the code..."

28

- echo "Compile complete."

29

30

unit-test-job:

31

stage: test

32

script:

33

- echo "Running unit tests... This will take about 60 seconds."

34

- sleep 60

35

- echo "Code coverage is 90%"

43

EL Hadji Gaye

Avec un peu de nettoyage le fichier .gitlab-ci.yml peut devenir :

CI/CD Catalog Help

```
1
2 stages:           # List of stages for jobs, and their order of execution
3   - build
4   - test
5   - deploy
6
7 build-job:        # This job runs in the build stage, which runs first.
8   stage: build
9   script:
10    - echo "Compiling the code..."
11    - echo "Compile complete."
12
13 unit-test-job:    # This job runs in the test stage.
14   stage: test      # It only starts when the job in the build stage completes successfully.
15   script:
16    - echo "Running unit tests..."
17    - echo "Tests successfully..."
18
19 deploy-job:       # This job runs in the deploy stage.
20   stage: deploy    # It only runs when *both* jobs in the test stage complete successfully.
21   script:
22    - echo "Deploying application..."
23    - echo "Application successfully deployed."
```

En version copiable :

```
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 "Application successfully compiled."

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..."
    - echo "Application successfully tested."

deploy-job:       # This job runs in the deploy stage.
  stage: deploy    # It only runs when *both* jobs in the test stage complete successfully.
  script:
    - echo "Deploying application..."
    - echo "Application successfully deployed."
```

Committer les changements.

Commit message

Update .gitlab-ci.yml file

Branch

main

Commit changes

Reset

All 6FinishedBranchesTags

Clear runner caches

New pipeline

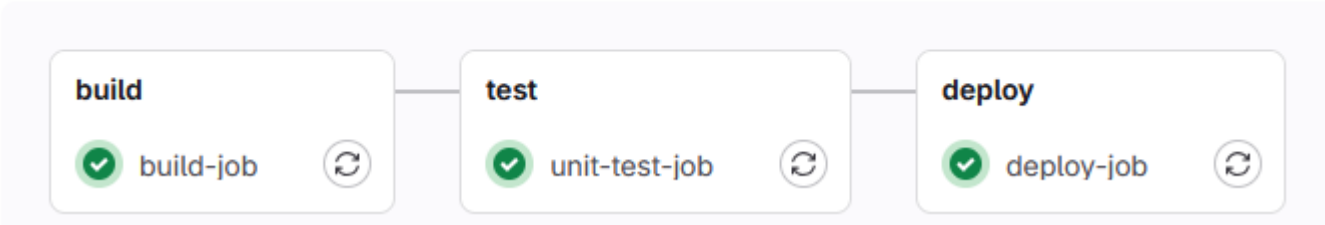
Filter pipelines

Q

Show Pipeline ID ▾

Status	Pipeline	Created by	Stages	
<div>Running</div>	<div>Update .gitlab-ci.yml file</div> <div>#1473546104 main f67384f7 </div> <div>latest</div>		<div> </div>	<div> </div>

Pipeline Jobs 3 Tests 0



En changeant les ordre de de **build** et **test** on obtient pour le fichier **.gitlab-ci.yml** :

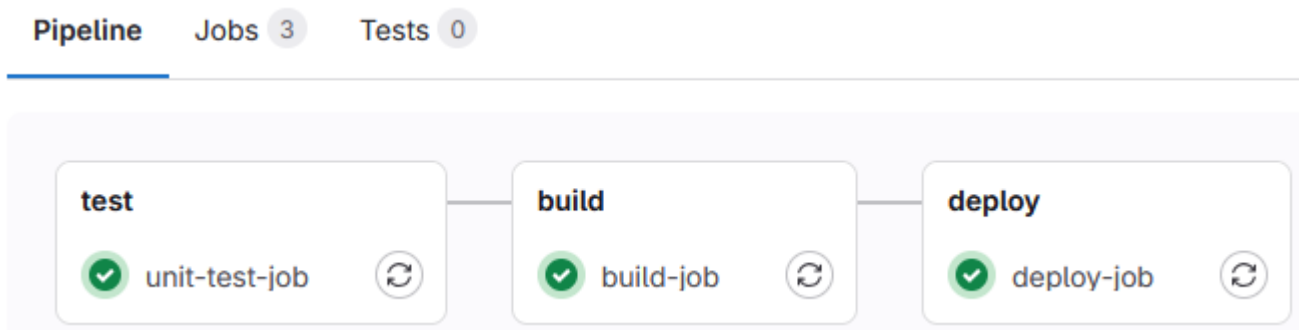
```
stages:      # List of stages for jobs, and their order of execution
- test
- build
- deploy

build-job:   # This job runs in the build stage, which runs first.
  stage: build
  script:
    - echo "Compiling the code..."
    - echo "Application successfully compiled."

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..."
    - echo "Application successfully tested."

deploy-job:  # This job runs in the deploy stage.
  stage: deploy # It only runs when *both* jobs in the test stage complete successfully.
  script:
    - echo "Deploying application..."
    - echo "Application successfully deployed."
```

On obtient maintenant :



La clés **needs** peut se définir de deux manière :

```
other-job:
  stage: other
  needs:
    - build-job
  script:
    - echo "Begining other-job..."
    - echo "other-job successfully executed."
```

Ou encore :

```
other-job:
  stage: other
  needs: [ "build-job" ]
  script:
    - echo "Begining other-job..."
    - echo "other-job successfully executed."
```

Avec la clés needs on peut obtenir le pipeline suivant :

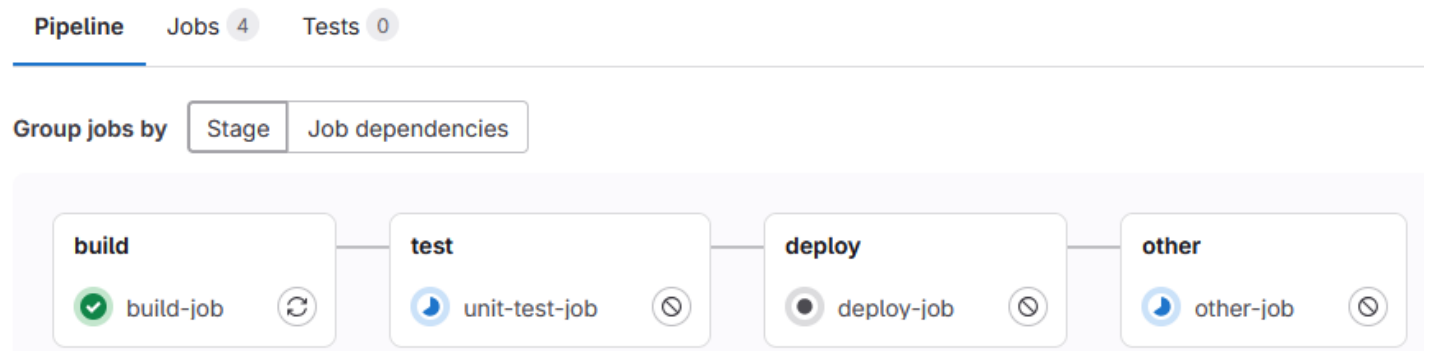
```
stages:
- build
- test
- deploy
- other

build-job:
stage: build
script:
- echo "Compiling the code..."
- echo "Application successfully compiled."

unit-test-job:
stage: test
script:
- echo "Running unit tests..."
- echo "Application successfully tested."

deploy-job:
stage: deploy
script:
- echo "Deploying application..."
- echo "Application successfully deployed."

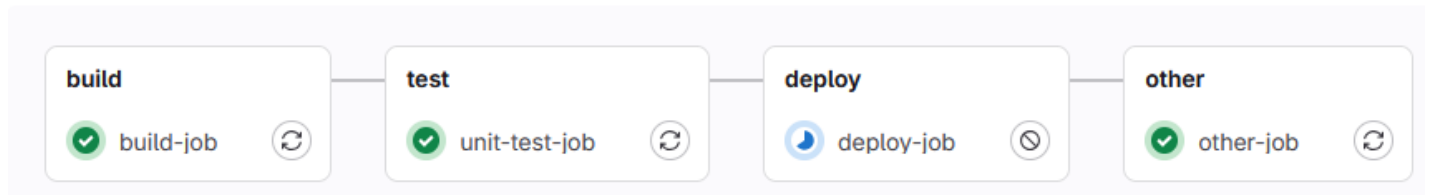
other-job:
stage: other
needs:
- build-job
script:
- echo "Begining other-job..."
- echo "other-job successfully executed."
```



Group jobs by

Stage

Job dependencies



2. Les scripts

a) Définition

Le mot-clé `script` permet de spécifier les commandes qui seront exécutées par le runner GitLab. C'est un élément essentiel dans la définition d'un job.

Il s'agit d'un mot-clé de job. Vous ne pouvez l'utiliser qu'en tant que partie d'un job.

```
job:
  script:
    - echo "Ce texte sera affiché."
```

before_script : préparation avant l'exécution du job

Le mot-clé `before_script` permet de définir une série de commandes qui seront exécutées avant le script principal d'un job.

En isolant certaines commandes de préparation ou de configuration dans un `before_script`, vous rendez le fichier `.gitlab-ci.yml` plus lisible et plus facile à comprendre. Vous pouvez rapidement voir quelles sont les étapes de préparation distinctes des étapes principales du job.

Il s'agit d'un mot-clé de job, utilisable soit dans la section d'un job spécifique soit dans la section par défaut du fichier `.gitlab-ci.yml`.

```
job:
  before_script:
    - echo "Ce texte sera affiché avant les commandes du 'script:' principal."
  script:
    - echo "Ce texte sera affiché après les commandes du 'before_script'."
```

Les scripts de `before_script` sont concaténés aux scripts du script principal. Ils s'exécutent donc ensemble dans un seul et même shell.

after_script : opérations après l'exécution du job

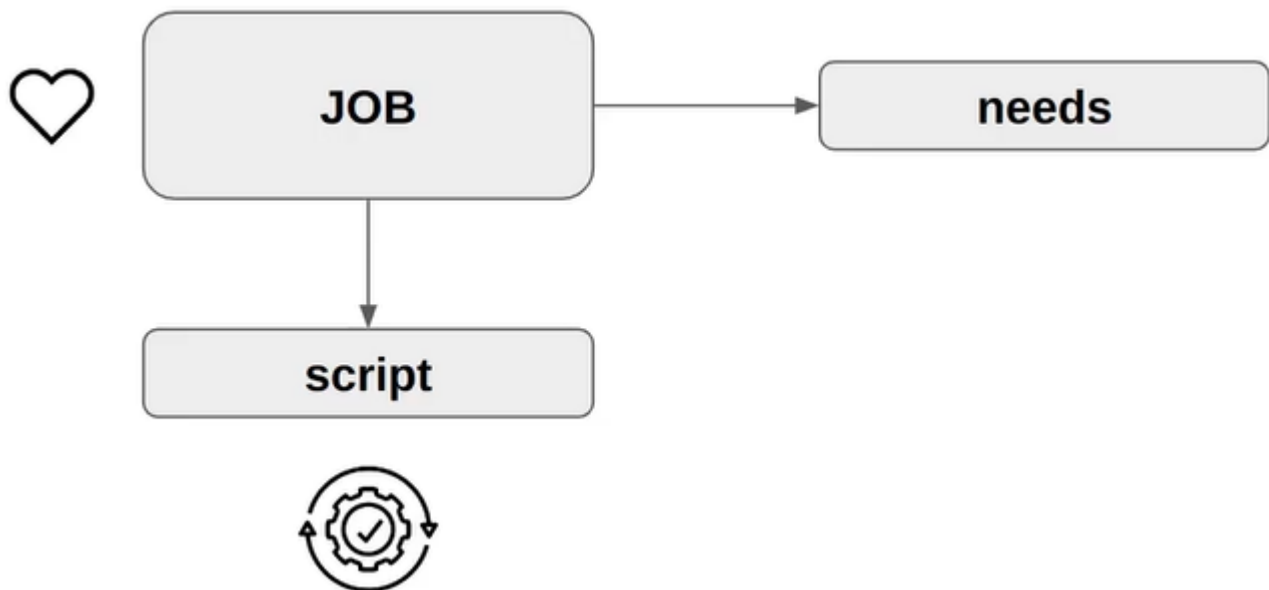
Le mot-clé `after_script` permet de spécifier une série de commandes qui seront exécutées après le script principal, même si ce dernier échoue.

Comme `before_script`, `after_script` est un mot-clé de job.

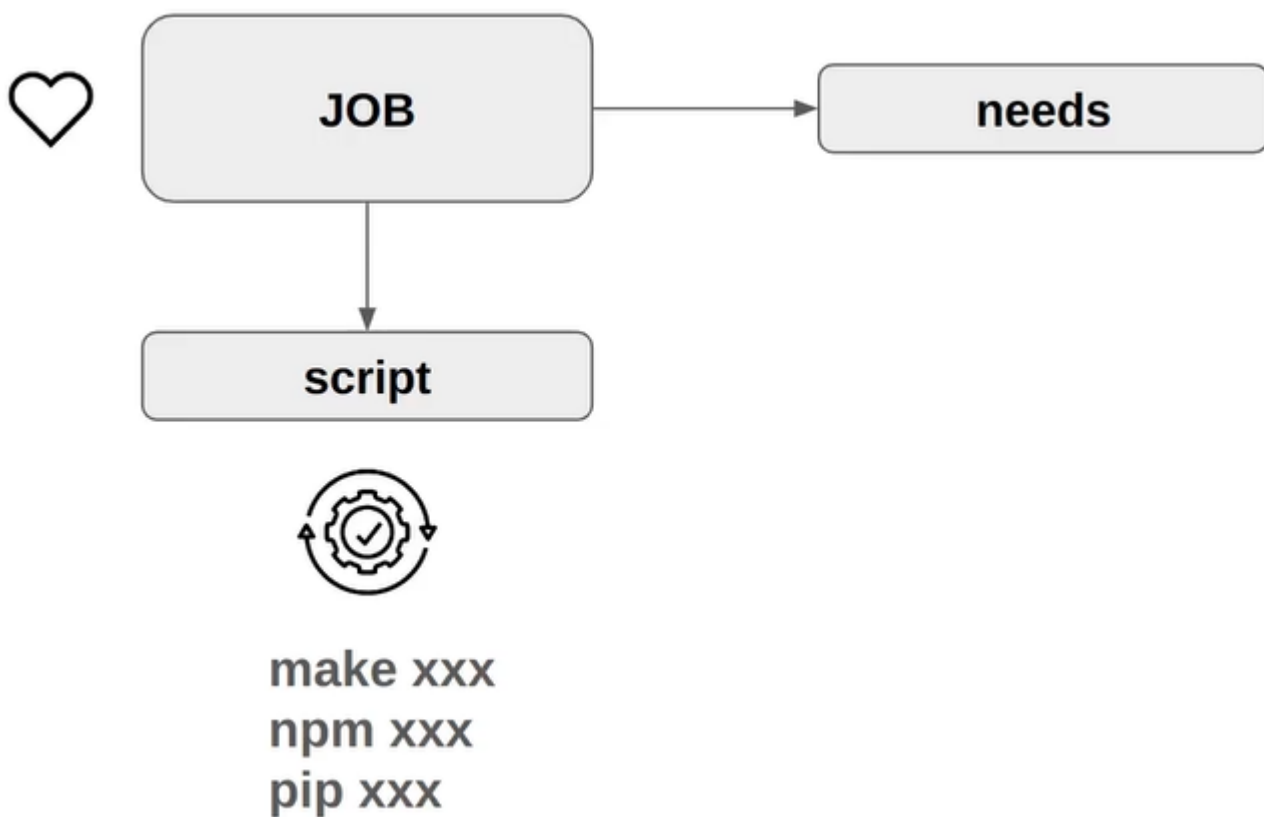
```
job:
  script:
    - echo "Ceci est le 'script' principal."
  after_script:
    - echo "Ceci sera exécuté après le 'script' principal."
```

b) Pratique

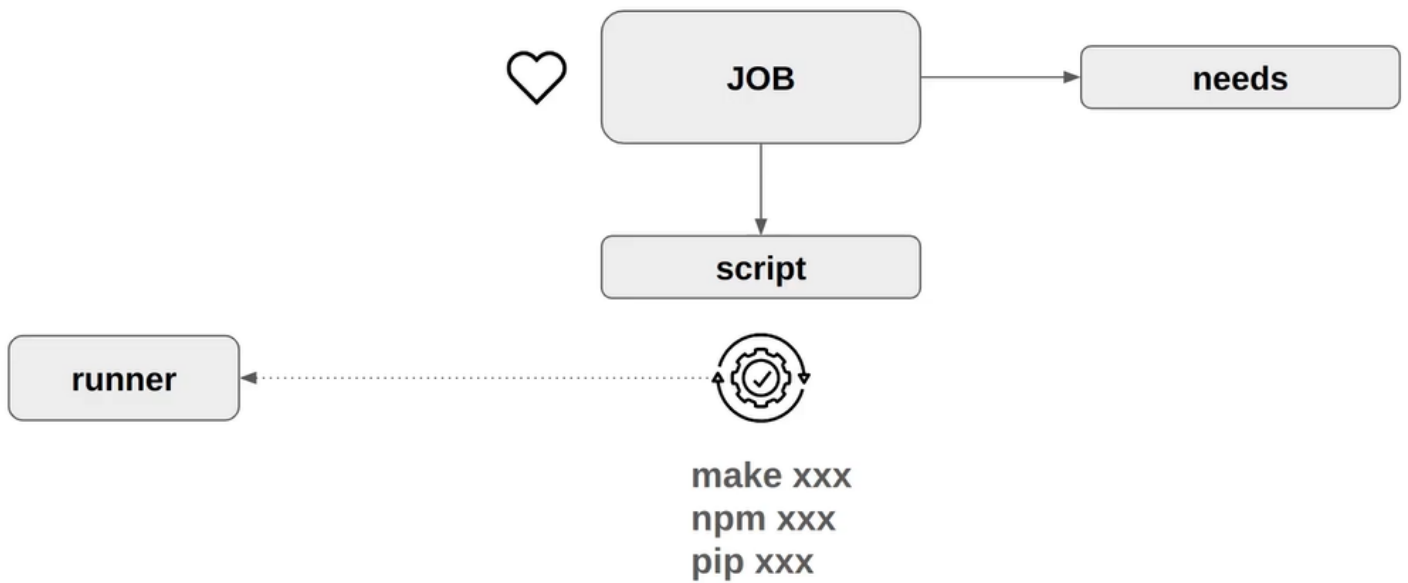
La clé script est la clé la plus importante du Job, elle va décrire l'action du Job.



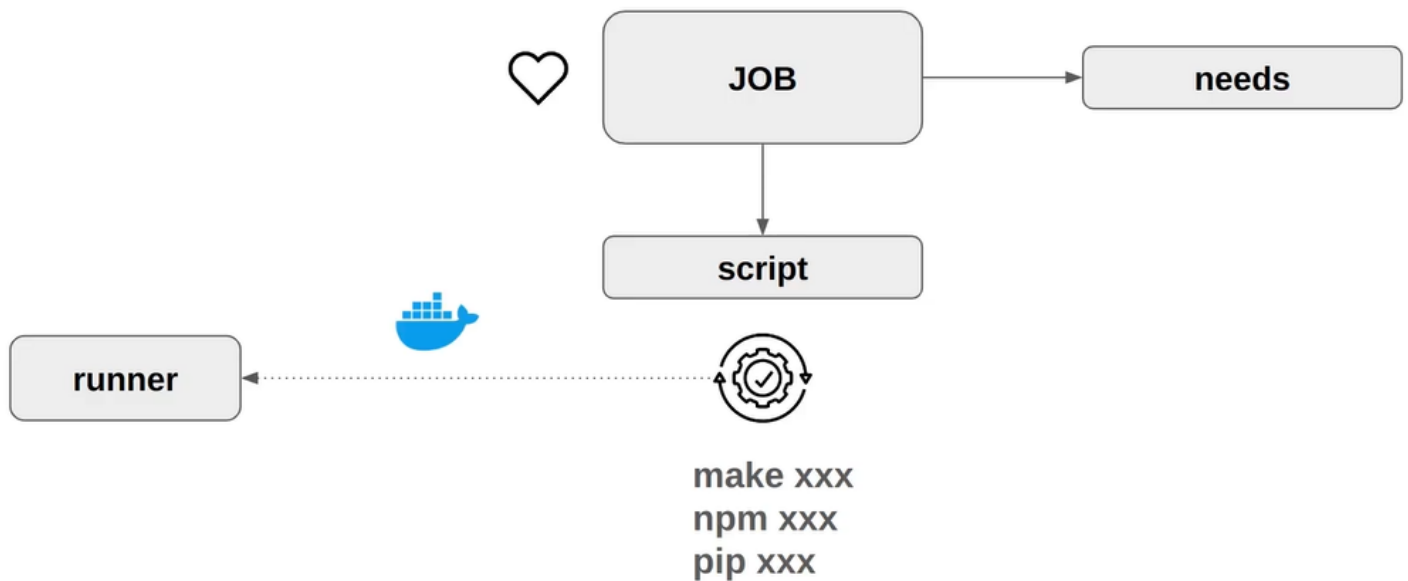
Dans un script on peut executé un certain nombre d'instruction de diverse type.

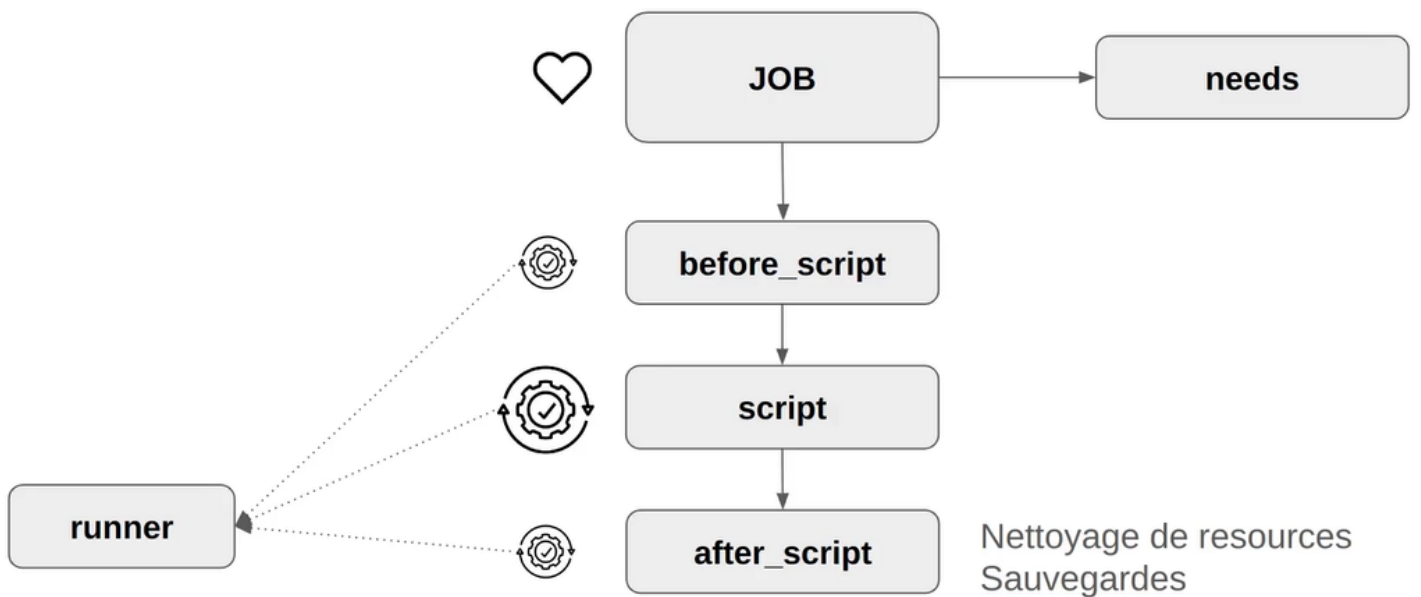
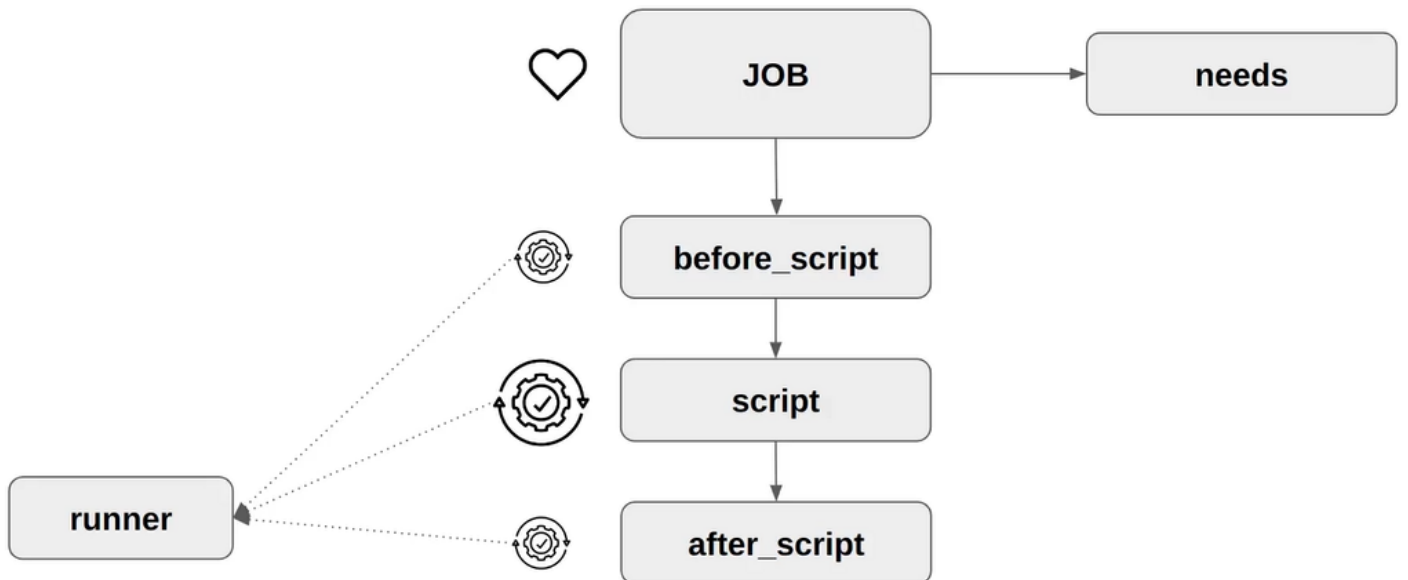
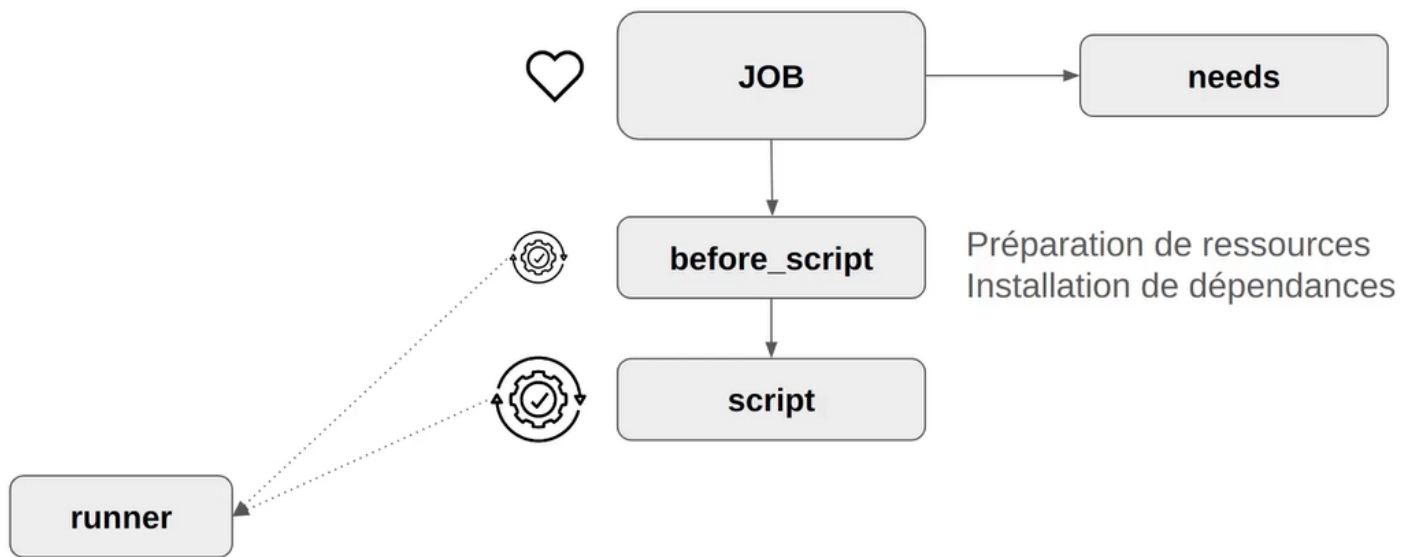


Le job va être positionner sur un Runner (une machine avec un agent GitLab) qui va exécuté le Job.



Attention parfois on aura besoin d'une image docker pour executer certains scripts.

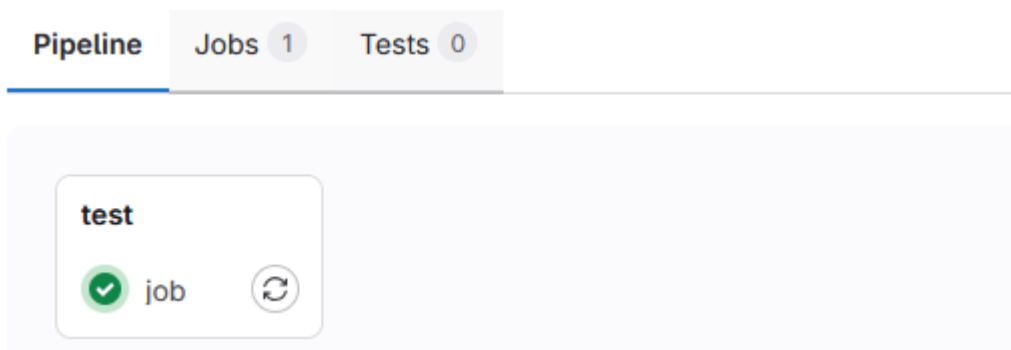




Le fichier `.gitlab-ci.yml` peut alors devenir :

```
job:
  before_script:
    - echo "Begining before_script..."
    - echo "Ending before_script."
  script:
    - echo "Begining script..."
    - echo "Ending script."
  after_script:
    - echo "Begining after_script..."
    - echo "Ending after_script."
```

L'exécution donne donc :



Effectivement comme nous n'avons pas défini de stage pour notre job `job` donc il sera exécuté comme appartenant au stage `test`.

```
1 Running with gitlab-runner 17.4.0-pre.110.g27400594 (27400594)
2   on blue-2.saas-linux-small-amd64.runners-manager.gitlab.com/default XxUrkriX, system ID: s_f46a988edce4
3 Resolving secrets
4 Preparing the "docker+machine" executor
5 Using Docker executor with image ruby:3.1 ...
6 Pulling docker image ruby:3.1 ...
7 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:1cd339d07ba2b9f6b8e61a7a2ca7549e1e6053af7250717d7c94c360be48decf ...
8 Preparing environment
9 Running on runner-xxurkriX-project-61863328-concurrent-0 via runner-xxurkriX-s-l-s-amd64-1727537090-11e54422...
10 Getting source from Git repository
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/formation9273638/formation/.git/
13 Created fresh repository.
14 Checking out 974fcfa8 as detached HEAD (ref is main)...
15 Skipping Git submodules setup
16 $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 Executing "step_script" stage of the job script
18 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:1cd339d07ba2b9f6b8e61a7a2ca7549e1e6053af7250717d7c94c360be48decf ...
19 $ echo "Begining before_script..."
20 Begining before_script...
21 $ echo "Ending before_script."
22 Ending before_script.
23 $ echo "Begining script..."
24 Begining script...
25 $ echo "Ending script."
26 Ending script.
27 Running after_script
28 Running after_script...
29 $ echo "Begining after_script..."
30 Begining after_script...
31 $ echo "Ending after_script."
32 Ending after_script.
33 Cleaning up project directory and file based variables
34 Job succeeded
```

3. Les variables

a) Définition

Les variables dans les pipelines CI/CD de GitLab sont extrêmement utiles pour gérer des paramètres configurables sans avoir à modifier le fichier de configuration `.gitlab-ci.yml` à chaque fois.

Ces variables peuvent être utilisées dans différentes parties du pipeline, notamment dans les scripts de job, et peuvent être définies soit au niveau global, soit au niveau du job.

Le nom peut utiliser seulement des chiffres, des lettres et des traits de soulignement (`_`).

Les variables peuvent être définies dans le fichier `.gitlab-ci.yml` ou dans l'UI de Gitlab. Il existe aussi un grand nombre de variables prédéfinies.

Types de Variables dans le fichier `.gitlab-ci.yml`

Variables globales

Définies une fois pour tous les jobs dans le fichier `.gitlab-ci.yml`. Elles agissent comme des valeurs par défaut et sont écrasées si une variable du même nom est définie au niveau du job.

```
variables:  
  DEPLOY_SITE: "https://example.com/"
```

Variables au niveau du Job

Spécifiques à un job particulier et écrasent les variables globales si elles portent le même nom.

```
deploy_review_job:  
  stage: deploy  
  variables:  
    REVIEW_PATH: "/review"
```

Utilisation des variables dans des Jobs

Les variables peuvent être utilisées dans les scripts `script`, `before_script`, et `after_script` et certains autres mots-clés de job comme `rules`.

```
deploy_job:  
  stage: deploy  
  script:  
    - deploy-script --url $DEPLOY_SITE --path "/"
```

Variables Prédéfinies

GitLab fournit un ensemble de variables prédéfinies accessibles dans tous les jobs. Voici quelques variables principales :

- `CI_COMMIT_SHA` : le SHA du commit
- `CI_COMMIT_MESSAGE` : le message du commit
- `CI_COMMIT_REF_NAME` : le nom de la branche ou de l'étiquette
- `CI_PROJECT_ID` : l'ID du projet
- `CI_PIPELINE_ID` : l'ID du pipeline
- `CI_JOB_ID` : l'ID du job

Pour accéder à la liste complète, cliquer ici

https://docs.gitlab.com/ee/ci/variables/predefined_variables.html

Variables sur l'UI Gitlab

Ces variables permettent de stocker des informations sensibles comme les clés d'API, les tokens, ou les mots de passe, en toute sécurité. Cette leçon vous expliquera comment définir et gérer ces variables au niveau d'un projet, d'un groupe et de toute une instance GitLab.

Définir une variable CI/CD au niveau d'un projet

- 1) Accéder aux paramètres (Settings) : connectez-vous à votre projet GitLab et allez dans "Settings" puis "CI/CD". Vous trouverez une section nommée "Variables".
- 2) Ajouter une variable : cliquez sur le bouton "Add variable". Un formulaire s'affiche.
 - Clé : la clé doit être unique, en une seule ligne, sans espaces et composée uniquement de lettres, de chiffres ou de caractères de soulignement (`_`). C'est le nom de la variable.
 - Valeur : insérez la valeur de la variable. Il n'y a pas de limitation sur le type ou le nombre de caractères.
 - Type : choisissez entre "Variable" ou "File".
 - Portée de l'environnement : vous pouvez spécifier si cette variable doit être disponible pour tous les environnements ou seulement pour certains.
 - Protéger la variable : si cette option est cochée, la variable ne sera disponible que pour les pipelines qui s'exécutent sur des branches ou des tags protégés (nous y reviendrons).
 - Masquer la variable : si cette option est activée, la valeur de la variable sera masquée dans les journaux de travail (logs).

- 3) Utilisation dans `.gitlab-ci.yml` : une fois la variable créée, elle peut être utilisée dans le fichier de configuration `.gitlab-ci.yml` ou dans des scripts de travail (job scripts) en utilisant son nom de clé.

Définir une variable CI/CD au niveau d'un groupe

- 1) Accès aux Paramètres du groupe : dans le groupe GitLab concerné, naviguez vers "Settings" puis "CI/CD".
- 2) Ajouter une variable : le processus est identique à celui du niveau projet, avec la différence que cette variable sera disponible pour tous les projets du groupe.

Définir une variable CI/CD au niveau d'une instance

- 1) Accès à la zone d'administration : sur la barre latérale gauche, cliquez sur "Rechercher" ou accédez directement à la "Zone d'administration".
- 2) Ajouter une variable : tout comme pour le projet et le groupe, vous pouvez ajouter une variable qui sera disponible pour tous les projets et groupes de cette instance GitLab.

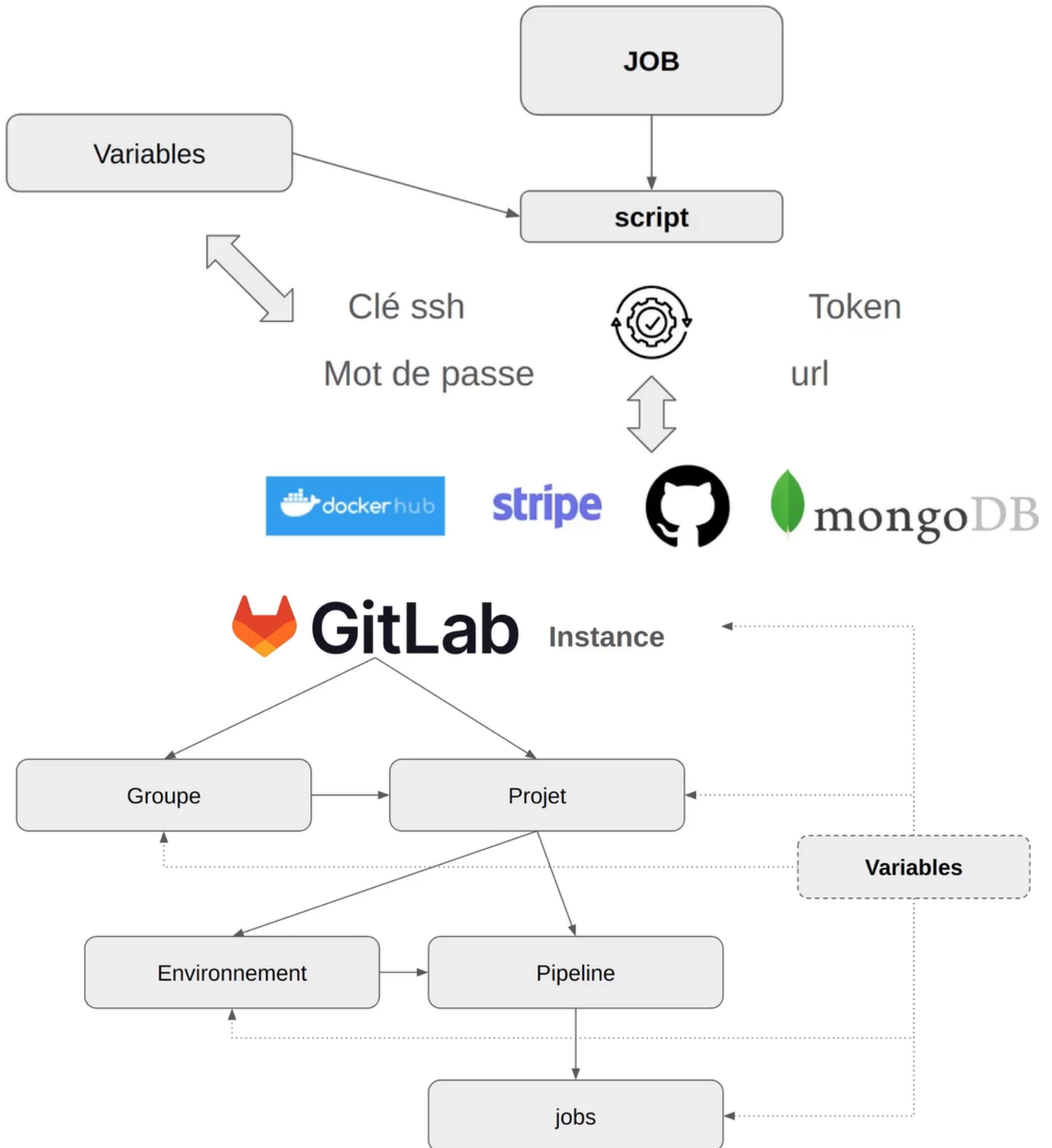
Masquage et protection de variables

Masquage : vous pouvez masquer une variable pour que sa valeur ne soit pas visible dans les logs. Cependant, ce n'est pas une méthode infaillible.

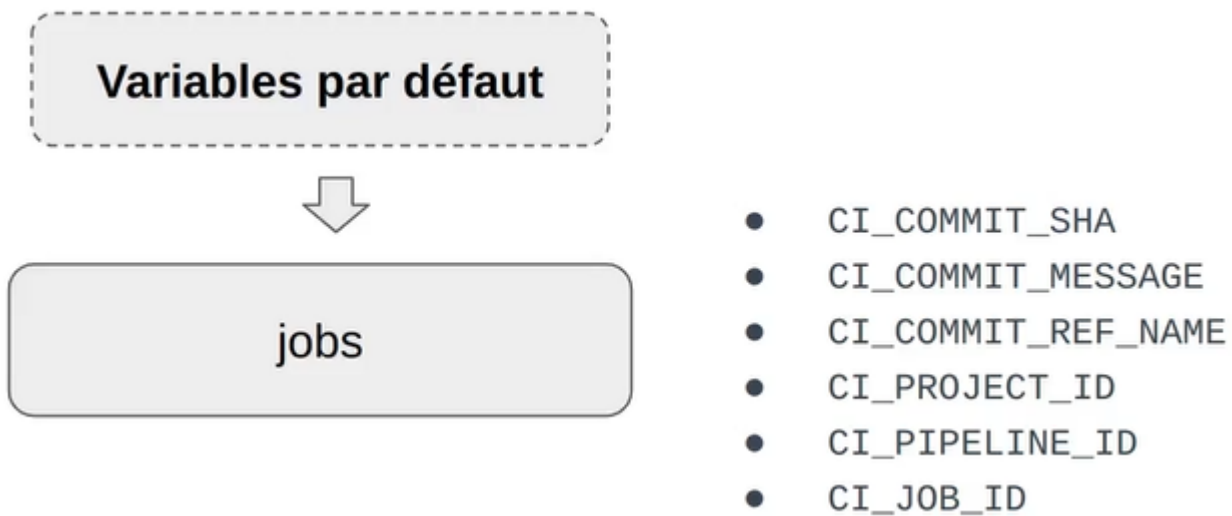
Protection: vous pouvez également protéger une variable pour qu'elle ne soit accessible que dans les pipelines qui s'exécutent sur des branches ou des tags protégés.

b) Pratique

Les variables sont indispensables pour échanger des informations entre les jobs GitLab mais aussi échanger des informations entre GitLab et d'autres composants d'une manière générale. Nous allons pouvoir sécuriser ces variables lorsqu'elles sont sensibles (exemple mot de passe).

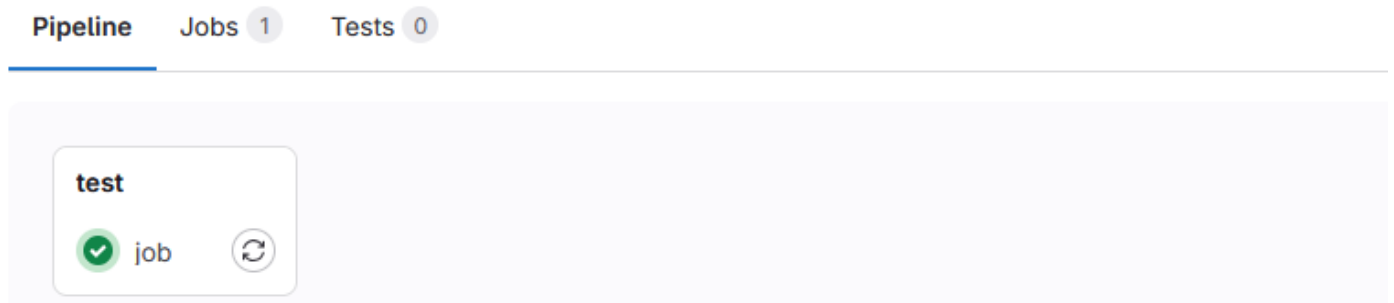


GitLab CI a une liste de variables predefinie, vous trouverez ci-dessous une petite liste :



Le fichier **.gitlab-ci.yml** devient :

```
variables:
  GLOBAL_VARIABLE_FROM_PIPELINE: global_variable_from_pipeline
job:
  variables:
    JOB_VARIABLE_JOB: job_variable_job
  script:
    - echo GLOBAL_VARIABLE_FROM_PIPELINE = $GLOBAL_VARIABLE_FROM_PIPELINE
    - echo JOB_VARIABLE_JOB = $JOB_VARIABLE_JOB
    - echo CI_COMMIT_SHA = $CI_COMMIT_SHA
    - echo CI_COMMIT_MESSAGE = $CI_COMMIT_MESSAGE
    - echo CI_COMMIT_REF_NAME = $CI_COMMIT_REF_NAME
    - echo CI_PROJECT_ID = $CI_PROJECT_ID
    - echo CI_PIPELINE_ID = $CI_PIPELINE_ID
    - echo CI_JOB_ID = $CI_JOB_ID
```



On obtient à l'exécution du pipeline :

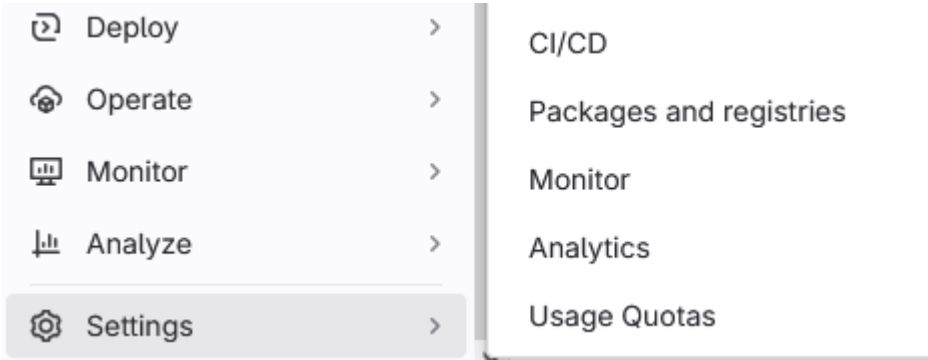
job

✓ Passed Started just now by  El Hadji Gaye

Search visible log output 🔍 📄 ↺ ⬆ ⬇ ↗

```
1 Running with gitlab-runner 17.4.0~pre.110.g27400594 (27400594)
2   on blue-2.saas-linux-small-amd64.runners-manager.gitlab.com/default XxUrkriX, system ID: s_f46a98edce4
3 Resolving secrets
4 Preparing the "docker+machine" executor 00:22
5 Using Docker executor with image ruby:3.1 ...
6 Pulling docker image ruby:3.1 ...
7 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:b7fe909968d1e473c5448ee2558
  75bdb65c67df0efe28a0991f97a91ce2e71e7 ...
8 Preparing environment 00:05
9 Running on runner-xxurkrix-project-61863328-concurrent-0 via runner-xxurkrix-s-l-s-amd64-1727558669-8230beee...
10 Getting source from Git repository 00:01
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/formation9273638/formation/.git/
13 Created fresh repository.
14 Checking out 4b5a6383 as detached HEAD (ref is main)...
15 Skipping Git submodules setup
16 $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 Executing "step_script" stage of the job script 00:01
18 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:b7fe909968d1e473c5448ee2558
  75bdb65c67df0efe28a0991f97a91ce2e71e7 ...
19 $ echo GLOBAL_VARIABLE_FROM_PIPELINE = $GLOBAL_VARIABLE_FROM_PIPELINE
20 GLOBAL_VARIABLE_FROM_PIPELINE = global_variable_from_pipeline
21 $ echo JOB_VARIABLE_JOB = $JOB_VARIABLE_JOB
22 JOB_VARIABLE_JOB = job_variable_job
23 $ echo CI_COMMIT_SHA = $CI_COMMIT_SHA
24 CI_COMMIT_SHA = 4b5a6383b9a423fc6ce1a859cefe5d956dbf53c5
25 $ echo CI_COMMIT_MESSAGE = $CI_COMMIT_MESSAGE
26 CI_COMMIT_MESSAGE = Update .gitlab-ci.yml file
27 $ echo CI_COMMIT_REF_NAME = $CI_COMMIT_REF_NAME
28 CI_COMMIT_REF_NAME = main
29 $ echo CI_PROJECT_ID = $CI_PROJECT_ID
30 CI_PROJECT_ID = 61863328
31 $ echo CI_PIPELINE_ID = $CI_PIPELINE_ID
32 CI_PIPELINE_ID = 1473773337
33 $ echo CI_JOB_ID = $CI_JOB_ID
34 CI_JOB_ID = 7945586687
35 Cleaning up project directory and file based variables 00:00
36 Job succeeded
```

Pour définir des variables CI/CD via GitLab il faut se rendre sur **Settings → CI/CD**



Artifacts

Expand

A job artifact is an archive of files and directories saved by a job when it finishes.

Variables

Collapse

Variables store information that you can use in job scripts. Each project can define a maximum of 8000 variables. [Learn more.](#)

Variables can be accidentally exposed in a job log, or maliciously sent to a third party server. The masked variable feature can help reduce the risk of accidentally exposing variable values, but is not a guaranteed method to prevent malicious users from accessing variables. [How can I make my variables more secure?](#)

Variables can have several attributes. [Learn more.](#)

- **Visibility:** Set the visibility level for the value. Can be visible, masked, or masked and hidden.
- **Flags**
 - **Protected:** Only exposed to protected branches or protected tags.
 - **Expanded:** Variables with `$` will be treated as the start of a reference to another variable.

CI/CD Variables </> 0			Add variable
Key ↑	Value	Environments	Actions
There are no variables yet.			

Group variables (inherited)

These variables are inherited from the parent group.

CI/CD Variables </> 0		
Key	Environments	Group
There are no variables yet.		

CI/CD Variables </> 0			<button>Add variable</button>
Key ↑	Value	Environments	Actions
There are no variables yet.			

Créer les variables **FROM_GITLAB_VISIBLE** et **FROM_GITLAB_HIDDEN**.

Add variable

×

Type

Variable (default) ▾

Environments ⓘ

All (default) ▾

Visibility

☒ Visible
Can be seen in job logs.

Key

FROM_GITLAB_VISIBLE

You can use CI/CD variables with the same name in different places, but the variables might overwrite each other. [What is the order of precedence for variables?](#)

Value

from_gitlab_visible

Variable value will be evaluated as raw string.

Add variable

Cancel

Add variable



Type

Variable (default)



Environments

All (default)



Visibility

☐ Visible

Can be seen in job logs.

☒ Masked

Masked in job logs but value can be revealed in CI/CD settings. Requires values to meet regular expressions requirements.

Key

FROM_GITLAB_HIDDEN

You can use CI/CD variables with the same name in different places, but the variables might overwrite each other. [What is the order of precedence for variables?](#)

Value











from_gitlab_hidden

Variable value will be evaluated as raw string.

Add variable

Cancel

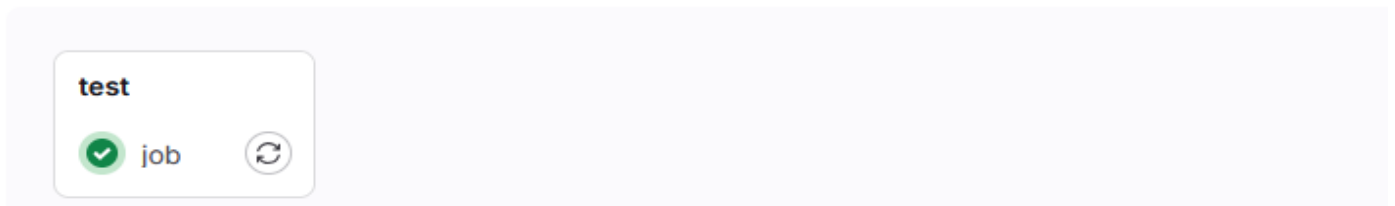
On obtient comme resultat :

CI/CD Variables </> 2				Reveal values	Add variable
Key ↑	Value	Environments	Actions		
FROM_GITLAB_HIDDEN  Masked	***** 	All (default) 	 		
FROM_GITLAB_VISIBLE 	***** 	All (default) 	 		

Le fichier **.gitlab-ci.yml** devient :

```
variables:
  GLOBAL_VARIABLE_FROM_PIPELINE: global_variable_from_pipeline
job:
  variables:
    JOB_VARIABLE_JOB: job_variable_job
  script:
    - echo GLOBAL_VARIABLE_FROM_PIPELINE = $GLOBAL_VARIABLE_FROM_PIPELINE
    - echo JOB_VARIABLE_JOB = $JOB_VARIABLE_JOB
    - echo CI_COMMIT_SHA = $CI_COMMIT_SHA
    - echo CI_COMMIT_MESSAGE = $CI_COMMIT_MESSAGE
    - echo CI_COMMIT_REF_NAME = $CI_COMMIT_REF_NAME
    - echo CI_PROJECT_ID = $CI_PROJECT_ID
    - echo CI_PIPELINE_ID = $CI_PIPELINE_ID
    - echo CI_JOB_ID = $CI_JOB_ID
    - echo FROM_GITLAB_VISIBLE = $FROM_GITLAB_VISIBLE
    - echo FROM_GITLAB_HIDDEN = $FROM_GITLAB_HIDDEN
```

Pipeline Jobs 1 Tests 0



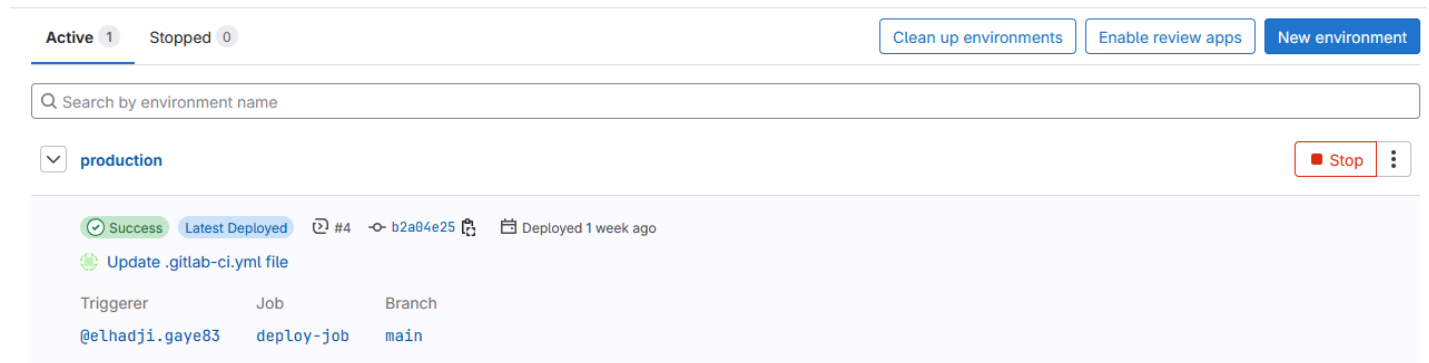
```
1 Running with gitlab-runner 17.4.0-pre.110.g27400594 (27400594)
2   on blue-4.saas-linux-small-amd64.runners-manager.gitlab.com/default J2nyww-s, system ID: s_cf1798852952
3 Resolving secrets
4 Preparing the "docker+machine" executor
5 Using Docker executor with image ruby:3.1 ...
6 Pulling docker image ruby:3.1 ...
7 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:b7fe909968d1e473c5448ee2558
75bdb65c67df0efe28a0991f97a91ce2e71e7 ...
8 Preparing environment
9 Running on runner-j2nyww-s-project-61863328-concurrent-0 via runner-j2nyww-s-s-l-s-amd64-1727560907-f38f3e13...
10 Getting source from Git repository
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/formation9273638/formation/.git/
13 Created fresh repository.
14 Checking out 658fafc7 as detached HEAD (ref is main)...
15 Skipping Git submodules setup
16 $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 Executing "step_script" stage of the job script
18 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:b7fe909968d1e473c5448ee2558
75bdb65c67df0efe28a0991f97a91ce2e71e7 ...
19 $ echo GLOBAL_VARIABLE_FROM_PIPELINE = $GLOBAL_VARIABLE_FROM_PIPELINE
20 GLOBAL_VARIABLE_FROM_PIPELINE = global_variable_from_pipeline
21 $ echo JOB_VARIABLE_JOB = $JOB_VARIABLE_JOB
22 JOB_VARIABLE_JOB = job_variable_job
23 $ echo CI_COMMIT_SHA = $CI_COMMIT_SHA
24 CI_COMMIT_SHA = 658fafc70e6aecffd2cacfa30d799fcb34635e43
25 $ echo CI_COMMIT_MESSAGE = $CI_COMMIT_MESSAGE
26 CI_COMMIT_MESSAGE = Update .gitlab-ci.yml file
27 $ echo CI_COMMIT_REF_NAME = $CI_COMMIT_REF_NAME
28 CI_COMMIT_REF_NAME = main
29 $ echo CI_PROJECT_ID = $CI_PROJECT_ID
30 CI_PROJECT_ID = 61863328
31 $ echo CI_PIPELINE_ID = $CI_PIPELINE_ID
32 CI_PIPELINE_ID = 1473785977
33 $ echo CI_JOB_ID = $CI_JOB_ID
34 CI_JOB_ID = 7945639438
35 $ echo FROM_GITLAB_VISIBLE = $FROM_GITLAB_VISIBLE
36 FROM_GITLAB_VISIBLE = from_gitlab_visible
37 $ echo FROM_GITLAB_HIDDEN = $FROM_GITLAB_HIDDEN
38 FROM_GITLAB_HIDDEN = [MASKED]
39 Cleaning up project directory and file based variables
40 Job succeeded
```


4. Les environnements

a) Définition

Les environnements dans GitLab CI/CD sont extrêmement utiles pour gérer les déploiements de votre application à différentes étapes, comme le développement, les tests, les environnements de contrôle qualité (QA), la pré-prod (staging), et la production.

Sur Gitlab les environnements sont présents dans Operate > Environments.



Vous y trouverez beaucoup d'informations sur le dernier déploiement dans chaque environnement (date, commit, Job, branche, déclencheur etc) et si le dernier Job de déploiement a réussi.

Syntaxe de base

```
deploy to production:
  stage: deploy
  script: git push production HEAD:main
  environment: production
```

Utilisation d'un nom et d'une URL

Vous pouvez aussi passer des clés / valeurs pour distinguer plus finement vos environnements :

```
deploy to production:
  stage: deploy
  script: git push production HEAD:main
  environment:
    name: production
    url: https://prod.example.com
```

Actions d'environnement

Vous pouvez spécifier comment le job interagit avec l'environnement. Les valeurs possibles sont: start, prepare, stop, verify, et access.

```
stop_review_app:
  stage: deploy
  script: make delete-app
  environment:
    name: review/$CI_COMMIT_REF_SLUG
  action: stop
```

Tier de déploiement

Cela permet de spécifier le tier de l'environnement de déploiement. Les valeurs possibles sont production, staging, testing, development, et other.

```
deploy:
  script: echo
  environment:
    name: customer-portal
    deployment_tier: production
```

Environnements dynamiques

Vous pouvez utiliser des variables CI/CD pour nommer dynamiquement des environnements.

```
deploy as review app:
  stage: deploy
  script: make deploy
  environment:
    name: review/$CI_COMMIT_REF_SLUG
    url: https://$CI_ENVIRONMENT_SLUG.example.com/
```

Exemples

Job de déploiement en pré-production

```
deploy_to_staging:
  script: ./deploy
  environment:
    name: staging
    url: https://staging.example.com
```

Dans cet exemple, un job appelé `deploy_to_staging` est défini. Lorsque ce job est exécuté, le code est déployé dans l'environnement "staging", accessible via <https://staging.example.com>.

Utilisation d'une action stop

Voici un exemple qui utilise une action stop :

```
stages:
- deploy
- cleanup

deploy_app:
  stage: deploy
  script:
    - echo "Déploiement de l'application..."
    # Commandes pour déployer l'application
  on_stop: stop_app

stop_app:
  stage: cleanup
  when: manual
  action: stop
  script:
    - echo "Arrêt de l'application..."
    # Commandes pour arrêter l'application
  allow_failure: true
```

- stages : deux étapes (stages) sont définies : deploy pour le déploiement et cleanup pour le nettoyage.
- deploy_app : ce job déploie l'application. Il est placé dans le stage deploy.
- on_stop : cette directive indique quel job exécuter lorsqu'un pipeline est arrêté. Dans ce cas, stop_app est le job qui sera exécuté.
- stop_app : ce job est destiné à être exécuté lorsque le pipeline est arrêté manuellement. Il utilise action: stop pour indiquer qu'il doit être exécuté en réponse à l'arrêt du job deploy_app.
- when: manual : cette option signifie que le job stop_app doit être exécuté manuellement. Cela peut être utile pour les situations où tu veux avoir le contrôle sur quand arrêter l'application, par exemple, pour le débogage ou la maintenance.
- allow_failure: true : cette option indique que l'échec de ce job ne causera pas l'échec du pipeline. Cela peut être utile pour les opérations de nettoyage ou de récupération qui ne sont pas critiques pour le succès global du pipeline.

Utilisation d'une action verify

Dans cet exemple, nous allons configurer un pipeline qui comprend un job de vérification après le déploiement de l'application. Ce job sera exécuté automatiquement pour vérifier que le déploiement s'est bien passé.

```
stages:
  - deploy
  - verify

deploy_app:
  stage: deploy
  script:
    - echo "Déploiement de l'application..."
    # Commandes pour déployer l'application

verify_deployment:
  stage: verify
  action: verify
  script:
    - echo "Vérification du déploiement..."
    # Commandes pour vérifier le déploiement
  allow_failure: false
```

- stages : deux étapes (stages) sont définies : deploy pour le déploiement et verify pour la vérification post-déploiement.
- deploy_app : ce job déploie l'application. Il est placé dans le stage deploy.
- verify_deployment : ce job est destiné à vérifier le déploiement. Il utilise action: verify pour indiquer qu'il s'agit d'un job de vérification qui suit le déploiement.
- script : les commandes nécessaires pour vérifier le déploiement de l'application. Cela pourrait inclure des tests de santé (health checks) comme des appels d'API pour s'assurer que l'application répond correctement, etc.
- allow_failure: false : indique que si ce job échoue, le pipeline sera considéré comme ayant échoué. Cela souligne l'importance de la vérification post-déploiement.

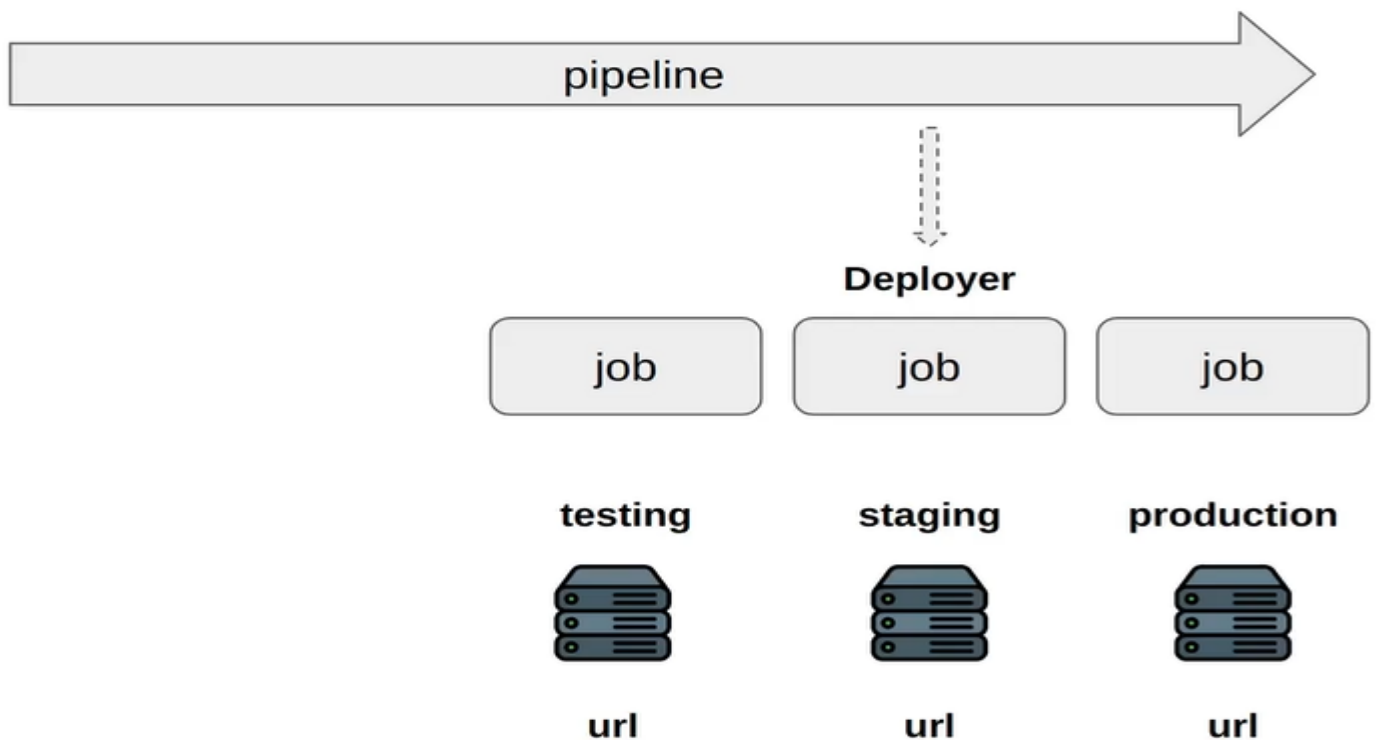
Sur l'interface utilisateur (UI) de GitLab :

- Visualisation du Pipeline : lorsque le pipeline est exécuté, tu verras les deux stages (deploy et verify) listés dans l'ordre. Chaque stage aura son propre job correspondant (deploy_app et verify_deployment).
- État du job : chaque job affichera son état (en cours, réussi, échoué). Si deploy_app réussit, verify_deployment sera exécuté automatiquement après.
- Détails du job : en cliquant sur chaque job, tu pourras voir les détails de l'exécution, y compris les logs de sortie qui montrent ce qui s'est passé pendant l'exécution du script.

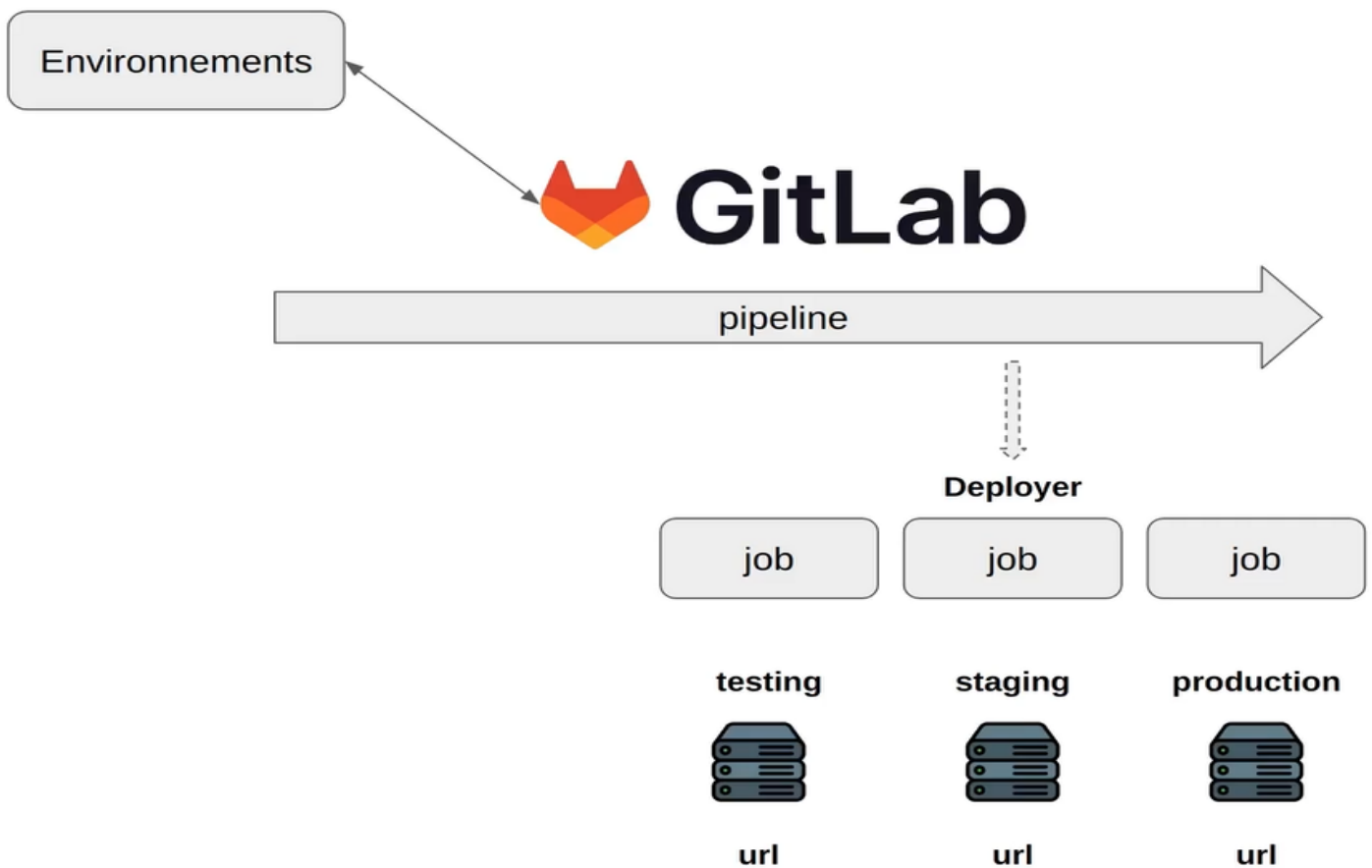
- État du pipeline : si `verify_deployment` échoue (et `allow_failure` est défini sur `false`), le pipeline sera marqué comme échoué, attirant l'attention sur le fait que quelque chose ne va pas avec le déploiement.

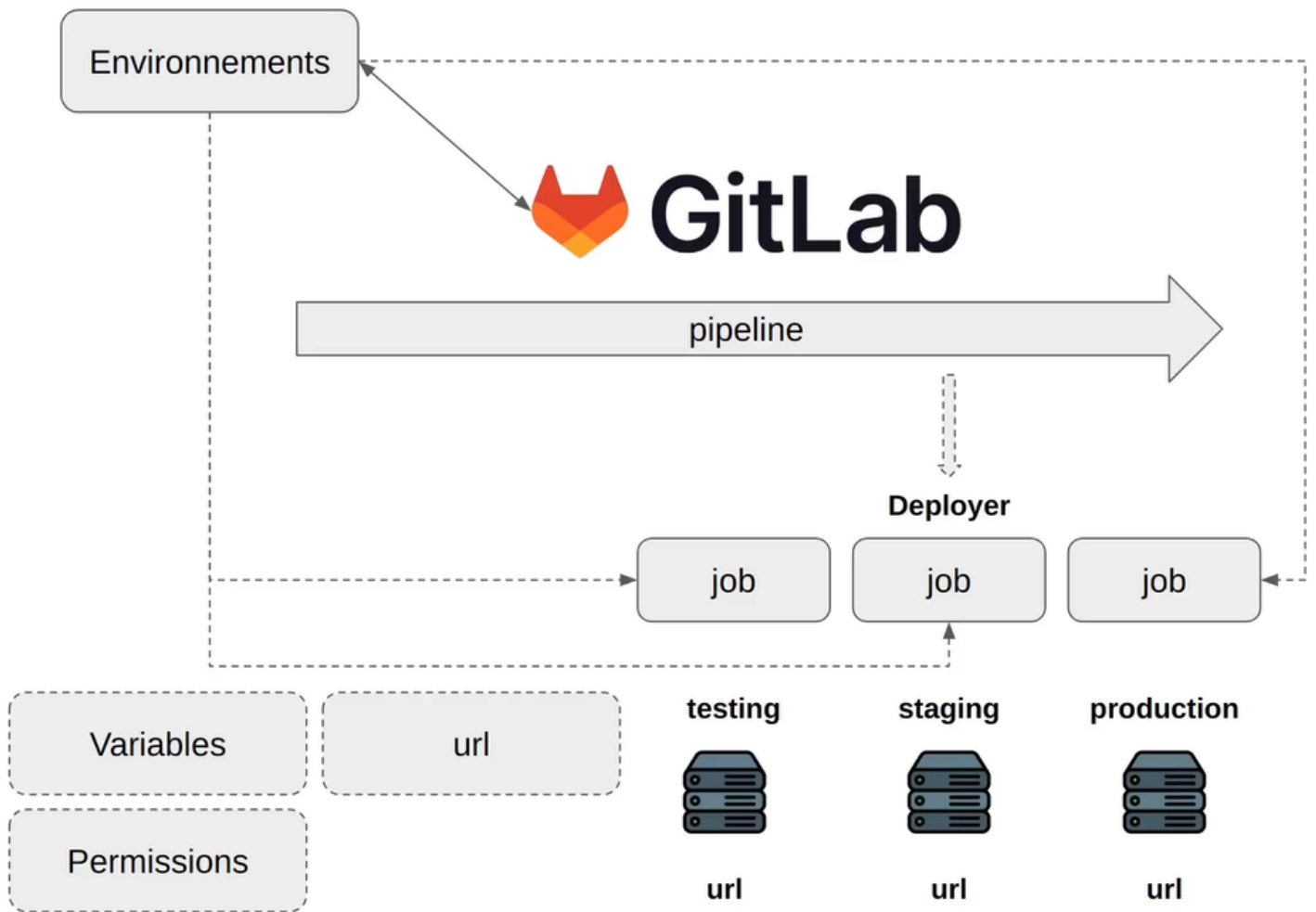
b) Pratique

Dans une application nous manipulons en général plusieurs environnements :



GitLab permet de créer des environnements divers ce qui va nous permettre par exemple d'exécuter les Jobs suivant leur environnement.





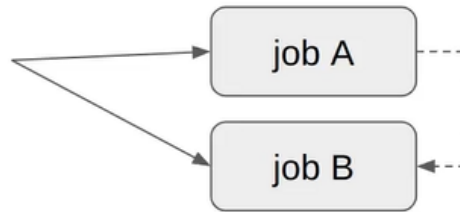
On pourra aussi définir des variables et permissions relatives à des environnements.

On peut définir un environnement avec des Jobs et on peut définir des actions sur ces Jobs.



Environnements

test



start, prepare, stop, verify, et access.

on_stop: Job B

action: stop

Retournons à notre projet GitLab puis cliquer sur Operate → Environnement :

Operate

Environments

Kubernetes clusters

Terraform states

Terraform modules

Google Cloud

Active 1 Stopped 0

Clean up environments

Enable review apps

New environment

Search by environment name

> production

Stop

New environment

Environments

Environments allow you to track deployments of your application. [More information.](#)

Name

test

Description

Preview

B *I* u | |

This is my test environnement.

Switch to rich text editing

External URL

https://test-env.com

GitLab agent

Select agent

Select an agent with Kubernetes access to the project or group. [How do I grant Kubernetes access?](#)

Save Cancel

Active 2 Stopped 0

Clean up environments

Enable review apps

New environment

Search by environment name

> production

Stop

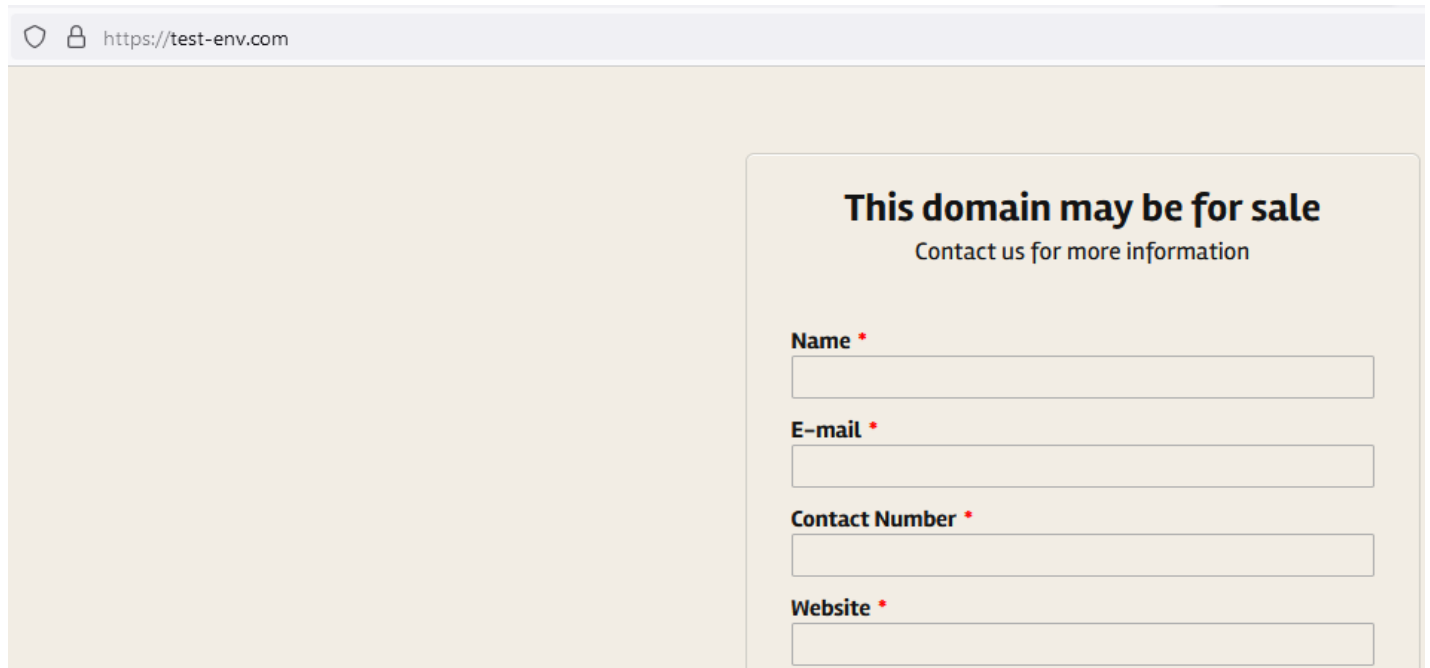
test

Open

Stop

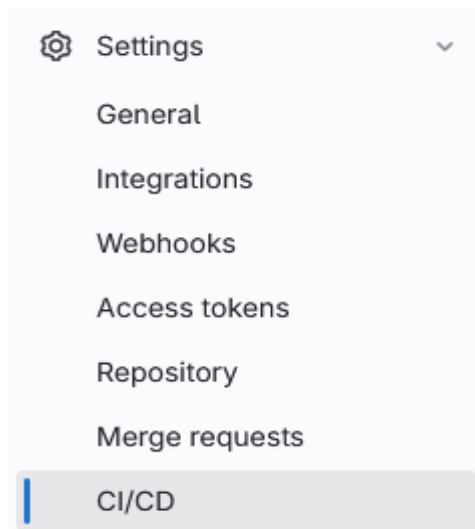
There are no deployments for this environment yet. [Learn more about setting up deployments.](#)

En cliquant sur le bouton Open on obtient :



The screenshot shows a web browser window with the address bar displaying `https://test-env.com`. The main content area has a light beige background. On the right side, there is a white box with a black border containing the text "This domain may be for sale" in bold, followed by "Contact us for more information". Below this, there are four form fields, each with a label and a red asterisk: "Name", "E-mail", "Contact Number", and "Website". Each field is represented by a white rectangular input box.

Nous allons maintenant créer la même variable en fonction des environnements. Soit **DB_URL** la variable qui désigne l'url pour se connecter à une base de données.



Variables

Expand

Variables store information that you can use in job scripts. Each project can define a maximum of 8000 variables. [Learn more.](#)

Pipeline trigger tokens

Expand

Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates a user's project access and permissions. [Learn more.](#)

Automatic deployment rollbacks

Expand

Automatically roll back to the last successful deployment when a critical problem is detected.

Deploy freezes

Expand

Add a freeze period to prevent unintended releases during a period of time for a given environment. You must update the deployment jobs in `.gitlab-ci.yml` according to the deploy freezes added here. [Learn more.](#) Specify deploy freezes using [cron syntax](#).











Variables

Variables store information that you can use in job scripts. Each project can define a maximum of 8000 variables. [Learn more.](#)

Variables can be accidentally exposed in a job log, or maliciously sent to a third party server. The masked variable feature can help reduce the risk of accidentally exposing variable values, but is not a guaranteed method to prevent malicious users from accessing variables. [How can I make my variables more secure?](#)

Variables can have several attributes. [Learn more.](#)

- **Visibility:** Set the visibility level for the value. Can be visible, masked, or masked and hidden.
- **Flags**
 - **Protected:** Only exposed to protected branches or protected tags.
 - **Expanded:** Variables with `$` will be treated as the start of a reference to another variable.

CI/CD Variables </> 2				Reveal values	Add variable
Key ↑	Value	Environments	Actions		
FROM_GITLAB_HIDDEN  Masked	***** 	All (default) 	 		
FROM_GITLAB_VISIBLE 	***** 	All (default) 	 		

Add variable



Type

Variable (default) ▼

Environments

test ▼

Visibility

- ☒ Visible
Can be seen in job logs.

Key

DB_URL

You can use CI/CD variables with the same name in different places, but the variables might overwrite each other. [What is the order of precedence for variables?](#)

Value

db-test-env

Variable value will be evaluated as raw string.

Add variable

Cancel

Add variable



Type

Variable (default)



Environments

production



Visibility



Visible

Can be seen in job logs.

Key

DB_URL


You can use CI/CD variables with the same name in different places, but the variables might overwrite each other. [What is the order of precedence for variables?](#)

Value

db-production-env











Add variable

Cancel

CI/CD Variables  4

Reveal values

Add variable

Key ↑	Value	Environments	Actions
DB_URL 	***** 	production 	 
DB_URL 	***** 	test 	 

Active 2
Stopped 0

Clean up environments

Enable review apps

New environment

Q Search by environment name

> production

Stop

test

Open

Stop

Running

#5

d5d035f9

Created just now

Update .gitlab-ci.yml file

Triggerer

Job

Branch

@elhadji.gaye83

test_deploy

main

Pipeline
Jobs 1
Tests 0

test

test_deploy

test_deploy

Passed Started 1 minute ago by El Hadji Gaye

This job is deployed to test.

Search visible log output

Q

1 Running with gitlab-runner 17.4.0~pre.110.g27400594 (27400594)
2 on blue-1.saas-linux-small.runners-manager.gitlab.com/default j1aLQxS, system ID: s_ccdc2f364be8
3 Resolving secrets
4 Preparing the "docker+machine" executor
5 Using Docker executor with image ruby:3.1 ...
6 Pulling docker image ruby:3.1 ...
7 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:b7fe909968d1e473c5448ee255875bdb65c67df0efe28a0991f97a91ce2e71e7 ...
8 Preparing environment
9 Running on runner-j1aldqxs-project-61863328-concurrent-0 via runner-j1aldqxs-s-l-s-amd64-1727644638-a3ea41cf...
10 Getting source from Git repository
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/formation9273638/formation/.git/
13 Created fresh repository.
14 Checking out d5d035f9 as detached HEAD (ref is main)...
15 Skipping Git submodules setup
16 \$ git remote set-url origin "\${CI_REPOSITORY_URL}"
17 Executing "step_script" stage of the job script
18 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:b7fe909968d1e473c5448ee255875bdb65c67df0efe28a0991f97a91ce2e71e7 ...
19 \$ echo "Pipeline test environnement"
20 Pipeline test environnement
21 \$ echo DB_URL = \$DB_URL
22 DB_URL = db-test-env
23 Cleaning up project directory and file based variables
24 Job succeeded

5. Les images docker

a) Définition

Dans les pipelines CI/CD de GitLab, il est souvent nécessaire d'exécuter des jobs dans un environnement spécifique. L'une des manières de faire cela est d'utiliser des images Docker.

Qu'est-ce que l'instruction image?

L'instruction image permet de spécifier une image Docker dans laquelle le job s'exécute. Elle est définie comme une instruction de type "Job", ce qui signifie que vous pouvez l'utiliser dans le contexte d'un job ou dans la section default.

Syntaxe

La syntaxe peut être de plusieurs types :

- `<image-name>` : utilise le tag latest par défaut.
- `<image-name>:<tag>` : spécifie le tag de l'image.
- `<image-name>@<digest>` : utilise un digest spécifique de l'image.

Définir une image par défaut

Vous pouvez définir une image par défaut pour tous les jobs du pipeline en utilisant la section default.

```
default:  
  image: node:18
```

Utilisation de image:name

La clé name sous image est similaire à utiliser image elle-même et sert à nommer l'image Docker.

```
image:  
  name: "registry.example.com/my/image:latest"
```

Utilisation de image:entrypoint

La clé entrypoint permet de définir la commande ou le script à exécuter lors de l'entrée dans le conteneur. Cela remplace le point d'entrée défini dans l'image Docker.

```
image:  
  name: super/sql:experimental  
  entrypoint: [""]
```

Utilisation de image:pull_policy

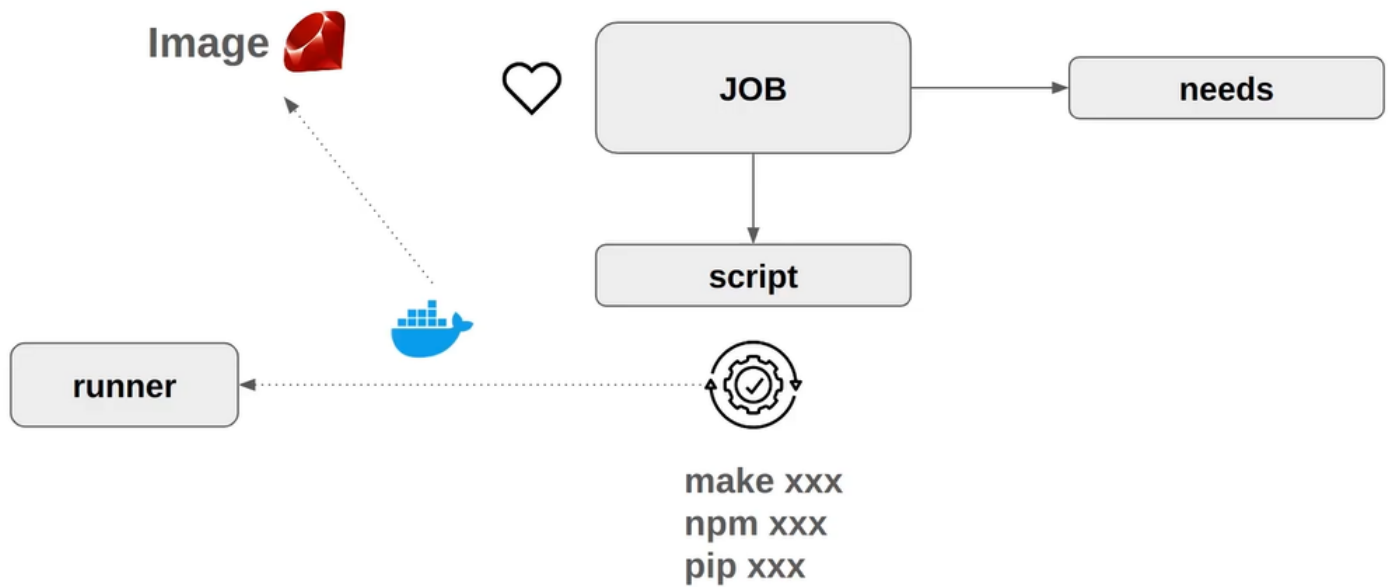
La clé pull_policy permet de spécifier la politique de récupération de l'image Docker. Les options possibles sont :

- **always** : toujours récupérer l'image.
- **if-not-present** : récupérer l'image seulement si elle n'est pas déjà présente.
- **never** : ne jamais récupérer l'image.

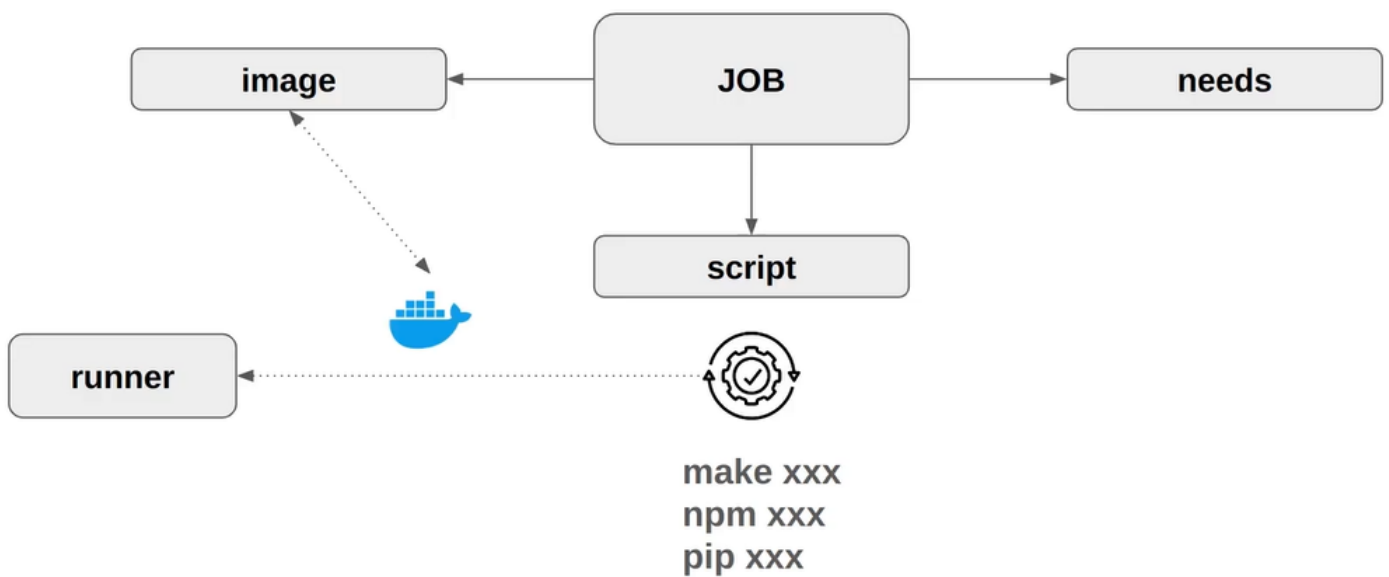
```
image:  
  name: node:1ts  
  pull_policy: if-not-present
```

b) Pratique

Nous avons vu précédemment que pour lancer des Job nous aurons besoin d'un Runner et ce Runner utilise par défaut une image Ruby.



Pour des cas encore plus spécifiques nous aurons besoins d'utiliser d'autres type d'image spécifique de docker.



Revenons sur notre projet repos GitLab :

Se rendre dans notre projet <https://gitlab.com/formation9273638/formation>

F

formation

main ▾

formation / ▾

History

Find file

Edit ▾

Code ▾

Update .gitlab-ci.yml file

El Hadji Gaye authored 2 hours ago

974fcfa8

Name	Last commit	Last update
.gitlab-ci.yml	Update .gitlab-ci.yml file	2 hours ago
README.md	Initial commit	1 week ago

README.md

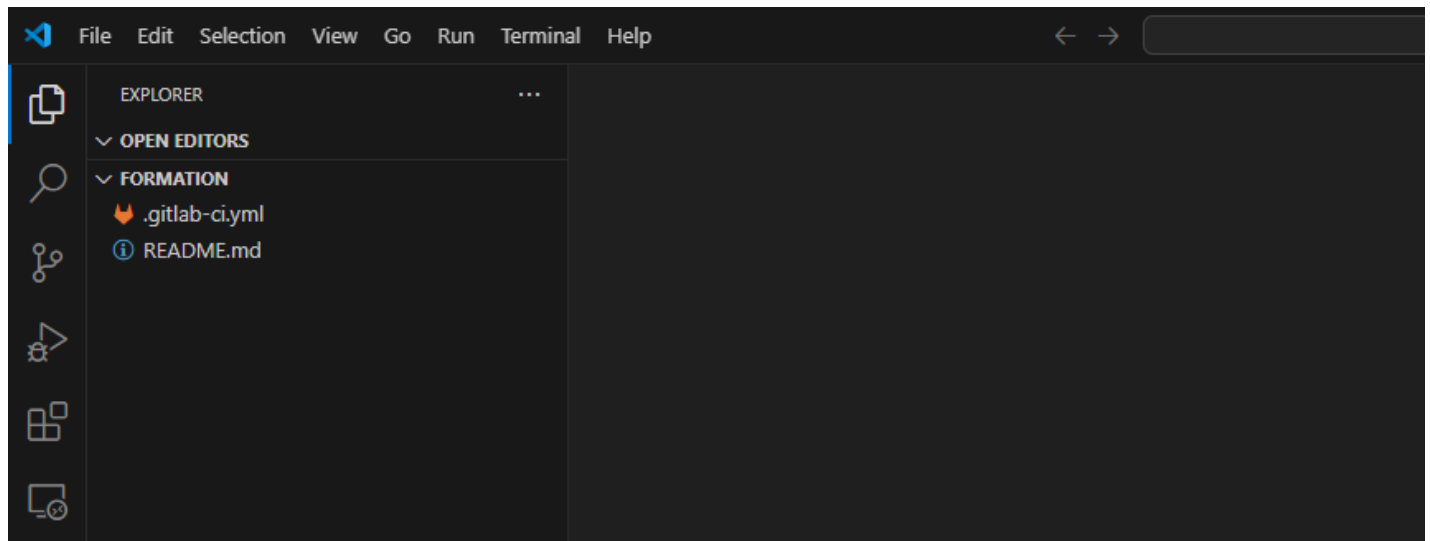
formation

Se placer dans le repertoire **Gitlab/courses** et lancer git clone.

cd Gitlab/courses

git clone [git@gitlab.com:formation9273638/formation.git](https://gitlab.com/formation9273638/formation.git)

```
MINGW64:/c/Users/elhad/Desktop/AutoEntrepreneur/Formations/Gitlab/courses
elhad@LAPTOP-C15H4CMH MINGW64 ~
$ cd C:/Users/elhad/Desktop/AutoEntrepreneur/Formations/Gitlab/courses
elhad@LAPTOP-C15H4CMH MINGW64 ~/Desktop/AutoEntrepreneur/Formations/Gitlab/courses
$ git clone git@gitlab.com:formation9273638/formation.git
Cloning into 'formation'...
remote: Enumerating objects: 24, done.
remote: Counting objects: 100% (24/24), done.
remote: Compressing objects: 100% (23/23), done.
remote: Total 24 (delta 5), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (24/24), 5.86 KiB | 5.86 MiB/s, done.
Resolving deltas: 100% (5/5), done.
```



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

Microsoft Windows [version 10.0.19045.4894]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation>
```

node -v
npm -v

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS

Microsoft Windows [version 10.0.19045.4894]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation>node -v
v22.9.0

C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation>npm -v
10.8.3

C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation>
```

npm create vite@latest

```
C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation>npm create vite@latest

> npx
> create-vite

? Project name: » vite-project
```

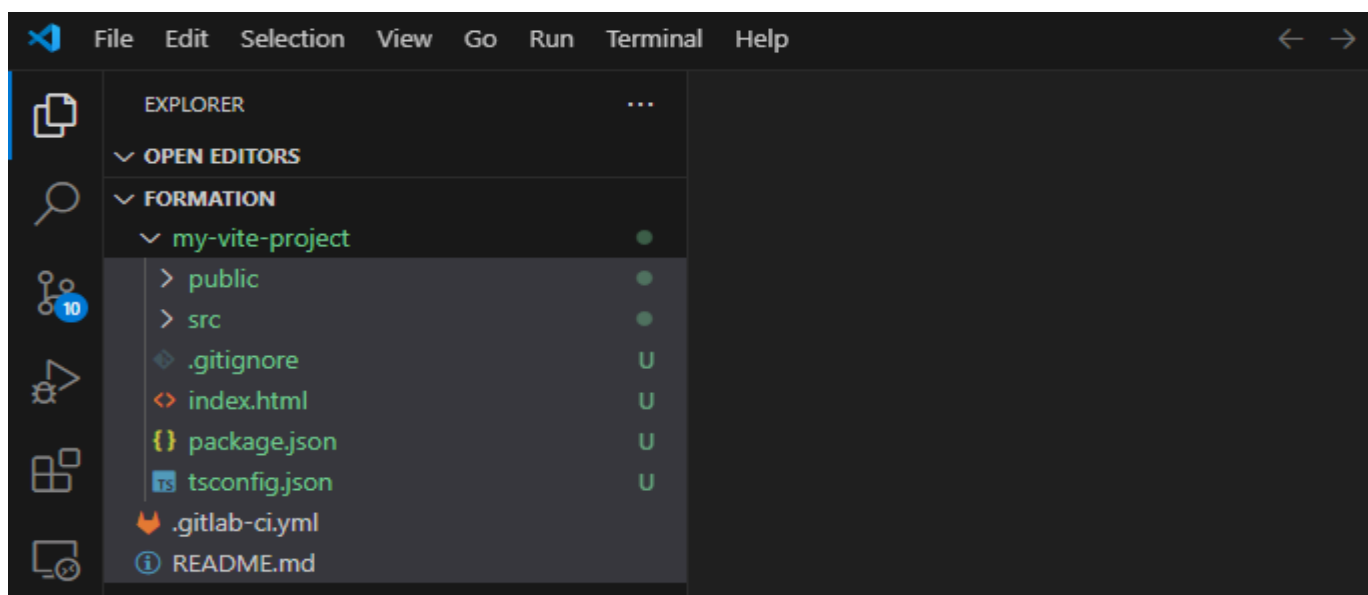
```
C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation>npm create vite@latest  
  
> npx  
> create-vite  
  
? Project name: » my-vite-project
```

```
✓ Project name: ... my-vite-project  
? Select a framework: » - Use arrow-keys. Return to submit.  
> Vanilla  
  Vue  
  React  
  Preact  
  Lit  
  Svelte  
  Solid  
  Qwik  
  Others
```

```
✓ Project name: ... my-vite-project  
✓ Select a framework: » Vanilla  
? Select a variant: » - Use arrow-keys. Return to submit.  
> TypeScript  
  JavaScript
```

```
Scaffolding project in C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation\my-vite-project...  
  
Done. Now run:  
  
cd my-vite-project  
npm install  
npm run dev
```

On obtient :



Déplacer le contenu de **my-vite-project** vers **Formation** puis supprimer le dossier **my-vite-project**.

`cd Gitlab/courses/formation`

`npm install`

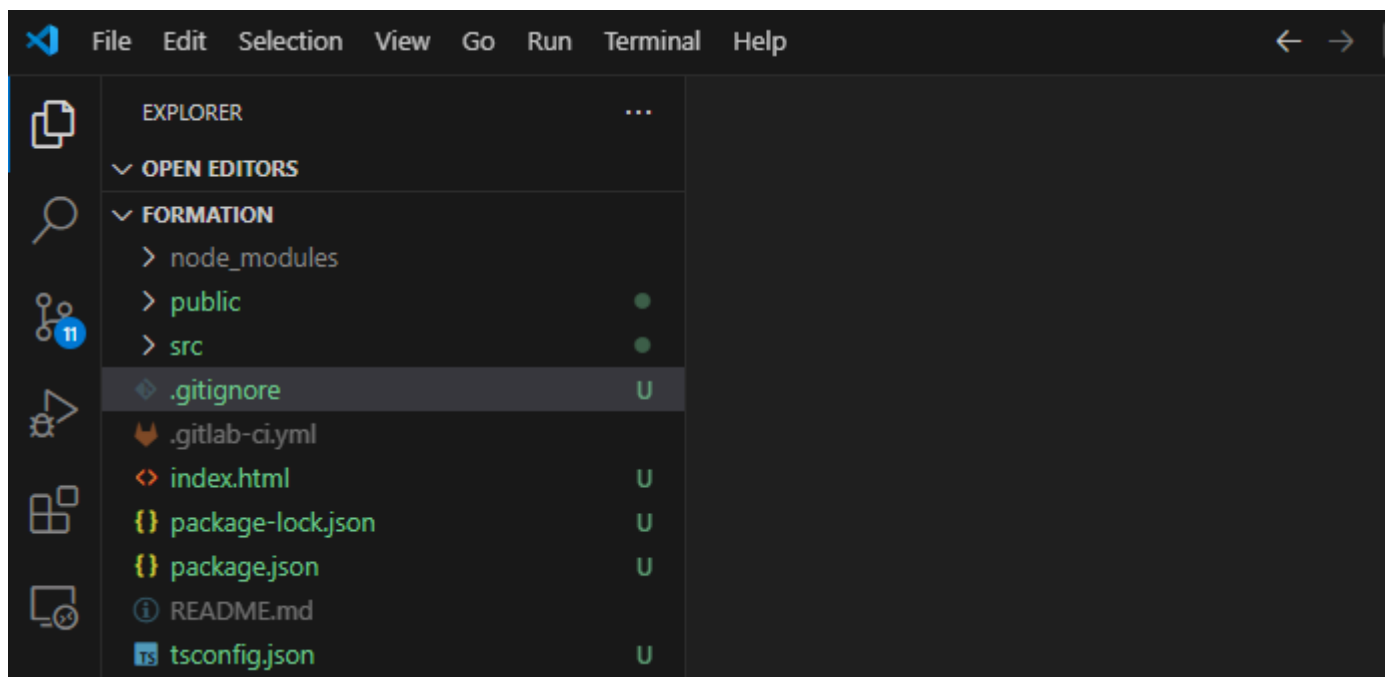
```
C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation>npm install

added 11 packages, and audited 12 packages in 4s

3 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

On note l'apparition d'un dossier **node_modules** :



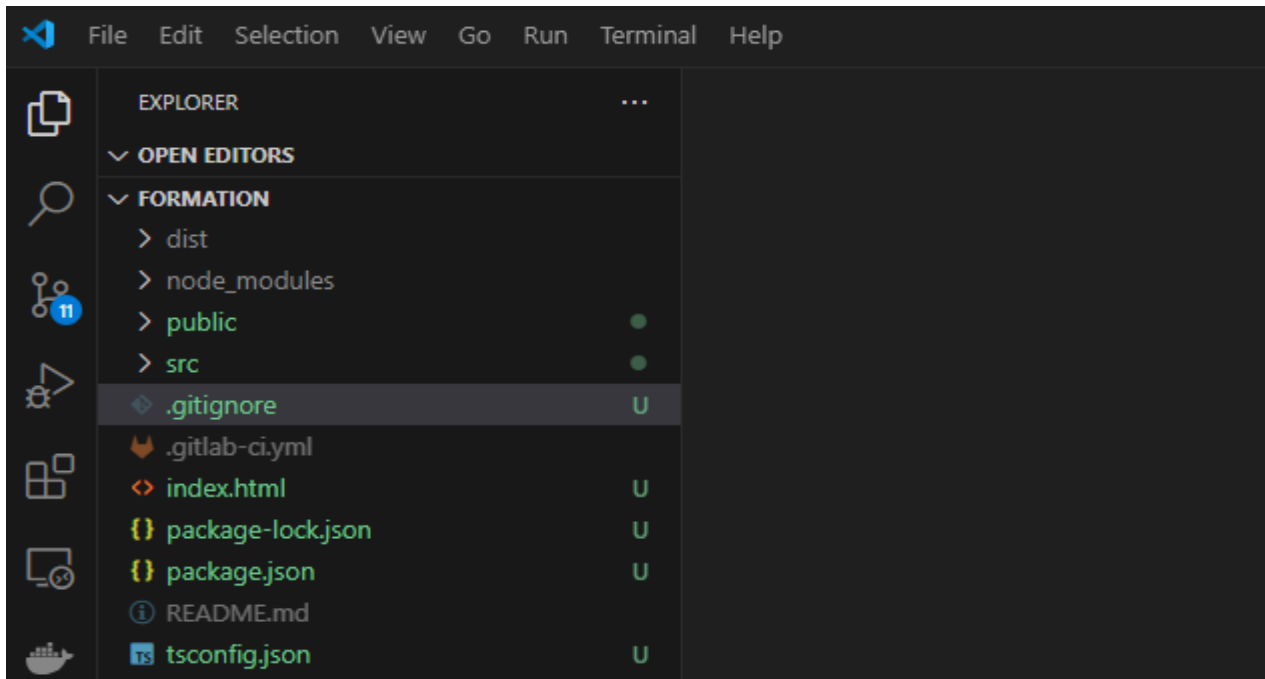
`npm run build`

```
C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation>npm run build

> my-vite-project@0.0.0 build
> tsc && vite build

vite v5.4.8 building for production...
✓ 7 modules transformed.
dist/index.html          0.46 kB | gzip: 0.29 kB
dist/assets/index-Cz4zGhbH.css 1.21 kB | gzip: 0.62 kB
dist/assets/index-Bd-pKGJy.js  3.05 kB | gzip: 1.64 kB
✓ built in 231ms
```

On note l'apparition d'un dossier **dist** :



`npm run dev`

```
C:\Users\elhad\Desktop\AutoEntrepreneur\Formations\Gitlab\courses\formation>npm run dev

> my-vite-project@0.0.0 dev
> vite

VITE v5.4.8 ready in 151 ms

➔ Local:   http://localhost:5173/
➔ Network: use --host to expose
➔ press h + enter to show help
```

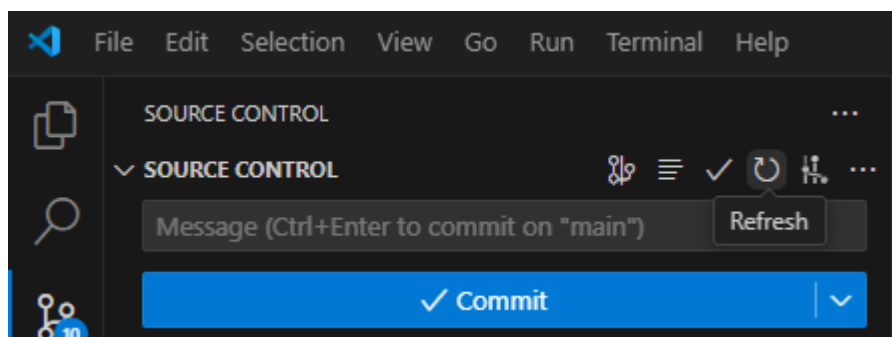
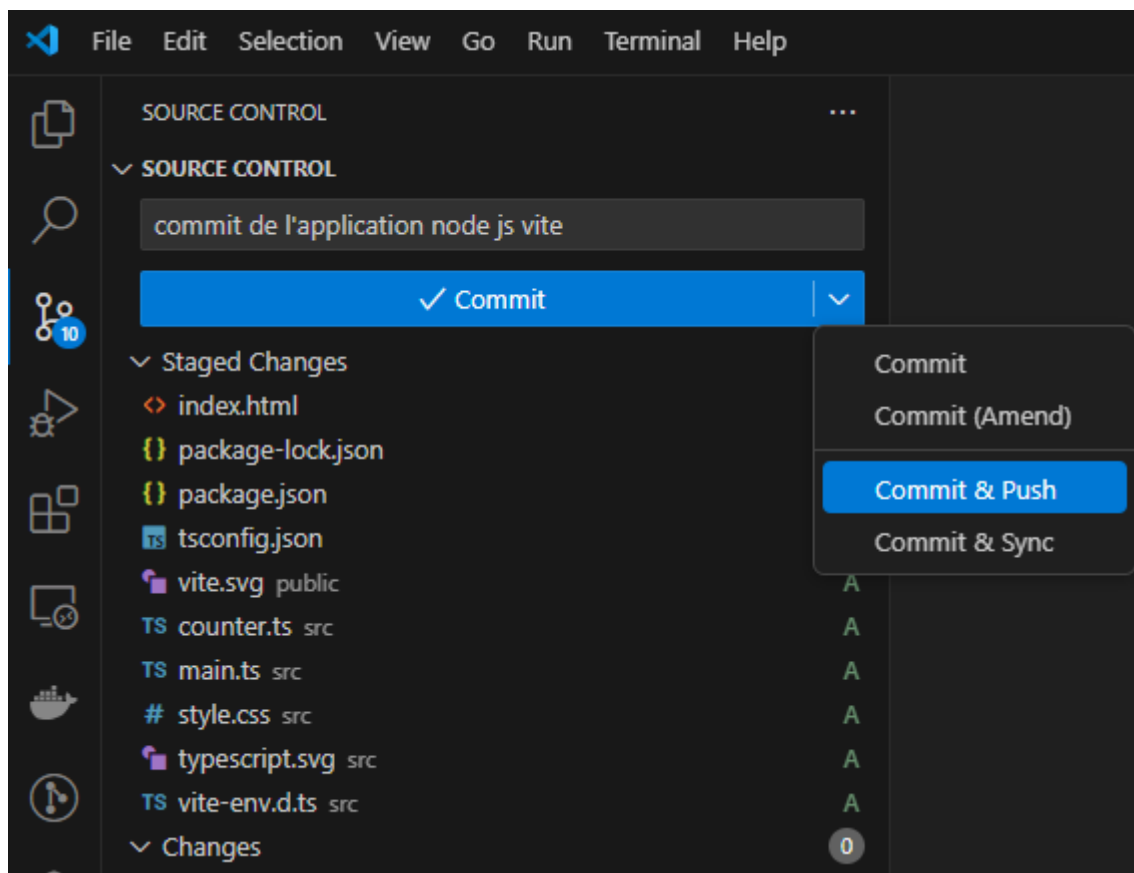
On obtient sur <http://localhost:5173/>



Vite + TypeScript

count is 0

Commiter les changement et pusher dans le repos distant :



On obtient sur le repos Git :

F

formation

main

formation /


+

History

Find file

Edit

Code




commit de l'application node js vite

El Hadji Gaye authored 1 minute ago

✓

bdf663e1



Name	Last commit	Last update
public	commit de l'application node js vite	1 minute ago
src	commit de l'application node js vite	1 minute ago
.gitignore	Commit de l'application node js vite	8 minutes ago
.gitlab-ci.yml	Update .gitlab-ci.yml file	3 hours ago
README.md	Initial commit	1 week ago
index.html	commit de l'application node js vite	1 minute ago
package-lock.json	commit de l'application node js vite	1 minute ago
package.json	commit de l'application node js vite	1 minute ago
tsconfig.json	commit de l'application node js vite	1 minute ago

Le fichier `.gitlab-ci.yml` peut alors devenir :

```
build-job:
  stage: build
  script:
    - npm run build
```

On obtient à l'exécution de ce pipeline :

Pipeline **Jobs 1** **Failed Jobs 1** **Tests 0**

build

✖ build-job ↻

build-job

✖ Failed Started just now by El Hadji Gaye

Search visible log output 🔍 📄 ⏮ ⏭ ⏪ ⏩ ⏴ ⏵

```
1 Running with gitlab-runner 17.4.0-pre.110.g27400594 (27400594)
2   on blue-3.saas-linux-small-amd64.runners-manager.gitlab.com/default zxwgkJP, system ID: s_d5d3abbd8a
3 Resolving secrets
4 Preparing the "docker+machine" executor
5 Using Docker executor with image ruby:3.1 ...
6 Pulling docker image ruby:3.1 ...
7 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:1cd339d07ba2b9f6b8e61a7a2ca75
  49e1e6053af7250717d7c94c360be48decf ...
8 Preparing environment
9 Running on runner-zxwgkJP-project-61863328-concurrent-0 via runner-zxwgkJP-s-l-s-amd64-1727551514-78738f0b...
10 Getting source from Git repository
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/formation9273638/formation/.git/
13 Created fresh repository.
14 Checking out 6baec307 as detached HEAD (ref is main)...
15 Skipping Git submodule setup
16 $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 Executing "step_script" stage of the job script
18 Using docker image sha256:12bc18a740469918b597219b1033d2fd4a60594a8ada2ec29383f64e39e8df0b for ruby:3.1 with digest ruby@sha256:1cd339d07ba2b9f6b8e61a7a2ca75
  49e1e6053af7250717d7c94c360be48decf ...
19 $ npm run build
20 /usr/bin/bash: line 146: npm: command not found
21 Cleaning up project directory and file based variables
22 ERROR: Job failed: exit code 1
```

Troubleshoot

L'exécution du pipeline a échoué car notre Runner ne connaît pas la commande **npm**.

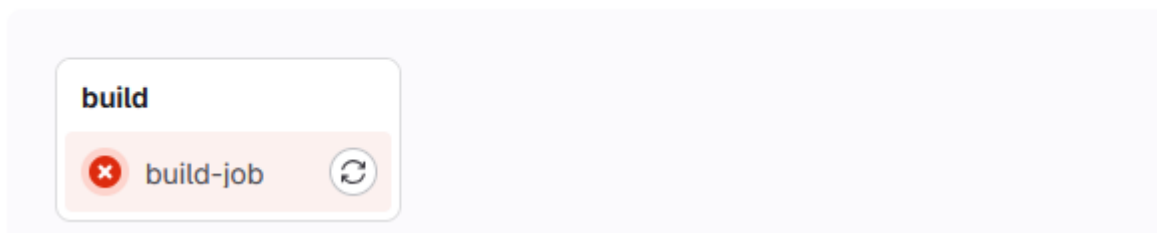
Nous allons donc utiliser une image docker de node pour palier à ce problème.

Aller sur le site https://hub.docker.com/_/node pour trouver une image docker de node.

Le fichier `.gitlab-ci.yml` peut alors devenir :

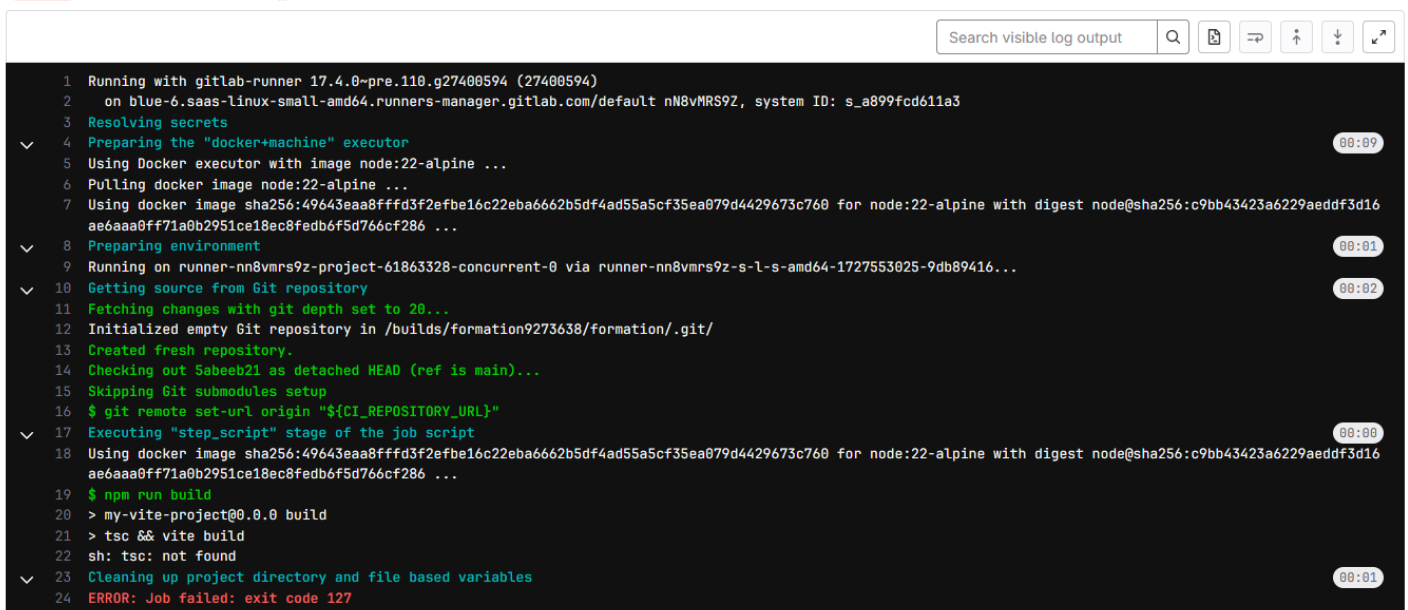
```
build-job:
  image: node:22-alpine
  stage: build
  script:
    - npm run build
```

Pipeline Jobs 1 Failed Jobs 1 Tests 0



build-job

Failed Started 2 minutes ago by El Hadji Gaye



Troubleshoot

Il manque donc la dépendance de vite :

Le fichier `.gitlab-ci.yml` peut alors devenir :

```
build-job:
  image: node:22-alpine
  stage: build
  before_script:
    - npm install
  script:
    - npm run build
```

build



build-job



build-job

✓ Passed Started 1 minute ago by El Hadji Gaye

Search visible log output



```
1 Running with gitlab-runner 17.4.0~pre.110.g27400594 (27400594)
2 on blue-1.saas-linux-small.runners-manager.gitlab.com/default j1aLdQxS, system ID: s_ccdc2f364be8
3 Resolving secrets
4 Preparing the "docker+machine" executor 00:00
5 Using Docker executor with image node:22-alpine ...
6 Pulling docker image node:22-alpine ...
7 Using docker image sha256:49643eaa8fffd3f2efbe16c22eba6662b5df4ad55a5cf35ea079d4429673c760 for node:22-alpine with digest node@sha256:c9bb43423a6229aedd3d16ae6aaa0ff71a0b2951ce18ec8fedb6f5d766cf286 ...
8 Preparing environment 00:02
9 Running on runner-j1aldqxs-project-61863328-concurrent-0 via runner-j1aldqxs-s-l-s-amd64-1727553479-2dcc0e31...
10 Getting source from Git repository 00:01
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/formation9273638/formation/.git/
13 Created fresh repository.
14 Checking out 23791a66 as detached HEAD (ref is main)...
15 Skipping Git submodules setup
16 $ git remote set-url origin "${CI_REPOSITORY_URL}"
17 Executing "step_script" stage of the job script 00:04
18 Using docker image sha256:49643eaa8fffd3f2efbe16c22eba6662b5df4ad55a5cf35ea079d4429673c760 for node:22-alpine with digest node@sha256:c9bb43423a6229aedd3d16ae6aaa0ff71a0b2951ce18ec8fedb6f5d766cf286 ...
19 $ npm install
20 added 12 packages, and audited 13 packages in 1s
21 3 packages are looking for funding
22   run `npm fund` for details
23 found 0 vulnerabilities
24 $ npm run build
25 > my-vite-project@0.0.0 build
26 > tsc && vite build
27 vite v5.4.8 building for production...
28 transforming...
29 ✓ 7 modules transformed.
30 rendering chunks...
31 computing gzip size...
32 dist/index.html      0.46 kB | gzip: 0.29 kB
33 dist/assets/index-Cz42GhbH.css 1.21 kB | gzip: 0.62 kB
34 dist/assets/index-Bd-pKGJy.js  3.05 kB | gzip: 1.64 kB
35 ✓ built in 132ms
36 Cleaning up project directory and file based variables 00:00
37 Job succeeded
```

6. *Contrôle du flux d'exécution*

a) Définition

L'option `allow_failure`

L'option `allow_failure` permet de définir comment le pipeline doit se comporter en cas d'échec d'un job. Vous avez deux choix principaux :

- `allow_failure: true` : si un job échoue, le pipeline continue d'exécuter les jobs suivants.
- `allow_failure: false` : si un job échoue, le pipeline s'arrête et n'exécute pas les jobs suivants.

Lorsque `allow_failure: true` est configuré, un avertissement orange apparaît pour indiquer l'échec du job, mais le pipeline est considéré comme réussi et le commit associé est marqué comme passé.

Par défaut, la valeur pour `allow_failure` est :

- `true` pour les jobs manuels
- `false` pour les jobs qui utilisent `when: manual` à l'intérieur de règles
- `false` dans tous les autres cas

Voici un premier exemple :

```
job1:
  stage: test
  script:
    - execute_script_1

job2:
  stage: test
  script:
    - execute_script_2
  allow_failure: true

job3:
  stage: deploy
  script:
    - deploy_to_staging
  environment: staging
```

Dans cet exemple, `job1` et `job2` sont exécutés en parallèle. Si `job1` échoue, les jobs de l'étape de déploiement ne démarrent pas. Si `job2` échoue, les jobs de l'étape de déploiement peuvent quand même démarrer.

Autres détails :

- Vous pouvez utiliser `allow_failure` comme une sous-clé de règles.
- Si `allow_failure: true` est défini, le job est toujours considéré comme réussi et les jobs ultérieurs avec `when: on_failure` ne démarrent pas si ce job échoue.

Voici un deuxième exemple :

```
stages:
- build
- test
- deploy

build:
  stage: build
  script:
    - echo "Compilation du code..."
    - sleep 5
    - echo "Compilation terminée."

test1:
  stage: test
  script:
    - echo "Exécution des tests unitaires..."
    - sleep 10
    - echo "Tests unitaires réussis."

test2:
  stage: test
  script:
    - echo "Ce job va échouer..."
    - exit 1 # Cette commande fait échouer le job.
  allow_failure: true # Même si ce job échoue, le pipeline continuera.

deploy:
  stage: deploy
  script:
    - echo "Déploiement en production..."
    - sleep 5
    - echo "Déploiement réussi."
```

allow_failure:exit_codes

Une autre version plus avancée du paramètre est `allow_failure:exit_codes`, qui permet de contrôler quand un job doit être autorisé à échouer en fonction des codes de sortie. Par exemple, vous pouvez définir des codes de sortie spécifiques pour lesquels le job est autorisé à échouer.

```
test_job_1:  
  script:  
    - exit 1  
  allow_failure:  
    exit_codes: 137
```

```
test_job_2:  
  script:  
    - exit 137  
  allow_failure:  
    exit_codes:  
    - 137  
    - 255
```

Dans cet exemple, `test_job_1` échouera mais ne stoppera pas le pipeline si le code de sortie est 137. `test_job_2` est autorisé à échouer pour les codes de sortie 137 et 255.

L'option when

Le mot-clé when permet de définir les conditions d'exécution d'un job dans un pipeline. Ce mot-clé peut prendre plusieurs valeurs possibles :

- on_success (par défaut) : exécute le job seulement si tous les jobs des étapes précédentes ont réussi ou ont allow_failure: true.
- on_failure : exécute le job seulement si au moins un job des étapes précédentes a échoué.
- never : ne jamais exécuter le job, indépendamment du statut des jobs précédents. Ce mot-clé est souvent utilisé avec rules ou workflow: rules.
- always : exécute le job quel que soit le résultat des étapes précédentes.
- manual : exécute le job seulement lorsqu'il est déclenché manuellement depuis l'interface utilisateur de GitLab.
- delayed : retarde l'exécution du job pendant une durée spécifiée.

Exemple :

```
stages:
- build
- cleanup_build
- test
- deploy
- cleanup

build_job:
  stage: build
  script:
    - make build

cleanup_build_job:
  stage: cleanup_build
  script:
    - cleanup build when failed
  when: on_failure

test_job:
  stage: test
  script:
    - make test

deploy_job:
  stage: deploy
  script:
    - make deploy
  when: manual
  environment: production
```

```
cleanup_job:  
  stage: cleanup  
  script:  
    - cleanup after jobs  
  when: always
```

b) Pratique

Revenons à notre projet GitLab :

7. *Filtrage et conditions*

a) Définition

Les règles (rules)

Les règles permettent une personnalisation fine des jobs dans les pipelines, en offrant des conditions basées sur des variables, des changements de fichier et d'autres critères.

Les règles sont évaluées lors de la création du pipeline et dans l'ordre jusqu'à la première correspondance. Lorsqu'une correspondance est trouvée, le travail est inclus ou exclu du pipeline, en fonction de la configuration.

Les règles remplacent `only/except` : donc si vous rencontrez ces mots clés vous saurez qu'il s'agit de l'ancien fonctionnement.

rules:if

`rules:if` vous permet de définir une condition pour inclure ou exclure un job dans un pipeline.

Lorsque `rules` et `when` sont utilisés ensemble, `rules` détermine si le job sera ajouté au pipeline en premier lieu, tandis que `when` décide du moment où ce job sera exécuté une fois qu'il fait partie du pipeline.

```
job:
  script: echo "Règle complexe"
  rules:
    - if: '$CI_COMMIT_BRANCH == "main" || $CI_COMMIT_BRANCH == "develop"'
      when: always
    - if: '$CI_COMMIT_BRANCH =~ /^feature.* / && $CI_PIPELINE_SOURCE == "schedule"'
      when: manual
    allow_failure: true
```

Dans cet exemple, le job est ajouté au pipeline si :

- 1) La branche actuelle est `main` ou `develop`. Le job sera exécuté immédiatement.
- 2) La branche actuelle commence par `feature` et la source du pipeline est un événement planifié. Dans ce cas, le job sera ajouté comme une tâche manuelle.

rules:changes

Ce mot-clé permet de déclencher des jobs en fonction des fichiers modifiés.

```
job_backend:
  script: echo "Building du backend"
  rules:
    - changes:
        - 'backend/*'
    - if: '$CI_PIPELINE_SOURCE == "schedule"'
  when: never
```

Ici, job_backend sera inclus dans le pipeline si des fichiers dans le répertoire backend/ ont été modifiés. Cependant, si la source du pipeline est un événement planifié, le job sera exclu.

rules:changes:compare_to

Vous pouvez utiliser compare_to pour spécifier la référence par rapport à laquelle les changements doivent être détectés.

```
job:
  script: echo "Building..."
  rules:
    - changes:
        paths:
          - README.md
        compare_to: 'refs/heads/main'
```

rules:exists

La directive rules:exists permet de conditionner l'exécution d'un job à la présence d'un ou plusieurs fichiers.

```
job_frontend :
  script: echo "Construction du frontend !"
  rules:
    - exists:
        - 'frontend/*'
    - if: '$CI_COMMIT_MESSAGE =~ /forcer la construction du frontend/'
```

Dans cet exemple, le job est ajouté au pipeline si des fichiers existent dans le dossier frontend/. De plus, un autre critère est ajouté : si le message de commit contient la phrase "forcer la construction du frontend", alors ce job sera aussi ajouté au pipeline.

rules:allow_failure

Ce mot-clé permet à un job d'échouer sans que cela n'affecte le reste du pipeline.

```
job :  
  script: echo "Ce job peut échouer."  
  rules:  
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'  
      allow_failure: true
```

Si la source du pipeline est un événement lié à une demande de fusion (merge_request_event), alors ce job est autorisé à échouer sans entraîner l'échec de tout le pipeline.

rules:needs

Cela permet de définir des dépendances conditionnelles entre les job.

```
test :  
  stage: test  
  needs: ["build"]  
  rules:  
    - if: '$CI_COMMIT_REF_NAME == "main"'  
      needs: ["lint", "build"]
```

rules:variables

Permet de définir des variables conditionnelles.

```
deploy :  
  script: ./deploy.sh  
  variables :  
    ENV: "staging"  
  rules:  
    - if: '$CI_COMMIT_REF_NAME == "main"'  
      variables :  
        ENV: "production"
```

8. *La notion de default*

a) Définition

La section default

La section default permet de définir des valeurs par défaut pour certaines clés globales.

Les jobs qui ne définissent pas une ou plusieurs des clés listées utiliseront la valeur définie dans la section default.

Ces clés peuvent avoir des valeurs par défaut personnalisées :

- after_script
- artifacts
- before_script
- cache
- hooks
- image
- interruptible
- retry
- services
- tags
- timeout

Lorsque le pipeline est créé, chaque valeur par défaut est copiée dans tous les jobs qui n'ont pas cette clé définie.

Si un job a déjà une des clés configurées, la configuration dans le job prend le dessus et n'est pas remplacée par la valeur par défaut.

Exemple de default

Voici un exemple d'utilisation :

```
default:  
  image: ruby:3.0  
  
rspec:  
  script: bundle exec rspec  
  
rspec 2.7:  
  image: ruby:2.7  
  script: bundle exec rspec
```

Dans cet exemple, ruby:3.0 est l'image par défaut pour tous les jobs du pipeline.

Le job rspec 2.7 n'utilise pas la valeur par défaut car il surcharge celle-ci avec une section image spécifique au job.

9. *Gestion des artefacts et dependencies*

a) Définition

Les artefacts

La clé `artifacts` permet de spécifier quels fichiers doivent être conservés comme artefacts d'un job.

Ces artefacts sont des fichiers et répertoires qui sont attachés au job lorsqu'il réussit, échoue ou dans tous les cas.

Ils sont envoyés à GitLab une fois le job terminé. Ils peuvent être téléchargés dans l'interface utilisateur de GitLab et être utilisés dans des étapes ultérieures du pipeline.

Par défaut, les jobs des étapes ultérieures téléchargent automatiquement tous les artefacts créés par les jobs des étapes précédentes.

Vous pouvez contrôler le téléchargement des artefacts dans les jobs avec des dépendances. Nous les verrons juste après.

Artifacts peut être une clé de job ou être utilisée dans la section `default`.

Voici un résumé des configurations disponibles avant de les voir en détail :

- `paths` : spécifie les fichiers et répertoires à enregistrer en tant qu'artefacts.
- `exclude` : permet d'empêcher certains fichiers d'être enregistrés en tant qu'artefacts.
- `expire_in` : définit la durée après laquelle l'artefact sera supprimé.
- `expose_as` : rend les artefacts accessibles via l'interface de demande de fusion de GitLab.
- `name` : permet de nommer l'archive d'artefacts.
- `public` : définit si les artefacts sont accessibles publiquement.
- `reports` : collecte des types spécifiques de rapports en tant qu'artefacts.
- `untracked` : permet à tous les fichiers Git non suivis d'être sauvegardés en tant qu'artefacts.
- `when` : contrôle le moment où les artefacts sont téléchargés, en fonction de la réussite ou de l'échec du job.

artifacts:paths

Il prend un tableau de chemins de fichiers, relatifs au répertoire du projet.

```
job_de_test:
  script:
    - echo "générer un fichier"
    - echo "hello world" > fichier.txt
  artifacts:
    paths:
      - fichier.txt
```

artifacts:exclude

Permet d'exclure certains fichiers ou dossiers d'être inclus dans l'artefact :

```
job_de_test:
  script:
    - echo "générer des fichiers"
    - mkdir binaries
    - echo "hello" > binaries/hello.txt
    - echo "world" > binaries/world.o
  artifacts:
    paths:
      - binaries/
    exclude:
      - binaries/*.o
```

artifacts:expire_in

Utilisez `expire_in` pour spécifier la durée de stockage des artefacts de travail avant qu'ils n'expirent et ne soient supprimés.

Le paramètre `expire_in` n'affecte pas : les artefacts du dernier job, sauf si la conservation des artefacts du dernier travail est désactivée au niveau du projet ou de l'instance.

Après leur expiration, les artefacts sont supprimés toutes les heures par défaut (à l'aide d'une tâche cron) et ne sont plus accessibles.

La durée d'expiration en différentes unités de temps (ex: '42', '3 mins 4 sec', '2 hrs 20 min', 'never'). Si aucune unité de temps n'est précisée, il s'agit de secondes.

Exemple :

```
job_de_test:
  script:
    - echo "hello world" > fichier.txt
  artifacts:
    paths:
      - fichier.txt
    expire_in: 1 week
```

artifacts:expose_as

expose_as permet de configurer le nom à afficher dans l'UI de la merge request pour le lien de téléchargement des artefacts :

```
job_de_test:
  script:
    - echo "hello world" > fichier.txt
  artifacts:
    expose_as: 'Mon fichier'
  paths:
    - fichier.txt
```

artifacts:name

Ce mot-clé permet de définir le nom de l'archive créée pour les artefacts.

```
job:
  script:
    - echo "Génération des artefacts"
  artifacts:
    name : "archive_artefacts"
  paths:
    - mon_fichier/
```

artifacts:public

Ce mot-clé permet de rendre les artefacts publics ou non.

```
job:
  script:
    - echo "Ce sont des artefacts privés"
  artifacts:
    public: false
```


artifacts:reports

Utilisé pour collecter des rapports générés par des modèles inclus dans les jobs.

```
rspec:  
  stage: test  
  script:  
    - echo "Installation des dépendances"  
    - bundle install  
    - echo "Lancement des tests"  
    - rspec  
artifacts:  
  reports:  
    junit: rspec.xml
```

artifacts:untracked

Cette option permet d'ajouter tous les fichiers non suivis par Git en tant qu'artefacts.

```
job:  
  script:  
    - echo "Ajout des fichiers non suivis"  
artifacts:  
  untracked: true
```

artifacts:when

Ce mot-clé permet de contrôler le moment où les artefacts sont téléchargés, en fonction du succès ou de l'échec du job.

```
job:  
  script:  
    - echo "Ce `job` pourrait échouer"  
artifacts:  
  when: on_failure
```

Les dépendances

Le mot-clé `dependencies` permet de spécifier une liste de jobs dont il faut récupérer les artefacts. Vous pouvez également configurer un job pour qu'il ne télécharge aucun artefact.

Il est utile pour optimiser les ressources en ne téléchargeant que les artefacts nécessaires pour un job donné, surtout si vous travaillez avec plusieurs étapes et de nombreux job.

C'est un mot-clé lié à un job. Vous ne pouvez l'utiliser qu'en tant que partie d'un job.

Le statut du job n'a pas d'importance. Si un job échoue ou s'il s'agit d'un job manuel qui n'est pas déclenché, aucune erreur ne se produit.

Si les artefacts d'un job dépendant sont expirés ou supprimés, alors le job échoue.

Valeurs possibles

- Les noms des jobs dont il faut récupérer les artefacts.
- Un tableau vide (`[]`), pour configurer le job de manière à ne télécharger aucun artefact.

Par exemple :

```
build osx:
  stage: build
  script: make build:osx
  artifacts:
    paths:
      - binaries/

build linux:
  stage: build
  script: make build:linux
  artifacts:
    paths:
      - binaries/

test osx:
  stage: test
  script: make test:osx
  dependencies:
    - build osx

test linux:
  stage: test
  script: make test:linux
  dependencies:
    - build linux

deploy:
```

```
stage: deploy  
script: make deploy  
environment: production
```