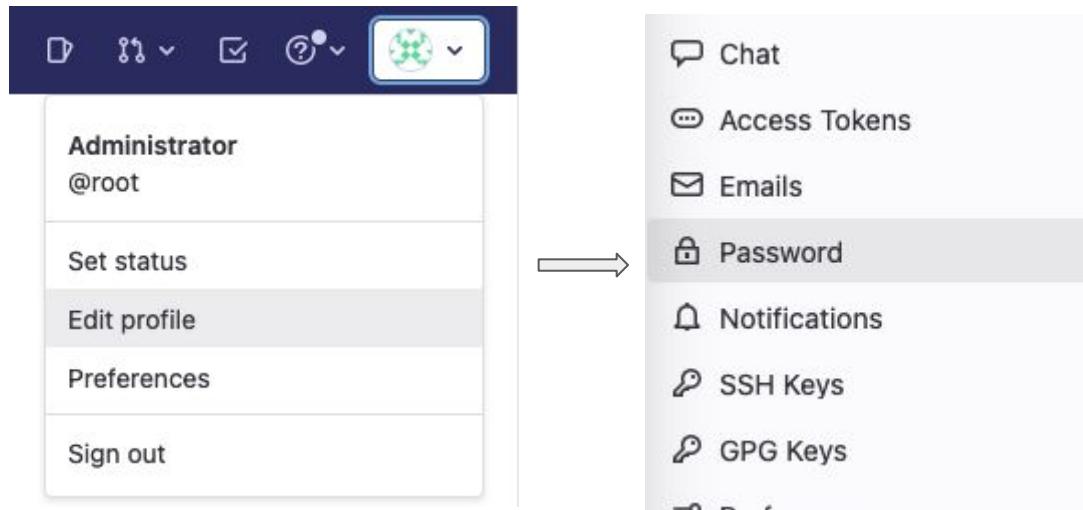


GitLab Pratique

Utilisateurs

Nous allons créer un nouvel utilisateur et changer votre mot de passe root

Il est bien sûr recommandé de le changer le plus vite possible



The screenshot shows the Gitea user profile interface. On the left, a sidebar lists options: 'Administrator' (@root), 'Set status', 'Edit profile' (which is highlighted with a grey background), 'Preferences', and 'Sign out'. An arrow points from the 'Edit profile' option to the right-hand menu. The main menu on the right includes 'Chat', 'Access Tokens', 'Emails', 'Password' (which is highlighted with a grey background), 'Notifications', 'SSH Keys', 'GPG Keys', and 'Preferences'. To the right of the 'Password' item, there is explanatory text: 'min 8 caractères' (minimum 8 characters) and 'puis se reconnecter' (then reconnect).

min 8 caractères

puis se reconnecter

Utilisateurs

Création du user **regularUser** pour le projet

Nous avons besoin d'aller en mode admin pour cela :

pas besoin d'avoir pour le moment une bonne

adresse mail ⇒ notification si le smtp est configuré

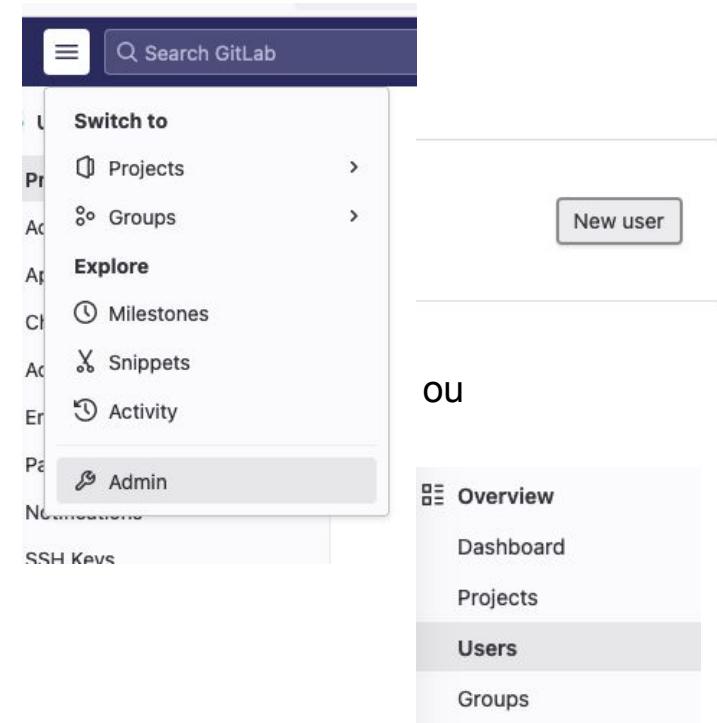
Access level ⇒ Regular

Possibilité d'insérer des liens affichés sur l'accueil

de son compte

Valider mais impossible de se connecter (pas de password) ⇒ cliquez sur Edit sur le résumé de création du compte

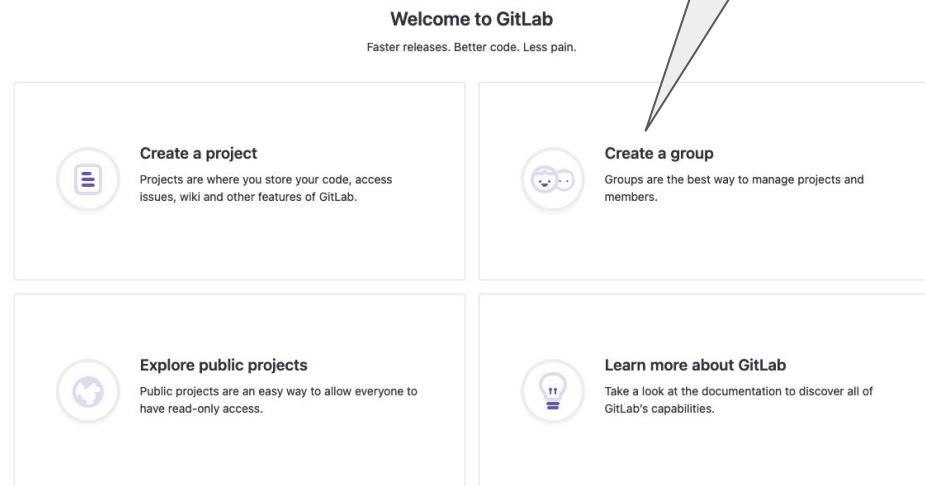
Se déconnecter du compte Admin et se relogger avec ce user



Premier Projet

Création d'un groupe avant de créer un repo. Cela permet :

- de gérer plus facilement les users qui interviendront sur ce repo
- de regrouper aussi les repos



Premier Projet

3 types de visibilité possibles :

- internal : tous les users de cette instance gitlab peuvent accéder aux projets de ce groupe
- on utilisera le private

New group > Create group

Group name

groupeProjet

Must start with letter, digit, emoji, or underscore. Can also contain per

Group URL

http://localhost/

groupeprojet

Visibility level

Who will be able to see this group? [View the documentation](#)

 Private

The group and its projects can only be viewed by members.

 Internal

The group and any internal projects can be viewed by any logged in user.

Premier Projet

On peut y ajuster les notifications

The screenshot shows a project management interface with the following elements:

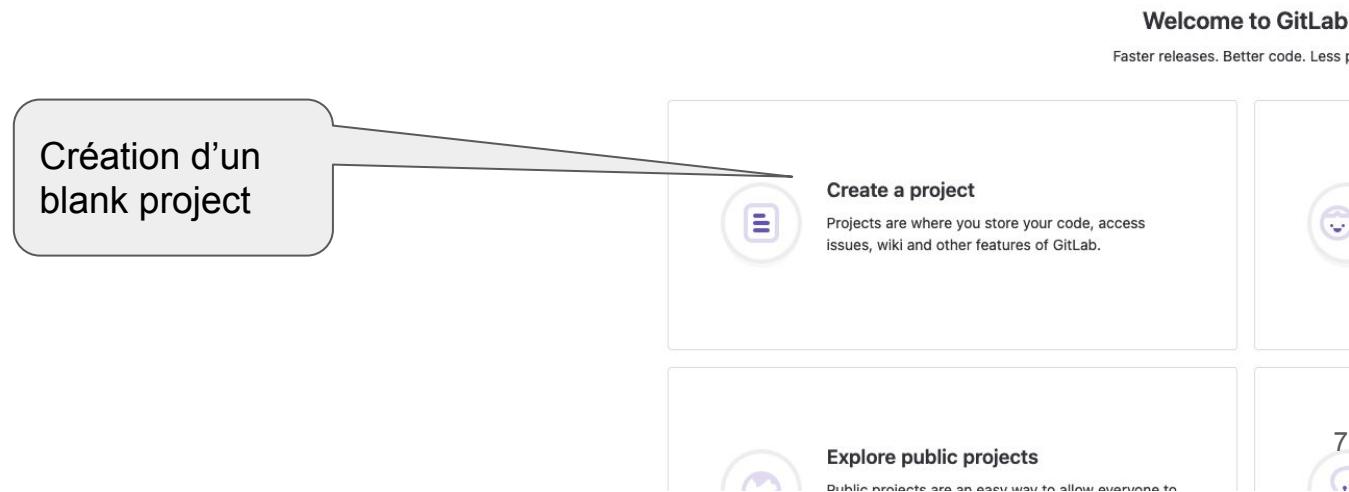
- Header:** A blue bar at the top contains the text "groupeProjet" and "Group ID: 38".
- Navigation:** Below the header are three tabs: "Subgroups and projects" (selected), "Shared projects", and "Archived projects".
- Content Area:** A large white area features a "Create new subgroup" button with a plus sign icon and the text "Groups are the best way to manage multiple projects and members."
- Notification Settings:** On the right, a sidebar titled "Notifications" lists three options:
 - Global:** "Use your global notification setting".
 - Watch:** "You will receive notifications for any activity".
 - Participate:** "You will only receive notifications for threads you have participated in".
- Buttons:** At the bottom of the sidebar are "New subgroup" and "New project" buttons.

Premier Projet

On peut modifier la configuration du groupe dans admin area ⇒ groups

Utile pour y ajouter des utilisateurs ⇒ notre user en fait déjà partie car c'est lui le propriétaire du groupe

Une fois dans le groupe, nous allons pouvoir créer un projet ou depuis la page d'accueil



Premier projet

The screenshot shows a user interface for creating a new project. At the top, there are fields for 'Project name' (set to 'firstProject'), 'Project URL' (set to 'http://localhost/'), and 'Project slug' (set to 'firstproject'). Below these, a 'Visibility Level' section is highlighted with a callout bubble. It contains three options: 'Private' (selected), 'Internal', and 'Public'. The 'Private' option is described as requiring project access. The 'Internal' option is described as preventing external access. The 'Public' option is described as allowing anyone to access the project. A dropdown menu is open over the 'Private' option, showing a search bar and a list of groups: 'groupeprojet' (which is selected) and 'regularUser'. A tooltip for 'groupeprojet' states: 'If this project is part of a group, access is granted to all members of the parent group.' A tooltip for 'regularUser' states: 'Access is granted to all users except external users.' Another callout bubble points to the 'Private' visibility level, stating: 'Les droits affectés sont ceux du groupe. Les autres ne sont pas autorisés. Si public au départ, on pourrait les affiner.'

Project name
firstProject

Project URL
http://localhost/
Project slug
firstproject

Visibility Level ?
 Private Project access must be granted.
 Internal This project cannot be internal to another project.
 Public The project can be public.

Want to organize several dependent projects? [Create a group](#).

Search

Groups
groupeprojet

Users
regularUser

If this project is part of a group, access is granted to all members of the parent group.

Access is granted to all users except external users.

Want to organize several dependent projects? [Create a group](#).

Visibility Level ?

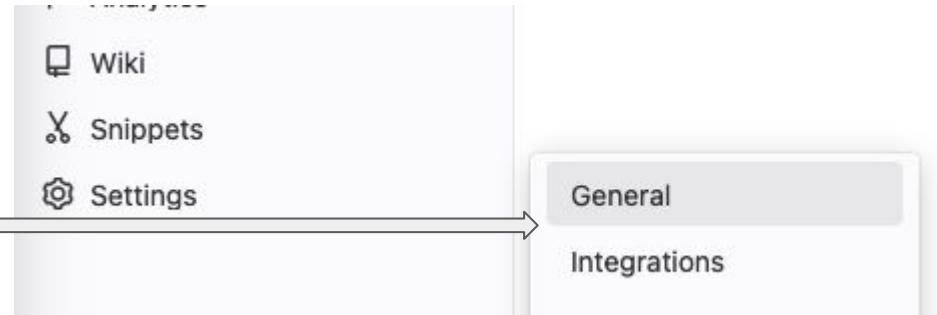
Private Project access must be granted.
 Internal This project cannot be internal to another project.
 Public The project can be public.

Les droits affectés sont ceux du groupe. Les autres ne sont pas autorisés. Si public au départ, on pourrait les affiner

On laisse le README en initialisation

Premier Projet

Dans les settings du projet



Project name

firstProject

Topics

docker X ansible X

Ajouts de topics

Ajout aussi d'un avatar

Premier Projet

Ajout de ticketing possible avec les issues

Tous ces comportements par défaut sont configurables dans le dossier admin

En désactivant des éléments, vous libérez des entrées sur le menu de gauche

Possibilité de squash commits : regrouper les commits au moment d'une merge request pour éviter d'en avoir trop et simplifier la lecture pour les users

Ou demande du succès du pipeline pour autoriser un merge

Visibility, project features, permissions

Choose visibility level, enable/disable project features and their emoji.

Project visibility

Manage who can see the project in the public access directory

Private

Only accessible by [project members](#). Membership must be invited.

Additional options

Require authentication to view media files

Prevents direct linking to potentially sensitive media files

Issues

Flexible tool to collaboratively develop ideas and plan work



Premier Projet

Dans **Advanced**, plutôt que de supprimer un projet, il est recommandé de l'archiver

Possible de transférer le projet au sein d'un autre groupe

Nous supprimons ce projet

Archive project

Archiving the project makes it entirely read-only.
repository cannot be committed to, and no issues can be created.

[Archive project](#)

This project is **NOT** a fork. This project is owned by:

Enter the following to confirm:

groupeprojet/firstproject

|

Templates

Create new project

ct
store your
ng other



Create from template

Create a project pre-populated
with the necessary files to get you
started quickly.

New project > Create from template

Learn how to [contribute to the built-in templates](#)

Built-in 28

Ruby on Rails

Includes an MVC structure, Gemfile, Rakefile, along with

Spring

Includes an MVC structure, mvnw and pom.xml to help y

NodeJS Express

Includes an MVC structure to help you get started

iOS (Swift)

Name
📁 .mvn/wrapper
📁 src
📦 .gitignore
🐳 Dockerfile
📝 README.md
📄 mvnw
📄 mvnw.cmd
📄 pom.xml

En revanche, avec cette option, les niveaux de visibilité sont tous open

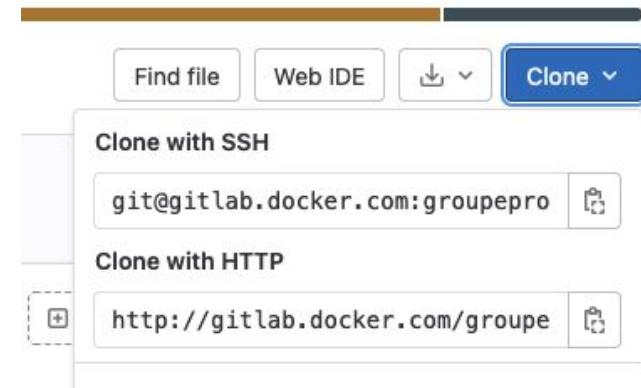
⇒ certainement bug car après création la visibilité revient en private

Clone de dépôt = travail déporté avec Git

Par défaut Gitlab nous propose l'ajout d'une clé SSH

3 types d'authentification possibles :

- * ssh
- * https : variables d'environnement
 `http://${GIT_USER}:${GIT_PASSWORD}@gitlaburl`
- * token (Personal/Project)
 `https://<token-name>:<token>@gitlaburl`



Si var d'env ⇒
sourcer les var pour
éviter tout chose en
clair

Clone de dépôt = travail déporté avec Git

Avec le mode **SSH** après avoir créé une clé en local, aller dans la partie user ⇒ edit profile :

-  Password
-  Notifications
-  SSH Keys
-  GPG Keys

```
$cat .ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQAB
ZTk75R/ciSe+GbK6AyXlHAEe+A3cJzt
3Kn1H4Y709ggTT5zS95N1ck1x5HN&c\
```

Attention au titre de la clé, mettre par exemple le nom du laptop

```
git clone git@localhost:groupeprojet/spring.git
Clonage dans 'spring'...
remote: Enumerating objects: 27, done.
remote: Total 27 (delta 0), reused 0 (delta 0), pack-reused 27
Réception d'objets: 100% (27/27), 47.28 Kio | 47.28 Mio/s, fait.
```

Clone de dépôt = travail déporté avec Git

```
$ls -la  
total 64  
drwxr-xr-x 11 adam staff 352 13 fév 11:59 .  
drwxr-x---+ 152 adam staff 4864 13 fév 11:59 ..  
drwxr-xr-x 12 adam staff 384 13 fév 11:59 .git  
-rw-r--r-- 1 adam staff 269 13 fév 11:59 .gitignore  
drwxr-xr-x 3 adam staff 96 13 fév 11:59 .mvn  
-rw-r--r-- 1 adam staff 177 13 fév 11:59 Dockerfile  
-rw-r--r-- 1 adam staff 557 13 fév 11:59 README.md  
-rw-r--r-- 1 adam staff 6468 13 fév 11:59 mvnw  
-rw-r--r-- 1 adam staff 4994 13 fév 11:59 mvnw.cmd  
-rw-r--r-- 1 adam staff 1414 13 fév 11:59 pom.xml  
drwxr-xr-x 4 adam staff 128 13 fév 11:59 src  
$less .git/config
```

git@ :
utilisation de
la clé SSH

```
[remote "origin"]  
url = git@localhost:groupeprojet/spring.git  
fetch = +refs/heads/*:refs/remotes/origin/*
```

Clone de dépôt = travail déporté avec Git

Suppression du repo local

Maintenant authentification en user/password de la connexion avec http , retourner sur le repo (ne fonctionne pas si 2FA active)

```
$git clone http://localhost/groupeprojet/spring.git  
Clonage dans 'spring'...  
Username for 'http://localhost':
```

Possibilité de mettre son username et password directement dans la CLI mais A NE PAS FAIRE car l'history pourrait etre consultee ⇒ le mettre dans un fichier .env par exemple

Clone de dépôt = travail déporté avec Git

```
#!/bin/bash
export GIT_USER="regularUser"
export GIT_PASSWORD="eceq8Erbx
~
```

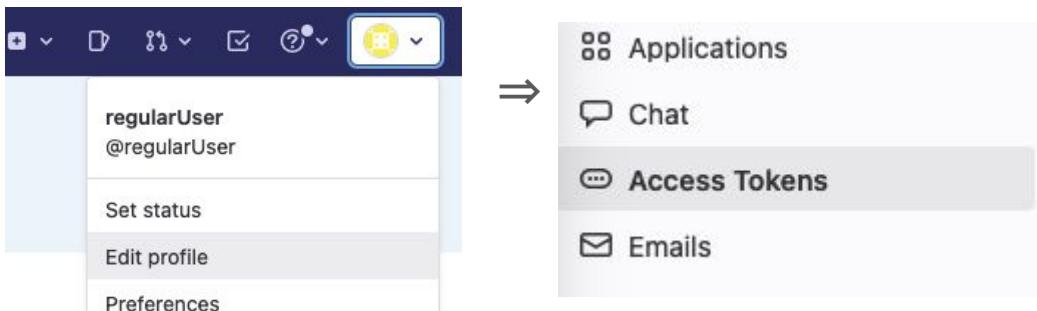
```
$sudo vim .env
Password:
$source .env
```

Attention aux droits de ce fichier .env

```
$env | grep GIT
GIT_PASSWORD=eceq8Erbx
GIT_USER=regularUser
```

Clone de dépôt = travail déporté avec Git

Avec un token ⇒



```
$git clone http://regularUser:glpat-dy2gkfAweRtwYBQACzq8@localhost/groupeprojet/spring.git
Clonage dans 'spring'...
remote: Enumerating objects: 27, done.
remote: Total 27 (delta 0), reused 0 (delta 0), pack-reused 27
Réception d'objets: 100% (27/27), 47.28 Kio | 47.28 Mio/s, fait.
```

Token name
spring-project
For example, the application using the token information for the name of the token, as it is called.

Expiration date
2023-03-15

Select scopes
Scopes set the permission levels granted to this token.

- api**
Grants complete read/write access to the registry, and the package registry.
- read_api**
Grants read access to the API, including the package registry.
- read_user**
Grants read-only access to the authentication endpoint which includes username, public email, and endpoints under /users.
- read_repository**
Grants read-only access to repositories and the Repository Files API.
- write_repository**
Grants read-write access to repositories via the API).

Create personal access token

Clone de dépôt = travail déporté avec Git

Or ici c'est un token pour ce user ⇒ bloquant si la personne quitte la société mieux de le mettre de la même manière mais pour le groupe dans le projet

The screenshot shows the 'Access Tokens' section of the GitLab settings. On the left sidebar, 'Access Tokens' is highlighted. The main page title is 'groupeProjet > Spring > Access Tokens'. A search bar is present. The 'Project Access Tokens' section includes a description about generating tokens for applications needing access to the GitLab API, and a note about using them with Git for HTTP(S) authentication. It features a form to 'Add a project access token' with fields for 'Token name' and 'Expiration date' (set to 2023-03-15). A placeholder text explains the token's purpose.

En général, le token est utilisé pour le travail sur projet et le SSH pour des connections pour dev

Issues - partie Agile dans la doc

<https://docs.gitlab.com/>

Concurrence avec Jira -

Possible de le connecter avec Jira aussi

Les issues sont classées par projets, visibles sur le menu de gauche



Agile with GitLab

Manage your work with built-in agile features.

Issues

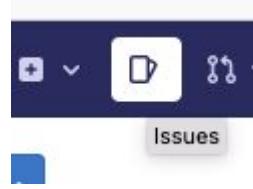
Issue boards

Labels

The screenshot shows the left sidebar of a GitLab interface. At the top is a header with the project name 'Repository'. Below it is a list of main categories: 'Issues' (selected), 'Merge requests', 'CI/CD', 'Security & Compliance', and 'Deployments'. To the right of this list is a vertical column with four items: 'List', 'Boards', 'Service Desk', and 'Milestones'. Each item has a small icon next to its name.

Category	Status
Issues	0
Merge requests	0
CI/CD	
Security & Compliance	
Deployments	

Issues



2 angles d'attaque: par le projet ou le user

Elles sont filtrables et on peut s'y abonner via un flux rSS

4 parties pour les tickets :

- List : liste des tickets
- Boards : vision sous forme de tableau des avancées des tickets
- Service Desk : permet l'utilisation via solution externes (mails par exemple)
- Milestones : tracking de certains tickets éventuellement couplé aux MR

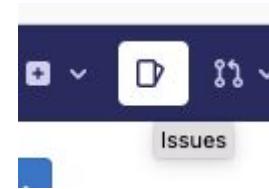
Issues

- Boards : pratique pour voir l'évolution des tickets, ou ce qu'il reste en backlog à traiter ou à gérer
- Service Desk : son objectif est de permettre l'utilisation de gitlab indirectement, pour ceux qui sont externes au projet (ex un autre service) et qui veulent soumettre des tickets, en leur fournissant un email
- Milestones : ca permet de regrouper un ensemble de tickets qui ont des points communs ou des liens dans leur résolution via la merge requests (c'est plus un tracking de tickets)

Issues

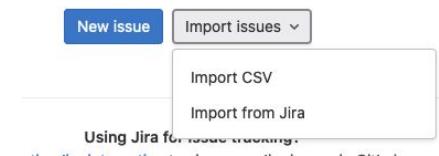
En tant que user, nous allons créer un ticket ou en se rendant directement dans le projet avec new issue avec import possible depuis Jira

Creation de ticket possible via l'api GitLab avec un curl
ou de l'affecter à un groupe de personnes lors d'une automatisation avec script



Create on ideas, solve problems, a

Learn more about issues.



Issues

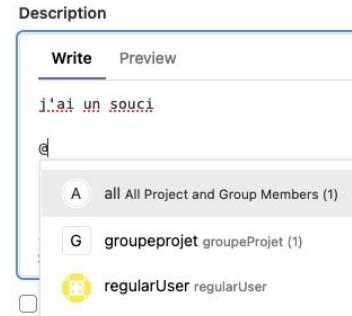
Intervention plus urgente avec Incident

Format Markdown donc introduction possible du code d'où preview utile http://localhost/help/user/project/quick_actions

Il existe des mots-clés : <http://localhost/help/user/markdown.md>

<http://localhost/help/user/markdown.md#gitlab-specific-references>

@ pour pinger un autre user



Type [?](#)

Incident

Select [?](#)

Issue

Incident

Issues

L'@ ne lui assigne pas le ticket mais pour lui donner l'info

Si notre code est très long :



Un ticket proprement formate est une aide importante à la résolution et pour les relations

Ces crochets permettent de lancer des tasks et les cocher signifie les valider

Description

Write Preview

j'ai un souci

@regularUser as-tu change qqch ?

```
<details><summary>Titre du code</summary>
`voici le code ici`
</details>
```

```
<details><summary>Titre du code</summary>
<code>voici le code ici</code>
</details>
```

- [x] verification pour les tests qualite
- [] verification tests de charge

Issues

Pour intégrer des liens vers les commits avec un #

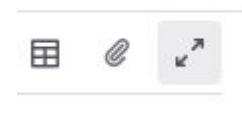
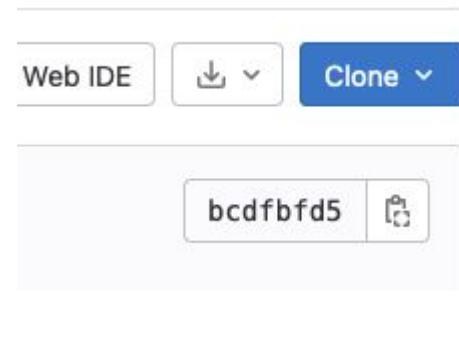
Retourner sur le repo et copier le no de commit

Ecriture possible également en full screen

Intégration d'un tableau



Possibilité de rendre ce ticket uniquement lisible par les membres du projet



Issues

Possibilité de l'annexer à un milestone si déjà créé

Les labels servent à organiser les tickets par zone (production, version, environnement, équipe...)

Nous allons nous assigner le ticket



Ajout dans la todo list

Ce ticket est en format open et est donc dans le board dans cette partie

Et aussi dans le milestones si déjà créé

Issues

On peut ajouter d'autres éléments comme le time tracking (tps nécessaire) dans les quick actions avec /estimate

regularUser assigned to @regularUser 1 hour
Feb 13, 2023 1d

Write Preview
`/estimate 1d`

et pour indiquer le temps passé dessus : /spend

Time tracking
Spent 1h
Time tracking report

Faciliter la rédaction des tickets avec les templates

Créons un nouveau projet pour tester

http://localhost/help/user/project/description_templates

Il faut un dossier **.gitlab** dans le repo avec à l'intérieur un répertoire **issue_templates**

Puis dans ce dernier, nous pourrons y insérer nos templates.

New project > Create blank project

Project name

Project URL

Want to organize several dependent projects under the same namespace?

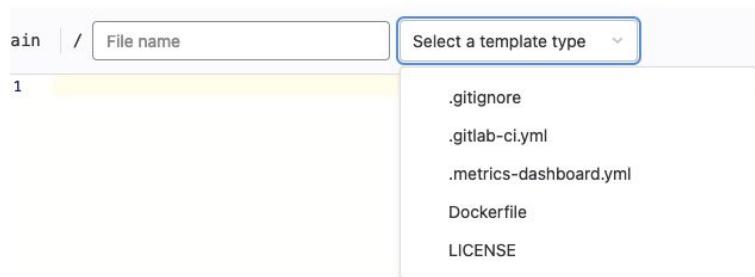
Visibility Level ②

Private
Project access must be granted explicitly to each user. If this project is made public, it will be visible to anyone.
 Internal

Faciliter la rédaction des tickets avec les templates

y créer un fichier en format markdown

Il existe des templates mais pas utiles dans notre cas



Issue template / + ▾

This directory

Created 5 minutes ago

New file

Upload file

New directory

Create New Directory

Directory name

Commit message

Target branch

Faciliter la rédaction des tickets avec les templates

Rédiger ensuite n'importe quel template avec markdown (mettre un enter entre chaque lignes)

Exploitation dorénavant possible depuis la création d'un ticket

Description

Choose a template

Choose a template

Filter

Project Templates

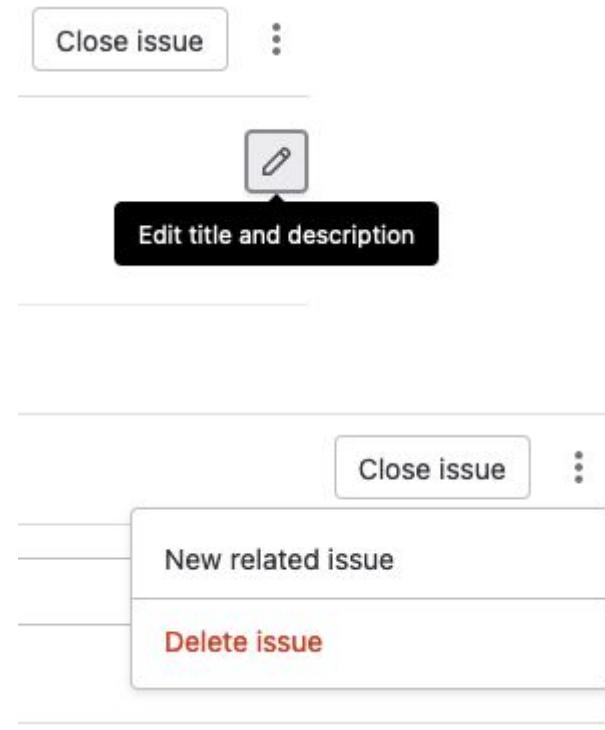
bugs

New file

```
main | / bugs.md
1 ## Description de l'incident
2
3 Date:
4 Duree:
5 Environnement:
6
7 /assign @
8 /estimate
9
```

Faciliter la rédaction des tickets avec les templates

Pour supprimer une issue, cliquer sur les trois points à droite



Labels

Les labels sont disponibles sur les **issues** mais également sur les **merge requests**

Outil utile surtout pour l'organisation du travail

Se rendre dans un repo ⇒ Manage ⇒ Labels ⇒ Supprimer les labels par défaut (générés sinon directement avec l'accueil du repo dans labels)

ça ne supprime pas les tickets qui y sont rattachés

Possibilité de changer la couleur et le nom. Lorsque l'on génère des labels dans le repo, ils ne sont dispo que dans le repo pour le moment

Labels

En cliquant sur l'étoile, cela nous permet de prioriser le label s'il est utilisé fréquemment

Possibilité de croiser les labels avec notre compte en y souscrivant, pour recevoir des notifications à chaque fois que des notifications sont ajoutées à ce label

Se rendre dans un projet ⇒ New Label ⇒
Création de

différents labels de niveau (N1, N2, N3 ...)

Labels can be applied 1 requests to categorize

You can also star a label to make it

New label Generate

Other Labels

N1	groupeProjet / Spring
N2	groupeProjet / Spring
N3	groupeProjet / Spring

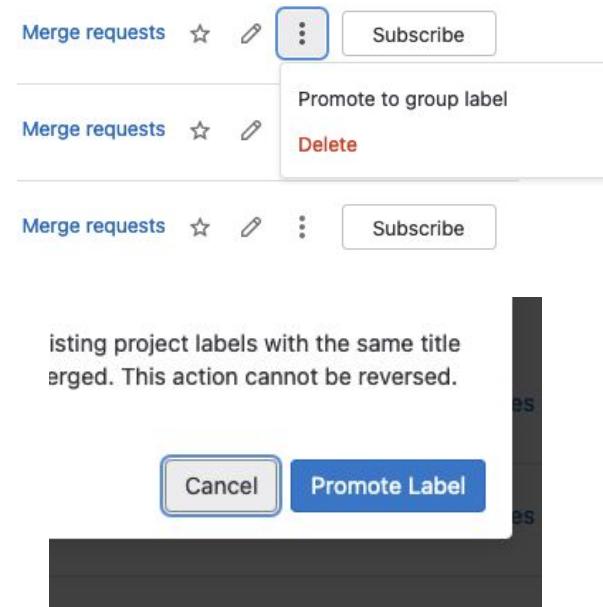
Labels

ATTENTION : avec le temps les labels peuvent devenir génériques a differents depots

Donc possibilité de le promouvoir au groupe

Retourner dans le groupe et vérifier dans
les labels

Important de trouver la bonne échelle



Labels

On va aussi pouvoir croiser les labels et utiliser plusieurs labels pour les issues

Aller dans Issues ⇒ choisir une issue et éditer la partie labels à droite. Nous pouvons y ajouter plusieurs labels (gravité et domaine)

Cela nous permet de filtrer les issues en fonction du label en cliquant notamment sur le label

N3

groupeProjet / Spring

env::development

groupeProjet / Spring

env::production

groupeProjet / Spring

env::development

groupeProjet / Spring

[Issues · Merge requests](#)



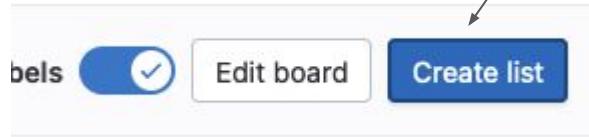
Labels

Ils seront également utiles lorsque nous voudrons utiliser les boards

Aller dans Plan ⇒ Issue boards

Possibilité de créer une nouvelle liste

dans le board



ou un nouveau board

(enlever les listes)

A screenshot of the Jira interface. At the top right, there is a search bar with the text 'Open' and a count of '1'. Below it is a card for an issue titled 'First Issue' with labels 'N2' and 'env::production', and status '#1 Yesterday' and '1d'. In the bottom right corner of the card, there is a yellow circular icon with a question mark. On the left side, there is a sidebar with a dropdown menu set to 'Development', a 'Switch board' button, a search bar, and a list of boards: 'Development' and 'Create new board'. A black arrow points from the text '(enlever les listes)' to the 'Create new board' option in the sidebar.

Boards

en enlevant les listes, nous avons le choix de limiter les nouvelles à un label particulier

The image shows a screenshot of a Jira board interface. On the left, a modal window titled "New list" is open, showing a "Scope" section with a dropdown menu set to "env::development". A black arrow points from the text "enlevant les listes, nous avons le choix de limiter les nouvelles à un label particulier" to this scope dropdown. Below the modal, the main board view displays four columns. The first column has a blue header "env::development" and contains a summary icon with "0" and a plus sign. The second column has a green header "N1" and contains a summary icon with "0" and a plus sign. The third column has a red header "N3" and contains a summary icon with "0" and a plus sign. The fourth column is empty. Each column has a small "v" icon at the top left.

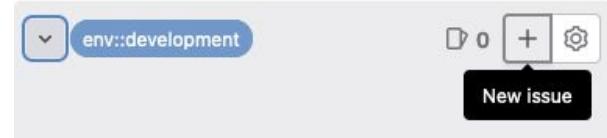
Pratique pour les rendus visuels en reunion

Boards

On peut aussi créer des issues via le board

Peu fournie mais bien sur customisable par la suite

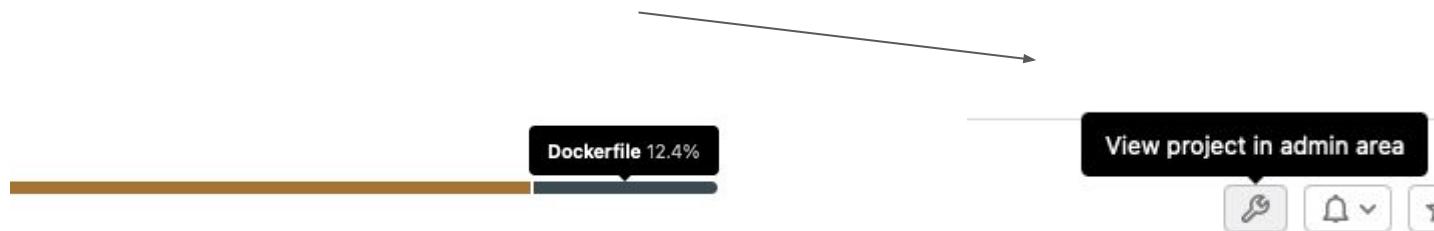
Plus tard, nous verrons comment les labels permettent de gérer les merge requests



Repos

Informations sur le dépôt en admin :

Les différentes couleurs indiquent les langages utilisés dans le repo



Repos ⇒ dans menu Code

Commit :

- ensemble d'ajouts/modifications/suppressions dans un dépôt git
- points de retours possibles
- ajout d'un commentaire pour identifier ce point

Branches :

- visibles depuis le menu Repository
- en créant une graphiquement, nous pouvons la nommer et choisir depuis laquelle elle démarre

Repos

Flow :

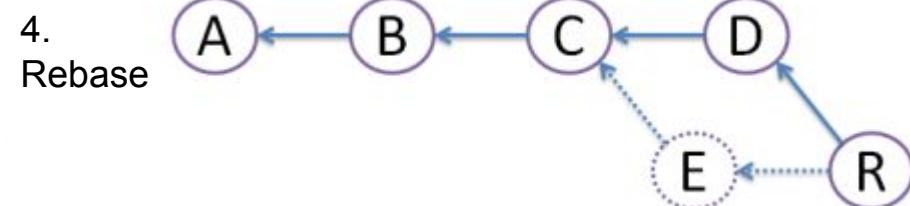
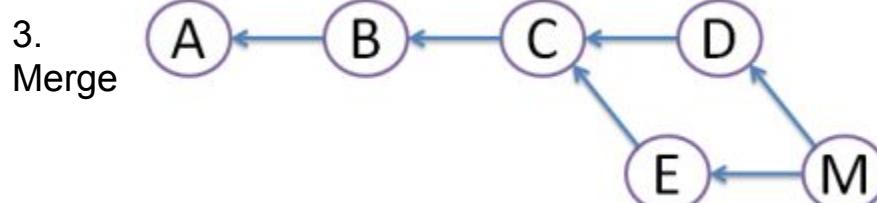
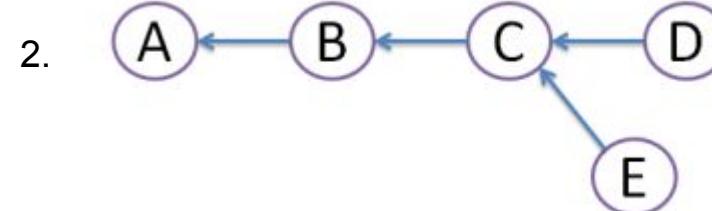
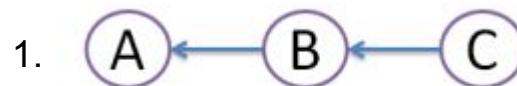
- Parcours de développement du code jusqu'à la prod ou depuis un bug vers sa résolution en fonction des environnements (1 - branch feature, 2 - branch dev, 3 - branch preprod, 4 - branche prod)
- Peut exister plusieurs flows
- gitflow, gitlab flow....

Tags :

- se rapporte toujours au même objet
- permet de faire sortir du lot un commit particulier contenant une feature intéressante

Repos

Merge/Rebase :



!! Rebase doit rester local

Repos

Revert/Reset : retours en arrière

- Revert : annuler les effets du commit précédent
- Reset : il s'agit simplement de régler HEAD sur un certain commit

```
A <- B <- C  
^ HEAD
```

```
A <- B <- C <- B'  
^ HEAD
```

```
A <- B <- C  
^ HEAD
```

```
A <- B <- C  
^ HEAD
```

Flows

Très important pour le travail en équipe

```
git tag  
git tag -a v0.1.0 -m "message de commit explicite"  
git push v0.1.0
```

Il existe plusieurs types de flows

Rôles et permissions

Assez intuitif

doc : <https://docs.gitlab.com/ee/user/permissions.html>

Matrice très importante donnant beaucoup de détails sur les différences entre les rôles et leurs permissions

5 rôles sont possibles

Guest	Reporter	Developer	Maintainer	Owner
-------	----------	-----------	------------	-------

Cela se gère au niveau des projets ou des groupes

Puis les autorisations sont classées en fonction des services

Rôles et permissions

Le reporter apporte des choses qui ont un impact minimal sur le projet

Comme créer des snippets (morceaux de codes ou de texte à partager)

On attend en revanche du dev de mettre en place un wiki

Le maintainer est là pour le suivi qualité du projet

Matrice disponible en local également

<http://localhost/help/user/permissions>

Il faut également y intégrer le type de projet cad s'il est interne, public ou privé avec les nos entre parenthèses

Rôles et permissions

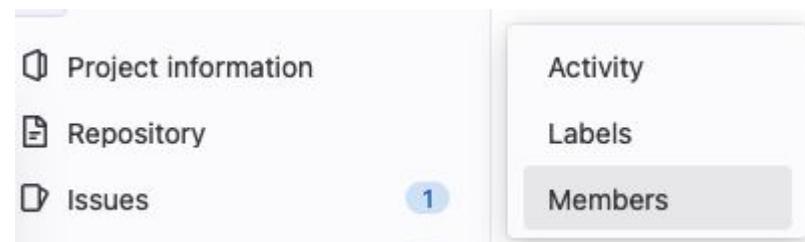
Types de projet:

- Private : cloné et vu par les membres du projets sauf guests
- Internal : cloné par les users authentifiés
- Public : tout le monde

Ajout membres à partir d'un rôle de maintainer/owner/admin

Project information > members

Group information > members



Rôles et permissions

En admin aller en Overview ⇒ Users

Projet ou groupe :

0 - Rappel : private / internal / public

A - ajout d'un utilisateur simple à un projet * date expiration (pour mission par exemple) * import d'un autre projet * choisir role (limite max selon le user, ici guest) * classement des utilisateurs/tri * source (possible import de users venant d'un autre projet) - Faire évoluer les droits du user créé et vérifier son accès et/ou son action possible sur le repo

B - ajout d'un user au groupe (créer un autre groupe et lui ajouter un membre)

C - ajout d'un groupe à un projet/group (en récupérant tous les users d'un groupe avec un level max de ...), en retournant dans admin→users→choisir un user→groups and projects→groupe ajoute

D - si suppression du groupe membre > les users restent

Account Groups and projects SSH keys

Groups

groupeProjet – access to 2 projects

Rôles et permissions

En important un groupe, on le retrouve pas les users dans members d'un projet, donc besoin de re-lister les membres du groupe pour avoir la liste complete des membres

Possibilité de trier

The screenshot shows a user interface with a search bar, an account selection dropdown, and a sorting icon. A context menu is open over a table row, listing sorting criteria: Account (checked), Access granted, Max role, Created on, Last activity, and Last sign-in. The 'Created on' option is highlighted with a red border.

Created on	Access granted	Max role	Created on	Last activity	Last sign-in
12 Feb, 2023	✓ Account	Access granted	Max role	Created on	Last activity
13 Feb, 2023	Access granted	Max role	Created on	Last activity	Last sign-in

Rôles et permissions

Possibilité également d'importer les users selon un projet

- 1 - Créer un projet avec un user
- 2 - Retourner dans un second projet
- 3 - Importer un membre via ce 1ier projet

Flows

Révision des bases git permettant de poser des briques pour voir les merge request par la suite.

Semantic version :

- major : est un numéro de version où vous avez introduit des modifications avec rupture (les modifications de votre nouvelle version ne sont PAS compatibles avec les versions précédentes) ;
- minor : est un numéro de version compatible avec les versions précédentes ;
- patch : est un incrément pour une correction de bogue ou un correctif effectué sur votre logiciel.

Flows

Pour travailler les flows, les notions suivantes sont exploitées :

- ligne de développement verticale d'un projet
- features = temporaires le temps de développer
- branches permanentes : main / preprod / staging /
- branches temporaires : development/features
- organisation des branches en flow

Flows

Le flow est la manière choisie pour faire évoluer une branche à une autre, pour lui faire atteindre finalement la production

Il existe différents types d'évolution : fix/bug/development...

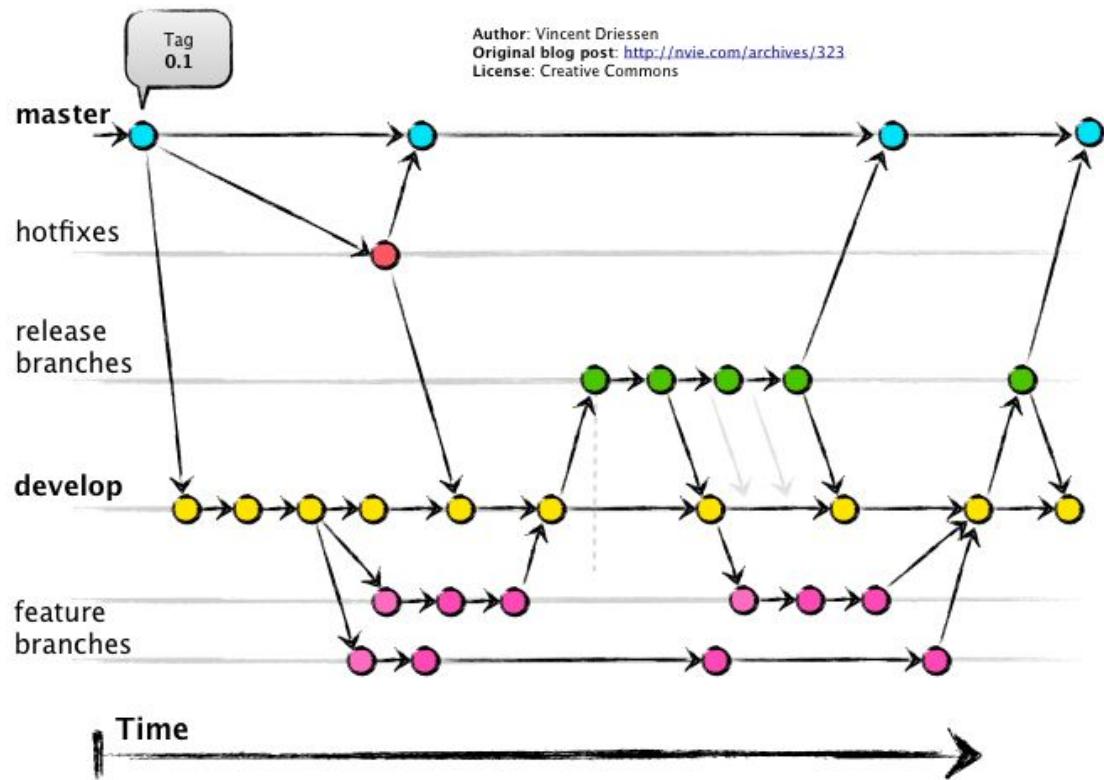
Il existe également différents types de flow

GitFlow

C'est un peu la suite de l'évolution depuis

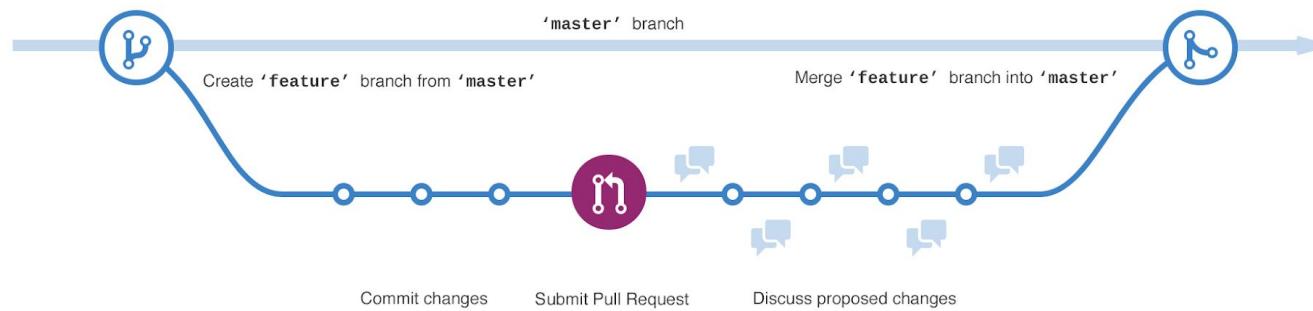
gitflow

<https://danielkummer.github.io/git-flow-cheatsheet/> ⇒ très rigide, bcp de merges et finalement historique illisible



Gitlab Flow

puis github flow ⇒ on part simplement de master(ou main) et tout s'opérait via des branches features, puis dès que le job était fini, on faisait des propositions de pull requests



Gitlab Flow

Avec gitlab flow, on garde ce principe :

- une branche master
- développement à travers une branche feature
- puis demande non pas de pull request mais de merge request
- on va pouvoir les travailler et les valider
- Il faut y rajouter la notion de merge request draft, ce qui se nommait avant le “working progress merge request” : indiquer qu’une merge request est en cours de travail, et des commits la complètent encore”
- Donc les reviewers ou contrôleurs QA peuvent vérifier l’avancement

Gitlab Flow

Évite de balancer d'un coup un travail avec 50 commits par exemple, lorsqu'elle est validée ⇒

- sonarqube
- build
- tests unitaires...

L'autre point : possibilité d'orienter nos branches. Dans Github flow, on ne sait quelle direction nous prenons : est-ce de la prod, de la staging... ?

Ici, nous choisirons que la main est une staging, donc des tests (intégration et livraison continue) sur un serveur de staging

Gitlab Flow

On va essayer pour éviter d'alourdir l'historique, donc de limiter les merge requests ou de les rassembler.

Mais principe de base : un ticket amène une merge request

Lancer un nouveau projet gitlab-flow avec des fichiers texte vides :

Name
1.txt
2.txt
README.md
chap-2.1.txt
chap-2.2.txt
chap-2.3.txt
chap-2.4.txt
chapter3-1.txt
chapter3-2.txt
chapter3.txt

Gitlab flow

```
git checkout -b 'features/chapter-003'
```

```
echo > chapter3.txt
```

```
git add . // ou git add --all
```

```
git commit -m "ajout chapter003"
```

Maintenant nous allons avoir une issue

Gitlab Flow

Issues ⇒ New Issue

New Issue

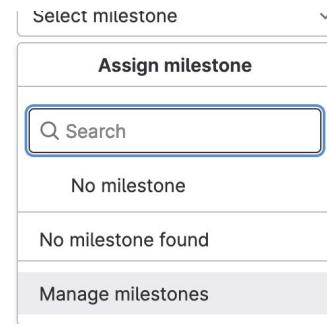
Title (required)

Prepare chapter 3 first version

Add [description templates](#) to help your contribu

Type 

Milestone : grandes étapes du projet



Title	Chapter 003
Start Date	2023-02-16
Due Date	2023-02-23

[Clear start date](#)

Puis générer issue

Gitlab Flow

Issues ⇒ New Issue

```
$echo > chapter3-1.txt
```

```
git add . // ou git add --all
```

```
git commit -m "chapter added. Fixes #1"
```

```
git push -u origin features/chapter-003
```

Utiliser le Fixes et # qui seront reconnus lors de la merge request

Puis il propose la possibilité de faire une merge request

```
remote: To create a merge request for features/chapter-003, visit:  
remote: http://home.gitlab.com/root/gitlab-flow/-/merge_requests/new?merge_request%5Bsource_branch%5D=features%2Fchapter-003
```

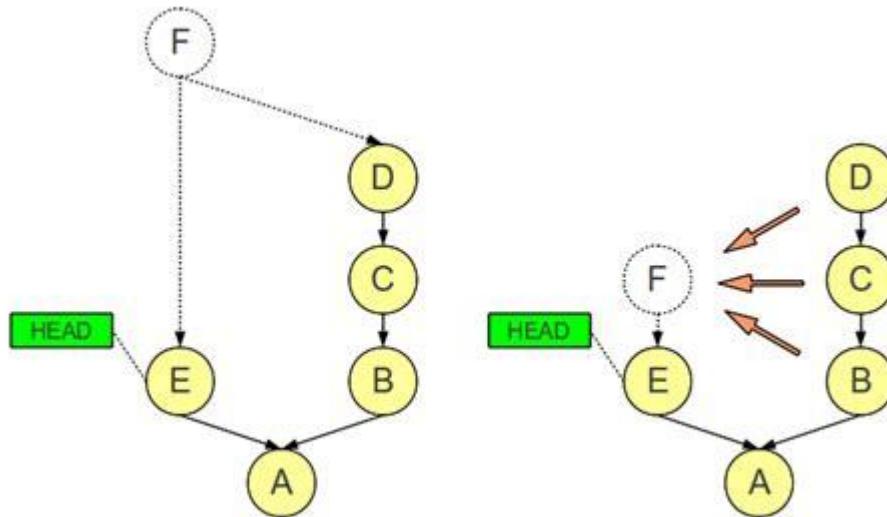
Gitlab Flow

The screenshot shows a 'New merge request' form on a GitLab page. At the top, there's a navigation bar with a shield icon, a lock icon, and the URL 'home.gitlab.com/root/gitlab-flow/-/merge_requests/new?merge_request[source_branch]=features%2Fchapter-003'. Below the URL is a dark header bar with the letters 'ab' and a '/' symbol. The main content area has a light background. It starts with a 'New merge request' section. Underneath it, there's a 'Merge options' section containing two checked checkboxes: 'Delete source branch when merge request is accepted.' and 'Squash commits when merge request is accepted.' with a question mark icon. At the bottom are two buttons: a blue 'Create merge request' button and a white 'Cancel' button.

choisir assigned et milestone

Et nous squashons les commits (voir page suivante)

Gitlab Flow



ce que nous faisons en squashant
nos commits

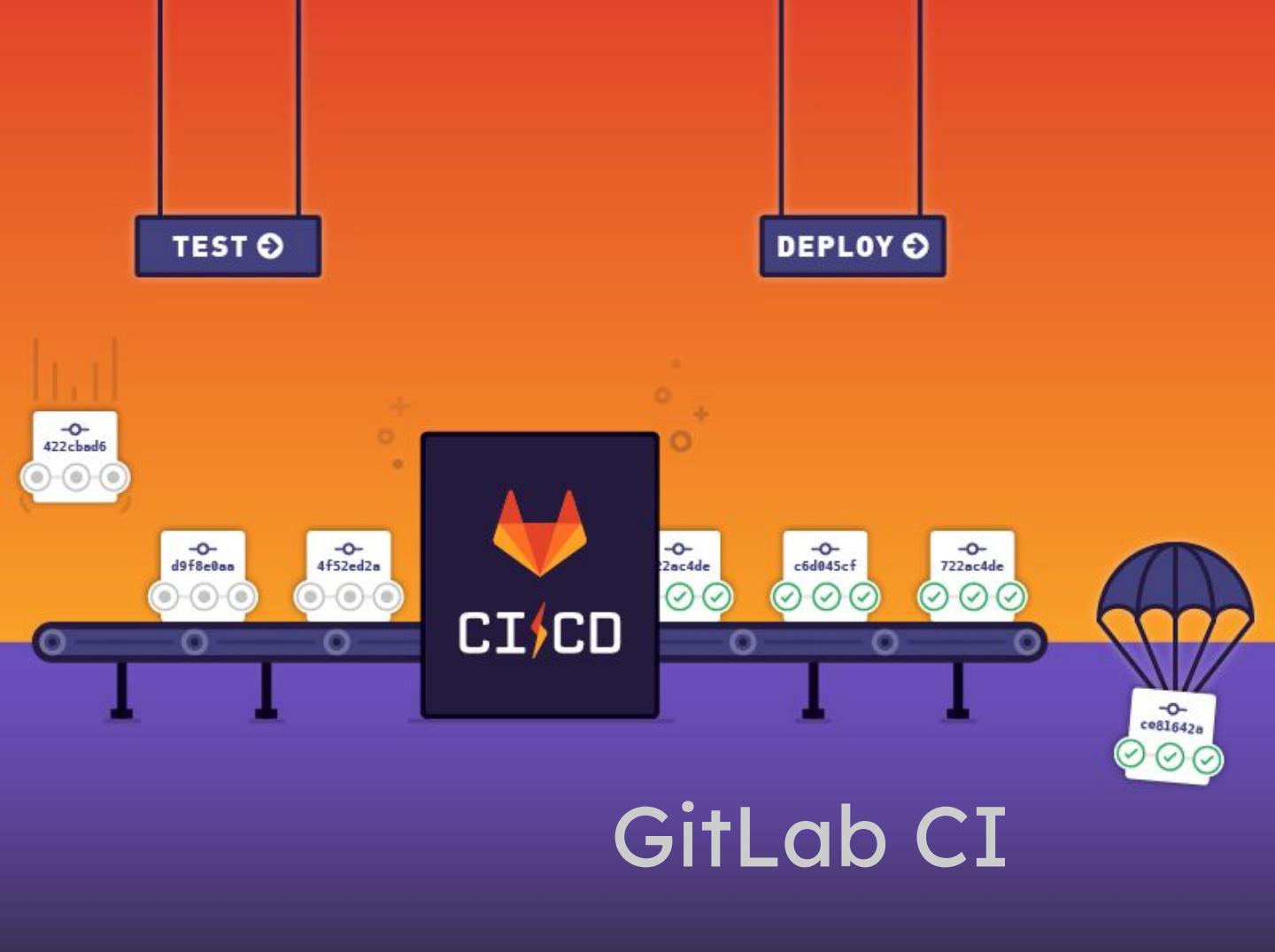
Gitlab Flow

```
echo > chapter3-2.txt
```

```
git add . // ou git add --all
```

```
git commit -m "chapter added 2.. Fixes #1"
```

```
git push //ajout de fonctionnalité
```



gitlab-ci

Continuous Integration :

- * action sur push du nouveau code (applicatif ou non)
- * aider les développeurs dans la phase de développement (code quality...)
- * création de l'applicatif final (binaire...)
- * vérification du code (tests unitaires...) pour trouver d'éventuelles régression
- * découverte le plus tôt possible des bugs et de les reporter loin de la production

gitlab-ci

Continuous Delivery:

- tests complémentaires dans un environnement proche de la production (comme par exemple sur une DB, tests de fonctionnalité, tests de charges)
- permet de releaser (versionner) avant mise en production : permet de garder des environnements pour simuler des environnements de production
- fournit le livrable pour la production : envoyé dans un repo de packages comme nexus par exemple

gitlab-ci

Continuous Deployment :

- déploiement automatisé en production du package hébergé dans nexus

Pipeline : permet de dérouler toutes ces étapes

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

gitlab-ci : Runners

Dans le cadre de gitlab-ci, nous devons parler des runners.

Afin de faire tourner ce process qui analysera et tester notre code, nous avons besoin de ressources.

- Elles seront fournies par soit des serveurs, des VM, soit des containers (ou k8s)...
- Ces runners lanceront les jobs les uns derrière les autres
- On pourra y installer des applicatifs (docker, ansible, maven...). Cela permet de typer les différents runners, différenciés avec des tags, pour aider à faire son choix

gitlab-ci : Runners

- il est TRÈS fortement recommandé d'utiliser une machine différente du host gitlab (sur le même principe que Jenkins), pour éviter de faire un mélange des ressources. Notion de scalabilité sur les runners
- Ils sont développés en Go, consomment peu de ressources à la base
- Ils sont dépendants de la version de l'instance
- nous aurons besoin de l'enregistrer auprès de l'instance mère, notamment pour des raisons de sécurité, pour éviter que d'autres fassent passer des éléments pour des runners
- Le SCOPE d'un runner est aussi très important (soit à un projet spécifique, soit à un groupe, soit à une instance complète gitlab)

gitlab-ci : Runners

- Ce partage dépend souvent de la taille de la société et de la quantité de tests
- gitlab fournit également des runners publics mais leur utilisation est limitée dans le temps
- doc runners : <https://docs.gitlab.com/runner/>
- EXECUTOR ?
- C'est le type de runner qui est appliqué, par ex. SSH, shell, powershell, docker, kubernetes, virtualbox
- On le sélectionne pour l'adapter au type de job
- <https://docs.gitlab.com/runner/executors/>

gitlab-ci : Runners

- Maintenant que le runner est sélectionné, nous allons pouvoir mettre en place un pipeline :
- C'est un outil qui va permettre d'assembler les tâches que l'on veut exécutées sur le code
- Il sera déclenché par un trigger ou une tâche cron (comme jenkins)
- Il aura une action finale, par ex. déploiement en prod ou builder une image docker
- Il est découpé en stages (des grandes étapes logiques)
- Il est complètement décrit dans un fichier yaml à la racine du projet
.gitlab-ci.yaml

gitlab-ci : Runners

- Les STAGES sont les étapes logiques d'un pipeline, constituées de 1 ou plusieurs jobs
- Un JOB est une tâche précise, d'où le rôle du runner; on y appose des contraintes comme des variables d'env ou des conditions
- Ce JOB est constitué d'un volet script contenant les commandes réalisées au sein de ce job
- Étapes before/after possibles
- Donc un **pipeline** contient des **stages** qui contiennent des **jobs** qui contiennent des **scripts**
- <https://about.gitlab.com/blog/2018/01/22/a-beginners-guide-to-continuous-integration/>

Activation et 1ier Runner shell sur VM

Ce 1ier runner sera une VM et aura un type executor SHELL

Les runners ont des scopes

Le plus simple est de créer un runner dans un projet

Blank project ⇒ premier pipeline ⇒ peu importe la visibilité

Gitlab > settings > CI/CD > Runners

Runners

Expand

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

Activation et 1ier Runner shell sur VM

2 types de runners qui pourront tourner :

Specific runners

Shared runners

L'inconvénient d'une ressource partagée c'est que si elle n'est pas disponible il va falloir attendre, etc...

Donc possibilité de désactiver les runners pour qu'ils soient spécifiques au projet

Procédure d'installation fournie :

Set up a specific runner for a project

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

`http://localhost/` 

Activation et 1ier Runner shell sur VM

1 - Installation de gitlab-runner

```
(base) $gitlab-runner status  
Runtime platform  
gitlab-runner: Service is running  
[...]
```

2 - Le mettre en contact avec notre instance gitlab

Set up a specific runner for a project

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

And this registration token:

`GR1348941FgHZtNyQHcEcL4Fbas8F` 

Activation et 1ier Runner shell sur VM

Enregistrement du runner :

```
export REGISTRATION_TOKEN="GR1348941FgHZtNyQHcEcL4Fbas8F"  
gitlab-runner register --url http://localhost/ --registration-token $REGISTRATION_TOKEN
```

par défaut nous sommes en mode interactif

entrer des tags : shell, (linux, windows, macos...) (le tagger par exemple selon le langage)

type d'exécuteur : shell

⇒ Rafraîchir la page des runners

Available specific runners

#2 (J28ZSW4x) 🔒

runner1

macos shell

  Remove runner

Activation et 1ier Runner shell sur VM

Pour avoir les détails du runner

Available specific runners

#2 (J28ZSW4x)

runner1

macos shell



```
--non-interactive
--url "http://localhost/"
--registration-token
"GR1348941FgHZtNyQHcEcL4Fbas8F"
--executor "shell"
--description "runner1"
--tag-list "shell,macos"
--run-untagged
--locked="false"
```

Pour vérifier la conf du runner (voir selon os) :

```
(config.toml) is in /Users/<username>/ .gitlab-runner/
```

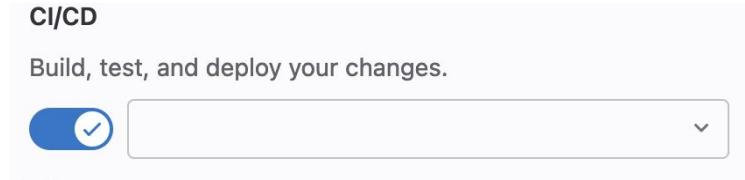
Activation et 1ier Runner shell sur VM

Il faut vérifier ensuite dans la conf gitlab.rb si l'utilisation du pipeline est activée

⇒ rechercher builds

```
141      # gitlab_rails['gitlab_default_projects_features_builds'] = true
```

en graphique ⇒ projet ⇒ settings ⇒ visibility ⇒



Activation et 1ier Runner shell sur VM

Maintenant dans le projet ⇒ Build ⇒ pipelines

Build ⇒ Editor

Optimize your workflow with CI/CD Pipelines
Create a new `.gitlab-ci.yml` file at the root of the repository to get started.

Configure pipeline

There are currently no pipelines.

Edit Visualize Vali

Browse templates

Nous allons utiliser pour le moment le template par défaut, qui a déjà structuré le pipeline par défaut

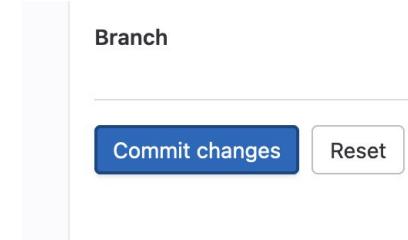
Activation et 1ier Runner shell sur VM

```
unit-test-job:  
  stage: test  
  script:  
    - echo "Runn  
    - sleep 60  
    - echo "Code"
```

Deux jobs dans le même stage

```
lint-test-job:  
  stage: test  
  script:  
    - echo "Lint  
    - sleep 10
```

Le job est en statut pending
car pas de runners assignés



Activation et 1ier Runner shell sur VM

Pourquoi ? Nous avions pourtant bien déclaré le runner shell dans le projet
Car, par défaut, les associations au runner se font via les tags
et ces tags n'ont pas été déclarés dans le fichier de pipeline

Retourner dans **Settings ⇒ CI/CD ⇒ Runners ⇒ notre runner ⇒ edit ⇒ Run untagged jobs**

Dans ce cas, tous les jobs non taggés seront exécutés via ce runner

Retourner dans CI/CD ⇒ Pipelines

Status	Job	Pipel
passed	#5 ⌚ main ~ 5115caa4	#1 creat
passed	#4 ⌚ main ~ 5115caa4	#1 creat

Activation et 1ier Runner shell sur VM

Pour voir les jobs exécutés, on a deux entrées possibles :

CI/CD ⇒ Pipelines et CI/CD ⇒ Jobs

Status	Pipeline	Triggerer	Stages
<div>passed</div> <p>⌚ 00:01:32 📅 5 minutes ago</p>	<p>Update .gitlab-ci.yml file #1 ⌚ main → 5115caa4 🎉 latest</p>		 <div>deploy: passed</div>

Retournons sur notre runner ⇒ nous voulons maintenant utiliser des tags

Run untagged jobs



Indicates whether this runner can pick jobs without tags

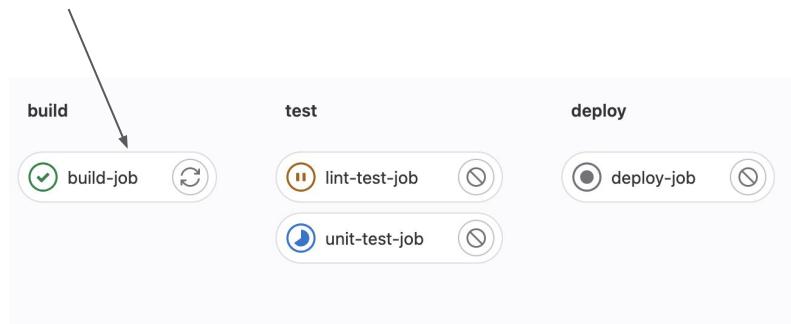
Activation et 1ier Runner shell sur VM

Retourner dans **Pipelines** ⇒ **Editor**

Ajout du tag shell au build-job

```
24   build-job:  
25     stage: build  
26     tags:  
27       - shell  
28     script:
```

Seul le 1ier job est lancé



Executor Docker

En repartant du pipeline précédent, **Settings ⇒ CI/CD**

Se connecter sur runner2 et vérifier que docker est présent :

```
*** System restart required ***
vagrant@runner2:~$ docker ps
CONTAINER ID        IMAGE          COMMAND       CREATED      STATUS      PORTS      NAMES
```

Nous allons créer un répertoire pour le runner pour qu'il puisse stocker sa conf

```
mkdir -p /data/
```

Executor Docker

puis lancement du container :

```
docker run -d \
  --name gitlab-runner \
  --restart always \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /data/gitlab-runner:/etc/gitlab-runner \
  gitlab/gitlab-runner:latest
```

Il faut également configurer le dns de notre instance gitlab dans notre runner2

```
sudo vim /etc/hosts
```

```
127.0.2.1 runner2 runner2
192.168.12.40 gitlab gitlab-home.com
192.168.12.40 gitlab-home.com registry.gitlab-h
192.168.12.41 gitlab-runner1
```

```
vagrant@runner2:~$ ping gitlab-home.com
PING gitlab (192.168.12.40) 56(84) bytes of data.
64 bytes from gitlab (192.168.12.40): icmp_seq=1 ttl=64 time=1.20 ms
```

Executor Docker

```
vagrant@runner2:~$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS
PORTS NAMES
15c41e755d0f gitlab/gitlab-runner:latest "/usr/bin/dumb-init ..." 6 minutes ago Up 6 minutes
gitlab-runner
```

on pourra par la suite avec un docker exec enregistrer ce runner docker

Dans l'instance gitlab **Settings ⇒ CI/CD**

```
docker exec -it gitlab-runner gitlab-runner register
```

```
Enter the GitLab instance URL (for example, https://gitlab.com/):
http://192.168.12.40
```

```
Enter the registration token:
GR13489416jiKoA3mQ1ZKrirwDiJ9
Enter a description for the runner:
[15c41e755d0f]: runner2
Enter tags for the runner (comma-separated):
docker,linux
```

Executor Docker

```
Enter an executor: docker,  
r-ssh+machine, kubernetes:  
docker
```

si on ne spécifie pas ensuite l'image par défaut il prendra celle-ci

```
docker  
Enter the default Docker image (for example, ruby:2.7):  
debian:latest
```

Succès ⇒ rafraîchir page sur instance gitlab

Available specific runners

 #1 (ctF7Nacm) ⋮	  Remove runner
runner2	
 docker	 linux

Executor Docker

Puis nous allons tester notre runner : Build ⇒ Pipeline Editor

Ne garder qu'un seul stage, on peut préciser une image mais aucune obligation

Cela signifie que le container runner en lancera un debian pour exécuter ce code

Puis commit changes

```
 8 Running on runner-s1flowrc-project-2-concurrent
 9 Getting source from Git repository
10 Fetching changes with git depth set to 20...
11 Initialized empty Git repository in /builds/gi
12 Created fresh repository.
13 Checking out 5f49a148 as main...
14 Skipping Git submodules setup
 15 Executing "step_script" stage of the job script
16 Using docker image sha256:54e726b437fbdb2dd7b43
17 $ echo "Début du pipeline..."
18 Début du pipeline...
19 $ sleep 30
• • •
```

```
19 stages:          # List of stages for
20   - build
21
22 build-job:      # This job runs in
23   stage: build
24   image: debian:latest
25   tags:
26     - docker
27   script:
28     - echo "Début du pipeline..."
29     - sleep 60
30     - echo "Fin du pipeline OK"
```

Executor Docker

Vérifier sur runner2 la présence du second container durant le run

```
vagrant@runner2:~$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
db41f4a3a03b        54e726b437fb      "sh -c 'if [ -x /usr..."   8 seconds ago       Up 7 seconds
15c41e755d0f        gitlab/gitlab-runner:latest  "/usr/bin/dumb-init ..."   About an hour ago  Up About an hour
vagrant@runner2:~$
```

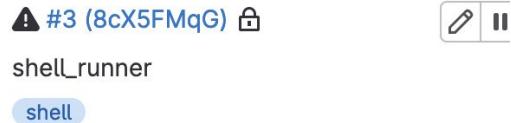
Et après le run, le container utilisé pour le pipeline est kill mais image debian toujours présente

```
vagrant@runner2:~$ docker images
REPOSITORY          TAG      IMAGE ID
registry.gitlab.com/gitlab-org/gitlab-runner/gitlab-runner-helper   x86_64-4d1ca121
gitlab/gitlab-runner          latest
debian                latest
```

Shared Runner

Le premier runner shell que nous avions lancé n'était pas exploitable par d'autres projets

Available specific runners



Créer un autre projet depuis mongroupe

puis CI/CD ⇒ Editor dans ce nouveau projet

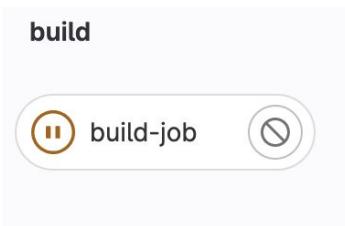
New project > Create blank project

Project name

Project URL / Project slug

Want to organize several dependent projects under the same namespace? [Create a group](#).

Shared Runner



Le pipeline reste en pending pourtant en retournant dans les settings ⇒ runners ⇒

l'option est bien activée

Pour Activer cela :

en Admin **seulement** ⇒

Status	Runner	Owner	Actions
Never contacted	#3 (8cX5FMqG) · Project Version 15.8.2 · shell_runner Idle · Last contact: Never · 192.168.12.40 · 0 · Created 7 minutes ago shell	premierpipeline	Edit

Shared runners

These runners are available to all groups a

Enable shared runners for this project



This GitLab instance does not provide any sh yet. Instance administrators can register shar

- Protected
Use the runner
- Run untagged
Use the runner
- Lock to current branch
Use the runner

Shared Runner

nouveau pipeline ⇒ CI/CD ⇒ Pipelines toujours en statut bloqué



Retourner dans CI/CD ⇒ le runner est maintenant présent dans ce projet mais il faut enable le runner shell pour ce projet

Nous annulons l'ancien job et relançons un autre

Shared Runner

Donc il faut prendre en compte ces différentes notions :

- runners bloqués
- runners en pause
- runners utilisés
- runners partagés

Variables

Les variables ont de multiples niveaux

Elles sont très importantes et utiles, elles permettent notamment :

- de gérer les secrets
- de réutiliser les sha1 des commits
- de taguer des images pour pouvoir les pousser par ex
- de transmettre des infos entre les jobs...

Variables

il existe différents types de variables :

- variables prédéfinies
- définies dans le gitlab-ci : locales (internes au job) ou globales
- soit au niveau du projet (settings ⇒ CI/CD) soit au niveau du groupe ou de l'instance

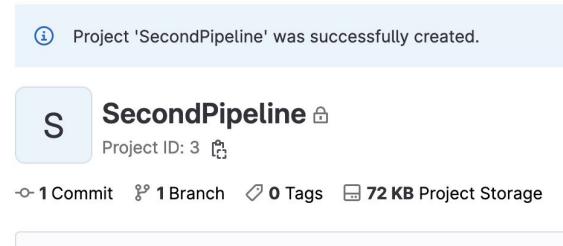
Variables prédéfinies

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

De base bcp d'infos sur les commits

Ces variables ont différents scopes pour un build, ou pour un push ou pour un résultat synthétique comme le résultat global d'un pipeline pour des récap

⇒ Crédit d'un nouveau projet ⇒



⇒ Settings ⇒ CI/CD ⇒ Runners

Variables prédéfinies

On y retrouve notre shared runner que nous allons activer

CI/CD ⇒ Editor ⇒ Configure Pipeline

Rq : le niveau stages dans un pipeline
n'est pas nécessaire

```
17 $ echo "Start..."  
18 Start...  
19 $ echo "$CI_JOB_ID"  
20 25  
22 Job succeeded
```

```
start-job:  
  tags:  
    - shell  
  script:  
    - echo "Start..."  
    - echo "$CI_JOB_ID"  
end-job:  
  tags:  
    - shell  
  script:  
    - echo "ended !!"
```

Other available runners

#4 (6MeiyjYF)

runner_shell

shell

Enable for this project

Variables GITLAB-CI

Ce sont des variables que l'on peut définir dans le ci

```
1  variables:
2    MyVAR: 'Hello'
3  start-job:
4    tags:
5      - shell
6    script:
7      - echo "Start..."
8      - echo "$MyVAR"
9  end-job:
10   tags:
11     - shell
12   script:
13     - echo "ended !!" |
```

```
17 Start...
18 $ echo "$MyVAR"
19 Hello
21 Job succeeded
```

C'est ici une variable globale car elle est valable pour tous les jobs

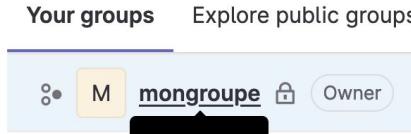
Variables GITLAB-CI

Si nous désirons des variables locales, nous les définissons au sein d'un job :

```
1  variables:
2  | | MyVAR: 'Hello'
3  start-job:
4  | variables:
5  | | MyVAR: "Ceci est une variable locale qui surcharge la globale"
6  tags:
7  | - shell
8  script:
9  | - echo "Start..."
10 | - echo "$MyVAR"
11 .
12 .
13 .
14 .
15 .
16 $ echo "Start..."
17 Start...
18 $ echo "$MyVAR"
19 Ceci est une variable locale qui surcharge la globale
20 .
21 Job succeeded
```

Variables définies dans Projet, Groupe et Formulaire

Groups ⇒



settings ⇒ CI/CD ⇒ Variables ⇒

Add variable

Key
PIPELINE_VAR

Value
my group

Type Variable Environment scope All (default)

Les variables masquées le sont complètement au niveau des logs

```
1 start-job:  
2   tags:  
3     - shell  
4   script:  
5     - echo "Start..."  
6     - echo "$PIPELINE_VAR"  
7 end-job:  
8   -----
```

Variables définies dans Projet, Groupe et Formulaire

Une fois que la variable a été définie dans le groupe,
il est également possible d'aller dans

notre **projet** ⇒ CI/CD ⇒

On peut y définir une variable qui portera le même nom pour surcharger la précédente au niveau du groupe

Affichage de sa présence héritée du groupe

Variables

Variables store information, like passwords and secret keys, tha
of 200 variables. [Learn more.](#)

Add variable

Key

Value

Type

```
16 $ echo "Start..."  
17 Start...  
18 $ echo "$PIPELINE_VAR"  
19 my project  
21 Job succeeded
```

Variables définies dans Projet, Groupe et Formulaire

Quand on lance le pipeline dans le projet, il nous propose également des entrées pour définir des variables ou des files



Et le dernier niveau est directement dans le script CI
MAIS il ne permet pas de surcharger
la variable de projet

```
1 start-job:  
2   variables:  
3     PIPELINE_VAR: "my job"  
4   tags:  
5     - shell  
6   script:  
7     - echo "Start..."  
8     - echo "$PIPELINE_VAR"  
9 end-job:
```

L'architecture des pipelines

Les pipelines sont en réalité assez plats mais ils ne servent pas qu'à faire des builds ...

On peut scheduler et donc revenir à des fonctionnements identiques à ceux de jenkins cad travailler de la data, créer des dépendances ...

https://docs.gitlab.com/ee/ci/pipelines/pipeline_architectures.html

Nous allons tester ces 3 pipelines et comprendre ainsi les possibilités qu'ils amènent

- * Basic pipelines : basic, très souvent présent ...
- * Direct Acyclic Graph (DAG) pipelines : modélisation, dépendances entre éléments
- * Parent/enfants pipelines : plusieurs CI avec des règles, cas de figure plus complexes

L'architecture des pipelines

Basic

Lancer un nouveau projet

New project > Create blank project

Project name

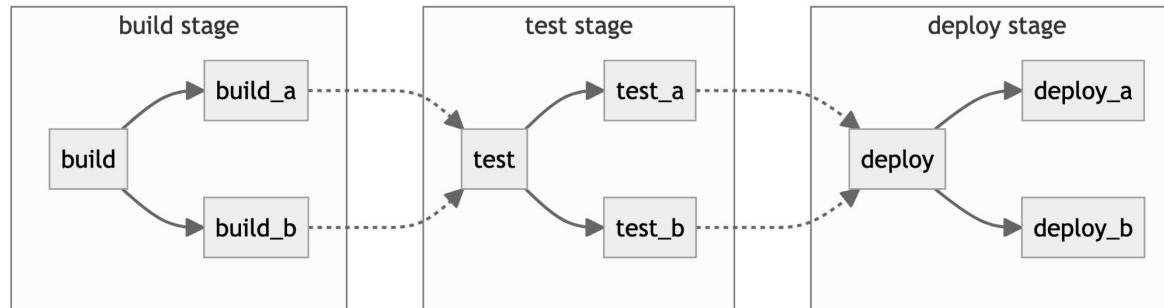
Project URL

 /

Project slug

Want to organize several dependent projects under the same namespace? [Create a group.](#)

Nous n'avons pas besoin de code pour se faire la main sur l'architecture des pipelines **CI/CD ⇒ Editor ⇒ Configure Pipeline**



L'architecture des pipelines

Mais nous n'allons pas appeler nos stages ainsi car un pipeline ne sert pas forcement a build & deploy ...

On peut processer de la data avec un runner et des commandes shell. On fait faire au runner ce que l'on veut : ETL, envoi dans DB ...

Etapes Jobs	Stage1	Stage2	Stage3
	Job1	Job3	Job5
	Job2	Job4	Job6

Ici c'est une demarche tres lineaire : chaque étape est bloquante

L'architecture des pipelines

Rq : ne pas préciser dans les messages de commit : "modif de gitlab-ci" puisque c'est ce que l'on va retrouver dans les logs.

Mais plutôt préciser la problématique résolue

```
stages:  
  - stage1  
  - stage2  
job1:  
  stage: stage1  
  script:  
    - echo "stage1 - job1"  
job2:  
  stage: stage1  
  script:  
    - echo "stage1 - job2"  
job3:  
  stage: stage2  
  script:  
    - echo "stage2 - job3"  
job4:  
  stage: stage2  
  script:  
    - echo "stage2 - job4"  
...
```

```
1  stages:  
2  - stage1  
3  - stage2  
4  - stage3  
5  job1:  
6  stage: stage1  
7  script:  
8  - echo "stage1 - job1"  
9  job2:  
10 stage: stage1  
11 script:  
12 - echo "stage1 - job2"  
13 job3:  
14 stage: stage2  
15 script:  
16 - echo "stage2 - job3"  
17 job4:  
18 stage: stage2  
19 script:  
20 - echo "stage2 - job4"  
21 job5:  
22 stage: stage3  
23 script:  
24 - echo "stage3 - job5"  
25 job6:  
26 stage: stage3  
27 script:  
28 - echo "stage3 - job6"
```

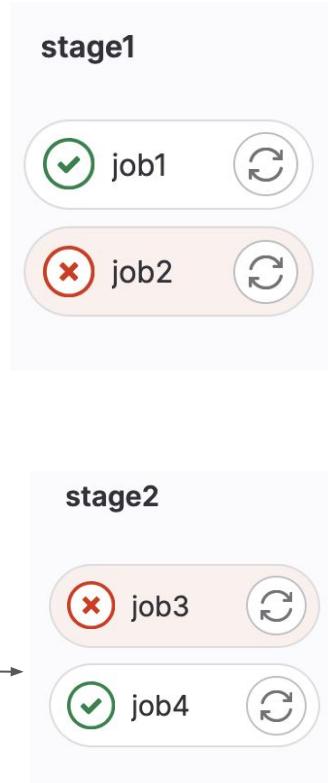
L'architecture des pipelines

Si nous générions une erreur :

Les exits sont intéressants pour tester les comportements des pipelines

et si failed en job3

il déroule quand même le job4



```
stages:
  - stage1
  - stage2
job1:
  stage: stage1
  script:
    - echo "stage1 - job1"
job2:
  stage: stage1
  script:
    - echo "stage1 - job2"
    - exit 1
job3:
  stage: stage2
  script:
    - echo "stage2 - job3"
job4:
  stage: stage2
  script:
    - echo "stage2 - job4"
  ...
```

L'architecture des pipelines

Pour répondre à ce cheminement bloquant du pipeline, nous pouvons utiliser le type DAG de pipeline dans lequel nous allons pouvoir créer des dépendances entre les jobs

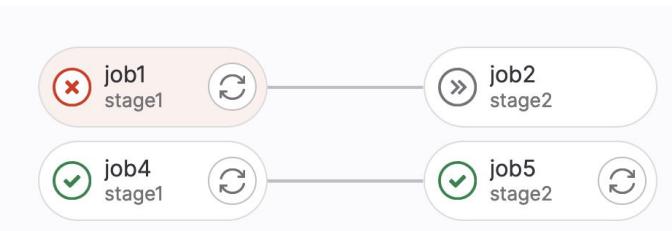


L'indépendance n'est pas totale entre ces 2 circuits n'est pas totale mais par ex si job2 échoue alors job3 ne sera pas déroulé

L'architecture des pipelines



Possibilité d'afficher les dépendances



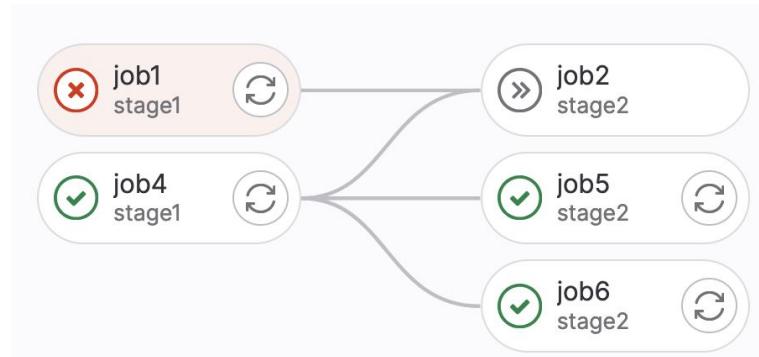
```
stages:
  - stage1
  - stage2
  - stage3
job1:
  stage: stage1
  script:
    - echo "stage1 - job1"
    - exit 1
job4:
  stage: stage1
  script:
    - echo "stage1 - job4"
job2:
  stage: stage2
  needs: [job1]
  script:
    - echo "stage2 - job2"
job5:
  stage: stage2
  needs: [job4]
  script:
    - echo "stage2 - job5"
```

L'architecture des pipelines

La séparation de ces deux “pipelines” n'est pas totale car le timing des jobs est respecté

Plusieurs jobs peuvent dépendre d'un autre job également

Ou un job qui dépende de deux ou trois autres précédents jobs



L'architecture des pipelines

Cas du pipeline PARENT/ENFANTS :

- découpe en différent gitlab-ci
- gestion de déclenchement suivant des répertoires spécifiques (ex de la doc gitlab avec des règles)
- gestion de blocs
- 2 cas : A et B correspondants chacun à un dossier dans le repo

```
caseA : stage1  stage2  stage3  
caseB : stage21  stage22  stage23
```

L'architecture des pipelines

Créer 2 répertoires dans files

Cela pourrait être deux langages par ex

Créer un nouveau fichier dans caseA et caseB:

New file



Name
caseA
caseB
.gitlab-ci.yml

```
stages:
  - stage1
  - stage2
  - stage3
job1:
  stage: stage1
  script:
    - echo "caseA - job1"
job2:
  stage: stage2
  needs: [job1]
  script:
    - echo "caseA - job2"
job3:
  stage: stage3
  needs: [job2]
  script:
    - echo "caseA - job3"
```

L'architecture des pipelines

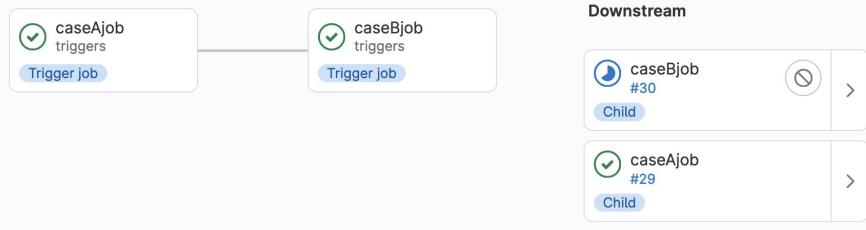
Si nous désirons modifier le nom du fichier de ci racine pour le repo ou son chemin :

Settings ⇒ CI/CD ⇒ General Pipelines

Puis dans l'éditeur du projet pour le ci racine :

Possibilité d'avoir plusieurs stages, aucun pb

Plus ajout d'une dépendance



CI/CD configuration file

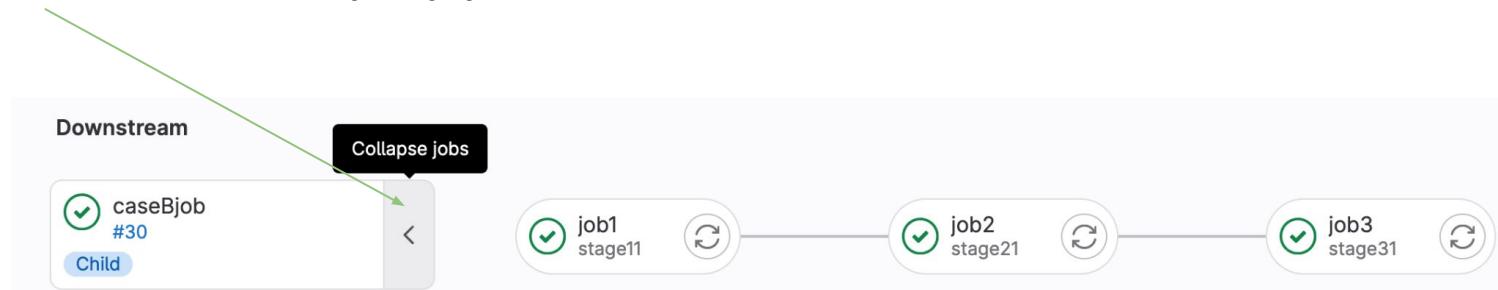
.gitlab-ci.yml

The name of the CI/CD configura
?

```
stages:  
  - triggers  
  
caseAjob:  
  stage: triggers  
  trigger:  
    include: caseA/.gitlab-ci.yml  
  
caseBjob:  
  stage: triggers  
  needs: [caseAjob]  
  trigger:  
    include: caseB/.gitlab-ci.yml
```

L'architecture des pipelines

Déroulement de chaque pipeline :



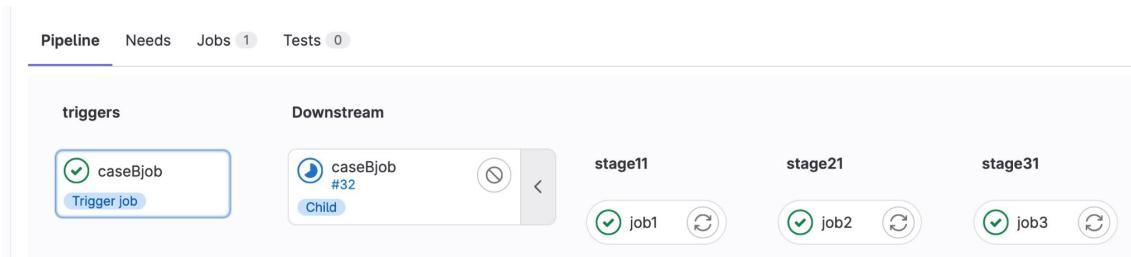
Comment ne jouer que le ci du caseA si uniquement modif dans caseA et pour caseB également ?

- enlever les dépendances
- ajout de rules :

```
rules:  
  - changes:  
    - caseA/*
```

L'architecture des pipelines

Dans notre cas la racine du repo n'est pas comprise dans les changes donc pour lancer ce pipeline nous devons faire une modif dans un des cases



```
1 stages:
2   - triggers
3
4 caseAjob:
5   stage: triggers
6   trigger:
7     include: caseA/.gitlab-ci.yaml
8   rules:
9     - changes:
10       - caseA/*
11
12
13 caseBjob:
14   stage: triggers
15   trigger:
16     include: caseB/.gitlab-ci.yaml
17   rules:
18     - changes:
19       - caseB/*
```

Keywords dans les Pipelines

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

- only : quand un job DOIT être lancé dans ce cas de figure
- except : quand un job NE DOIT PAS être lancé
- ATTENTION : ils ne s'appliquent que dans un bloc job
- cela pourra etre des elements simples ou composés
- Rq : dans la doc de GitLab ⇒ préférer les **rules**, car ces 2 précédents mots-clés ne sont pas activement développés

Keywords dans les Pipelines

Ils sont applicables à 4 sous-keywords :

- refs : bcp de choses différentes
- variables
- changes
- kubernetes : assez spécifique

REFS :

- nom de branches (ou regex) ou type de pipeline
 - lors de merge_requests : lint, tests...
 - api : si déclenchement du pipeline via l'api
 - schedules : comme des cron job linux
- 2 manières de les rédiger : capable de les retrouver

```
only:  
refs:  
  - branches
```

```
only:  
  - branches
```

Keywords dans les Pipelines

exemple DANS UN NOUVEAU PROJET sur les tags et les branches :

job build ne se déclenche que si un commit est tagué

job test se déclenche si le commit n'est pas tagué

deploy ne se déclenche que sur des branches préfixées par ab

Rq : pas nécessaire de citer ces 3 stages car ces noms pré-existent déjà dans gitlab

```
stages:
  - build
  - test
  - deploy
build:
  stage: build
  script:
    - echo "build"
  only:
    refs:
      - tags
test:
  stage: test
  script:
    - echo "test"
  except:
    refs:
      - tags
deploy:
  stage: deploy
  script:
    - echo "deploy"
  only:
    refs:
      - /^ab-.*/$/
# rules:
- # if: '$CI_COMMIT_BRANCH'
# =~ /ab-.*/$/'
```

Keywords dans les Pipelines

A juste lancé la partie test pour le moment

Nous créons une nouvelle branche

New Branch

Branch name

Create from

Existing branch name,

depuis main ⇒ hérite du .gitlab-ci.yaml

The screenshot shows a pipeline interface with a single stage named 'test'. The stage status is marked with a green checkmark icon and the word 'test'. To the right of the stage are two buttons: a green circle with a white checkmark and a circular arrow icon. Below the pipeline, a sidebar titled 'Repository' is visible, featuring options for 'Files', 'Commits', and 'Branches'. The 'Branches' option is highlighted with a gray background, indicating it is the active tab.

Keywords dans les Pipelines

Toujours sur cette branche maintenant, nous allons la taguer

New Tag

Do you want to create a release wi

Tag name

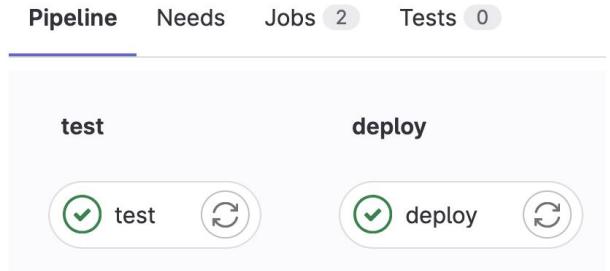
v1.0.0

Create from

ab-feature-10

Existing branch name, tag, or comi

Message



Pipeline Needs Jobs 2 Tests 0

test deploy

(checkmark) test (refresh) (checkmark) deploy (refresh)

Repository 0

Issues 0

Merge requests 0

CI/CD

Tags

Gestion des variables dans un pipeline

utilisation de variables définies et leurs expressions dans la CI/CD

```
$VARIABLE == "some value"      # à une valeur
$VARIABLE_1 == $VARIABLE_2    # à une autre variable
$VARIABLE == null            # si la variable est définie ou non
$VARIABLE                  # si la variable existe ou non
$VARIABLE =~ /content.*/     # un pattern
$VARIABLE1 =~ /content.*/ || $VARIABLE2 =~ /thing$/ && $VARIABLE3 # combinaison
$CI_COMMIT_BRANCH == "my-branch" || (($VARIABLE1 == "thing" || $VARIABLE2 == "thing") &&
$VARIABLE3)
```

Gestion des variables dans un pipeline

Ce job ne s'exécutera que si les deux variables ont ces valeurs

Nous allons lancer ensuite manuellement le pipeline

ATTENTION

ET

OU

```
build:  
  stage: build  
  script:  
    - echo "build"  
only:  
  variables:  
    - $VARIABLE1 == "jacques" && $VARIABLE2 == "jean"
```

```
build:  
  stage: build  
  script:  
    - echo "build"  
only:  
  variables:  
    - $VARIABLE1 == "jacques"  
    - $VARIABLE2 == "jean"
```

Gestion des variables dans un pipeline

Ce job ne
s'exécutera que si
les deux variables
ont ces valeurs

Nous allons lancer
ensuite
manuellement le
pipeline

Run for branch name or tag

main ▾

Variables

Variable	VARIABLE1	jacques.....
Variable	VARIABLE2	jean



Gestion des variables dans un pipeline

Pour combiner
variables et refs

Tester les deux
conditions

```
build:  
    stage: build  
    script:  
        - echo "build"  
only:  
    variables:  
        - $VARIABLE1 == "jean"  
refs:  
    - /^ab-.*$/
```

Keyword changes

Lance OU ne lance pas le pipeline si tel répertoire a été édité

Utilise des valeurs :

- chemin de fichier
- avec wildcard caseA/*
- ou avec du global caseA/**/*.{yaml,yml}

Inconvénient des manip depuis le départ : dès qu'il y a une modif sur gitlab-ci il se lance automatiquement ⇒ SOLUTION :

```
except:  
  changes:  
    - ".gitlab-ci.*"
```

Keyword changes

New file

Si nous ajoutons ensuite un fichier dans main :

Le pipeline se déclenche bien car la modif ne concerne pas le .gitlab-ci

```
g' main | / test.txt  
1 test|
```

Les Services

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

Ce sont des éléments annexes et nécessaires dans votre CI :

- quand on est en train de builder un applicatif, on peut lui intégrer une DB pour pouvoir l'utiliser et le tester
- on va pouvoir simuler une api
- par défaut, gitlab appelle les images de containers : mysql, postgresql, redis et gitlab...
- Au final, on va pouvoir utiliser énormément de services différents car nous faisons appel à des containers dockers

Service postgresql

Comme ces services sont des containers, nous avons besoin d'un runner docker

Admin ⇒ CI/CD ⇒ Runners

Vérifier qu'il soit bien partagé (untagged)

```
variables:  
  POSTGRES_DB: postgres  
  POSTGRES_USER: runner  
  POSTGRES_PASSWORD: postgres  
  POSTGRES_HOST_AUTH_METHOD: trust  
stages:  
  - test  
Testing:  
  stage: test  
  image: debian:latest  
  services:  
    - postgres  
  script:  
    - apt update && apt install -y postgresql-client  
    - PGPASSWORD=$POSTGRES_PASSWORD psql -U $POSTGRES_USER -h postgres -d $POSTGRES_DB -c  
      "\l"  
  tags:  
    - docker
```



Service postgresql

Se connecter à la machine contenant ce runner :

```
ssh vagrant@192.168.12.42
```

Vérifier avec un docker ps que le container gitlab-runner tourne bien

Details Jobs 2

Description	runner_docker
Last contact	right now
Version	15.8.2
IP Address	192.168.12.42
Executor	docker

```
vagrant@runner2:~$ docker ps
CONTAINER ID   IMAGE          COMMAND
STATUS        PORTS      NAMES
15c41e755d0f  gitlab/gitlab-runner:latest   "/usr/bin/dumb-init ..."
Up 43 minutes           gitlab-runner
```

Service postgresql

Création d'un projet :

New project > Create blank project

Project name

Project URL

 /

Project slug

Want to organize several dependent projects under the same namespace? [Create a group.](#)

Activer le runner docker dans les **Settings** ⇒ **CI/CD**

Available specific runners



CI/CD ⇒ **Editor** y coller le gitlab-ci precedent

Rq : le mot de passe est en clair mais peu gênant car c'est une action qui ne persiste pas

Service postgresql

Le service postgres fourni de base va ici utiliser les variables génériques définies en haut du .gitlab-ci

Les scripts vont être exécutés dans l'image debian : simulation d'une connection avec un pgsql

- install d'un client postgres
- etablissement d'une variable password
- -h postgres ⇒ résolution dns automatique sur le nom du service postgres
- -l ⇒ lister les databases
- tags : docker ⇒ permet de matcher le runner docker

Service postgresql

- Rq : le pipeline aura besoin de télécharger l'image donc faire attention au choix des images (légère, avec juste les binaires nécessaires)

```
121 Setting up postgresql-client (13+225) ...
122 Processing triggers for libc-bin (2.31-13+deb11u5) ...
123 $ PGPASSWORD=$POSTGRES_PASSWORD psql -U $POSTGRES_USER -h postgres -d $POSTGRES_DB -c "\l"
124
125           List of databases
126
127   Name    | Owner     | Encoding | Collate      | Ctype      | Access privileges
128   -----+-----+-----+-----+-----+
129   postgres | runner   | UTF8     | en_US.utf8  | en_US.utf8 |
130   template0 | runner   | UTF8     | en_US.utf8  | en_US.utf8 | =c/runner      +
131               |          |          |             |             | runner=CTc/runner
132   template1 | runner   | UTF8     | en_US.utf8  | en_US.utf8 | =c/runner      +
133               |          |          |             |             | runner=CTc/runner
134
135 (3 rows)
136 Job succeeded
```

Service postgresql

Il faudra souvent choisir la version correspondant au projet

```
stage: test
image: debian:latest
services:
  - name: postgres:alpine
    alias: mydb
script:
  - apt update && apt install -y postgresql-client
  - PGPASSWORD=$POSTGRES_PASSWORD psql -U $POSTGRES_USER -h mydb -d $POSTGRES_DB -c "\l"
tags:
  - docker
```

Service postgresql

Nous pourrions aussi avoir besoin de plusieurs versions de postgres

Ce n'est pas limité à postgres bien sûr, on pourrait ajouter autant de containers que nécessaire

Ici tout est dans le même job mais l'idéal est plutôt d'avoir un job par test de version

```
Testing:
  stage: test
  image: alpine:latest
  services:
    - name: postgres:15.2-alpine3.17
      alias: mydb1
    - name: postgres:11.16-stretch
      alias: mydb2
    - name: postgres:13.10-alpine
      alias: mydb3
  script:
    - apk add postgresql
    - PGPASSWORD=$POSTGRES_PASSWORD psql -U $POSTGRES_USER -h mydb1 -d postgres -c
      "SELECT version();"
    - PGPASSWORD=$POSTGRES_PASSWORD psql -U $POSTGRES_USER -h mydb2 -d postgres -c
      "SELECT version();"
    - PGPASSWORD=$POSTGRES_PASSWORD psql -U $POSTGRES_USER -h mydb3 -d postgres -c
      "SELECT version();"
  tags:
    - docker
```

Service mysql

```
variables:
  MYSQL_DATABASE: "db_name"
  MYSQL_ROOT_PASSWORD: "dbpass"
  MYSQL_USER: "username"
  MYSQL_PASSWORD: "dbpass"
  MYSQL_HOST: mysql
stages:
  - test
Testing:
  image: alpine:latest
  services:
    - mysql
  stage: test
  script:
    - apk add mariadb-client
    - echo "SHOW tables;" | mysql -u root -p"${MYSQL_ROOT_PASSWORD}" -h mysql "${MYSQL_DATABASE}"
    - echo "CREATE TABLE test_table (field1 int);;" | mysql -u root -p"${MYSQL_ROOT_PASSWORD}" -h mysql "${MYSQL_DATABASE}"
    - echo "SHOW tables;" | mysql -u root -p"${MYSQL_ROOT_PASSWORD}" -h mysql "${MYSQL_DATABASE}"
tags:
  - docker
```

Service avec nginx

On pourrait y simuler des appels d'api

```
Testing:
  stage: test
  image: debian:latest
  services:
    - name: nginx:latest
      alias: mynginx
  script:
    - apt update -qq 2>&1 >/dev/null && apt -qq install -y curl 2>&1 >/dev/null
    - curl mynginx
  tags:
    - docker
```

Remarques sur services

Dans la doc :

- possibilité d'utiliser des entrypoints pour pouvoir réécrire des commandes exemple sur nginx
- possible d'utiliser le service mysql avec du shell executor également : en effet une fois que mysql est installé sur la machine runner, nous pouvons simuler des commandes shell pour mysql
- donc on peut retrouver un runner qui reproduira un environnement mais ceci est bcp moins souple ...
- intéressant de cleaner les images docker avec une tache cron (voir docker images depuis le début)

CACHE

Élément important dans gitlab-ci pour gagner en fluidité et gagner du temps

New Project ⇒ cache ⇒ utilisation d'un runner type docker

il existe deux modes de stockage dans gitlab-ci

- cache
- artifacts

CACHE

Le cache permet de faire beaucoup de choses :

- stocker et partager des libs et des packages (pour pip, maven...)
- chaque runner a son propre cache (utiliser le même pour les jobs, un de ses répertoires)
 - réutilisable (intérêt du tag)
- organiser éventuellement en fonction de vos workflows
- cache distribué via du stockage objet S3 (multi-runners), mais artifacts stockés sur instance gitlab
- utilisation de clef pour définir diff caches : specific a des branches, des jobs, des stages
- CACHE_FALLBACK_KEY (variable) pour définir une clef par défaut
- désactivable sur demande (cache:[])
- utilisation des ancrées possibles : blocs de cache appelés régulièrement
- définition de policy possibles

CACHE

Créer un pipeline ⇒

key : permet d'avoir différents caches, de mieux organiser les éléments, de déclencher certains caches et pas d'autres. Possibilité de pointer vers un fichier spécifique

paths: on prend ce répertoire comme cache, il existe des policies qui permettent de ne pas alimenter le cache

Ici le cache permet de partager un fichier

Ce cache est maintenant utilisable par ce runner pour d'autres jobs

```
cache:
  - key:
      # files
      # - test.txt
    paths:
      - .lib
stages:
  - step1
  - step2
job1:
  stage: step1
  script:
    - mkdir -p .lib
    - echo test_job1 > .lib/test.txt
  tags:
    - docker
job2:
  stage: step2
  script: cat .lib/test.txt
  tags:
    - docker
```

Cache

Désormais dans le menu pipelines ⇒ “Clear runner caches”

2e exemple : nous voulons exécuter un pipeline avec un cache mais que pour une branche donnée

Ici le job1 ne tourne que sur la branche main

⇒ Créer une nouvelle branche et tester ce pipeline

⇒ comme le cache n'est pas vide, alors le job2 nous donne quand même un résultat avec le cat

⇒ Tester une seconde fois ce pipeline sur la seconde branche mais après avoir vidé les caches ⇒ ERROR

```
cache:
  - key:
    paths:
      - .lib
stages:
  - step1
  - step2
j1:
  stage: step1
  script:
    - mkdir -p .lib
    - echo test > .lib/test.txt
tags:
  - docker
only:
  - main
j2:
  stage: step2
  script: cat .lib/test.txt
tags:
  - docker
```

Cache

Comment peut-on faire pour gérer un cache par branche?

Définition d'une clef qui va se baser sur le nom de la branche

Donc il y aura un cache main et un cache pour la branche qui est à côté

Tester dans les deux branches

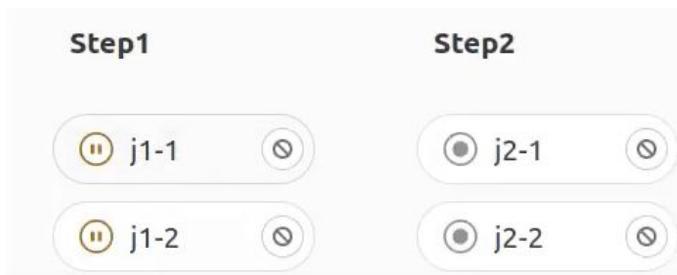
```
24 test
25 $ ls .lib/
26 test.txt
✓ 27 Saving cache for su
28 Creating cache test
```

```
cache:
  - key: $CI_COMMIT_REF_SLUG
    paths:
      - .lib
stages:
  - step1
  - step2
j1:
  stage: step1
  script:
    - mkdir -p .lib
    - echo $CI_COMMIT_REF_SLUG > .lib/$CI_COMMIT_REF_SLUG.txt
  tags:
    - docker
j2:
  stage: step2
  script:
    - cat .lib/$CI_COMMIT_REF_SLUG.txt
    - ls .lib/
  tags:
    - docker
```

Cache

L'utilisation du cache peut également s'effectuer au niveau des stages

Y ajouter les jobs2-1 et 2-2 dans le stage step2



```
stages:
  - step1
  - step2
j1-1:
  stage: step1
  script:
    - mkdir -p .lib
    - echo $CI_COMMIT_REF_SLUG-$CI_JOB_STAGE >
.lib/$CI_COMMIT_REF_SLUG-$CI_JOB_STAGE.txt
  cache:
    - key: $CI_JOB_STAGE-$CI_COMMIT_REF_SLUG
      paths:
        - .lib
tags:
  - docker
j1-2:
  stage: step1
  script:
    - cat .lib/$CI_COMMIT_REF_SLUG-$CI_JOB_STAGE.txt
    - ls .lib/
  cache:
    - key: $CI_JOB_STAGE-$CI_COMMIT_REF_SLUG
      paths:
        - .lib
tags:
  - docker
```

Cache

Plus généralement :

- cache par branche
cache: key: **\$CI_COMMIT_REF_SLUG**
- par job et branche
cache: key: "**\$CIJOBNAME-CI_JOB_NAME-\$CIJOBNAME-\$CI_COMMIT_REF_SLUG**"
- par stage et branche
cache: key: "**\$CIJOBSTAGE-CI_JOB_STAGE-\$CIJOBSTAGE-\$CI_COMMIT_REF_SLUG**"
- pour le partager entre le job de diff branches
cache: key: **\$CI_JOB_NAME**
- pour le partager partout (branches, jobs...)
cache: key: one-key-to-rule-them-all

Cache

Comment nettoyer les caches sur gitlab ?

- 1 - méthode graphique
- 2 - changer le nom de key

Cache - Policies

Différentes politiques possibles :

- pull : récupère le cache (début de job) mais ne le modifie pas
- push : ne récupère pas mais modifie le cache (fin de job)
- pull-push (default) : récupère et modifie ce que contient le cache

Cache - Policies

désactiver la mise à jour du cache (clean avant pour tester)

Le fichier créé en job1-1 ne sera pas
pushe a la fin du job

donc le job suivant sera en erreur

```
25 Saving cache for successful job
26 Not uploading cache step1-main-protected due to policy
27 Job succeeded
```

```
stages:
  - step1
  - step2
j1-1:
  stage: step1
  script:
    - mkdir -p .lib
    - echo $CI_COMMIT_REF_SLUG-$CI_JOB_STAGE >
.lib/$CI_COMMIT_REF_SLUG-$CI_JOB_STAGE.txt
  cache:
    - key: $CI_JOB_STAGE-$CI_COMMIT_REF_SLUG
      paths:
        - .lib
      policy: pull
  tags:
    - docker
j1-2:
  stage: step1
  script:
    - cat .lib/$CI_COMMIT_REF_SLUG-$CI_JOB_STAGE.txt
    - ls .lib/
  cache:
    - key: $CI_JOB_STAGE-$CI_COMMIT_REF_SLUG
      paths:
        - .lib
  tags:
    - docker
```

Cache - Policies

Possibilité de faire un cache masqué : créer un job masqué avec le keyword **extends** ce qui permet toujours de charger les datas

Le cache n'est pas montré pourtant il l'utilise bien

```
15 Checking cache for 0e171231e0817f1
16 No URL provided, cache will not be
17 Successfully extracted cache
▼ 18 Executing "step_script" stage of t
```

```
stages:
  - step1
.dependencies_cache:
  cache:
    - key: $CI_COMMIT_SHA
      paths:
        - .lib
j1-1:
  stage: step1
  script:
    - mkdir -p .lib
    - echo "toto" > .lib/test.txt
  extends: .dependencies_cache
  tags:
    - docker

j1-2:
  stage: step1
  script:
    - ls .lib
    - cat .lib/test.txt
  extends: .dependencies_cache
  tags:
    - docker
```

Les Artifacts

Deux modes de stockage pour gitlabci :

- cache : sur chaque runner
- artifacts : stocké sur l'instance gitlab (rapport de test par ex, ou un .jar ou un binaire...)

Deux types d'artefacts, disponibles une fois le pipeline terminé :

- pour les jobs : extrait pour chaque job
- pour les pipelines (en fin) : coverage

Attention : bien gérer ces éléments stockés car ils prennent de la place

⇒ possibilité de définir des durées d'expiration (conserver l'artifact un certain temps)

⇒ possible aussi de leur donner un espace de stockage

```
artifacts:  
  enabled: true  
  path: /mnt/storage/artifacts
```

Les Artifacts

Quelles sont leurs options ?

- **expire_in** : temps de conservation
- **untracked** : ajout des fichiers untracked (sens git) à votre artifact
- **exclude** : exclure certains fichiers
- **paths** : répertoire complet
- **expose_as** : mettre à disposition dans les MR
- **name** : nom associé

Les Artifacts

Nous ne créons pas volontairement de stages

On génère l'artefact avec ce fichier file.txt

On le retrouve sur le côté droit des logs du job

Plusieurs possibilités de downloads

```
job1:  
  script:  
    - echo "test" > file.txt  
  artifacts:  
    paths:  
      - file.txt
```

Job artifacts

These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

Status	Pipeline	Triggerer	Stages	
passed	Update .gitlab-ci.yml file #84 ⌚ main ➔ f2741268 🌐 latest			Download artifacts

Artifacts

Ajout d'un temps d'expiration

Rq: cette expiration ne s'applique que sur les précédents pipelines, pas sur le courant

Le lancer deux fois ⇒ attendre une min et vérifier sur le précédent job exécuté si l'artefact a bien été supprimé

On peut également lui ajouter un nom pour éviter l'association avec le job lors du download

```
job1:  
  script:  
    - echo "test" > file.txt  
  artifacts:  
    paths:  
      - file.txt  
    expire_in: 1m
```

Job artifacts

The artifacts were removed 2 minutes ago 

```
job1:  
  script:  
    - echo "test" > file.txt  
  artifacts:  
    paths:  
      - file.txt  
    expire_in: 1m  
    name: "$CI_JOB_NAME"
```

Artifacts

On va pouvoir aussi inclure ou exclure des fichiers ou dossiers

```
job1:  
  script:  
    - echo "test" > file.txt  
  artifacts:  
    paths:  
      - ./  
    expire_in: 1m  
    name: "$CI_JOB_NAME"  
    exclude:  
      - "*.yml"  
      - ".git/**"
```

```
21 Uploading artifacts...  
22 .: found 3 matching artifact f  
23 .git/**: excluded 79 files  
24 *.yml: excluded 1 files  
25 Uploading artifacts as "archiv
```

Artifacts

Retrouver les artifacts classés par job :

```
job1:  
  script:  
    - echo "test" > file.txt  
  artifacts:  
    paths:  
      - file.txt  
    expire_in: 1m  
    name: $CI_JOB_NAME  
  
job2:  
  script:  
    - echo "test" > file2.txt  
  artifacts:  
    paths:  
      - file2.txt  
    expire_in: 1m  
    name: $CI_JOB_NAME
```

Artifacts

On va pouvoir aussi exposer cet artifact au sein d'une MR

C'est cette valeur qui sera utilisée lors de la MR

1. Commiter ce pipeline dans main
2. Créer une branche test
3. Créer une MR :
 - a. Editer la branche test : y créer un nouveau fichier
 - b. Commit changes
 - c. cliquer sur MR
 - d. Artifact utilise



Title (required)

```
job1:  
  script:  
    - echo "test" > file.txt  
  artifacts:  
    paths: ['file.txt']  
    expire_in: 1m  
    name: $CI_JOB_NAME  
    expose_as: file_job
```

New file

test / test.txt
1 test

You pushed to test just now

Create merge request

Artifacts : ex avec un maven

Pusher dans un nouveau
projet le dossier myapp
avec son gitlabci a cote

Verifier la presence du .jar
dans les artifacts

```
-----  
 1446 Uploading artifacts for success  
 1447 Uploading artifacts...  
 1448 myapp1/target/*.jar: found 1 artifact  
 1449 Uploading artifacts as "archiver"  
      responseStatus=201 Created token
```

```
variables:  
  POSTGRES_DB: postgres  
  POSTGRES_USER: runner  
  POSTGRES_PASSWORD: postgres  
  POSTGRES_HOST_AUTH_METHOD: trust  
  MAVEN_OPTS: "-Dmaven.repo.local=${CI_PROJECT_DIR}/.m2/repository  
-Dstyle.color=always"  
Build_Jar:  
  image: maven:3.5.0-jdk-8  
  services:  
    - postgres  
  script:  
    - cd myapp1/  
    - 'mvn ${MAVEN_OPTS} -Drevision=x.y.z-SUFFIX -B clean install '  
  tags:  
    - docker  
  artifacts:  
    paths:  
      - myapp1/target/*.jar  
    expire_in: 7 days
```

Artifacts :

Comment ne conserver que le dernier ?

Menu ⇒ Projects Settings ⇒ CI/CD Expand Artifacts ⇒ Clear the Keep artifacts from most recent successful jobs checkbox

Comment aller supprimer à la main un artifact ?

1. aller sur la page de job
2. en haut à droite au-dessus des logs
3. Erase job log



SSH runner vers target

Pré-requis :

- créer une paire de clefs
- créer un user (avec les droits souhaités)
- ajouter la clef publique sur le serveur cible

Se connecter à la VM target1

1 - creation d'un user

pas de password pour la key gitlabkey

```
ssh-keygen -t ecdsa -b 521
sudo adduser gitlab
sudo mkdir -p /home/gitlab/.ssh
cat gitlabkey.pub | sudo tee
/home/gitlab/.ssh/authorized_keys

sudo chown -R gitlab:gitlab
/home/gitlab/.ssh
```

```
vagrant@target1:~$ ssh-keygen -t ecdsa -b 521
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_ecdsa): gitlabkey_
```

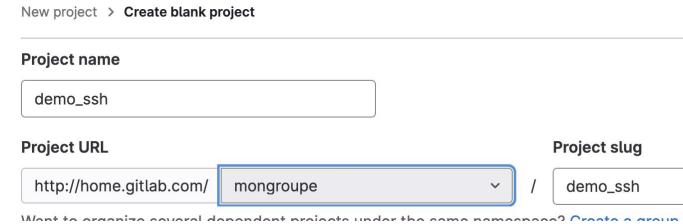
SSH runner vers target

```
sudo usermod -aG sudo gitlab  
id gitlab
```

2 - Creation d'un projet gitlab

3 - Settings ⇒ CI/Cd ⇒ activer le runner docker

4 - Settings ⇒ CI/CD ⇒ Variables ⇒ 4 variables à créer
SSH_PRIVATE

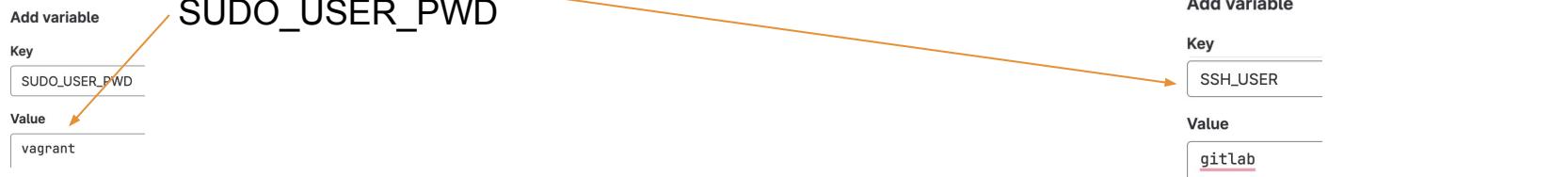


```
vagrant@target1:~$ cat gitlabkey  
-----BEGIN OPENSSH PRIVATE KEY-----  
-----END OPENSSH PRIVATE KEY-----
```

SSH_TARGET

SSH_USER

SUDO_USER_PWD



SSH runner vers target

Lancer un 1ier pipeline pour tester la connection

```
124 $ chmod 700 ~/.ssh
125 $ ssh-keyscan $SSH_TARGET >> ~/.ssh/known_hosts
126 # 192.168.12.43:22 SSH-2.0-OpenSSH_8.2p1
127 # 192.168.12.43:22 SSH-2.0-OpenSSH_8.2p1
128 # 192.168.12.43:22 SSH-2.0-OpenSSH_8.2p1
129 # 192.168.12.43:22 SSH-2.0-OpenSSH_8.2p1
130 # 192.168.12.43:22 SSH-2.0-OpenSSH_8.2p1
131 $ chmod 644 ~/.ssh/known_hosts
132 $ ssh $SSH_USER@$SSH_TARGET "hostname"
133 target1
134 Job succeeded
```

```
image: debian:latest # pour tous les jobs (surchargeable dans les jobs)
job1:
  before_script:
    - 'command -v ssh-agent >/dev/null || ( apt update && apt install -y openssh-client )'
    - eval $(ssh-agent -s)
    - echo "$SSH_PRIVATE" | tr -d '\r' | ssh-add -
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - ssh-keyscan $SSH_TARGET >> ~/.ssh/known_hosts #resoud le fingerprint du server
    - chmod 644 ~/.ssh/known_hosts
  script:
    - ssh $SSH_USER@$SSH_TARGET "hostname"
  tags:
    - docker
```

SSH runner vers target

Maintenant nous allons pouvoir utiliser ansible car nous avons une connection ssh

Ajouter l'install d'ansible dans le before_script :

```
- chmod 644 ~/.ssh/known_hosts  
- apt install -y ansible
```

⇒ remplacer le précédent script avec :

```
script:  
- ansible -i "$SSH_TARGET," all -u $SSH_USER -m command -a uptime
```

Rq : ici nous utilisons une image debian assez lourde, il serait préférable que vous élaboriez la votre plus légère par ex à partir d'une alpine

```
493 $ ansible -i "$SSH_TARGET," all -u $SSH_USER -m command -a uptime  
494 192.168.12.43 | CHANGED | rc=0 >>  
495 14:38:59 up 6:23, 2 users, load average: 0.08, 0.02, 0.01
```

SSH runner vers target

Nous voudrions installer nginx sur le server distant avec un playbook

Ajout du playbook dans nos fichiers

Puis appel de ce playbook dans le script du CI:

```
script:
  - echo "[all]" > inventory.ini
  - echo "$SSH_TARGET" >> inventory.ini
  - ANSIBLE_BECOME_PASS=$SUDO_USER_PWD ansible-playbook -i inventory.ini -l $SSH_TARGET -u $SSH_USER play.yml
```

ou mieux encore :

```
script:
  - echo "[all]" > inventory.ini
  - echo "$SSH_TARGET ansible_become_pass=$SUDO_USER_PWD" >> inventory.ini
  - ansible-playbook -i inventory.ini -l $SSH_TARGET -u $SSH_USER play.yml
```

```

main | / play.yml
      |
1   - name: my playbook
2     hosts: all
3     become: yes
4     tasks:
5       - name: t1
6         apt:
7           name: nginx
8           state: latest
9

```

SSH runner vers target

```
496 $ ANSIBLE_BECOME_PASS=$SUDO_USER_PWD ansible-playbook -i inventory.ini -l $SSH_TARGET -u $SSH_USER play.yml
497 PLAY [my playbook] ****
498 TASK [Gathering Facts] ****
499 ok: [192.168.12.43]
500 TASK [t1] ****
501 changed: [192.168.12.43]
502 PLAY RECAP ****
503 192.168.12.43 : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
504 Job succeeded
```

```
vagrant@target1:~$ curl localhost
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
```

Ancres

Elles permettent de gérer les répétitions et donc de factoriser notre code

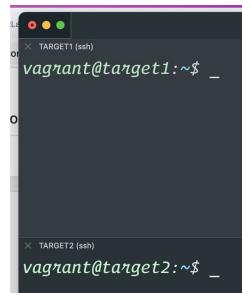
On repart de notre exemple précédent avec le serveur target1 avec ses variables

L'idée est que nous pourrions devoir gérer plusieurs servers avec des variations à chaque server

Ici, avec target2, nous aurions deux users et ip différents dans notre action ansible

Donc nous pouvons supprimer au niveau du projet les variables SSH_USER et SSH_TARGET

L'idéal serait de changer les 4 bien sur



Ancres

✗ TARGET2 (ssh)

```
vagrant@target2:~$ sudo adduser gitlab2_
```

```
vagrant@target2:~$ sudo mkdir /home/gitlab2/.ssh  
vagrant@target2:~$ sudo usermod -aG sudo gitlab2_
```

✗ TARGET1 (ssh)

```
vagrant@target1:~$ cat gitlabkey.pub  
ecdsa-sha2-nistp521 AAAAE2VjZHNhLXNoYTItbm&ldquo;zdHA1MjEAAAIBm&ldquo;zdHA1I  
B+&ldquo;Qi&ldquo;ZcqLwwoZ+EbB1BEuEDWvL2k7jU2qUpwEUplvgDuUeOGyCFqMdqpPr9XJ/K,  
= vagrant@target1  
vagrant@target1:~$ _
```

✗ TARGET2 (ssh)

```
vagrant@target2:~$ sudo vim /home/gitlab2/.ssh/authorized_keys_
```

✗ TARGET2 (ssh)

```
vagrant@target2:~$ sudo chown gitlab2:gitlab2 /home/gitlab2/.ssh
```

Ancres

```
vagrant@target2:~$ curl localhost  
curl: (7) Failed to connect to localhost port 80: Connection refused
```

Les ancrez ne sont pas spécifiques à gitlabCI, c'est juste du YAML

Même logique par exemple dans docker-compose

1 - déterminer le contenu de l'ancre

2 - dupliquer le job1 dans le job2

3 - Définir les variables au niveau des jobs

4 - Committer et tester les 2 jobs

```
job1:  
  variables:  
    SSH_USER: gitlab  
    SSH_TARGET: 192.168.12.43
```

```
job2:  
  variables:  
    SSH_USER: gitlab2  
    SSH_TARGET: 192.168.12.44
```

Ancres

Variable cachée et nom de l'ancre

```
1 image: debian:latest
2
3 .ssh: &ssh
4
```

Puis le bloc before_script est répété

Insertion dans l'ancre et appel dans les jobs

Committer et vérifier les 2 jobs

```
.ssh: &ssh
- 'command -v ssh-agen
- eval $(ssh-agent -s)
- echo "$SSH_PRIVATE"
- mkdir -p ~/.ssh
- chmod 700 ~/.ssh
- ssh-keyscan $SSH_TAR
- chmod 644 ~/.ssh/kno
- apt install -y ansib
```

```
job1:
variables:
  SSH_USER: gitlab
  SSH_TARGET: 192.168.
before_script: *ssh
script:
```

Ancres

Seconde factorisation possible : bloc ansible

Par la suite il est possible de mettre en place une librairie d'actions

Tester un curl sur nginx dans le target2

```
.ansible: &ansible
  - echo "[all]" > inventory
  - echo "$SSH_TARGET" >> inventory
  - ANSIBLE_BECOME_PASS=$SSH_PASSWORD

job1:
  variables:
    SSH_USER: gitlab
    SSH_TARGET: 192.168.1.1
  before_script: *ssh
  script: *ansible
  tags:
```

Templates et Sécurité

! Dependency scanning detected **10 new critical** and **22 new high** severity vulnerabilities out of 72. ?

New

- Critical Bypass of a protection mechanism in libxslt in nokogiri
- Critical Command Injection in nokogiri
- Critical Improper Input Validation in rails
- Critical Path Traversal in rubyzip
- Critical Insufficient Entropy in cryptiles

Templates et Sécurité

Static Application Security Tests (SAST) sert à vérifier votre code source pour les vulnérabilités connues.

Vous pouvez exécuter des analyseurs SAST dans n'importe quel niveau GitLab.

Les analyseurs génèrent des rapports au format JSON sous forme d'artefacts de tâche.

Templates et Sécurité

Beaucoup d'autres tests possibles:

Type de numérisation	Niveau minimum	Conditions préalables	Exigences de candidature
Tests de sécurité des applications statiques (SAST)	Gratuit	Aucun	Voir les exigences SAST
Détection secrète	Gratuit	Aucun	Aucun
Analyse des conteneurs	Gratuit	Image de conteneur créée et transférée vers le registre	Docker 18.09.03 ou supérieur installé sur le même ordinateur que le runner
Analyse de l'infrastructure en tant que code (IaC)	Gratuit	Aucun	Voir les langages et frameworks pris en charge
Analyse des dépendances : inclut la conformité des licences	Ultime	Aucun	L'application doit utiliser l'un des langages et gestionnaires de packages pris en charge
Tests dynamiques de sécurité des applications (DAST)	Ultime	Application cible déployée	Voir les options d'analyse GitLab DAST
Tests de fuzz guidés par la couverture	Ultime	Version instrumentée de l'application	Voir les moteurs et langues de fuzzing pris en charge
Tests fuzz de l'API Web	Ultime	Application cible déployée	Voir les types d'API pris en charge

Templates et Sécurité

Pour activer l'analyse de sécurité pour un projet, vous devez disposer des éléments suivants :

- un projet GitLab qui répond aux exigences de l'analyse de sécurité que vous choisissez d'activer, avec CI activé
- un fichier `.gitlab-ci.yml` pour le projet pour lequel au moins une tâche de build est définie
- un GitLab Runner basé sur Linux avec l'exécuteur Docker ou Kubernetes

Templates et Sécurité

- Avec AutoDevOps activé et son build auto:

Registry Docker

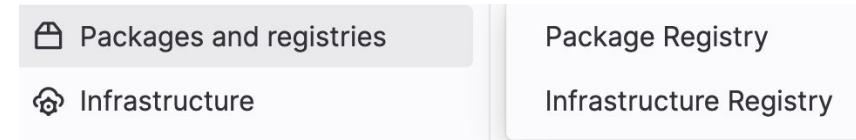
Comment installer une registry docker avec un certificat auto signé?

New Project ⇒ myregistry

Pour l'instant il n'y a pas de registry docker

Il faudra donc :

- activer la registry
- bénéficier d'une registry verified via https
- génération d'un certificat auto-signé



Registry Docker

Se connecter en root sur l'instance gitlab

```
vagrant@gitlab:~$ sudo -s
root@gitlab:/home/vagrant# cd /etc/gitlab
root@gitlab:/etc/gitlab# ls
gitlab-secrets.json  gitlab.rb  initial_root_password  trusted-certs
root@gitlab:/etc/gitlab# vim gitlab.rb _
```

Registry Docker

Nous avons plusieurs choses à activer dans gitlab :

```
gitlab_rails['gitlab_default_projects_features_container_registry'] = true  
  
registry_external_url 'https://registry.home.gitlab'  
  
gitlab_rails['registry_enabled'] = true  
gitlab_rails['registry_host'] = "registry.home.gitlab"  
  
registry_nginx['enable'] = true  
registry_nginx['listen_port'] = 443  
registry_nginx['ssl_certificate'] = "/etc/gitlab/registry.home.gitlab.cert"  
registry_nginx['ssl_certificate_key'] = "/etc/gitlab/registry.home.gitlab.key"
```

registry visible par défaut pas oblig. pour tous les users

utiliser le port 443 sur ce dns

Registry Docker

Il pourrait être intéressant de mettre un disque spécifique pour le stockage des layers :

```
### Settings used by GitLab application
gitlab_rails['registry_enabled'] = true
gitlab_rails['registry_host'] = "registry.home.com"
# gitlab_rails['registry_port'] = "5005"
# gitlab_rails['registry_path'] = "/var/opt/gitlab/gitlab-rails/shared/registry"
```

Évite de faire planter l'instance gitlab lorsqu'elle est trop remplie des images

Registry Docker

```
#####
## Container Registry settings
##! Docs: https://docs.gitlab.com/ee/administration/
#####

registry_external_url 'https://registry.home.com'

### Settings used by GitLab application
gitlab_rails['registry_enabled'] = true
gitlab_rails['registry_host'] = "registry.home.com"
"gitlab_rails['registry_port'] = "5005"
```

Puis rechercher registry_nginx

Registry Docker

Nous allons activer un serveur nginx qui est déjà dans le package de gitlab

```
#####
## Registry NGINX
#####
```

```
registry_nginx['enable'] = true
registry_nginx['listen_port'] = 443
registry_nginx['ssl_certificate'] = "/etc/gitlab/registry.home.gitlab.cert"
registry_nginx['ssl_certificate_key'] = "/etc/gitlab/registry.home.gitlab.key"
```

Registry Docker

Nous devons maintenant générer ce certificat auto-signé depuis /etc/gitlab en root

```
sudo openssl req -newkey rsa:4096 -nodes -sha256 -keyout  
registry.home.com.key -addext "subjectAltName = DNS:registry.home.com"  
-x509 -days 365 -out registry.home.com.cert
```

```
Country Name (2 letter code) [AU]:  
State or Province Name (full name) [Some-State]:  
Locality Name (eg, city) []:  
Organization Name (eg, company) [Internet Widgits Pty Ltd]:  
Organizational Unit Name (eg, section) []:  
Common Name (e.g. server FQDN or YOUR name) []:registry.home.gitlab
```

```
root@gitlab:/etc/gitlab# ls -la  
total 184  
drwxrwxr-x  3 root root  4096 Feb 23 10:37 .  
drwxr-xr-x 101 root root  4096 Feb 23 09:38 ..  
-rw-----  1 root root 19408 Feb 15 15:08 gitlab-secrets.json  
-rw-----  1 root root 140408 Feb 23 10:35 gitlab.rb  
-rw-----  1 root root   749 Feb 15 12:03 initial_root_password  
-rw-r--r--  1 root root  2069 Feb 23 10:37 registry.home.gitlab.cert  
-rw-----  1 root root  3268 Feb 23 10:35 registry.home.gitlab.key  
drwxr-xr-x  2 root root  4096 Feb 15 12:03 trusted-certs
```

Registry Docker

Il faut ensuite reconfigurer gitlab car nous avons modifié le fichier rb

```
gitlab-ctl reconfigure
```

Cela relance l'orchestrateur chef

```
gitlab-ctl status nginx
```

```
gitlab Reconfigured!
root@gitlab:/etc/gitlab# gitlab-ctl status nginx
run: nginx: (pid 15721) 105s; run: log: (pid 10336) 3687s
```

Registry Docker

Rafraîchir page du repo pour voir si la registry docker est exploitable puis se logger



Package Registry
Container Registry
Infrastructure Registry

Two-Factor Authentication is enabled, use a Personal Access Token !!

```
docker login registry.home.com
```

You can add an image to this registry with the following command:

```
docker build -t registry.home.com/mongroupe/myregistry .
```

```
docker push registry.home.com/mongroupe/myregistry
```

Registry Docker

Or le client pour le moment ne connaît pas notre certificat

il faut donc récupérer le certificat sur le server et l'enregistrer sur un fichier local

```
docker login registry.home.gitlab
openssl s_client -showcerts -connect registry.home.com:443 < /dev/null | sed -ne '/-BEGIN
CERTIFICATE-/,/-END CERTIFICATE-/p' > registry.crt
cat registry.crt
sudo cp registry.crt /usr/local/share/ca-certificates/
sudo update-ca-certificates
systemctl restart docker
docker login registry.home.gitlab
```

```
sudo security add-trusted-cert -d -r trustRoot -k /Library/Keychains/System.keychain $CERT
```

Registry Docker

relancer le client docker sur la machine hôte

```
$docker login registry.home.com  
Username: root  
Password:  
Login Succeeded
```

```
$docker pull alpine:latest
```

```
docker tag alpine:latest registry.home.com/mongroupe/myregistry/myalpine:v1.0  
docker push registry.home.com/mongroupe/myregistry/myalpine:v1.0
```

Rafraîchir la page sur gitlab

Registry Docker

Nous pouvons également paramétrer ce registry

Settings ⇒ Packages & Registries

Garder un certain nombre d'images avec un tag et fréquence du cleanup

Utilisation d'une regex

Comment utiliser ce registry ?

Nous allons builder une image. Pour cela :

- utilisation de docker in docker
- prêter une attention particulière au certificat de la registry

Nous allons devoir ajouter le certificat sur la vm du runner

Création d'un nouveau projet mybuild

Ajouter la VM runner2 comme runner docker

Comment utiliser ce registry ?

Vérifier que le container registry soit bien disponible

La question sera : comment utiliser un certificat sur le runner2?

Sur le runner2, on a un docker engine qui porte un runner qui va pouvoir lancer d'autres containers

On relance la commande précédente :

```
openssl s_client -showcerts -connect registry.home.com:443 < /dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p' | sudo tee /usr/local/share/ca-certificates/registry.crt  
update-ca-certificates
```

Comment utiliser ce registry ?

Une fois que ce certificat a bien été enregistré, comment le passer aux différents containers qui s'exécutent ?

Nous avons la possibilité dans la conf du runner de poser un montage qui se placera sur n'importe quelle image

```
docker inspect gitlab-runner | grep "Source" |  
head -n 1
```

⇒ volume hébergeant sa conf

```
vagrant@runner2:~$ sudo vim /data/gitlab-runner/config.toml
```

Comment utiliser ce registry ?

Pour exploiter le dind

```
[runners.cache.azure]
[runners.docker]
  tls_verify = false
  image = "debian:latest"
  privileged = true
  disable_entrypoint_overwrite = false
```

Montage des certs dans la vm
vers le container

```
oom_kill_disable = false
disable_cache = false
volumes = ["/cache", "/etc/ssl/certs:/etc/ssl/certs:ro"]
shm_size = 0
```

```
vagrant@runner2:~$ docker restart gitlab-runner
```

Comment utiliser ce registry ?

Création ensuite de notre 1ère image

Donc mise en place d'un fichier
Dockerfile dans le projet

File templates Dockerfile ▾ Cho...

```
1 FROM debian:11.2
2
3 RUN apt update && apt install -y nginx
4
```

Puis dans l'editor du CI/CD

```
stages:
  - build
build image:
  stage: build
  image: docker # pour utiliser les commandes docker
  services:
    - name: docker:dind
      alias: docker
  variables:
    DOCKER_BUILDKIT: "1" #pour les performances
    DOCKER_DRIVER: overlay2 #un peu plus performant pour les builds
    DOCKER_HOST: tcp://docker:2375 #indispensable, correspond a l'alias
    DOCKER_TLS_CERTDIR: "" #reset du cert par defaut
  script:
    - docker build -t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA
      -t $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG
      .
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA
    - docker push $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG
  tags:
    - docker
```

https://docs.gitlab.com/ee/ci/pipelines/downstream_pipelines.html

Attention une partie de cette partie est payante et ne rentre pas dans le free tiers

Créer deux projets ⇒ pipeline1 et pipeline2

Le 1 va déclencher le 2 ⇒ il va y envoyer deux variables

Pour le pipeline2 :

Lui adjoindre

le runner2 docker

```
image: debian:latest
job2:
  only:
    - pipelines
  script:
    - echo $VAR1
    - echo $VAR2
  tags:
    - docker
```

dans le cas d'un déclenchement
par un autre pipeline

Pipeline Multiprojet

Vérifier aussi que le runner 2 docker est dispo pour le pipeline1

Nouvel élément

depend : dependances entre les
2 projets, s'affiche en graphique

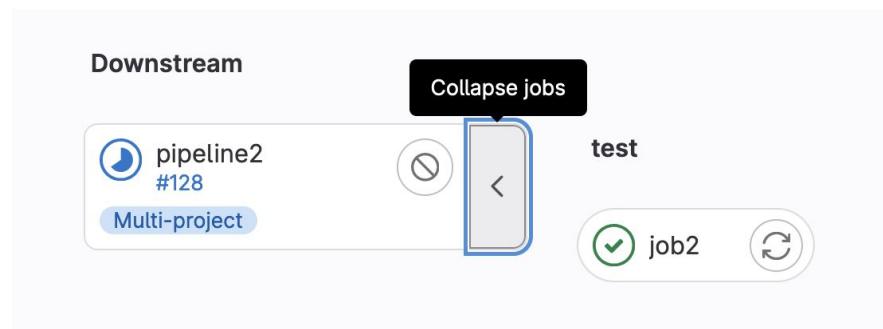
Pour le 1ier commit, commenter les
excepts, sinon il ne validera pas le
gitlabci

```
stages:
  - s1
  - s2
job1-1:
  stage: s1
  image: docker
  script:
    - echo "Mon premier job !!!"
  tags:
    - docker
  except:
    changes:
      - ".gitlab-ci.*"
job1-2:
  variables:
    VAR1: "variable1"
    VAR2: "variable2"
  stage: s2
trigger:
  project: mongroupe/pipeline2
  branch: main
  strategy: depend
except:
  changes:
    - ".gitlab-ci.*"
```

Pipeline Multiprojet

Lancer à la main le pipeline1

ce que permet l'option strategy: depend ⇒



Multiprojet avec Ansible

Nous allons reprendre le schema précédent :

- le pipeline 2 va jouer Ansible
- et va surcharger des valeurs passées depuis le pipeline 1

Les tâches seront les suivantes :

- installer un serveur postgres ou à vérifier qu'il est déjà installé
- enregistrer un user
- installer une DB dans ce postgres

Multiprojet avec Ansible

```
vagrant@target2:~$ sudo adduser gitlab
```

```
vagrant@target2:~$ sudo usermod -aG sudo gitlab
```

```
vagrant@target2:~$ sudo su - gitlab
```

```
gitlab@target2:~$ ssh-keygen -t ecdsa -b 521
Generating public/private ecdsa key pair.
Enter file in which to save the key (/home/gitlab/.ssh/id_ecdsa):
```

Multiprojet avec Ansible

```
gitlab@target2:~$ cat .ssh/id_ecdsa.pub > .ssh/authorized_keys
```

```
gitlab@target2:~$ cat .ssh/authorized_keys  
ecdsa-sha2-nistp521 AAAAE2VjZHNhLXNoYTItbmlzd
```

Nous utiliserons la clé privée sur le server gitlab

Création d'un nouveau projet [ansible_dp](#)

New project > Create blank project

Project name

ansible_dp

Project URL

<http://home.gitlab.com/> mongroupe

Want to organize several dependent projects und

Visibility Level [?](#)

-  Private
Project access must be granted explicitly to e
-  Internal
The project can be accessed by any logged in
-  Public
The project can be accessed without any aut

Project Configuration

- Initialize repository with a README
Allows you to immediately clone this project's
- Enable Static Application Security Testing (SA

Multiprojet avec Ansible

Dossier pg ⇒

- playbook deploy-postgres.yml
- nous allons auto implémenter la partie inventory

```
$ls
deploy-postgres.yml      env
$tree env
env
└── dev
    ├── group_vars
    │   └── all.yml
    │       └── myapp1.yml
    └── hosts
        └── pg.yml
└── master
    ├── group_vars
    │   └── all.yml
    │       └── myapp1.yml
    └── hosts
        └── pg.yml
└── stage
    ├── group_vars
    │   └── all.yml
    │       └── myapp1.yml
    └── hosts
        └── pg.yml
```

postgres

Multiprojet avec Ansible

Un dictionnaire pg_settings compose de 2 dictionnaires :

db_name pour la création de la db et users

Car dans les tasks de ce rôle postgres :

- installation postgres
- install psycopg pour permettre à ansible de gérer notre instance pg
- modif dans la conf postgres pour écouter sur l'ensemble des interfaces
- création du user
- et de la DB

```
$cat env/dev/group_vars/pg.yml
pg_settings:
  db_name:
    - { name: "{{env}}", owner: "vagrant" }
  users:
    - { name: "vagrant", password: "vagrant" }
```

```
$vim postgres/tasks/main.yml
```

Multiprojet avec Ansible

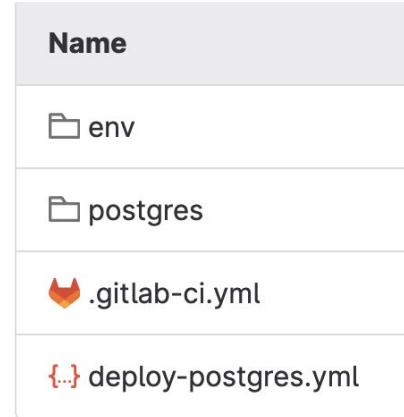
Depuis le dossier pg `git init`

Ajouter le remote dont la commande a été donnée lors de la création du projet

```
git remote add origin http://home.gitlab.com/mongroupe/ansible_dp.git
```

```
$git add .  
$git commit -m "init"
```

```
create mode 100644 postgres  
$git push -u origin main
```



Multiprojet avec Ansible

Puis nous allons lui passer des variables dans CI/CD

Add variable

Key

SSH_PRIVATE

Add variable

Key

SSH_USER

Value

gitlab

gitlab@target2:~\$ cat .ssh/id_ecdsa

-----BEGIN OPENSSH PRIVATE KEY-----

b3B1bnNzaC1rZXktdiEAAAAABG5vbmlUAAAEEh

Add variable

Key

SSH_TARGET

Value

192.168.12.44

gitlab@target2:~\$ hostname -I
10.0.2.15 192.168.12.44

Multiprojet avec Ansible

Add variable

Key

SUDO_USER_PWD

Value

vagrant

Puis il nous reste à poser nos éléments de déclenchements

⇒ vérifier que le runner docker est disponible

⇒ Explication du gitlab-ci avec commentaires

Multiprojet avec Ansible

Nous allons remplacer le ci du pipeline1 précédent

Committer les changements



Verifier sur target2 le status de postgres

```
stages:
  - build
  - database
build image:
  stage: build
  image: docker
  script:
    - echo "first part"
    - docker
  tags:
    - docker

db:
  variables:
    PG_USER: "user1"
    PG_PASSWORD: "password"
    PG_DB: "mydatabase"
  stage: database
  trigger:
    project: mongroupe/ansible_dp
    branch: main
    strategy: depend
```

Multiprojet avec Ansible

```
493 $ mkdir -p /etc/ansible/
494 $ echo "[defaults]" > /etc/ansible/ansible.cfg
495 $ echo "allow_world_readable_tmpfiles=true" >> /etc/ansible/ansible.cfg
496 $ echo "[pg]" > $ANSIBLE_INVENTORY
497 $ echo "$SSH_TARGET ansible_become_pass=$SUDO_USER_PWD" >> $ANSIBLE_INVENTORY
498 $ ANSIBLE_BECOME_PASS=$SUDO_USER_PWD ansible-playbook -i $ANSIBLE_INVENTORY -l :
  '{"pg_settings": {"db_name": [{"name": "$PG_DB", "owner": "$PG_USER"}]} }'
499 [DEPRECATION WARNING]: ALLOW_WORLD_READABLE_TMPFILES option, moved to a per
500 plugin approach that is more flexible, use mostly the same config will work,
```

```
gitlab@target2:~$ psql -h 127.0.0.1 -U user1 mydatabase
```

mydatabase⇒ \l	
Name	Owner
mydatabase	user1

Multiprojet build jar

Nous allons continuer avec ce précédent travail d'ansible ou la partie build était plutôt symbolique

Ce build permettra de construire une image docker en base java puis la pusher sur le registry local dans le 1ier pipeline

Modifier le nom du pipeline1 en

Naming, to

Update your prc

Project name

myapp

Nous pourrions par la suite avoir un autre dépôt ansible qui déployera l'application

Multiprojet build jar

Cloner localement myapp

L'inconvénient c'est que Maven aura ses propres certificats dans son container et si nous venons les surcharger en écrasant /etc/ssl/certs, nous ne pourrons plus les résoudre. Donc on ne monte qu'un seul certificat et on n'efface pas les autres

```
vagrant@runner2:~$ sudo vim /data/gitlab-runner/config.toml
```

```
/usr/local/share/ca-certificates/registry.crt:/etc/ssl/certs/registry.crt:ro
```

```
network_mode = "gitlab_default"
volumes = ["/cache", "/usr/local/share/ca-certificates/registry.crt:/etc/ssl/certs/registry.crt
:ro"]
```

Multiprojet build jar

Ajouter dans le dépôt copie local le dossier myapp1 pour construire notre projet

Tout pusher sur le repo

Nous pouvons maintenant modifier le pipeline myapp

Variables en haut

Puis trois stages

```
variables:  
  VARIABLES_FILE: ./variables.txt  
  POSTGRES_DB: postgres  
  POSTGRES_USER: runner  
  POSTGRES_PASSWORD: postgres  
  POSTGRES_HOST_AUTH_METHOD: trust  
  MAVEN_OPTS:  
    "-Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository  
    -Dstyle.color=always"  
cache:  
  paths:  
    - .m2
```

```
stages:  
  - test  
  - build  
  - database
```

Multiprojet build jar

Etape de test

```
test:
  stage: test
  image: maven:3.5.0-jdk-8
  services:
    - postgres
  script:
    - mvn ${MAVEN_OPTS} -B clean test
    - MVN_VERSION=$(mvn --non-recursive help:evaluate -Dexpression=project.version | grep -v '\[.*' | perl -nle 'm{(.*)?([0-9]+\.[0-9]+\.[0-9]+)};print $2')
    - echo $MVN_VERSION
    - echo "export MVN_VERSION=$MVN_VERSION" > ${VARIABLES_FILE}
  artifacts:
    paths:
      - $VARIABLES_FILE
  tags:
    - docker
```

Multiprojet build jar

Build du jar

```
build_jar:
  stage: build
  image: maven:3.5.0-jdk-8
  services:
    - postgres
  script:
    - 'mvn ${MAVEN_OPTS} -Drevision=x.y.z-SUFFIX -B clean install'
  artifacts:
    paths:
      - target/*.jar
    expire_in: 7 days
  tags:
    - docker
```

Multiprojet build jar

construction de l'image docker

```
build_docker:
  needs:
    - test
    - build_jar
  stage: build
  image: docker
  services:
    - name: docker:dind
      alias: docker
  variables:
    DOCKER_BUILDKIT: "1"
    DOCKER_DRIVER: overlay2
    DOCKER_HOST: tcp://docker:2375
    DOCKER_TLS_CERTDIR: ""
```

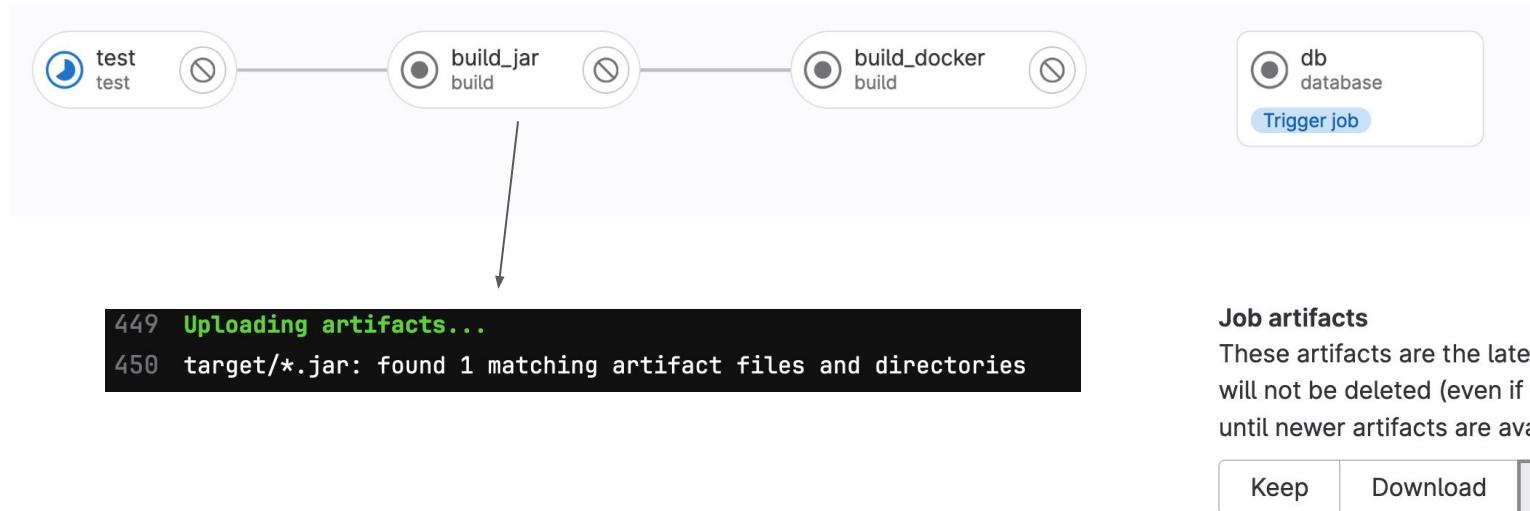
Multiprojet build jar

ajout des scripts pour cette étape de build docker

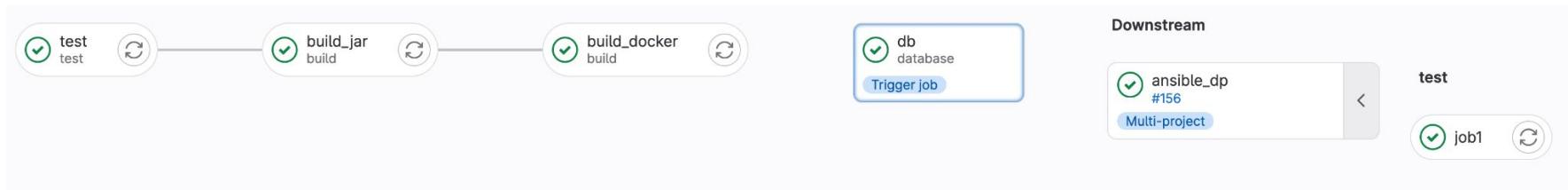
```
script:
  - source ${VARIABLES_FILE}
  - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
  - docker build  -t $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA
                  -t $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG
                  -t $CI_REGISTRY_IMAGE:${MVN_VERSION}
                  .
  - docker push  $CI_REGISTRY_IMAGE:$CI_COMMIT_SHORT_SHA
  - docker push  $CI_REGISTRY_IMAGE:$CI_COMMIT_REF_SLUG
  - docker push  $CI_REGISTRY_IMAGE:${MVN_VERSION}
tags:
  - docker
```

Multiprojet build jar

Après le push :



Multiprojet build jar



<input type="checkbox"/> 3 tags	<input type="button" value="Delete Selected"/>
<input type="checkbox"/> ac0b8699	Published 5 minutes ago Digest: 5fbcd54
251.07 MiB	
<input type="checkbox"/> main	Published 5 minutes ago Digest: 5fbcd54
251.07 MiB	
<input type="checkbox"/> v0.0.1	Published 5 minutes ago Digest: 5fbcd54
251.07 MiB	

Pipeline Multiprojet : ajout d'étapes

Nous allons pouvoir ajouter des améliorations et un scan de sécurité avec Trivy

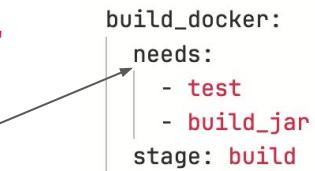
⇒

- Option maven ajouter : -DskipTests car sans cette option il joue toutes les étapes antérieures de tests en plus, donc permet d'aller plus vite

- 'mvn \${MAVEN_OPTS} -Drevision=x.y.z-SUFFIX -DskipTests -B clean install'

- keyword needs: faible différence avec les dependencies

- dependencies permet de récupérer les artifacts de job précédents
 - needs même chose plus en permettant de l'acyclic graph (traits graphiques entre jobs)



Pipeline Multiprojet : ajout d'étapes

Possibilité avec needs d'agir plus finement et donc de choisir quel artifact nous voulons ou non récupérer

Intérêt : toujours gagner du temps

Example of `needs:artifacts:`

```
test-job1:  
  stage: test  
  needs:  
    - job: build_job1  
      artifacts: true  
  
test-job2:  
  stage: test  
  needs:  
    - job: build_job2  
      artifacts: false  
  
test-job3:  
  needs:  
    - job: build_job1  
      artifacts: true  
    - job: build_job2  
    - build_job3
```

Pipeline Multiprojet : ajout d'étapes

Puis nous ajoutons l'option cache-from avec le docker build :

⇒ au lieu de builder une image docker de 0 juste a partir du dockerfile

Nous voulons dans ce cas exploiter le tag latest de notre image

script:

```
- source ${VARIABLES_FILE}
- docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
- docker build --cache-from $CI_REGISTRY_IMAGE/myapp:latest
  -t $CI_REGISTRY_IMAGE/myapp:v${MVN_VERSION}
  -t $CI_REGISTRY_IMAGE/myapp:latest .
- docker push $CI_REGISTRY_IMAGE/myapp:v${MVN_VERSION}
- docker push $CI_REGISTRY_IMAGE/myapp:latest
```

Pipeline Multiprojet : ajout d'étapes

Puis nous ajoutons une étape de scan compose de plusieurs parties :

```
scan:  
  stage: scan  
  needs:  
    - test  
    - build_jar  
    - build_docker
```

dépendances

```
image:  
  name: aquasec/trivy  
  entrypoint: [""] #permet de sourcer la version de notre MVN  
  VERSION  
  
variables:  
  TRIVY_AUTH_URL: $CI_REGISTRY # on stocke tout dans la registry gitlab  
  TRIVY_USERNAME: $CI_REGISTRY_USER  
  TRIVY_PASSWORD: $CI_REGISTRY_PASSWORD
```

conf de l'environnement

Pipeline Multiprojet : ajout d'étapes

Puis nous ajoutons une étape de scan compose de plusieurs parties :

```
script:  
  - source ${VARIABLES_FILE}  
  - trivy --cache-dir .cache image  
    --exit-code 0  
    --severity HIGH,CRITICAL  
    --format table --output report.md  
    --vuln-type os  
    $CI_REGISTRY_IMAGE/myapp:v${MVN_VERSION}
```

commandes et arguments
Utilisation du cache de gitlab
exit code a 0 pour forcer le
déroulé du pipeline
Ca permet au moins de générer
un rapport
Ici nous demandons un scan d'os

```
artifacts:  
  name: "Scan Report ${CI_COMMIT_SHA}"  
  paths:  
    - report.md  
  expire_in: 7 days  
  when: on_failure  
cache:  
  paths:  
    - ".cache"  
tags:  
  - docker
```

dernière étape :
stockage du rapport de
scan dans les artifacts

Pipeline Multiprojet : déploiement

Les objectifs seront :

- installation user gitlab sur target1 et ajout dans groupe sudo (déjà fait, à vérifier)
- création variables de connexion ssh
- rôle docker avec ansible sur target1
- déploiement test
- rôle deploy

Nous travaillons toujours avec les repos myapp (code applicatif) et ansible_dp (sert juste à l'install de postgres)

Pipeline Multiprojet : déploiement

```
vagrant@target1:~$ sudo adduser gitlab
```

```
vagrant@target1:~$ sudo su - gitlab  
gitlab@target1:~$
```

Puis se connecter au second server target2

```
vagrant@target2:~$ sudo su - gitlab  
gitlab@target2:~$ cat .ssh/authorized_keys  
ecdsa-sha2-nistp521 AAAAE2VjZHNhLXNoYTItbmlz
```

Pipeline Multiprojet : déploiement

Copier cette key dans le .ssh/authorized_keys de target1

```
gitlab@target1:~$ sudo vim .ssh/authorized_keys
```

```
SGKEcKuWKO1Dwa4DdFYCW1  
/V0u17w27xj9HI13oYeVS0  
w= gitlab@target1
```

⇒ modifier target2 en target1

Pipeline Multiprojet : déploiement

Copier cette key dans le .ssh/authorized_keys de target1

```
gitlab@target1:~$ sudo vim .ssh/authorized_keys
```

```
SEKECKNWK01DWa4DdFYCW1  
/V0u17w27xj9HI13oYeVS0  
w= gitlab@target1
```

→ modifier target2 en target1

dans target1

Repo myapp ⇒ Settings ⇒ CI/CD ⇒ Variables

```
ssh-keygen -t ecdsa -b 521  
sudo adduser  
sudo usermod -aG sudo gitlab  
sudo su - gitlab  
mkdir .ssh  
vim .ssh/authorized_keys
```

Pipeline Multiprojet : déploiement

SSH_PRIVATE ⇒ depuis la clé privée générée dans target1

SSH_TARGET ⇒ ip target1 : 192.168.12.43

SSH_USER ⇒ gitlab

SUDO_USER_PWD ⇒ vagrant

Le but sera que notre gitlabCI appelle un autre trigger pour installer l'applicatif et le docker

Pipeline Multiprojet : déploiement

dossier install-app avec ansible avec lequel nous allons créer un nouveau projet
SANS README.md : **ansible_myapp**

Sur notre machine target1, il va nous falloir notre certificat pour notre registry

```
openssl s_client -showcerts -connect registry.home.com:443 < /dev/null |  
sed -ne '/-BEGIN CERT  
IFICATE-/,-END CERTIFICATE-/p' >  
/usr/local/share/ca-certificates/registry.crt  
update-ca-certificates
```

```
gitlab@target1:~$ sudo -s  
[sudo] password for gitlab:  
root@target1:/home/gitlab# opens
```

```
root@target1:/home/gitlab#  
Updating certificates in /etc/ssl/certs...  
rehash: warning: skipping certificate or CRL  
rehash: warning: skipping certificate or CRL  
1 added, 0 removed; done.  
Running hooks in /etc/ca-c  
done.
```

Pipeline Multiprojet : déploiement

Maintenant le
gitlab-ci de ce projet

:

```
image: debian:latest
ansible_myapp:
    variables:
        ENV: "dev"
        ANSIBLE_DIR: "env/$ENV/"
        ANSIBLE_INVENTORY: "$ANSIBLE_DIR/inventory.ini"
    only:
        - pipelines
    before_script:
        - 'command -v ssh-agent >/dev/null || ( apt update && apt install -y openssh-client )'
        - eval $(ssh-agent -s)
        - echo "$SSH_PRIVATE" | tr -d '\r' | ssh-add -
        - mkdir -p ~/.ssh
        - chmod 700 ~/.ssh
        - ssh-keyscan $SSH_TARGET >> ~/.ssh/known_hosts
        - chmod 644 ~/.ssh/known_hosts
        - apt install -y ansible
    script:
        - mkdir -p /etc/ansible/ $ANSIBLE_DIR
        - echo "[defaults]" > /etc/ansible/ansible.cfg
        - echo "allow_world_readable_tmpfiles=true" >> /etc/ansible/ansible.cfg
        - echo "[myapp]" > $ANSIBLE_INVENTORY
        - echo "$SSH_TARGET ansible_become_pass=$SUDO_USER_PWD" >> $ANSIBLE_INVENTORY
        - ANSIBLE_BECOME_PASS=$SUDO_USER_PWD
            ansible-playbook -i $ANSIBLE_INVENTORY -l $SSH_TARGET -u $SSH_USER install-myapp.yml
    tags:
        - docker
```

Pipeline Multiprojet : déploiement

Nous avions créé notre clé ssh dans l'autre dépôt myapp et l'avantage avec le mode trigger c'est que nous pourrons pousser ces valeurs dans une variable qui sera exploitée dans ansible_myapp La partie before_script nous permet l'install d'ansible et openssh et le script permet d'éditer la conf et l'inventory d'ansible puis de jouer le playbook ⇒ Committer Nous pouvons retourner maintenant sur myapp

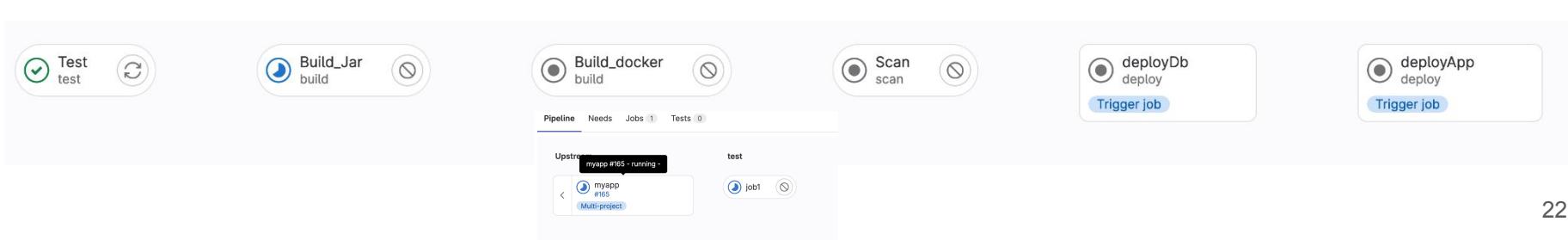
Pipeline Multiprojet : déploiement

Voici donc le trigger que nous allons réaliser :

A ajouter en bas du pipeline de myapp, nous envoyons ainsi les variables au précédent job ansible_myapp

Committer et tout vérifier

```
deployApp:  
  needs:  
    - test  
    - build_jar  
    - build_docker  
    - scan  
    - deployDb  
  stage: deploy  
  variables:  
    SSH_TARGET: $SSH_TARGET  
    SSH_PRIVATE: $SSH_PRIVATE  
    SUDO_USER_PWD: $SUDO_USER_PWD  
    SSH_USER: $SSH_USER  
  trigger:  
    project: mongroupe/ansible_myapp  
    branch: main  
    strategy: depend
```



Starter-Pack

Administrator > 🌐 starter-pack > Issue Boards

Gestion de projet ▾

Search

Open

4

+

Reunion de lancement

Animation

#2 2h



Atelier de definitions des fonctionnalites

Animation

#3 1h



Creation de l'Infra as Code

Conception

#4



Initialiser le projet Symfony

Développement

#5



Open

2

+

Creation de l'Infra as Code

Conception

INFRA

#4



Initialiser le projet Symfony

Développement

Back

#5



Sprint

2

+

Reunion de lancement

Animation

#2 2h



Atelier de definitions des fonctionnalites

Animation

Conception

Feature

#3 1h

