Dawan
Training is our way

GITLAB

I. Découverte de Gitlab
II. Gitlab runner
III. CICD dans Gitlab
IV. SCRUM dans Gitlab
V. Workflow Gitlab

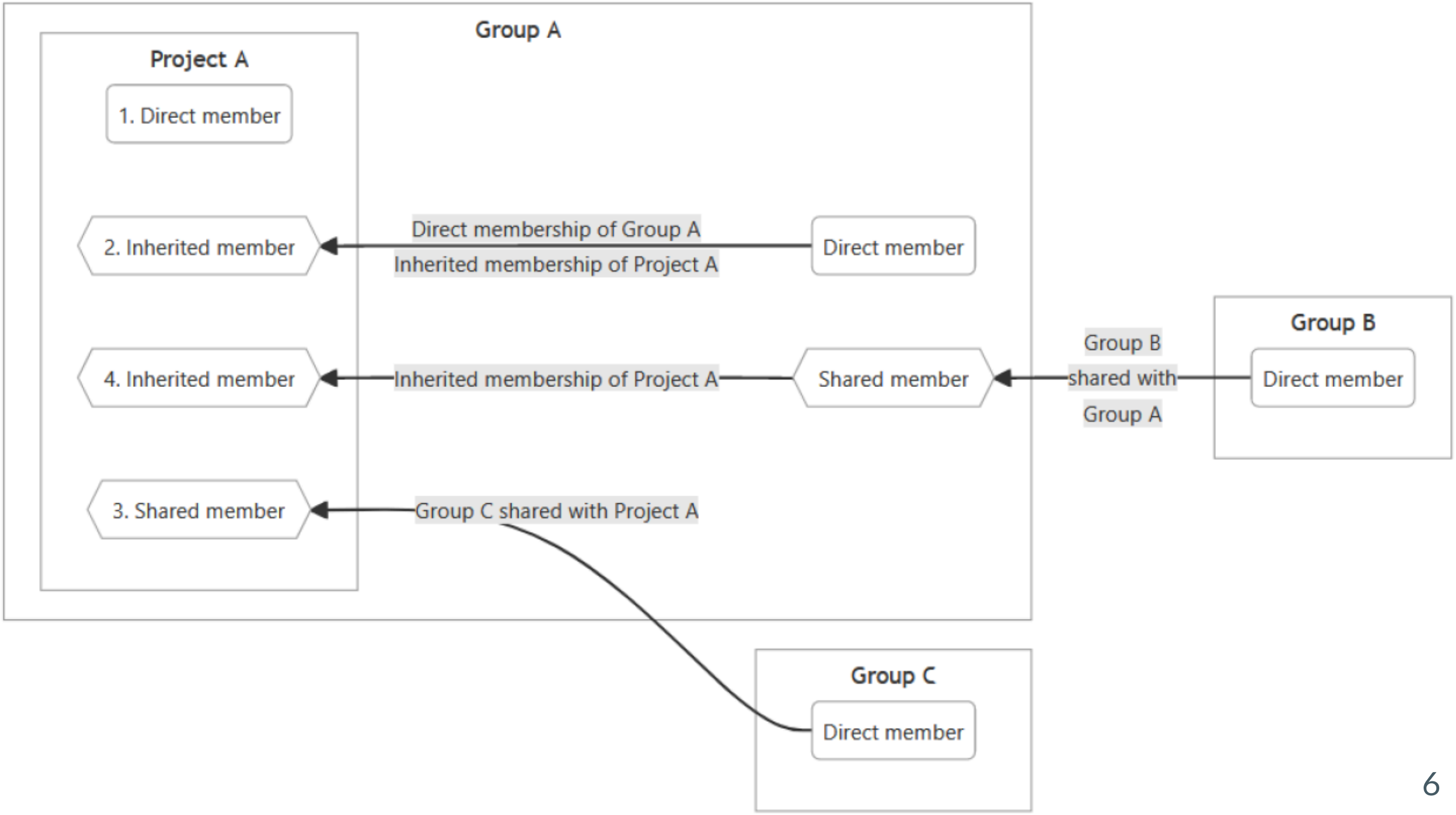
I. Découverte de Gitlab

accès (à la VM formation)

- Login / mot de passe
  Url Bridge: **https://gitlab.lan.fr**
  Url NAT: **https://gitlab.lan.fr:8443** ou **https://127.0.0.1:8443**
  Login: **root** ou **dev_a** ou **dev_b**
  Mdp: **R00tt00R**
- Mailhog (compte fake centralisant tous les appels SMTP)
  - *Url DNS:* **http://gitlab.lan.fr:8025**
  - *Url sans DNS:* **http://127.0.0.1:8025**

Concepts organisationnels dans gitlab

- **Utilisateurs:** peuvent appartenir à un *projet* ou un *groupe*
- **Projets:** gèrent un *dépôt git* - peuvent appartenir à un *groupe*
- **Groupes:** groupes de *projets*.
- **Espaces de noms:** Les utilisateurs et les groupes définissent des namespaces pour dédoublonner les noms de projets
  - un projet dans le namespace utilisateur n'est connu que de cet utilisateur : *projet personnel*
  - deux groupes peuvent définir un projet de même nom : *backend/monitoring* et *frontend/monitoring*


Group A
Project A
1. Direct member
2. Inherited member
Direct membership of Group A
Inherited membership of Project A
Direct member
4. Inherited member
Inherited membership of Project A
Shared member
Group B
shared with
Group A
Group B
Direct member
3. Shared member
Group C shared with Project A
Group C
Direct member

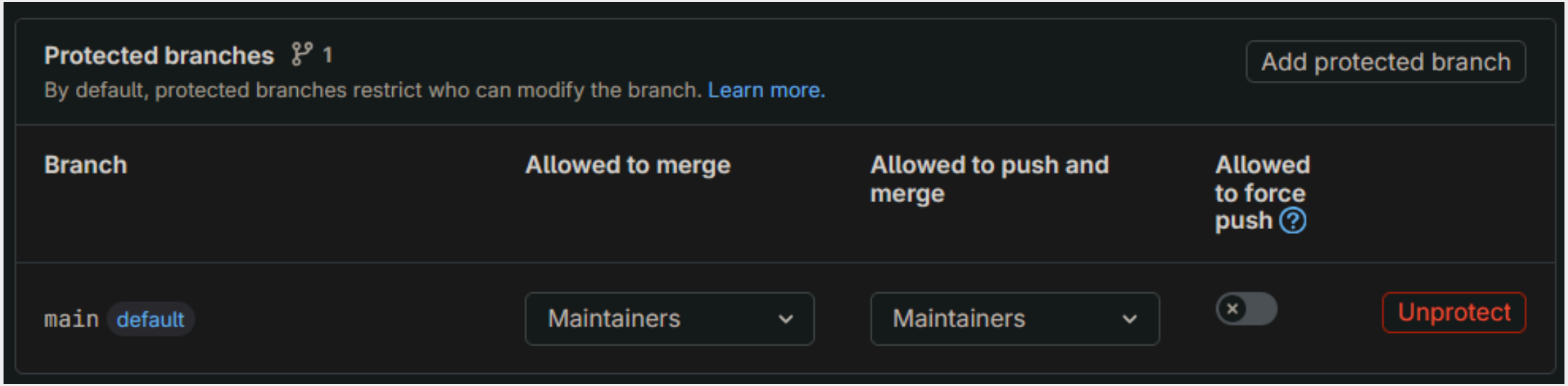
droits d'accès

- **Owner:** créateur d'un projet / groupe - *tous les droits*
- **Maintainer:** *Admin* - tout sauf suppression + « *Settings* » projet / groupe
- **Developper:** écriture sur le *code, merge requests, issues, CI / CD*
- **Reporter:** droits de *commenter* - écriture sur *issues, reviews*
- **Guest:** droit de *lecture*

  ici

branches protégées

- par défaut la branche par défaut `main` est **protégée**
- Qui peut fusionner / pousser (en force) sur cette branche ?
- *Project > Settings > Repository > Protected branches*



accès API

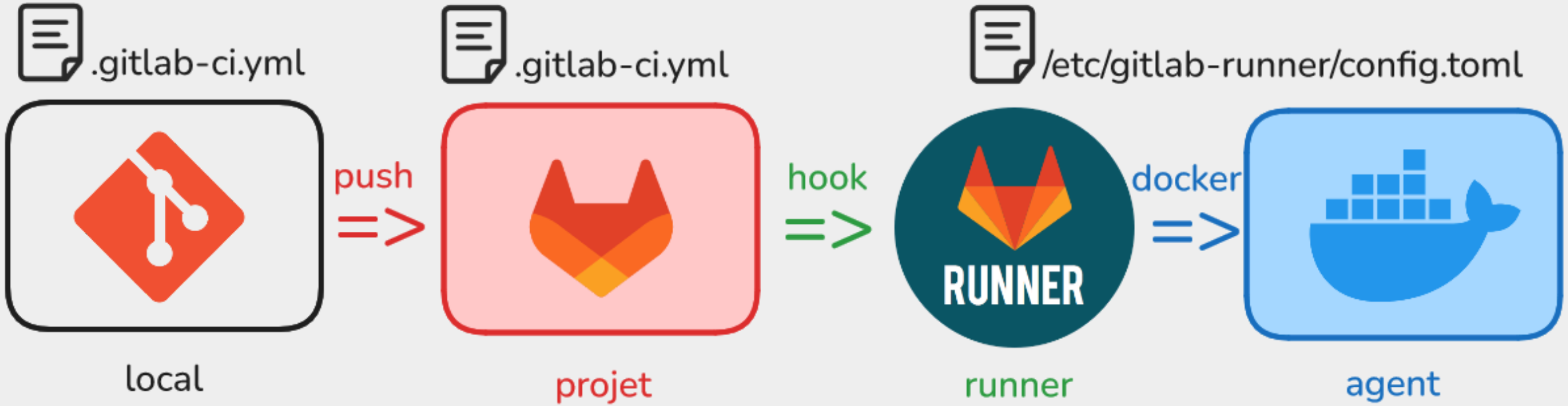
- UI: Jeton d'accès à l'API
  - *Avatar utilisateur > Preferences > « Acess Token » > api*

```
## IDEM AVEC GITLAB-RAILS
sudo gitlab-rails console
> user = User.find_by_username('automation-bot')
> token = user.personal_access_tokens.create(scopes: [:api], name: 'Automation token')
> token.set_token('name-your-own-token')
> token.save!

gitlab-rails runner "user = User.find_by_username;..." ## IDEM en ligne !

## utilisation dans un CLIENT d'API: ex. python
# pip install --upgrade python-gitlab
import gitlab
gl = Gitlab(url="https://<host>", token="name-your-own-token")
```

II. Gitlab Runner

présentation

- **gitlab-runner** composant indépendant de Gitlab chargé du *CI/CD*
- configure des *processus appelés runners*
- un **runner** exécute des *jobs* écrits dans le fichier **.gitlab-ci.yml**
- Un **job** est exécuté dans un *agent* pouvant être :
  - une machine physique
  - une VM *Virtualbox, VMWare, Hyper-V ...*
  - un conteneur *Docker, LXC, Kubernetes, ...*


.gitlab-ci.yml
.gitlab-ci.yml
/etc/gitlab-runner/config.toml
push
=>
hook
=>
RUNNER
docker
=>
local
projet
runner
agent

configuration

- Un runner est associé à un *projet/groupe* d'instances de gitlab
- *projet/groupe => « Settings » => « CICD » => « Runners »*
- fichier de configuration: **/etc/gitlab-runner/config.toml**

```
concurrent = 4 # nb de jobs exécutables en parallèle

[runners.docker]
volumes = [
"/cache",
"/etc/gitlab/trusted-certs/<host>.crt:/etc/gitlab-runner/certs/ca.crt:ro",
"/var/run/docker.sock:/var/run/docker.sock"
]# dns interne au réseau docker pour un gitlab en local
extra_hosts = ["gitlab.lan.fr:172.17.0.1"]
```

III. CICD dans Gitlab

YAML: introduction

- format de représentation de données par *sérialisation*, conçu pour être *aisément modifiable et lisible*
- Dérivé de la représentation d'un objet *JSON déplié*

```
{                                       ---
  "key" : "value",                      key: value
  "other key" : "other value", =>       other key: other value
  "num" : "3.14"                        num: 3.14
  "bool" : "false",                     bool: false
  "none" : "null"                       none: null
}                                       "with quotes": "possible"
```

YAML: imbrications

- objets JSON **{ ... }** => *indentation* de 2 / 4 chars.
- listes JSON **[...]** => *indentation* de 2 / 4 chars. **+ « - [espace] »**

```
{                                          ---
  "object" : {                             object:
  "key" : "value",                  =>       key: value
  "items" : [                                 items:
    "item1",                                    - item1
    { "k1": "v1", "k2": ... },                  - k1: v1
                                                  k2: v2
    "item3"                                     - item3
  ]
}
```

YAML: alias et ancres

```yaml
struct: &ref          ## "&": nomme un objet réutilisable
  k1: v1
  k2: v2

copy:
  my_alias: *ref      ## "*": copie exacte du contenu de "struct:"

extends:
  my_anchor:          ## "<<: *" : copie + ajout ou modifications
    <<: *ref
    k2: v2-bis
    k3: new-value
```

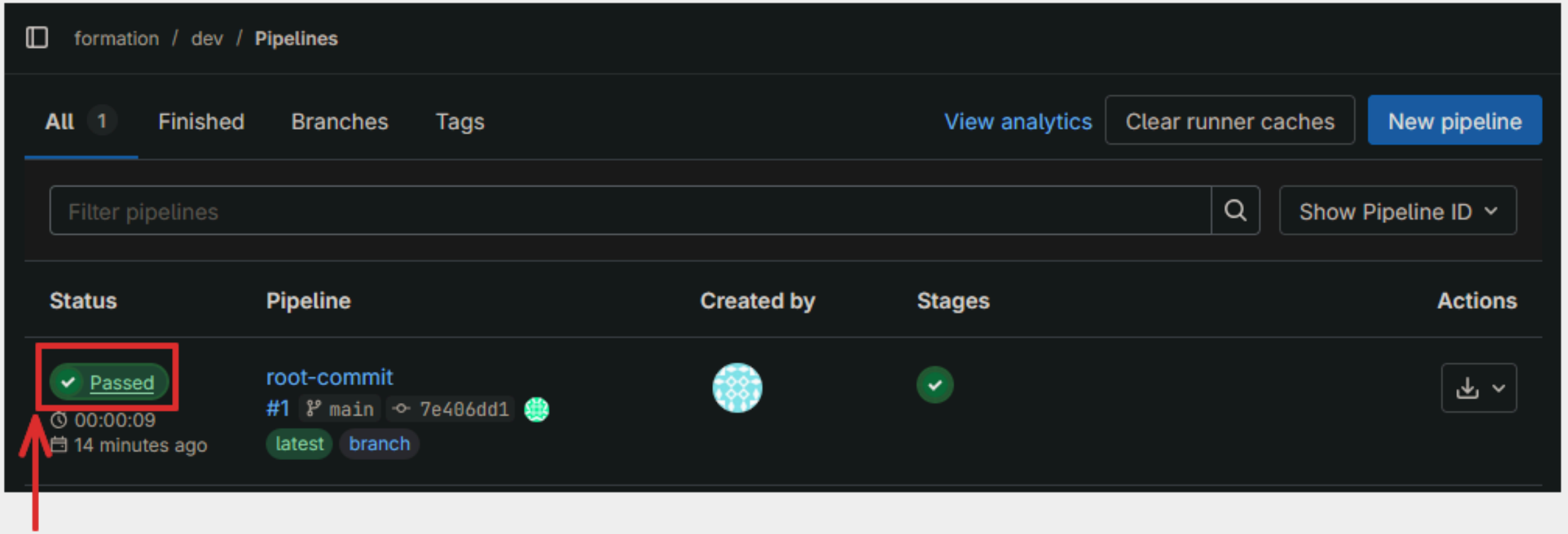
écrire un job

- Un job est un *objet yaml* dans le fichier **.gitlab-ci.yml**
  - dont le nom est *arbitraire*
  - contient une liste **script[]** contenant *des lignes de commandes*
  - on ajoute une liste de **tags[]** correspondants aux *runners* susceptibles d'exécuter le job
  - **before_script[]:** exécute des commandes *préparatoires*
  - **after_script[]:** exécute des commandes de *nettoyage*

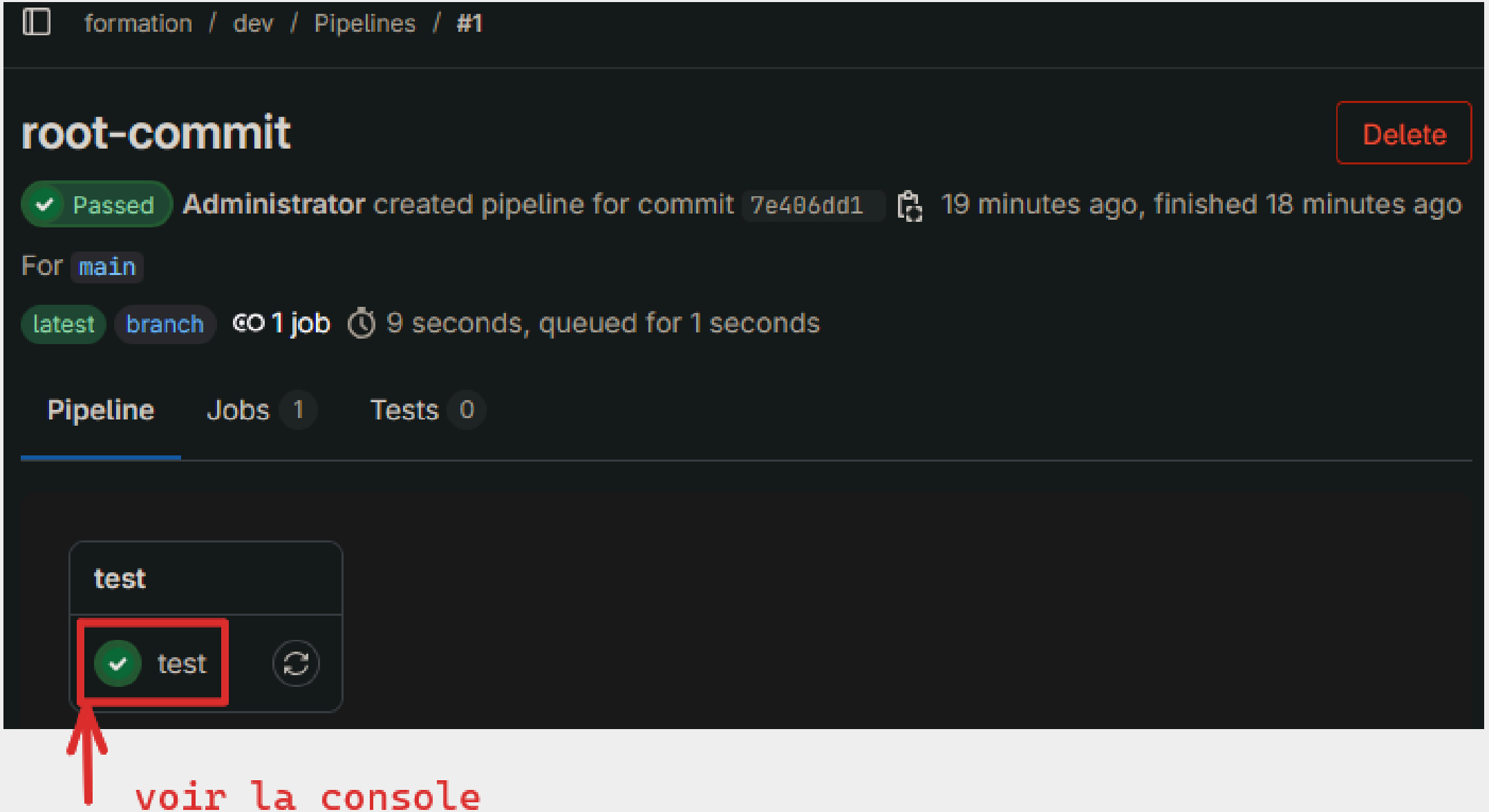
```yaml
unit tests:
  before_script: [ ... ]
  script:
    - echo "launch tests here !"
    - ...
  after_script: [ ... ]
  tags:
    - myusine
    - ...
```

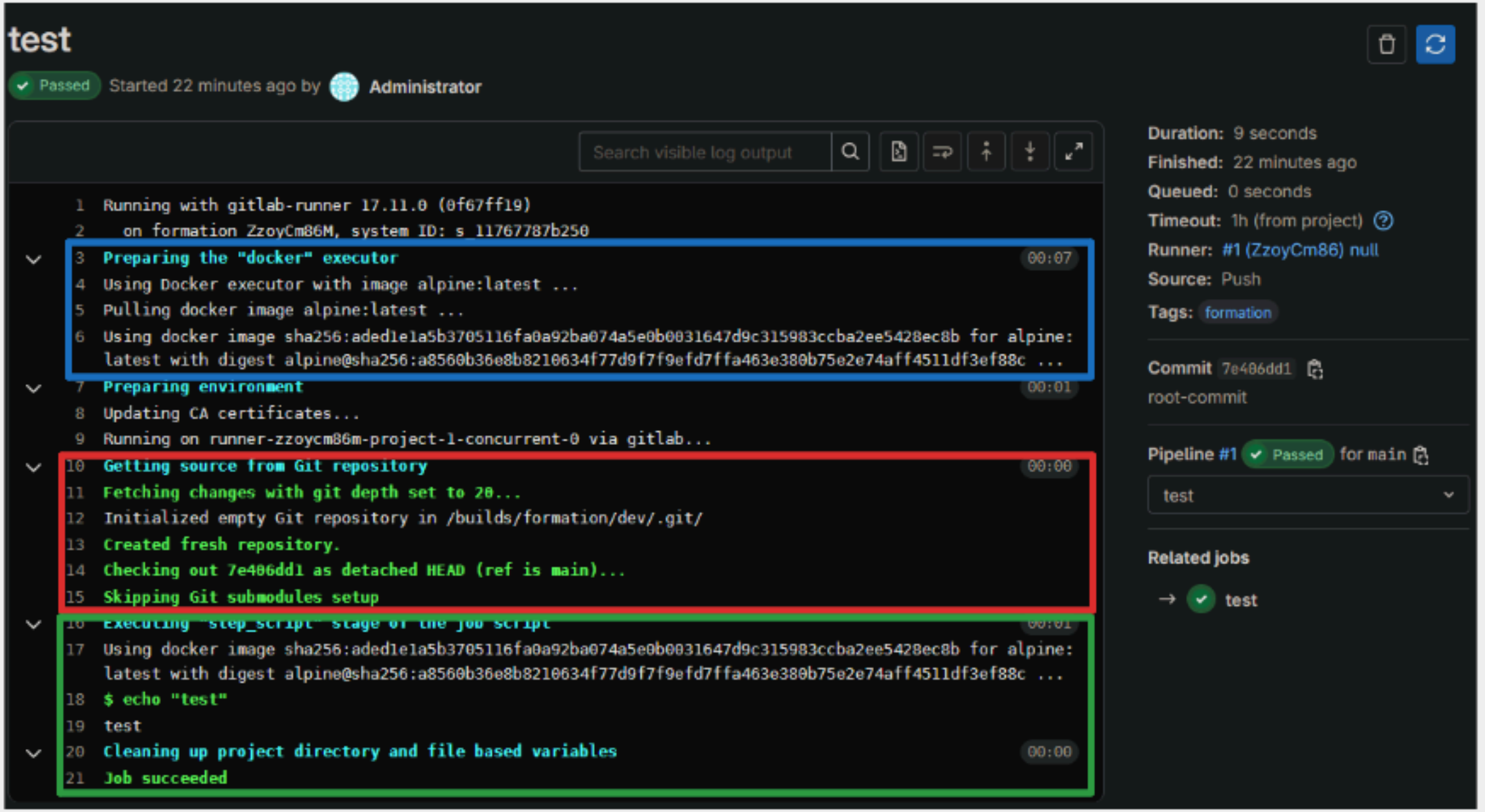
- préfixer le job avec **"."** => `.unit tests:` *désactive le job*

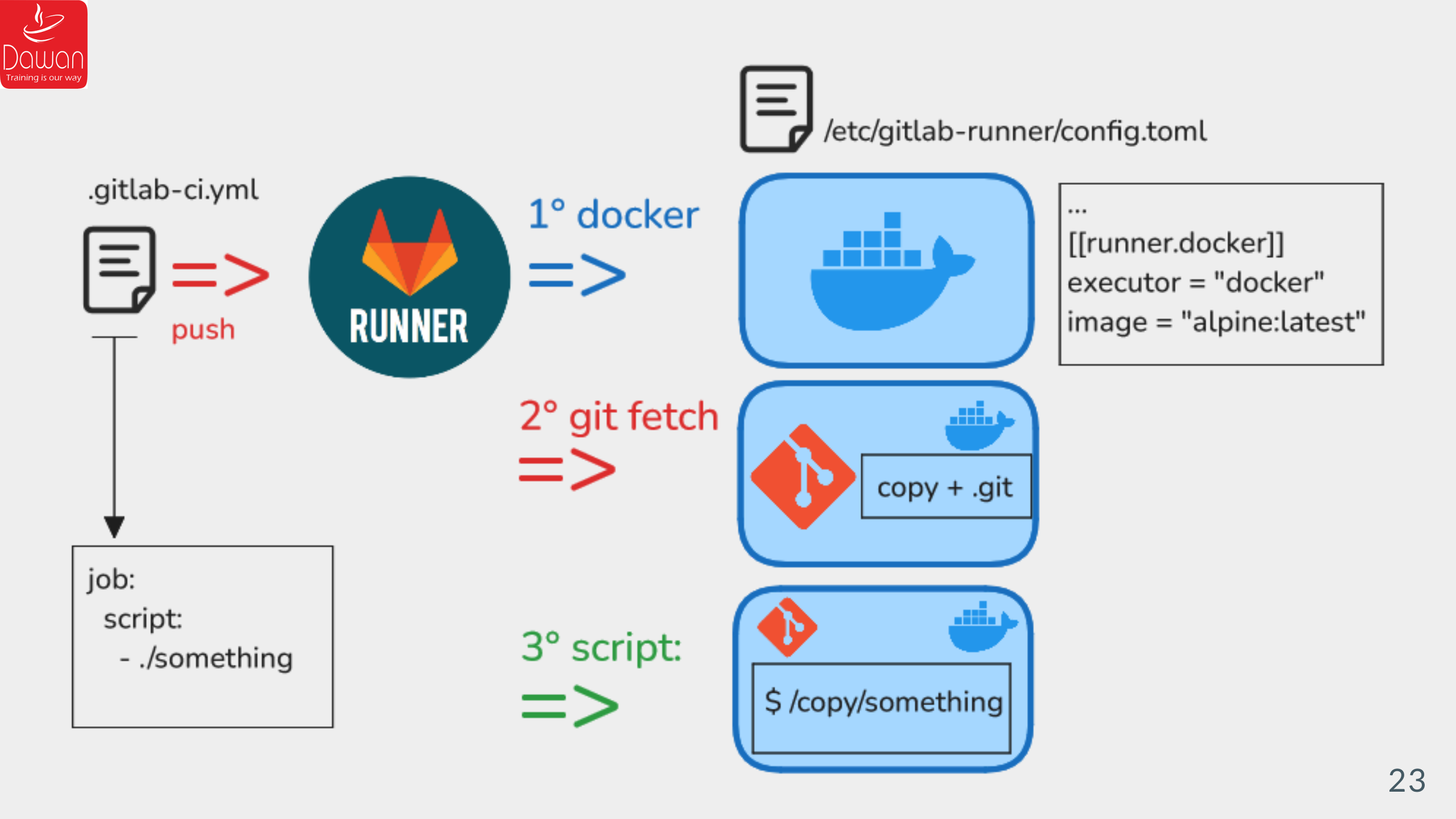
> Chaque fois que le fichier est poussé sur gitlab, on peut observer l'exécution du job dans la section à gauche **Buid > pipelines**



voir le pipeline

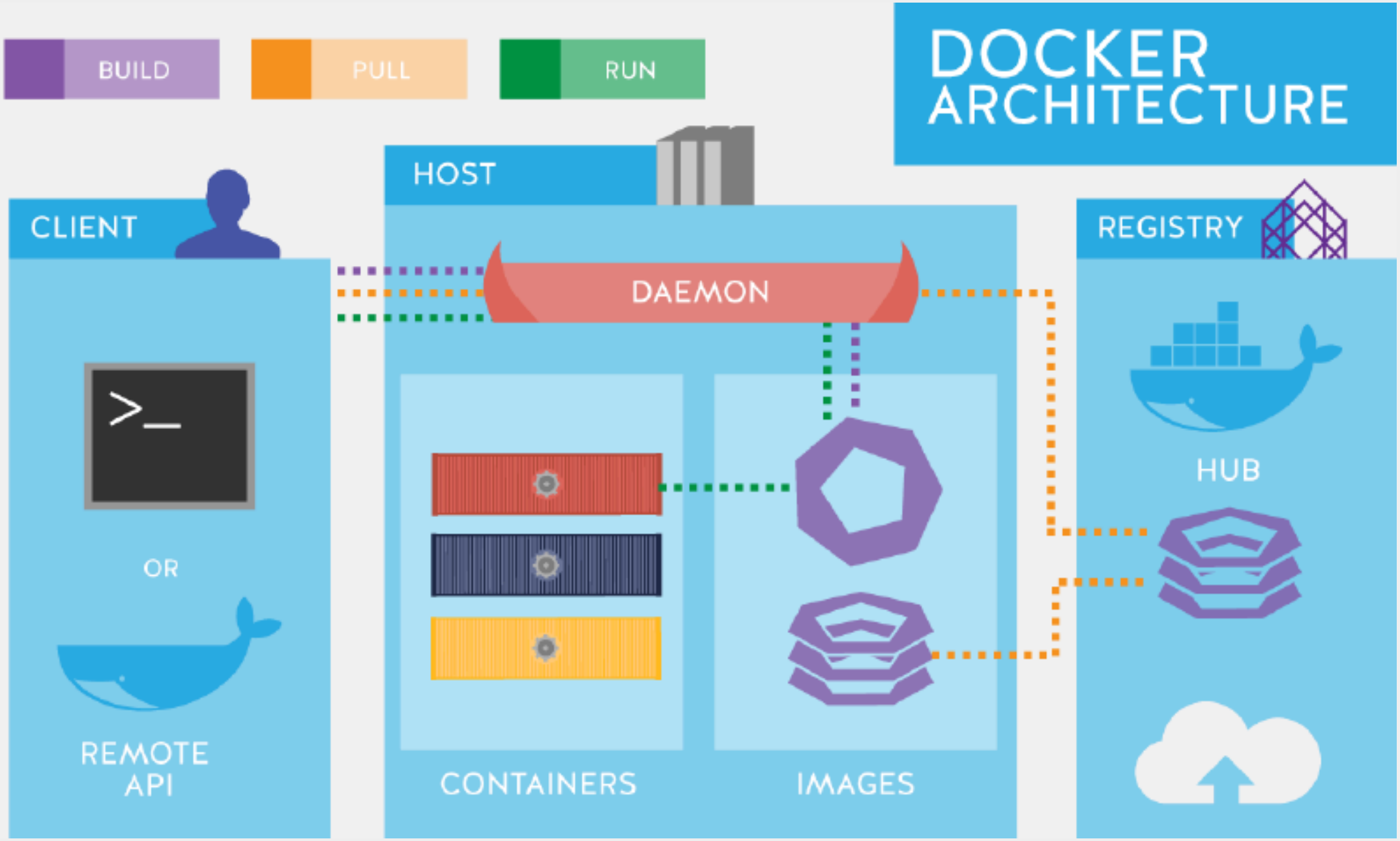

formation / dev / Pipelines / #1
root-commit
Delete
Passed
Administrator created pipeline for commit 7e406dd1
19 minutes ago, finished 18 minutes ago
For main
latest
branch
1 job
9 seconds, queued for 1 seconds
Pipeline
Jobs 1
Tests 0
test
test
voir la console


test
Passed
Started 22 minutes ago by
Administrator
Search visible log output
1 Running with gitlab-runner 17.11.0 (0f67ff19)
2 on formation ZzoyCm86M, system ID: s_11767787b250
1°
3 Preparing the "docker" executor 00:07
4 Using Docker executor with image alpine:latest ...
5 Pulling docker image alpine:latest ...
6 Using docker image sha256:aded1e1a5b3705116fa0a92ba074a5e0b0031647d9c315983ccba2ee5428ec8b for alpine:latest with digest alpine@sha256:a8560b36e8b8210634f77d9f7f9efd7ffa463e380b75e2e74aff4511df3ef88c ...
7 Preparing environment 00:01
8 Updating CA certificates...
9 Running on runner-zzoycm86m-project-1-concurrent-0 via gitlab...
2°
10 Getting source from Git repository 00:00
11 Fetching changes with git depth set to 20...
12 Initialized empty Git repository in /builds/formation/dev/.git/
13 Created fresh repository.
14 Checking out 7e406dd1 as detached HEAD (ref is main)...
15 Skipping Git submodules setup
3°
16 Executing "step_script" stage of the job script 00:01
17 Using docker image sha256:aded1e1a5b3705116fa0a92ba074a5e0b0031647d9c315983ccba2ee5428ec8b for alpine:latest with digest alpine@sha256:a8560b36e8b8210634f77d9f7f9efd7ffa463e380b75e2e74aff4511df3ef88c ...
18 $ echo "test"
19 test
20 Cleaning up project directory and file based variables 00:00
21 Job succeeded
Duration: 9 seconds
Finished: 22 minutes ago
Queued: 0 seconds
Timeout: 1h (from project)
Runner: #1 (ZzoyCm86) null
Source: Push
Tags: formation
Commit 7e406dd1
root-commit
Pipeline #1 Passed for main
test
Related jobs
test


/etc/gitlab-runner/config.toml
.gitlab-ci.yml
=>
push
RUNNER
1° docker
=>
...
[[runner.docker]]
executor = "docker"
image = "alpine:latest"
2° git fetch
=>
copy + .git
job:
script:
- ./something
3° script:
=>
$ /copy/something

choisir une image (agent docker)

- A l'instar d'une *image disque*, une **image docker** contient l'état d'un système de fichier *instantané / snapshot*
  i.e librairies et exécutables installés
- c'est un **environnement isolé** du reste de la machine
- les images viennent a priori d'un registre d'images public sur **https://hub.docker.com**
- *Deploy > Container Registry:* registre d'images privé
  - *+rapide & images custom.*



registre d'images privé

- Ajout d'une image au registre accessible à l'adresse **gitlab.myusine.fr:5050**

```
# télécharger depuis hub.docker.com
docker pull <image>:<tag>
# copier l'image en préfixant un nouveau nom avec l'URL du projet
docker tag <image>:<tag> gitlab.lan.fr:5050/formation/dev/<image>:<tag>
# s'authentifier sur le registre avec les identifiants
docker login gitlab.lan.fr:5050 -u root -p R00tt00R
# pousser l'image sur le registre
docker push gitlab.lan.fr:5050/formation/dev/<image>:<tag>
```

image – mise en oeuvre

- On utilise la clé `image: nom_image:nom_tag`
  - *nom:* techno / *tag:* version ou environnement
- périmètre (subsidiarité)
  - runner => `/etc/gitlab-runner/config.toml`
  - pipeline / job => dans le fichier **.gitlab-ci.yml**

```
image: pipeline_image_nom:pipeline_image_tag

job:
  image: job_image_nom:job_image_tag
```

référence et variables prédéfinies

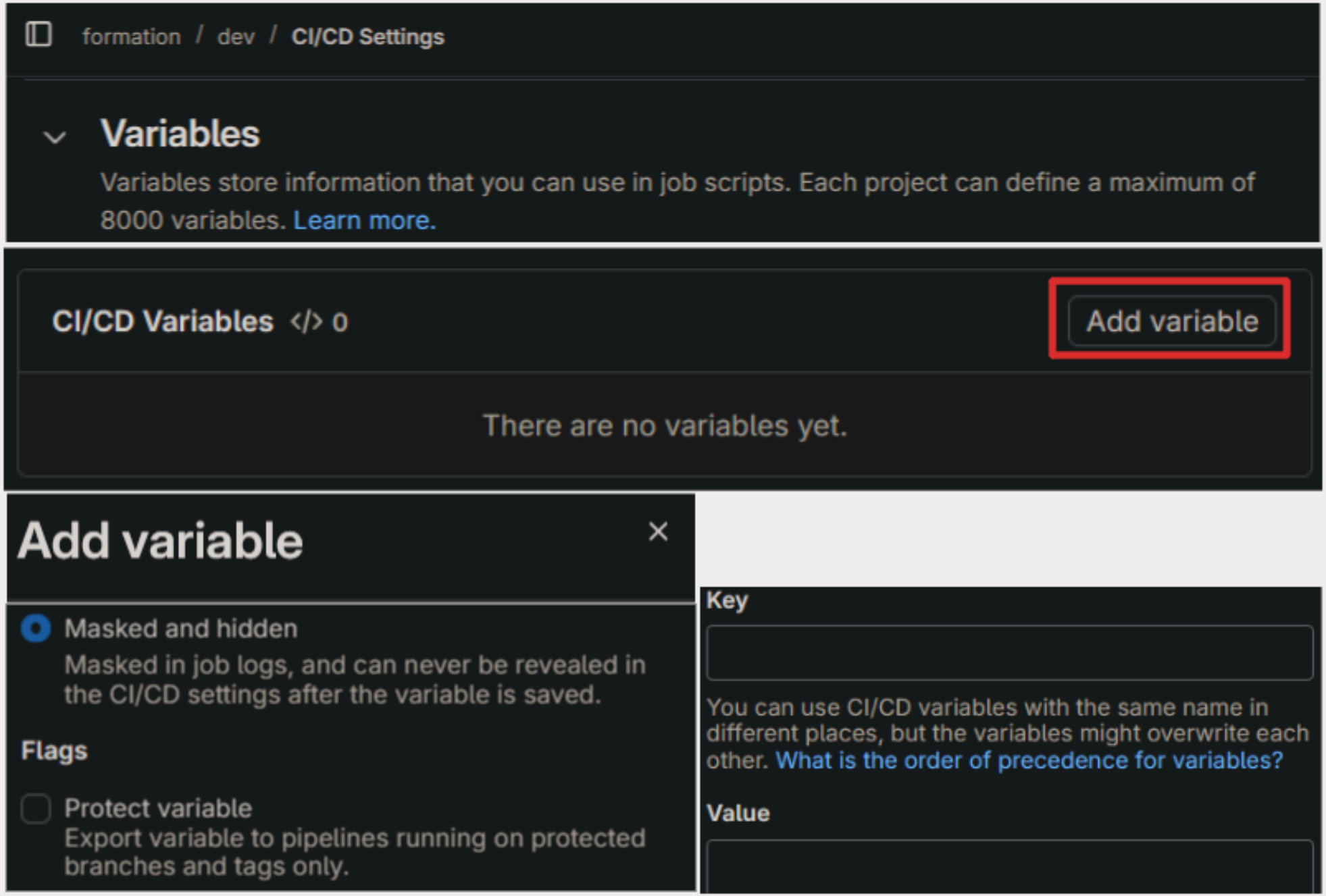
- clés utiles dans gitlab CI : **https://docs.gitlab.com/ee/ci/yaml/**
- variables d'environnement prédéfinies: **ici**
- *utiliser* la valeur des variables, en les préfixant par **« $ »**

```
## variables arbitraires créées par le développeur
variables:
  MY_VAR: "here !"

unit tests:
  ## variable prédéfinie créée par Gitlab CI
  image: $CI_REGISTRY_IMAGE/python:rc-slim-buster
  script:
    - echo "launch tests $MY_VAR"
```

variables cachées

- pour les informations sensibles:
  - **secrets:** mots de passe, certificats, clés privées
  - adresses IP, ports réseau, login
- *projet/groupe > Settings > CICD > Variables > Add Variable*
- caché ET masqué ET NON protégé - *branche de feature*

```
# PKEY, SSH_USER, SSH_HOST dans gitlab cI
script:
  - ssh -i $PKEY SSH_USER@SSH_HOST
```


formation / dev / CI/CD Settings
Variables
Variables store information that you can use in job scripts. Each project can define a maximum of 8000 variables. Learn more.
CI/CD Variables 0
Add variable
There are no variables yet.
Add variable
Masked and hidden
Masked in job logs, and can never be revealed in the CI/CD settings after the variable is saved.
Flags
Protect variable
Export variable to pipelines running on protected branches and tags only.
Key
You can use CI/CD variables with the same name in different places, but the variables might overwrite each other. What is the order of precedence for variables?
Value

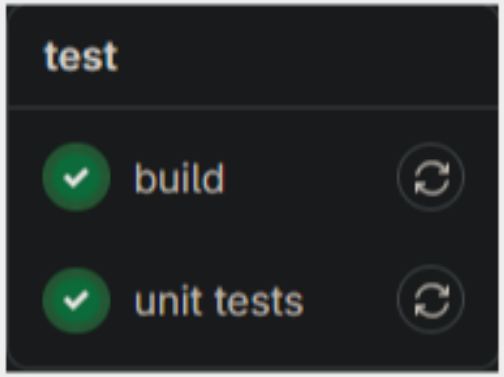
séquencement des jobs

- Par défaut, les jobs sont exécutés en parallèle
  - fonction du *nb cpus* + config dans */etc/gitlab-runner/config.toml*
- un ordre d'exécution des jobs par étapes est créé avec **stages[]:**
  - un élément de cette liste est un **« stage »**
  - les job sont associés aux stages par la clé **stage:**

> *deux jobs dans un même stage sont exécutés en parallèle*

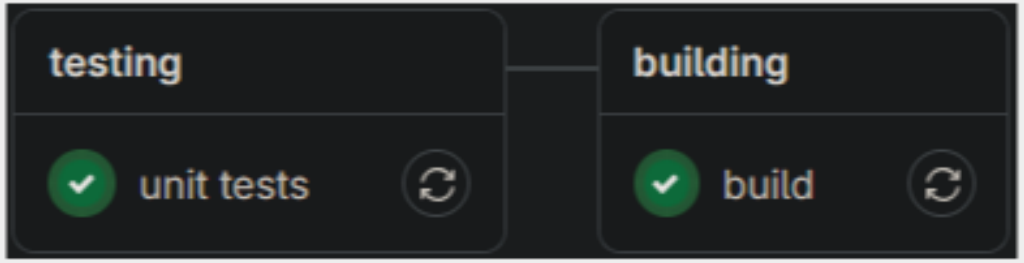
```yaml
stages:
  - testing
  - building

unit tests:
  stage: testing
  image: $CI_REGISTRY_IMAGE/python:rc-slim-buster
  script:
    - echo "launch tests here !"
```

parrallèle



⇒

séquentielle avec les stages



conditionnalité des jobs

- **rules[]:** liste des *règles* d'exécution d'un job au moyen de *clauses*
  - chaque règle vraie exécute le job - *OU logique*
  - une règle est vraie si toutes ces clauses sont vraies - *ET logique*
- `if: $VAR [== | != | =~ | !~] "value" [&& | ||]`
  - tester la valeur d'une variable - *exacte ou liée à une regex*
- **changes[]:** vraie si au - un des chemins listés a été modifié
  exists[]: IDEM pour l'existence
- **when:** contexte de déclenchement - *on_success -> par défaut, on_failure, always, never, manual*

```yaml
## déclenchement du pipeline
workflow:
  rules:
    - if: $CI_PIPELINE_SOURCE == "push"
    - if: $CI_PIPELINE_SOURCE == "api"

## déclenchement du job
unit tests:
  ...
  ## variables d'environnement arbitraires
  variables:
    MANUAL_TRIGGER: "off"
  rules:
    - if: $CI_COMMIT_BRANCH ~= /^feature-[0-9]+/
      changes:
        - README.md
    - if: $MANUAL_TRIGGER == "on"
```

prélever des artefacts

- **artifacts:** fait remonter toute ressource crée à l'exécution d'un job
  - dans les *jobs suivants, pour réutilisation*
  - dans *Gitlab UI pour téléchargement*
- **artefacts:paths[]:** chemins à faire remonter
- **artefacts:untracked:** fait remonter *tous les fichiers* **«U»** dans git
- **artefacts:expire_in:** temps de disponibilité avant suppression
  par défaut *"30 days"*

```
unit tests:
  script:
    - echo "content" > artifact.txt
  artifacts:
    paths:
      - ./artifact.txt
    expire_in: "1 hour"
```

> *les artefacts sont accessibles depuis la console ou avec le bouton ci-dessous dans les pipelines, le dépôt ...*

faire monter les rapports de jobs

- **artifacts:reports:** fait remonter des *rapports* dans Gitlab UI en *datavisualisation*
- rapports compatibles avec un format compris par Gitlab
  - ex: `junit:` pour les tests unitaires
- *Exception:* la couverture de code globale peut être prélevée depuis la sortie d'un job avec la clé
  `coverage: "/<regex>/"`

```
unit tests:
  script:
    - mvn test
  coverage: "/Total.*?([0-9]{1,3})%/"
  artifacts:
    expire_in: "1 hour"
    reports:
      junit: "target/surefire-reports/TEST-*.xml"
```

“ *les rapports sont visibles dans l'espace du pipeline, dans l'onglet «Tests»* ”

créer / utiliser le cache

- **cache:** fait remonter toute ressource crée à l'exécution d'un job
  - utilisé pour *mettre en cache les dépendances du projet*
- **cache: key:** nomme les objets mis en cache
- **cache:paths[]:** désigne les chemins à mettre en cache
- **cache:untracked:** ajoute au cache les fichiers à l'état **«U»** dans git
- **cache:policy:** indique si l'on veut
  - *pull* => installer le *cache existant* dans le job
  - *push* => faire remonter le *cache généré* dans le job
  - *pull-push* => le 2

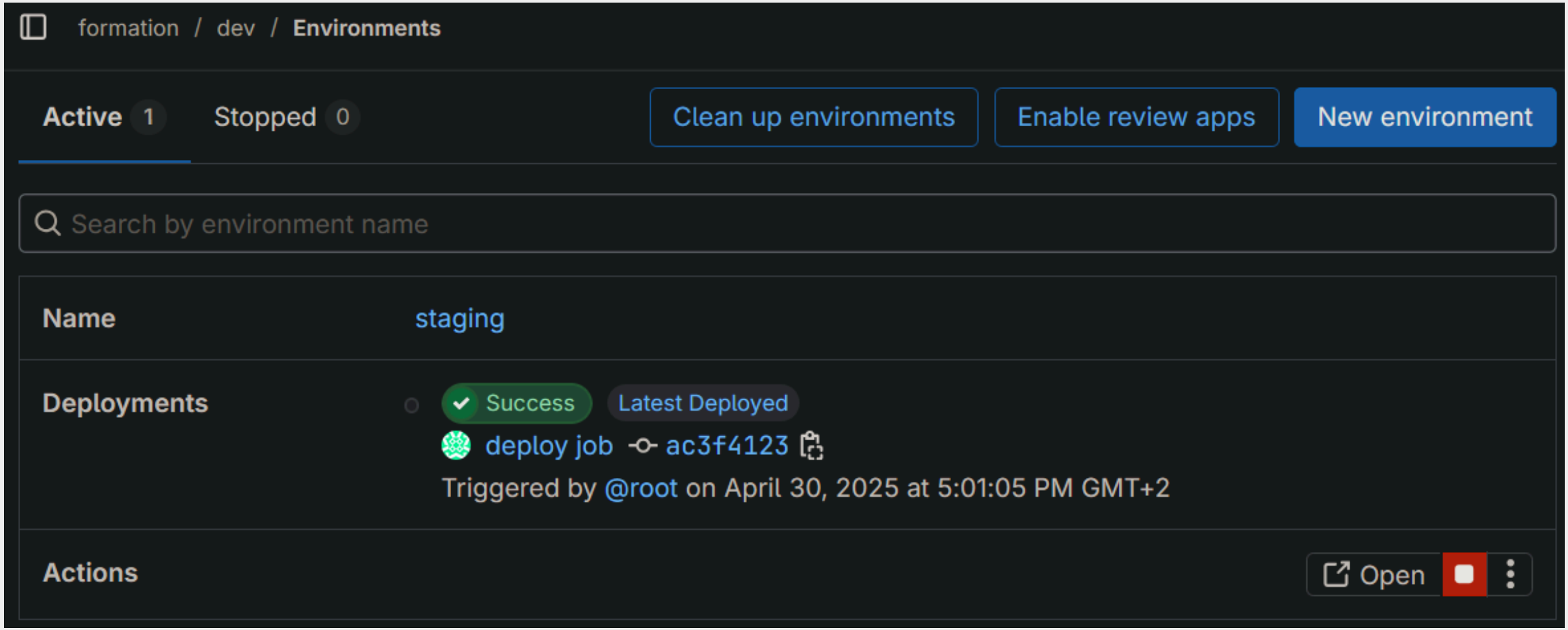
```
unit tests:
  script:
    - mvn compile
  cache:
    key: cache
    untracked: true
    paths:
      - ./.m2/repository
```

“ `cache:untracked:true` *est obligatoire en pull car le runner exécute* `git clean` *après l'installation du cache !!!* ”

déploiement: usage des environnements

- **environment:** créer un espace dans gitlab pour gérer et tester un déploiement
- *Project > Operate > Environment*

```
deploy:
  ...
  environment:
    name: staging
    url: "https://dawan.fr"
  ...
  # on déploiement soit sur main ou MIEUX sur un TAG GIT
  rules:
    - if: $CI_COMMIT_BRANCH == "main"
    - if: $CI_COMMIT_TAG =~ /v[0-9]+\.[0-9]+/
```


formation / dev / Environments
Active 1
Stopped 0
Clean up environments
Enable review apps
New environment
Search by environment name
Name
staging
Deployments
Success
Latest Deployed
deploy job
ac3f4123
Triggered by @root on April 30, 2025 at 5:01:05 PM GMT+2
Actions
Open

utiliser les services

- **services[]:** configure un *agent auxiliaire* adossé à un job
  utile quand le job nécessite une *techno client / serveur*
- **services[0].name:** nom de l'*image*
- **services[0].alias:** alias/nom d'hôte réseau pour la *cnx client*

```
job:
  image: client:latest
  script:
    - ./client.sh --host=server-domain-host
  services:
    - name: server:latest
      alias: server-domain-host
```

IV. SCRUM dans Gitlab

Scrum = 3 Rôles + 3 Artefacts + 5 Evénements

Equipe de développement

Product Owner

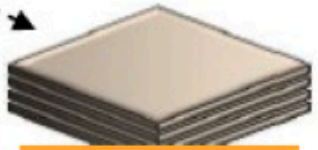
Scrum Master

Réunion de planification (Facultatif)

Sprint planning (4-8 heures)

Daily Scrum (15 minutes)

Product Backlog



Sprint Backlog

24 heures

2 - 4 semaines

Sprint review (2-4 heures)

Sprint retrospective (1,5-3 heures)

SPRINT

Aucun changement de périmètre !

Incrément

Scrum est basé sur des itérations de blocs de temps.

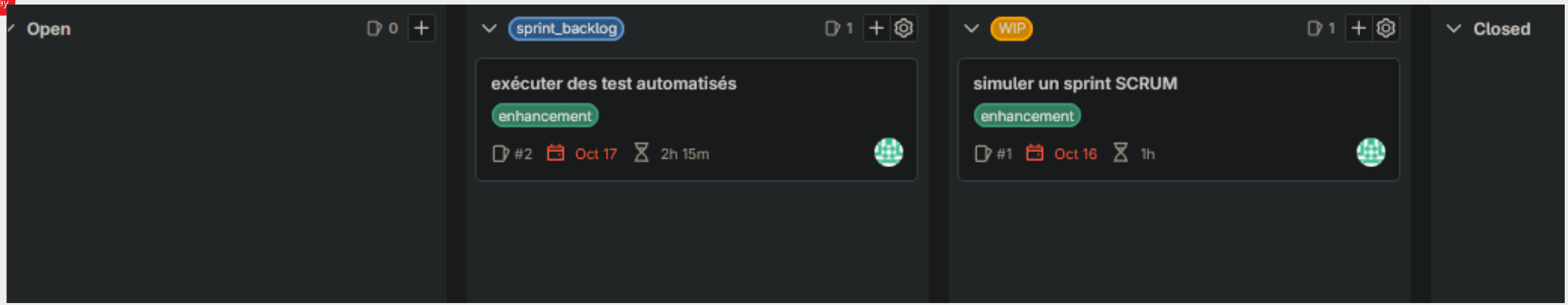
Où interviennent les opérations IT ?

rôles dans Scrum

- **Product Owner:** possède la vision du produit, exprime le besoin
  - observe régulièrement l'avancée du projet
  - au rythme de la diffusion de livrables
  - il est soit client, soit en interface avec le client
- **Agile Process Owner:** utilise les principes Agile et Scrum
  - conçoit, gère et mesurer les processus individuels
- **Scrum Master (Dev) et Agile Service Manager (Ops)**
  - garants du cadre méthodologique de l' équipe.
  - veillent à l'ordre du jour et rythment les communications

plannification

1. Créer des labels
   Projet/Groupe > Manage > Labels
2. catégoriser Boards, issues
   Projet/Groupe > Plan > Boards | Issues
3. Créer au - 2 labels pour les colones *sprint backlog* et *WIP*
4. Créer des labels classiques *Bug, Enhancement, Hot Fix, ...*


Open
0
sprint_backlog
1
exécuter des test automatisés
enhancement
#2
Oct 17
2h 15m
WIP
1
simuler un sprint SCRUM
enhancement
#1
Oct 16
1h
Closed

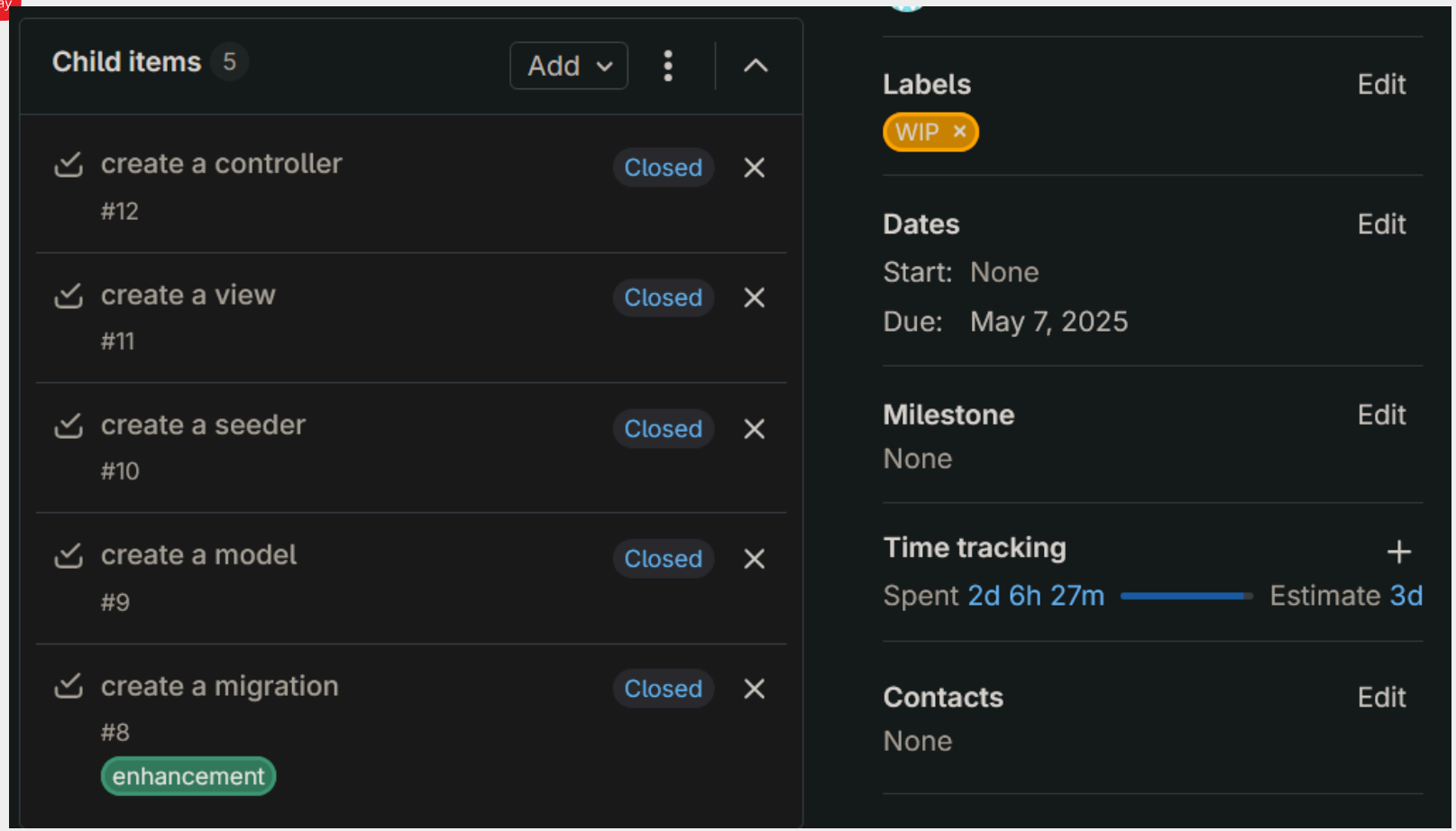
créer des issues sous forme de « users stories »

- **« user story »:** description rédigée du point de vue de l'utilisateur
- **« epic »:** grande user story à décomposer
- critères d'une user story à respacter **INVEST**
  - **I**ndépendante: sur le sprint en cours
  - **N**égociable: fait l'objet d'une concertation
  - **V**aleur: doit apporter de la valeur ajoutée aux clients
  - **E**stimable: en temps *«due date»* ou complexité *«points de sprint»*
  - **S**uffisamment petite: livraison au sein d'un seul Sprint.
  - **T**estable: permet d'en **déduire un test - TDD ou BDD**

templates de user story « US »

- templates de description dans `.gitlab/issue_templates/xxxx.md`
  OU *Projet/groupe > Settings > General > default.md*

```
### Given: Context ...
<!-- WHEN? WHERE? WHO? -->

### When: Action To Do ...
* [ ] xxxxxx

### Then: Expected Results
<!-- WHAT? Metrics ? -->
1. yyyy
<!-- Quick Actions -->
/label ~enhancement    # catégorisation depuis la desc.
/estimate ???          # estimation en temps Ex. 1h 30m
```



“ *les «To-Dos» peuvent être convertis en sous-tâches pour plus de détails* ”

V. Workflow Gitlab

Animer SCRUM & GITLAB & GIT

1. **Design** - créer une **issue** > *liste "Open" de Boards*
    - *éditer l'issue* pour décrire l'**US** avec le template
2. **Sprint planning** - assigner + estimer l'US > *liste "Sprint backlog"*
3. **Go** - créer une **Merge Request (MR)** > *liste "WIP"*
    - bouton *«Create Merge Request»* dans l'édition de l'issue
    - configurer la MR et *«Create»* > donne une **branche de feature**
    - rapatrier la branche de feature pour **travailler en local**

```
git fetch --all
git checkout --track origin/feature-branch-name
```

Merge Request MR

- espcace de **travail collectif** dans gitlab
  - lié à une **issue** et/ou une **branche de fonctionnalité**
  - rassemble les *informations de travail* sur la fonctionnalité *modifs, commits, commentaires, pipelines sur la branche*
- prépare la **fusion** de la branche de fonctionnalité dans la branche commune
  - commence dans l'état *« Draft »*: non fusionnable
  - on assigne un responsable de la *MR*
  - on assigne un responsable de la **revue de code**

From `3-tester-les-merge-requests` into `main` Change branches

Title (required)

Draft: Resolve "tester les merge requests"

Mark as draft
Drafts cannot be merged until marked ready.

Description

Preview

Closes #3

Switch to rich text editing

Add description templates to help your contributors to communicate effectively!

Assignee

Dave A

Reviewer

Administrator

Draft: Resolve "tester les merge requests"

Open **Dave A** requested to merge `3-tester-les-merge-requests` into `main` just now

Overview 0 | Commits - | Pipelines 0 | Changes 0

Closes #3

👍 0 👎 0 ☺

Merge request contains no changes

Use merge requests to propose changes to your project and discuss them with your team. To make
Code dropdown list above, then test them with CI/CD before merging.

Activity

- **Dave A** added WIP enhancement labels just now
- **Dave A** requested review from @root just now
- **Dave A** assigned to @dev_a just now

revue de code par les pairs

- observations, débat sur le code d'une fonctionnalité **ante fusion**

1. **onglet "changes":** étude des diffs des divers commits de la branche
2. télécharger la branche à fusionner en local
   et considérer le **Δ avec une autre branche** de fonctionnalité

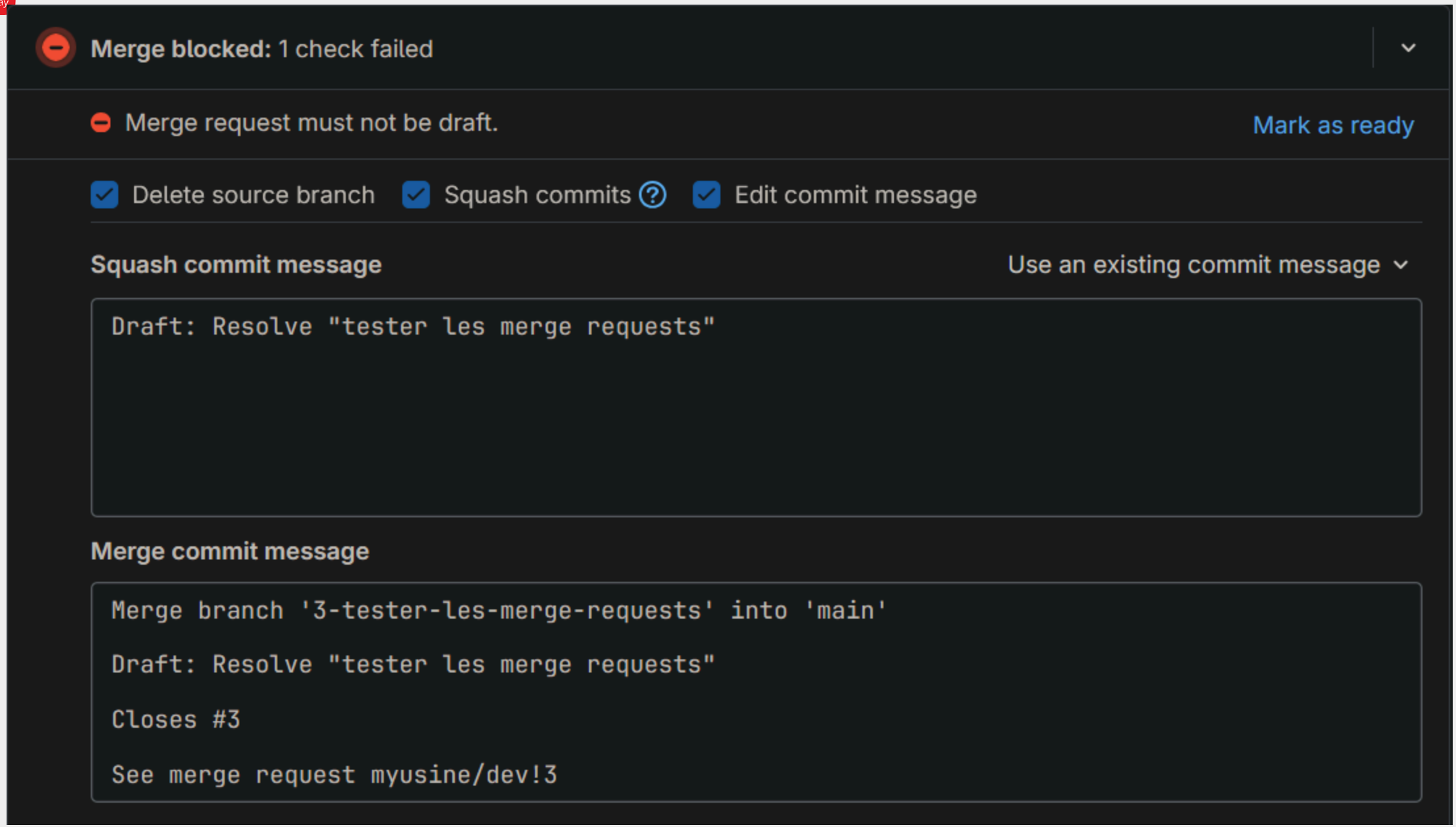
```
git fetch --all && git checkout <distant-hash> ## Mode détaché !!
git checkout -b <tmp-branch> && git merge <local-branch>
## OU appliquer un patch
git checkout <local-hash>
curl -k "https://../merge_requests/xx.patch" | git am
```

3. utiliser un job **«Review App»**, i.e *déploiement dynamique* de test

finaliser une MR

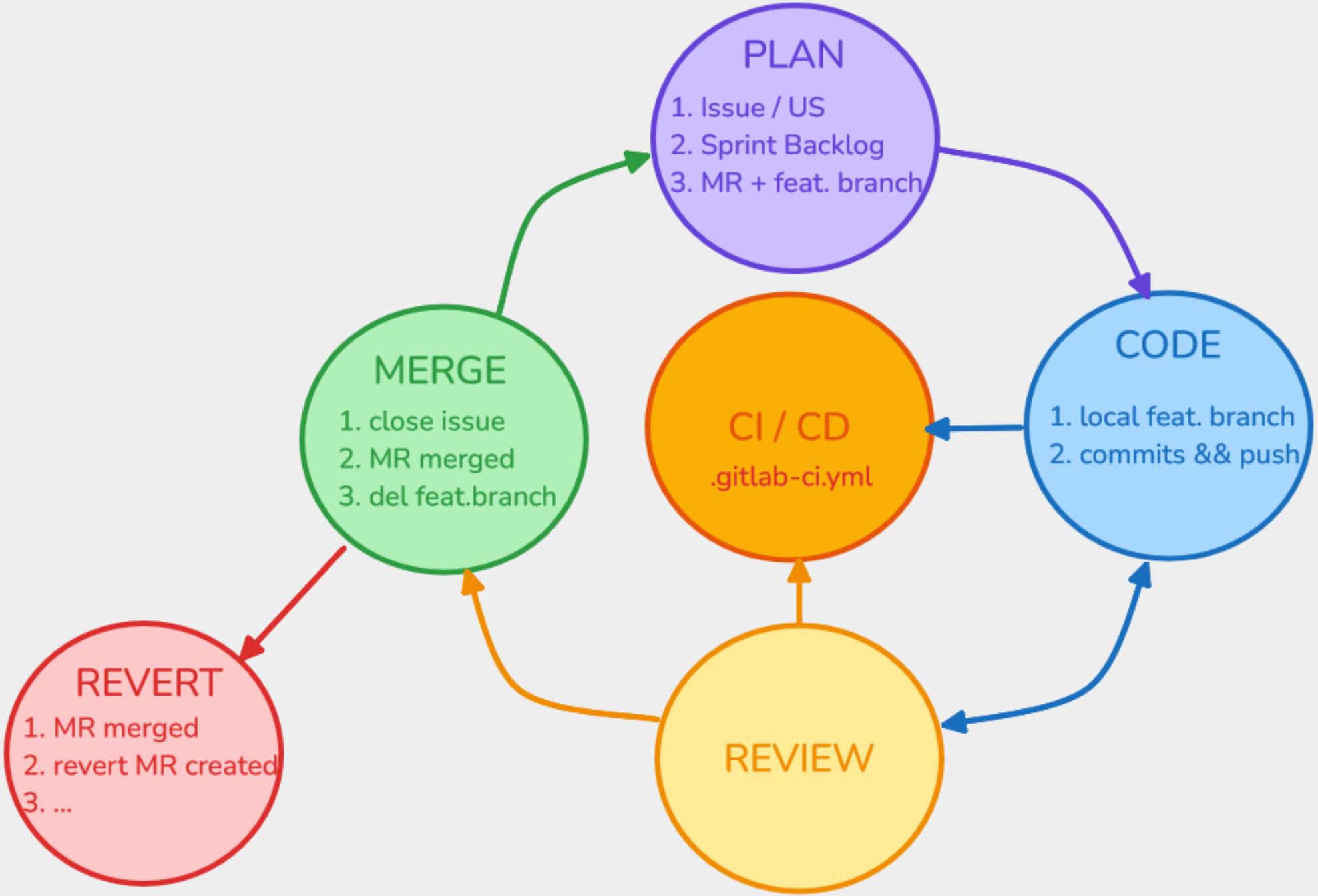
- si la revue de code est probante on configure la fusion

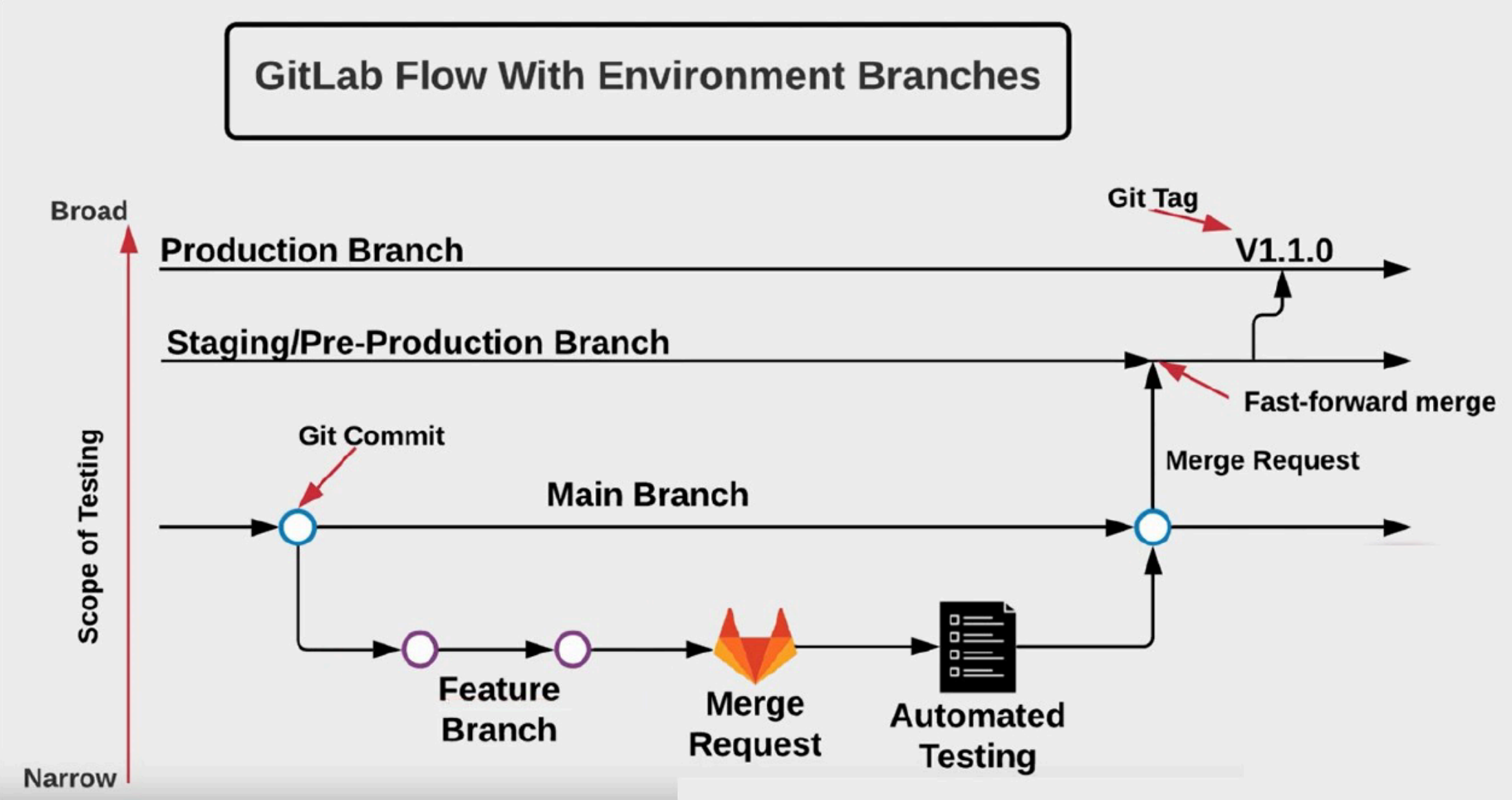
1. bouton *« Mark as Ready »* si la MR est à l'état *« Draft »*
2. le pipeline devrait ou doit **être en succès** si
   Settings > Merge Requests > Merge checks > case cochée
3. **suppression auto** de la branche de fonctionnalité
4. **squash** tous les commits de la branche *en un seul commit condensé*
   + un message de commit condensé
5. *Si Erreur* - inverser les modifs du commit de fusion *--no-ff*
   Revert: `git revert -m 1 <merge-commit>`


Merge blocked: 1 check failed
Merge request must not be draft.
Mark as ready
Delete source branch
Squash commits
Edit commit message
Squash commit message
Use an existing commit message
Draft: Resolve "tester les merge requests"
Merge commit message
Merge branch '3-tester-les-merge-requests' into 'main'
Draft: Resolve "tester les merge requests"
Closes #3
See merge request myusine/dev!3

nettoyer le travail en local

```
## télécharger la fusion faite sur gitlab
git checkout main && git pull

## supprimer les branches de fonctionnalité et de suivi inutiles
git branch -d <local-feature-branch>
git fetch --prune
```


PLAN
1. Issue / US
2. Sprint Backlog
3. MR + feat. branch
CODE
1. local feat. branch
2. commits && push
CI / CD
.gitlab-ci.yml
MERGE
1. close issue
2. MR merged
3. del feat.branch
REVERT
1. MR merged
2. revert MR created
3. ...
REVIEW


GitLab Flow With Environment Branches
Broad
Scope of Testing
Narrow
Git Tag
V1.1.0
Production Branch
Staging/Pre-Production Branch
Fast-forward merge
Merge Request
Git Commit
Main Branch
Feature Branch
Merge Request
Automated Testing