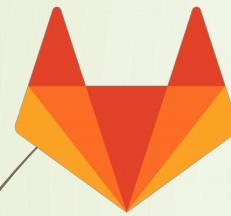


DAWAN Paris  
DAWAN Nantes  
DAWAN Lyon

11,rue Antoine Bourdelle, 75015 PARIS  
32, Bd Vincent Gâche, 5e étage - 44200 NANTES  
Bt Banque Rhône Alpes, 2ème étage - 235 cours Lafayette 69006 LYON



# Gitlab CI/CD

## installation, et configuration

Plus d'info sur <http://www.dawan.fr> ou **0810.001.917**

**Formateur: Matthieu LAMAMRA**



# **INSTALLATION MANUELLE**



# Rappels liminaires

sudo : commande d'exécution de commande en tant qu'autre utilisateur

\$ su - => connexion au compte root

# => invite de commande root

# apt-get (ou apt) update => mise à jour du cache apt-get

#apt-get (ou apt) install -y <paquet> => installation du paquet <paquet>

Ajout de l'utilisateur xxxx au sudoers

# vi /etc/sudoers.d/xxxx => création du fichier sudoers dédié à xxxx

xxxx ALL = (ALL) ALL => xxxx à tous les droits de tous les utilisateurs

<=> root

# Dépôt git local

Bash :      mkdir ~/projet/

                cd ~/projet/

Git :        git init    *<= création dépôt (dossier caché .git)*

[ ajouts / modification de fichiers]

                git add .    *<= sélection du code à enregistrer*

                git commit -am "content commit"    *<= enregistrement*

Commit = enregistrement dans le dépôt avec allocation d'un id

# Installation gitlab

# Prérequis

```
sudo apt update
```

```
sudo apt install -y curl openssh-server ca-certificates tzdata
```

# Téléchargement et installation du paquet depuis le dépôt gitlab

```
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ee/script.deb.sh | sudo bash
```

# Installation du paquet avec variable d'environnement spécifique

```
sudo EXTERNAL_URL="http://gitlab.myusine.fr" apt-get install gitlab-ee
```

# Créer un dépôt gitlab (1/2)

DNS: sudo nano /etc/hosts (dns local)

Ajouter la ligne : 127.0.0.1 gitlab.myusine.fr

Gitlab : Accéder à http://gitlab.myusine.fr

Choisir et confirmer un mdp root

Se logger en root / mdp

( troubleshooting mdp :

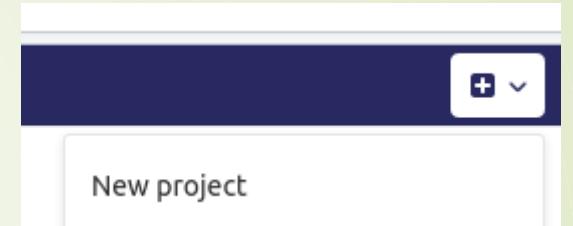
[https://docs.gitlab.com/12.10/ee/security/reset\\_root\\_password.html](https://docs.gitlab.com/12.10/ee/security/reset_root_password.html) )

Cliquer sur « new project » → « Blank project »

Project-name : myusine

Visibility-level : private

Cliquer sur « create project »



# Créer un dépôt gitlab (2/2)

Mise en place d'une communication ssh par clé publique entre le dépôt local et gitlab

bash :    \$ ssh-keygen -t rsa

Chemin : /home/<user>/.ssh/myusine <= + ~/.ssh/myusine.pub (clé publique)

Passphrase : [aucune]

\$ cat ~/.ssh/myusine.pub <= afficher le contenu de la clé publique

Gitlab :    Accéder à http://gitlab.myusine.fr/profile/keys

Copier le contenu de ~/.ssh/myusine.pub dans le textarea

Nommer la clé

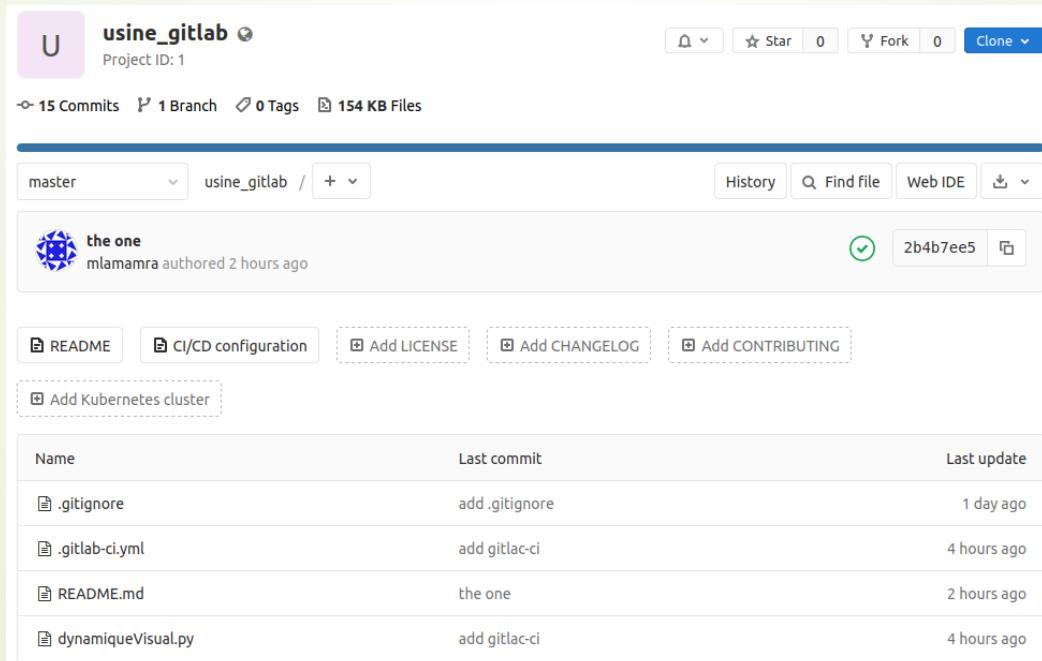
# Synchroniser les dépôts

Connexion du dépôt local au dépôt gitlab via ssh

git :    git remote add gitlab git@gitlab.myusine.fr:root/myusine.git

git remote -v <= affiche les dépôts distants (vérification)

git push gitlab master <= transfert le code sur le dépôt distant

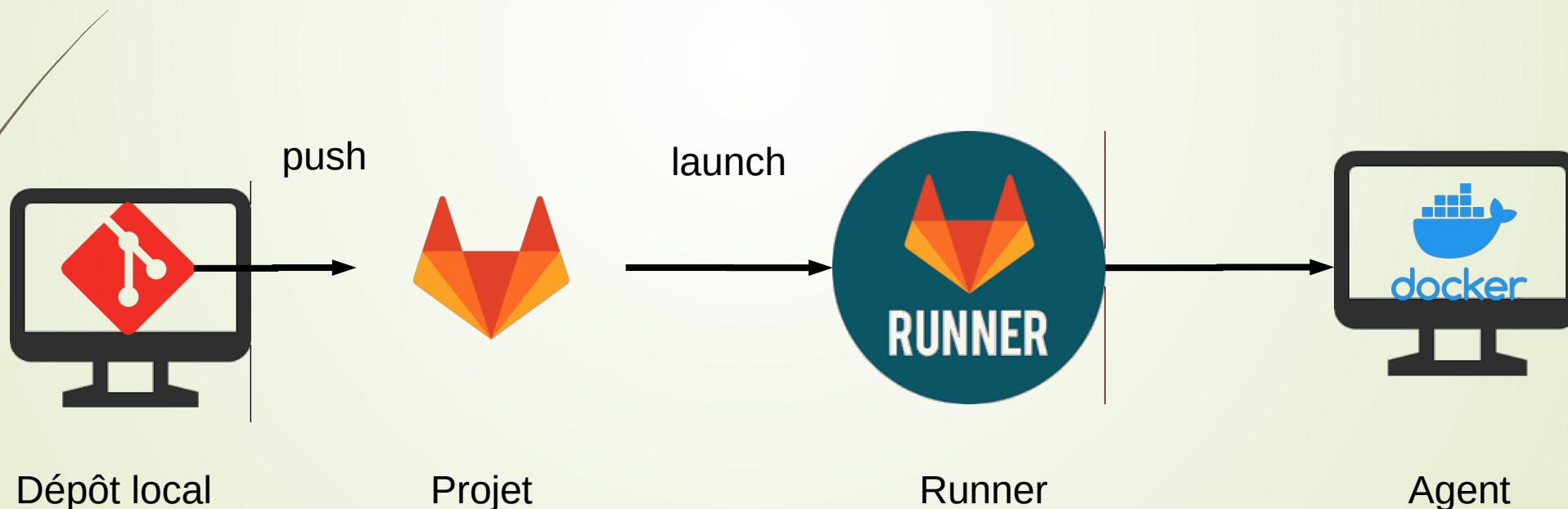


# Installation docker

```
# OS : Ubuntu 20.04
sudo apt update
# si docker déjà installé, pour mise à jour
sudo apt remove docker docker-engine docker.io containerd runc
# prérequis
sudo apt install -y apt-transport-https \
ca-certificates curl gnupg-agent software-properties-common
# ajout de la clé GPG pour authentifier le dépôt
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
# ajout du dépôt docker à apt
sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
# mise à jour du cache apt pour tenir compte du nouveau dépôt
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io
```

# gitlab-runner

- **gitlab-runner** est le composant Gitlab chargé de l'intégration continue
- gitlab-runner **associe des runners aux projets** gitlab
- Un runner fournit au projet des « **agents** » : machines physiques, vms, conteneurs
- Ces agents vont copier le dépôt gitlab et exécuter des tâches d 'intégration continues
- **Gitlab + Gitlab-runner = Gitlab CI**



# Installation gitlab-runner

```
# téléchargement du script d'installation
```

```
curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash
```

```
# installation avec variable d'environnement
```

```
export GITLAB_RUNNER_DISABLE_SKEL=true; sudo -E apt-get install gitlab-runner
```

# lier le runner à gitlab (1/3)

Gitlab : Accéder à [http://gitlab.myusine.fr/root/myusine/-/settings/ci\\_cd](http://gitlab.myusine.fr/root/myusine/-/settings/ci_cd)  
Section « runners », Cliquer sur « expand »

The screenshot shows a step-by-step guide for setting up a runner. It includes a list of four steps, each with a corresponding URL or token in a red box, and a 'Reset runners registration token' button.

**Set up a specific Runner manually**

1. [Install GitLab Runner](#)
2. Specify the following URL during the Runner setup:  
<http://gitlab.myusine.fr/>
3. Use the following registration token during setup:  
[A38Hes8TWySL533Mfa-g](#)
4. Start the Runner!

[Reset runners registration token](#)

# lier le runner à gitlab (2/3)

- **gitlab-runner register** permet d'enregistrer un runner auprès d'une instance de gitlab
- Cet utilitaire s'exécute sous forme de « wizzard » interactif ou via des options
- Ici, le runner lancera des agents sous forme de conteneurs dockers
- Par défaut, ces conteneurs contiendront un système alpine, i.e une debian légère

```
# création d'un runner pour le projet
sudo gitlab-runner register \
--non-interactive \
--url http://gitlab.myusine.fr/ \
--registration-token <token> \
--executor docker \
--docker-image alpine:latest
```

# lier le runner à gitlab (3/3)

- Le runner doit connaître l'ip de gitlab.myusine.fr pour pouvoir installer le dépôt du projet sur les conteneurs

<https://docs.gitlab.com/runner/configuration/advanced-configuration.html>

```
sudo nano /etc/gitlab-runner/config.toml
```

```
...
```

```
[runners.docker]
```

```
...
```

```
extra_hosts = ["gitlab.myusine.fr:172.17.0.1"]
```

# Tester le runner (1/2)

Bash : nano <dépôt local>/.gitlab-ci.yml

```
test:  
  script:  
    - echo "Tests en cours"  
  
build:  
  script:  
    - echo "Build en cours"  
  
deploy:  
  script:  
    - echo "Déploiement de l'application"
```

# Tester le runner (2/2)

Git: \$ git add .

\$ git commit -am "ajout gitlab-ci.yml"

\$ git push gitlab master

Gitlab : Accéder à <http://gitlab.myusine.fr/root/myusine/pipelines>

Status	Pipeline	Triggerer	Commit	Stages
<span>passed</span>	#10 latest		master -> 2b4b7ee5 the one	

# Le format YAML (1/5)

format de représentation de données par sérialisation, conçu pour être aisément modifiable et lisible

- Scalaires sur une ligne

```
clé: valeur
autre_clé: une autre valeur
valeur_numérique: 100
notation_scientifique: 1e+12
booléen: true
valeur_null: null
clé avec espaces: valeur
# Bien qu'il ne soit pas nécessaire de mettre les chaînes de caractères
# entre guillemets, cela reste possible, et parfois utile.
toutefois: "Une chaîne, peut être contenue entre guillemets."
"Une clé entre guillemets.": "Utile si l'on veut utiliser ':' dans la clé."
```

# Le format YAML (2/5)

- Scalaires en blocs

```
bloc_littéral: |
```

Tout ce bloc de texte sera la valeur de la clé "bloc\_littéral", avec préservation des retours à la ligne.

Le littéral continue jusqu'à ce que l'indentation soit annulée.

Toutes lignes qui seraient "davantage indentées" conservent leur indentation, constituée de 4 espaces.

```
bloc_replié: >
```

Tout ce bloc de texte sera la valeur de la clé "bloc\_replié", mais cette fois-ci, toutes les nouvelles lignes deviendront un simple espace.

Les lignes vides, comme ci-dessus, seront converties en caractère de nouvelle ligne.

# Le format YAML (3/5)

- Collections

```
# L'imbrication est créée par indentation.  
une_map_imbriquée:  
    clé: valeur  
    autre_clé: autre valeur  
    autre_map_imbriquée:  
        bonjour: bonjour
```

```
# Les séquences (équivalent des listes)  
une_séquence:  
    - Objet 1  
    - Objet 2  
    - 0.5 # les séquences peuvent contenir  
    - Objet 4  
    - clé: valeur  
    autre_clé: autre_valeur  
    -  
        - Ceci est une séquence  
        - dans une autre séquence
```

**<=> Dictionnaire en python**

**Attention : indenter 2 ou 4 espaces != \t**

**<=> Liste en python**

# Le format YAML (4/5)

- Alias et ancrés

```
generic job: &generic
  name: build
  script:
    - echo 'something'
jobs:
  build: *generic
  test:
    <<: *generic
    name: test
```

Ici, on définit une structure qu'on peut accrocher à une autre clé du document via son nom préfixé par « & »

l'alias « \*generic » réplique exactement le contenu de « generic job » dans la clé « build »

l'ancre « << : \*generic » comme clé de « test » réplique le contenu de « generic job », en autorisant des ajouts et des redéfinitions de clés

# Le format YAML (5/5)

- Compatibilité avec le format JSON

```
# YAML étant un proche parent de JSON
# des maps et séquences façon JSON
json_map: {"clé": "valeur"}
json_seq: [1, 2, 3, "soleil"]
```

Et plus encore ...

<https://learnxinyminutes.com/docs/fr-fr/yaml-fr/>

# Le linter Gitlab

- Vérifier la syntaxe yaml

The screenshot shows the GitLab Pipelines interface. At the top, there are three buttons: "Run Pipeline" (green), "Clear Runner Caches" (white), and "CI Lint" (white). The "CI Lint" button is highlighted with a red box. Below the buttons, there are filter options: "All 12" (highlighted with a grey box), "Finished", "Branches", and "Tags". A "Filter pipelines" input field is present. The main table has columns: Status, Pipeline, Triggerer, Commit, and Stages. The first pipeline row shows: "passed" (green button), "#138 latest" (button), Jenkins icon, "master → e782bd9d" (text), "test12" (text), two green checkmarks in a circle, "00:00:15" (text), and "30 minutes ago" (text).

Status	Pipeline	Triggerer	Commit	Stages	
<span>passed</span>	#138 latest	Jenkins	master → e782bd9d	test12	00:00:15 30 minutes ago

# CI.yml : dépendances (1/2)

- Compatibilité « image runner » / « dépendances projet »
  - Par défaut, le runner est enregistré comme utilisant une image alpine (debian)
  - PB : certaines dépendances du projet ne sont pas gérées par la distribution
  - Solution : gitlab-ci.yml propose la clé « **image** » pour modifier l'image exécutée

```
image: ubuntu:bionic
```

```
...
```

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

# CI.yml : dépendances (2/2)

- Un premier job : units
  - Un **job** est une unité d'exécution. Ici units pour « tests unitaires » (**arbitraire**)
  - La clé **script** peut contenir une liste de lignes de commandes shell linux
  - La clé **before\_script** permet d'exécuter des commandes avant **script => installations**
  - Gitlab-ci.yml met à disposition une série de **variables d'environnements** prédéfinies

```
units:  
  before_script:  
    - apt update  
    - apt install -y python3  
    - apt install -y python3-pip  
    - pip3 install -r $CI_PROJECT_DIR/requirements.txt  
  script:  
    - echo "deps installed!"
```

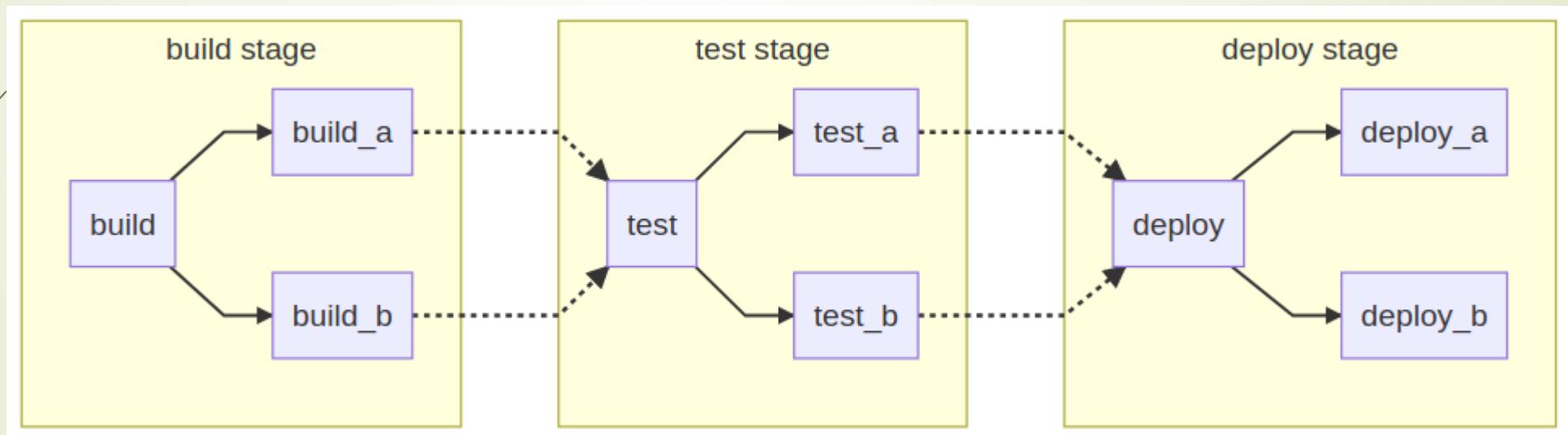
# CI.yml : synchronicité

- Ajout des « stages » : exécutions séquentielles
  - La clé **stages** définit une liste valeurs qui garantissent l'ordre d'exécution des jobs
  - La clé **stage** d'un job indique donc sa position dans la liste
  - Attention : deux jobs possédant la même clé **stage** seront exécutés en parallèle

```
stages :  
  - tests  
  - builds  
  
units:  
  stage: tests  
  ...
```

# CI.yml : synchronicité

- Pipeline classique
  - Les jobs s'exécutent successivement dans l'ordre des stages



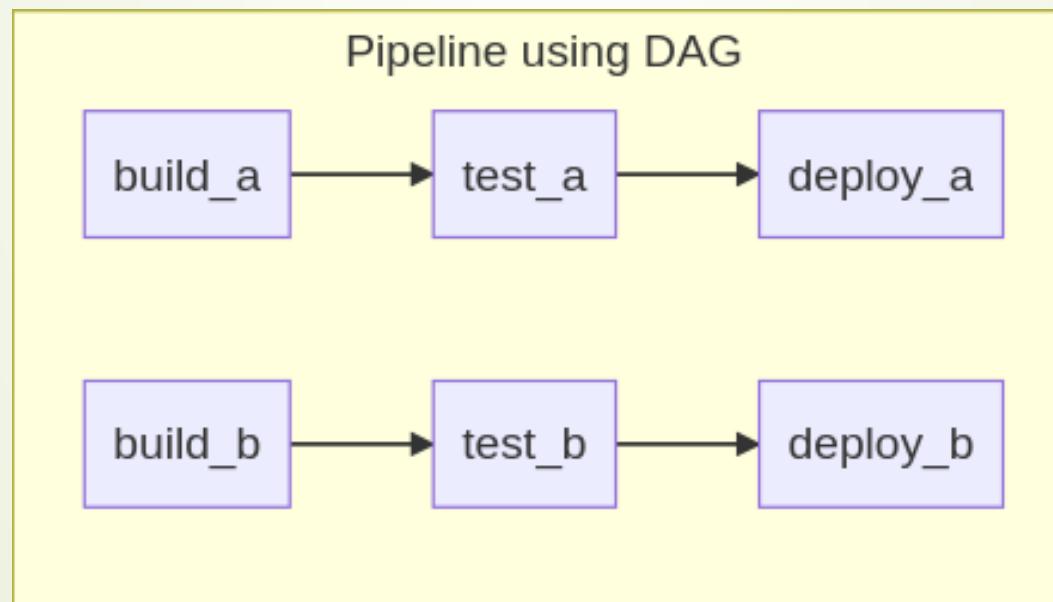
# CI.yml : synchronicité

- Dépendances entre jobs
  - La clé **needs** définit une relation de dépendance entre un job et une liste de jobs
  - La clé **needs** autorise un job à démarrer dès que les dépendances ont fini leur exécution, indépendamment du **stage**
  - Cette fonctionnalité permet d'appliquer un pipeline sur différentes parties du code de façon concurrente

```
test_a:  
  stage: test  
  needs: [build_a]  
  script:  
    - echo "This test job will start as soon as build_a finishes."  
    - echo "It will not wait for build_b, or other jobs in the build stage, to finish."
```

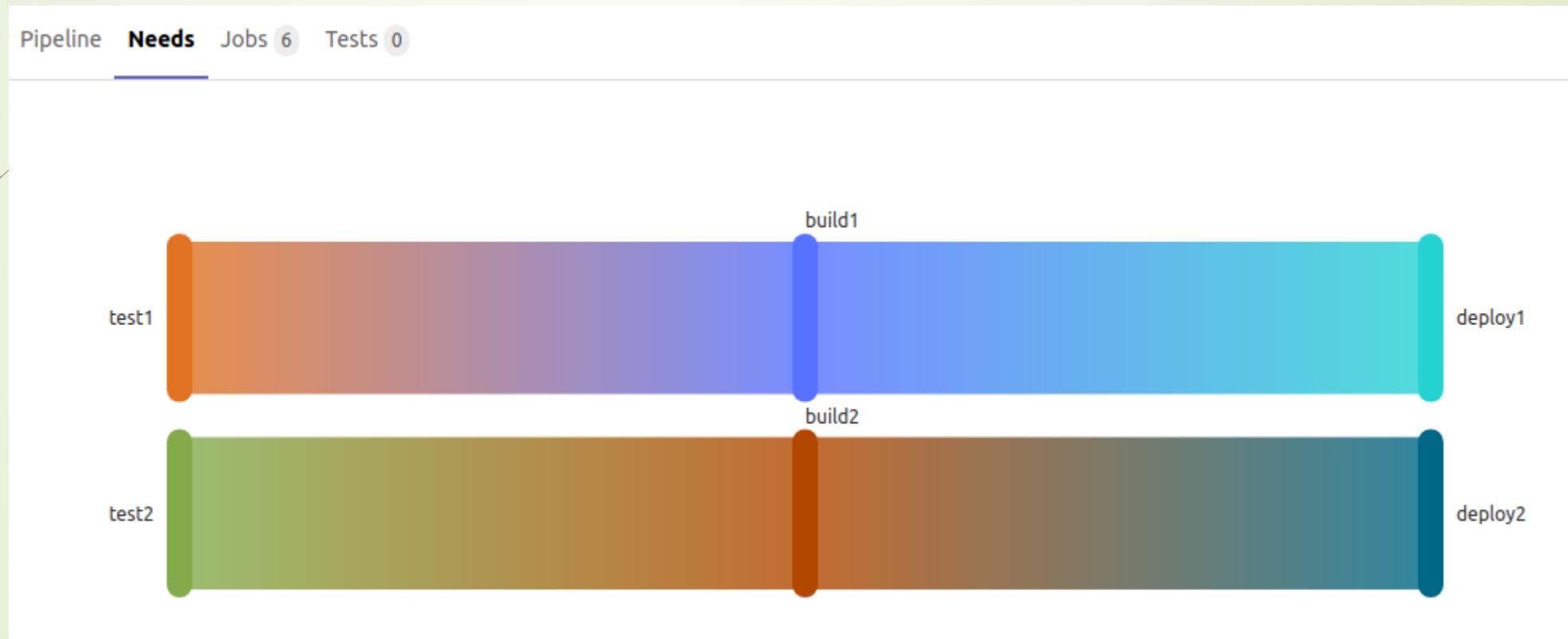
# CI.yml : synchronicité

- Pipeline DAG
  - « Directed Acyclic Graph », ce pipeline exécute les jobs dans le but de réduire l'exécution au plus vite



# CI.yml : synchronicité

- Pipeline DAG dans gitlab CI



# CI.yml : synchronicité

- Découpage du projet en sous-pipelines

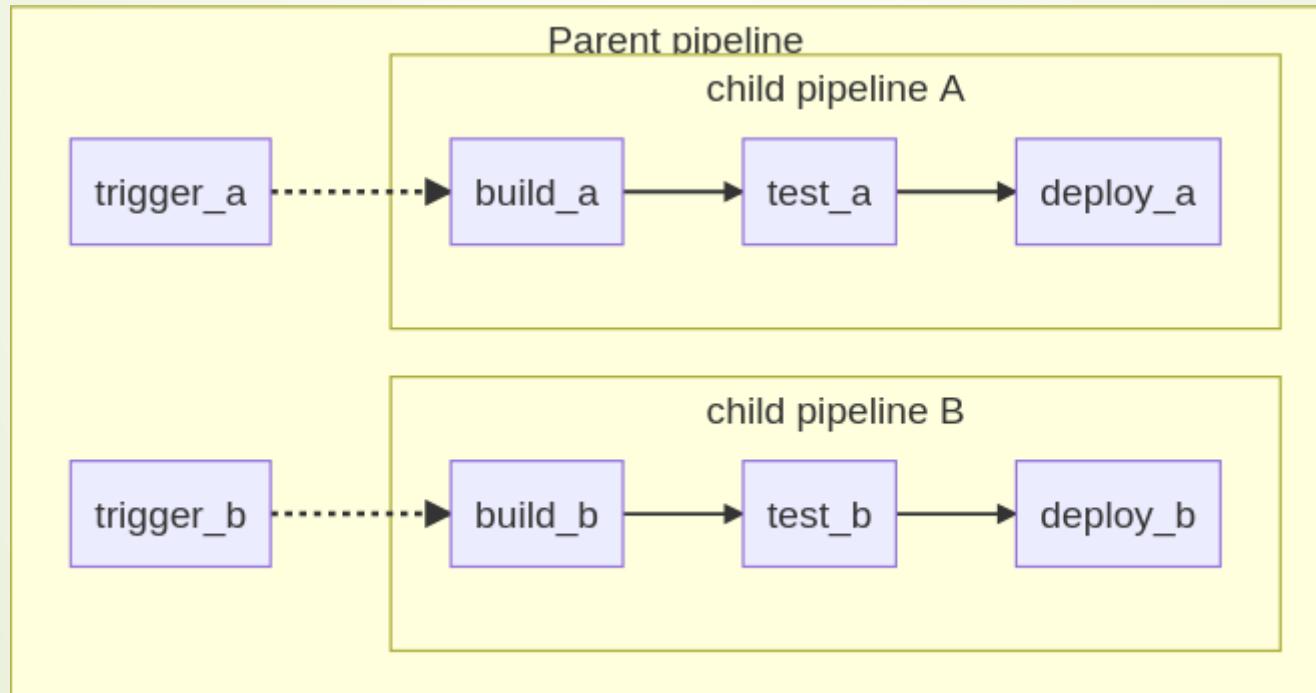
- La clé **trigger** permet d'appeler l'exécution d'un pipeline depuis un autre
- La clé **include** permet d'inclure l'exécution d'un fichier « gitlab-ci » au format yaml
- On peut par ce mécanisme exécuter de façon concurrente ou non des pipelines au niveaux des modules constitutifs de la découpe du projet, voire en cross-projet
- Il faut cependant veiller à éviter les confusions sur les noms des jobs et des stages dans le fichier inclus « espace de noms »

```
trigger_a:  
  stage: triggers  
  trigger:  
    include: module_a/.gitlab-ci.yml
```

```
trigger_b :  
  ...
```

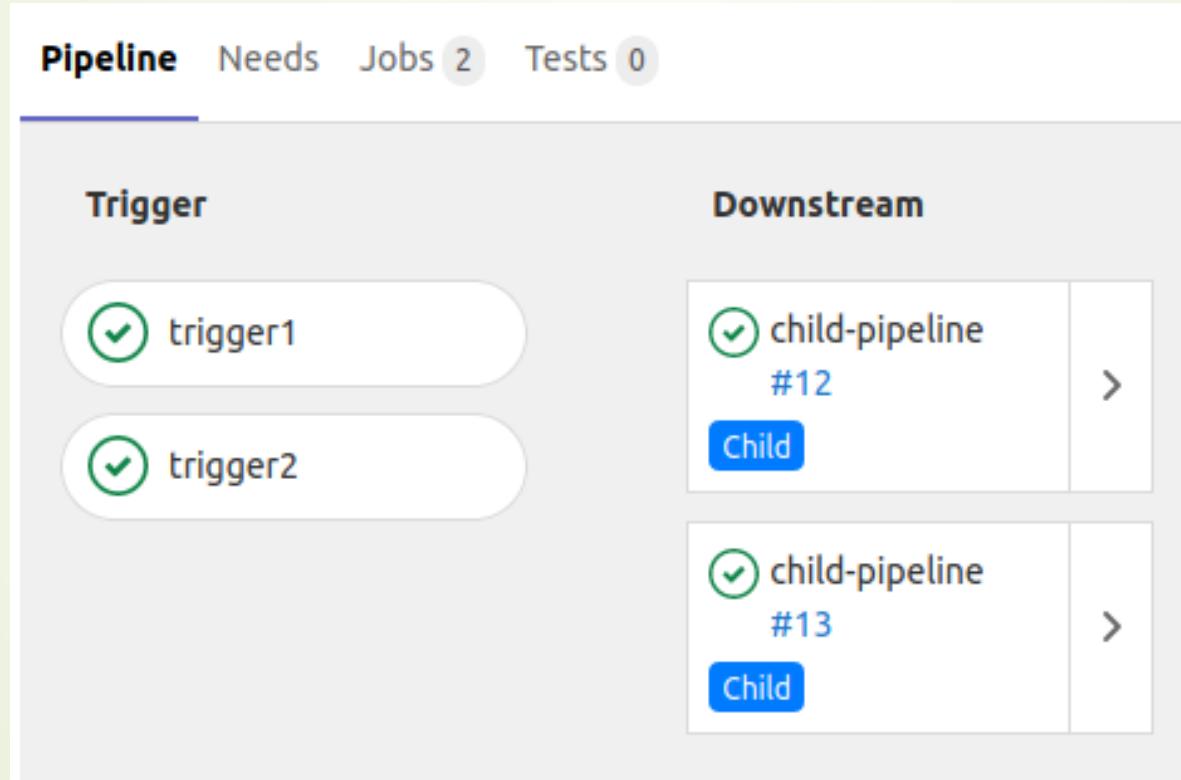
# CI.yml : synchronicité

- Pipeline Parent / Enfant
  - Exécution de sous-pipelines eux même possiblement DAG



# CI.yml : synchronicité

- Pipeline Parent / Enfant dans Gitlab CI



# Cl.ym : conditionnalité

- When, only
  - La clé **when** permet de renseigner une condition d'exécution d'un job
    - **when : on\_success** déclenche si tous les jobs précédents sont en succès
    - **when : on\_failure** déclenche si au moins un des jobs précédents est en échec
    - **when : always** déclenche quelque soient les résultats des jobs précédents
    - **when : manual** délègue le déclenchement à une interaction utilisateur sur l'interface
    - **when : delayed** décale l'exécution du job de la durée en paramètre de la clé **start\_in**
  - Par défaut, la clé **only** permet de renseigner une branche ou un tag git autorisés à exécuter ce job
  - La clé **except** implique l'opposé de **only**. **only** et **except** peuvent être combinés logiquement
  - Le sous clés disponibles :
    - refs**: pour renseigner les branches, tags, les merge requests...
    - changes**: autoriser le job si des changements sont identifiés sur des fichiers du commit courant
    - variables**: autoriser le job en fonction de l'évaluation dde variables

# Cl.yml : conditionnalité

- When, only : exemples

```
job :  
  stage: ...  
  script: ...  
  when: on_failure  
  only:  
    refs:  
      - master  
      - /issue[0-9]+/  
    changes:  
      - bank/*.py  
      - coverage_tests.py
```

# CI.yml : conditionnalité

- rules

- La clé **rules** permet d'écrire des conditions complexes d'exécution au moyens de clauses
  - **if** : déclenche si l'équation logique en paramètre est vraie.  
On peut renseigner plusieurs clauses if
  - **changes** : déclenche si au moins un des chemins en paramètre a subi une modification
  - **exists** : déclenche si au moins un des chemins en paramètre existe
  - **when** (cf infra)
- La clé **workflow** insérée au niveau supérieur du document détermine les règles **rules** de déclenchement du pipeline

```
workflow:  
  rules:  
    - if: '$CI_PIPELINE_SOURCE == "push"  
job:  
  rules:  
    - if: '$CI_COMMIT_BRANCH == "master"  
      when: delayed  
      start_in: '3 hours'  
      allow_failure: true
```

# CI.yml : artefacts

- Gérer les produits des jobs
  - Avec la clé « **artifacts** », toute ressource créée à l'exécution d'un job peut être téléversée
    - dans Gitlab Runner pour être réutilisable dans les jobs suivants,
    - dans Gitlab pour y être téléchargée depuis l'interface
  - La clé « **paths** » renseigne les chemins de prélèvement des artefacts
  - La clé « **expire\_in** » indique le temps de disponibilité d'un artefact avant sa suppression

```
build:  
  stage: builds  
  script:  
    - mkdir test_files  
    - echo "test artifact !" > test_files/test.txt  
artifacts:  
  expire_in: 1 hour  
  paths:  
    - test_files/  
  
test:  
  stage: tests  
  script:  
    - cat test_files/test.txt
```

- Par défaut, les artefacts de tous les jobs précédents du pipelines sont accessibles dans un job donné.
- La clé « **dependencies** » permet de sélectionner les jobs dont les artefacts sont requis pour un job donné
- Ici on pourrait rajouter dans test  
**dependencies : [build]**

# CI.yml : artefacts

- Télécharger les artefacts
  - Au niveau du pipeline, on télécharge les artefacts des différents jobs

Status Pipeline Triggerer Commit Stages

#102 passed latest master 6002f9b5 .gitlab-ci3 00:00:11 42 minutes ago

#101

Download build:archive artifact

- Au niveau du job, via une interface dédiée

Job artifacts

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

Keep Download Browse

Artifacts / test\_files

Name Size

..

test.txt 16 Bytes

Download artifacts archive

# CI.yml : cache

- Cache des dépendances

- Avec la clé « **cache** », toute ressource créée à l'exécution d'un job peut être mise en cache localement ou via une url
- Le cache est utilisé pour éviter de télécharger des dépendances pour chaque job
- La clé « **cache:key** » permet de sélectionner les jobs ayant accès au cache
- La clé « **cache:paths** » permet de renseigner les chemins à mettre en cache

```
job:  
  stage: ...  
  script: ...  
  cache:  
    key: $CI_COMMIT_BRANCH  
    untracked: true  
    paths:  
      - venv/  
  policy: pull-push
```

- **cache:untracked** permet d'ajouter au cache des fichiers non suivis dans le dépôt gitlab
- **cache:policy** permet de renseigner les principaux modes de fonctionnement du cache : par défaut pull-push : le cache est déposé au début et réuploadé à la fin

# CI.yml : cache

- Vider le cache

The screenshot shows a web-based CI pipeline interface. At the top, there is a navigation bar with 'Administrator > test > Pipelines'. Below the navigation, there are three buttons: 'Run Pipeline' (green), 'Clear Runner Caches' (white with a red border), and 'CI Lint' (gray). Underneath the buttons, there is a filter section with 'All 12' selected, followed by 'Finished', 'Branches', and 'Tags'. A 'Filter pipelines' input field is also present. The main area displays a table with columns: Status, Pipeline, Triggerer, Commit, and Stages. One pipeline row is visible, showing a green 'passed' status with a checkmark icon, the pipeline ID '#138 latest', a Jenkins triggerer icon, a commit hash 'e782bd9d' associated with branch 'master', two green checkmark icons in the Stages column, a duration of '00:00:15', and a timestamp of '30 minutes ago'. The entire interface has a clean, modern design with a light gray background and blue/teal accents.