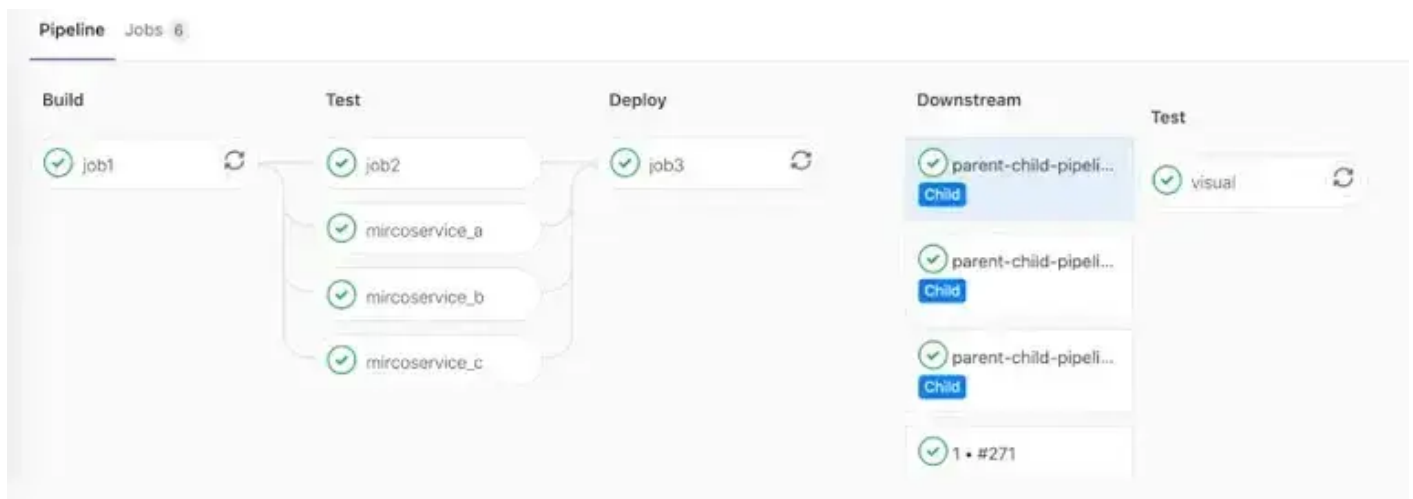


# Les pipelines Parent/Enfant



# GitLab

GitLab CI/CD depuis la version 12.7 a apporté le concept de **pipelines parent-child** (Parent-enfant) qui permet de décomposer des pipelines.



## Comprendre les Pipelines Parent-Enfant

Avec les **pipelines parent-enfant**, un pipeline principal, ou “parent”, peut déclencher un ou plusieurs pipelines “enfants”. Cette hiérarchisation permet une meilleure organisation, une séparation claire des tâches et une plus grande modularité. Par exemple, un pipeline parent peut être responsable de la construction générale et des tests, tandis que les pipelines enfants peuvent être dédiés à des déploiements spécifiques ou à des tests plus approfondis.

Le principal avantage de cette approche est la flexibilité. En séparant les différentes étapes du processus CI/CD en pipelines distincts, on peut mieux gérer les dépendances, les ressources et les autorisations. Cela permet également de réutiliser des configurations de pipeline dans différents projets, améliorant ainsi l’efficacité et réduisant les redondances.

En outre, les **pipelines parent-enfant** favorisent un flux de travail plus propre et plus organisé. Plutôt que d'avoir un pipeline monolithique avec de nombreuses étapes, ce modèle permet de diviser les tâches en unités plus petites et plus gérables. Cela se traduit par une meilleure visibilité sur chaque étape du processus et une détection plus rapide des problèmes potentiels.

## La Directive Trigger de Gitlab CI/CD

La directive `trigger` est utilisée dans le fichier de configuration `.gitlab-ci.yml` d'un projet GitLab. Elle spécifie qu'une tâche dans le pipeline parent doit déclencher un autre pipeline, généralement dans un projet différent. Cela crée une relation hiérarchique où le pipeline parent orchestre le déroulement des pipelines enfants selon les besoins du processus de CI/CD.

Pour configurer cette directive, vous devez spécifier un chemin vers un fichier ou un projet cible. Voici un exemple simple avec un projet :

```
deploy_to_production:
  stage: deploy
  trigger:
    project: my-group/my-production-project
    branch: master
```

Dans cet exemple, la tâche `deploy_to_production` dans le pipeline parent va déclencher un pipeline dans le projet `my-group/my-production-project` sur la branche `master`.

Un autre aspect important de la directive `trigger` est la possibilité de passer des variables d'un pipeline parent à un pipeline enfant. Cela permet une personnalisation fine des pipelines enfants en fonction des résultats ou des paramètres du pipeline parent. Par exemple, vous pourriez passer une variable indiquant le succès des tests dans le pipeline parent pour conditionner le déroulement du pipeline enfant.

La directive `trigger` offre également des options avancées, comme le déclenchement conditionnel des pipelines enfants, basé sur certains critères ou résultats du pipeline parent. Cela ajoute une couche supplémentaire de contrôle et d'efficacité, permettant une gestion plus fine des processus de CI/CD.

## Exemple dans un seul projet

On peut utiliser ce concept dans un seul projet. L'écriture des pipelines enfants est identique à celui qu'on utilise classiquement. Par contre, pour celui du parent, il faudra utiliser la clé `trigger` pour lancer son exécution.

```
stages:
- builds

build1:
stage: builds
trigger:
include: build1.yml
strategy: depend

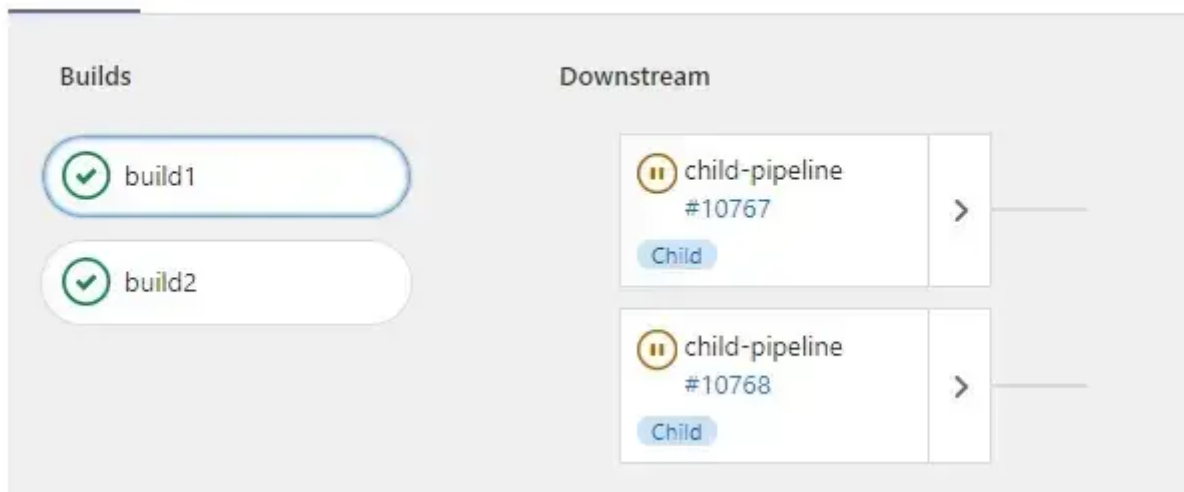
build2:
stage: builds
trigger:
include: build2.yml
strategy: depend
```

Par défaut le pipeline parent se termine tout de suite en status `success` avant même que les enfants ne se soient exécutés. Pour l'obliger à attendre la fin de l'exécution des enfants et de lier son status avec ceux-ci, il faut indiquer une stratégie avec la clé `strategy: depend`

Un des pipelines enfant (l'autre est identique, c'est pour l'exemple) :

```
#build1.yml
build:
stage: build
script:
- touch hello.md
artifacts:
paths:
- hello.md
```

Ce qui donne :



## Exemple en utilisant des templates

Il est possible de lancer des templates de pipelines. Il suffit juste d'indiquer le nom du projet avec la clé `project: stephane.r/test2`

Ce qui donne dans notre exemple :

```
stages:
- builds

build1:
stage: builds
trigger:
include: build1.yml
strategy: depend

build2:
stage: builds
trigger:
include: build2.yml
strategy: depend
```

```
build3:
  stage: builds
  trigger:
  project: stephane.r/test2
  strategy: depend
```

Dans le projet test2, il suffit de créer un fichier `.gitlab-ci.yml`.

Pipeline parent-enfant

## Passer des variables à un enfant

Pour passer des variables aux enfants, il suffit d'utiliser la clé `variables`.

```
build3:
  variables:
  ENVI: staging
  stage: builds
  trigger:
  project: stephane.r/test2
  strategy: depend
```

Dans le ci du projet test2, j'ai modifié le script pour qu'il utilise la variable :

```
build:
  tags:
  - php
  stage: build
  script:
  - echo $ENVI > hello.md
  artifacts:
  paths:
  - hello.md
```

Dans le fichier hello.md, je retrouve bien le contenu de ma variable ENVI.

# Passer des artefacts aux enfants

On peut aussi passer des artefacts, pour cela, il suffit d'ajouter `needs` dans le trigger. Par exemple pour passer des variables du projet test à test2 en utilisant build.env :

```
#test/.gitlab-ci.yml

stages:
- version
- deploy

version:
tags:
- javascript
stage: version
script:
- echo "VERSION=$CI_COMMIT_TAG" >> build.env
- echo "$CI_COMMIT_TAG"
artifacts:
reports:
dotenv: build.env

deploy:
needs:
- build
variables:
VERSION: $VERSION
stage: deploy
trigger:
project: stephane.r/test2
strategy: depend
only:
- tags
```

```
#test2/.gitlab-ci.yml
# build1.yml
```

```
build:
tags:
- javascript
stage: build
script:
- echo "$VERSION"
```

En sortie de script, je récupère bien le tag de test.

```
...

$ echo "$VERSION"
v0.1
Cleaning up file based variables
00:01
Job succeeded
```

## Cas d'usages

Pour illustrer l'impact et l'efficacité des **pipelines parent-enfant** dans GitLab, examinons quelques cas d'utilisation réels.

## Déploiements Multi-Environnements

En utilisant les **pipelines parent-enfant**, on peut créer un pipeline parent qui gère les étapes de construction et de test et des pipelines enfants distincts pour chaque environnement de déploiement. Cela permet de personnaliser les déploiements pour chaque environnement tout en maintenant un processus centralisé et cohérent.