



GitLab Flow

Pipelines avancés

Plan

- 1 Les principaux workflows associés à Git
- 2 Environnements statiques
- 3 Rules
- 4 Environnements dynamiques
- 5 Amélioration du pipeline

Plan

- 1 **Les principaux workflows associés à Git**
- 2 Environnements statiques
- 3 Rules
- 4 Environnements dynamiques
- 5 Amélioration du pipeline

Nécessité d'adopter un workflow

Pour l'instant nos pipelines ne prennent pas en compte les possibilités offertes par Git, à savoir le mécanisme de branches.

L'idée est d'organiser le flux du développement en respectant un processus standardisé pour la création des features, releases, fixes, etc. et d'organiser le CI/CD en conséquence.

Principes généraux d'un workflow basé sur Git

- Permettre un travail collectif efficace
- Suivre les modifications apportées au code source simplement
- Introduire des points de relecture avant validation (merge requests)
- Distinguer pré-production vs. production
- Automatiser le déploiement et la création de releases, en fonction des branches

GitFlow

Basé sur 2 branches principales : *master* et *develop*

- les branches de fonctionnalités sont créées à partir *develop*
- puis fusionnées dans *develop*, une fois testées et validées
- *develop* est fusionnée dans *master* lorsqu'on souhaite créer une release
- les versions, commits tagués, sont créées sur *master*

GitHub Flow

Basé sur une seule branche principale (e.g. *master*)

- les branches de fonctionnalités sont créées à partir de *master*
- puis fusionnées dans *master*, une fois testées et validées
- les versions, commits tagués, sont créées sur *master*

GitLab Flow

Une variante de GitFlow qui ne fixe pas le nom et le nombre de branches pour la pré/production : pas de branche *develop*

► GitLab Flow

Il en existe deux variantes :

Avec des branches d'environnements : *environment branches*

- Deux branches principales : *main* et *production*
- les branches de fonctionnalités sont créées à partir de *main*
- puis testées/déployées dans autant de branches d'environnements (pré-production) que nécessaires : tests, acceptation, ...
- puis fusionnées dans la branche *main*, puis dans la branche *production*

Avec des branches *releases*

- Idem mais avec une branche principale et des branches de version (long terme)
- ⇒ convient lorsqu'on produit du logiciel téléchargeable

GitLab Flow / Principes de mises en œuvre

Bonnes pratiques

▶ GitLab Flow best practices

- 1 Toujours utiliser des branches annexes pour développer
- 2 Effectuer tous les tests pour chaque commit : un push déclenche un pipeline
- 3 Effectuer une code review pour chaque merge request
- 4 Le déploiement est automatisé en fonction des branches, e.g. *production*
- 5 Les tags sont créés par les développeurs, pas par le CI
- 6 Les releases sont basées sur les tags : une release par tag
- 7 Éviter le rebase lors de push, pour un suivi clair de l'historique
- 8 Les développeurs partent de *main* et ciblent *main* pour chaque nouvelle feature
- 9 Fixer les bugs dans *main*, puis dans les branches de releases (cherry-pick)
- 10 Les messages de commit doivent dire explicitement pourquoi il a été réalisé, pas seulement comment

▶ [How to Write a Git Commit Message](#)

Plan

- 1 Les principaux workflows associés à Git
- 2 Environnements statiques**
- 3 Rules
- 4 Environnements dynamiques
- 5 Amélioration du pipeline

Environments

GitLab permet de définir des *environnements* [► Environments](#)
Ils permettent de spécifier le contexte dans lequel un job (déploiement/tests) s'exécute et de suivre facilement le résultat des opérations

Par exemple pour le déploiement, on peut créer plusieurs environnements :

- *staging* : pré-production
- *production* : version active
- créés de façon dynamique et unique pour chaque pipeline, par ex. pour tester une feature

Création d'un environnement statique

Prérequis depuis l'UI de GitLab

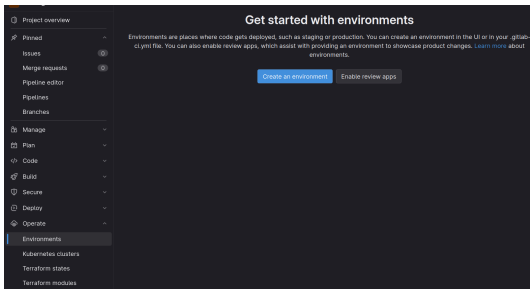
In the UI

Prerequisites:

- You must have at least the Developer role.

To create a static environment in the UI:

1. On the left sidebar, at the top, select **Search GitLab** (`{search}`) to find your project.
2. Select **Operate > Environments**.
3. Select **Create an environment**.
4. Complete the fields.
5. Select **Save**.



Conventions pour le nommage des environnements

Deployment tier of environments

Introduced in GitLab 13.10.

Sometimes, instead of using an [industry standard](#) environment name, like `production`, you might want to use a code name, like `customer-portal`. While there is no technical reason not to use a name like `customer-portal`, the name no longer indicates that the environment is used for production.

To indicate that a specific environment is for a specific use, you can use tiers:

Environment tier	Environment name examples
<code>production</code>	Production, Live
<code>staging</code>	Staging, Model, Demo
<code>testing</code>	Test, QC
<code>development</code>	Dev, Review apps , Trunk
<code>other</code>	

By default, GitLab assumes a tier based on [the environment name](#). Instead, you can use the `deployment_tier` keyword to specify a tier.

Utilisation des environnements

File: .gitlab-ci.yml

```
1 default:
2   image: eclipse-temurin:11-jdk-alpine
3
4 variables:
5   VERSION: "2.0"
6   PACKAGE_NAME: "hello-$VERSION.tar.gz"
7   STAGING_URL: "cicd/staging"
8   PRODUCTION_URL: "cicd/production"
9   FTP_BASE_DIR: "public_html"
10
11 build:
12   stage: build
13   script:
14     - javac hello/Hello.java
15   artifacts:
16     paths:
17       - hello/Hello.class
18
19 test:
20   stage: test
21   script: java hello.Hello | grep -q 'Hello world'
22
23 deploy-staging:
24   image: finalgene/openssh
25   stage: deploy
26   variables:
27     TARGET_DIR: "$FTP_BASE_DIR/$STAGING_URL"
28   script:
29     - echo $PACKAGE_NAME
30     - tar cvzf $PACKAGE_NAME --directory=. hello/*class
31     - echo "<a href=\"\$PACKAGE_NAME\">Last version -> $PACKAGE_NAME</a>" > index.html
32     - apk add lftp #install the lftp package on the alpine distro
33     - lftp -u $IUT_USERNAME,$IUT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR; put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.iutmontp.univ-montp2.fr
34   environment:
35     name: staging
36     url: https://webinfo.iutmontp.univ-montp2.fr/~michel/$STAGING_URL
37
38 deploy-production:
39   image: finalgene/openssh
40   stage: deploy
41   variables:
42     TARGET_DIR: "$FTP_BASE_DIR/$PRODUCTION_URL"
43   script:
44     - echo $PACKAGE_NAME
45     - tar cvzf $PACKAGE_NAME --directory=. hello/*class
46     - echo "<a href=\"\$PACKAGE_NAME\">Last version -> $PACKAGE_NAME</a>" > index.html
47     - apk add lftp #install the lftp package on the alpine distro
48     - lftp -u $IUT_USERNAME,$IUT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR; put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.iutmontp.univ-montp2.fr
49   environment:
50     name: production
```

Visualisation des activités par environnement

staging								Edit	Stop
Status	ID	Triggerer	Commit	Job	Created	Deployed	Actions		
Running	5		 01_Using_static_environments dc9c4c2d Using static environments f...	deploy-staging (#39272)	6 minutes ago				
Cancelled	3		 01_Using_static_environments 8e6a4878 Using static environments f...	deploy-staging (#3926...)	23 minutes ago	6 minutes ago			

Résultats du déploiement

Open permet d'aller directement sur l'URL

The screenshot shows the GitLab 'Environments' page for the project 'ci_cd_advanced_GitLab'. The page displays two environments: 'production' and 'staging'. Both environments show a successful deployment status with a 'Latest Deployed' badge. The 'production' environment is associated with commit #11 (de2ab782) and was deployed 'just now'. The 'staging' environment is associated with commit #10 (de2ab782) and was also deployed 'just now'. Both environments are using the 'Using static environments for staging and deploying' strategy. The page includes a sidebar with navigation options like 'Project overview', 'Pinned', 'Issues', 'Merge requests', 'Pipeline editor', 'Pipelines', 'Branches', 'Environments', 'Manage', 'Plan', 'Code', 'Build', 'Secure', 'Deploy', 'Operate', and 'Environments'. The main content area has tabs for 'Available' (2) and 'Stopped' (0) environments, along with buttons for 'Clean up environments', 'Enable review apps', and 'New environment'.

Environment	Status	Latest Deployed	Commit	Branch	Strategy
production	Success	Latest Deployed	#11 de2ab782	01_Using_stat...	Using static environments for staging and deploying
staging	Success	Latest Deployed	#10 de2ab782	01_Using_stat...	Using static environments for staging and deploying

Exemple sur

► <https://webinfo.iutmontp.univ-montp2.fr/michel/cicd/staging/>

Plan

- 1 Les principaux workflows associés à Git
- 2 Environnements statiques
- 3 Rules**
- 4 Environnements dynamiques
- 5 Amélioration du pipeline

Mot-clé *rules*

Il permet de spécifier quand les jobs doivent être exécutés [rules](#)

Certains jobs ne doivent pas être exécutés tout le temps

- *deploy-production* doit être exécuté uniquement pour un tag sur la branche *production*
- *deploy-staging* uniquement pour *main*

Specify when jobs run with `rules`

Introduced in GitLab 12.3.

Use `rules` to include or exclude jobs in pipelines.

Rules are evaluated in order until the first match. When a match is found, the job is either included or excluded from the pipeline, depending on the configuration. See the [rules](#) reference for more details.

Future keyword improvements are being discussed in our [epic for improving rules](#), where anyone can add suggestions or requests.

Exemples d'utilisation 1

rules examples

The following example uses `if` to define that the job runs in only two specific cases:

```
job:
  script: echo "Hello, Rules!"
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      when: manual
      allow_failure: true
    - if: $CI_PIPELINE_SOURCE == "schedule"
```

- If the pipeline is for a merge request, the first rule matches, and the job is added to the [merge request pipeline](#) with attributes of:
 - `when: manual` (manual job)
 - `allow_failure: true` (the pipeline continues running even if the manual job is not run)
- If the pipeline is **not** for a merge request, the first rule doesn't match, and the second rule is evaluated.
- If the pipeline is a scheduled pipeline, the second rule matches, and the job is added to the scheduled pipeline. No attributes were defined, so it is added with:
 - `when: on_success` (default)
 - `allow_failure: false` (default)
- In **all other cases**, no rules match, so the job is **not** added to any other pipeline.

Exemples d'utilisation 2

Alternatively, you can define a set of rules to exclude jobs in a few cases, but run them in all other cases:

```
job:
  script: echo "Hello, Rules!"
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
      when: never
    - if: $CI_PIPELINE_SOURCE == "schedule"
      when: never
    - when: on_success
```

- If the pipeline is for a merge request, the job is **not** added to the pipeline.
- If the pipeline is a scheduled pipeline, the job is **not** added to the pipeline.
- In **all other cases**, the job is added to the pipeline, with `when: on_success`.

⚠ If you use a `when` clause as the final rule (not including `when: never`), two simultaneous pipelines may start. Both push pipelines and merge request pipelines can be triggered by the same event (a push to the source branch for an open merge request). See how to [prevent duplicate pipelines](#) for more details.

Exemples d'utilisation 3

Skip job if the branch is empty

Use `rules:changes:compare_to` to avoid running a job when the branch is empty, which saves CI/CD resources. Compare the branch to the default branch, and if the branch:

- Doesn't have changed files, the job doesn't run.
- Has changed files, the job runs.

For example, in a project with `main` as the default branch:

```
job:
  script:
    - echo "This job only runs for branches that are not empty"
  rules:
    - if: $CI_COMMIT_BRANCH
      changes:
        compare_to: 'refs/heads/main'
        paths:
          - '*'
```

The rule for this job compares all files and paths (*) in the current branch against the default branch `main`. The rule matches and the job runs only when there are changes to the files in the branch.

Utilisation classiques de variables prédéfinies

Other commonly used variables for `if` clauses:

- `if: $CI_COMMIT_TAG`: If changes are pushed for a tag.
- `if: $CI_COMMIT_BRANCH`: If changes are pushed to any branch.
- `if: $CI_COMMIT_BRANCH == "main"`: If changes are pushed to `main`.
- `if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH`: If changes are pushed to the default branch. Use when you want to have the same configuration in multiple projects with different default branches.
- `if: $CI_COMMIT_BRANCH =~ /regex-expression/`: If the commit branch matches a regular expression.
- `if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH && $CI_COMMIT_TITLE =~ /Merge branch.*/`: If the commit branch is the default branch and the commit message title matches a regular expression. For example, the default commit message for a merge commit starts with `Merge branch`.
- `if: $CUSTOM_VARIABLE != /regex-expression/`: If the [custom variable](#) `CUSTOM_VARIABLE` does **not** match a regular expression.
- `if: $CUSTOM_VARIABLE == "value1"`: If the custom variable `CUSTOM_VARIABLE` is exactly `value1`.

Utilisation classiques de variables prédéfinies

The following table lists some of the variables that you can use, and the pipeline types the variables can control for:

- Branch pipelines that run for Git `push` events to a branch, like new commits or tags.
- Tag pipelines that run only when a new Git tag is pushed to a branch.
- [Merge request pipelines](#) that run for changes to a merge request, like new commits or selecting the **Run pipeline** button in a merge request's pipelines tab.
- [Scheduled pipelines](#).

Variables	Branch	Tag	Merge request	Scheduled
<code>CI_COMMIT_BRANCH</code>	Yes			Yes
<code>CI_COMMIT_TAG</code>		Yes		Yes, if the scheduled pipeline is configured to run on a tag.
<code>CI_PIPELINE_SOURCE = push</code>	Yes	Yes		
<code>CI_PIPELINE_SOURCE = schedule</code>				Yes
<code>CI_PIPELINE_SOURCE = merge_request_event</code>			Yes	
<code>CI_MERGE_REQUEST_IID</code>			Yes	

Réutilisation de règles

Il est possible de réutiliser des règles afin de factoriser le code et d'améliorer sa lisibilité

Reuse rules in different jobs

Introduced in GitLab 14.3.

Use `!reference tags` to reuse rules in different jobs. You can combine `!reference` rules with regular job-defined rules:

```
.default_rules:
  rules:
    - if: $CI_PIPELINE_SOURCE == "schedule"
      when: never
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

job1:
  rules:
    - !reference [.default_rules, rules]
  script:
    - echo "This job runs for the default branch, but not schedules."

job2:
  rules:
    - !reference [.default_rules, rules]
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
  script:
    - echo "This job runs for the default branch, but not schedules."
    - echo "It also runs for merge requests."
```

Application sur notre pipeline

On modifie le pipeline pour s'aligner sur un GitLab Flow (environment)

```
23  deploy-staging:
24    image: finalgene/openssh
25    stage: deploy
26    variables:
27      TARGET_DIR: "$FTP_BASE_DIR/$STAGING_URL"
28    script:
29      - echo $PACKAGE_NAME
30      - tar cvzf $PACKAGE_NAME --directory=. hello/*class
31      - echo "<a href=\"\$PACKAGE_NAME\">Last version -> $PACKAGE_NAME</a>" > index.html
32      - apk add lftp #install the lftp package on the alpine distro
33      - echo $TARGET_DIR
34      - lftp -u $IUT_USERNAME,$IUT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR;
put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.iutmontp2.univ-montp2.fr
35    environment:
36      name: staging
37      url: https://webinfo.iutmontp2.univ-montp2.fr/~michel/$STAGING_URL/
38    rules:
39      - if: $CI_COMMIT_BRANCH == "main"
40
41  deploy-production:
42    image: finalgene/openssh
43    stage: deploy
44    variables:
45      TARGET_DIR: "$FTP_BASE_DIR/$PRODUCTION_URL"
46    script:
47      - echo $TARGET_DIR
48      - echo $PACKAGE_NAME
49      - tar cvzf $PACKAGE_NAME --directory=. hello/*class
50      - echo "<a href=\"\$PACKAGE_NAME\">Last version -> $PACKAGE_NAME</a>" > index.html
51      - apk add lftp #install the lftp package on the alpine distro
52      - lftp -u $IUT_USERNAME,$IUT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR;
put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.iutmontp2.univ-montp2.fr
53    environment:
54      name: production
55    rules:
```

Les jobs sont maintenant filtrés en fonction du contexte

Les commits sur la branche *feature* ne déclenchent plus les déploiements :

✓ passed Pipeline #18556 triggered 1 minute ago by MICHEL Fabien

Staging and production are only triggered when on the right branches

🕒 2 jobs for [02_Using_rules_for_specifying_when_jobs_run](#)
in 13 seconds, using 0.0 compute credits, and was queued for 1 second

🚩 latest

🔗 cdcafae4

🔍 No related merge requests found.

Pipeline Needs Jobs 2 Tests 0

build test

✓ build ✓ test

Création d'une merge request sur main

New merge request

From `02_Using_rules_for_specifying_when_jobs_run` into `main` [Change branches](#)

Title (required)

Staging and production are only triggered when on the right branches

☒ Mark as draft
Drafts cannot be merged until marked ready.

Description

B I H L U T **Preview**

Describe the goal of the changes and what reviewers should be aware of.

Supports Markdown. For quick actions, type `/`.

Add description templates to help your contributors to communicate effectively!

Assignees

MICHEL Fabien

Reviewers

MICHEL Fabien

Approvals are optional.
[Approval rules](#)

Milestone

Select milestone

Labels

Labels

Merge request dependencies

List the merge requests that must be merged before this one. [Learn more.](#)

Enter merge request URLs or references

References should be in the form of `path/to/project/merge_request_id`

Merge options

☒ Delete source branch when merge request is accepted.

☐ Squash commits when merge request is accepted. ⓘ

[Create merge request](#) [Cancel](#)

Déclenchement du déploiement sur staging

02 using rules for specifying when jobs run Edit Code

MICHEL Fabien requested to merge `02_Using_rules_for_spe...` into `main` 7 minutes ago

Overview 0 Commits 2 Pipelines 3 Changes 1

0 0

✓ Pipeline #18558 passed for `f4be58c1` on `02_Using_rules_for...` 2 minutes ago

8 Approval is optional

Merged by MICHEL Fabien 1 minute ago Revert Cherry-pick Delete source branch

Merge details

- Changes merged into `main` with `5ceeb52f`.
- Did not delete the source branch.

🔄 Pipeline #18559 running for `5ceeb52f` on `main`

Deployed to `staging` 1 minute ago View app

Activity

Sort or filter

- MICHEL Fabien requested review from @michel 7 minutes ago
- MICHEL Fabien assigned to @michel 7 minutes ago
- MICHEL Fabien added 1 commit 2 minutes ago
 - `f4be58c1` - Staging and production are only triggered when on the right branches
 - [Compare with previous version](#)
- MICHEL Fabien mentioned in commit `5ceeb52f` 1 minute ago
- MICHEL Fabien merged 1 minute ago

Exécution du pipeline sur la branche *main*

 **passed** Pipeline #18559 triggered just now by  MICHEL Fabien

Merge branch '02_Using_rules_for_specifying_when_jobs_run' into 'main'

02 using rules for specifying when jobs run

See merge request !1

 3 jobs for **main**

 **latest**

 5ceeb52f 

 No related merge requests found.

Pipeline Needs Jobs 3 Tests 0

build	test	deploy
 build 	 test 	 deploy-staging 

Fin du merge de la merge request ⇒ pré-prod OK

The screenshot displays a GitLab merge request interface. At the top, there are tabs for Overview (0), Commits (2), Pipelines (3), and Changes (1). The main content area shows a successful merge by MICHEL Fabien 1 hour ago. Below this, the merge details are listed: Changes merged into main with 5ceeb52f, and Did not delete the source branch. The interface also shows two successful CI pipelines: Pipeline #18558 passed for f4be58c1 on 02_Using_rules_for_... 1 hour ago, and Pipeline #18559 passed for 5ceeb52f on main 1 hour ago. The merge is deployed to staging 1 hour ago. The Activity section shows a list of actions: MICHEL Fabien requested review from @michel 1 hour ago, MICHEL Fabien assigned to @michel 1 hour ago, MICHEL Fabien added 1 commit 1 hour ago (f4be58c1 - Staging and production are only triggered when on the right branches), MICHEL Fabien mentioned in commit 5ceeb52f 1 hour ago, and MICHEL Fabien merged 1 hour ago. The bottom of the interface shows a rich text editor with various formatting options and a Preview button.

Overview 0 Commits 2 Pipelines 3 Changes 1

✓ Pipeline #18558 passed for f4be58c1 on 02_Using_rules_for_... 1 hour ago

8✓ Approval is optional

Merged by MICHEL Fabien 1 hour ago

Revert Cherry-pick

Merge details

- Changes merged into main with 5ceeb52f.
- Did not delete the source branch.

✓ Pipeline #18559 passed for 5ceeb52f on main 1 hour ago

Deployed to staging 1 hour ago

View app

Activity

Sort or filter

- MICHEL Fabien requested review from @michel 1 hour ago
- MICHEL Fabien assigned to @michel 1 hour ago
- MICHEL Fabien added 1 commit 1 hour ago
 - f4be58c1 - Staging and production are only triggered when on the right branches[Compare with previous version](#)
- MICHEL Fabien mentioned in commit 5ceeb52f 1 hour ago
- MICHEL Fabien merged 1 hour ago

B I Preview

Mise en production

Il faut maintenant faire une merge request de *main* vers *production*

The screenshot shows a GitHub pull request titled "release 2.0". At the top, it says "MICHEL Fabien requested to merge `main` into `production` just now". There are buttons for "Edit" and "Code". Below this, there are tabs for "Overview 0", "Commits", "Pipelines 1", and "Changes". The main content area shows a commit message "Nice Hello World" with 0 likes and 0 comments. Below the commit, there is a green checkmark indicating that "Pipeline #18563 passed for 5cd2c8d3 on main 27 minutes ago". There is also a status "Deployed to staging 27 minutes ago" with a "View app" button. A section titled "Approval is optional" shows a green checkmark and the text "Ready to merge!". Below this, there are checkboxes for "Squash commits" and "Edit commit message", with a note that "5 commits and 1 merge commit will be added to production." and a "Merge" button. At the bottom, the "Activity" section shows two items: "MICHEL Fabien requested review from @michel just now" and "MICHEL Fabien assigned to @michel just now".

release 2.0

Open MICHEL Fabien requested to merge `main` into `production` just now

Edit Code

Overview 0 Commits Pipelines 1 Changes

Nice Hello World

0 0

✓ Pipeline #18563 passed for 5cd2c8d3 on main 27 minutes ago

Deployed to staging 27 minutes ago View app

8✓ Approval is optional

✓ Ready to merge!

☐ Squash commits ☐ Edit commit message

5 commits and 1 merge commit will be added to production.

Merge

Activity

- MICHEL Fabien requested review from @michel just now
- MICHEL Fabien assigned to @michel just now

Mais on a une petite erreur dans le job...

Le bloc *rules* ne pouvait pas être déclenché par la création d'un tag
Une fois corrigée dans une branche feature, on recommence les MR,
puis la création d'un tag sur *production* déclenche bien le pipeline

Merge branch 'main' into 'production'

2.0 fix

See merge request [15](#)

Parents: [b9a811c1 b0a8fb81](#)

Branches: [production](#)

Tags: [2.0.1](#)

1 merge request [15](#) 2.0 fix

Pipeline [#18570](#) running with stages

Changes [1](#) Pipelines [2](#)

Showing [1](#) changed file with [1](#) addition and [1](#) deletion

[Hide whitespace changes](#) [Inline](#) [Side-by-side](#)

▼ [.gitlab-ci.yml](#)

+1 -1 [View file @84faa99e](#)

```
... .. @@ -49,4 +49,4 @@ deploy-production:
49 49   environment:
50 50     name: production
51 51     rules:
52 52     - if: $CI_COMMIT_BRANCH == "production" && $CI_COMMIT_TAG
52 52     + if: $CI_COMMIT_TAG
```

Déclencher un job manuellement

Il est possible de définir qu'un job est uniquement déclenché manuellement, depuis l'UI de GitLab avec le mot-clé *when* dans le bloc *rules*

Par exemple pour déclencher le déploiement en production au moment le plus propice, et pas automatiquement en fin de pipeline (les jobs précédents peuvent être longs)

```
deploy-production:
  image: finalgene/openssh
  stage: deploy
  variables:
    TARGET_DIR: "$FTP_BASE_DIR/$PRODUCTION_URL"
  script:
    - tar cvzf $PACKAGE_NAME --directory=. hello/*class
    - 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 $IUT_USERNAME,$IUT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR;
      put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.iutmontp.univ-montp2.fr
  environment:
    name: production
  rules:
    - if: $CI_COMMIT_TAG
      when: manual
```

Déclencher un job manuellement

Le déploiement n'est plus automatique :

MICHEL Fabien > ci_cd_advanced_GitLab > Pipelines

All 29 Finished Branches Tags

Clear runner caches CI lint Run pipeline

Filter pipelines

Show Pipeline ID

Status	Pipeline	Triggerer	Stages
⏸ blocked ⌚ 00:00:12	Merge branch 'main' into 'production' #18575 3.0 5305efca latest		✓ ✓ ⚙ Stage: deploy ⏸ deploy-production
✓ passed ⌚ 00:00:12 🕒 1 minute ago	Merge branch 'main' into 'production' #18574 ? production 5305efca latest		⏸ deploy-production

Plan

- 1 Les principaux workflows associés à Git
- 2 Environnements statiques
- 3 Rules
- 4 Environnements dynamiques**
- 5 Amélioration du pipeline

Environnements dynamiques

Il est possible de créer des environnements éphémères dynamiquement, e.g. pour tester le résultat d'une feature avant un merge.

On a un squelette proposé dans la section environment de GitLab, avec le bouton **Enable Review Apps** :

Enable Review Apps

Review apps are dynamic environments that you can use to provide a live preview of changes made in a feature branch.

To configure a dynamic review app, you must:

1. Have access to infrastructure that can host and deploy the review apps. ⓘ
2. Install and configure a runner to do the deployment.
3. Add a job in your CI/CD configuration that:
 - Only runs for feature branches or merge requests.
 - Uses a predefined CI/CD variable like `$(CI_COMMIT_REF_SLUG)` to dynamically create the review app environments. For example, for a configuration using merge request pipelines:

```
deploy_review:
  stage: deploy
  script:
    - echo "Add script here that deploys the code to your infrastructure"
  environment:
    name: review/${CI_COMMIT_REF_NAME}
    url: https://${CI_ENVIRONMENT_SLUG}.example.com
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
```



4. Recommended: Set up a job that manually stops the Review Apps.

Dans notre pipeline

On ajoute un job `deploy-review` qui sera déclenché par une merge request :

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

test:
  stage: test
  script: java hello.Hello | grep -q 'Hello world'

deploy-review:
  image: finalgene/openssh
  stage: deploy
  variables:
    TARGET_DIR: "$FTP_BASE_DIR/review/$CI_COMMIT_REF_NAME"
  script:
    - echo $TARGET_DIR
    - tar cvzf $PACKAGE_NAME --directory=. hello/*.class
    - echo "<a href=\""$PACKAGE_NAME\"">Testing version -> $PACKAGE_NAME</a>" > index.html
    - apk add lftp #install the lftp package on the alpine distro
    - lftp -u $IUT_USERNAME,$IUT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR; put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.iutmontp.univ-montp2.fr
  environment:
    name: review/$CI_COMMIT_REF_NAME
    url: https://webinfo.iutmontp.univ-montp2.fr/~michel/review/$CI_COMMIT_REF_NAME
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
```

Création de la merge request

Elle déclenche bien le job, mais il reste quelque chose à faire. . .

Deploying a live preview on merge requests

🕒 1 job for !8 with 05_Using_dynamic_environments
in 5 seconds, using 0.0 compute credits, and was queued for 8 seconds

🚩 latest merge request

🔗 17d6dc7b

🔗 1 related merge request: !8 Deploying a live preview on merge requests

Pipeline Needs Jobs 1 Failed Jobs 1 Tests 0

deploy

❌ deploy-review ↻

```
23 Using docker image sha256:f5b59cd89a3d62fc66bef3119537442a83bc61aa0da9c0
    8498f379f4866b1b3a for finalgene/openssh with digest finalgene/openssh@sh
    a256:3488c5dd43ec2bd7125b75bd2984775ae84ab3e6e24a1e99b802d6ac7abaa267 ...
24 $ echo $TARGET_DIR
25 public_html/review/05_Using_dynamic_environments
26 $ tar cvzf $PACKAGE_NAME --directory=. hello/*class
27 tar: hello/*class: No such file or directory
28 tar: error exit delayed from previous errors
29
30 Cleaning up project directory and file based variables
31
32 ERROR: Job failed: exit code 1
```

❌ Pipeline #18681 for !8 with 05_Using_dynamic_environments

🔗

deploy ▼

→ ❌ deploy-review

00:00

Ajout d'une règle au job `build`

Pour que notre job `deploy-review` fonctionne, il a besoin de l'artefact produit par le job `build` : il faut qu'il soit lancé !

⇒ On utilise pour cela la règle `when: always`

```
build:
  stage: build
  script:
    - javac hello/Hello.java
  artifacts:
    paths:
      - hello/Hello.class
  rules:
    - when: always

test:
  stage: test
  script: java hello.Hello | grep -q 'Hello world'

deploy-review:
  image: finalgene/openssh
  stage: deploy
  variables:
    TARGET_DIR: "$FTP_BASE_DIR/review/$CI_COMMIT_REF_NAME"
  script:
    - echo $TARGET_DIR
    - tar cvzf $PACKAGE_NAME --directory=. hello/*.class
    - echo "<a href=\"\$PACKAGE_NAME\">Testing version -> \$PACKAGE_NAME</a>" > index.html
    - apk add lftp #install the lftp package on the alpine distro
    - lftp -u $IUT_USERNAME,$IUT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR;
e;" sftp://ftpinfo.iutmontp.univ-montp2.fr
  environment:
    name: review/$CI_COMMIT_REF_NAME
    url: https://webinfo.iutmontp.univ-montp2.fr/~michel/review/$CI_COMMIT_REF_NAME
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
```

Nouveau résultat pour la merge request

Les deux jobs ont bien été lancés

The screenshot displays a GitLab merge request interface. At the top, the title is "Deploying a live preview on merge requests". Below the title, it indicates that "MICHEL Fabien requested to merge 05_Using_dynamic_envir... into main" 16 hours ago. The interface includes tabs for "Overview", "Commits", "Pipelines", and "Changes". There are buttons for "Open", "Edit", and "Code". Below the tabs, there are reaction buttons (thumbs up, thumbs down, and a neutral face) with counts of 0. The main content area shows a green checkmark indicating that the "Merge request pipeline #18683 passed for bce47218" 15 hours ago. Below this, it states "Deployed to review/05_Using_dynamic_environments" 15 hours ago, with a "View app" button. A section titled "Approval is optional" is also visible. At the bottom, a "Ready to merge!" section shows options to "Delete source branch", "Squash commits", and "Edit commit message". It notes that "1 commit and 1 merge commit will be added to main." and includes a "Merge" button.

Stopper et nettoyer un environnement dynamique

Une fois la MR validée ou fermée, l'environnement peut être stoppé, puis effacé. [► Stop an environment](#)

In the following example, a `deploy_review` job calls a `stop_review` job to clean up and stop the environment.

- Both jobs must have the same `rules` or `only/except` configuration. Otherwise, the `stop_review` job might not be included in all pipelines that include the `deploy_review` job, and you cannot trigger `action: stop` to stop the environment automatically.
- The job with `action: stop` might not run if it's in a later stage than the job that started the environment.
- If you can't use `merge request pipelines`, set the `GIT_STRATEGY` to `none` in the `stop_review` job. Then the `runner` doesn't try to check out the code after the branch is deleted.

```
deploy_review:
  stage: deploy
  script:
    - echo "Deploy a review app"
  environment:
    name: review/$CI_COMMIT_REF_SLUG
    url: https://$CI_ENVIRONMENT_SLUG.example.com
    on_stop: stop_review

stop_review:
  stage: deploy
  script:
    - echo "Remove review app"
  environment:
    name: review/$CI_COMMIT_REF_SLUG
    action: stop
  when: manual
```

Ajout du job `stop_review` dans le pipeline

Il consiste dans un simple nettoyage du répertoire correspondant

```
deploy-review:
  image: finalgene/openssh
  stage: deploy
  variables:
    TARGET_DIR: "$FTP_BASE_DIR/review/$CI_COMMIT_REF_NAME"
  script:
    - tar cvzf $PACKAGE_NAME --directory=. hello/*class
    - echo "<a href=\"${PACKAGE_NAME}\">Testing version -> $PACKAGE_NAME</a>" > index.html
    - apk add lftp #install the lftp package on the alpine distro
    - lftp -u $IUT_USERNAME,$IUT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR; put $PACKAGE_NAME index.htm
.univ-montp2.fr
  environment:
    name: review/$CI_COMMIT_REF_NAME
    url: https://webinfo.iutmontp.univ-montp2.fr/~michel/review/$CI_COMMIT_REF_NAME
    on_stop: stop_review
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"

stop_review:
  image: finalgene/openssh
  stage: deploy
  variables:
    TARGET_DIR: "$FTP_BASE_DIR/review/$CI_COMMIT_REF_NAME"
  script:
    - apk add lftp
    - lftp -u $IUT_USERNAME,$IUT_PASSWORD -e "set ftp:ssl-allow 0; rm -rf $TARGET_DIR; bye;" sftp://ftpinfo.iutmontp.univ-montp2.fr
  environment:
    name: review/$CI_COMMIT_REF_NAME
    action: stop
  rules:
    - if: $CI_MERGE_REQUEST_ID
      when: manual
```

Apparition du job `stop_review` dans la MR

06 stopping dynamic environments Edit Code ▾ ⋮

Open MICHEL Fabien requested to merge `06_Stopping_dynamic_en...` into `main` 5 minutes ago

Overview 0 **Commits** - **Pipelines** 1 **Changes** -

0 0

Merge request pipeline #18688 waiting for manual action for 3a85dcc8

Deployed to [review/06_Stopping_dynamic_environments](#) 20 minutes ago

8✓ Approval is optional

Ready to merge!

☐ Delete source branch ☐ Squash commits ☐ Edit commit message

2 commits and 1 merge commit will be added to `main`.

Set to auto-merge ▾ Merge when pipeline succeeds

Stage: deploy

- stop_review
- deploy-review

Déclenchement du job `stop_review`

Il sera déclenché manuellement : par (1) merge/close de la MR ou (2) un clic sur le job.

Exemple avec un *close* de la MR

06 stopping dynamic environments

Edit

Code ▾

⋮



MICHEL Fabien requested to merge 06_Stopping_dynamic_en... into main 26 minutes ago

Overview 0

Commits 2

Pipelines 8

Changes 1



0



0



Merge request pipeline #18704 passed for b5d8b738 7 minutes ago



Deployed to review/06_Stopping_dynamic_environments 9 minutes ago

8✓

Approval is optional

Stage: deploy



deploy-review



stop_review



Closed by MICHEL Fabien 7 minutes ago

Reopen

Merge details

- The changes were not merged into main.

Plan

- 1 Les principaux workflows associés à Git
- 2 Environnements statiques
- 3 Rules
- 4 Environnements dynamiques
- 5 Amélioration du pipeline

Beaucoup de choses sont à améliorer

```

default:
  image: eclipse-temurin:11-jdk-alpine

vars:
  VERSION: "4.0"
  PACKAGE_NAME: "hello-VERSION.tar.gz"
  STAGING_URL: "http://staging"
  PRODUCTION_URL: "http://production"
  FTP_BASE_DIR: "public_html"

build:
  stage: build
  script:
    - java hello/hello.java
  artifacts:
    - path:
      - hello/hello.class
  rules:
    - when: always

test:
  stage: test
  script: java hello/hello | grep -q 'Hello world'

deploy-review:
  image: finalgame/openssh
  stage: deploy
  variables:
    TARGET_DIR: "${FTP_BASE_DIR}/cicd/review/${SCI_COMMIT_REF_NAME}"
  script:
    - tar czf $PACKAGE_NAME --directory= hello/class
    - echo "a href='${PACKAGE_NAME}'>testing version -> $PACKAGE_NAME/a" > index.html
    - apk add ftp install the ftp package on the alpine distro
    - ftp -u $!UT_USERNAME $!UT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR; put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.tutomtp.univ-montp2.fr
  environment:
    name: review/${SCI_COMMIT_REF_NAME}
    url: https://webinfo.tutomtp.univ-montp2.fr/~michel/cicd/review/${SCI_COMMIT_REF_NAME}
    on_stop: stop_review
  rules:
    - if: SCI_PIPELINE_SOURCE == "merge_request_event"

stop_review:
  image: finalgame/openssh
  stage: deploy
  variables:
    TARGET_DIR: "${FTP_BASE_DIR}/cicd/review/${SCI_COMMIT_REF_NAME}"
  script:
    - apk add ftp
    - ftp -u $!UT_USERNAME $!UT_PASSWORD -e "set ftp:ssl-allow 0; rm -rf $TARGET_DIR; bye;" sftp://ftpinfo.tutomtp.univ-montp2.fr
  environment:
    name: review/${SCI_COMMIT_REF_NAME}
    action: stop
  rules:
    - if: SCI_MERGE_REQUEST_ID
      when: manual

deploy-staging:
  image: finalgame/openssh
  stage: deploy
  variables:
    TARGET_DIR: "${FTP_BASE_DIR}/staging_url"
  script:
    - tar czf $PACKAGE_NAME --directory= hello/class
    - echo "a href='${PACKAGE_NAME}'>last version -> $PACKAGE_NAME/a" > index.html
    - apk add ftp install the ftp package on the alpine distro
    - ftp -u $!UT_USERNAME $!UT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR; put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.tutomtp.univ-montp2.fr
  environment:
    name: staging
    url: https://webinfo.tutomtp.univ-montp2.fr/~michel/$STAGING_URL/
  rules:
    - if: SCI_COMMIT_BRANCH == "main"

deploy-production:
  image: finalgame/openssh
  stage: deploy
  variables:
    TARGET_DIR: "${FTP_BASE_DIR}/production_url"
  script:
    - tar czf $PACKAGE_NAME --directory= hello/class
    - echo "a href='${PACKAGE_NAME}'>last version -> $PACKAGE_NAME/a" > index.html
    - apk add ftp install the ftp package on the alpine distro
    - ftp -u $!UT_USERNAME $!UT_PASSWORD -e "set ftp:ssl-allow 0; mkdir -p $TARGET_DIR; cd $TARGET_DIR; put $PACKAGE_NAME index.html; bye;" sftp://ftpinfo.tutomtp.univ-montp2.fr
  environment:
    name: production
  rules:
    - if: SCI_COMMIT_TAG
      when: manual

```

Beaucoup de choses sont à améliorer

Défauts potentiels

- ❶ Il y a trop de lignes semblables
- ❷ Le numéro de version ne devrait pas être toujours identique
- ❸ `PRODUCTION_URL / STAGING_URL...` ne devraient pas être là

Améliorations possibles

- ❶ Solutions pour la factorisation
 - Utiliser le mot-clé `include` pour factoriser 
 - Utiliser un moteur de production comme Gradle
 - Chercher ou créer une image docker adaptée aux besoins de déploiement
- ❷ Utiliser le contexte pour nommer la version : feature / pré-prod / tags
- ❸ Utiliser des variables définies dans l'UI de GitLab

Ce que nous n'avons pas vu

L'utilisation du mot-clé `workflow` pour définir le comportement du pipeline en fonction du contexte [▶ workflow](#)

Examples of `workflow:name`:

A simple pipeline name with a predefined variable:

```
workflow:
  name: 'Pipeline for branch: $CI_COMMIT_BRANCH'
```

A configuration with different pipeline names depending on the pipeline conditions:

```
variables:
  PROJECT1_PIPELINE_NAME: 'Default pipeline name' # A default is not required.

workflow:
  name: '$PROJECT1_PIPELINE_NAME'
  rules:
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
      variables:
        PROJECT1_PIPELINE_NAME: 'MR pipeline: $CI_MERGE_REQUEST_SOURCE_BRANCH_NAME'
    - if: '$CI_MERGE_REQUEST_LABELS =~ /pipeline:run-in-ruby3/'
      variables:
        PROJECT1_PIPELINE_NAME: 'Ruby 3 pipeline'
    - when: always # Other pipelines can run, but use the default name
```

Conclusion

Nécessité de suivre un workflow

- ⇒ Le développement doit suivre un workflow
- ⇒ Le GitLab flow existe en deux versions : environnement et releases
- ⇒ Les environnements de GitLab permettent de simplifier le suivi du workflow

Les rules permettent d'exécuter les jobs en fonction du workflow

- L'exécution des jobs est en fonction du contexte : feature / pré-prod / tags
- On utilise pour ce la les variables prédéfinies, e.g. `CI_COMMIT_TAG`