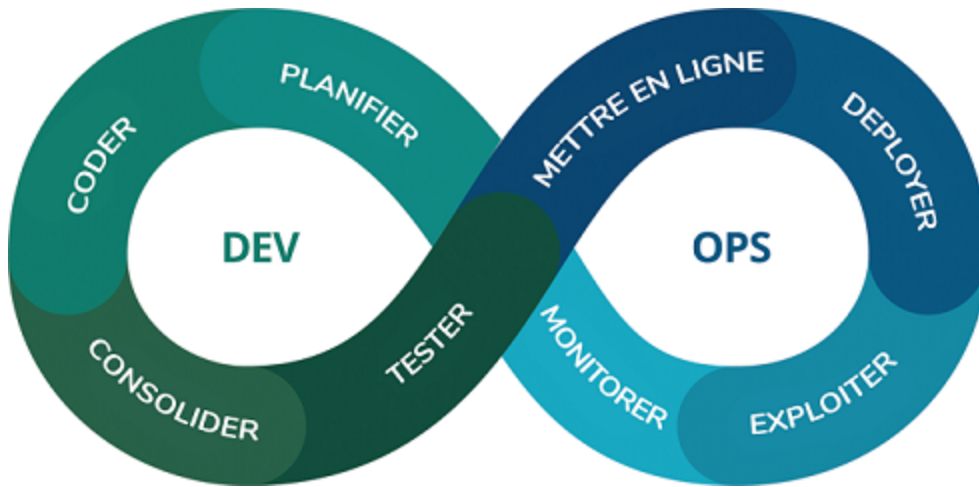


# Intégration continue (CI) et déploiement continu (CD)



Source de l'illustration : [nemesis-studio.com](https://nemesis-studio.com)

**Mise en pratique CI-CD sous Gitlab**

par **Fabien Barbaud** - [@BarbaudFabien](https://twitter.com/BarbaudFabien)

# Intégration continue

*Continuous integration*

L'**intégration continue** est un ensemble de pratiques utilisées en génie logiciel consistant à **vérifier** à chaque **modification de code source** que le résultat des modifications ne produit **pas de régression** dans l'application développée.

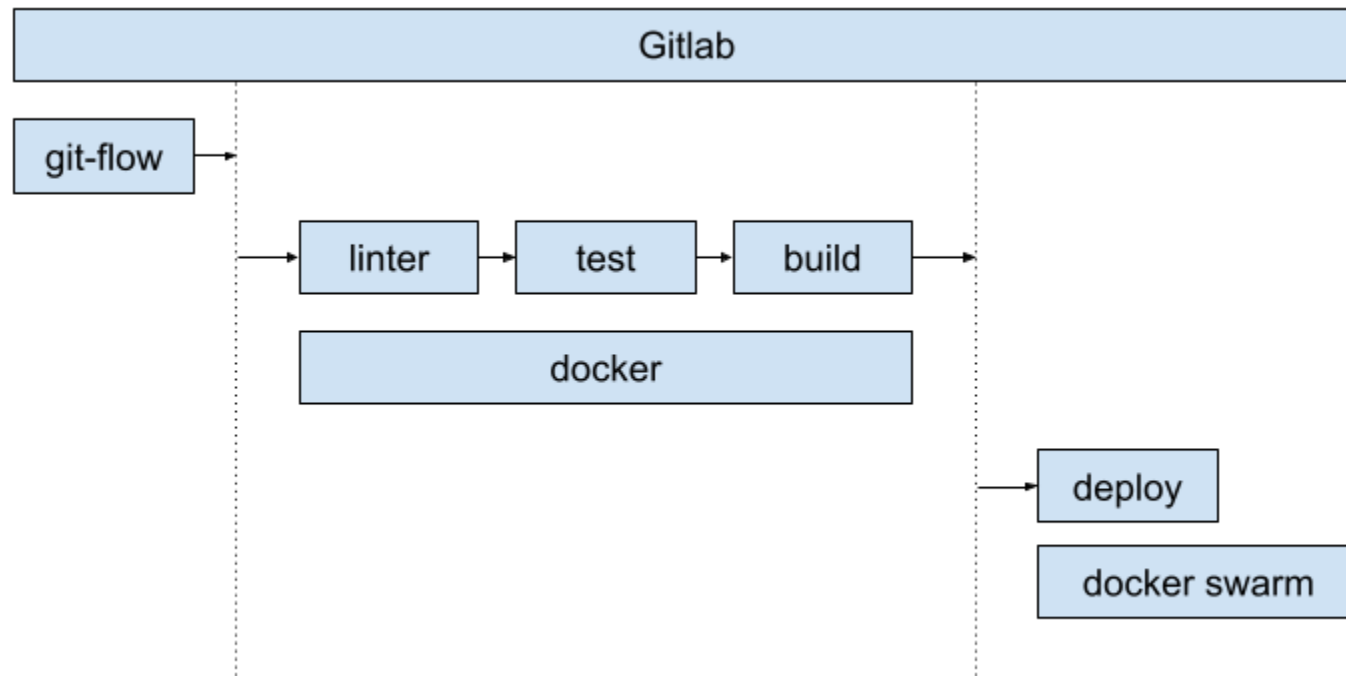
[Wikipedia](#)

# Déploiement continu

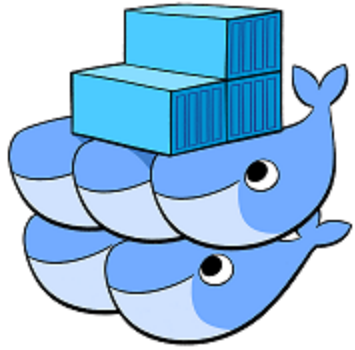
Le **déploiement continu** ou *Continuous deployment* (CD) en anglais, est une approche d'ingénierie logicielle dans laquelle les fonctionnalités logicielles sont **livrées fréquemment** par le biais de **déploiements automatisés**.

[Wikipedia](#)

# Notre objectif :



# Docker Swarm



## Orchestrateur Docker

```
$ docker swarm init
```

# Gitlab



# GitLab

**GitLab** est un **logiciel libre** de **forge** basé sur git proposant les fonctionnalités de wiki, un système de suivi des bugs, **l'intégration continue et la livraison continue**.

[Wikipedia](#)

# Gitlab

```
$ git clone https://github.com/fabienbarbaud/gitlab-docker.git  
$ cd gitlab-docker  
$ docker network create -d overlay cesi --scope swarm --attachable  
$ docker stack deploy --compose-file docker-compose.yml gitlab
```


```
$ docker service ls  
$ docker service logs --tail 50 -f gitlab_gitlab
```

<http://localhost>

u: root

p: MySuperSecretAndSecurePass0rd!

# Gitlab

 You won't be able to pull or push repositories via SSH until you add an SSH key to your profile

Add SSH key

Don't show again

<http://localhost/-/profile/keys>

<http://localhost/help/ssh/README#generate-an-ssh-key-pair>

```
$ ssh git@localhost
```



# Gitlab

- Créer un premier projet "test" - <http://localhost/root/test>
- Cloner ce projet sur votre poste
- Faire un premier commit
- Un push

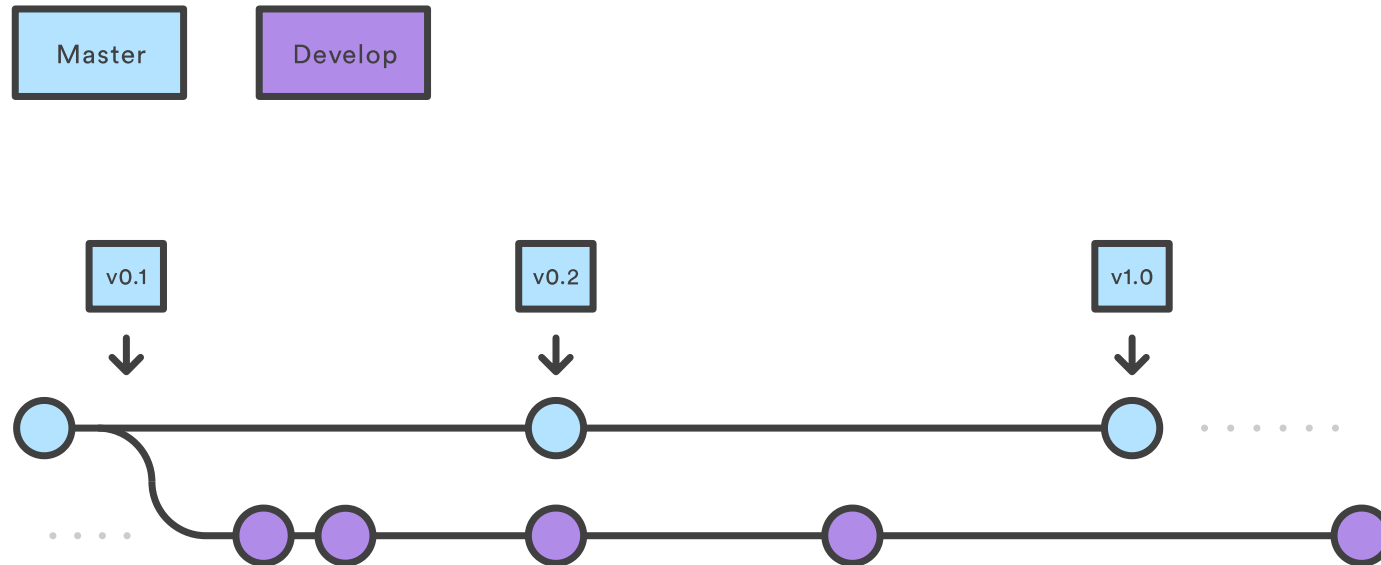
# git-flow

Le workflow Gitflow définit **un modèle de création de branche strict** conçu autour de la **livraison de projet**. Cela fournit un **framework** solide pour la gestion de projets plus importants.

[Atlassian](#)

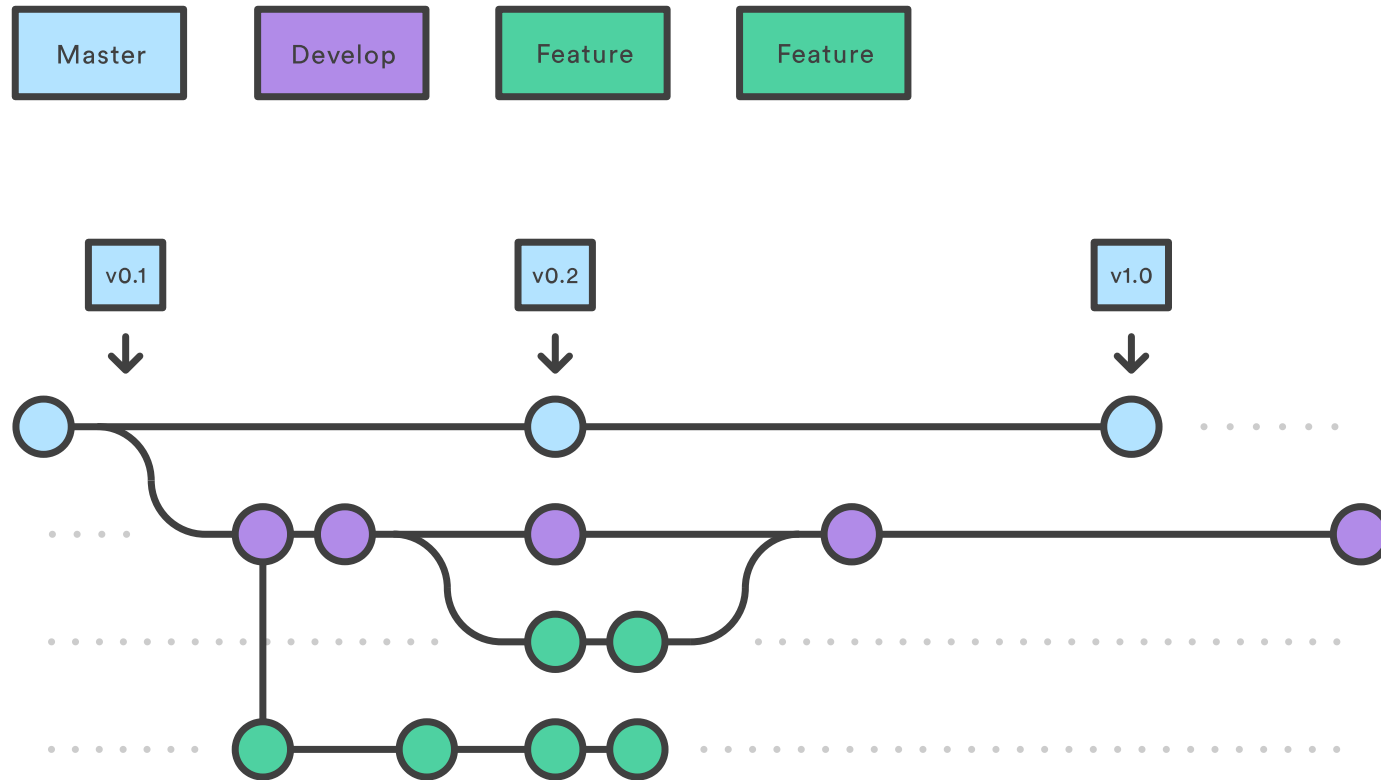
# git-flow

## Branches develop et master



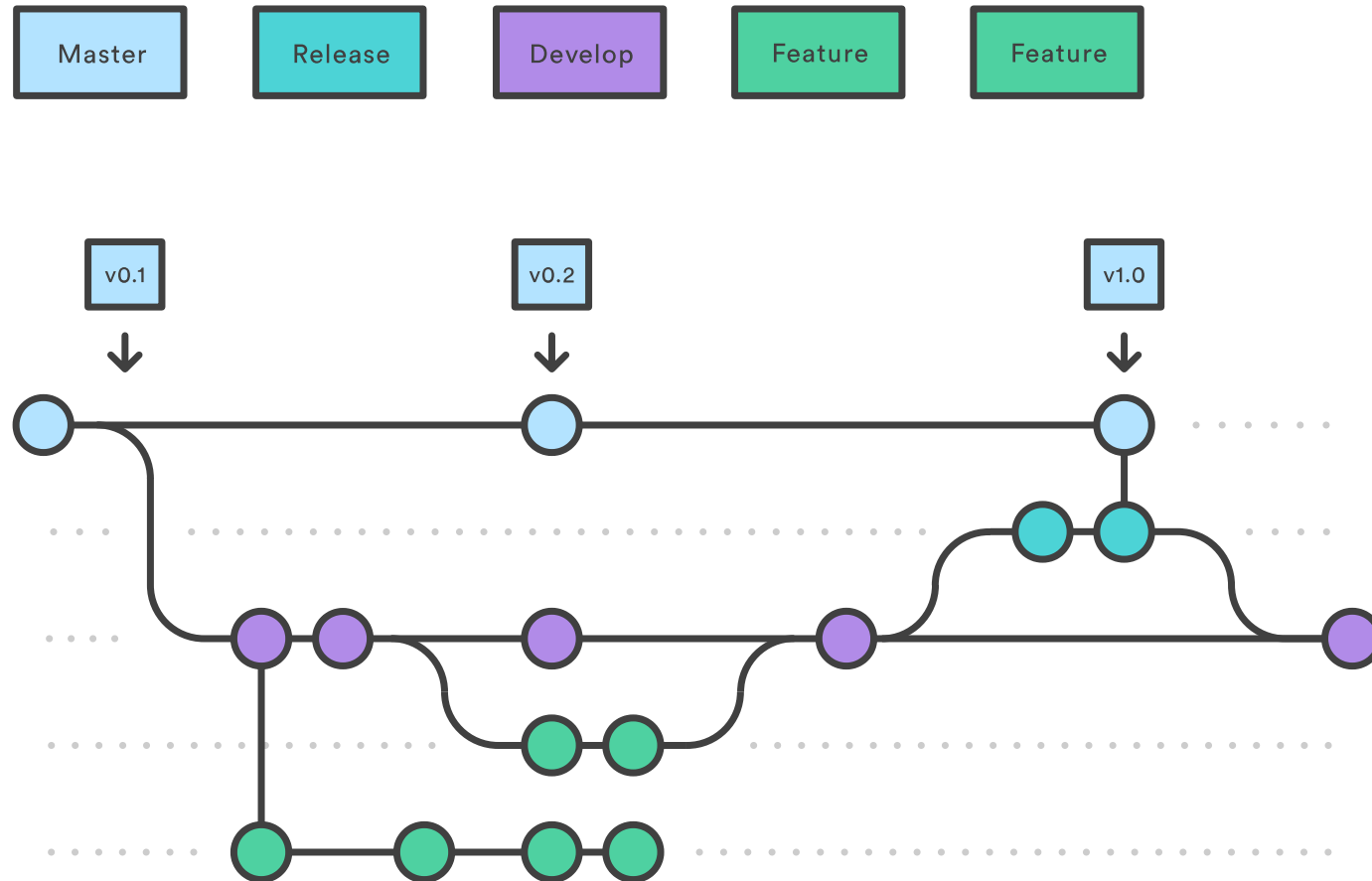
# git-flow

## Branche feature



# git-flow

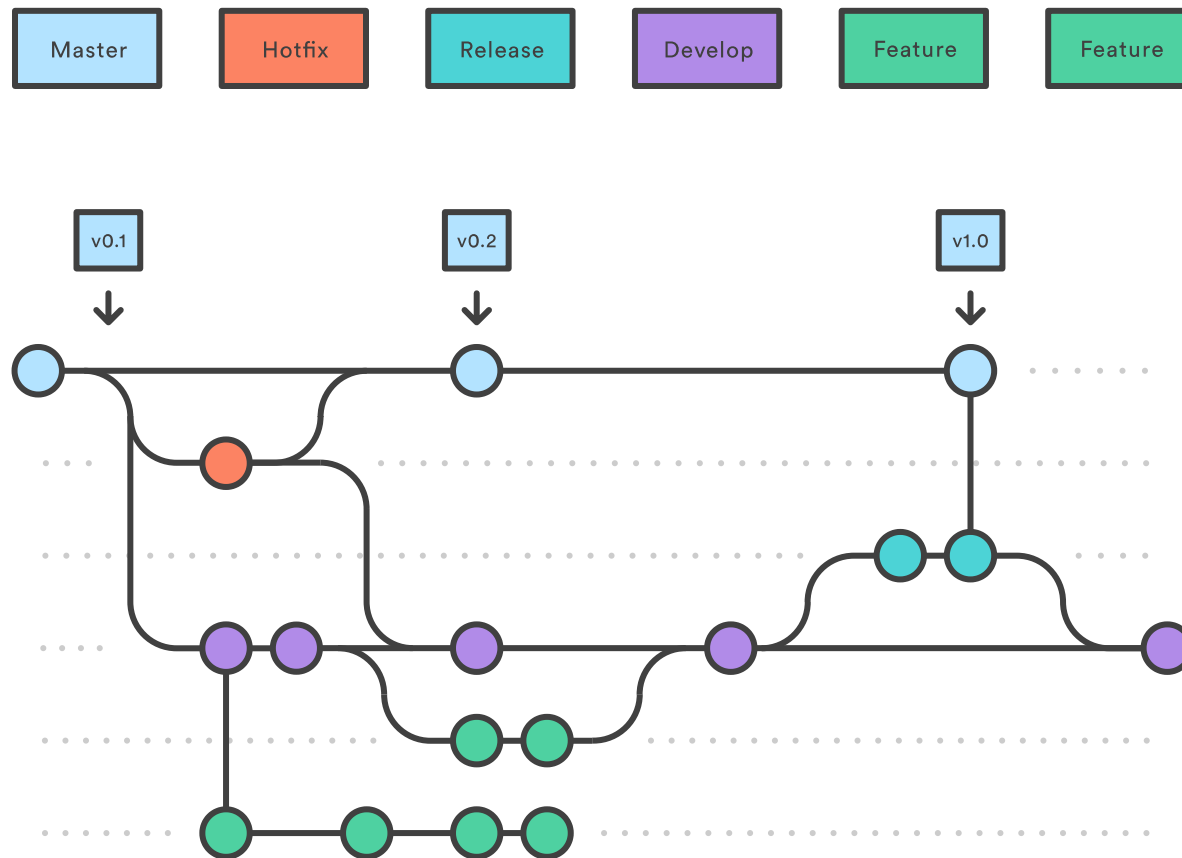
## Branche release



Intégration continue (CI) et déploiement continu (CD)

# git-flow

## Branche hotfix



Intégration continue (CI) et déploiement continu (CD)

# git-flow

## Les commandes

```
$ git flow init
$ git flow feature start ma-feature
$ git flow finish -p
$ git flow release start 0.1.0
$ git flow finish -p
$ git flow hotfix start 0.1.1
$ git flow finish -p
```

# Lint

## Outils d'analyse statique

- Javascript : ESLint
- PHP : PHPLint, PHPStan
- Python : pylint



# Test unitaire

En programmation informatique, le test unitaire (ou « T.U. », ou « U.T. » en anglais) est une procédure permettant de **vérifier** le bon **fonctionnement** d'une **partie précise** d'un logiciel ou d'une **portion d'un programme** (appelée « unité » ou « module »).

[Wikipedia](#)

# Test fonctionnel

Les tests fonctionnels sont destinés à s'assurer que, **dans le contexte d'utilisation réelle**, le comportement fonctionnel obtenu est **bien conforme** avec celui attendu.

Un test fonctionnel a donc pour objectif de **dérouler un scénario** composé d'une liste d'actions, et pour chaque action d'effectuer une **liste de vérifications** validant la conformité de l'exigence avec l'attendu.

Qu'est-ce qu'un test fonctionnel ?

# Un exemple sur un app Python : Flaskex

## L'app Flaskex

```
$ git clone https://github.com/fabienbarbaud/flaskex  
$ cd flaskex  
$ docker-compose up app
```

<http://localhost:5000>

# Un exemple sur un app Python : Flaskex

## Le linter

```
$ docker-compose up linter
```

# Un exemple sur un app Python : Flaskex

## Les tests

```
$ docker-compose up test
```

# Selenium

**Selenium automates browsers. That's it!**



# Selenium

<https://www.selenium.dev/>

# Selenium

## Selenium IDE

Open source record and playback test automation for the web

<https://www.selenium.dev/selenium-ide/>

# CI-CD

## Gitlab runner

```
$ docker ps
$ docker exec -it gitlab_gitlab-runner.1.3hdx4ch6guq5c88xpmed9yk0s bash
bash-5.0# gitlab-runner \
  register -n \
  --name "Docker Runner" \
  --executor docker \
  --docker-image docker:latest \
  --docker-volumes /var/run/docker.sock:/var/run/docker.sock \
  --url http://gitlab_gitlab \
  --clone-url http://gitlab_gitlab \
  --docker-privileged \
  --docker-network-mode cesi \
  --registration-token z5xxGJZZpzJp5Wz_s4h3
```



# CI-CD

## .gitlab-ci.yml

```
image: docker:latest

stages:
  - build
  - linter
  - test

build:
  stage: build
  script:
    - docker build -t image .

linter:
  stage: linter
  script:
    - docker run image bash docker/start_linter.sh
```

# CI-CD

## Démarrage du service

```
$ docker build -f Dockerfile_prod -t root/flaskex:latest .  
$ docker service create -p 5000:5000 --name flaskex root/flaskex:latest
```

# CI-CD

## Déploiement

```
deploy:
  stage: deploy
  only:
    - tags
  script:
    - docker build -f Dockerfile_prod -t $CI_PROJECT_PATH:$CI_COMMIT_REF_NAME .
    - docker service update --image $CI_PROJECT_PATH:$CI_COMMIT_REF_NAME flaskex
```

# TP

Mettre en place une application de votre choix dans le langage de votre choix comprenant :

- un linter
- des tests unitaires
- un déploiement

Tout ça sur un projet en CI-CD avec Gitlab et Docker Swarm