Automatiser sa gestion de version sur des packages python

- 1/ Pourquoi faire des versions
- 2/ Comment mettre un place les versions
- 3/ Automatiser le traitement

1/ Pourquoi faire des versions

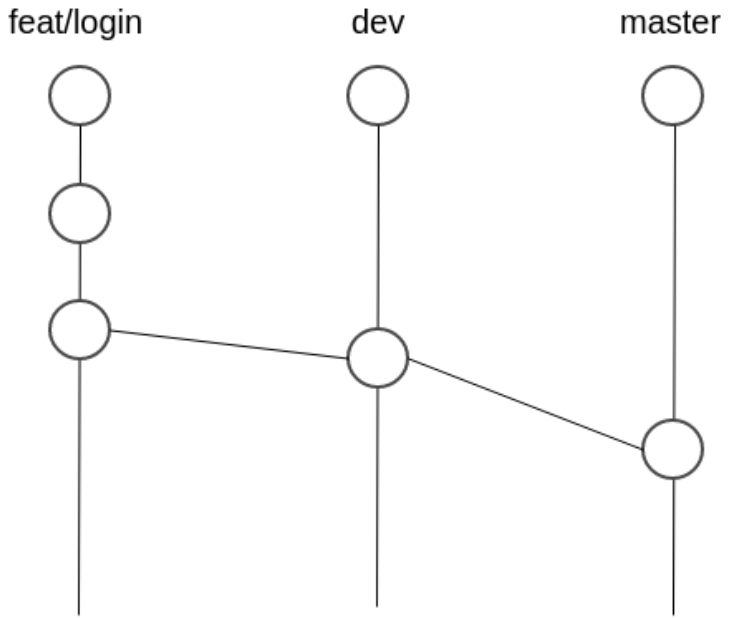
- Fige un état de l'application à un moment donnée
- Permet de faire un retour arrière plus simplement
- On peut lister les évolutions sur cette version

Conventions sur le nom des versions

- SemVer : `x.y.z-opt`
- `x` est la version majeure : fonctionnalité obsolète, renommage, changement structurelle ...
- `y` est la version mineure : fonctionnalité nouvelle, warning sur les méthodes à déprécier...
- `z` est la version corrective : faille sécu, optimisation, bug...
- `opt` est un label de pre-release : alpha, beta ...

Les branches GIT

- Branches éphémères : `feat/<name_feat>`
- Branche protégée `dev` : recueille les demandes de MR
- Branche protégée `master` : version stable



Les différentes étapes

2/ Comment mettre en place les versions

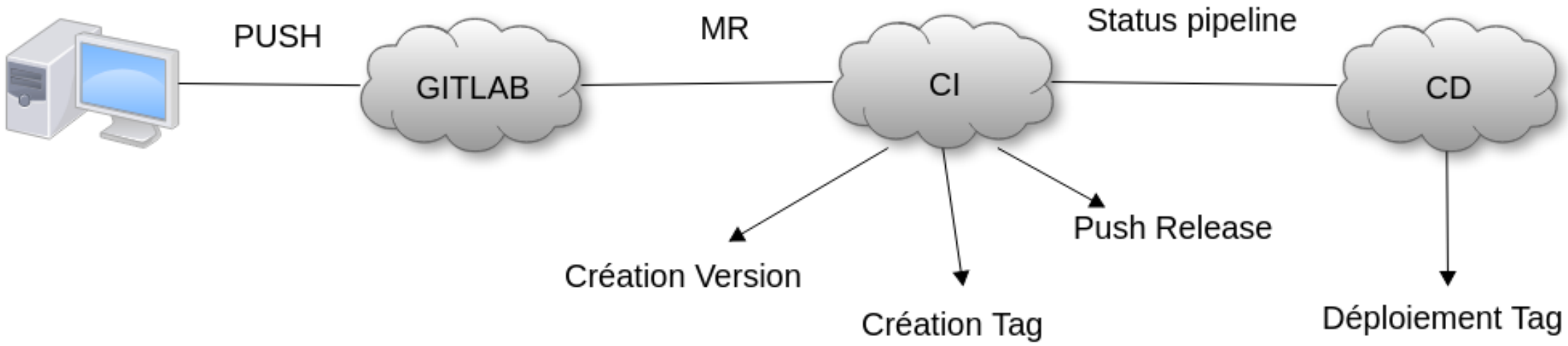
Outil de génération

- Librairie `python-semantic-release` python
- Code convention d'Angular : https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#commits
- Repose sur le contenu des messages de commit

  `git commit -m "feat: Add login route"`
- Automatisation (gestion version app, génération CHANGELOG, création des tags GIT, publication de release...)

  ⚠️ breaking change >= 8

Les différentes étapes avec `semantic-release`



Le code avec `python-semantic-release`

Installation

- Uniquement pour les dépendances de développement
- `pipenv install python-semantic-release --dev`
- Usage : `pipenv run python-semantic-release`
- Nécessite d'installer git (dans docker pour la CI)

Stockage de la version `setup.py`

- Fichier à la racine du projet pour build un package

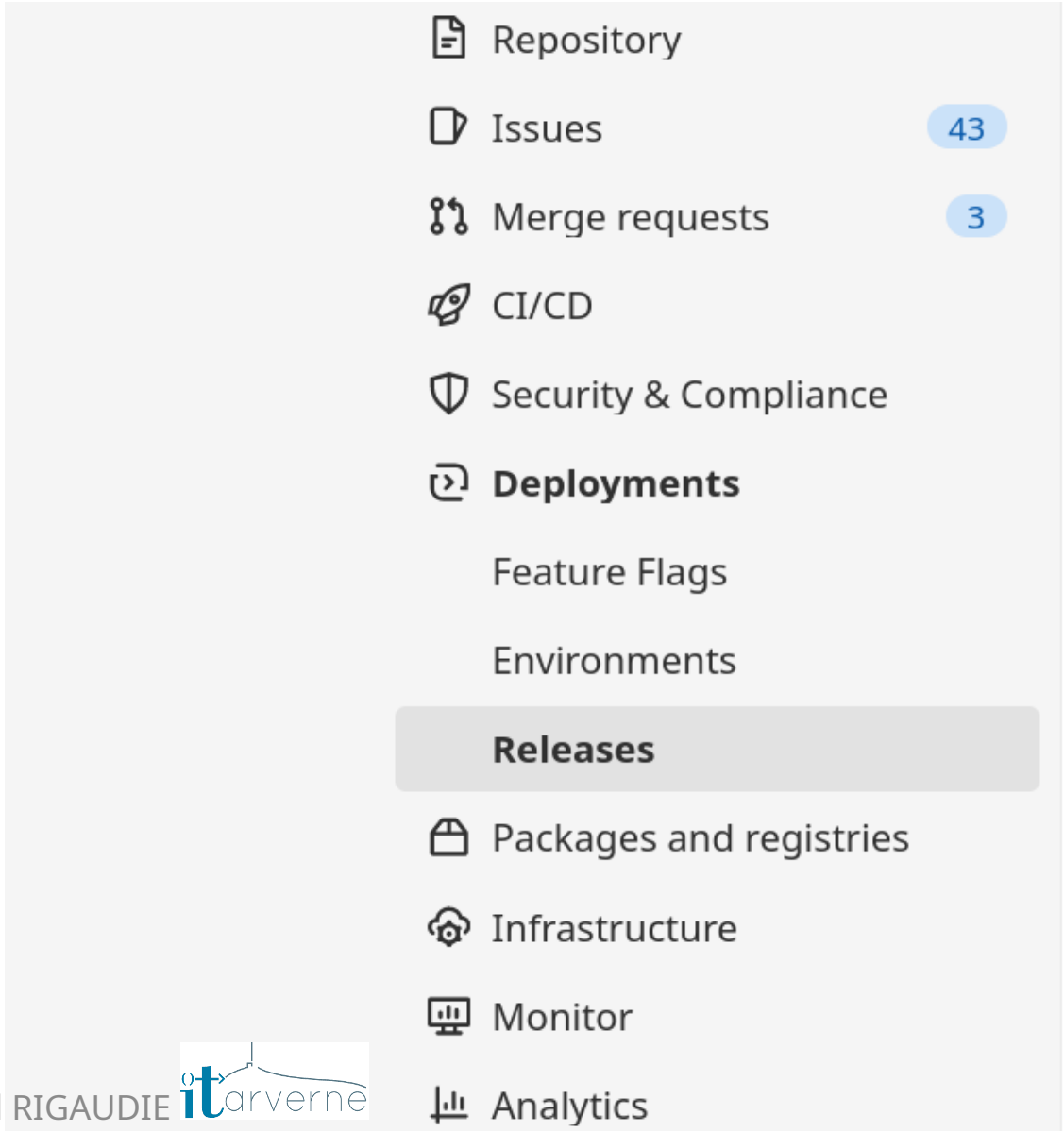
```
from setuptools import setup

__version__ = "0.3.8"

if __name__ == "__main__":
    setup(
        name="Blog",
        version=__version__,
        author="ITARVERNE",
        author_email="david@itarverne.fr"
    )
```

Configuration

- Dans `pyproject.toml`

```
[tool.semantic_release]

version_variable='setup.py:__version__'

hvcs='gitlab'
hvcs_domain='gitlab.grenoble.com'

# Mettre à jour le repo avec le CHANGELOG et la nouvelle version
repository="gitlab"
git_committer_name="david"
git_committer_email="david@itarverne.fr"
commit_message="See the CHANGELOG.md content file for more details"

# Publie la release dans Gitlab
upload_to_repository=true # Artifact
upload_to_release=true # Release Gitlab
repository_url="https://gitlab.grenoble.com/api/v4/projects/<id>/packages/pypi"
```

Release Gitlab



Release 0.3.8

Assets 4

- Source code (zip)
- Source code (tar.gz)
- Source code (tar.bz2)
- Source code (tar)

Evidence collection

v0.3.8-evidences-13.json ••• 284d437

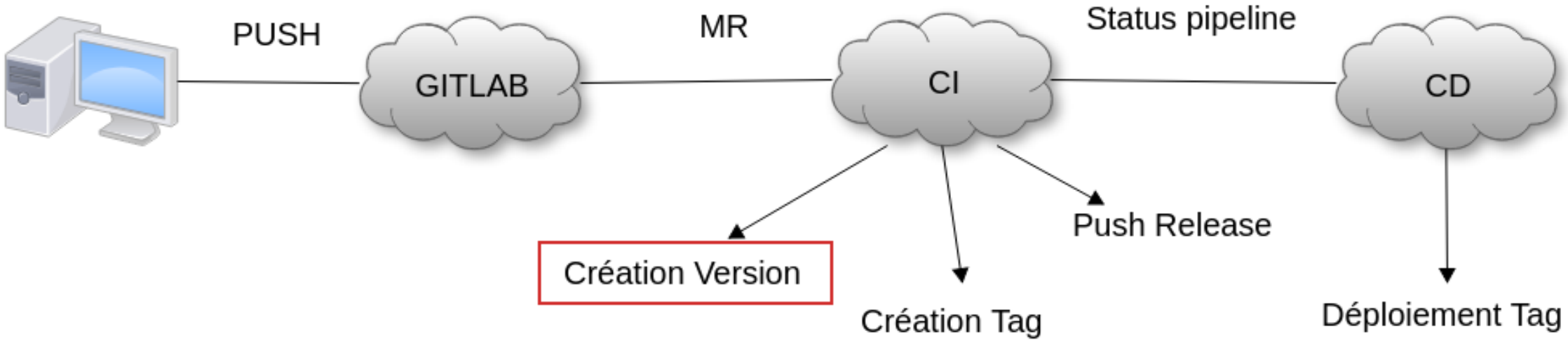
Collected 7 hours ago

Fix

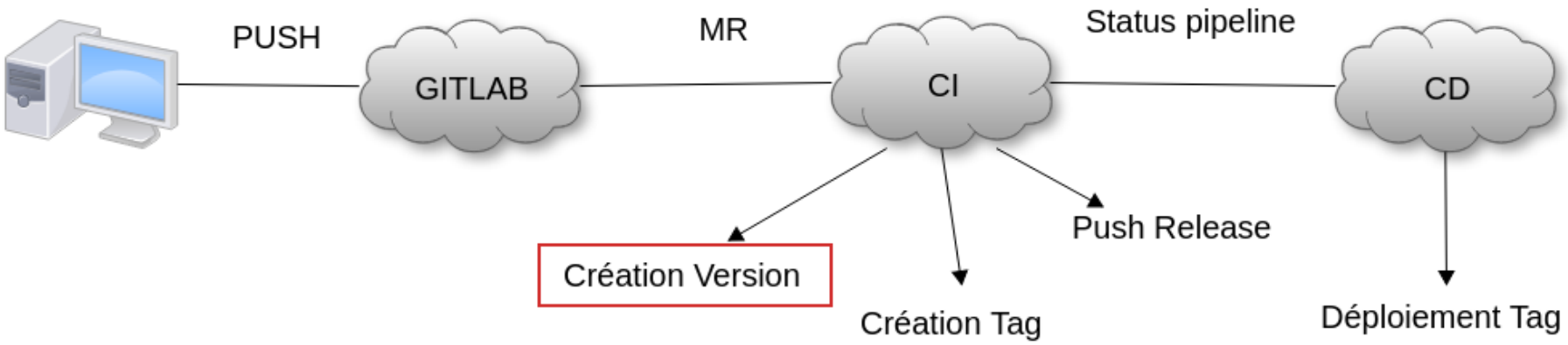
- Split playbook (`b2ba94c`)
- Split playbook (`ead6ced`)

Configuration d'accès au code par la CI

- Dans les préférences utilisateur > Token Access (api and read_repository)



Création de la version puis tag

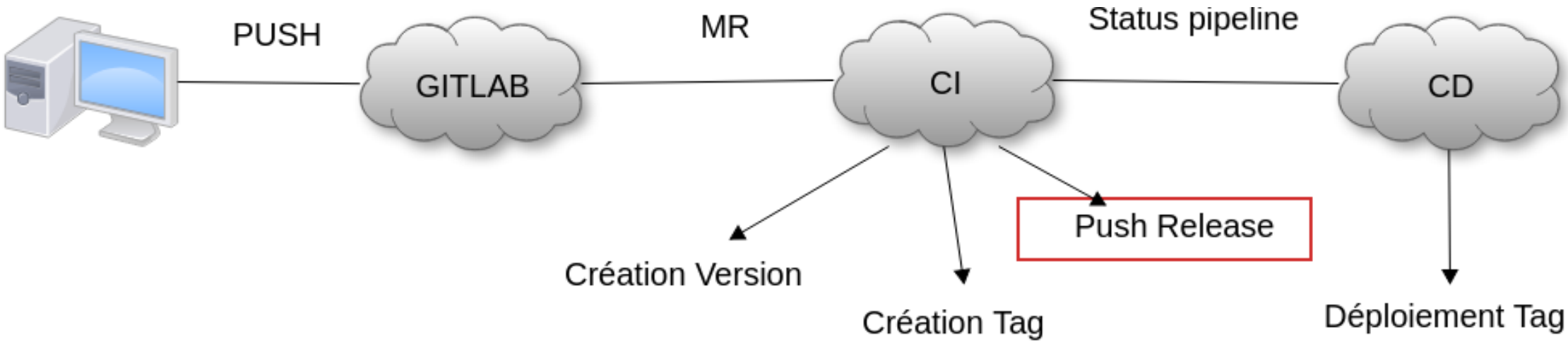



```
pipenv run semantic-release publish -v DEBUG
```

- met à jour le CHANGELOG.md
- incremente la version dans le setup.py
- pousse le code sur le repo gitlab
- build le projet (ne pousse pas sous PYPI)

Release

- Par défaut sur pypi
- On peut surcharger pour le mettre dans Artifactory par exemple



Configuration d'accès à l'artifact par la CI

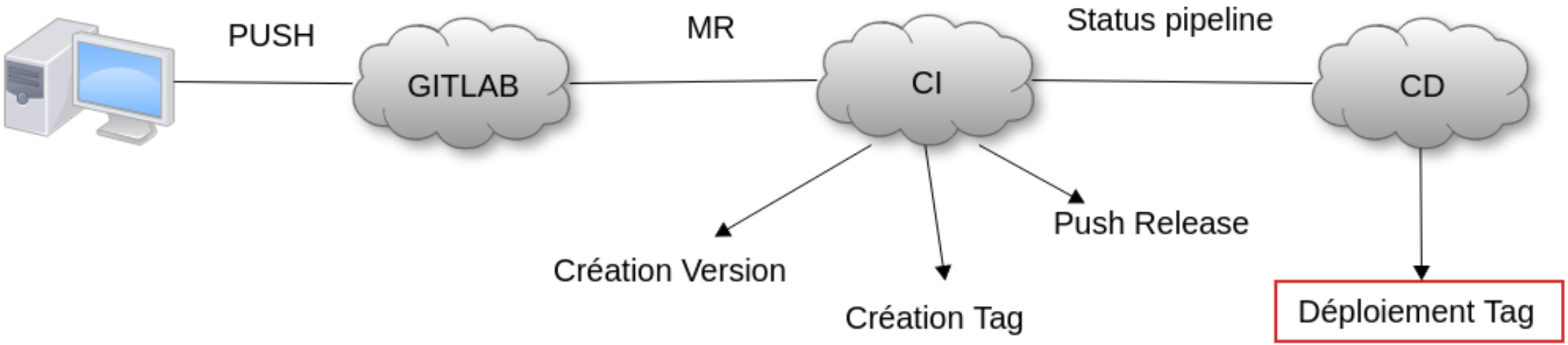
- Dans `.pypirc`

```
[distutils]
index-servers =
    gitlab

[gitlab]
repository = https://gitlab.grenoble.com/api/v4/projects/python/packages/pypi
username = gitlab-ci-token
password = ${env.CI_JOB_TOKEN}
```

- `CI_JOB_TOKEN` est une variable prédéfinie par Gitlab lors de l'exécution du job (même droit que l'utilisateur qui lance la tâche)

Déploiement

- Ansible



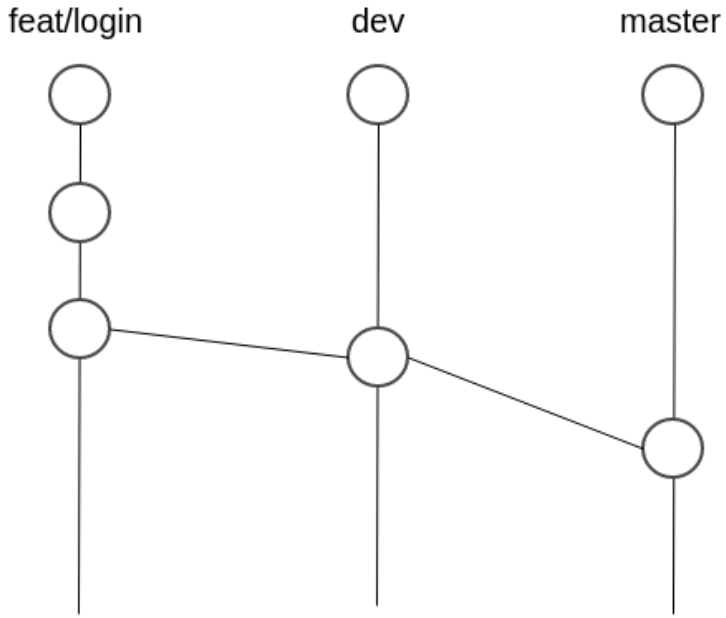
Extrait du Playbook

```
- name: Get last application version
    command: |
    python3 -c "import sys; sys.path.insert(0,'{{ path_web }}');from setup import __version__;print(__version__)"
    register: version_app

- name: Git checkout last tag
    git:
    repo: 'git@gitlab.grenoble.com:python/blog.git'
    dest: "{{ path_web }}"
    version: "v{{ version_app.stdout }}"
    accept_hostkey: yes
    force: yes
    key_file: "{{ path_gitlab_key }}"
```

3/ Automatiser le traitement

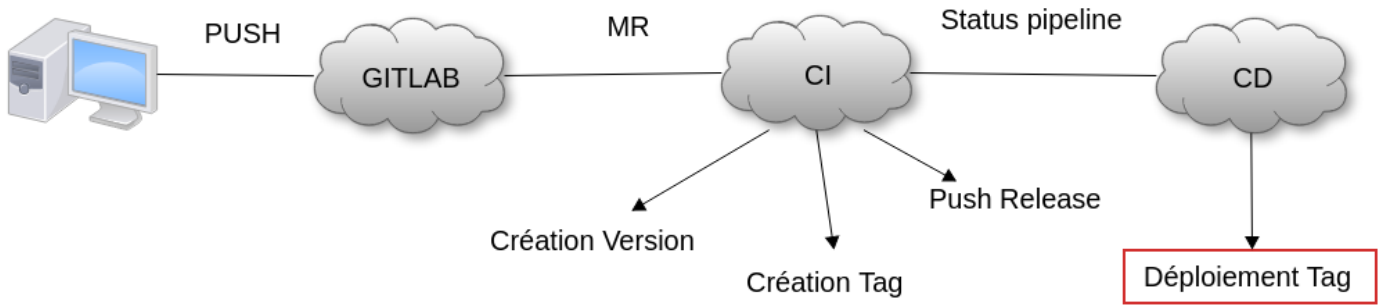
Orchectrer les différentes étapes

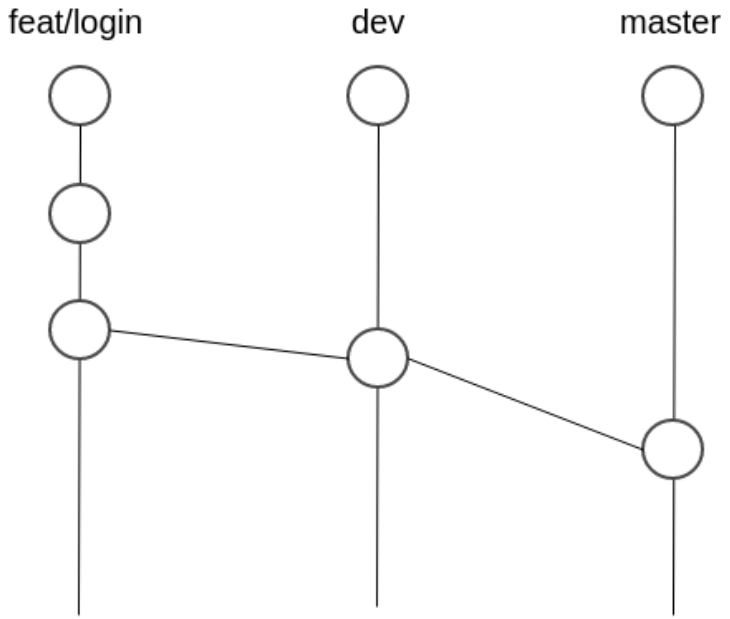
- stage `version` dans `.gitlab-ci.yml`
- lance `python-semantic-release` sur la branche `dev`
- checkout de `master` pour création du tag
- push code de `master` dans `dev`



Orchectrer les différentes étapes

- stage `production` dans `.gitlab-ci.yml`





- lance playbook ansible
- checkout depuis le tag

Forcer les messages

- /!\ Si les messages ne sont pas corrects
- Hook `commit-msg`

```
commit_regex_flag='feat|fix|docs|style|perf|test|refactor'
if ! grep -qE "$commit_regex_flag" "$1"; then
    printf "${RED}Please add flag front of message (fix, feat, docs, style, perf, test or refactor) before : caracter \n"
    printf "${RED}Ex => feat: worflow email updated\n"
    printf "${RED}Abort commit !%s\n"
    exit 1
fi
```

Evolutions

- Synchro Master -> Dev
- Rejouer un ancien tag via CD
- Lors des MR attention au squash

David RIGAUDIE

Senior tech lead / Python lover

- 16 ans XP en dev
- Python 3
- React
- Blog technique
  http://rigaudie.fr
- Evènements
  https://pyclermont.org