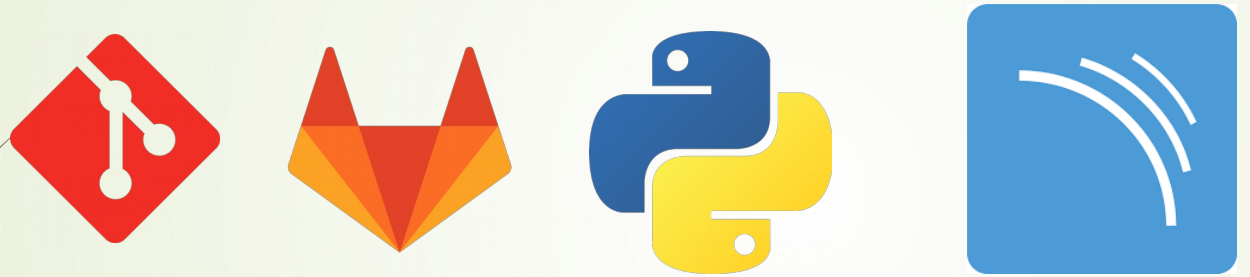
DAWAN Paris 11,rue Antoine Bourdelle, 75015 PARIS
DAWAN Nantes 32, Bd Vincent Gâche, 5e étage - 44200 NANTES
DAWAN Lyon Bt Banque Rhône Alpes, 2ème étage – 235 cours Lafayette 69006 LYON



Intégration Continue: Analyse statique du code

Formateur: Matthieu LAMAMRA

Linter PYHTON

- Le module pylint : analyse statique de code source
  - Intérêt: évaluation de la qualité du code source python
  - Installation : **sudo pip3 install pylint** (à l'intérieur du venv)
  - Détecte et liste les **erreurs de syntaxe** et les entorses aux conventions **pep8**
  - Propose des tableaux de synthèse et assigne une note au code

```
$ pylint [fichier]

$ pylint -E [fichier] #n'affiche que les erreurs

$ pylint --reports=y [fichier] #ajoute le rapport de synthèse
```

https://docs.pylint.org/en/1.6.0/features.html

Linter PYHTON

- Fichier .pylintrc : configuration du linter
  - Permet de régler le niveau et la précision de l'analyse
  - Permet d'ajuster certaines conventions d'écritures (camelCase, snake_case…)
  - Permet de limiter les messages d'informations à certaines règles

```
$ pylint --generate-rcfile > .pylintrc #création du fichier

[…]
method-naming-style=snake_case
[...]
disable=print-statement,...
```

Linter PYHTON

- Analyse syntaxique dans gitlab
  - On fait exécuter pylint -E sur l'ensemble des fichiers python du dépôt

```
pylint-E:
  stage: syntax
  before_script:
    - pip3 install -r $CI_PROJECT_DIR/requirements.txt
  script:
    - 'find . -type f -name "*.py" -not -path "./lib/*" | xargs pylint -E'
```

Linter PYHTON

- Analyse de la qualité du code dans gitlab

- On veut récupérer le score de pylint pour l'afficher sur un badge dans gitlab

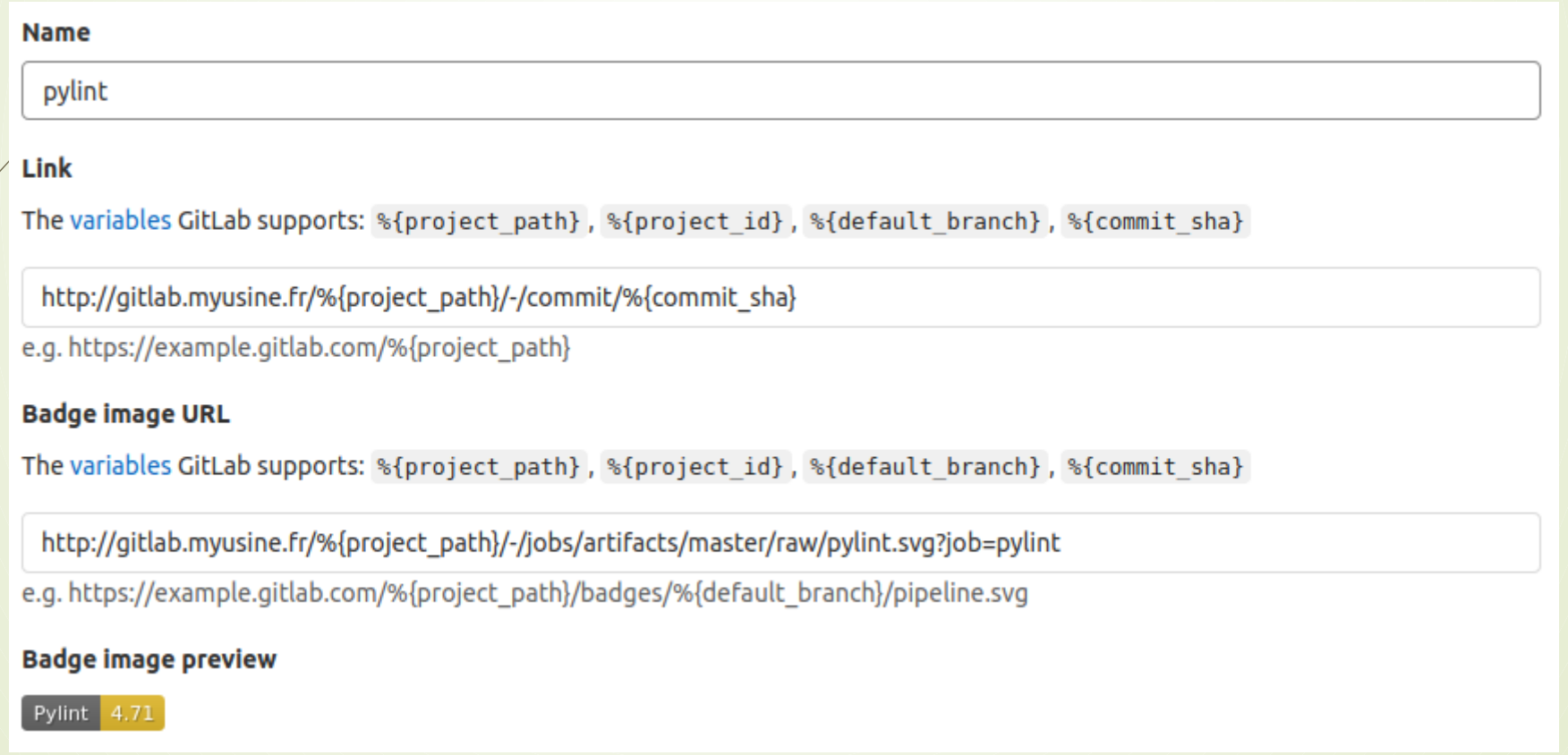
```
pylint:
  stage: style
  before_script:
    - pip3 install -r $CI_PROJECT_DIR/requirements.txt
  script:
    - >
      find . -type f -name "*.py" -not -path "./lib/*" |
      xargs pylint --output-format=text --exit-zero | tee pylint.txt
    - score=$(sed -n 's/^Your code has been rated at \([-0-9.]*\)\/.*/\1/p' pylint.txt)
    - >
      anybadge --label=Pylint
      --file=pylint.svg
      --value=$score
      2=red 4=orange 8=yellow 10=green
  artifacts:
    paths:
      - pylint.svg
```

Linter PYHTON

- Création d'un badge dans Gitlab CI

- On renseigne l'url de l'artefact dans Settings → General → Badges

- Ce badge s 'affiche sur la page d'accueil du projet



Linter PYHTON

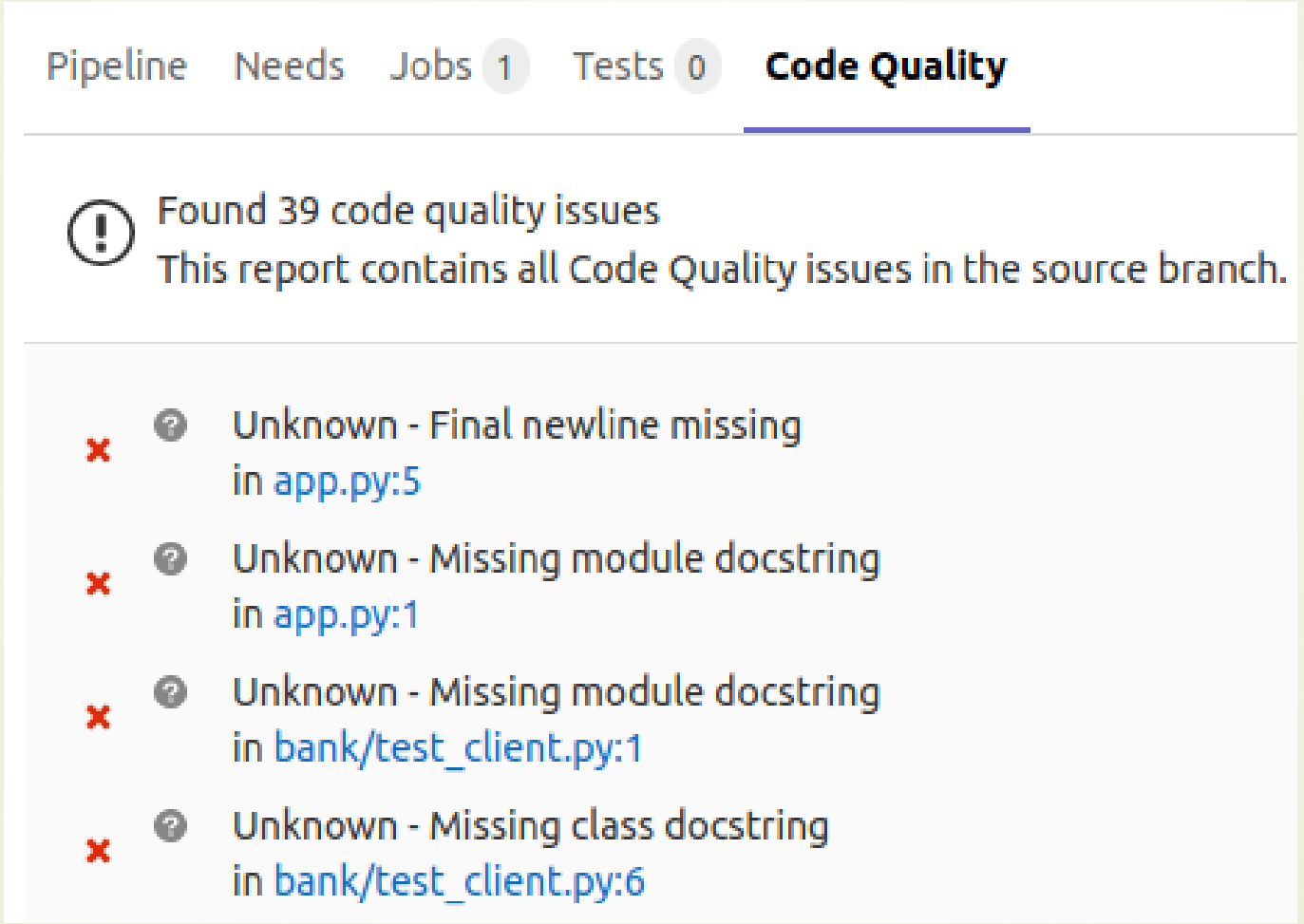
- Remontée du rapport pylint

- Nécessite l'installation du module python pylint-gitlab pour générer un rapport JSON au format « Code Quality »

```
pylint:
  stage: style
  before_script:
    - pip3 install -r $CI_PROJECT_DIR/requirements.txt
  script:
    - >
      find . -type f -name "*.py" -not -path "./lib/*" |
      xargs pylint –output-format=pylint_gitlab.GitlabCodeClimateReporter --exit-zero
      > codeclimate.json
  artifacts:
    paths:
      - codecimate.json
    reports :
      codequality : codeclimate.json
  rules :
    - if: '$CI_COMMIT_BRANCH == "master"'
    - if: '$CI_PIPELINE_SOURCE == "merge_request_event"'
```

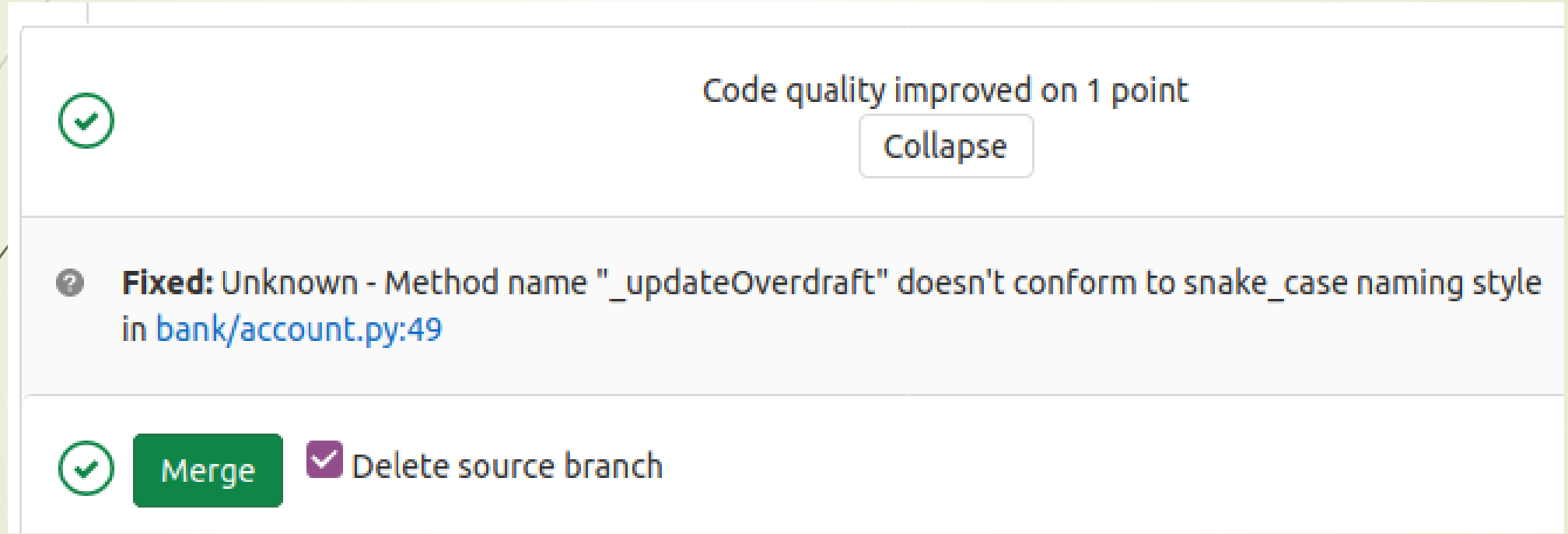
Linter PYHTON

- Affichage du rapport pylint dans le job



Linter PYHTON

- Affichage de l'évolution de la qualité dans les « merge requests »



SONARQUBE

- Solution d’analyse qualité universelle
  - Intérêt: centraliser l’outil d’assurance qualité pour tous les langages utilisés
  - Solution open source, gratuite (community edition)
  - Permet de construire des profiles d’analyses qualité
  - Permet de renseigner des seuils de validation basés sur des métriques

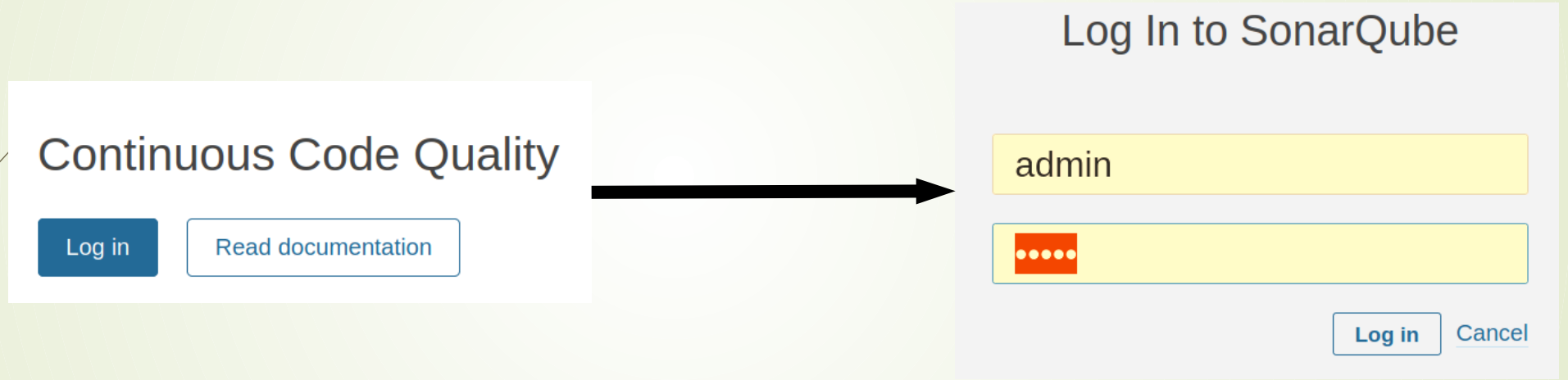
```
$ sudo docker pull sonarqube:lts

$ sudo docker container run -d --name sonar --restart always \

> -p 9000:9000 sonarqube:lts
```

SONARQUBE : installation

- Création du projet

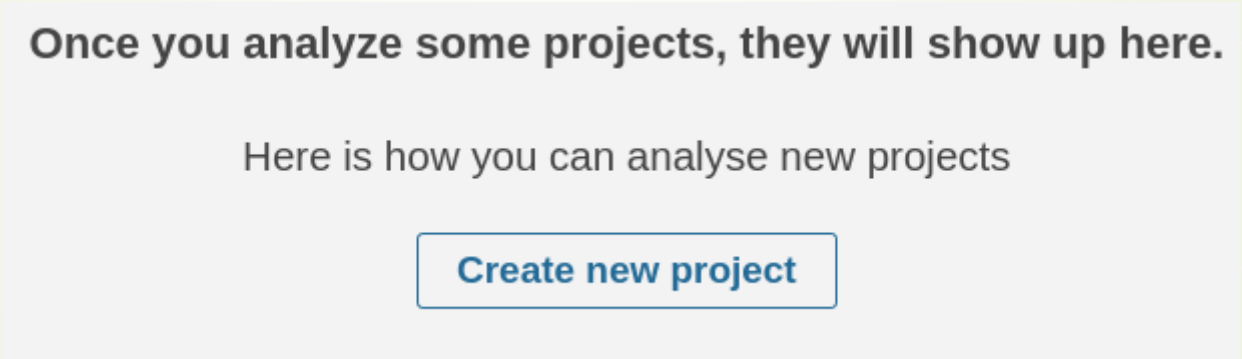
http://localhost:9000

admin / admin



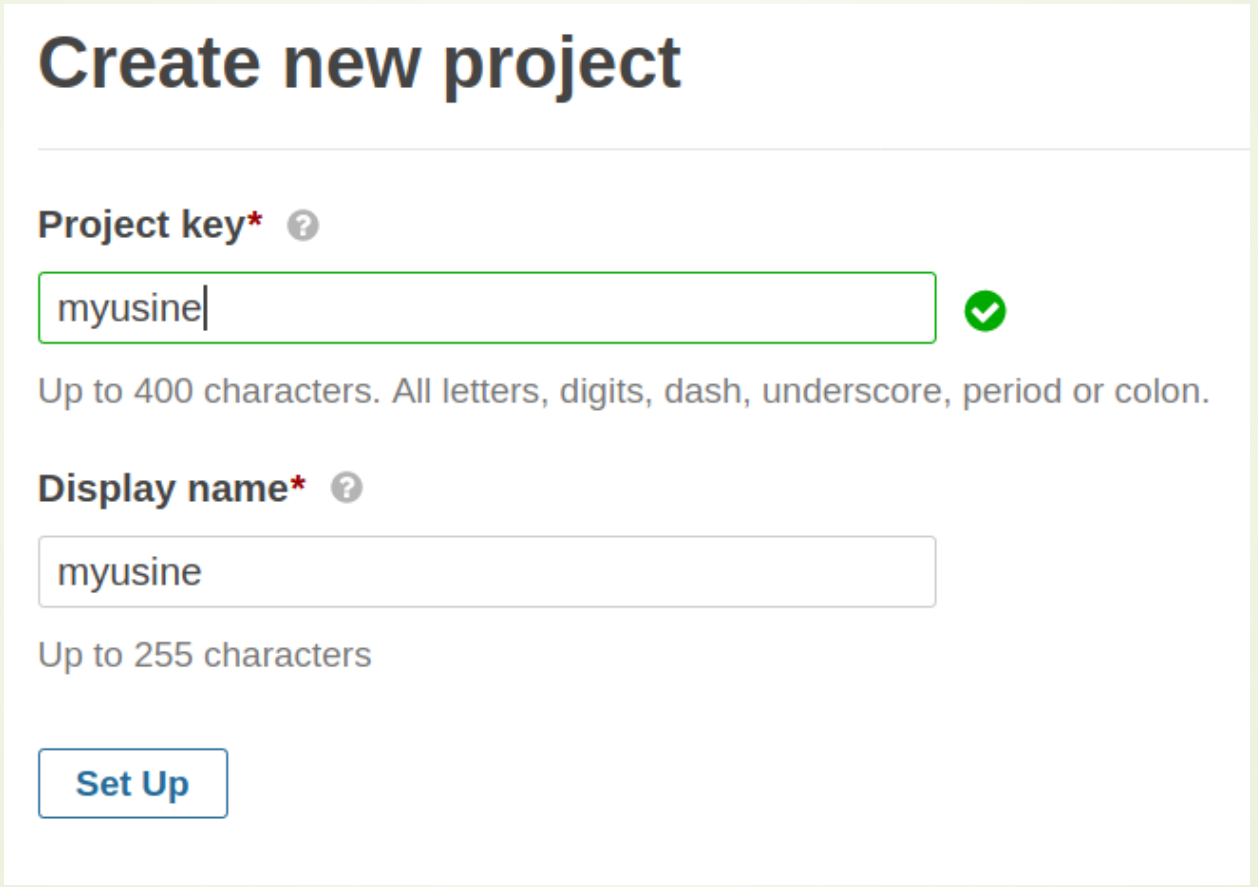
SONARQUBE : installation

- Création du projet



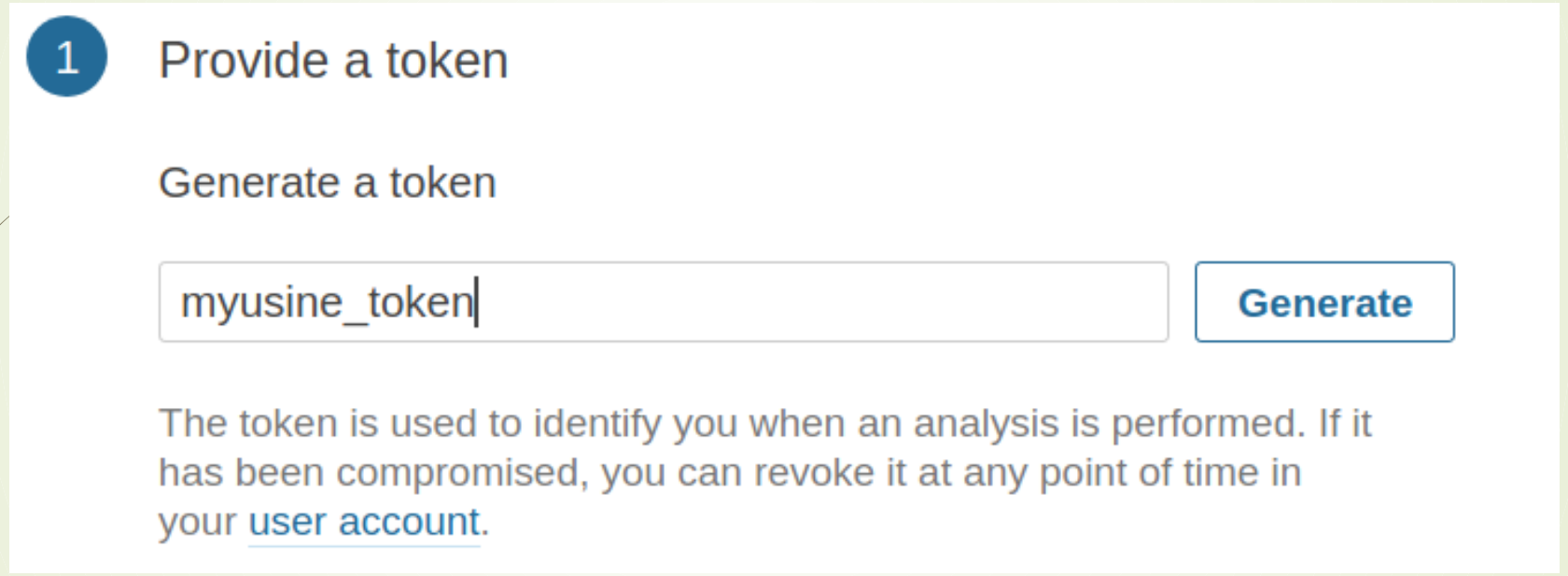
SONARQUBE : installation

- Création du projet



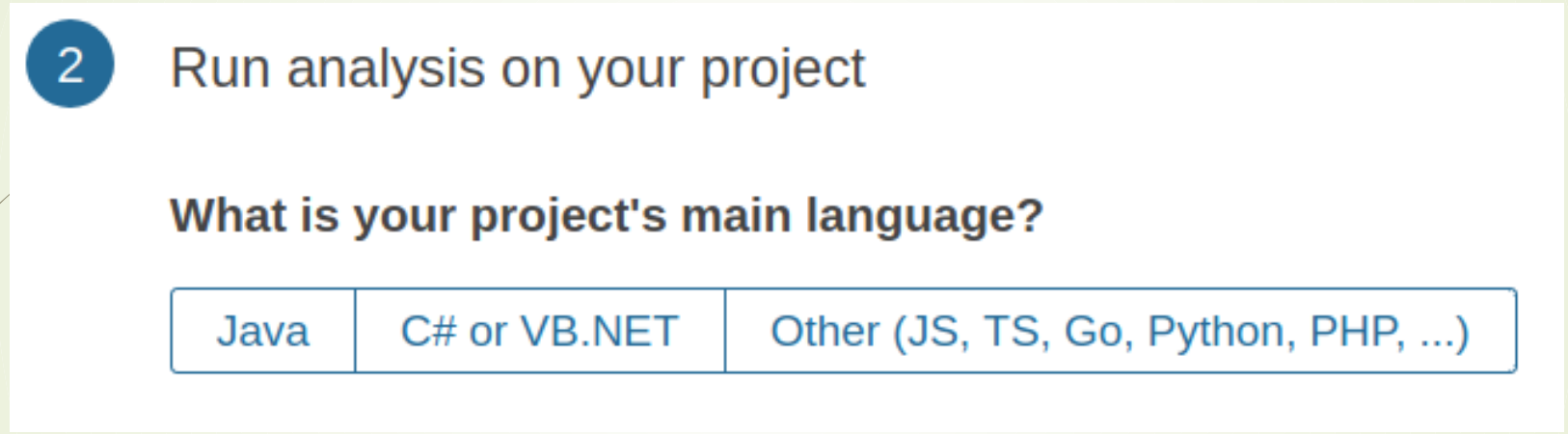
SONARQUBE : installation

- Création du projet



SONARQUBE : installation

- Création du projet



SONARQUBE : installation

- Création du projet

Download and unzip the Scanner for Linux

And add the `bin` directory to the `PATH` environment variable

Download

Execute the Scanner from your computer

Running a SonarQube analysis is straighforward. You just need to execute the following commands in your project's folder.

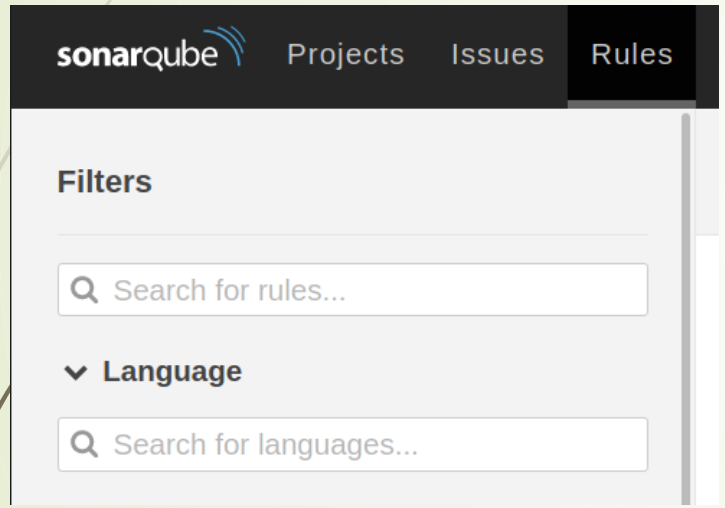
```
sonar-scanner \
  -Dsonar.projectKey=myusine \
  -Dsonar.sources=. \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.login=5aa127a39fba7b9702217c9479c6609f972cfc94
```

Copy

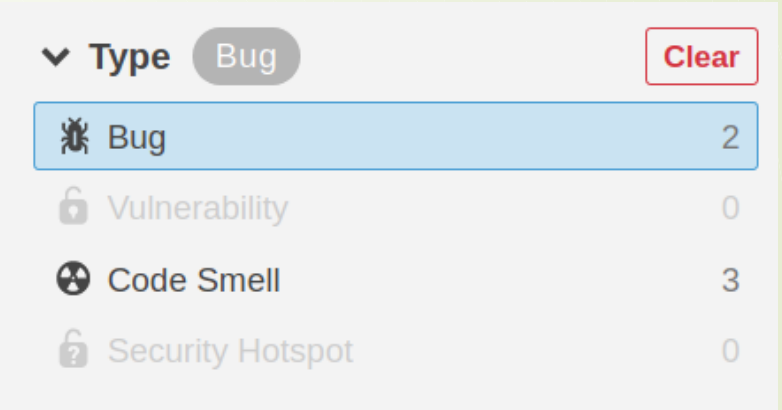
SONARQUBE : installation

- La base de règles

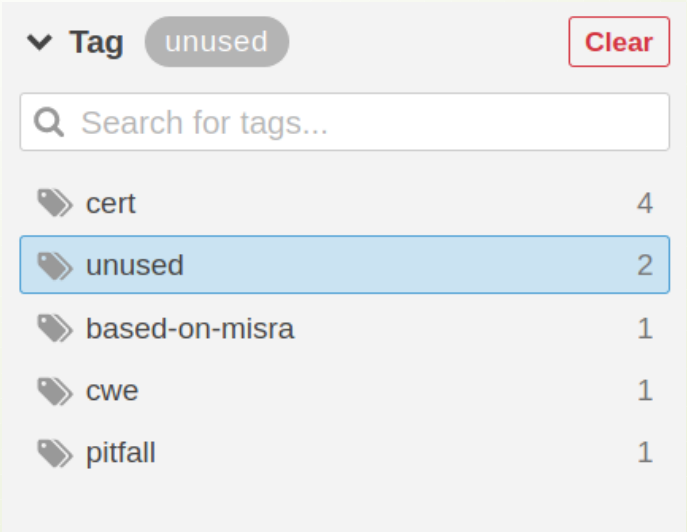
Recherche libre et par langage



Recherche par type

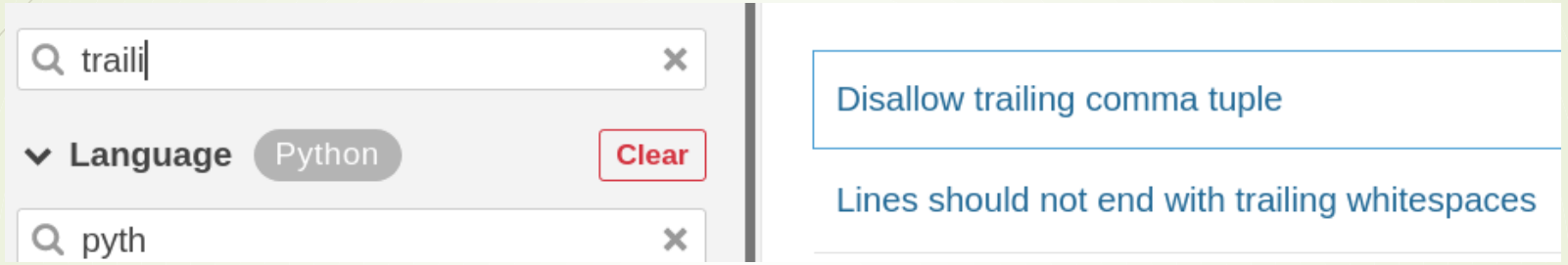


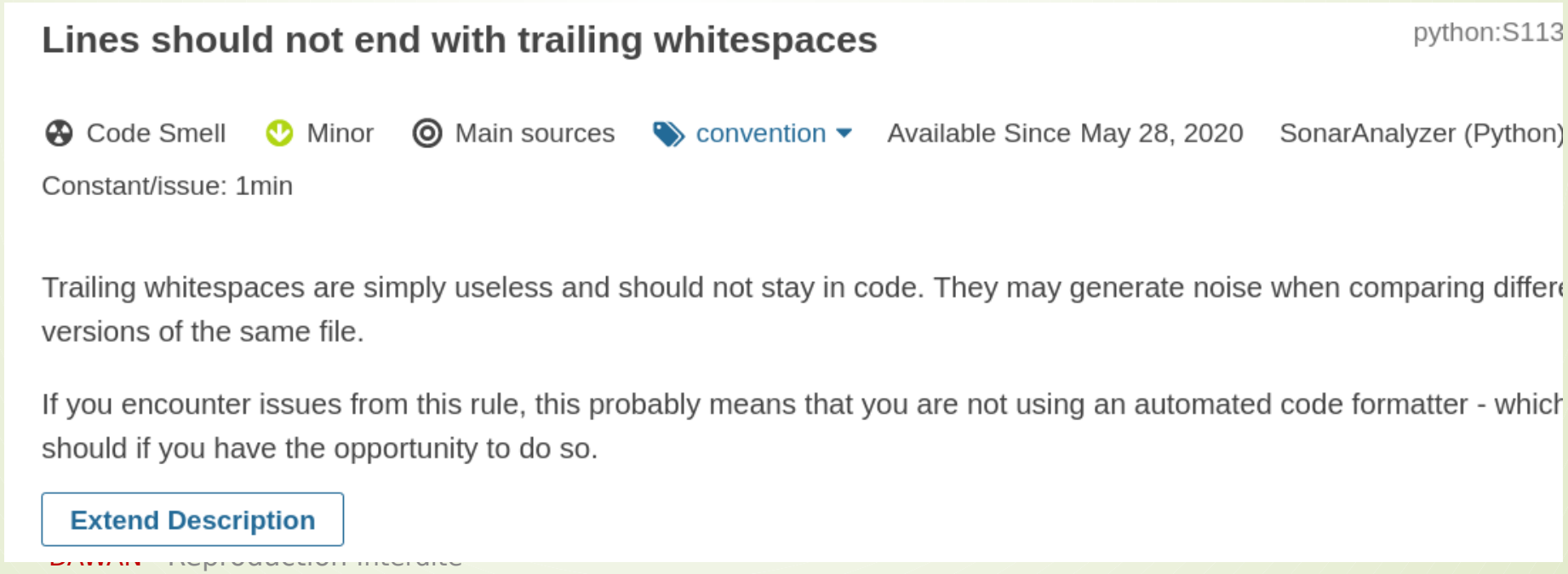
Recherche par mot clé



SONARQUBE : installation

- La base de règles : exemple



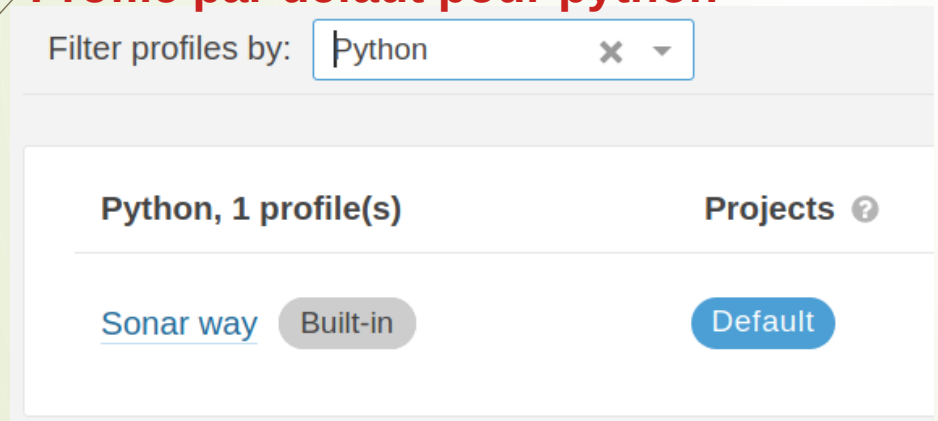


SONARQUBE : installation

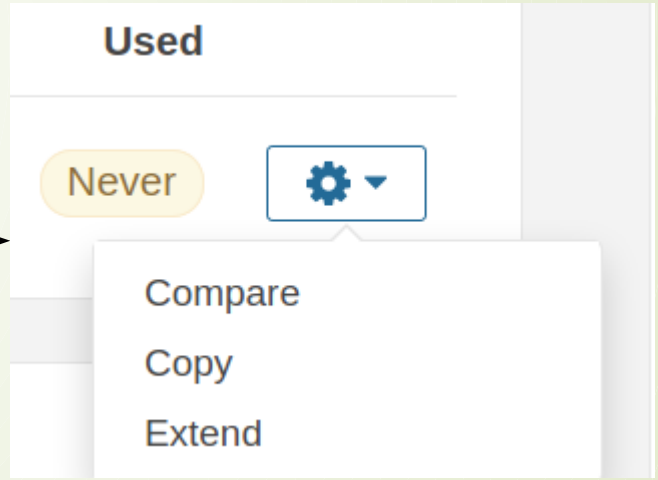
- Création du Profile qualité

Profile : ensemble de règles à appliquer

Profile par défaut pour python



Extend:créer un nouveau profile à partir du profile par défaut

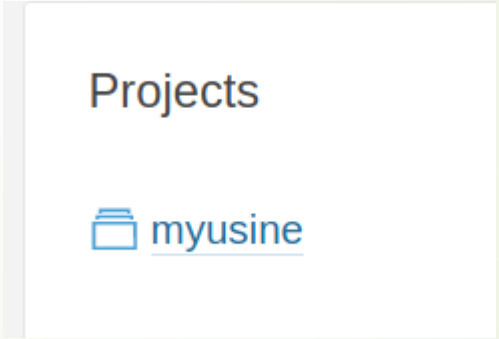


SONARQUBE : installation

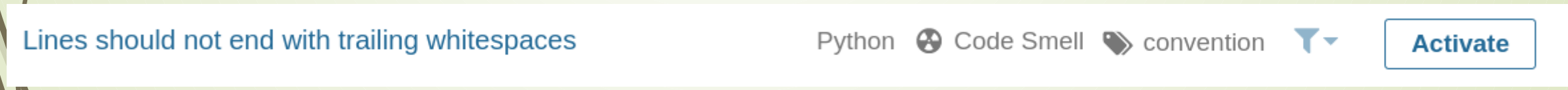
- Création du Profile qualité

Nouveau profile

Association du projet au profile

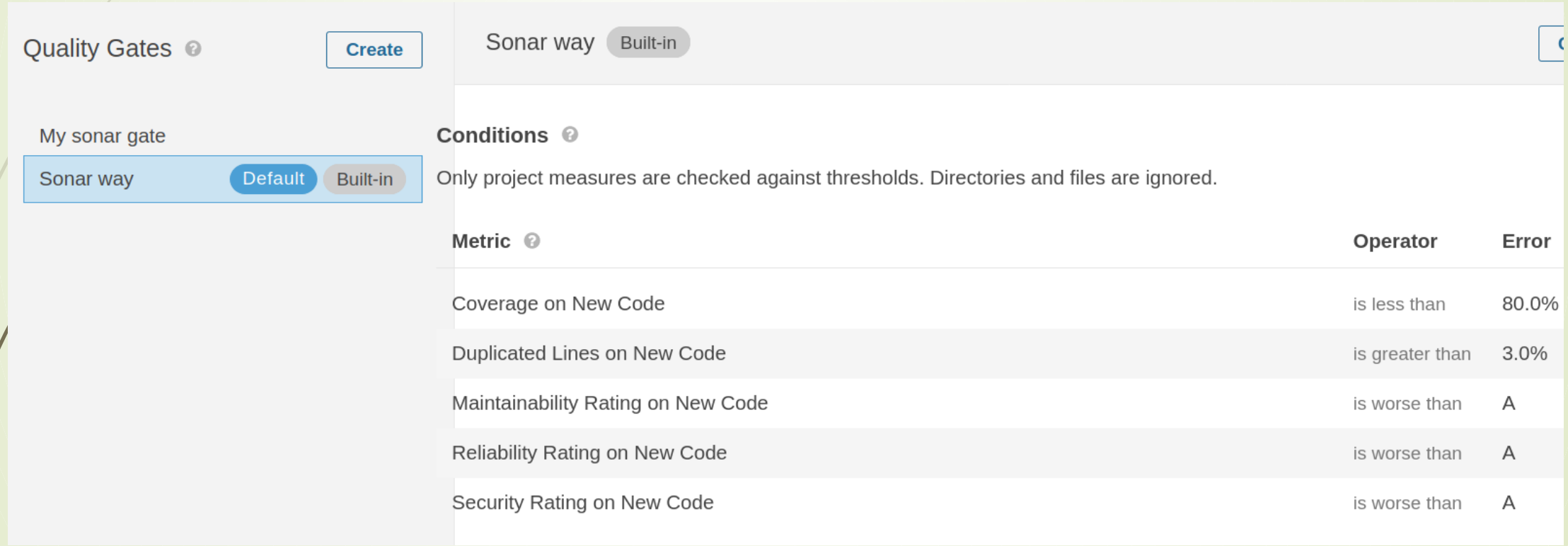


Ajout d'une nouvelle règle



SONARQUBE : installation

- « quality gates » : seuils de validation



SONARQUBE : scanner

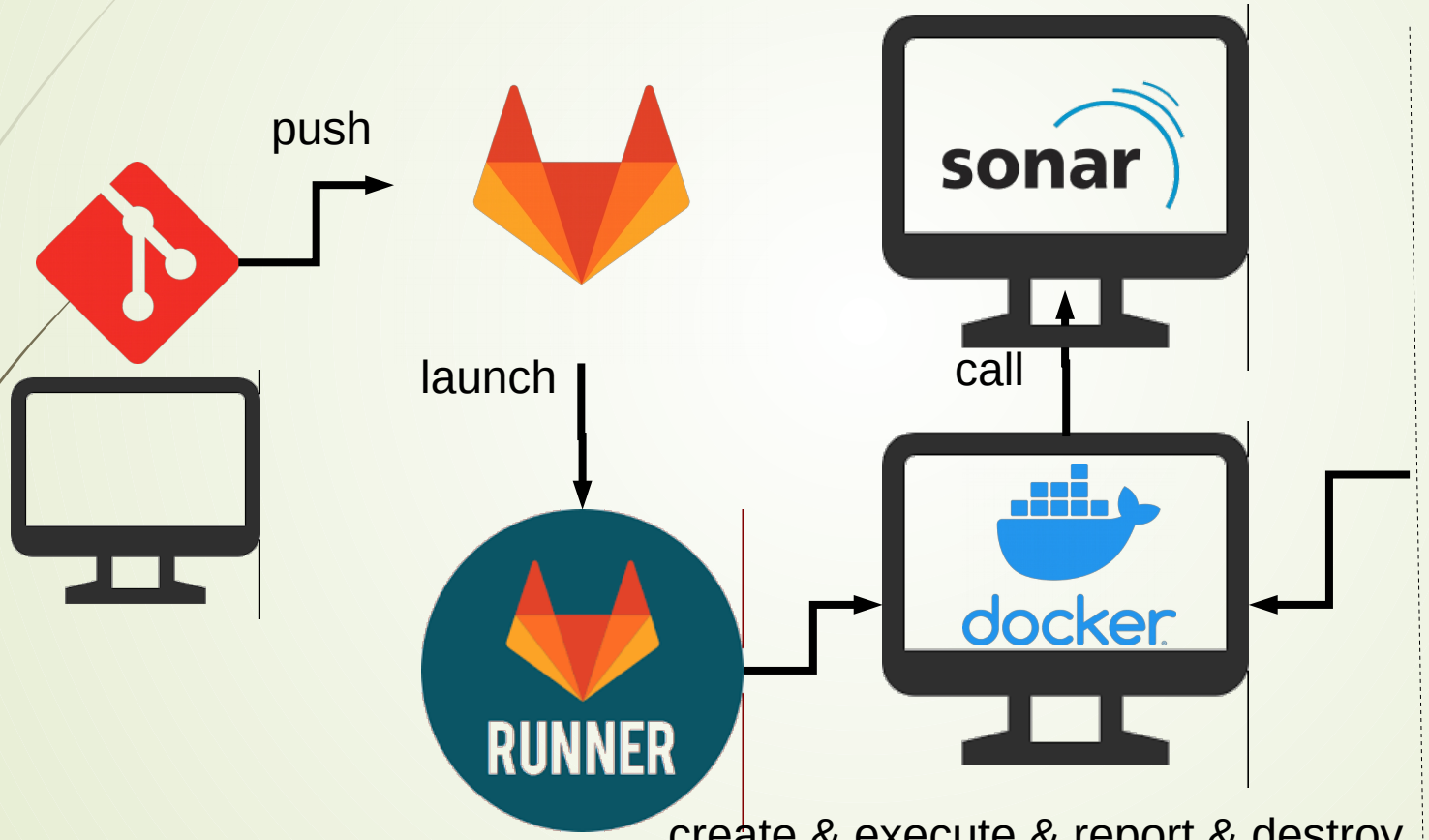
- Sonar-scanner : l'outil de scrutation du code

  - Logiciel à télécharger sur sonarqube, préselectionné en fonction du projet
  - Fichier de configuration .properties (monde java) à la racine du dépôt local
  - Permet de distinguer le code à analyser et le type d'analyse

```
sonar.projectKey=myusine
sonar.projectName=myusine
sonar.host.url=http://172.17.0.1:9000
sonar.login=5aa127a39fba7b9702217c9479c6609f972cfc94
sonar.sources=app
sonar.qualitygate.wait=true
```

SONARQUBE: intégration

- Déclenchement d'un job sonarqube




```
sonar:
  stages :
    - quality
  image:
    name :
sonarsource/sonar-
scanner-cli:latest
    ...
  script:
    - sonar-scanner
  ...
```

SONARQUBE : intégration

- Configuration du job gitlab-ci.yml
  - Image dédiée au job : **sonarsource/sonar-scanner-cli:latest**
  - Clé « **variables** » : Variables d'environnement spécifiques au job
  - Clé « **allow_failure** » : job non bloquant

```
sonar:
  stage: style
  image:
    name: sonarsource/sonar-scanner-cli:latest
    entrypoint: [""]
  variables:
    SONAR_TOKEN: 4ad2cdccf773cb8fa5b28ef76baefab366fcbe24
    SONAR_HOST_URL: http://172.17.0.1:9000
  script:
    - >
      sonar-scanner -Dsonar.projectKey=myusine
      -Dsonar.sources=.
      -Dsonar.exclusions=lib/**/*,lib64/**/*
      -Dsonar.host.url=$SONAR_HOST_URL
      -Dsonar.login=$SONAR_TOKEN
      -Dsonar.scm.provider=git
  allow_failure: true
```

SONARQUBE : intégration

- Protection des donnés sensibles
  - On peut déclarer les variables d'environnement à l'extérieur de la configuration
  - http://gitlab.myusine.fr/root/myusine/-/settings/ci_cd (volet « Variables »)
  - Les données sont protégées en lecture une fois renseignées
  - Les données sont disponibles directement dans les conteneurs de CI exécutés

Type	Key	Value	Protected	Masked
Variable	SONAR_HOST_URL	**********	✓	✗
Variable	SONAR_TOKEN	**********	✓	✗

SONARQUBE : analyse

- Résultats sur le serveur