Le CI/CD avec GitLab

Ou comment laisser le cloud faire votre taff

Comment travailler avec tous les outils modernes ?

Et le mode agile ???

Développer, c'est écrire du code (non)


```
vim src/mypackage/module.py     # Make a minor edit
pytest -rf --cov=MyApp -cov-report=xml:report.xml  # Run tests and coverage
black --check --diff            # Check linting (code style)
git add *                       # Get ready to commit
git commit -m "Fix thingy"      # Commit to local repository
git push                        # Push to Git remote
docker build -t myapp:latest .  # Rebuild docker image
docker push --all               # Push Docker image to registry
ssh user@server                 # Login to production server
docker pull myapp:latest        # Pull new image
docker restart mycontainer      # Restart production container
echo "Hey I pushed to prod“ | mail -s Update boss@company  # Let our boss know
```

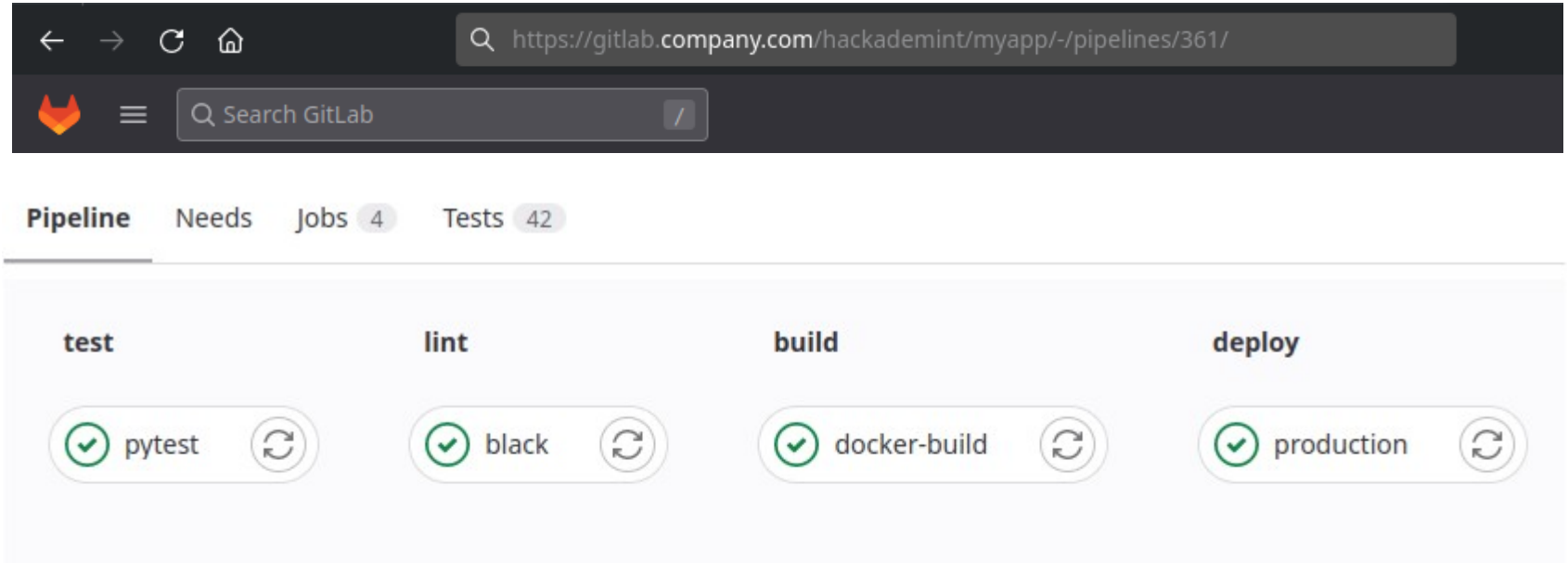
Développer, c'est écrire du code (non)

```
vim src/mypackage/module.py      # Make a minor edit
pytest -rf --cov=MyApp -cov-report=xml:report.xml  # Run tests and coverage
black --check --diff             # Check linting (code style)
git add *                        # Get ready to commit
git commit -m "Fix thingy"       # Commit to local repository
git push                         # Push to Git remote
docker build -t myapp:latest .   # Rebuild docker image
docker push --all                # Push Docker image to registry
ssh user@server                  # Login to production server
docker pull myapp:latest         # Pull new image
docker restart mycontainer       # Restart production container
echo "Hey I pushed to prod“ | mail -s Update boss@company  # Let our boss know
```

Développer, c'est écrire du code (non)


```
vim src/mypackage/module.py     # Make a minor edit
pytest -rf --cov=MyApp -cov-report=xml:report.xml
black --check --diff
git add *                       # Get ready to commit
git commit -m "Fix thingy"      # Commit to local repository
git push                        # Push to Git remote
docker build -t myapp:latest .
docker push --all
ssh user@server
docker pull myapp:latest
docker restart mycontainer
echo "Hey I pushed to prod“ | mail -s Update boss@company
```

Un peu de modernité



CI : Continuous Integration

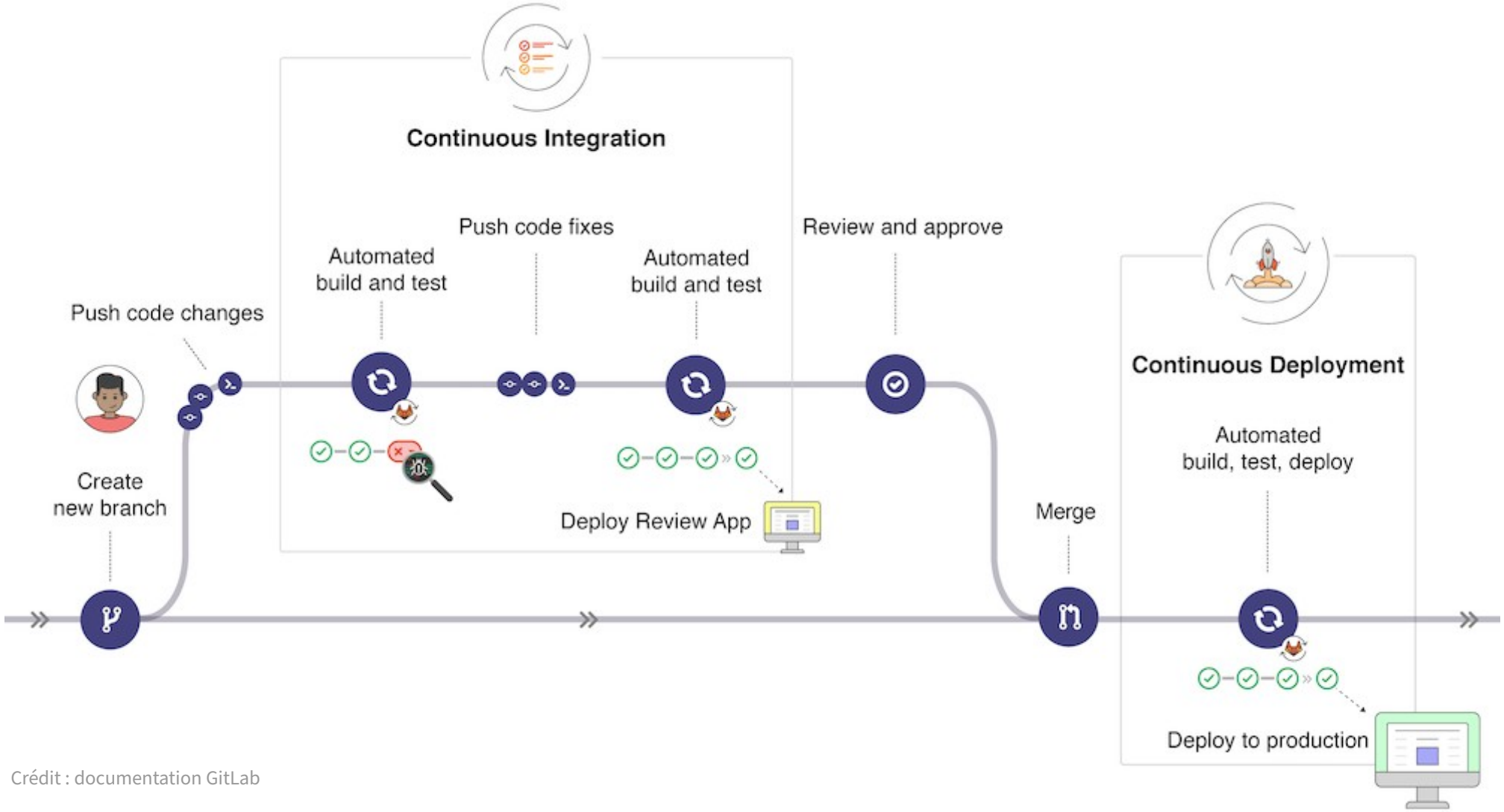
Intégration continuelle des outils de validation dans le processus de développement

- Détection des problèmes au plus tôt (régression, vulnérabilités, formatage, ...)
- Simplification de la vie de l'équipe de développement (c'est le CI qui fait tout !)
- Gain de temps (les tests sur une grosse application peuvent prendre plusieurs heures)
- Traçabilité (tout les résultats sont gardés)
- Collaboration (possibilité de discuter directement sur l'intégration)

CD : Continuous Deployment / Delivery

Rapprochement des environnements de développement et de production

- Publication automatique des applications & librairies
- Mise en production automatique des mises à jour
- Attention : suppose une gestion des droits et une validation du code fiable
- Extrêmement puissant à l'ère du Cloud



Crédit : documentation GitLab

L’intégration continue avec GitLab

Les deux acteurs principaux

GitLab CI/CD

GitHub Actions

GitLab – fonctionnalitées

- Pipelines
- Affichage des résultats des tests
- Affichage des résultats de scans de vulnérabilités (ultimate uniquement)
- Affichage des résultats de scans des dépendances
- Prise en compte des rapports de couverture des tests
- Registres de conteneurs et de paquets intégrés
- Intégration Kubernetes
- Intégration Terraform
- Intégration GCP
- CRON
- ...

Merge branch 'upstream/update' into 'develop' ...

Smyler authored 4 days ago

c7a884aa

develop ctfd / +

History Find file Web IDE Clone

Name	Last commit	Last update
.github	Migrated GitHub linting action to Gitlab C...	4 months ago
CTFd	Add individual DATABASE_* options, as an...	2 weeks ago
conf/nginx	2.5.0 dev (#1453)	2 years ago
migrations	Fix issue with scoreboard ordering when ...	2 months ago
scripts	Bump Python to 3.9 (#2096)	8 months ago
tests	Cache challenge data for faster loading of...	1 month ago

passed **Pipeline #739** triggered 4 days ago by **Smyler** Delete

Merge branch 'upstream/update' into 'develop'

```
Update from upstream

See merge request !7
```

16 jobs for `develop` in 11 minutes and 19 seconds (queued for 31 seconds)

latest

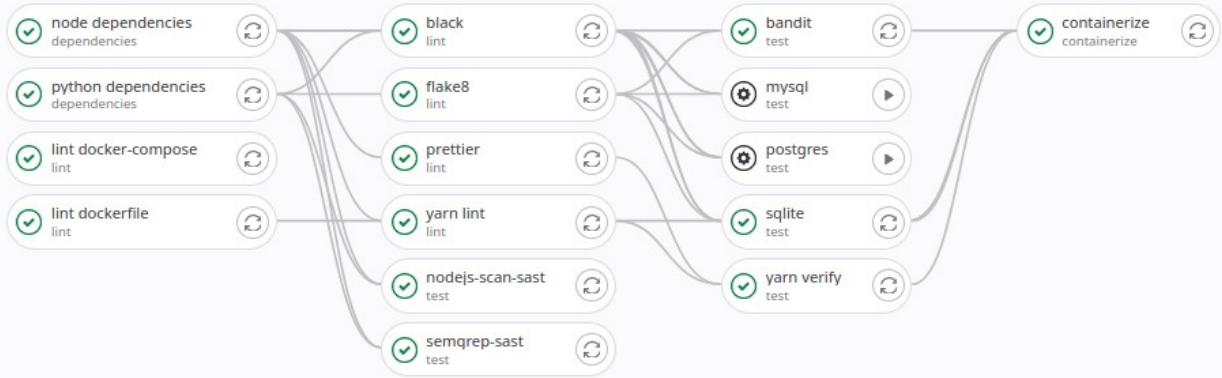
c7a884aa

No related merge requests found.

Pipeline Needs Jobs 16 Tests 495

Group jobs by Stage | Job dependencies **Show dependencies**

Tip: Hover over a job to see the jobs it depends on to run.



passed **Pipeline #739** triggered 4 days ago by **Smyler** Delete

Merge branch 'upstream/update' into 'develop'

```
Update from upstream

See merge request !7
```

16 jobs for `develop` in 11 minutes and 19 seconds (queued for 31 seconds)

latest

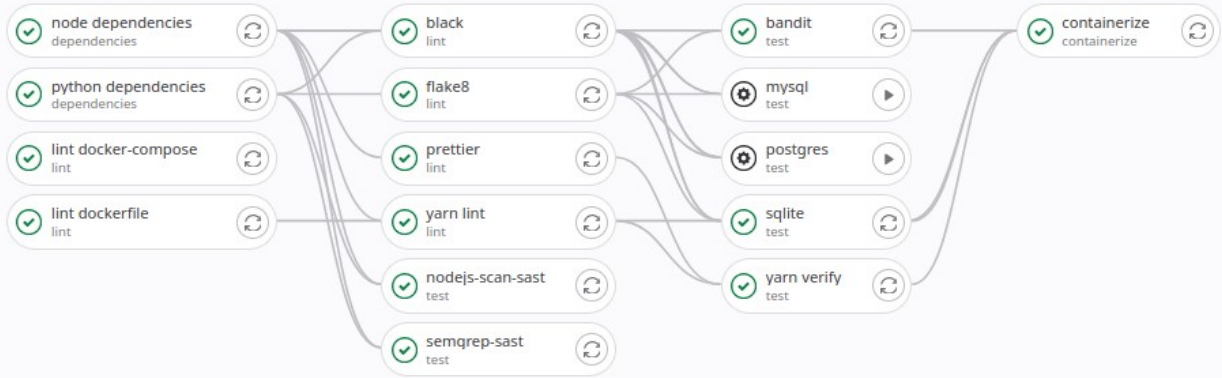
c7a884aa

No related merge requests found.

Pipeline Needs Jobs 16 Tests 495

Group jobs by Stage | Job dependencies **Show dependencies**

Tip: Hover over a job to see the jobs it depends on to run.



Update from upstream

Edit | Code

Merged **Smyler** requested to merge `upstream/update` into `develop` 5 days ago

Overview 0 | Commits 16 | Pipelines 3 | Changes 47

👍 0 👎 0

Pipeline #736 passed for 5880c971 on upstream/update 5 days ago

Approval is optional

Test summary: **no** changed test results, **495** total tests — Full report

Security scans have run — Download results

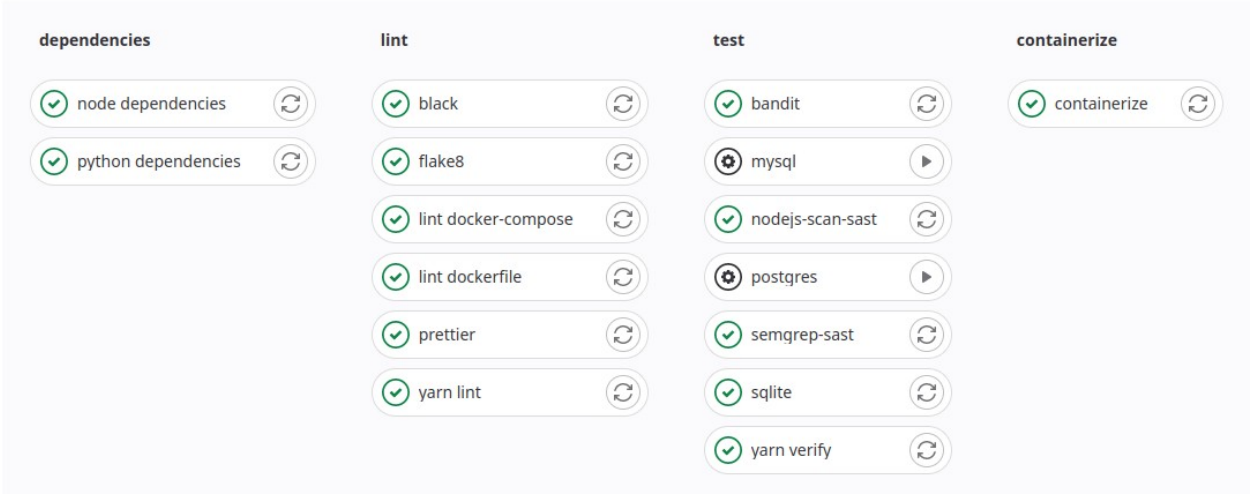
Merged by Smyler 4 days ago — Revert | Cherry-pick

Merge details

- Changes merged into **develop** with c7a884aa.
- Deleted the source branch.

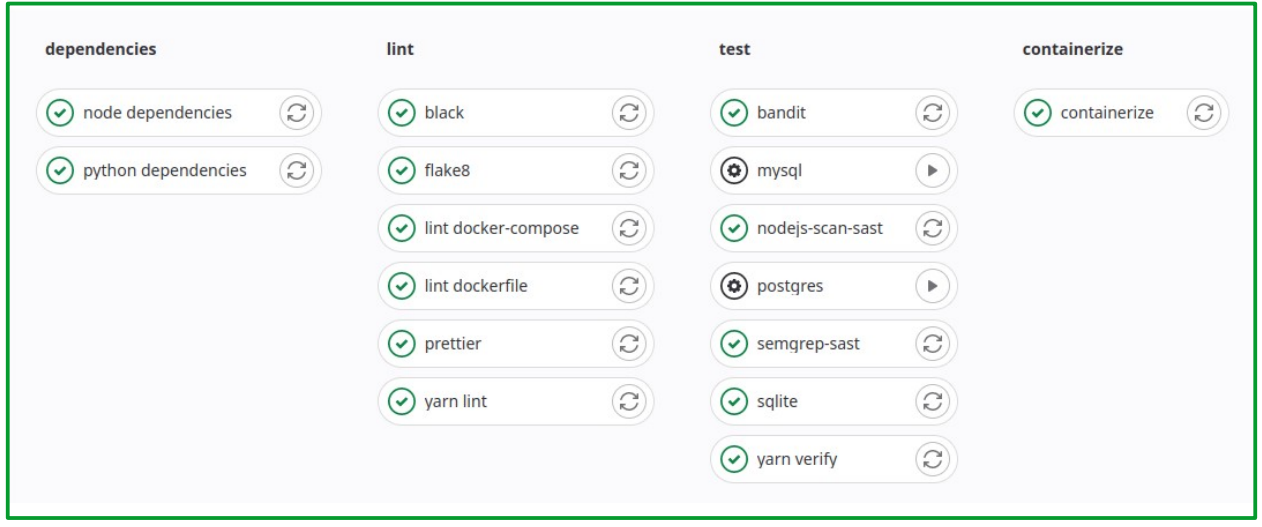
Pipeline #739 passed for c7a884aa on develop 4 days ago

Un peu de vocabulaire



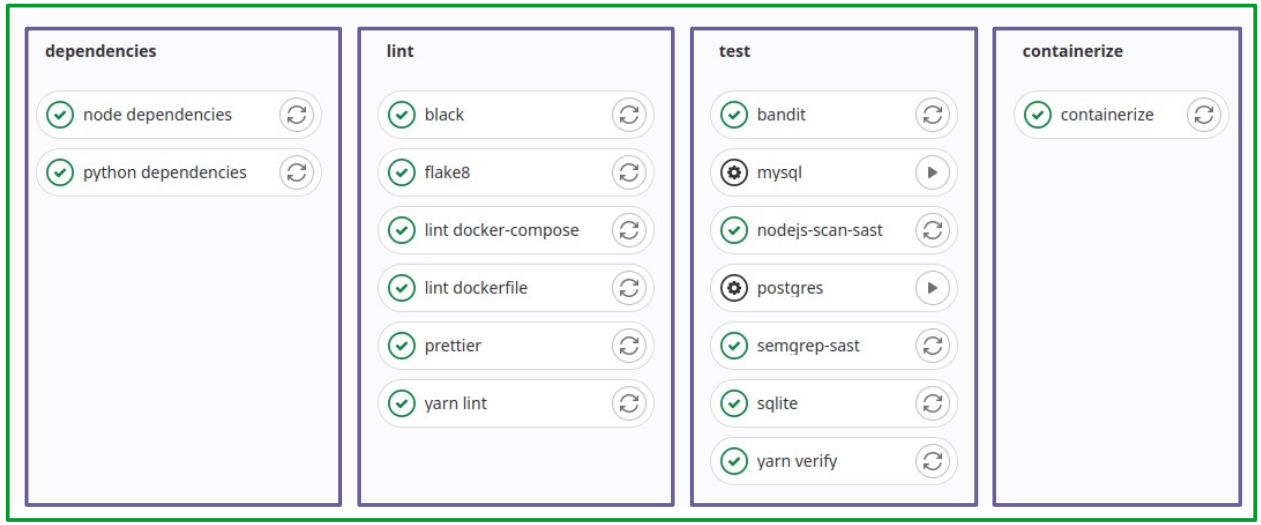
Un peu de vocabulaire

- Pipeline : instance d'exécution de CI-CD



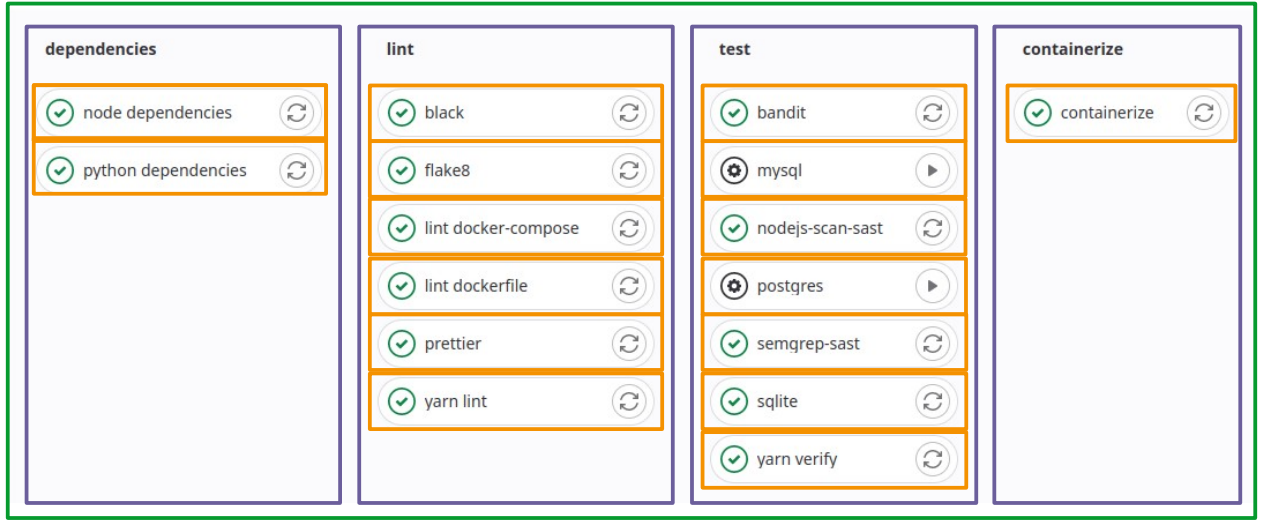
Un peu de vocabulaire

- Pipeline : instance d'exécution d'une suite de CI-CD
- Stage : une étape dans une *pipeline*



Un peu de vocabulaire

- Pipeline : instance d'exécution d'une suite de CI-CD
- Stage : une étape dans une *pipeline*
- Job : une tâche à réaliser lors d'un *stage*, isolé dans un conteneur



Un peu de vocabulaire

Un peu de vocabulaire

- Tests unitaires : test du fonctionement microscopique

```
void add(int a, int b) {
    return a + b;
}
```

Implémentation

```
@Test
void canAddTwoIntegers() {
    assertEquals(42, add(40, 2);
}
```

Test

Un peu de vocabulaire

- Tests unitaires : test du fonctionement microscopique
- Tests d'intégration : test du fonctionnement macroscopique

```
@Test
void isApiWorking() {
    WebApp app = new MyApp();
    app.startInBackground("localhost", 80);

    WebClient client = new WebClient();
    WebResponse response = client.request("http://localhost/api/");
    assertEquals(200, response.statusCode);
}
```

Un peu de vocabulaire

- Tests unitaires : test du fonctionement microscopique
- Tests d'intégration : test du fonctionnement macroscopique

```
@Test
void isApiWorking() {
    WebApp app = new MyApp();
    app.startInBackground("localhost", 80);

    WebClient client = new WebClient();
    WebResponse response = client.request("http://localhost/api/");
    assertEquals(200, response.statusCode);
}
```

Un peu de vocabulaire

- Tests unitaires : test du fonctionement microscopique
- Tests d’intégration : test du fonctionnement macroscopique

Un peu de vocabulaire

- Tests unitaires : test du fonctionement microscopique
- Tests d'intégration : test du fonctionnement macroscopique
- Linting : vérification de la mise en forme du code

```
void main(String… args)        {
    int a = 40                 ;
    int b = 2                  ;
    System
        .out.println(   a+b)   ;
                               }
```

```
void main(String… args) {
    int a = 40;
    int b = 2;
    System.out.println(a + b);
}
```

Un peu de vocabulaire

- Tests unitaires : test du fonctionement microscopique
- Tests d'intégration : test du fonctionnement macroscopique
- Linting : vérification de la mise en forme du code

```
void main(String… args)       {
    int a = 40                ;
    int b = 2                 ;
    System
        .out.println(   a+b);}
```


```
void main(String… args) {
    int a = 40;
    int b = 2;
    System.out.println(a+b);
}
```

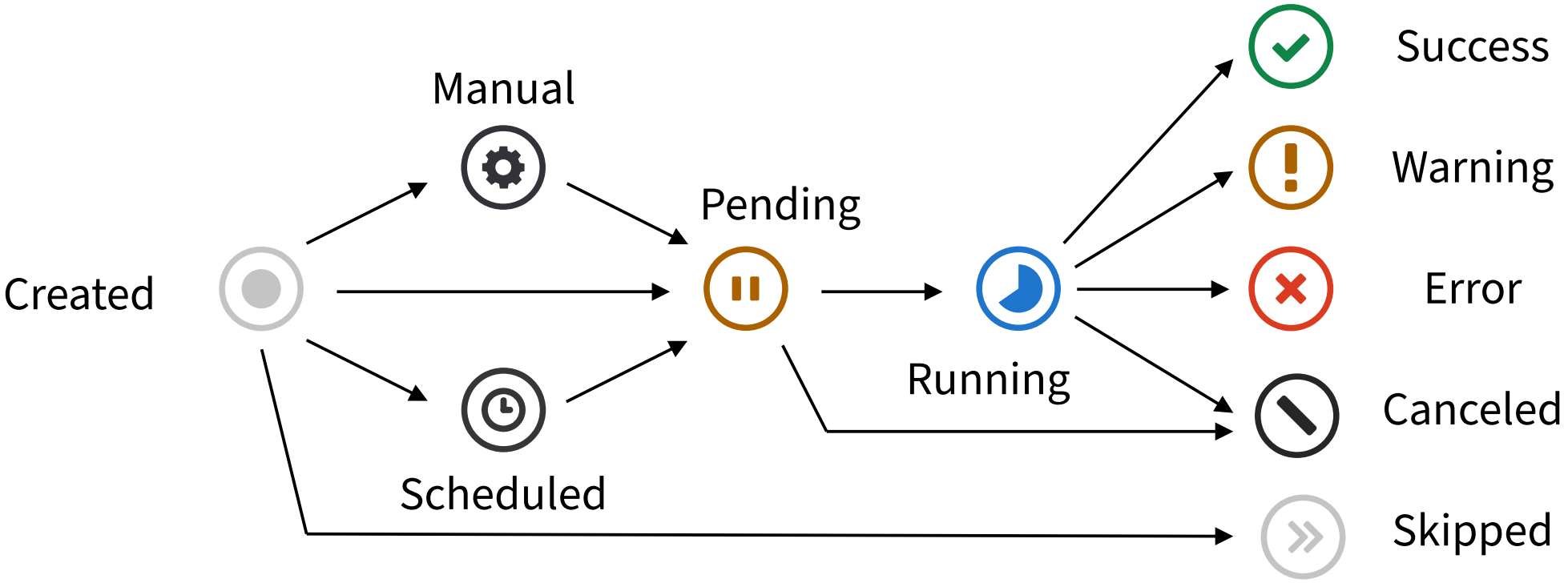
Un peu de vocabulaire

- Tests unitaires : test du fonctionement microscopique
- Tests d’intégration : test du fonctionnement macroscopique
- Linting : vérification de la mise en forme du code
- SAST : Static Application Security Testing (Semgrep)

L’interface avec un peu plus de détails

Démo

Les indicateurs de status



C’est bien tout ça, mais comment qu’on fait ???

Disgression - YAML

- Superset de JSON
- Très utilisé pour les fichier de configuration
- Très simple à lire et à éditer
- Types de données : Booléens, nombres, chaînes de caractères, listes et dictionnaires
- Ça s’apprend sur le tas

.gitlab-ci.yml – Structure

```yaml
stages:
  - build
  - test

image: "ubuntu:latest"
variables:
  ...

build-code-job:
  stage: build
  ...

test-code-job1:
  stage: test
  ...

test-code-job2:
  stage: test
  ...
```

.gitlab-ci.yml – Structure

```
stages:
  - build
  - test

image: "ubuntu:latest"
variables:
  ...

build-code-job:
  stage: build
  ...

test-code-job1:
  stage: test
  ...

test-code-job2:
  stage: test
  ...
```

Déclaration des stages

.gitlab-ci.yml – Structure

```yaml
stages:
  - build
  - test

image: "ubuntu:latest"
variables:
  ...

build-code-job:
  stage: build
  ...

test-code-job1:
  stage: test
  ...

test-code-job2:
  stage: test
  ...
```

Déclaration des stages

Options communes à tous les jobs

.gitlab-ci.yml – Structure

```yaml
stages:
  - build
  - test

image: "ubuntu:latest"
variables:
  ...

build-code-job:
  stage: build
  ...

test-code-job1:
  stage: test
  ...

test-code-job2:
  stage: test
  ...
```

Déclaration des stages

Options communes à tous les jobs

Déclaration des jobs, en indiquant les stages dont ils dépendent

.gitlab-ci.yml – Jobs

```yaml
build-code-job:
  stage: build
  script:
    - echo "Check the ruby version, then build some Ruby project files:"
    - ruby -v
    - rake
```

```yaml
build-code-job:
  stage: build
  image: "ubuntu:latest"
  script: python setup.py wheel
```

.gitlab-ci.yml – Artifacts

- Permet de sauvegarder des fichiers produits par un job
- Les artéfacts peuvent être passés d'un job à l'autre, ajoutés à une release, téléchargés depuis la page de la pipeline…
- Dans le cas où ce sont des rapports, ils peuvent être interprétés par GitLab

```yaml
gradle-build:
  stage: build-java
  script: ./gradlew build
  artifacts:
    paths:
      - build/libs
```

```yaml
gradle-test:
  stage: test-java
  script: ./gradlew test
  artifacts:
    reports:
      junit: "**/build/test-results/test/**/TEST-*.xml"
```

.gitlab-ci.yml – Needs/Dependencies

- `Needs` permet de spécifier les dépendances entre les job et active l'exécution parallèle des stages
- `Dependencies` permet de préciser qu'un job doit récupérer les artéfacts d'une d'un job dont il dépend (si non précisé, tout les artéfacts sont récupérés)

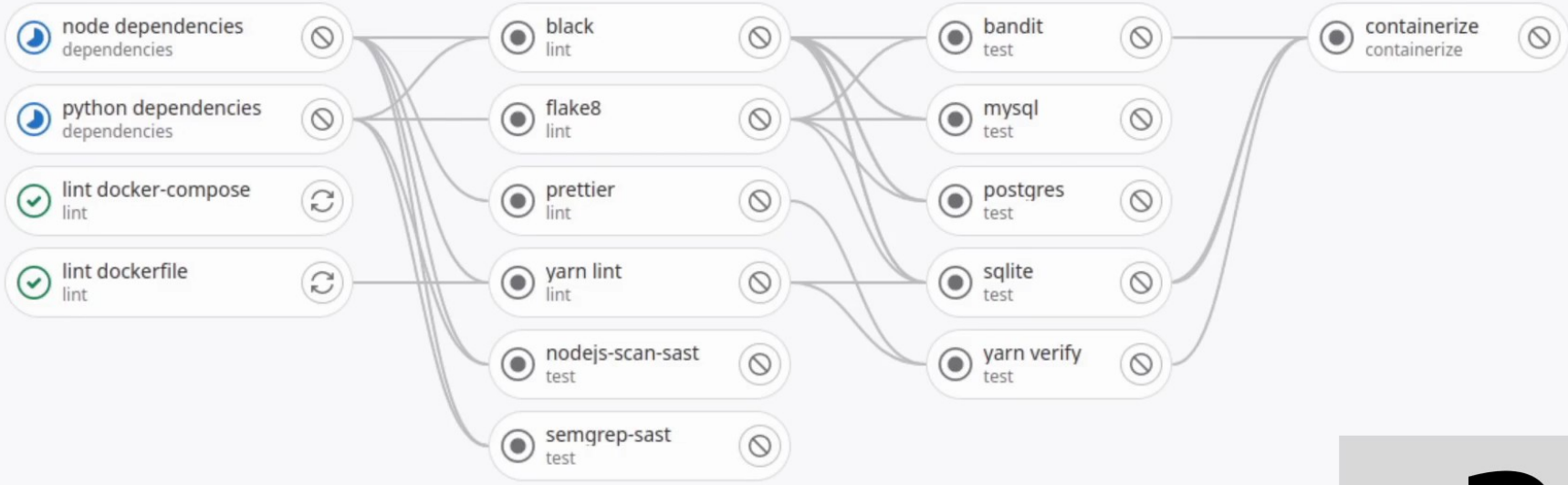
```yaml
gradle-build:
  stage: build-java
  script: ./gradlew build
  artifacts:
    paths:
      - build/libs
```

```yaml
gradle-test:
  stage: test-java
  script: ./gradlew test
  needs: ["gradle-build"]
  dependencies: ["gradle-build"]
  artifacts:
    reports:
      junit: "**/build/test-results/test/**/TEST-*.xml"
```

No related merge requests found.

Pipeline Needs Jobs 16 Tests 0

Group jobs by Stage | Job dependencies **Show dependencies**



x30

Et autres mot clés…

- `cache` : Utilisation d'un cache persistent d'une pipeline à l'autre
- `coverage` : Détection des rapport de couverture de test
- `environment` : Déclaration de l'environment de déploiement
- `when` : Condition d'exécution
- `parallel` : Exécution de plusieurs instances du job en parallèle, avec des variables différentes
- `release` : Création d'une release
- `services` : Déclaration d'un conteneur de service à lancer en parallèle (ex. Une base de donées)
- `variables` : Variables d'environment
- …

It’s Kahoot time !