

GitLab Action with MySQL on a Java Spring Application

Cansell, Maxime Le-Point-Technique, March/2022

abstract: Ce document présente la configuration d'une action GitLab avec une base de données MySQL dans un environnement Java Spring.

keywords: GitLab, DevOps, MySQL, Java, Spring

Introduction

Ce document décrit les étapes de configuration d'une action GitLab avec une base de données MySQL dans une application Java utilisant le framework Spring. Il s'inspire en partie du tutoriel DevOps with GitLab. Ce document comprend deux sections: création du pipeline CI et conclusion. La création du pipeline CI contient cinq sous-sections : configuration du pipeline, Instanciation du service MySQL, enregistrement des variables d'environnement avec GitLab, exécution de la pipeline et présentation des artifacts de builds.

Création du pipeline CI

Dans cette section, nous décrivons les différentes étapes de la configuration du fichier `.gitlab-ci.yml`, configuration centrale de notre pipeline. Nous verrons ensuite comment se déroulent les tests, les rapports associés et comment déboguer notre pipeline jusqu'à son fonctionnement.

Configuration

Afin de paramétrer notre pipeline, nous devons commencer par créer un fichier, nommé `.gitlab-ci.yml`, à la racine de notre repository. Ce fichier est composé de l'extrait de code visible *Figure 1*. Nous allons décrire sa composition pour chaque élément de code dans les sous-sections suivantes. Plus tard nous rajouterons le code qui concerne la configuration de l'image de la base de données MySQL.

A chaque push du code source des applications ou du fichier `.gitlab-ci.yml` sur le repository, le code de ce fichier `.gitlab-ci.yml` sera exécuté automatiquement.

Prenons le temps d'analyser cette exemple ligne par ligne:

- **image** (ligne 1) : Nous installons l'image docker Alpine qui est une distribution Linux ultra-légère que notre pipeline va utiliser pour exécuter nos scripts. Nous y intégrons le JDK JAVA 11.
- **stages** (ligne 5) : Nous décrivons ici de quoi sera composée notre pipeline ; trois stage, un nommé build qui testera que le build de l'application fonctionne normalement et un nommé tests qui réalisera les tests unitaires

```

1 image: adoptopenjdk/openjdk11:jdk-11.0.11_9-alpine
2 # Nous installons l'image docker Alpine qui est une distribution Linux ultra-légère
3 # que le runner va utiliser pour exécuter nos scripts avec le JDK JAVA 11
4
5 stages:
6   - build
7   - tests
8   - .post
9
10 build-ms-ERS-Emergency-Responder:
11   stage: build
12   script:
13     - cd ERS-Emergency-Responder
14     - ./mvnw compile # Commande maven pour compiler le code source du projet
15     - echo "Le projet ERS-Emergency-Responder build..."
16
17 tests-ms-ERS-Emergency-Responder:
18   stage: tests
19
20   script:
21     - cd ERS-Emergency-Responder
22     - ./mvnw surefire-report:report # Créé un rapport d'exécution des tests au format html
23     - echo "Les tests ERS-Emergency-Responder s'exécutent..."
24
25   artifacts:
26     when: always
27     # paths permet de sauvegarder les artefacts générés pendant l'exécution du script sur le GitLab Server
28     # et de les retrouver dans l'onglet browse du job ou le bouton download du pipeline
29     paths:
30       - ERS-Emergency-Responder/target/site/surefire-report.html
31       - ERS-Emergency-Responder/target/surefire-reports/TEST-*.xml
32     # reports:junit permet de récupérer les artefacts de Test xml afin d'intégrer les rapports
33     # dans l'onglet test des détails d'un job
34     reports:
35       junit:
36         - ERS-Emergency-Responder/target/surefire-reports/TEST-*.xml
37
38 check-tests-failure:
39   stage: .post
40   dependencies:
41     - tests-ms-ERS-Emergency-Responder
42   script:
43     - (! grep "<failure" ERS-Emergency-Responder/target/surefire-reports/TEST-*.xml)
44     - (! grep "<error" ERS-Emergency-Responder/target/surefire-reports/TEST-*.xml)

```

Figure 1: Exemple d'action GitLab.

et d'intégration. Le troisième stage sera chargé de vérifier si les tests ont réussis ou échoués. Ces trois étapes sont totalement indépendantes. En effet, chaque stage se déroule un après l'autre et crée son conteneur docker. La base de données MySQL sera installée dans le conteneur du stage "tests" et donc en ligne de commande dans son script (voir point MySQL).

- **build-ms-ERS-Emergency-Responder** (ligne 10) : Nous nommons ici notre premier job et l'attribuons au stage build (ligne 11). Les lignes de commandes du script (ligne 13) sont ensuite exécutées : parcourir vers le dossier de l'application, build l'application avec Maven et enfin afficher dans nos logs que cela s'exécute.
- **tests-ms-Emertency-Responder** (ligne 17) : Deuxième job, même principe que le job précédent. C'est ici les tests unitaires et d'intégrations que contient notre application qui sont lancés lors du script. La clé nommée artifacts décrit les méthodes de sauvegarde des rapports (se référer aux commentaires de l'extrait de code). Le pipeline échoue uniquement si une des commandes échoue (Elles renvoient alors le code 1 ou 2 qui font fail le pipeline et le job). La réussite d'une commande renvoie 0 et le job passe à la commande suivante. Ici le job n'analyse pas la réussite ou l'échec des tests mais seulement le bon déroulement des commandes. Le job réussi donc même si des tests unitaires ou d'intégration échouent. Le job suivant se chargera de vérifier et de faire fail le pipeline si un des tests ne passe pas.
- **check-tests-failure** (ligne 38) : Ce troisième job nous permet donc d'analyser le rapport Junit pour vérifier si les tests unitaires ou d'intégration ont échoué.
- **dependencies** (ligne 40) : Le mot clef dependencies permet de transmettre les rapports générés dans le job précédent et de les rendre accessibles dans ce stage.
- **script** (ligne 42) : Le script utilise la commande grep. Grep est un acronyme qui signifie Global Regular Expression Print. C'est un outil en ligne de commande Linux / Unix utilisé pour rechercher une chaîne de caractères dans un fichier spécifié. Le modèle de recherche de texte est appelé une expression régulière (regex). Nous cherchons donc à savoir si une balise "<failure[...]" ou "<error[...]" existe dans le rapport Junit. Cette commande renvoie 0 si elle trouve l'expression et 1 si elle ne la trouve pas. Nous inversons la condition avec le point d'exclamation : s' il la trouve il renverra 1 qui fera échouer le job et donc notre pipeline.

Instanciation du service MySQL

Nous devons tout d'abord ajouter (ligne 9) nos variables MySQL pour permettre la connexion à la base de données. Attention, il conviendra de mettre en place External Secret GitLab pour sécuriser les variables de connexion qui sont visibles

(Figure 2).

```
9   variables:
10   MYSQL_DATABASE_NAME: hospitals
11   MYSQL_DATABASE: hospitals
12   MYSQL_ROOT_PASSWORD: ""
13   MYSQL_ALLOW_EMPTY_PASSWORD: 1
14   MYSQL_USER: admin
15   MYSQL_PASSWORD: 1234
16   MYSQL_HOST: mariadb
17   MYSQL_PORT: 3306
18
```

Figure 2: Configuration des variables MySQL.

Nous modifions ensuite le stage tests (Figure 3).

```
27   tests-ms-ERS-Emergency-Responder:
28     stage: tests
29
30     services:
31       - mariadb:latest
32
33     script:
34       - apk --no-cache add mysql-client
35       - mysql --version
36       - sleep 20
37       - mysql --host=mariadb -P 3306 --user=root --password="" "${MYSQL_DATABASE}" < data.sql
38       - mysql --host=mariadb -P 3306 --user=root --password="" -e 'SHOW TABLES FROM `hospitals`';
39       - cd ERS-Emergency-Responder
40       - ./mvnw surefire-report:report # Créé un rapport d'exécution des tests au format html
41       - echo "Les tests ERS-Emergency-Responder s'exécutent..."
```

Figure 3: Screenshot des modifications sur le stage tests.

Nous allons utiliser un service nommé mariadb pour y créer notre base de données MySQL (ligne 31). Un service est lui même créé dans un conteneur docker à l'intérieur du conteneur du stage test. Pour accéder à celui-ci il faut donc s'adresser au conteneur docker portant son nom "mariaDB". L'URL d'accès à la base de données ne sera donc pas localhost mais mariadb (exemple mariadb://3306/hospitals). Nous y reviendrons au point suivant. La commande

apk (ligne 34) installe le client mysql. Comme nous le constatons sur ce lien il convient d'attendre 20 secondes (ligne 36) pour s'assurer de l'installation correcte. (Nous pourrions optimiser cela dans les développements futurs en testant si ce temps pourrait être réduit). Nous nous connectons ensuite et créons notre base de données grâce au script data.sql contenu à la racine de notre application (ligne 37). Nous affichons les tables créées de la base de données dans nos logs (ligne 38) puis exécutons les tests grâce aux commandes déjà vues précédemment (lignes 39/40).

Enregistrement des variables GitLab

Gitlab permet d'enregistrer des variables d'environnement qui seront réutilisées dans notre pipeline et donc par notre application en fonctionnement dans nos stages (voir *Figure 4*). Suivre : **seetings > Ci/CD > Variables** et **Expand**.

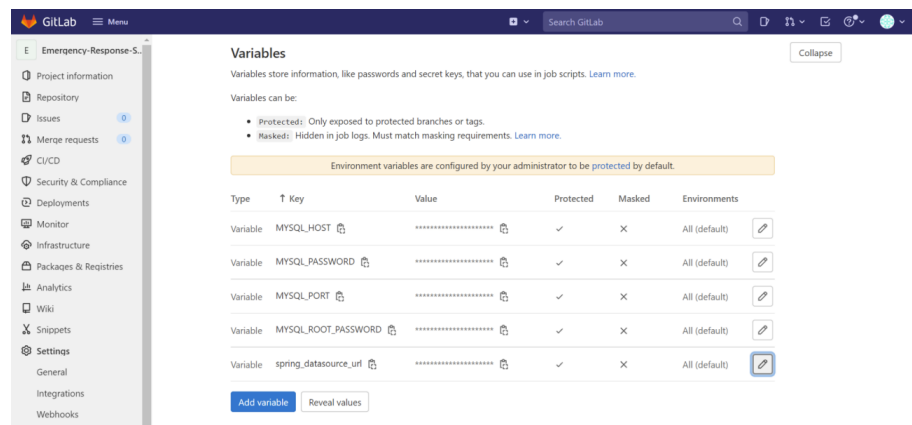


Figure 4: Capture d'écran des variables d'environnement sous GitLab.

Ces variables peuvent écraser et remplacer les variables de notre application comme celles contenues dans le fichier de configuration Spring de notre application : application.properties mais aussi celles du fichier de configuration du pipeline : .gitlab-ci.yml.

Il convient de respecter la case majuscule underscore pour écraser les variables du fichier yml et minuscule underscore pour celles du fichier application.properties. Pour que notre conteneur où s'exécute l'application et ses tests puissent communiquer avec le conteneur du service mariadb et sa base de donnée MySQL nous devons écraser la valeur de la propriété "spring.datasource.url" du fichier application.properties de notre application Spring (*Figure 5*).

```
spring.datasource.url=jdbc:mysql://localhost:3306/hospitals
```

Figure 5: Mise à jour des propriétés spring boot.

La variable se nommera donc “spring_datasource_url” et aura pour valeur, comme nous l’avons vu précédemment : “jdbc:mysql://mariadb:3306/hospitals”
Note concernant la conception de la base de donnée : il convient de respecter la case minuscule underscore dans le nom des tables et des de champs de la base de donnée pour respecter les paramètres par défauts de JPA hibernate. Attention, une variable gitlab même indiquée comme masquée n’est pas entièrement sécurisée. Pour sécuriser correctement les variables comme par exemple des clefs API ou des clefs de connexion à des bases de données il conviendra d’utiliser des external secret gitlab

Exécution de la pipeline

A chaque commit le pipeline est exécuté. Son exécution peut passer par l’état “running” quand il est en cours d’exécution, “passed” quand l’exécution s’est déroulée correctement ou “failed” quand elle à échoué (indépendant de la réussite des tests ou de leurs échecs) - *Figure 6*.

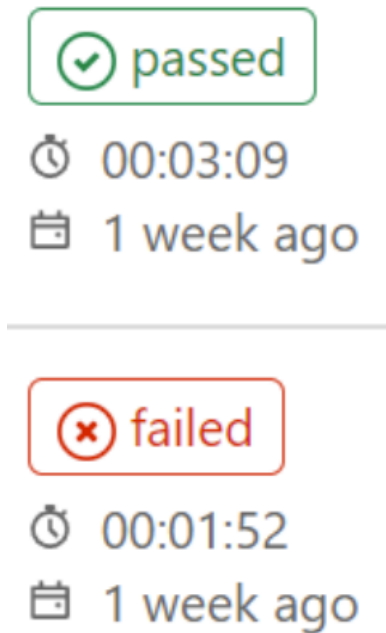


Figure 6: Statut de la pipeline.

Emergency-Response-S...

- Project information
- Repository
- Issues
- Merge requests
- CI/CD
- Pipelines
- Editor
- Jobs
- Schedules
- Security & Compliance
- Deployments
- Monitor
- Infrastructure
- Packages & Registries
- Analytics
- Wiki
- Snippets
- Settings

Showing last 492.93 kb of log - [Complete Raw](#)

```

2387 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-core/2.0.jar
2388 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-core/2.0.jar (189 kB at 292 kB/s)
2389 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar
2390 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar (68 kB at 97 kB/s)
2391 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar
2392 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar (347 kB at 515 kB/s)
2393 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar
2394 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar (450 kB at 644 kB/s)
2395 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar
2396 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar (457 kB at 619 kB/s)
2397 Downloading from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar
2398 Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/velocity/velocity-engine-velocity/3.2.2/velocity-engine-velocity-3.2.2.jar (588 kB at 788 kB/s)
2399 [WARNING] Unable to locate Test Source xRef to link to - DISABLED
2400 [INFO] -----
2401 [INFO] BUILD SUCCESS
2402 [INFO] -----
2403 [INFO] Total time: 01:04 min
2404 [INFO] Finished at: 2022-03-05T13:34:22Z
2405 [INFO] -----
2406 Uploading artifacts for successful job
2407 Uploading artifacts...
2408 ERS-Emergency-Response/target/surefire-report.html: found 1 matching files and directories
2409 Uploading artifacts as "archive" to coordinator... 201 Created id=2167415337 responseStatus=201 Created token n=Z2RCKn-Z
2410 Uploading artifacts...
2411 ERS-Emergency-Response/target/surefire-reports/TEST-*.xml: found 3 matching files and directories
2412 Uploading artifacts as "JUnit" to coordinator... 201 Created id=2167415337 responseStatus=201 Created token n=Z2RCKn-Z
2413 Uploading artifacts...
2414 Uploading project directory and file based variables
2415 Job succeeded

```

unit-test-ms-ERS...

Duration: 2 minutes 16 seconds
 Finished: 1 week ago
 Timeout: 1h (from project)
 Runner: f12270840 (x86_64 Linux)
 Show runner logs
[manager.github.com/default](#)

Job artifacts
 These artifacts are listed. They will not be deleted (even if expired) until newer artifacts are available.


[Keep](#) [Download](#) [Browse](#)

Commit [6e271ef8](#)
 tentative intégration de la bdd dans gitlab-ci.yml

[Pipeline #485307360 for main](#)
 unit-test

→ [unit-test-ms-ERS-Emergency...](#)

Artefacts créés lors du pipeline



The screenshot shows the GitHub Actions interface for a workflow named 'Emergency-Response-Suite'. The left sidebar contains navigation links: Project information, Repository, Issues, Merge requests, CI/CD, Pipelines (selected), Editor, Jobs, Schedules, Security & Compliance, Deployments, Monitor, Infrastructure, and Packages & Registries. The main content area is titled 'Tests' and displays a table of test results. The table has columns for Suite, Name, Filename, Status, Duration, and Details. There are three test entries, all with a status of 'Failed' (indicated by a red exclamation mark icon). The first two tests are 'testGetEmergencyBed' and 'testGetEmergencyBed', both with a duration of 1.54s. The third test is 'testCalculateDistance', with a duration of 33.00ms. Each test entry has a 'View details' button next to it. The bottom of the page shows a 'Summary' section with a 'View details' button.

Suite	Name	Filename	Status	Duration	Details
com.hospitalsoftware.ers.controller.HospitalControllersIntegrationTest	testGetEmergencyBed		Failed	1.54s	View details
com.hospitalsoftware.ers.controller.HospitalControllersUnitTest	testGetEmergencyBed		Failed	49.00ms	View details
com.hospitalsoftware.ers.service.geoTools.DistanceCalculatorServiceTest	testCalculateDistance		Failed	33.00ms	View details

29

Conclusion

Ce document nous a permis de mettre en place notre pipeline d'intégration continue pour une application Spring avec sa base de données de test MySQL. Ce document pourra être complété avec la mise en place d'outils de sécurité, notamment des tests de fuzzing.

References

- 'Secure Your Application | GitLab'. Accessed 29 June 2022. https://docs.gitlab.com/ee/user/application_security/.
- 'Using External Secrets in CI | GitLab'. Accessed 29 June 2022. <https://docs.gitlab.com/ee/ci/secrets/>.
- 'Web API Fuzz Testing | GitLab'. Accessed 29 June 2022. https://docs.gitlab.com/ee/user/application_security/api_fuzzing/.