

MON PROJECT GITOPS COMPLET CI/CD HELM ARGOCD PROMETHUS E GRAFANA

lunedì 22 settembre 2025 16:48

écrire un docker file qui implemente le build multistage les test avec bdd externe cet a dire avant le build metre en place une bdd et pendant le build l'app va tenter de se connecter a la bdd et on pourra lancer des test tous sa dans le dockerfile en suite mettre en place des helcheck

ci : mettre en place un linter du code coverage faire des test unitaire des test d'integration des test de non regression tous sa avant de builder l'app et apres la build et la run faire les test dans le cas de mon app se connecter et faire un test sur l'addition

```
stage('Initialize') {
    def dockerHome = tool 'dockerlatest'
    def mavenHome = tool 'mavenlatest'
    def javaHome = tool 'Java 11' // Force l'utilisation de Java 11

    env.JAVA_HOME = javaHome
    env PATH = "${javaHome}/bin:${dockerHome}/bin:${mavenHome}/bin:
${env.PATH}"
}
```

def dockerHome = tool 'dockerlatest'

- Utilise la fonction tool() de Jenkins pour récupérer l'installation de Docker
- 'dockerlatest' est le nom/identifiant de l'installation Docker configurée dans Jenkins (dans "Manage Jenkins" > "Global Tool Configuration")
- La variable dockerHome contiendra le chemin vers le répertoire d'installation de Docker

def nodeHome = tool 'nodelatest'

- Même principe pour Node.js
- 'nodelatest' est l'identifiant de l'installation Node.js dans Jenkins
- nodeHome contiendra le chemin vers Node.js

**env.PATH = "\${dockerHome}/bin:\${nodeHome}/bin:
\${env.WORKSPACE}/bin:\${env.PATH}"**

Cette ligne modifie la variable d'environnement PATH en ajoutant :

- \${dockerHome}/bin : le répertoire bin de Docker
- \${nodeHome}/bin : le répertoire bin de Node.js
- \${env.WORKSPACE}/bin : un répertoire bin dans l'espace de travail Jenkins (probablement pour Kustomize)
- \${env.PATH} : conserve le PATH existant

```
withCredentials([usernamePassword(credentialsId: 'dockerhubcredential', usernameVariable:
'USERNAME', passwordVariable: 'PASSWORD')]) { pushToImage(CONTAINER_NAME,
CONTAINER_TAG, USERNAME, PASSWORD)
```

Ce code utilise le système de gestion des credentials de Jenkins pour récupérer des identifiants Docker Hub de manière sécurisée. Voici l'explication :

withCredentials([...])

- Bloc Jenkins qui permet d'accéder temporairement à des credentials stockés de manière sécurisée
- Les credentials ne sont disponibles que dans la portée de ce bloc
- Jenkins masque automatiquement ces valeurs dans les logs

usernamePassword(credentialsId:

'dockerhubcredential', ...)

- Récupère un credential de type "username/password"
- credentialsId: 'dockerhubcredential' : identifiant du credential configuré dans Jenkins (dans "Manage Jenkins" > "Manage Credentials")
- usernameVariable: 'USERNAME' : le nom d'utilisateur sera accessible via la variable USERNAME
- passwordVariable: 'PASSWORD' : le mot de passe sera accessible via la variable PASSWORD

Pour atteindre sonarqube il faut configurer un token sur sonar et l'enregistre sur jenkins dans Administre jenkins -> credentials ensuite aller dans Administre jenkins -> system pour créer une variable d'environnement configurer l'url de sonar et choisir le token . La variable sera appellée sur la pipeline cicd comsa

withSonarQubeEnv('SonarQubeLocalServer') : fonction qui sert à injecter la variable d'environnement pour ce connecter à sonarqube

mvn clean compile // compile le code

mvn test // génère le rapport jaco après avoir lancer les tests

```
mvn sonar:sonar -s .m2/settings.xml "-Dsonar.projectKey=tech.zerofiltre.testing:calculator" "-Dsonar.host.url=http://109.176.198.187:9000" "-Dsonar.token=squ_560168e21429ecde3798ee92bcfd47b027c3994c" // envoi le rapport jaco à sonar
```

Pour faire une docker build après ça il faut absolument faire une mvn clean package car la mvn test modifie le dossier target et efface le jar généré si ça avait déjà été dans la pipeline il faut faire pareil

Pour que sonar communique avec jenkins donc renvoie son ok il faut configurer un webhook sur sonar et faire pointer l'adresse de jenkins

La pipeline qui fonctionne et qui est simple est sur la branche : feature

Je continue sur la branche main

Ce que je vais ajouter :

1. **Linting du code** - Checkstyle pour vérifier le style // **Le linting est une vérification automatique du style et de la qualité du code avant le build**, pour s'assurer que tout le code respecte les standards définis

Le **linting** est le processus d'analyse automatique du code source pour détecter :

- ✗ Erreurs de syntaxe
- ⚠ Problèmes potentiels
- 🔎 Non-respect des conventions de style

- ✖️ Mauvaises pratiques
- 🔎 Code suspect

Quand on lance la commande mvn checkstyle:checkstyle il ya un fichier sun_checks.xml qui est dans maven qui analyse le code et creer un rapport il est **trop strict** pour la plupart des projets donc en general chacun creer son propre fichier et l'importe dans le fichier pom.xml

☒ Le DevOps NE corrige PAS le code

✗ Ce que le DevOps ne fait PAS :

- Corriger les erreurs de style dans le code Java
- Modifier le code métier des développeurs
- Réécrire les classes/méthodes

✓ Ce que le DevOps fait :

- Configure les pipelines CI/CD
- Met en place les outils (Checkstyle, SonarQube, etc.)
- Définit les règles et seuils de qualité
- Bloque le déploiement si qualité insuffisante
- Génère et partage les rapports

2. **Tests unitaires** - Déjà présent, conservé

3. **Tests d'intégration** - Avec mvn verify

4. **Couverture de code** - JaCoCo avec rapport

5. **Tests de non-régression** - Avant la construction de l'image

Après le build : 6. **Health Check** - Vérification que l'app démarre 7. **Smoke Tests** - Tests basiques incluant le test de connexion et d'addition 8. **Tests d'intégration API** - Tests de toutes les opérations du calculator

Approche 2 : Pipeline NON-BLOQUANTE avec Seuils

(Progressive)

Configuration

```
xml
<plugin>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <configuration>
    <failOnViolation>true</failOnViolation>
    <maxAllowedViolations>10</maxAllowedViolations>
    <!-- Bloque seulement si > 10 violations -->
  </configuration>
</plugin>
```

```

### ### Comportement

- ⚠️ \*\*Warning\*\* si quelques violations (< seuil)
- ✖️ \*\*Échec\*\* si trop de violations (> seuil)
- 📊 \*\*Rapport généré\*\* et publié dans la CI/CD

### ### Stratégie d'amélioration progressive

...

Semaine 1: maxAllowedViolations = 223 (état actuel)

Semaine 2: maxAllowedViolations = 200

Semaine 3: maxAllowedViolations = 150

...

Semaine N: maxAllowedViolations = 0

## Ma recommandation CONCRÈTE pour vous

1. **Maintenant** : Pipeline NON-BLOQUANTE + rapport
2. **Dans 2 semaines** : Activer le blocage avec seuil progressif
3. **Dans 2 mois** : Pipeline strictement bloquante (0 violation)

groovy

```
// Jenkinsfile pragmatique
stage('Checkstyle') {
 steps {
 script {
 def result = sh(script: 'mvn checkstyle:check', returnStatus: true)

 if (result != 0) {
 echo "⚠ ${result} violations Checkstyle"
 // Publier le rapport mais continuer
 publishHTML(...)

 // Bloquer seulement si régression majeure
 if (result > 250) {
 error("🚫 Trop de nouvelles violations!")
 }
 }
 }
 }
}
```

NB : l'approche normale est que quand le code est nouveau:

6. **Maintenant** : Pipeline NON-BLOQUANTE + rapport
7. **Dans 2 semaines** : Activer le blocage avec seuil progressif
8. **Dans 2 mois** : Pipeline strictement bloquante (0 violation)

Dans mon cas j'ai fait une pipeline non blocante + rapport mais en cas d'erreur block la pull request et envoie le mail au dev

NB : CE QUE JE DOIT ETRE CAPABLE DE FAIRE SUR LE PROJECT PARTIE PAR PARTIE:

<https://claude.ai/public/artifacts/72849683-6258-41da-9c41-0581c8b5c129> -> J-A-H-K-0  
<https://claude.ai/public/artifacts/ce1320bc-b3b2-4884-acba-934023beab78> -> J-A-H-K-0-1  
<https://claude.ai/public/artifacts/813647bb-feed-487c-881c-62ab52f25511> -> J-A-H-K-0-2

VOICI LE CANEVA A SUIVRE POUR ETRE COMPLET :

UNE SEUL PIPELINE AVEC LES CONDITION SELON LES BRANCHE LIRE POUR MIEUX COMPRENDRE

<https://claude.ai/public/artifacts/99159b01-0f10-4eed-b233-875d256f58b7> -> J-A-H-K-1

<https://claude.ai/public/artifacts/adfb9712-7842-4f3a-bde4-a4868a5e5ec0> -> J-A-H-K-1

NB : L'EQUIVALEN DE J-A-H-K-1 EST SUR LE PROJECT AVEC LE README QUAND J'AURAI FINIS DE FAIRE SA JE BUCHE JUSTE LE README LA ET C EST BON

Voilà! C'est l'approche standard d'une **moyenne entreprise en Europe** (PME 50-200 personnes).

### Les points clés:

### **1 Repository (app-repo):**

- Code + Helm charts dedans
- Plus simple qu'avoir 2 repos séparés

### **1 Jenkinsfile (la pipeline que j'ai créée):**

- Gère toutes les branches automatiquement
- Détermine quoi faire selon la branche
- Build/Test/Docker, c'est tout
- **Jamais de déploiement direct** sur Kubernetes

### **3 Clusters:**

- Dev (sync automatique)
- Staging (sync manuel)
- Production (sync manuel)

### **ArgoCD:**

- Surveille le repo
- Déploie automatiquement en dev
- Demande approbation pour staging/prod

### **Flux simple:**

feature/\* → Test seulement

develop → Auto-deploy en dev

main → Deploy manuel en staging/prod

NB : CONAITRE LES DIFFERENTS PARTIE DE KUBERNETES (SCEDULER , API SERVER , CONTROLLER ,ETCD) POURVOIR BIEN EXPLIQUER CHACUN

Pourquoi changer : en entreprise je me suis rendu compte après 6 mois 1 an que je ne suis pas assez autonome et je ne me voyais pas changer d'entreprise pour aller avoir le même problème c'est vrai que j'avais déjà les propositions de part et d'autre mais je voulais avoir une certaine autonomie surtout au niveau de l'expérience et aussi des différents contours de l'entreprise savoir comment les choses fonctionnent c'est ce qui m'a poussé à rester et former davantage et aussi travailler sur des projets en parallèle pour grandir encore plus en expérience et maintenant j'ai atteint le niveau que je voulais avoir et je me sens autonome et sûr de pouvoir apporter quelque chose de plus partout où je vais passer et aussi rapporter au poste qui cadre exactement avec mes compétences et ma vision future

### MONITORING:

VOICI CE QUE JE DOIT APPRENDRE :

Exigences :

- Configurer Grafana et Prometheus
- Configurer des tableaux de bord pour :
- Mesures de performance
- Surveillance des erreurs
- Mesures personnalisées
- Configurer des alertes pour :
- Alertes basées sur des seuils
- Détection d'anomalies
- Alertes de requêtes personnalisées

Compétences idéales :

- Maîtrise de Grafana et Prometheus
- Expérience en création de tableaux de bord
- Solide expérience en configuration d'alertes

DANS LE COLLOQUIO JE PEUT DIRE

POUR LA PARTIE :

- Configurer Grafana et Prometheus

- 1) Installation de prometheus e grafana
- 2) Creation d'object service monitor qui vont automatiquement a partir d'un service cible qui expose les donner pouvoir en effet avertir directement promethues (en gros il va recuperer les metrics que expose une application par l'intermediaire de son service) nb : argocd expose nativement le metric a travers son api

1 ER MISSION REALISER :

installation et configuration de prometheus e grafana

Dans la configuration de prometheus : creation d'object service monitor pour recuperer les metrics dans argocd

- Configuration de grafana pour communiquer avec prometheus
- Creation de dashboard dans grafana pour analyser les metric de prometheus

<https://argoproj.github.io/argo-cd/operator-manual/metrics>

CHAPITRE 2 )

Pour le monitoring dans le cluster on a la grande classe des exporters : qui sont les services qu'on installe pour aller lire le service a monitore lire le composant a monitore et l'exposer a prometheus (quand promethues va venir pour recuperer la metric il va interroger l'exporter)

En gros (un exporter est un service que tu installe qui va aller lire le service ou le composant a monitore collecter les metrics et le passer a prometheus)

Exemple: je veux monitore mon cluster j'installe le node-exporter qui vas aller lire tous mon cluster et passer les metrics a prometheus

Je veux monitore ma base de donnee mysql je doit chercher l'exporter qui correspond a lire une base de donnee collecter les metrics et le passer a prometheus

Il ya 2 cas si tu as installer prometheus avec les yaml tu doit modifier le fichier configmap.yaml

Et ajouter seke j'ai mis en bas la mais dans mon cas j'ai installer avec helm donc il fallait juste surcharger le fichier prometheus-values.yaml

Dans mon cas pour installer le node-exporter j'ai juste

Dans le cas ou tu insalles prometheus sans helm le dossier est dans la racine du server -> prometheus-training -> source-> prometheus pour installer le node exporteur ou pour configurer les alert tu modifie le fichier configmap.yaml c'est plus simple comme ça

# 1. Créer ou éditer le fichier values vi prometheus-values.yaml

Ajouter :

prometheus:

  prometheusSpec:

    additionalScrapeConfigs:

      - job\_name: 'node-exporter'

      static\_configs:

        - targets: ['prometheus-prometheus-node-exporter.default.svc.cluster.local:9100']

prometheus-prometheus-node-exporter.default.svc.cluster.local:9100 : non du service node-exporter apres l'avoir installer tu fait juste kubectl get svc -n nom-du-namespace sa va te montrer le service et le port

# 3. Mettre à jour Prometheus avec Helm helm upgrade prometheus prometheus-community/kube-prometheus-stack \ -f prometheus-values.yaml

# 4. Vérifier que le pod Prometheus redémarre kubectl get pods -w | grep prometheus-prometheus

# 5. Vérifier les logs kubectl logs prometheus-prometheus-kube-prometheus-prometheus-0 -c prometheus

Ensuite aller dans grafana -> dashboard -> import metre 1860 clicker sur load c est le numero de la dashboard sava importer la dashboard et les donnees

NB : LE NODE-EXPORTER MONITOR JUSTE L'INGRA EN GENERAL OU LE CLUSTER POUR MONITORE LES API POD SERVICE ET AUTRE IL FAUDRA INSTALLER D'AUTRE EXPORTER

## LISTE DES PRINCIPALES PARTIES MONITORÉES PAR NODE EXPORTER

### LES 4 COMPOSANTS ESSENTIELS

1. **CPU** (Processeur)
2. **MEMORY** (Mémoire RAM)
3. **DISK** (Disque dur)
4. **NETWORK** (Réseau)

### COMPOSANTS SECONDAIRES

1. **FILESYSTEM** (Système de fichiers)
2. **SYSTEM** (Informations système)
3. **LOAD** (Charge système)
4. **SWAP** (Mémoire virtuelle)
5. **PROCESSES** (Processus)
6. **HARDWARE** (Température, ventilateurs)

- Pour monitore argocd : service monitor
- Pour monitore l'infra ou cluster : node exporter
- pour monitore prometheus :

Mise en place de tests automatisés en CI/CD + analyse qualité SonarQube : couverture de tests, détection de vulnérabilités et amélioration continue du code ☺

INSTALLATION DE PROMETHEUS SANS HELM : TU VAS JUSTE DANS LE DOSSIER PROMETHEUS-TRAINING ET TU FAIT LES KUBECTL APPLY TOUT EST DEJA CONFIGURE SI TU VEUX MONITORE ARGOCD TU FAIT JUSTE LES KUBECTL SUR LES YAML DES METRIC QUI SONT DANS GIT TU NE CHANGE RIEN : POUR INSTALLER GRAFANA C EST AVEC HELM VOIR CE QUI S'EST PASSER PRECEDENMENT

kubectl rollout restart deployment/prometheus-deployment -n monitoring

### MONITORING DU CLUSTER KUBERNETES

Chaque microservice expose des metrics  
Avec le monitoring du cluster on pourra savoir si  
Le kubernetes api server  
Le controller manager  
Le scheduler  
l'etcd  
Sont disponible  
Le kube-state-metrics : permet de voir :

- Le nombre de pod demare/arreter/terminer
- Le nombre de fois qu'un pod a redemare
- Analyse le temp de repose des service kubernetes determine :
- Le n point le plus adresser par les user
- Le npoint http le plus lent
- Les requetes qui ont revoyer un code d'erreur

Pour le mettre en place il suffit de modifier le configmap de prometheus  
Mettre le code qu'il ya sur git pour faire pointer prometheus sur le service  
qui a ete deployer (kube-state-metric) dans le cluster

Dashboard qui affiche le nombre de pod vivant et casser et le reste :monitnage cluster default  
Kubernetes (pod,deploy,svc ect ..)

Apiserver : pour monitore les api de kubernetes : tel que

Le kubernetes api server  
Le controller manager  
Le scheduler  
l'etcd

#### MONITORING DES CONTAINER

Cadvisor : service pour monitore les pod  
Dashboard pour monitore container docker : 11277