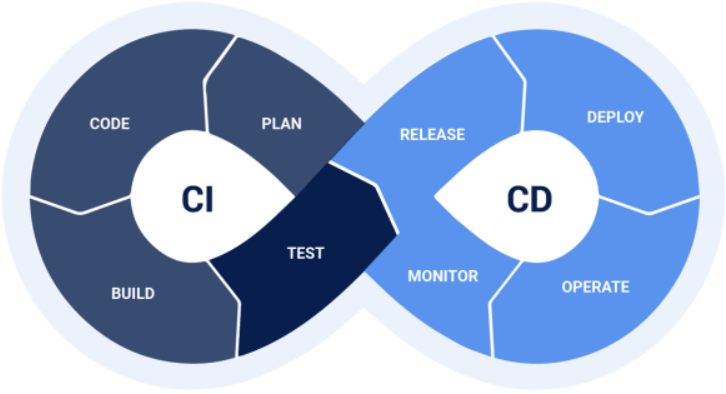
Bob App

Document explicatif



1. Contexte

BobApp est une application Open Source permettant de lire et de partager des blagues.

Afin d'améliorer la qualité de l'application, tout en limitant au maximum le temps humain investi, une démarche de CI/CD a été mise en place.

L'objectif de cette démarche est de :

- Valider que les tests passent sur une pull request ;
- Permette de lancer les tests, vérifier la qualité du code, builder le projet et créer les images Docker ;
- Automatiser la génération des rapports de couverture ;
- Déployer les conteneurs sur Docker Hub.

2. GitHub Actions mises en place

Afin de remplir ces objectifs, un workflow GitHub Actions a été mis en place, qui consiste en 3 jobs :

- ***Analyze-back-end*** :
  - Se base sur la dernière distribution Ubuntu ;
  - Récupère le Repository ;
  - Installe Java 17 ;
  - Lance les tests et génère le rapport JaCoCo ;
  - Publie le rapport de couverture sur GitHub ;
  - Lance le test de qualité SonarQube sur le Back-End ;
  - Publie les résultats de qualité sur le serveur.

- ***Analyze-front-end*** :
  - Se base sur la dernière distribution Ubuntu ;
  - Récupère le Repository ;
  - Installe les dépendances Node.js ;
  - Lance les tests et génère le rapport ;
  - Publie le rapport de couverture sur GitHub ;
  - Installe SonarQube et lance le test de qualité sur le Front-End ;
  - Publie les résultats de qualité sur le serveur.

- ***Build-push-dockers*** :
  - Ne se lance que si les 2 jobs précédents ont réussi ;
  - Se connecte sur DockerHub ;
  - Builde les images Docker du Front-End et du Back-End ;
  - Pushe ces images sur DockerHub.

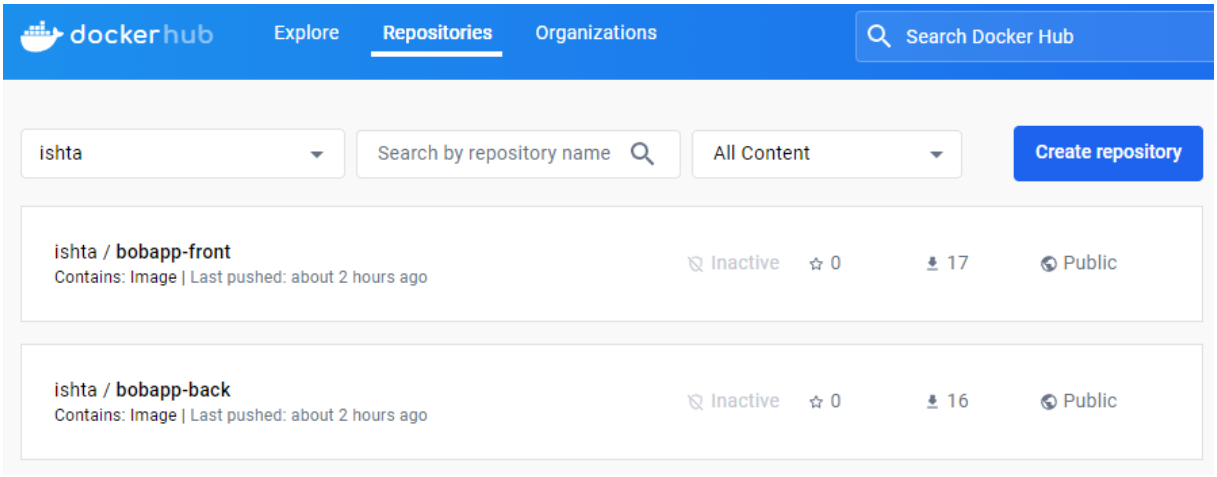
Le Workflow se lance à chaque pull request et à chaque push du propriétaire sur la branche *main*.

L'analyse du Back-End et l'analyse du Front-End échouent si une étape de compilation génère une erreur, si un des tests unitaires échoue, ou si les tests de qualité ne respectent pas les minima définis dans le Quality Gate SonarQube.

Si ces deux analyses réussissent, les images Dockers sont automatiquement buildées et pushées sur le Docker Hub :

https://hub.docker.com/repository/docker/ishta/bobapp-front/general

https://hub.docker.com/repository/docker/ishta/bobapp-back/general



3. Proposition d'indicateurs de performance (*KPI*)

Pour garantir une qualité de code satisfaisante, des seuils objectifs et mesurables doivent être décidés et implémentés dans les différents Quality Gates de SonarQube.

Par défaut, SonarQube propose le Quality Gate « *Sonar way* » :

Nombre de problèmes (*Issues*) du code ajouté	0
Nombre de nouveaux points sensibles (*hotspots*) de sécurité	0
Code Coverage du nouveau code	$\geq 80\%$
Duplicates dans le nouveau code	$\leq 3\%$

Ce Quality Gate se veut être la référence de base d'un code de qualité et facilement maintenable, limitant au maximum les bugs et la dette technique.

Il convient donc parfaitement au cas de Bob App.

Concernant les *Blocker Issues* (litt. « Problèmes bloquants »), il s'agit de problèmes considérés comme suffisamment problématiques par SonarQube pour empêcher le déploiement de l'application en l'état.

> *Exemple : Un projet peut contenir 3 petits problèmes (Issues) et être de bonne qualité alors qu'un autre projet peut avoir un unique gros problème (Blocker Issue) et devoir être corrigé avant son déploiement.*

Dans le cadre de Bob App, il serait effectivement possible de remplacer le nombre d'*Issues* par le nombre de *Blocker Issues*, et ainsi optenir les KPI suivants :

Nombre de problèmes bloquants (Blocker *Issues*) du code ajouté	0
Nombre de nouveaux points sensibles (*hotspots*) de sécurité	0
Code Coverage du nouveau code	≥ 80%
Duplicates dans le nouveau code	≤ 3%

4. Analyse des métriques et avis actuels

4.1. Avis des utilisateurs

L'application est très mal notée sur les réseaux, et deux points de vigilance ressortent des commentaires :

- L'application a au moins un bug bloquant ;
- Les utilisateurs ne se sentent pas écoutés.

4.2. Analyse des métriques obtenues

Le workflow GitHub Actions mis en place génère deux rapports de couverture de tests :

Back-End

bobapp — Sessions

bobapp

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
com.openclassrooms.bobapp.model		0%		n/a	7	7	13	13	7	7	1	1
com.openclassrooms.bobapp.data		49%		50%	5	7	8	18	3	5	1	2
com.openclassrooms.bobapp.service		25%		n/a	1	2	4	7	1	2	0	1
com.openclassrooms.bobapp.controller		54%		n/a	1	2	1	4	1	2	0	1
com.openclassrooms.bobapp		37%		n/a	1	2	2	3	1	2	0	1
Total	90 of 134	32%	2 of 4	50%	15	20	28	45	13	18	2	6

Created with JaCoCo 0.8.5.201910111838

Front-End

All files

76.92% Statements 10/13 **100%** Branches 0/0 **57.14%** Functions 4/7 **83.33%** Lines 10/12

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

Filter:

File	Statements		Branches		Functions		Lines	
app	60%	3/5	100%	0/0	33.33%	1/3	60%	3/5
app/services	87.5%	7/8	100%	0/0	75%	3/4	100%	7/7

SonarQube reprend, quant à lui, les éléments suivants concernant la qualité du code (Quality Gate : *Sonar way*) :

P10_CI_CD_back PUBLIC — Passed

Last analysis: 3 hours ago • **180** Lines of Code • Java, XML

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications
D 1	A 0	E 0.0%	A 8	38.8%	0.0%

P10_CI_CD_front PUBLIC — Passed

Last analysis: 3 hours ago • **161** Lines of Code • TypeScript, HTML, ...

Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications
A 0	A 0	A —	A 1	47.6%	0.0%

Le Front-End est de bonne qualité, même si le Code Coverage demeure largement insuffisant.

Le point faible de l'application est le Back-End, qui contient un bug, deux points sensibles de sécurité, et un Code Coverage nettement insuffisant.

Remarque : les valeurs de Code coverage sont différentes entre les tests unitaires et SonarQube, chaque logiciel ayant sa manière propre de le calculer.

4.3. Conclusions

Afin d'améliorer la qualité de l'application Bob App, il faut en priorité :

- Corriger les bugs de l'application, particulièrement sur le Back-End ;
- Améliorer largement le Code Coverage du Front-End et du Back-End ;
- Rediriger le *report* des bugs depuis une adresse e-mail vers le repository GitHub, afin que la communauté Open Source puisse s'en saisir et les corriger.