 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 1	PAGES 12

Département de génie logiciel et des TI

TP2

MGL842

Implémentation du processus de qualité [DevOps] Description des tâches

Auteur

BABA MOHAMMED SOORI
FRANK JUNIOR POUNGOUE SIWETCHEU

Professeur

FABIO PETRILLO

Date

6 avril 2024



 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 2	PAGES 12

Table des matières

1. PRÉSENTATION / INTRODUCTION.....	2
2. MODÉLISATION DU PROCESSUS DE QUALITÉ.....	3
3. GUIDE DE CONFIGURATION ET DÉPLOIEMENT.....	3
4. STANDARD ET CHECKLISTS DE QUALITÉ DU CODE.....	3
5. ÉVALUATION DU PROCESSUS DE QUALITÉ.....	3

 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 3	PAGES 12

1. PRÉSENTATION / INTRODUCTION

Bienvenue dans la documentation du projet PR Reviewer, une aventure débutée lors de notre cours MGL870 en génie logiciel. Ce projet initial visait à explorer l'intégration des modèles de langage à grande échelle (LLMs) dans le cadre d'une utilisation innovante pour les revues de code en DevOps.

Avec PR Reviewer, nous avons créé une application capable d'interagir avec OpenAI, automatisant la révision de code pour les Pull Requests (PRs). Cette application, développée en Python, a démontré la faisabilité et l'intérêt d'intégrer l'intelligence artificielle dans l'évaluation des modifications de code, tout en prodiguant des conseils d'amélioration.

Le projet original peut être exploré à travers le dépôt GitHub suivant : [PR Reviewer Repo MGL870](#).

Dans le prolongement de ces travaux et dans le cadre du cours MGL842, nous avons entrepris le développement d'une nouvelle version de PR Reviewer. Ce nouveau volet se concentre sur l'amélioration de la qualité du code, l'enrichissement des tests, l'intégration de fonctionnalités de logging et la construction d'une API déployable sur AWS, marquant ainsi notre progression dans le domaine du DevOps axé sur la qualité.

La nouvelle incarnation de notre application est accessible via le dépôt GitHub : [PR Reviewer Repo MGL842](#).

Le présent document fait état de notre démarche DevOps affinée et du modèle de qualité élaboré. Vous y trouverez des instructions détaillées pour la configuration et le déploiement, les normes de qualité du code à respecter, ainsi que l'évaluation du processus de qualité que nous avons mis en place, basée sur les critères ISO 25010 et les indicateurs DORA.

2. MODÉLISATION DU PROCESSUS DE QUALITÉ

La Figure 1 présente notre processus de qualité sous forme de diagramme BPMN. Ce diagramme illustre les étapes systématiques et les flux de décision entrepris depuis l'identification d'une contribution jusqu'à la surveillance de l'application en production.

ÉTS Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 4	PAGES 12

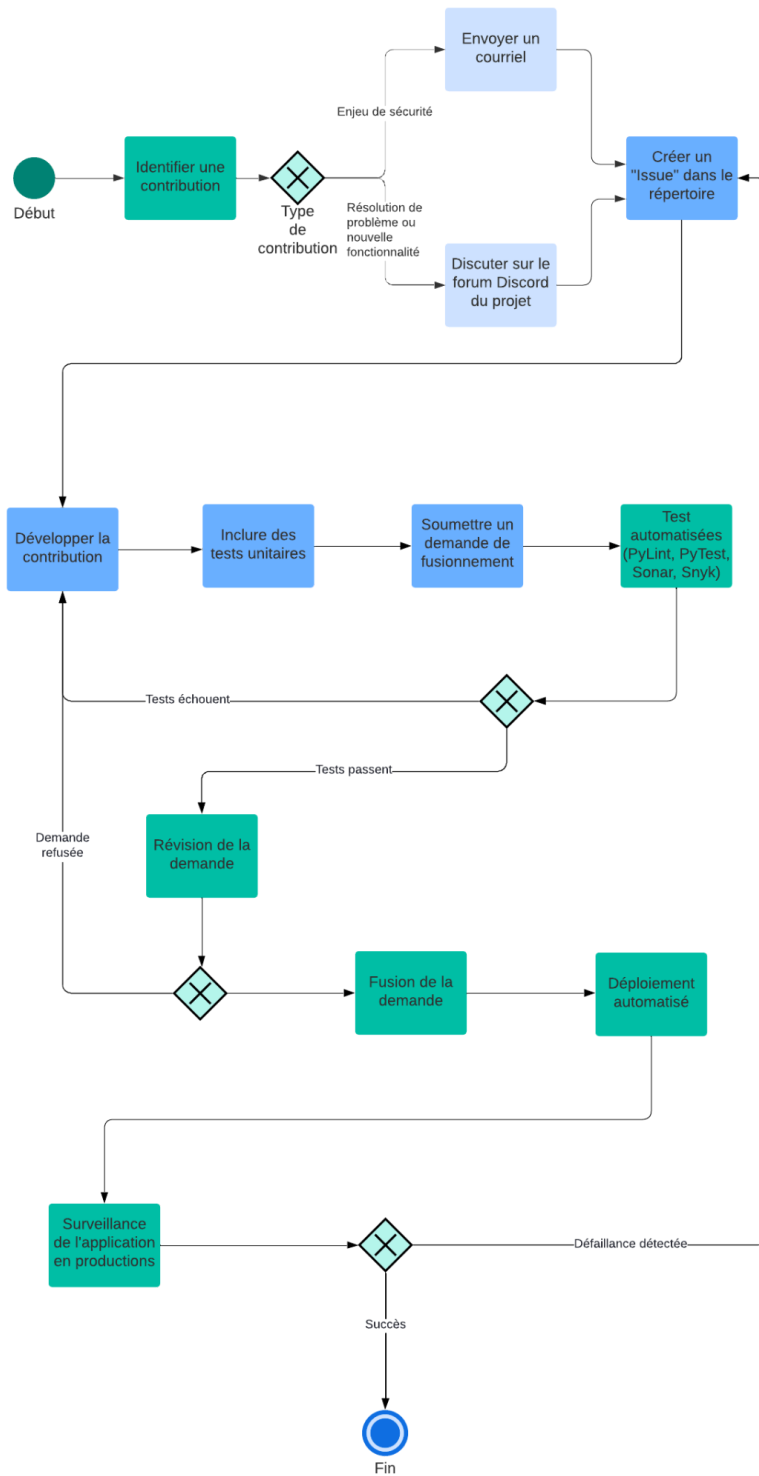




Figure 1 - Processus de qualité

 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 5	PAGES 12

Chaque étape a été pensée pour maximiser la qualité tout en optimisant l'efficacité :

- **Identification de contribution:** Le point de départ pour toute modification ou amélioration, encourageant une documentation et un suivi clairs dès le début.
- **Discussions et résolutions de problèmes:** En utilisant des plateformes collaboratives comme Discord, nous garantissons la transparence et la discussion communautaire, des aspects essentiels pour la résolution collaborative des problèmes.
- **Intégration de tests unitaires:** Un pilier dans notre engagement pour la qualité, assurant que chaque contribution soit vérifiée et fiable.
- **Automatisation des tests et déploiements:** Les outils comme PyLint, PyTest, Sonar et Snyk permettent une évaluation automatisée et continue, ce qui améliore la qualité du code.
- **Surveillance et observabilité:** La surveillance englobe les logs, les performances de l'application et la gestion des performances applicatives (APM). Ce qui nous permet d'avoir une vue complète de l'état de notre application. Cela comprend la surveillance des erreurs de runtime, le profiling des performances et le suivi des métriques clés, garantissant que nous pouvons réagir rapidement à tout problème qui surviendrait.

Pour guider et standardiser les contributions au projet PR Reviewer, nous avons inclus un fichier CONTRIBUTING.md dans notre répertoire GitHub. Ce document détaillé sert de boussole pour les développeurs, leur fournissant toutes les informations nécessaires pour contribuer au projet avec efficacité, tout en établissant des normes claires et cohérentes. Parallèlement, nous utilisons les "issues" de GitHub pour une gestion transparente des demandes des utilisateurs, permettant de suivre, catégoriser et prioriser les améliorations proposées ainsi que les éventuels bugs. Pour la gestion holistique du projet, GitHub Project s'avère être un outil précieux, fonctionnant comme un tableau de bord centralisé qui articule les "issues" et les tâches internes, ce qui est essentiel pour une organisation et une hiérarchisation efficace du travail. La visualisation de notre backlog, qui sera présentée dans la figure 2 , offre un aperçu de notre méthode de gestion du projet, illustrant notre processus de travail et notre approche de la priorisation au sein du projet.

 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 6	PAGES 12

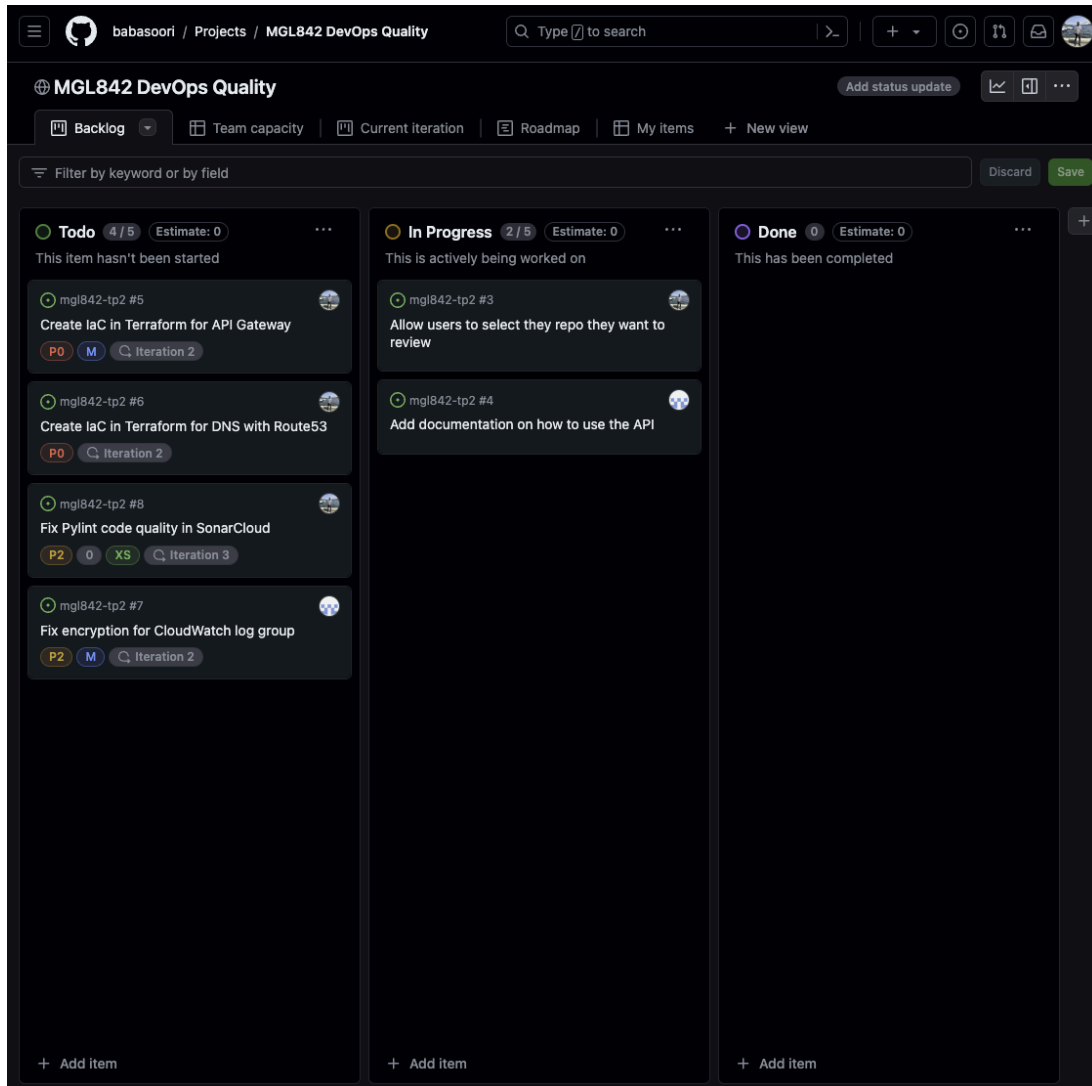



Figure 2 - Projet Github pour la gestion de projet

La figure 3 offre une vue d'ensemble de l'architecture de notre pipeline DevOps et des outils utilisés à chaque étape. Cette représentation illustre la manière dont les discussions, le contrôle de version, la construction, les tests, le déploiement et la surveillance s'intègrent de manière cohérente dans notre processus de développement :

 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 7	PAGES 12

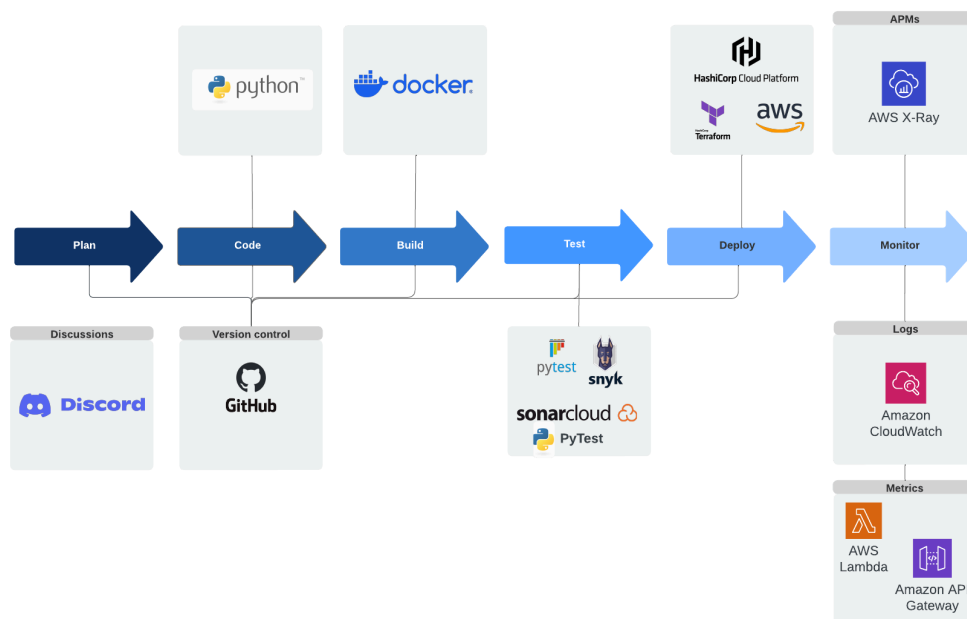



Figure 3 - Processus DevOps

- **Planification et codage:** Avec GitHub au cœur de notre contrôle de version, nous structurons nos efforts de développement autour de meilleures pratiques de planification et de collaboration.
- **Construction et tests:** Docker est utilisé pour créer des conteneurs, tandis que les tests automatisés garantissent que notre code reste sain et robuste.
- **Déploiement et surveillance:** Terraform, Hashicorp Cloud et AWS orchestrant le déploiement, nous avons une approche qui favorise une mise en production fluide et surveillée.

Dans la phase de planification et codage, nous avons mis en place une communauté sur Discord pour permettre des discussions approfondies sur les problèmes et les fonctionnalités envisagées. Cet espace permet aux membres de la communauté de collaborer, partager des idées et obtenir un retour sur leurs contributions, créant ainsi une synergie qui alimente l'innovation et l'amélioration continue du projet.

Durant la phase de construction et tests, une série d'outils automatisés sont utilisés pour garantir la qualité du code et sa robustesse. Pylint est mis à contribution pour l'analyse statique du code, permettant de détecter les erreurs de style, les bugs et les problèmes potentiels de manière précoce. PyTest est utilisé pour exécuter les tests unitaires, s'assurant que les nouvelles fonctionnalités et les corrections de bugs répondent aux exigences avant l'intégration dans la base de code principale. SonarCloud intervient pour une inspection continue de la qualité du code et Snyk pour la détection des vulnérabilités de sécurité, offrant ainsi une assurance supplémentaire de l'intégrité de notre code.

 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 8	PAGES 12

Pour ce qui est du déploiement et de la surveillance, nous utilisons Terraform, fourni par HashiCorp Cloud, pour créer et gérer l'infrastructure de manière déclarative et reproductible, facilitant ainsi les déploiements dans AWS. La surveillance est assurée grâce à AWS CloudWatch et AWS X-Ray, nous donnant une visibilité en temps réel sur les logs, les performances de l'application et les métriques opérationnelles, ce qui est essentiel pour maintenir la fiabilité et la disponibilité du système.

Ces décisions modélisent notre engagement envers une culture DevOps qui ne transige pas avec la qualité, chaque outil et pratique étant choisi pour son impact positif sur le produit final.

3. GUIDE DE CONFIGURATION ET DÉPLOIEMENT

Cette partie du rapport explique les étapes de configuration et de déploiement de l'application PR Reviewer, référencée en tant que figure 4, qui détaille l'architecture sur laquelle l'application est construite.

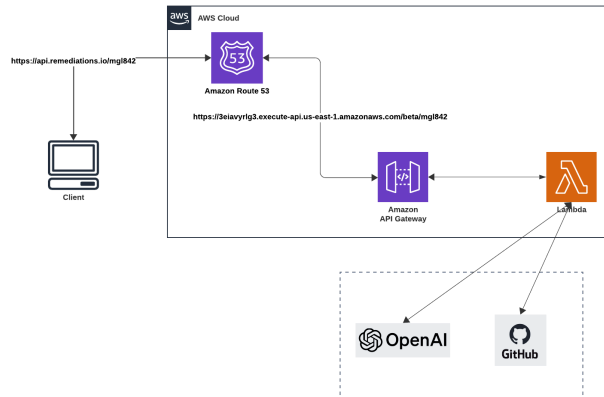


Figure 4 -Architecture cloud de PR Reviewer


Configuration initiale

Le répertoire .workflows, qui contient notre pipeline DevOps orchestré par GitHub Actions.

Intégration des applications GitHub

Notre configuration intègre trois applications GitHub clés: Hachicorp Cloud, SonarCloud et Snyk. Pour les intégrer à nos workflows GitHub Actions, nous avons configuré des secrets de dépôt qui servent de variables d'environnement sécurisées :

- **SNYK_TOKEN**: Pour les analyses de sécurité de Snyk.

 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 9	PAGES 12

- **SONAR_TOKEN**: Pour l'inspection continue de la qualité du code avec SonarCloud.
- **TF_API_TOKEN**: Pour gérer l'infrastructure avec Terraform Cloud.

Déploiement via HashiCorp Cloud

Le déploiement s'effectue à travers HashiCorp Cloud, qui gère notre infrastructure AWS. Dans votre espace de travail HashiCorp, définissez un ensemble de variables sensibles pour les informations d'identification AWS :

- **AWS_ACCESS_KEY_ID** et **AWS_SECRET_ACCESS_KEY**: Ces clés d'accès sont traitées comme des variables sensibles et sont cruciales pour l'authentification et la gestion sécurisée des ressources AWS.

Configuration des Variables Sensibles Terraform

Configurez également les variables sensibles suivantes dans Terraform pour l'interaction avec les services externes :

- **assistant_id**: Identifiant unique pour l'assistant OpenAI.
- **authorized_repo**: Le répertoire GitHub spécifique pour les revues de code.
- **my_github_token**: Token d'authentification pour interagir avec les PR GitHub.
- **openai_api_key**: La clé d'API nécessaire pour les requêtes vers OpenAI.

Ces variables assurent la communication sécurisée entre notre application, OpenAI et GitHub, et sont indispensables pour une revue de code automatisée et efficace.

Suivez ces instructions pour déployer l'application PR Reviewer dans l'environnement AWS, profitant ainsi de la puissance d'OpenAI pour les revues de code.

4. STANDARD ET CHECKLISTS DE QUALITÉ DU CODE

Standards de codage


Nous adhérons aux standards de codage PEP 8, qui promeut la lisibilité et la cohérence dans le code Python. L'utilisation de Pylint nous permet de nous assurer que notre code est non seulement fonctionnel mais aussi propre et conforme à ces conventions. Les règles et recommandations détaillées peuvent être consultées sur [la page officielle de Pylint](#) et [le guide PEP 8](#).

Politiques de contrôle de version

Pour le contrôle de version, nous employons les meilleures pratiques de GitHub, avec des branches dédiées pour les fonctionnalités, les corrections de bugs et les mises en production. Chaque Pull Request doit être associée à une issue pour en assurer la traçabilité.

Pratiques de révision de code recommandées

SonarCloud nous assiste dans le maintien d'un code propre et de qualité avec ses standards de "Clean Code", qui valorise la cohérence, l'intentionnalité, l'adaptabilité et la responsabilité. Pour plus d'informations, consultez [la définition de Clean Code chez SonarSource](#). Nous suivons également [les Quality Gates par défaut de SonarCloud](#) pour garantir la sécurité, la fiabilité et la

 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 10	PAGES 12

maintenabilité du code. La figure 5 illustre la qualité de notre projet en utilisant les standards de code utilisés.

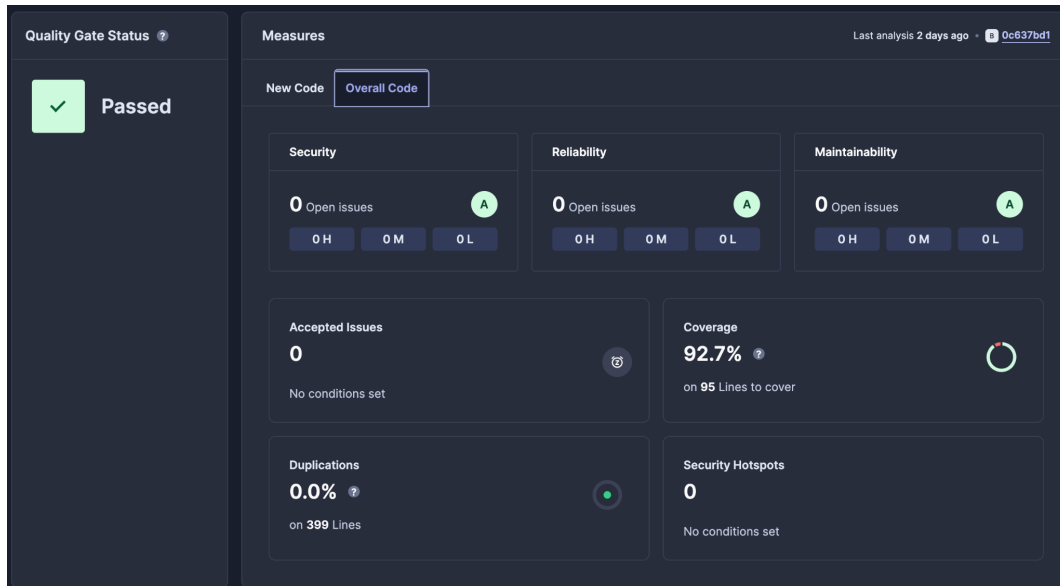


Figure 5 - Rapport de qualité pour PR Reviewer

Politiques de logging et de monitoring

Pour le logging, nous adoptons quatre niveaux : debug, warning, error et info, chacun correspondant à des besoins spécifiques de traçabilité et d'alerte. Les logs sont stockés et visibles dans AWS CloudWatch, tandis que les métriques de performance sont disponibles directement sur AWS Lambda et API Gateway. Pour un suivi détaillé, AWS XRay fournit une traçabilité approfondie de l'API.


Sécurité et conformité

Snyk joue un rôle essentiel dans l'application des meilleures pratiques de sécurité pour l'Infrastructure as Code (IaC) sur AWS, conformément à diverses normes de conformité telles que les CIS Benchmarks, PCI, SOC 2, etc. Pour plus de détails, veuillez consulter [la documentation Snyk pour IaC](#). Snyk est également utilisé pour analyser le code Python à la recherche de vulnérabilités, comme expliqué dans leur documentation pour Python.

En complément de ces outils automatisés, les mainteneurs du projet examinent manuellement les PR pour assurer la cohérence et le respect des conventions établies.

5. ÉVALUATION DU PROCESSUS DE QUALITÉ

Notre démarche d'évaluation du processus de qualité s'appuie sur les indicateurs de performance DORA ainsi que sur les critères de la norme ISO/IEC 25010. Voici comment notre projet se mesure à ces standards.

 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 11	PAGES 12

Évaluation DORA

- **Intégration continue** : GitHub Actions automatise la construction et les tests pour chaque commit, assurant une qualité et une préparation constantes du code pour le déploiement.
- **Livraison continue** : Avec Terraform Cloud et AWS, la gestion de l'infrastructure en tant que code permet des déploiements fluides grâce à des changements d'infrastructure cohérents et révisables.
- **Fiabilité** : L'architecture sans serveur d'AWS Lambda et API Gateway assure une mise à l'échelle automatique et une fiabilité du système grâce à un suivi performant.
- **Développement basé sur le tronc** : L'utilisation de branches principales pour les fonctionnalités et corrections de bugs, associées à des pull requests, permet une intégration et des révisions de code approfondies.
- **Infrastructure flexible** : AWS Lambda et API Gateway supportent des services évolutifs, favorisant le déploiement rapide et l'actualisation du système.
- **Contribution AI** : L'application PR Reviewer utilise l'IA pour automatiser les révisions de code, enrichissant le flux de travail des développeurs avec des analyses intelligentes.

Évaluation ISO/IEC 25010

- **Fiabilité** : AWS Lambda et API Gateway offrent une disponibilité continue, assurant une accessibilité ininterrompue aux services.
- **Sécurité** : L'implémentation de certificats TLS et l'utilisation de variables d'environnement pour les secrets garantissent des communications sécurisées et la confidentialité.
- **Maintenabilité** : L'intégration de Pytest, SonarCloud et Snyk améliore la testabilité et assure une haute qualité du code avec une détection précoce des vulnérabilités.

Temps d'exécution des Outils


L'exécution de l'analyse SonarCloud, incluant Pylint et Pytest avec couverture, prend 1 minute et 5 secondes. Le scan IaC de Snyk dure 20 secondes, et l'analyse Python de Snyk prend 37 secondes.

En complément de notre évaluation, nous observons également l'efficacité de notre chaîne d'intégration et de déploiement continue. Une fois une Pull Request fusionnée, GitHub Actions et HashiCorp Cloud prennent le relais pour automatiser le déploiement. La construction de l'archive ZIP, qui contient à la fois le code source et les dépendances nécessaires à AWS Lambda, est exécutée en 23 secondes. Le déploiement proprement dit prend 1 minute et 26 secondes, un témoignage de la réactivité et de l'efficacité de notre pipeline DevOps.

Cette rapidité de déploiement, associée à la robustesse du processus de qualité mis en place, nous permet de garantir une mise en production fiable et sécurisée, en adéquation avec les principes établis par les normes ISO/IEC 25010 et les indicateurs DORA (voir figure 3 pour les détails du processus de déploiement).

Surveillance et Performance

Pour le logging, nous exploitons AWS CloudWatch, qui récolte des logs détaillés, incluant quatre niveaux de sévérité : debug, warning, error et info. Les métriques de performance sont

 Département de génie logiciel et des TI	COURS MGL842	DOCUMENT NO.	DATE 2024-04-06	VERSION 1.0
	TITRE TP2		PAGE 12	PAGES 12

Département de génie
logiciel et des TI

DOCUMENT NO.

VERSION
1.0

TITRE

TP2

PAGES	
	12

directement accessibles via AWS Lambda et API Gateway. Des traces plus détaillées sont disponibles grâce à AWS XRay, offrant une visibilité accrue sur l'application (voir les figure 5 et 6 pour les logs CloudWatch et les traces XRay).

Traces (19)

This table shows traces with updates within the last 30 minutes, with an average response time of 1.38s. It shows as many as 1000 traces.

ID	Trace status	Timestamp	Response code	Response Time	Duration	HTTP Method	URL Address
...2a8b788c1820841c271fbeb52	Error (4xx)	18.3min (2024-04-07 19:02:31)	403	0.001s	0.001s	GET	https://api.remediations.io/favicon.ico
...32d5e9f725edab9651f559831	Fault (5xx)	18.3min (2024-04-07 19:02:30)	502	1.317s	1.317s	GET	https://api.remediations.io/mgl842
...69481e5b314e450056f55a1e	Error (4xx)	18.5min (2024-04-07 19:02:17)	403	0.039s	0.039s	GET	https://api.remediations.io/mgl842
...0dd0d069516a15332072b877	Error (4xx)	18.5min (2024-04-07 19:02:17)	403	0.001s	0.001s	GET	https://api.remediations.io/favicon.ico
...6c68a74c73994439d18bae12e1	Error (4xx)	18.6min (2024-04-07 19:02:11)	403	0s	0s	GET	https://api.remediations.io/favicon.ico
...51543437277e7434c1748782a	Error (4xx)	18.6min (2024-04-07 19:02:11)	403	0.02s	0.021s	GET	https://api.remediations.io/mgl842
...783d183b64d377166980c931	Error (4xx)	18.7min (2024-04-07 19:02:04)	403	0.018s	0.032s	GET	https://api.remediations.io/mgl842
...343d9bac126db9674db563172	Error (4xx)	18.7min (2024-04-07 19:02:04)	403	0.001s	0.001s	GET	https://api.remediations.io/favicon.ico
...663dea1631d68aff73dfff1f	Error (4xx)	18.9min (2024-04-07 19:01:56)	403	0.001s	0.001s	GET	https://api.remediations.io/favicon.ico
...35e21f4fd4c80e240f5f5e50	OK	19.1min (2024-04-07 19:01:45)	200	10.705s	10.724s	GET	https://api.remediations.io/mgl842
...15db07cb6cdfd1d918ac1898	Error (4xx)	19.1min (2024-04-07 19:01:41)	403	0.001s	0.001s	GET	https://api.remediations.io/favicon.ico
...7091b7f959e80cb7f49257f3	OK	19.3min (2024-04-07 19:01:29)	200	11.411s	11.428s	GET	https://api.remediations.io/mgl842
...5c1896ba31e03c7163c07ba3	OK	19.5min (2024-04-07 19:01:20)	200	0.043s	0.078s	GET	https://api.remediations.io/mgl842
...79f659b39f64518145440e26	Error (4xx)	19.6min (2024-04-07 19:01:15)	403	0.001s	0.001s	GET	https://api.remediations.io/favicon.ico
...4ea8cc016d679dbd550e1deb7	Error (4xx)	19.6min (2024-04-07 19:01:15)	403	0.001s	0.001s	GET	https://api.remediations.io/mgl842/match

Figure 6 - Traces détaillées de l'API

[illegible]

Figure 7 - Logs détaillés de l'API

En utilisant ces méthodologies et outils, nous avons établi un processus de qualité robuste qui se conforme aux standards internationaux, tout en bénéficiant d'une mesure et d'une visualisation précise de la performance de notre application.