Architecture partie Devops

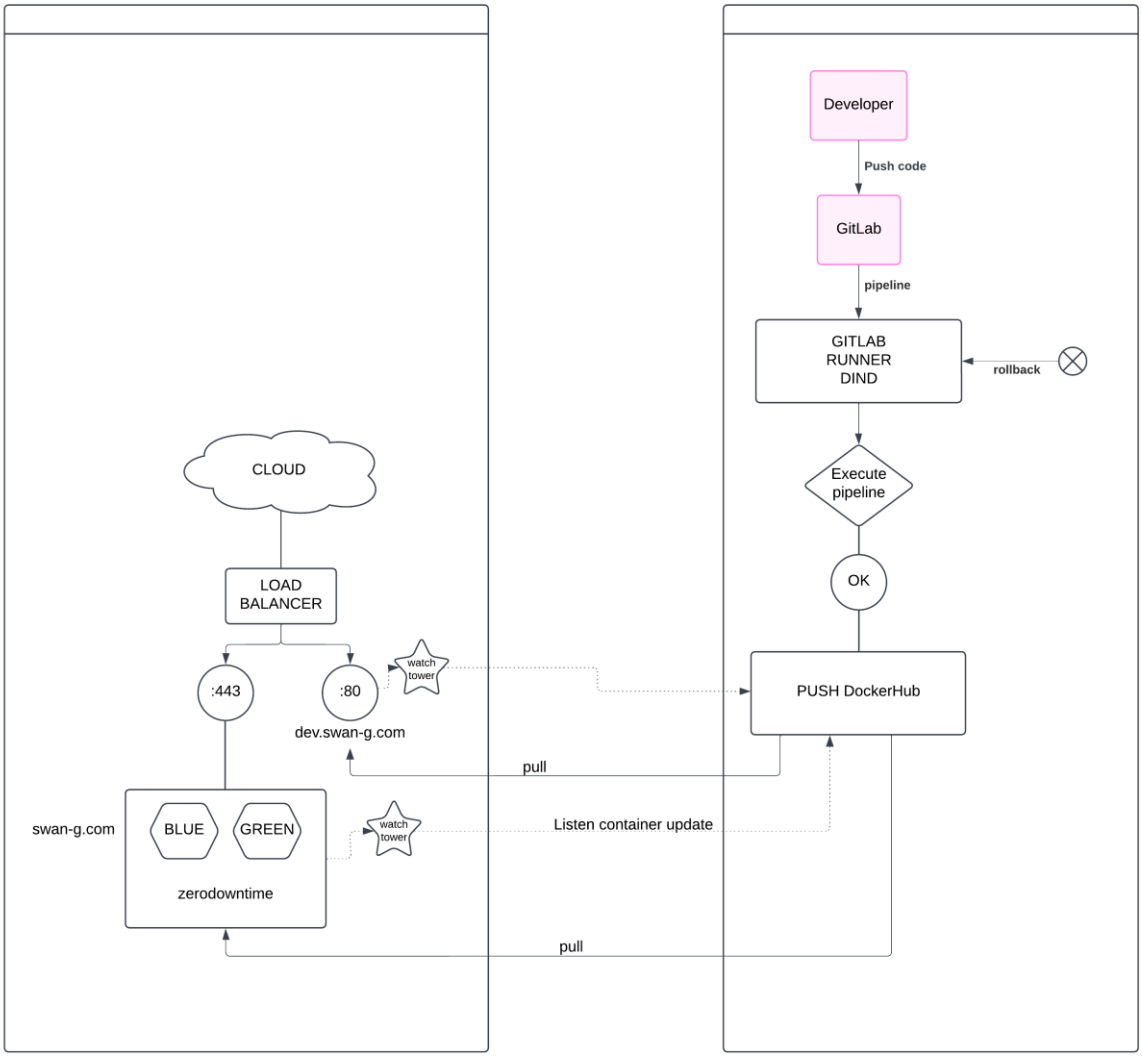
1. Introduction

Cette documentation détaille l’architecture DevOps conçue pour le projet T-DEV-700, les interactions entre ses différents composants et les étapes clés de la pipeline CI/CD. Elle couvre l'ensemble du processus, depuis la phase de développement avec les tests unitaires jusqu'au déploiement en production, le tout en intégrant une stratégie Blue/Green pour garantir un déploiement sans interruption. Les relations entre les services, comme le backend, le frontend, les runners CI/CD et le load balancer NginX, y seront également décrites.

Le DevOps occupe une place centrale dans ce projet. Il nous permet de standardiser les environnements, d’automatiser les workflows et d’assurer une continuité de service grâce à des outils comme Docker pour la conteneurisation et GitLab CI/CD pour les pipelines. L’objectif est de combiner performance, fiabilité et scalabilité, tout en

minimisant les risques d’erreurs humaines et en simplifiant la gestion des environnements.

2. Vue d’ensemble de l’architecture DevOps



Résumé des choix technologiques :

Docker a été choisi pour sa capacité à créer des environnements isolés et reproductibles, ce qui est essentiel pour un projet comprenant des développeurs utilisant des machines et systèmes variés. Cette technologie permet de gérer facilement les environnements backend et frontend, que ce soit en développement ou en production, tout en réduisant les risques liés à des configurations différentes.

L'intégration des **runners GitLab**, configurés pour répondre aux besoins des environnements dev et prod, garantit une exécution rapide et sécurisée des jobs.

La **stratégie Blue-Green** apporte une dimension de fiabilité supplémentaire. En répartissant le trafic entre deux environnements (Blue et Green), elle permet des déploiements sans interruption et offre une capacité immédiate de rollback en cas de problème. Ce choix réduit les risques d'indisponibilité tout en assurant une transition fluide entre les versions. Cette combinaison de technologies offre une architecture DevOps performante et adaptée aux objectifs du projet.

3. Pipeline CI/CD

Étapes clés :

Le pipeline CI/CD automatise les étapes essentielles du développement au déploiement, tout en garantissant cohérence et fiabilité. Il commence par la construction des images Docker pour le backend et le frontend, suivie de tests automatisés (unitaires, intégration, bout en bout) déclenchés après chaque push ou merge request.

Le déploiement se fait d'abord sur l'environnement de développement (dev) pour validation, avant de passer en production via une stratégie Blue-Green. Le trafic est initialement dirigé vers l'environnement Green pour vérification, puis basculé via le load balancer NginX. En cas de problème, un rollback rapide permet de revenir à l'environnement précédent, assurant une continuité de service. Enfin, Grafana surveille les performances pour garantir la stabilité après déploiement.

Variables CI/CD :

Les secrets sont configurés directement dans les Variables CI/CD de GitLab. Ces variables sont protégées et masquées, empêchant leur affichage dans les logs du pipeline. De plus, elles sont accessibles uniquement aux branches protégées, limitant ainsi les risques d'exposition accidentelle.
Elles sont référencées comme variables d'environnement dans le fichier .gitlab-ci.yml.

4. Interactions entre les services

Les différents services interagissent de manière cohérente pour garantir une gestion fluide des processus de Dev, test, et déploiement. Ces interactions sont centralisées autour des runners GitLab et du load balancer NginX.

Le backend, basé sur Deno, est au cœur de l'architecture API. Il interagit directement avec la base de données pour gérer les données de l'application et dépend des runners CI/CD pour automatiser ses tests et ses déploiements. Chaque modification du backend est validée via des tests automatisés avant d'être déployée dans des conteneurs Docker.

Le frontend utilise React/Next.js, avec un processus de build automatisé déclenché par les pipelines GitLab.

Les GitLab runners, configurés en Docker-in-Docker, orchestrent les étapes de la pipeline. Chaque runner est associé à des tags (dev, prod) pour exécuter les jobs correspondant aux environnements appropriés.

Le load balancer NginX gère la répartition du trafic entre les environnements Blue et Green. Lors d'un déploiement, le pipeline CI/CD met à jour l'un des deux environnements, et une fois validé, le trafic est redirigé vers l'environnement mis à jour. Grâce à NginX, si l'un des environnements est indisponible, il est automatiquement exclu de la répartition du trafic, assurant ainsi une continuité de service. Cette intégration avec les pipelines garantit des déploiements fluides et fiables.

5. Conclusion

L'architecture DevOps mise en place dans ce projet illustre une combinaison efficace d'automatisation, de sécurité et de flexibilité. Grâce à des outils modernes comme Docker et GitLab CI/CD, ainsi qu'à une stratégie de déploiement Blue-Green soutenue par un load balancer NginX, nous avons construit un système capable de gérer les déploiements sans interruption tout en garantissant une transition fluide entre les environnements.

Chaque composant joue un rôle essentiel dans cette architecture. Les interactions entre ces services, optimisées par des pipelines rigoureux, assurent non seulement une livraison continue mais également une sécurité renforcée. Cette structure évolutive permet de répondre aux besoins actuels tout en offrant une base solide pour les développements futurs du projet.