Table des matières

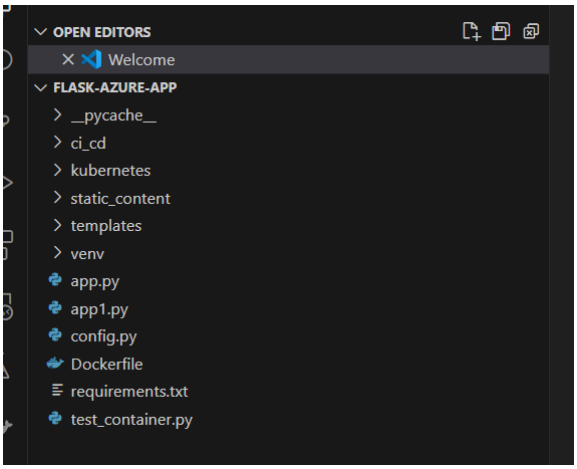
Projet : Plateforme de diffusion de contenu statique basée sur le cloud

Vous êtes chargé de développer une plateforme basée sur Flask qui diffuse dynamiquement du contenu statique stocké sous format JSON/YAML, fonctionnant sur Azure Kubernetes Service (AKS) avec un pipeline CI/CD.

Exigences et composants du projet

Gestion de version avec Git :

Utiliser Git pour suivre les modifications du code et des configurations.
Gérer les branches de fonctionnalités, les pull requests et appliquer les bonnes pratiques Git.



Inclure README.md.
Inclure un fichier .gitignore pour exclure les fichiers sensibles ou inutiles.

Questions à explorer :

- *Comment structurer votre référentiel Git pour assurer un workflow clair et maintenable ?*
- *Quelle stratégie de branchement (Git Flow, Trunk-based development) soutiendra le mieux votre pipeline CI/CD ?*
- *Quelles règles .gitignore devez-vous appliquer pour éviter de commettre des fichiers sensibles ?*

Git Flow (Idéal pour les équipes)

- Utilise deux branches principales : main (production stable) et develop (intégration/tests).
- Le développement des fonctionnalités se fait sur des branches de fonctionnalités, qui sont fusionnées dans develop après validation.
- Les versions sont stabilisées dans des branches de release, puis fusionnées dans main lorsqu'elles sont prêtes.

Pourquoi ? Prend en charge les pipelines CI/CD en gardant develop pour les tests et main pour les déploiements stables.

Trunk-Based Development (Idéal pour un déploiement rapide)

- Utilise une seule branche principale (main) avec des branches de fonctionnalités de courte durée.

- Les développeurs effectuent des pushs fréquents et utilisent des feature flags pour gérer les modifications.
- Favorise l'intégration continue (CI), ce qui est idéal pour des itérations rapides.

Pourquoi ? Fonctionne bien avec CI/CD, car il permet d'envoyer en continu des modifications petites et testables.

Application Flask :

Développer une application Flask qui :
1. Lit du contenu statique (événements, actualités, FAQ) à partir de fichiers JSON ou YAML.
2. Diffuse le contenu via des endpoints API REST :

Endpoint	Méthode	Description
/api/events	GET	Récupère les dernières données d'événements en JSON/YAML
/api/news	GET	Récupère les dernières actualités en JSON/YAML

3. Inclut une interface simple pour afficher dynamiquement les données.

Questions à explorer :

- *Pourquoi utiliser Azure Blob Storage au lieu d'un stockage local ?*
- *Quelles méthodes Azure SDK ou APIs peuvent être utilisées pour récupérer les fichiers JSON/YAML en temps réel ?*
- *Comment garantir un accès à faible latence aux fichiers statiques ?*

Conteneurisation avec Docker :

Rédiger un Dockerfile pour conteneuriser l'application Flask.
Tester le conteneur localement pour garantir son bon fonctionnement.

Questions à explorer :

Quelles optimisations Dockerfile permettent d'améliorer le temps de build et de réduire la taille de l'image ?

Orchestration avec Kubernetes :

Déployer l'application conteneurisée sur Azure Kubernetes Service (AKS).
Définir des manifests Kubernetes pour :

1. Deployment : Exécuter le conteneur Flask.
2. Service : Exposer l'application aux utilisateurs.
3. ConfigMaps : Gérer la configuration, y compris l'emplacement des fichiers dans Azure Blob Storage.

Questions à explorer :

- *Comment définir des manifests Kubernetes pour le déploiement, l'exposition du service et la gestion des configurations ?*
- *Quels éléments prendre en compte pour la scalabilité au sein d'AKS ?*

Pipeline CI/CD :

Configurer un pipeline CI/CD avec GitHub Actions, Azure DevOps, ou un outil similaire.

Étapes du pipeline :

1. Build : Construction de l'image Docker.
2. Test : Exécution de tests automatisés sur l'application Flask.
3. Déploiement : Poussée de l'image vers Azure Container Registry (ACR) et déploiement sur AKS.

Questions à explorer :

- *Comment automatiser la construction, les tests et le déploiement des conteneurs dans le pipeline CI/CD ?*
- *Quelles mesures de sécurité doivent être mises en place pour le déploiement dans Azure Container Registry (ACR) ?*
- *Comment gérer les rollbacks en cas d'échec du déploiement ?*

Surveillance et journalisation :

Utiliser Azure Monitor pour suivre la performance et la santé de l'application. Intégrer un système de logs pour collecter des données en temps réel.

Questions à explorer :

- *Comment suivre la performance de l'application dans Azure Monitor ?*
- *Quelle stratégie de journalisation adopter pour faciliter le débogage ?*

- *Comment appliquer des politiques d'auto-scaling dans Kubernetes pour gérer la charge de trafic ?*

Mise à l'échelle et équilibrage de charge :

Configurer Kubernetes pour mettre à l'échelle dynamiquement l'application Flask.
Utiliser un Ingress Controller pour gérer le trafic.

Sécurité :

Utiliser Azure Managed Identities pour sécuriser l'accès à Blob Storage.
Configurer Kubernetes Secrets pour stocker des données sensibles (clés API, tokens).

Questions à explorer :

- *Comment sécuriser l'accès à Azure Blob Storage ?*
- *Quel rôle jouent les Kubernetes Secrets dans la gestion des données sensibles ?*
- *Comment mettre en place une authentification basée sur l'identité pour les services ?*