Publier un ePortfolio

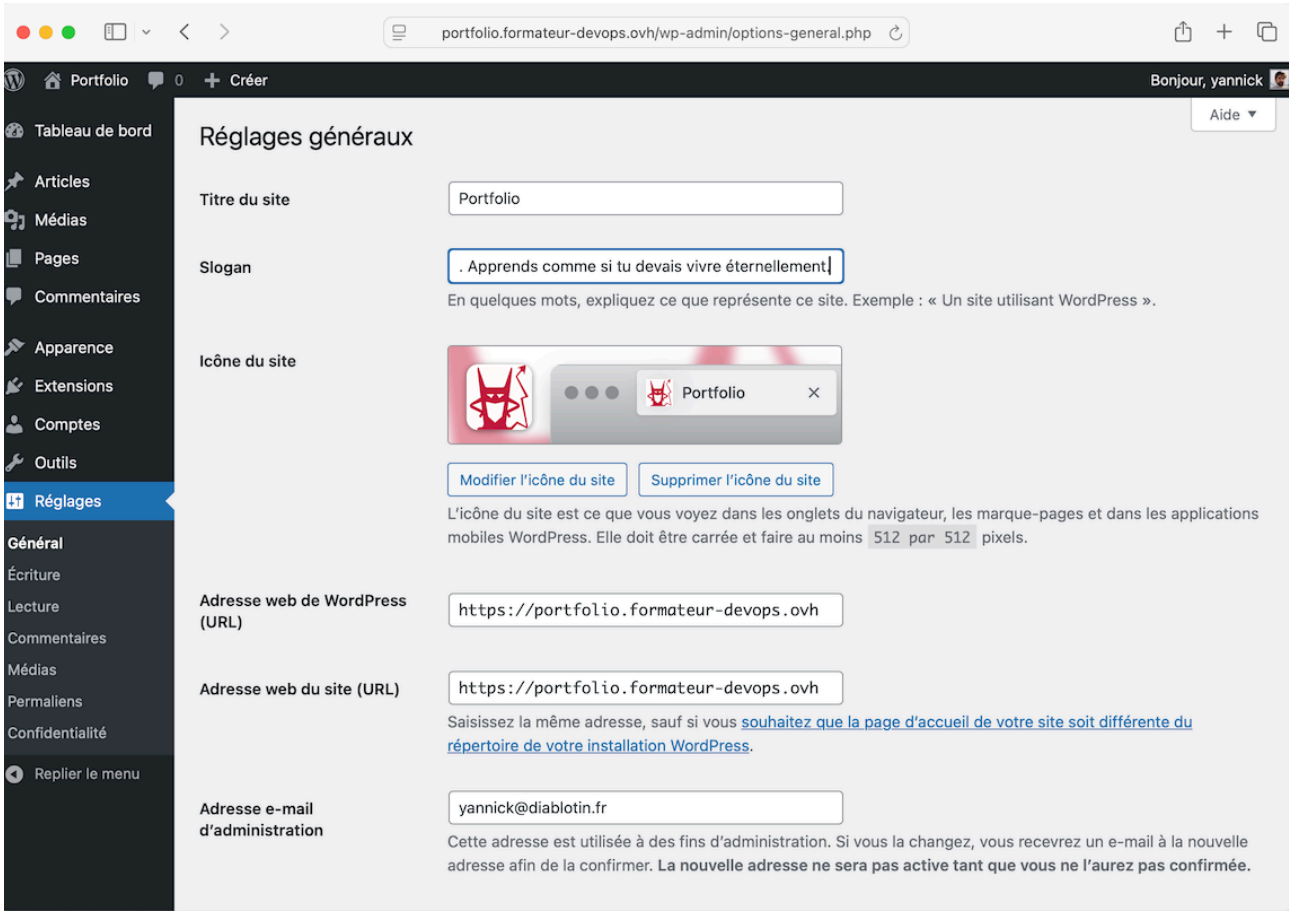
Publier un ePortfolio basé sur WordPress dans un cluster K8s

Avant de démarrer ce TP, il convient d'avoir suivi le cours « AKS : Azure Kubernetes Services Orchestration de conteneurs dans le Cloud public ».

Durée estimée
Entre 1 et 3 heures

Énoncé

Le but est d'avoir, visuellement, un résultat qui ressemble à la capture ci-dessous.



Définir des labels "proprio: ycadin" (adapter avec vos prénom et nom)
et "genre: portfolio" pour toutes les ressources qui les supportent.
MAIS ATTENTION, ces labels ne sont pas suffisants, ils doivent souvent être complétés car utilisés pour identifier les liaisons entre certaines ressources.

Dans un espace de nommage appelé "blog" :

- créer un PVC pour un PV dynamique (basé sur les classes de stockage azurefile ou azurefile-csi) de 250 MiO en ReadWriteOnce (NOTE, la création du PV et sa liaison avec le PVC peut prendre une bonne minute).

- Créer un déploiement mariadb:lts avec 1 seul replica :

  avec le PV créé ci-avant,

  variables d'environnement :
  MARIADB_ROOT_PASSWORD=portfolio
  MARIADB_DATABASE=portfolio
  MARIADB_USER=portfolio
  MARIADB_PASSWORD=portfolio

- Créer un service sgbdr "headless" (ou simplement ClusterIP si vous ne trouvez pas comment créer un service headless).

- Créer un déploiement wordpress avec 1 seul replica :

  variables d'environnement :
  WORDPRESS_DB_HOST=<a déduire de ce qui précède>
  WORDPRESS_DB_NAME=portfolio
  WORDPRESS_DB_USER=portfolio
  WORDPRESS_DB_PASSWORD=portfolio

- Créer une ressource Ingress pour accéder à wordpress (portfolio.votre.domaine) avec certificat SSL/TLS (l'obtention du certificat peut prendre quelques dizaines de secondes).
  NOTE : Nginx limite, par défaut, à 1 Mo la taille des téléversements !
  (Voir annotation dans manifeste fourni pour lever cette limitation.)

Recommandations et remarques

- Consulter (sur le hub Docker) les fiches des images mises en œuvre dans cet exercice pour y trouver notamment les chemins d'accès sur lesquels monter les volumes persistants.

- Utiliser l'action explain de kubectl pour retrouver les bonnes imbrications et syntaxes pour la rédaction des manifestes.

- Ne pas hésiter à utiliser la combinaison d'options -o yaml --dry-run=client pour générer tout ou partie des manifestes si cela peut simplifier ou accélérer les choses.

- Employer l'action port-forward pour tester le bon fonctionnement du duo WordPress / SGBDR avec de définir la règle ingress.

ATTENTION : l'utilisation d'une connexion avec port-forward sur un port différent de 80 peut avoir un effet indésirable, visible dans les réglages généraux de WordPress, il y a deux champs "URL" qui sont automatiquement renseignés lors de la finalisation de l'installation.

- Suggestion pour un "acteur" agissant comme contrôleur :

kubectl run -it --rm testeur -n blog --image busybox -- ash

- En cas de problème, utiliser les actions describe ou logs de kubectl pour enquêter.

- En cas de doute sur la création de la base de données, vérifier minutieusement l'orthographe des variables d'environnement et accessoirement les valeurs qui y sont affectées.

Pour vérifier : kubectl exec -it sgbdr -- mariadb -pportfolio -e "show databases;"

ATTENTION : la persistance des données (dans les volumes éponymes) peut se révéler handicapante tant que des erreurs sont présentes dans les noms des variables d'environnement et, dans une moindre mesure, leurs valeurs. En effet, la correction d'une erreur dans un manifeste sera probablement ignorée à cause de la variante erronée présente dans un volume persistant et qui sera prioritaire.
Exemple concret : si la variable MARIADB_DATABASE est mal orthographiée alors aucune base de données ne sera créée lors du démarrage du pod sgbdr. Si le pod est supprimé puis recréé, il se contentera de constater l'absence de toute base dans le volume persistant qui aura été remonté.

Challenges (épreuves facultatives)

NOTE : il est recommandé de supprimer les ressources existantes avant de les récréer ou de travailler avec un nouveau jeu de ressources sur un espace de nommage alternatif.

• Remplacer le déploiement sgbdr par un simple pod (puisque, pour l'heure, l'on ne saurait avoir plus d'une instance du moteur de bases de données).

• Le fonctionnement modélisé ci-avant est on ne peut plus éphémère car beaucoup d'opérations (comme l'ajout d'un média par exemple) entraînent des modifications dans le système de fichiers du pod wordpress et ne survivent évidemment pas à sa disparition.
Un pis-aller consiste à utiliser un PV (basé sur les classes de stockage azurefile-csi ou azurefile) d'environ 300 MiO en ReadWriteMany (à monter sur /var/www/html).

REMARQUE : il s'avère que WordPress n'est pas le plus simple (et encore moins le plus adapté) des programmes à déployer au sein d'un orchestrateur.

• Ajouter "manuellement" un réplica au déploiement wordpress (avec edit, scale ou apply).

• Ajouter des spécifications de requests et surtout limits RAM et CPU (en extrapolant à partir des mesures relevées avec kubectl top po -n blog).

ATTENTION : des valeurs trop extrêmes risquent d'occasionner des dysfonctionnements.

- Définir un HPA pour WordPress (seuil à 60%, 1 min., 5 max.).
- Définir une restriction réseau (seuls les pods wordpress doivent pouvoir au pod sgbdr).
- Relever la taille maximale des médias téléversés de 2 Mo à 8 Mo (piste : https://stackoverflow.com/a/57394392).
- Enfin définir un statefulset Galera. (Ce sera pour un prochain cours.)

Solution

Tous les détails se trouvent dans les manifestes fournis.