Exercices Kubernetes

Exercice 1 : déploiement de l'application PHP Guestbook avec Redis.

Cet exercice explique comment créer et déployer une application Web simple multi-tier à l'aide de Kubernetes et de Docker. Cet exemple comprend les composants suivants :

- Un master Redis à instance unique pour stocker les entrées du livre d'or.
- Plusieurs instances Redis répliquées.
- Plusieurs instances de front web.

Objectifs

- Démarrer un master Redis.
- Démarrez des esclaves Redis.
- Démarrez l'interface du livre d'or.
- Exposer et afficher le service frontal.
- Nettoyer.

Avant de commencer

Vous devez disposer d'un cluster Kubernetes et l'outil de ligne de commande **kubectl** doit être configuré pour communiquer avec votre cluster. Si vous ne possédez pas déjà de cluster, vous pouvez en créer un en utilisant Minikube.

Pour vérifier la version, entrez **kubectl version**.

Démarrer le master Redis

L'application guestbook utilise Redis pour stocker ses données. Elle écrit ses données sur une instance maître Redis et lit les données de plusieurs instances esclaves Redis.

Création du déploiement Redis Master

Le fichier manifeste, inclus ci-dessous, spécifie un contrôleur de déploiement qui exécute un seul réplica master Pod Redis.

redis-master-deployment.yaml

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: redis-master
  labels:
    app: redis
spec:
  selector:
    matchLabels:
      app: redis
      role: master
      tier: backend
  replicas: 1
```

redis-master-deployment.yaml

```
template:
  metadata:
    labels:
      app: redis
      role: master
      tier: backend
  spec:
    containers:
    - name: master
      image: k8s.gcr.io/redis:e2e  # or just image: redis
      resources:
        requests:
          cpu: 100m
          memory: 100Mi
      ports:
      - containerPort: 6379
```

1. Lancez une fenêtre de terminal dans le répertoire dans lequel vous avez téléchargé les fichiers de manifeste.

2. Appliquez le **Redis Master Deployment** à partir du fichier redis-master-deployment.yaml :

sudo kubectl apply -f redis-master-deployment.yaml

3. Interrogez la liste des pods pour vérifier que le pod Redis Master est en cours d'exécution:

sudo kubectl get pods

La réponse devrait ressembler à ceci:

```
NAME                     READY   STATUS    RESTARTS   AGE
redis-master-1068406935-3lswp   1/1     Running   0          28s
```

4. Exécutez la commande suivante pour afficher les journaux du pod Redis Master :

sudo kubectl logs -f POD-NAME

Remarque : remplacez POD-NAME par le nom de votre pod.

Création du service mater Redis

Les applications du livre d'or doivent communiquer avec le master Redis pour écrire ses données. Vous devez appliquer un service pour transférer le trafic par proxy vers le pod Redis master. Un service définit une politique pour accéder aux pods.

redis-master-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  labels:
    app: redis
    role: master
    tier: backend
spec:
  ports:
  - port: 6379
    targetPort: 6379
  selector:
    app: redis
    role: master
    tier: backend
```

5. Appliquez le service mater Redis à partir du fichier **redis-master-service.yaml** suivant :

sudo kubectl apply -f redis-master-service.yaml

6. Interrogez la liste des services pour vérifier que le service master de Redis est en cours d'exécution :

sudo kubectl get service

La réponse devrait ressembler à ceci :

```
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
kubernetes    ClusterIP   10.0.0.1      <none>        443/TCP    1m
redis-master  ClusterIP   10.0.0.151    <none>        6379/TCP   8s
```

Remarque : Ce fichier manifeste crée un service appelé redis-master avec un ensemble d'étiquettes correspondant aux étiquettes définies précédemment. Ainsi, le service route le trafic réseau vers le pod Redis master.

Démarrez les esclaves Redis

Bien que le maître Redis soit un seul pod, vous pouvez le rendre hautement disponible pour répondre aux besoins du trafic en ajoutant des répliques (replicas) d'esclaves Redis.

Création du déploiement de Redis Slave

Les déploiements évoluent en fonction des configurations définies dans le fichier manifeste. Dans ce cas, l'objet Deployment spécifie deux réplicas.

S'il n'y a pas de réplicas en cours d'exécution, ce déploiement lancera les deux réplicas sur votre cluster de conteneurs. À l'inverse, s'il y a plus de deux réplicas en cours d'exécution, la réduction (scale down) est effectuée jusqu'à ce que deux réplicas soient en cours d'exécution.

redis-slave-deployment.yaml

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: redis-slave
  labels:
    app: redis
spec:
  selector:
    matchLabels:
      app: redis
      role: slave
      tier: backend
  replicas: 2
  template:
    metadata:
      labels:
        app: redis
        role: slave
        tier: backend
    spec:
      containers:
      - name: slave
        image: gcr.io/google_samples/gb-redisslave:v1
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # Using `GET_HOSTS_FROM=dns` requires your cluster to
          # provide a dns service. As of Kubernetes 1.3, DNS is a built-in
          # service launched automatically. However, if the cluster you are using
          # does not have a built-in DNS service, you can instead
          # access an environment variable to find the master
          # service's host. To do so, comment out the 'value: dns' line above, and
          # uncomment the line below:
          # value: env
        ports:
        - containerPort: 6379
```

7. Appliquez le déploiement Redis Slave à partir du fichier **redis-slave-deployment.yaml** :

sudo kubectl apply -f redis-slave-deployment.yaml

8. Interrogez la liste des pods pour vérifier que les pods esclaves Redis sont en cours d'exécution :

sudo kubectl get pods

La réponse devrait ressembler à ceci :

```
NAME                          READY     STATUS              RESTARTS   AGE
redis-master-1068406935-3lswp 1/1       Running             0          1m
redis-slave-2005841000-fpvqc  0/1       ContainerCreating   0          6s
redis-slave-2005841000-phfv9  0/1       ContainerCreating   0          6s
```

Création du service Redis Slave
L'application guestbook doit communiquer avec les esclaves Redis pour lire les données. Pour rendre les esclaves Redis découvrables, vous devez configurer un service. Un service fournit un équilibrage de charge transparent à un ensemble de pods.

redis-slave-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: redis-slave
  labels:
    app: redis
    role: slave
    tier: backend
spec:
  ports:
  - port: 6379
  selector:
    app: redis
    role: slave
    tier: backend
```

9. Appliquez le service Redis Slave à partir du fichier **redis-slave-service.yaml** suivant :

sudo kubectl apply -f redis-slave-service.yaml

10. Interrogez la liste des services pour vérifier que le service esclave Redis est en cours d'exécution :

sudo kubectl get services

La réponse devrait ressembler à ceci :

```
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes    ClusterIP   10.0.0.1     <none>        443/TCP    2m
redis-master  ClusterIP   10.0.0.151   <none>        6379/TCP   1m
redis-slave   ClusterIP   10.0.0.223   <none>        6379/TCP   6s
```

Configurer et exposer le livre d'or.
L'application guestbook a une interface Web servant les requêtes HTTP écrites en PHP. Il est configuré pour se connecter au service redis-master pour les demandes d'écriture et au service redis-esclave pour les demandes de lecture.

Création du déploiement de livre d'or en mode frontal (Frontend).

frontend-deployment.yaml

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: frontend
  labels:
    app: guestbook
spec:
  selector:
    matchLabels:
      app: guestbook
      tier: frontend
  replicas: 3
  template:
    metadata:
      labels:
        app: guestbook
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google-samples/gb-frontend:v4
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # Using `GET_HOSTS_FROM=dns` requires your cluster to
          # provide a dns service. As of Kubernetes 1.3, DNS is a built-in
          # service launched automatically. However, if the cluster you are using
          # does not have a built-in DNS service, you can instead
          # access an environment variable to find the master
          # service's host. To do so, comment out the 'value: dns' line above, and
          # uncomment the line below:
          # value: env
        ports:
        - containerPort: 80
```

11. Appliquez le déploiement frontend à partir du fichier **frontend-deployment.yaml** :

sudo kubectl apply -f frontend-deployment.yaml

12. Interrogez la liste des pods pour vérifier que les trois réplicas frontaux sont en cours d'exécution :

sudo kubectl get pods -l app=guestbook -l tier=frontend

La réponse devrait ressembler à ceci:

```
NAME                        READY   STATUS    RESTARTS   AGE
frontend-3823415956-dsvc5   1/1     Running   0          54s
frontend-3823415956-k22zn   1/1     Running   0          54s
frontend-3823415956-w9gbt   1/1     Running   0          54s
```

Création du service frontend

Les services redis-slave et redis-master que vous avez appliqués ne sont accessibles que dans le cluster de conteneurs car le type par défaut d'un service est **ClusterIP**. ClusterIP fournit une adresse IP unique pour l'ensemble des pods vers lesquels le service pointe. Cette adresse IP est accessible uniquement dans le cluster.

Si vous souhaitez que les invités puissent accéder à votre livre d'or, vous devez configurer le service frontend pour qu'il soit visible de l'extérieur, afin qu'un client puisse demander le service de l'extérieur du cluster de conteneurs. Minikube ne peut exposer les services que via NodePort.

Remarque : certains fournisseurs de cloud, tels que Google Compute Engine ou Google Kubernetes Engine, prennent en charge des équilibreurs de charge externes. Si votre fournisseur de cloud prend en charge les équilibreurs de charge et que vous souhaitez l'utiliser, supprimez ou commentez simplement le type : NodePort et décommentez le type : LoadBalancer.

frontend-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # comment or delete the following line if you want to use a LoadBalancer
  type: NodePort
  # if your cluster supports it, uncomment the following to automatically create
  # an external load-balanced IP for the frontend service.
  # type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: guestbook
    tier: frontend
```

13. Appliquez le service frontend à partir du fichier **frontend-service.yaml** :

sudo kubectl apply -f frontend-service.yaml

14. Interrogez la liste des services pour vérifier que le service frontal est en cours d'exécution :

sudo kubectl get services

La réponse devrait ressembler à ceci :

```
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)       AGE
frontend      ClusterIP   10.0.0.112   <none>        80:31323/TCP  6s
kubernetes    ClusterIP   10.0.0.1     <none>        443/TCP       4m
redis-master  ClusterIP   10.0.0.151   <none>        6379/TCP      2m
redis-slave   ClusterIP   10.0.0.223   <none>        6379/TCP      1m
```

Affichage du service frontend via NodePort

Si vous avez déployé cette application sur un mini-cube ou un cluster local, vous devez trouver l'adresse IP pour afficher votre livre d'or.

15. Exécutez la commande suivante pour obtenir l'adresse IP du service frontend.

sudo minikube service frontend --url

La réponse devrait ressembler à ceci:

http://192.168.99.100:31323

16. Copiez l'adresse IP et chargez la page dans votre navigateur pour afficher votre livre d'or.

Affichage du service frontend via LoadBalancer

Si vous avez déployé le manifeste **frontend-service.yaml** avec le type: **LoadBalancer**, vous devez trouver l'adresse IP pour afficher votre livre d'or.

17. Exécutez la commande suivante pour obtenir l'adresse IP du service frontal.

sudo kubectl get service frontend

La réponse devrait ressembler à ceci:

```
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP       PORT(S)       AGE
frontend  ClusterIP   10.51.242.136   109.197.92.229    80:32372/TCP  1m
```

18. Copiez l'adresse IP externe et chargez la page dans votre navigateur pour afficher votre livre d'or.

Mise à l'échelle de l'interface Web (Scale the Web Frontend)

La mise à l'échelle est réduite car vos serveurs sont définis comme un service utilisant un contrôleur de déploiement.

19. Exécutez la commande suivante pour augmenter (scale up) le nombre de pods frontaux :

sudo kubectl scale deployment frontend --replicas=5

20. Interrogez la liste des pods pour vérifier le nombre de pods frontend en cours d'exécution :

sudo kubectl get pods

La réponse devrait ressembler à ceci :

```
NAME                    READY   STATUS    RESTARTS   AGE
```

```
frontend-3823415956-70qj5       1/1     Running   0          5s
frontend-3823415956-dsvc5       1/1     Running   0          54m
frontend-3823415956-k22zn       1/1     Running   0          54m
frontend-3823415956-w9gbt       1/1     Running   0          54m
frontend-3823415956-x2pld       1/1     Running   0          5s
redis-master-1068406935-3lswp   1/1     Running   0          56m
redis-slave-2005841000-fpvqc    1/1     Running   0          55m
redis-slave-2005841000-phfv9    1/1     Running   0          55m
```

21. Exécutez la commande suivante pour réduire le nombre de pods frontaux :

sudo kubectl scale deployment frontend --replicas=2

22. Interrogez la liste des pods pour vérifier le nombre de pods frontend en cours d'exécution :

sudo kubectl get pods

La réponse devrait ressembler à ceci :

```
NAME                            READY     STATUS    RESTARTS   AGE
frontend-3823415956-k22zn       1/1       Running   0          1h
frontend-3823415956-w9gbt       1/1       Running   0          1h
redis-master-1068406935-3lswp   1/1       Running   0          1h
redis-slave-2005841000-fpvqc    1/1       Running   0          1h
redis-slave-2005841000-phfv9    1/1       Running   0          1h
```

Nettoyer

La suppression des déploiements et des services supprime également les pods en cours d'exécution. Utilisez des étiquettes pour supprimer plusieurs ressources avec une seule commande.

23. Exécutez les commandes suivantes pour supprimer tous les pods, déploiements et services.

sudo kubectl delete deployment -l app=redis

sudo kubectl delete service -l app=redis

sudo kubectl delete deployment -l app=guestbook

sudo kubectl delete service -l app=guestbook

Les réponses devraient ressembler à ceci :

```
deployment.apps "redis-master" deleted
deployment.apps "redis-slave" deleted
service "redis-master" deleted
service "redis-slave" deleted
deployment.apps "frontend" deleted
service "frontend" deleted
```

24. Interrogez la liste des pods pour vérifier qu'aucun n'est en cours d'exécution :

sudo kubectl get pods

La réponse devrait ressembler à ceci :

No resources found.

Exercice 2 : déploiement de WordPress et MySQL avec des volumes persistants

Cet exercice vous montre comment déployer un site WordPress et une base de données MySQL avec Minikube. Les deux applications utilisent **PersistentVolumes** et **PersistentVolumeClaims** pour stocker des données.

Un fichier **PersistentVolume (PV)** est un élément de stockage dans le cluster qui a été provisionné manuellement par un administrateur ou dynamiquement par Kubernetes à l'aide d'une StorageClass. Un **PVC (PersistentVolumeClaim)** est une demande de stockage d'un utilisateur pouvant être satisfaite par un PV. **PersistentVolumes** et **PersistentVolumeClaims** sont indépendants des cycles de vie des pods et préservent les données par le redémarrage, la replanification et même la suppression de pods.

Avertissement : Ce déploiement ne convient pas aux cas d'utilisation en production, car il utilise des instances WordPress et MySQL Pod à instance unique. Pensez à utiliser WordPress Helm Chart pour déployer WordPress en production.

Remarque : Les fichiers fournis dans ce didacticiel utilisent les API de déploiement GA et sont spécifiques à kubernetes version 1.9 et ultérieure. Si vous souhaitez utiliser cet exemple avec une version antérieure de Kubernetes, veuillez mettre à jour la version de l'API de manière appropriée ou référencer des versions antérieures de cet exercice.

Objectifs

- Créer des PersistentVolumeClaims et des PersistentVolumes
- Créer un secret
- Déployer MySQL
- Déployer WordPress
- Nettoyer

Avant de commencer

Vous devez disposer d'un cluster Kubernetes et l'outil de ligne de commande **kubectl** doit être configuré pour communiquer avec votre cluster. Si vous ne possédez pas déjà de cluster, vous pouvez en créer un en utilisant Minikube.

Pour vérifier la version, entrez **kubectl version**.

Récupérer les fichiers de configuration suivants : **mysql-deployment.yaml** et **wordpress-deployment.yaml**

Créer PersistentVolumeClaims et PersistentVolumes

MySQL et Wordpress nécessitent chacun un **PersistentVolume** pour stocker les données. Leurs **PersistentVolumeClaims** seront créées à l'étape du déploiement.

StorageClass par défaut est installé dans de nombreux environnements de cluster. Lorsqu'une classe StorageClass n'est pas spécifiée dans **PersistentVolumeClaim**, la classe **StorageClass** par défaut du cluster est utilisée.

Lorsqu'un PersistentVolumeClaim est créé, un PersistentVolume est provisionné de manière dynamique en fonction de la configuration de StorageClass.

Avertissement : Dans les clusters locaux, la StorageClass par défaut utilise le provisioner **hostPath**. Les volumes hostPath ne conviennent que pour le développement et les tests. Avec les volumes hostPath, vos données résident dans **/tmp** sur le nœud sur lequel le pod est planifié et ne sont pas déplacées entre les nœuds. Si un pod meurt et est planifié sur un autre nœud du cluster, ou si le nœud est redémarré, les données sont perdues.

Remarque : Si vous créez un cluster qui doit utiliser le provisioner hostPath, l'indicateur **--enable-hostpath-provisioner** doit être défini dans le composant controller-manager.

Remarque : Si vous avez un cluster Kubernetes s'exécutant sur Google Kubernetes Engine, veuillez voir la documentation.

Créer un secret pour le mot de passe MySQL

Un secret est un objet qui stocke une donnée sensible comme un mot de passe ou une clé. Les fichiers de manifeste sont déjà configurés pour utiliser un secret, mais vous devez créer votre propre secret.

1. Créez l'objet Secret à l'aide de la commande suivante. Vous devrez remplacer YOUR_PASSWORD par le mot de passe que vous souhaitez utiliser.

sudo kubectl create secret generic mysql-pass --from-literal=password=YOUR_PASSWORD

2. Vérifiez que le secret existe en exécutant la commande suivante:

sudo kubectl get secrets

La réponse devrait ressembler à ceci :

```
NAME          TYPE          DATA   AGE
mysql-pass    Opaque        1      42s
```

Remarque : pour protéger le secret de l'exposition, ni obtenir, ni décrire, ni montrer son contenu.

Déployer MySQL

Le manifeste suivant décrit un déploiement MySQL à instance unique. Le conteneur MySQL monte PersistentVolume dans **/var/lib/mysql**. La variable d'environnement MYSQL_ROOT_PASSWORD définit le mot de passe de la base de données à partir du secret.

mysql-deployment.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
```

mysql-deployment.yaml

```
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
  clusterIP: None
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: wordpress-mysql
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysql-pass
              key: password
        ports:
```

mysql-deployment.yaml

```
  - containerPort: 3306
    name: mysql
  volumeMounts:
  - name: mysql-persistent-storage
    mountPath: /var/lib/mysql
volumes:
- name: mysql-persistent-storage
  persistentVolumeClaim:
    claimName: mysql-pv-claim
```

3. Déployez MySQL à partir du fichier **mysql-deployment.yaml** :

sudo kubectl create -f mysql-deployment.yaml

4. Vérifiez qu'un PersistentVolume a été provisionné de manière dynamique. Notez que cela peut prendre jusqu'à quelques minutes pour que les PV soient provisionnés et liés.

sudo kubectl get pvc

La réponse devrait ressembler à ceci:

```
 NAME             STATUS   VOLUME                                     CAPACITY ACCESS MODES
STORAGECLASS   AGE
 mysql-pv-claim   Bound    pvc-91e44fbf-d477-11e7-ac6a-42010a800002   20Gi     RWO
standard       29s
```

5. Vérifiez que le pod est en cours d'exécution en exécutant la commande suivante :

sudo kubectl get pods

Remarque : Cela peut prendre quelques minutes pour que le statut du pod soit RUNNING.

La réponse devrait ressembler à ceci :

```
 NAME                         READY   STATUS    RESTARTS   AGE
 wordpress-mysql-1894417608-x5dzt   1/1     Running   0          40s
```

Déployer WordPress

Le manifeste suivant décrit un service et déploiement de WordPress à instance unique. Il utilise beaucoup des mêmes fonctionnalités, comme un PVC pour le stockage persistant et un secret pour le mot de passe. Mais il utilise également un paramètre différent : type : LoadBalancer. Ce paramètre expose WordPress au trafic provenant de l'extérieur du cluster.

wordpress-deployment.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  labels:
```

wordpress-deployment.yaml

```
    app: wordpress
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 20Gi
---
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
      - image: wordpress:4.8-apache
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
```

wordpress-deployment.yaml

```
      secretKeyRef:
        name: mysql-pass
        key: password
    ports:
    - containerPort: 80
      name: wordpress
    volumeMounts:
    - name: wordpress-persistent-storage
      mountPath: /var/www/html
  volumes:
  - name: wordpress-persistent-storage
    persistentVolumeClaim:
      claimName: wp-pv-claim
```

6. Créez un service et un déploiement WordPress à partir du fichier **wordpress-deployment.yaml** :

sudo kubectl create -f wordpress-deployment.yaml

7. Vérifiez qu'un volume persistant a été provisionné de manière dynamique :

sudo kubectl get pvc

Remarque : Il peut s'écouler quelques minutes avant que les PV soient provisionnés et liés.

La réponse devrait ressembler à ceci :

```
 NAME            STATUS    VOLUME                                     CAPACITY ACCESS MODES
STORAGECLASS   AGE
 wp-pv-claim     Bound     pvc-e69d834d-d477-11e7-ac6a-42010a800002   20Gi     RWO
standard       7s
```

8. Vérifiez que le service est en cours d'exécution en exécutant la commande suivante :
sudo kubectl get services wordpress

La réponse devrait ressembler à ceci :

```
 NAME        TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
 wordpress   ClusterIP   10.0.0.89    <pending>     80:32406/TCP   4m
```

Remarque : Minikube ne peut exposer les services que via NodePort. L'adresse IP externe est toujours en attente.

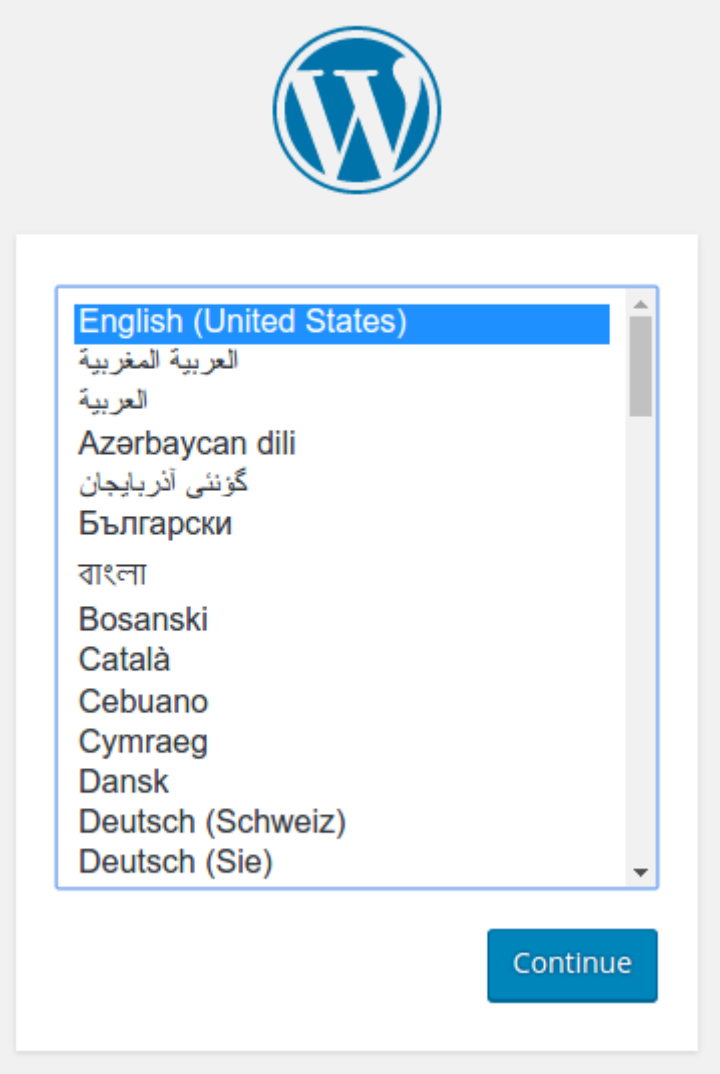
9. Exécutez la commande suivante pour obtenir l'adresse IP du service WordPress :

sudo minikube service wordpress --url

La réponse devrait ressembler à ceci :

http://1.2.3.4:32406

10. Copiez l'adresse IP et chargez la page dans votre navigateur pour afficher votre site. La page de configuration de WordPress devrait ressembler à la capture d'écran suivante.



Avertissement : Ne laissez pas votre installation WordPress sur cette page. Si un autre utilisateur la trouve, il peut configurer un site Web sur votre instance et l'utiliser pour diffuser du contenu malveillant.

Installez WordPress en créant un nom d'utilisateur et un mot de passe ou supprimez votre instance.

Nettoyage

11. Exécutez la commande suivante pour supprimer votre secret :

```
sudo  kubectl delete secret mysql-pass
```

12. Exécutez les commandes suivantes pour supprimer tous les déploiements et services :

sudo kubectl delete deployment -l app=wordpress

sudo kubectl delete service -l app=wordpress

13. Exécutez les commandes suivantes pour supprimer **PersistentVolumeClaims**. Les volumes persistants provisionnés de manière dynamique seront automatiquement supprimés.

sudo kubectl delete pvc -l app=wordpress