Despliegues parametrizados

Despliegues parametrizados

En muchas ocasiones necesitamos parametrizar nuestros despliegues, para ello podemos guardar información de distinta forma. En este módulo vamos a estudiar distintas maneras de guardar información en kubernetes, que nos permitirá posteriormente parametrizar los Deployments:

- Variables de entorno
- ConfigMaps
- Secrets

Terminaremos estudiaremos un ejemplo completo donde desplegaremos un WoordPress + MariaDB.

Variables de entorno

Para añadir alguna configuración específica a la hora de lanzar un contenedor, se usan variables de entorno del contenedor cuyos valores se especifican al crear el contenedor para realizar una configuración concreta del mismo.

Por ejemplo, si estudiamos la documentación de la imagen `mariadb` en Docker Hub, podemos comprobar que podemos definir un conjunto de variables de entorno como `MYSQL_ROOT_PASSWORD`, `MYSQL_DATABASE`, `MYSQL_USER`, `MYSQL_PASSWORD`, etc., que nos permitirán configurar de alguna forma determinada nuestro servidor de base de datos (indicando la contraseña del usuario root, creando una determinada base de datos o creando un usuario con una contraseña por ejemplo.

De la misma manera, al especificar los contenedores que contendrán los Pods que se van a crear desde un Deployment, también se pondrán inicializar las variables de entorno necesarias.

Configuración de aplicaciones usando variables de entorno

Vamos a hacer un despliegue de un servidor de base de datos mariadb. Si volvemos a estudiar la documentación de esta imagen en Docker Hub comprobamos que obligatoriamente tenemos que indicar la contraseña del usuario root inicializando la variable de entorno `MYSQL_ROOT_PASSWORD`. El fichero de despliegue que vamos a usar es `mariadb-deployment-env.yaml`, y vemos el fragmento del fichero donde se define el contenedor:

```
...
    spec:
      containers:
        - name: contenedor-mariadb
          image: mariadb
          ports:
            - containerPort: 3306
              name: db-port
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: my-password
```

En el apartado `containers` hemos incluido la sección `env` donde vamos indicando, como una lista, el nombre de la variable (`name`) y su valor (`value`). En este caso hemos indicado la

contraseña `my-password`.

Vamos a comprobar si realmente se ha creado el servidor de base de datos con esa contraseña del root:

```
kubectl apply -f mariadb-deploymen-env.yaml

kubectl get all
...
NAME                                  READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/mariadb-deployment    1/1     1            1           5s

kubectl exec -it deployment.apps/mariadb-deployment -- mysql -u root -p
Enter password:
...
MariaDB [(none)]>
```

Vídeo: Variables de entorno

https://www.youtube.com/embed/MazyA8LP8Oc

Vídeo: Variables de entorno

ConfigMaps

En el apartado anterior hemos estudiado como podemos definir las variables de entorno de los contenedores que vamos a desplegar. Sin embargo, la solución que presentamos puede tener alguna limitación:

- Los valores de las variables de entorno están escritos directamente en el fichero yaml. Estos ficheros yaml suelen estar en repositorios git y lógicamente no es el sitio más adecuado para ubicarlos.
- Por otro lado, escribiendo los valores de las variables de entorno directamente en los ficheros, hacemos que estos ficheros no sean reutilizables en otros despliegues y que el procedimiento de cambiar las variables sea tedioso y propenso a errores, porque hay que hacerlo en varios sitios.

Para solucionar estas limitaciones, podemos usar un nuevo recurso de Kubernetes llamado ConfigMap.

Configuración de aplicaciones usando ConfigMaps

ConfigMap permite definir un diccionario (clave,valor) para guardar información que se puede utilizar para configurar una aplicación.

Aunque hay distintas formas de indicar el conjunto de claves-valor de nuestro ConfigMap, en este caso vamos a usar literales, por ejemplo:

```
kubectl create cm mariadb --from-literal=root_password=my-password \
                          --from-literal=mysql_usuario=usuario     \
                          --from-literal=mysql_password=password-user \
                          --from-literal=basededatos=test
```

En el ejemplo anteriore, hemos creado un ConfigMap llamado `mariadb` con cuatro pares clave-valor. Para ver los ConfigMap que tenemos creados, podemos utilizar:

```
kubectl get cm
```

Y para ver los detalles del mismo:

```
kubectl describe cm mariadb
```

Una vez que creado el ConfigMap se puede crear un despliegue donde las variables de entorno se inicializan con los valores guardados en el ConfigMap. Por ejemplo, un despliegue de una base de datos lo podemos encontrar en el fichero `mariadb-deployment-configmap.yaml` y el fragmento donde definimos las variables de entorno quedaría:

```
...
    spec:
      containers:
        - name: contenedor-mariadb
          image: mariadb
          ports:
            - containerPort: 3306
              name: db-port
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                configMapKeyRef:
                  name: mariadb
                  key: root_password
            - name: MYSQL_USER
              valueFrom:
                configMapKeyRef:
                  name: mariadb
                  key: mysql_usuario
            - name: MYSQL_PASSWORD
              valueFrom:
                configMapKeyRef:
                  name: mariadb
                  key: mysql_password
            - name: MYSQL_DATABASE
              valueFrom:
                configMapKeyRef:
                  name: mariadb
                  key: basededatos
```

Observamos como al indicar las variables de entorno (sección `env`) seguimos indicado el nombre (`name`) pero el valor se indica con una clave de un ConfigMap (`valueFrom: - configMapKeyRef:`), para ello se indica el nombre del ConfigMap (`name`) y el valor que tiene una determinada clave (`key`). De esta manera, no guardamos en los ficheros yaml los valores específicos de las variables de entorno, y además, estos valores se pueden reutilizar para otros despliegues, por ejemplo, al desplegar un CMS indicar los mismos valores para las credenciales de acceso a la base de datos.

Vídeo: ConfigMaps

https://www.youtube.com/embed/QVC9TsHpXvc

Vídeo: ConfigMaps

Secrets

Cuando en un variable de entorno indicamos una información sensible, como por ejemplo, una contraseña o una clave ssh, es mejor utilizar un nuevo recurso de Kubernetes llamado Secret.

Los Secrets permiten guardar información sensible que será **codificada** o **cifrada**.

Hay distintos tipos de Secret, en este curso vamos a usar los genéricos y los vamos a crear a partir de un literal. Por ejemplo para guardar la contraseña del usuario root de una base de datos, crearíamos un Secret de la siguiente manera:

```
kubectl create secret generic mariadb --from-literal=password=my-password
```

Podemos obtener información de los Secret que hemos creado con las instrucciones:

```
kubectl get secret
kubectl describe secret mariadb
```

Veamos a continuación cómo quedaría un despliegue que usa el valor de un Secret para inicializar una variable de entorno. Vamos a usar el fichero `mariadb-deployment-secret.yaml` y el fragmento donde definimos las variables de entorno quedaría:

```
...
    spec:
      containers:
        - name: mariadb
          image: mariadb
          ports:
            - containerPort: 3306
              name: db-port
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mariadb
                  key: password
```

Observamos como al indicar las variables de entorno (sección `env`) seguimos indicado el nombre (`name`) pero el valor se indica con un valor de un Secret (`valueFrom: - secretKeyRef:`), indicando el nombre del Secret (`name`) y la clave correspondiente. (`key`).

Nota: Por defecto, k8s no cifra el valor de los Secrets, simplemente los codifica en base64. El cifrado de los Secrets requiere configuraciones adicionales que se detallan en Encrypting Secret Data at Rest.

Vídeo: Secrets

https://www.youtube.com/embed/lckojEFiUiw

Vídeo: Secrets

Ejemplo completo: Despliegue y acceso a Wordpress + MariaDB

En este ejemplo vamos a desplegar el CMS Wordpress y la base de datos MariaDB para su correcto funcionamiento. Cada uno de los servicios se van a desplegar independientemente, y configuraremos los Services para acceder desde el exterior a la aplicación y para que el Wordpress pueda acceder a la base de datos:

Configuración de los contenedores

- Como hemos estudiado en los ejemplos vistos en este módulo, al crear el despliegue de MariaDB tendremos que configurar las siguientes variables de entorno: `MYSQL_ROOT_PASSWORD` (contraseña del usuario root de la base de datos), `MYSQL_DATABASE` (el nombre de la base de datos que se va a crear), `MYSQL_USER` (nombre del usuario de la base de datos que se va carear), `MYSQL_PASSWORD` (contraseña de este usuario).
- Si comprobamos la documentación de la imagen Wordpress en Docker Hub las variables de entorno que vamos a definir son las siguientes: `WORDPRESS_DB_HOST` (la dirección del servidor de base de datos), `WORDPRESS_DB_USER` (el usuario que se va a usar para acceder a la base de datos), `WORDPRESS_DB_PASSWORD` (la contraseña de dicho usuario) y `WORDPRESS_DB_NAME` (el nombre de la base de datos a las que vamos a conectar para gestionar las tablas de Wordpress).

Evidentemente los valores de estas variables tienen que coincidir, es decir, el usuario y la contraseña que creemos en la base de datos serán las mismas que utilicemos desde Wordpress para acceder a la base de datos, y el nombre de la base de datos usada por WordPres será el mismo que la base de datos creada en MariaDB.

Lo veremos posteriormente, pero adelantamos, que el valor de la variable `WORDPRESS_DB_HOST` será el nombre del Service que creemos para acceder a la base de datos, como ya hemos estudiado, se creará un registro en el DNS del cluster que permitirá que Wordpress acceda a la base de datos usando el nombre del Service.

Los valores para crear la base de datos y el usuario en MariaDB, que corresponden a las credenciales que vamos a usar en Wordpress, los vamos a guardar en dos recursos de nuestro cluster: los datos no sensibles (nombre de usuario y nombre de la base de datos) lo

guardaremos en un ConfigMap y los datos sensibles, las contraseñas, la guardaremos en un Secret.

Para ello ejecutamos las siguientes instrucciones:

```
kubectl create cm bd-datos --from-literal=bd_user=user_wordpress \
                           --from-literal=bd_dbname=wordpress

kubectl create secret generic bd-passwords --from-literal=bd_password=password1234
                                           --from-literal=bd_rootpassword=root1234
```

Nota: No hemos guardado la definición del ConfigMap y el Secret en un fichero yaml. De esta manera evitamos que información sensible sea guardada por ejemplo en un repositorio git.

Sin embargo, a partir de las instrucciones anteriores podemos generar ficheros yaml que posteriormente añadimos a la configuración del cluster. Podemos ejecutar:

```
kubectl create cm bd-datos --from-literal=bd_user=user_wordpress \
                           --from-literal=bd_dbname=wordpress \
                           -o yaml --dry-run=client > bd_datos_configmap.yaml

kubectl create secret generic bd-passwords --from-literal=bd_password=password1234
                                           --from-literal=bd_rootpassword=root1234
                                           -o yaml --dry-run=client > bd_passwords_
```

Con la opción `--dry-run=client`, *kubectl* va a simular la creación del recurso, pero no se llega a ejecutar, sin embargo con la opción `-o yaml` generamos el fichero yaml con la definición del recurso. Posteriormente sólamente tendremos que ejecutar las siguientes instrucciones para crear los dos recursos:

```
kubectl apply -f bd_datos_configmap.yaml
kubectl apply -f bd_passwords_secret.yaml
```

Despliegue de la base de datos

Para desplegar la base de datos vamos a usar el fichero `mariadb-deployment.yaml`. Podemos observar en el fichero cómo los datos de las variables de entorno del contenedor se inicalizan con los valores que hemos creado en el ConfigMap y en el Secret anterior. Ejecutamos:

```
kubectl apply -f mariadb-deployment.yaml
```

La definición del Service que vamos a crear lo tenemos en el fichero: `mariadb-srv.yaml`. Como comprobamos en la definición estamos creando un Service del tipo ClusterIP, ya que no vamos a acceder a la base de datos desde el exterior. Además es importante recordar el nombre que hemos puesto al Service (`mariadb-service`), ya que cómo hemos indicado posteriormente,

usaremos este nombre para configurar la aplicación Wordpress a la hora de indicar el servidor de base de datos. Ejecutamos:

```
kubectl apply -f mariadb-srv.yaml
```

Despliegue de la aplicación Wordpress

Para desplegar la aplicación Wordpress vamos a usar el fichero `wordpress-deployment.yaml`. Podemos observar en el fichero cómo los datos de las variables de entorno del contenedor se inicializan con los valores que hemos creado en el ConfigMap y en el Secret anterior. Además, comprobamos que la variable de entorno `WORDPRESS_DB_HOST` se inicializa a `mariadb-service`, que es el nombre del Service creado para a acceder a mariaDB. Ejecutamos:

```
kubectl apply -f wordpress-deployment.yaml
```

La definición del Service que vamos a crear la tenemos en el fichero: wordpress-srv.yaml. Como comprobamos en la definición estamos creando un Service del tipo NodePort, pero también podríamos haberlo configurado de tipo ClusterIP, porque posteriormente vamos a crear un recurso Ingress para acceder a la aplicación. Ejecutamos:

```
kubectl apply -f wordpress-srv.yaml
```

Acceso a la aplicación

Para acceder a la aplicación vamos a crear un recurso Ingress que tenemos definido en el fichero: `wordpress-ingress.yaml`. Como podemos observar vamos a usar el nombre `www.miwordpress.org` que tendremos que añadir en la resolución estática del ordenador desde el que vamos a acceder. También es interesante observar a que Service se va acceder con este recurso Ingress, el nombre del Service es `wordpress-service` que evidentemente es el mismo que hemos puesto en la definición del Service de Wordpress.

Una vez que comprobemos que todos los recursos están funcionando, podemos acceder a nuestra aplicación:

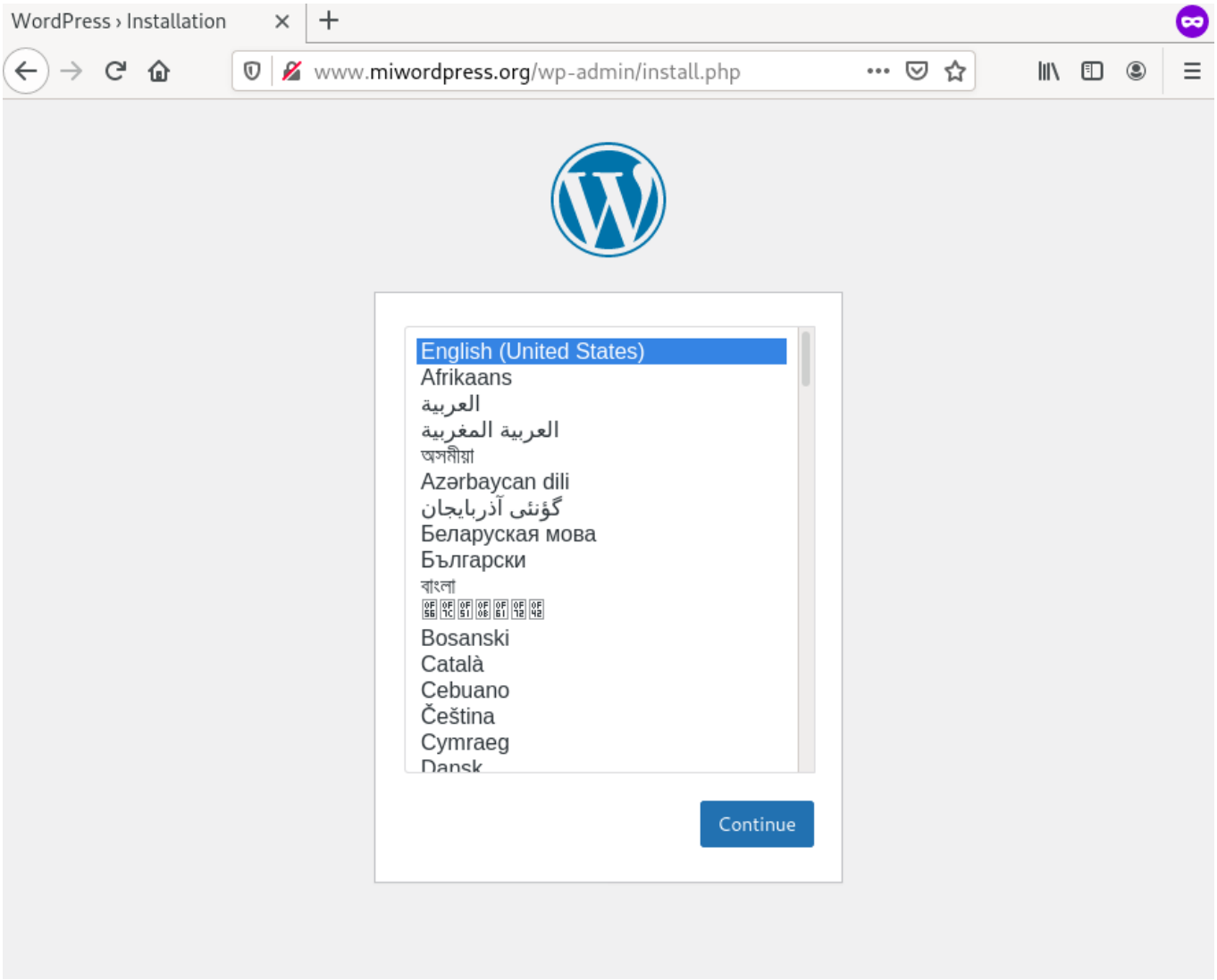


Imagen de elaboración propia (CC BY-NC-SA)

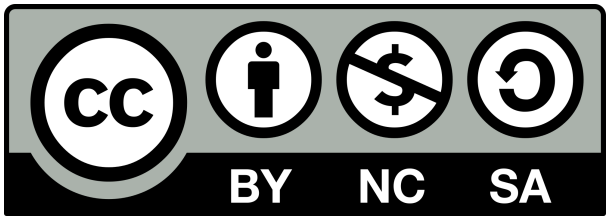
Vídeo: Ejemplo completo: Despliegue y acceso a Wordpress + MariaDB

https://www.youtube.com/embed/Iy6MaixXSrw

Vídeo: Ejemplo completo: Despliegue y acceso a Wordpress + MariaDB

Créditos

Materiales desarrollados por:
Alberto Molina Coballes
José Domingo Muñoz Rodríguez
Propiedad de la Consejería de Educación y Deporte de la Junta de Andalucía



2021