



Deploy en Kubernetes - PADSA

Ejemplo Deploy de una app en Kubernetes en un cluster de docker kubernetes.

En este ejemplo se desplegara un servicio con un una imagen de base de datos de mongo y mongo-express para levantar el servicio en un puerto e interactuar con ella desde el navegador.

Ambiente de desarrollo y herramientas utilizadas:

Windows 10

WSL2

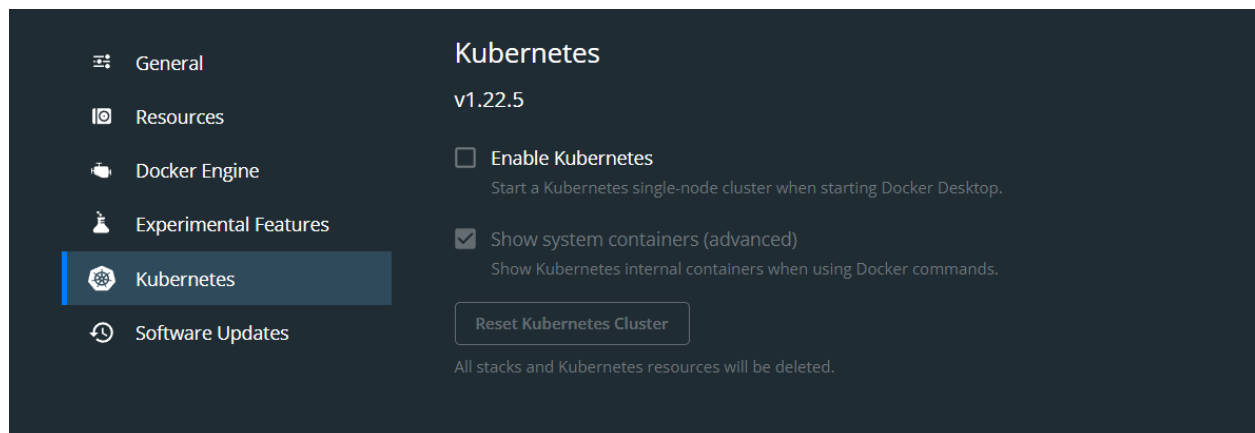
Docker 4.5

Terminal de comandos

kubectl

Desarrollo:

Habilitar kubernetes en Docker en la sección de Configuración/Kubernetes



Comprobar la versión de server y client.

```
kubectl version
```

Para ver la información del clúster usar el comando: (Docker Kubernetes crea una única instancia de un nodo).

```
kubectl get nodes
```

 o para una descripción detallada

```
kubectl describe node
```

Creación de los archivos de configuración del despliegue

mongo-secret.yaml

Con archivo tipo yaml *"mongo-secret.yaml"* creamos el servicio *"mongodb-secrets"* para contener las credenciales de usuario y password de mongo de manera encriptada mediante el algoritmo base64. Este deploy es de clase *Secret* y de tipo *Opaque*.

```
# mongo-secret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
type: Opaque
data:
  mongo-root-username: dXNlcm5hbWU=
  mongo-root-password: cGFzc3dvcmQ=
```

mongo.yaml

En un archivo yaml configuramos el deployment de *"mongodb-deployment"* que crea un pod de la imagen de mongo, se expone en el puerto 27017 dentro del pod, su identificador de app será *mongodb*, agregamos las variables de usuario y password que irá a buscar al del deploy tipo secret antes desplegado *"mongodb-secret"*

Dentro del mismo archivo se incluye la configuración de la creación del servicio de nombre *"mongodb-secret"* de la app desplegada *"mongodb"*, se configura el puerto dentro del pod que es el mismo que ya configuramos antes y el puerto target de exposición 27017:27017 :

```
# mongo.yaml
apiVersion: apps/v1
kind: Deployment
```

```

metadata:
  name: mongodb-deployment
  labels:
    app: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
        - name: mongodb
          image: mongo
          ports:
            - containerPort: 27017
          env:
            - name: MONGO_INITDB_ROOT_USERNAME
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret
                  key: mongo-root-username
            - name: MONGO_INITDB_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret
                  key: mongo-root-password
      ---
    apiVersion: v1
    kind: Service
    metadata:
      name: mongodb-service
    spec:
      selector:
        app: mongodb
      ports:
        - protocol: TCP
          port: 27017
          targetPort: 27017

```

mongo-configmap.yaml

Se crea un despliegue de clase ConfigMap con el nombre “*mongodb-configmap*”, este deploy se encarga de la configuración para obtener la conexión a la base de datos identificada como el servicio “*mongodb-service*”.

```
# mongo-configmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongodb-configmap
data:
  database_url: mongodb-service
```

mongo-express.yaml

En este archivo se crea el despliegue de un contenedor y un servicio expuesto del mismo con la imagen de mongo-express para exponerlo en el navegador.

El deploy del pod de nombre “*mongo-express*” se conectara a la base de datos de mongo en el servicio “*mongodb-service*”, para hacer esta conexión necesita obtener las credenciales de user y password del deploy tipo Secret “*mongodb-secret*” y la url de la database del deploy tipo config “*mongodb-configmap*”.

El servicio se crea con el nombre “*mongo-express-service*” de la app “*mongo-express*”, el servicio es de tipo LoadBalancer para poder ser expuesto con el protocolo TCP/IP que tendrá el puerto 30000 en el nodo, y el 8081 para ser expuesto.

```
# mongo-express.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-express
  labels:
    app: mongo-express
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo-express
  template:
    metadata:
      labels:
        app: mongo-express
    spec:
      containers:
        - name: mongo-express
          image: mongo-express
          ports:
            - containerPort: 8081
          env:
            - name: ME_CONFIG_MONGODB_ADMINUSERNAME
```

```

      valueFrom:
        secretKeyRef:
          name: mongodb-secret
          key: mongo-root-username
- name: ME_CONFIG_MONGODB_ADMINPASSWORD
  valueFrom:
    secretKeyRef:
      name: mongodb-secret
      key: mongo-root-password
- name: ME_CONFIG_MONGODB_SERVER
  valueFrom:
    configMapKeyRef:
      name: mongodb-configmap
      key: database_url
---
apiVersion: v1
kind: Service
metadata:
  name: mongo-express-service
spec:
  selector:
    app: mongo-express
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
      nodePort: 30000

```

Despliegue desde la terminal de comandos

Con el cluster de docker habilitado iniciado verificamos la instalación correcta con:

```
kubectl version
```

```

C:\WINDOWS\system32>kubectl version
Client Version: version.Info{Major:"1", Minor:"22", GitVersion:"v1.22.5", GitCommit:"5c99e2ac2ff9a3c549d9ca665e7bc05a3e18f07e", GitTreeState:"clean", BuildDate:"2021-12-16T08:38:33Z", GoVersion:"go1.16.12", Compiler:"gc", Platform:"windows/amd64"}
Server Version: version.Info{Major:"1", Minor:"22", GitVersion:"v1.22.5", GitCommit:"5c99e2ac2ff9a3c549d9ca665e7bc05a3e18f07e", GitTreeState:"clean", BuildDate:"2021-12-16T08:32:32Z", GoVersion:"go1.16.12", Compiler:"gc", Platform:"linux/amd64"}

```

Para ejecutar los siguientes comandos es necesario ubicarse en el directorio donde se guardaron los archivos

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>
```

Ejecutamos el primer comando para desplegar el servicio mongodb-secret

```
kubectl apply -f mongo-secret.yaml
```

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>kubectl apply -f mongo-secret.yaml
secret/mongodb-secret created
```

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>kubectl get secret
NAME                                TYPE                                DATA  AGE
default-token-gdtp9                kubernetes.io/service-account-token  3      2d22h
mongodb-secret                     Opaque                              2      11m
```

Con el archivo mongo.yaml de despliega la imagen de mongo y se crea el servicio.

```
kubectl apply -f mongo.yaml
```

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>kubectl apply -f mongo.yaml
deployment.apps/mongodb-deployment created
service/mongodb-service created
```

Comprobamos que todo va bien

```
kubectl get pod && kubectl get service
```

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>kubectl get pod && kubectl get service
NAME                                READY    STATUS    RESTARTS   AGE
mongodb-deployment-8f6675bc5-7t62s  1/1      Running   0           4m
NAME                                TYPE     CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
kubernetes                         ClusterIP 10.96.0.1      <none>        443/TCP    2d22h
mongodb-service                    ClusterIP 10.100.32.199 <none>        27017/TCP  4m1s
```

Se despliega el configMap

```
kubectl apply -f mongo-configmap.yaml
```

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>kubectl apply -f mongo-configmap.yaml
configmap/mongodb-configmap created
```

```
kubectl get configmap
```

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>kubectl get configmap
NAME                                DATA  AGE
kube-root-ca.crt                   1      2d22h
mongodb-configmap                  1      7m16s
```

Enseguida se despliega la imagen de mongo-express y el servicio, este puede demostrar un poco pues tiene que descargar los archivos.

```
kubectl apply -f mongo-express.yaml
```

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>kubectl apply -f mongo-express.yaml
deployment.apps/mongo-express created
service/mongo-express-service created
```

Comprobamos de nuevo que los deploy y services estén activos.

```
kubectl get pod && kubectl get service
```

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>kubectl get pod && kubectl get service
```

NAME	READY	STATUS	RESTARTS	AGE
mongo-express-78fcf796b8-fgfmm	1/1	Running	0	6m12s
mongodb-deployment-8f6675bc5-7t62s	1/1	Running	0	15m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	2d22h
mongo-express-service	LoadBalancer	10.96.55.214	<pending>	8081:30000/TCP	6m11s
mongodb-service	ClusterIP	10.100.32.199	<none>	27017/TCP	15m

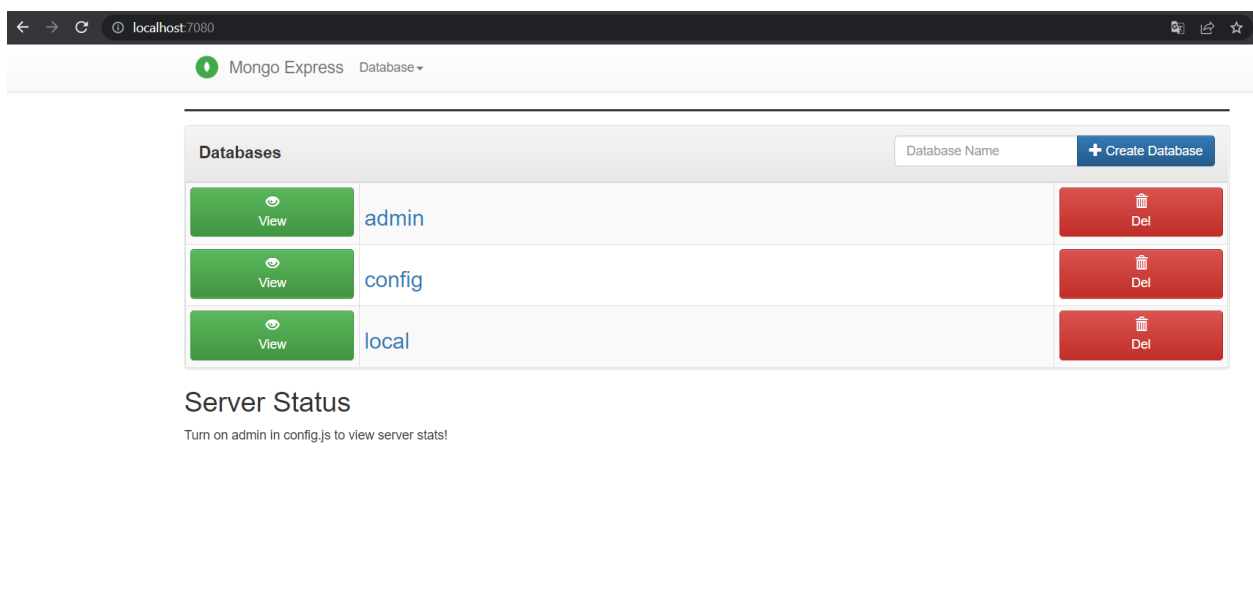
Ahora necesitamos exponer el servicio de express para verlo desde el navegador

El servicio podría verse desde la ip del pod y el puerto asignado (`http://{POD_IP}:{NODE_PORT}/`), en caso de presentar problemas se puede exponer en un puerto en el localhost con el siguiente comando, elegimos el servicio *mongo-express-service* para este ejemplo lo trae del puerto 8081 del nodo y lo expone en el 7080 del localhost :

```
kubectl port-forward service/mongo-express-service 7080:8081
```

```
C:\Users\edgar\Documents\kubernetes_home\mongo-express-deploy>kubectl port-forward service/mongo-express-service 7080:8081
Forwarding from 127.0.0.1:7080 -> 8081
Forwarding from [::1]:7080 -> 8081
```

El resultado es poder ver la interface de mongo con express desde el navegador



Documentación con datos de configuración de las imágenes utilizadas:

Mongo - Official Image | Docker Hub

MongoDB document databases provide high availability and easy scalability.

 https://hub.docker.com/_/mongo

Mongo-express - Official Image | Docker Hub


Web-based MongoDB admin interface, written with Node.js and express

 https://hub.docker.com/_/mongo-express

Documentación de ayuda para la creación de archivos de Deployment yaml:

Entender los Objetos de Kubernetes


Esta página explica cómo se representan los objetos de Kubernetes en la API de Kubernetes, y cómo puedes definirlos en formato .yaml. Entender los Objetos de Kubernetes Los Objetos de

 <https://kubernetes.io/es/docs/concepts/overview/working-with-objects/kubernetes-objects/>

kubern

Deployment

Un controlador de Deployment proporciona actualizaciones declarativas para los Pods y los ReplicaSets. Cuando describes el estado deseado en un objeto Deployment, el controlador del

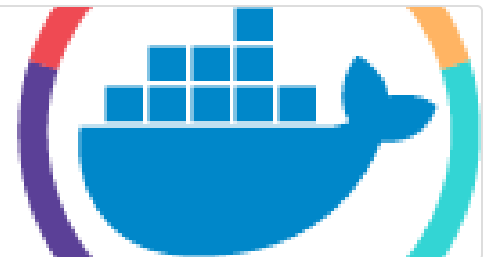
 <https://kubernetes.io/es/docs/concepts/workloads/controllers/deployment/>

kubern

Deploy to Kubernetes


Estimated reading time: 6 minutes Now that we've demonstrated that the individual components of our application run as stand-alone containers, it's time to arrange for them to be managed by an

 <https://docs.docker.com/get-started/kube-deploy/>



Kubernetes Deployment Tutorial with YAML - Kubernetes Book


Everyone running applications on Kubernetes cluster uses a deployment. It's what you use to scale, roll out, and roll back versions of your applications. With a deployment, you tell Kubernetes how many

 <https://matthewpalmer.net/kubernetes-app-developer/articles/kubernetes-deployment-tutorial-example-yaml.html>



Deployment | Kubernetes Engine Documentation | Google Cloud

This page describes Kubernetes Deployment objects and their use in Google Kubernetes Engine (GKE). Deployments represent a set of multiple, identical Pods with no unique identities. A Deployment runs

 <https://cloud.google.com/kubernetes-engine/docs/concepts/deployment>

