Texto, Logotipo

Descripción generada automáticamente con confianza media

Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Máster Universitario en Desarrollo y Operaciones (DevOps)

Construcción de imágenes Docker y despliegue en un clúster de Kubernetes (grupal)

Integrantes:

* Youssef Benmansour
* Álvaro Santamaría
* Juan Gómez Hernández
* Juan Pablo Muñoz

Asignatura:

* Contenedores

# Generación y publicación de la imagen de Docker

En esta actividad se va a desplegar la aplicación de NodeJS con mongoDB usada en la actividad 2 y, por lo tanto, se va a reutilizar la imagen de Docker almacenada en Dockerhub que posteriormente se va a desplegar en un clúster de Kubernetes en EKS.

Los pasos a seguir, por lo tanto, son similares al caso anterior:

* Se crea el archivo de Dockerfile y .dockeringore
* Se ejecuta docker buildx build . –tag repositorio/imagen:tag
* Se ejecuta docker push repositorio/imagen:tag para almacenar la imagen en el repositorio correspondiente

# Configuración inicial del cluster

## Creación del cluster

Lo primero que se hace es crear el clúster en AWS. En este caso se va a hacer por la línea de comandos.

“eksctl create cluster --name actividad3-cluster --version 1.21 --region us-east-2 --nodegroup-name standard-workers --node-type t3.micro --nodes 3 --nodes-min 1 --nodes-max 5 --managed”

Texto

Descripción generada automáticamente con confianza media

Después de unos minutos, la creación del clúster ha finalizado correctamente

Texto

Descripción generada automáticamente

Se accede a la consola de AWS para confirmar la creación del clúster

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

# Configuración del cluster

## Definición del namespace

A continuación, se crea un namespace en el que se crearán los recursos que se desplieguen en el clúster que se acaba de crear, usando el siguiente comando.

“kubectl create namespace actividad3-dev”



## Creación del contexto asociado

Se crea un nuevo contexto para el nuevo namespace mediante el siguiente comando

“kubectl config set-context actividad3-ctxt --namespace=actividad3-dev --cluster=arn:aws:eks:us-east-2:673294157311:cluster/actividad3-cluster --user=arn:aws:eks:us-east-2:673294157311:cluster/actividad3-cluster”



Asignamos el contexto a usar, el nuevo contexto con el comando

“kubectl config use-context actividad3-ctxt”



Comprobamos la configuración

Texto

Descripción generada automáticamente

# Definición y despliegue de los recursos

Se crean los ficheros de despliegue y servicios para la aplicación NodeJS y mongo (Se adjunta junto con la siguiente memoria donde se podrán ver en detalle).

Cabe destacar que se han agrupado en un mismo fichero los recursos a crear, separando las capas de front y base de datos.

En el fichero web-deployment.yml se define el recurso de tipo deployment y de tipo service para la parte web y en el fichero mongo-deployment.yml se define el recurso de tipo deployment y de tipo service. Lo más destacable es que estos recursos se asocian al nuevo namespace creado.

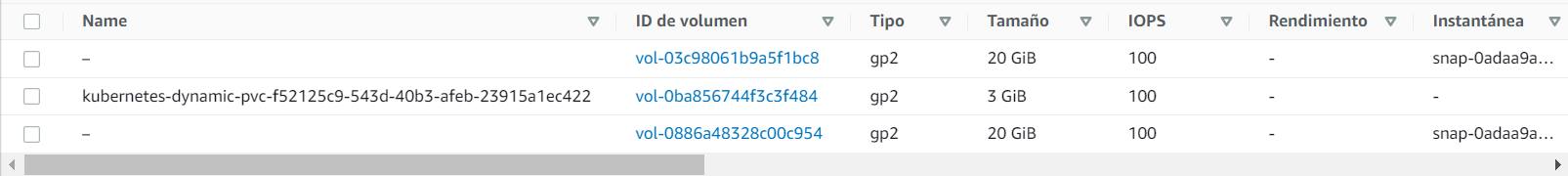
Adicionalmente se incluye el archivo de mongo-storage.yml que incluye los recursos necesarios para el almacenamiento permanente de los datos de mongodb. Este archivo consta de un storageClass con provisioner kubernetes.io/aws-ebs, que permite la configuración de volúmenes ebs de forma automática en AWS cuando sean solicitados. Para esto, se incluye un pvc que haga uso de este storageClass el cual será utilizado por el deployment de mongo posteriormente asociado al volumen montado sobre la ruta “/data/db”.

Debido a que es necesaria la presencia del volumen con anterioridad a la creación de los diferentes deployments, se ejecutan los siguientes comandos en orden.

“kubectl apply -f .\mongo-storage.yml”



Se puede comprobar que se han creado los volúmenes correspondientes accediendo a la consola de AWS



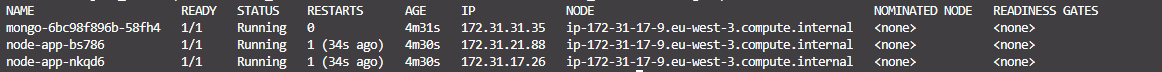
Una vez se han creado la solicitud de volúmenes, podemos proseguir con la creación de los recursos de la aplicación con el siguiente comando en la carpeta correspondiente

kubectl apply -f .



Tras esto, ejecutamos el siguiente comando para ver los pods asociados al namespace creado anteriormente

“kubectl get pods --all-namespaces -o wide”



Listamos todos los recursos creados

Texto

Descripción generada automáticamente

Accedemos a los logs de los pods y comprobamos como está conectada la aplicación a la base de datos.





Adicionalmente, se han incluido los logs correspondientes a los diferentes pods junto con los archivos de configuración “.yml”.

Tras esto, como se han definido los servicios de acceso a los recursos, podemos acceder a la dirección del LoadBalancer y comprobar que funciona y responde de forma correcta.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

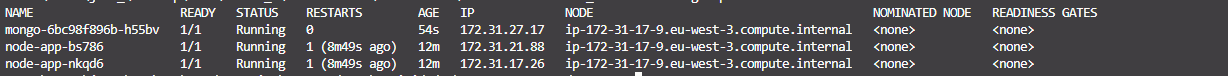
Descripción generada automáticamente

Finalmente, podemos eliminar los pods relacionados con la base de datos para comprobar que los volúmenes almacenan de forma permanente los datos, de forma independiente al ciclo de vida de los pods.

Kubectl delete pods mongo-<prefix> (en este caso mongo-6bc98f896b-58fh4)



Podemos comprobar que se ha creado un nuevo pod asociado al Deployment



Se puede comprobar de esta forma que se almacenan los datos en el volumen correspondiente

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente