[MDAS] - Principios y herramientas de desarrollo
Entregable 2 - Kubernetes (v1.0, 22/10/25)
Rylan James Graham

1. [2 ptos] Indica para cada uno de los siguientes elementos si se encuentra en un nodo master o un nodo worker del cluster.

A. [0,4 ptos] etcd (base de datos interna de k8s)
Master

B. [0,4 ptos] El contenedor de un pod en marcha
Worker

C. [0,4 ptos] Un volúmen de tipo hostPath
Worker

D. [0,4 ptos] Kubernetes API
Master

E. [0,4 ptos] Una base de datos MongoDB para nuestro backend y desplegada mediante un stateful set
Worker

2. [2 ptos] Indica para cada uno de los siguientes elementos si se encuentra dentro de nuestro cluster, o por el contrario está fuera.

A. [0,4 ptos] Un pod de test creado a partir de la imágen nginx
Dentro

B. [0,4 ptos] El registro de contenedores con nuestras imágenes Docker
Fuera

C. [0,4 ptos] Los datos de un *PersistentVolume* en un cluster de GoogleCloud
Dentro

D. [0,4 ptos] El componente que gestiona la IP de un *service* tipo *LoadBalancer*
Fuera

E. [0,4 ptos] Los datos de un *volume* de tipo *emptyDir*
Dentro

3. [2 pto] Un compañero nos indica que un backend que consumimos ya está disponible en el entorno de TEST como deployment "backend-0" y el servicio "backend-svc-0", y hay 4 réplicas. Ahora procedemos a obtener la información que necesitamos para validarlo y configurar nuestra aplicación.

A. [0,4 ptos] Indica solamente 2 comandos puedes usar para ver cuantos pods hay disponibles y en qué estado están. *Nota: recordad qué elementos gestionan el número de pods, no se aceptan soluciones basadas en contar manualmente o uso de 'wc'. La solución debe ser simple y valer tanto para 4 como 42 replicas.*

kubectl get deployments -n namespace
kubectl get pods -n namespace

B. [0,4 ptos] ¿Qué comando podemos usar para ver el tipo de servicio y los puertos que expone? Añade también un ejemplo de la salida del comando donde se vean claramente la información requerida.

CMD: kubectl get service backend-svc-0 -n TEST
Ouput:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
backend-svc-0	ClusterIP	10.100.20.5	<none>	80/TCP	12m

C. [0,4 ptos] Vemos que los pods están Ready y el servicio levantado y con IP. Sin embargo no logramos conectar a través del servicio. ¿Qué elementos debemos validar en el service y el pod para asegurar que ambos están conectados? Indica claramente su nombre y posición en sus respectivas descripciones YAML. *Nota: puedes usar "dot notation" (por ejemplo, la propiedad "template" de un deployment sería 'spec.template').*

Debemos validar que las etiquetas definidas en el selector del servicio coincidan exactamente con las etiquetas asignadas a los pods en el deployment.

- *En el **Service:** Validar la propiedad **spec.selector***
- *En el **Deployment:** Validar la propiedad **spec.template.metadata.labels***

D. [0,4 ptos] Después de arreglar la configuración, procedemos a obtener los parámetros para nuestra aplicación. ¿Qué DNS (hostname) podemos usar para configurar nuestro hostname?*Nota: existen múltiples opciones, pero sólo se aceptarán como válidas aquellas que no dependen del número de pods existentes.*

*Podemos usar el nombre del servicio: **backend-svc-0***

E. [0,4 ptos] Tras resolver la configuración del servicio-pod, vemos que estamos mal de recursos en el cluster. Nosotros sólo queremos validar nuestra integración y no necesitamos HA (alta disponibilidad) ¿qué comando usarías para reducir el número de instancias a 1?

kubectl scale deployment backend-0 --replicas=1 -n TEST

4. [2 ptos] Crea un pod de forma declarativa (mediante fichero) con las siguientes especificaciones:

- ***Imagen: nginx***
- ***Version: 1.25.3***
- ***Labels: app=nginx-server***
- ***Recursos mínimo***
  - ***CPU: 100m***
  - ***Mem: 256 M***
- ***Recursos máximos***
  - ***CPU: 200m***
  - ***Mem: 512 M***

Realiza un despliegue en Kubernetes y responde a las siguientes preguntas:

A. [0,3 ptos] Adjunta el fichero usado.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
  labels:
    app: nginx-server
spec:
  containers:
  - name: nginx-container
    image: nginx:1.25.3
    resources:
      requests:
        cpu: "100m"
        memory: "256Mi"
      limits:
        cpu: "200m"
        memory: "512Mi"
```

B. [0,3 ptos] ¿Qué comando usarías para obtener las últimas 10 líneas de la salida estándar (logs generados por la aplicación)?

kubectl logs nginx-pod --tail=10

C. [0,3 ptos] ¿Qué comando usarías para obtener la IP interna del pod? ¿Es una IP pública? Añade también un ejemplo de la salida del comando donde se vean claramente la información requerida.

CMD: kubectl get pod nginx-pod -o wide
Output:
NAME READY STATUS RESTARTS AGE IP NODE NOM. NODE
nginx-pod 1/1 Running 0 5m 10.x.x.x kind-worker <none>

D. [0,3 ptos] ¿Qué comando utilizarías para entrar dentro del pod?

kubectl exec -it nginx-pod -- /bin/bash

E. [0,4 ptos] Necesitas validar el contenido publicado en NGINX, ¿qué acciones debes llevar a cabo para conectarte desde local (sin requerir una IP pública)? Indica todas las acciones, incluyendo especialmente las urls para navegador o terminal. Asume que tienes acceso directo al cluster pero no hay ningún servicio LoadBalancer o Ingress disponible.

Como no tenemos Ingress ni LoadBalancer, debemos crear un túnel directo desde nuestra máquina local al puerto del contenedor.

- *kubectl port-forward pod/nginx-pod 8080:80*
- *Cmd: curl http://localhost:8080*

F. [0,4 ptos] Dado un cluster con 2 workers con:

Nodo 1:

CPU disponible: 100m

Mem disponible: 256M

Nodo 2:

CPU disponible: 200m

Mem disponible: 1024M

Indica en qué nodos se puede desplegar el pod y explica brevemente porqué.

El pod se puede desplegar en AMBOS NODOS (Nodo 1 y Nodo 2). El planificador de Kubernetes toma decisiones basándose en los Requests, NO en los Limits.

5. [2 ptos] Anteriormente copiábamos nuestra configuración en la imágen de nuestra app. Pero nos comenta un compañero que podemos externalizar TODA la configuración y así simplificar el CI al prescindir de pasos cómo generar imágen y almacenarla en nuestro registro securizado.

Construye un deployment con las siguientes especificaciones:

- ***Imágen: hashicorp/http-echo***

- ***Versión: 0.2.3***
- ***Replicas: 2***
- ***Variables de entorno:***
  - ***BACKEND_HOST=my-backend***
  - ***BACKEND_PORT=9090***
- ***Ficheros de configuración:***
  - ***"config.properties" montado en /etc/config con el contenido***

    color.good=purple

    color.bad=yellow

    allow.textmode=true
- ***Variable de entorno:***
  - ***BACKEND_ACCESS_TOKEN=V2hhdCBpcyA0Mj8=***

A. [1,2 ptos] Indica todos los pasos, ficheros y comandos utilizados.

*1. Crear el **ConfigMap** (configmap.yaml) Este objeto almacena el fichero config.properties*

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: echo-config
data:
  config.properties: |
    color.good=purple
    color.bad=yellow
    allow.textmode=true
```

*2. Crear el **Secret** (secret.yaml) Este objeto almacena el token sensible.*

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: echo-secret
type: Opaque
data:
  # El valor ya está codificado en Base64 según el enunciado
  BACKEND_ACCESS_TOKEN: V2hhdCBpcyA0Mj8=
```

*3. Crear el **Deployment** (deployment.yaml) Inyectamos las variables simples, leemos el secreto como variable de entorno y montamos el ConfigMap como volumen.*

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: echo-deployment
  labels:
    app: echo-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: echo-app
  template:
    metadata:
      labels:
        app: echo-app
    spec:
      containers:
      - name: http-echo
        image: hashicorp/http-echo:0.2.3
        # http-echo necesita un argumento para arrancar, añadimos uno por defecto
        args:
          - "-text=MDAS Kubernetes"
        ports:
        - containerPort: 5678 # Puerto por defecto de esta imagen
        env:
        # 1. Variables de entorno literales (Datos NO sensibles)
        - name: BACKEND_HOST
          value: "my-backend"
        - name: BACKEND_PORT
          value: "9090"
        # 2. Variable de entorno desde Secret (Dato SENSIBLE)
        - name: BACKEND_ACCESS_TOKEN
          valueFrom:
            secretKeyRef:
              name: echo-secret
              key: BACKEND_ACCESS_TOKEN
        # 3. Montaje del volumen
        volumeMounts:
        - name: config-vol
          mountPath: /etc/config
          readOnly: true
      volumes:
      - name: config-vol
        configMap:
          name: echo-config
```

4. CMDs para desplegar:

- *kubectl apply -f configmap.yaml*
- *kubectl apply -f secret.yaml*
- *kubectl apply -f deployment.yaml*

B. [0,4 ptos] ¿Que has usado para configurar los datos no sensibles? ¿Y los sensibles? Indica para cada uno si es sensible o no.

He clasificado y configurado los datos de la siguiente manera:

- *ConfigMap (echo-config): Usado para los datos NO sensibles.*
  Contiene: El fichero config.properties (colores, modos de texto).

- *Variables de Entorno Literales: Usado para datos NO sensibles.*
  Contiene: BACKEND_HOST y BACKEND_PORT.

- *Secret (echo-secret): Usado para los datos SENSIBLES.*
  Contiene: BACKEND_ACCESS_TOKEN.

C. [0,4 pto] Expón el deployment mediante un servicio en el puerto 8080. Adjunta fichero de configuración.

Creamos el fichero service.yaml. Mapeamos el puerto 8080 del servicio al puerto 5678 porque es el default de la imagen hashicorp/http-echo

```yaml
apiVersion: v1
kind: Service
metadata:
  name: echo-service
spec:
  type: ClusterIP  # Por defecto, suficiente para comunicación interna
  selector:
    app: echo-app
  ports:
    - protocol: TCP
      port: 8080          # Puerto expuesto por el servicio
      targetPort: 5678    # Puerto donde escucha el contenedor
```

CMD: kubectl apply -f service.yaml

Fe de erratas

Versión	*Fecha*	*Cambio(s)*