

Abstract geometric lines in the top left corner, consisting of several overlapping, irregular polygons and lines in a light beige color.

EJERCICIOS DE KUBERNETES K8S

Paloma Pérez de Madrid

ÍNDICE

1. Creación de un entorno trabajo con Kubernetes (desde una máquina virtual en vez desde GCP)
2. Elegir un ejercicio
 - I. Levantar un pod en GKE con cloud scheduler que ejecute una tarea en py
 - II. Sobre un cluster dataproc, lanzar el statefulset de H2O y ejecutar un código que os copiaré
 - III. Pasar el Composer que os he dejado a un servicio K8S

ÍNDICE

- 1. Creación de un entorno trabajo con Kubernetes (desde una máquina virtual en vez desde GCP)**
2. Elegir un ejercicio
 - I. Levantar un pod en GKE con cloud scheduler que ejecute una tarea en py
 - II. Sobre un cluster dataproc, lanzar el statefulset de H2O y ejecutar un código que os copiaré
 - III. Pasar el Composer que os he dejado a un servicio K8S

CREACIÓN DE UN ENTORNO TRABAJO CON KUBERNETES

Minikube es una herramienta que permite **ejecutar clústeres de Kubernetes** de un solo nodo en una máquina local o en una máquina virtual. Proporciona una forma fácil y rápida de experimentar con Kubernetes para el desarrollo y la prueba de aplicaciones sin necesidad de un entorno en la nube.

- Descargar e instalar minikube: `curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64`
- Darle permisos al ejecutable descargado: `sudo chmod +x minikube-linux-amd64`
- Mover el ejecutable a 'usr/local/bin' (PATH, para ejecutarlo desde cualquier ubicación): `sudo mv minikube-linux-amd64 /usr/local/bin/minikube`
- INICIAR MiniKube: `minikube start --vm-driver=vmware|virtualbox|none|docker` (driver con el que ejecutará el cluster de kubernetes)
 - Si tienes VMWare puedes poner el driver vmware pero necesitas un archivo vmrun (yo no lo tengo con la versión gratuita)
 - Para buscar el archivo: `which vmrun`
 - Si lo pones a `none` → iniciará Minikube utilizando el driver 'none', lo que ejecutará el clúster de Kubernetes directamente en mi sistema host (recomendado solo para expertos)
 - `sudo apt-get install -y docker.io`
 - `sudo usermod -aG docker $USER` # **Agrega tu usuario al grupo Docker** para poder ejecutar comandos de Docker sin sudo
 - `newgrp docker`
 - `minikube start --driver=docker` # Iniciar minikube

Minikube stop + `sudo minikube delete` para borrar un minikube creado
Si no funciona:
`sudo rm -rf ~/.minikube`
`sudo rm -rf /etc/kubernetes/`
`sudo rm -rf /var/lib/minikube/`
`sudo rm -rf /var/lib/kubelet/`

CREACIÓN DE UN ENTORNO TRABAJO CON KUBERNETES

```
ubuntu@ubuntu:~$ minikube start --driver=docker
🐸 minikube v1.32.0 on Ubuntu 20.04
🌟 Using the docker driver based on user configuration
📌 Using Docker driver with root privileges
👍 Starting control plane node minikube in cluster minikube
🚚 Pulling base image ...
📦 Downloading Kubernetes v1.28.3 preload ...
> preloaded-images-k8s-v18-v1...: 403.35 MiB / 403.35 MiB 100.00% 13.13 M
> gcr.io/k8s-minikube/kicbase...: 453.61 MiB / 453.90 MiB 99.94% 12.21 Mi
🔥 Creating docker container (CPUs=2, Memory=2200MB) ...
🐳 Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
  ■ Generating certificates and keys ...
  ■ Booting up control plane ...
  ■ Configuring RBAC rules ...
🔗 Configuring bridge CNI (Container Networking Interface) ...
🔍 Verifying Kubernetes components...
  ■ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟 Enabled addons: default-storageclass, storage-provisioner
💡 kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
🚀 Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
ubuntu@ubuntu:~$
```

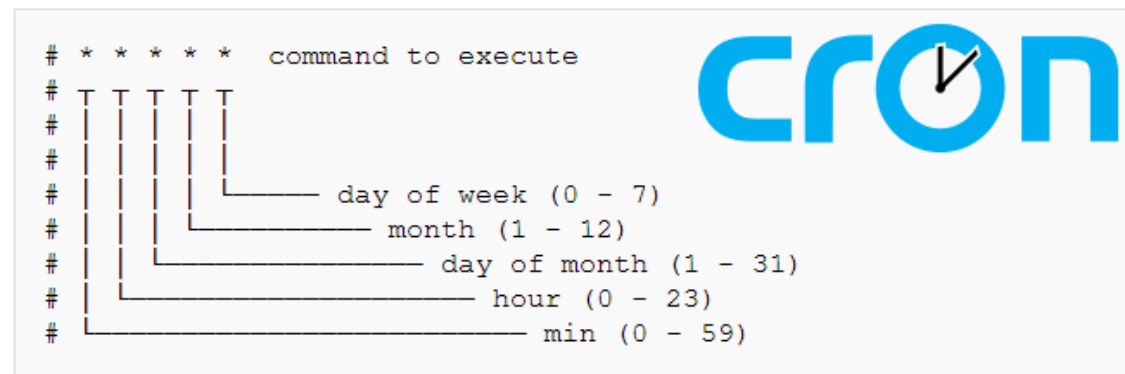
ÍNDICE

1. Creación de un entorno trabajo con Kubernetes
(desde una máquina virtual en vez desde GCP)
2. **Elegir un ejercicio**
 - I. **Levantar un pod en GKE con cloud scheduler que ejecute una tarea en py**
 - II. Sobre un cluster dataproc, lanzar el statefulset de H2O y ejecutar un código que os copiaré
 - III. Pasar el Composer que os he dejado a un servicio K8S

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

Para no consumir recursos de GCP y que nos cobren vamos a simular el Cloud Scheduler

- En vez de levantar un pod en GCP usaremos minikube para simularlo
- Programaremos la tarea con la tarea con CronJob en vez de con cloud scheduler
- Pasaremos a comprobar y verificar el funcionamiento



2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

1. Crear un Pod en Minikube:

- Definir un archivo de configuración YAML para el pod que incluya un contenedor que ejecute la tarea en Python.
- Usar ``kubectl apply -f archivo.yaml`` para desplegar el pod en tu clúster de Minikube.

2. Programar la tarea con CronJob:

- Definir un objeto CronJob en YAML que describa la tarea que deseas ejecutar periódicamente, especificando el pod que creaste anteriormente como su plantilla.
- Utilizar ``kubectl apply -f archivo-cronjob.yaml`` para crear el CronJob en el clúster de Minikube.

3. Verificar el funcionamiento:

- Una vez se haya desplegado el CronJob, verificar que esté funcionando correctamente utilizando ``kubectl get cronjob`` y ``kubectl describe cronjob``.

4. Ver los registros de la tarea:

- Puedes usar ``kubectl logs`` para ver los registros del pod y asegurarte

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

1. Crear un Pod en Minikube:

Archivo de Python a ejecutar (script.py):

```
import time
import random

def main():
    print("Iniciando tarea de procesamiento de datos...")
    # Simulamos un proceso de procesamiento de datos que tarda entre 1 y 5 segundos
    tiempo_de_proceso = random.randint(1, 5)
    print(f"Procesando datos... (Estimado: {tiempo_de_proceso} segundos)")
    time.sleep(tiempo_de_proceso)
    print("Tarea de procesamiento de datos completada")

if __name__ == "__main__":
    main()
```

Dockerfile (yo lo he guardado en la misma carpeta q script.py)

```
# Dockerfile
FROM python:3.9-slim

COPY script.py /app/script.py

CMD ["python", "/app/script.py"]
```

Dockerfile copiará el archivo script.py al directorio /app dentro del contenedor y configurará el comando por defecto para ejecutar el script Python cuando se inicie el contenedor.

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

1. Crear un Pod en Minikube:

```
# Archivo YAML → pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: python-task-pod
spec:
  containers:
    - name: python-task-container
      image: imagen_practica_sie_kubernetes
      imagePullPolicy: Never # nunca descargar la imagen desde
un repositorio remoto y solo usa la imagen local que ya está
presente en el nodo de Kubernetes.
```

Construir la imagen del Docker:

(en mi caso desde Desktop/kubernetes donde está script.py y dockerfile)

`docker login`

`docker build -t imagen_practica_sie_kubernetes .`

```
ubuntu@ubuntu:~/Desktop$ cd kubernetes/
ubuntu@ubuntu:~/Desktop/kubernetes$ vim script.py
ubuntu@ubuntu:~/Desktop/kubernetes$ vim dockerfile
ubuntu@ubuntu:~/Desktop/kubernetes$ vim pod.yaml
ubuntu@ubuntu:~/Desktop/kubernetes$ docker build -t imagen_practica_sie_kubernetes .
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  4.096kB
Step 1/3 : FROM python:3.9-slim
3.9-slim: Pulling from library/python
13808c22b207: Pull complete
6c9a484475c1: Pull complete
4a81626d2c6b: Pull complete
e8690706172b: Pull complete
8c60f620628d: Pull complete
Digest: sha256:fd8693fc6115345452a19654c1388bc9a1029cda7b98e1bce45ede67e74190f0
Status: Downloaded newer image for python:3.9-slim
--> 14f71980249a
Step 2/3 : COPY script.py /app/script.py
--> 7ce0d1a432ea
Step 3/3 : CMD ["python", "/app/script.py"]
--> Running in ebeb21be8e32
Removing intermediate container ebeb21be8e32
--> b1456f8a8676
Successfully built b1456f8a8676
Successfully tagged imagen_practica_sie_kubernetes:latest
ubuntu@ubuntu:~/Desktop/kubernetes$
```

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

1. Crear un Pod en Minikube:

Comprobar que existe:

`docker images`

```
docker images: command not found
ubuntu@ubuntu:~/Desktop/kubernetes$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE
imagen_practica_sie_kubernetes  latest      b1456f8a8676     About a minute ago  126MB
```

Desplegar el pod en Minikube:

`kubectl apply` → desplegar el pod en el clúster de Minikube

`kubectl apply -f pod.yaml`

(Yo he tenido que hacer antes `sudo snap install kubectl --classic`)

```
ubuntu@ubuntu:~/Desktop/kubernetes$ kubectl apply -f pod.yaml
pod/python-task-pod created
```

`kubectl delete pod <<python-task-pod>>` → si la has cambiado el pod.yaml tienes que borrar el anterior)

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

1. Crear un Pod en Minikube:

IMPORTANTE

Si creas el Docker con “`docker build -t imagen_practica_sie_kubernetes .`” se te crea de forma LOCAL

¡Queremos crearlo en MiniKube!

- Guarda la imagen etiquetada en un archivo tar: `docker save -o imagen_practica_sie_kubernetes.tar imagen_practica_sie_kubernetes:latest`
- Transfiere el archivo tar a la máquina virtual de Minikube: `minikube cp imagen_practica_sie_kubernetes.tar minikube:/home/docker/imagen_practica_sie_kubernetes.tar`
- En la máquina virtual de Minikube, carga la imagen desde el archivo tar: `minikube ssh "docker load -i /home/docker/imagen_practica_sie_kubernetes.tar"`


```
ubuntu@ubuntu:~/Desktop/kubernetes$ minikube ssh "docker load -i /home/docker/imagen_practica_sie_kubernetes.tar"
1f00ff201478: Loading layer [=====>] 77.83MB/77.83MB
bfc9081d1eb2: Loading layer [=====>] 9.552MB/9.552MB
57f5b08e62c4: Loading layer [=====>] 32.31MB/32.31MB
53451a08b688: Loading layer [=====>] 5.12kB/5.12kB
e25966a5c9f4: Loading layer [=====>] 11.7MB/11.7MB
3eb0a292e089: Loading layer [=====>] 2.56kB/2.56kB
Loaded image: imagen_practica_sie_kubernetes:latest
```

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

2. Programar la tarea con CronJob:

Definir un objeto CronJob en YAML que describa la tarea que queremos ejecutar periódicamente, especificando el pod que hemos creado anteriormente como su plantilla

```
# cronjob.yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: mi-cronjob
spec:
  schedule: "*/1 * * * *"
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: python-task-container
              image: imagen_practica_sie_kubernetes # nombre imagen del pod anterior
              imagePullPolicy: Never
              restartPolicy: OnFailure
```



- `*/1`: Ejecutar cada valor posible en el primer campo, que representa los minutos.
- `*`: Ejecutar cada valor posible en los campos restantes, que representan las horas, día del mes, mes y día de la semana.

la expresión `*/1 * * * *` significa "ejecutar el trabajo cada minuto, independientemente de la hora, día del mes, mes o día de la semana".

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

2. Programar la tarea con CronJob:

Crear el CronJob en MiniKube:

```
kubectl apply -f cronjob.yaml
```

```
ubuntu@ubuntu:~/Desktop/kubernetes$ kubectl apply -f cronjob.yaml
cronjob.batch/mi-cronjob created
```

Verificar estado de cronjob:

```
kubectl get cronjob
```

```
ubuntu@ubuntu:~/Desktop/kubernetes$ kubectl get cronjob
NAME          SCHEDULE    SUSPEND   ACTIVE   LAST SCHEDULE   AGE
mi-cronjob    */1 * * * * False      1         12s       67s
```

```
kubectl describe cronjob mi-cronjob # "mi-cronjob" es el nombre de mi CronJob
```

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

2. Programar la tarea con CronJob:

`kubectl describe cronjob mi-cronjob` # "mi-cronjob" es el nombre de mi CronJob

```
ubuntu@ubuntu:~/Desktop/kubernetes$ kubectl describe cronjob mi-cronjob
Name:                mi-cronjob
Namespace:           default
Labels:              <none>
Annotations:         <none>
Schedule:            */1 * * * *
Concurrency Policy:   Allow
Suspend:             False
Successful Job History Limit: 3
Failed Job History Limit: 1
Starting Deadline Seconds: <unset>
Selector:            <unset>
Parallelism:         <unset>
Completions:         <unset>
Pod Template:
  Labels:  <none>
  Containers:
    python-task-container:
      Image:          imagen_practica_sie_kubernetes
      Port:          <none>
      Host Port:     <none>
      Environment:   <none>
      Mounts:        <none>
      Volumes:       <none>
  Last Schedule Time: Thu, 11 Apr 2024 10:36:00 -0700
  Active Jobs:      mi-cronjob-28547615, mi-cronjob-28547616
Events:
  Type    Reason            Age   From                  Message
  ----    -
  Normal  SuccessfulCreate   88s   cronjob-controller    Created job mi-cronjob-28547615
  Normal  SuccessfulCreate   28s   cronjob-controller    Created job mi-cronjob-28547616
ubuntu@ubuntu:~/Desktop/kubernetes$ ss
```

Resumen

- Nombre: `mi-cronjob`
- Programación: Cada minuto (`*/1 * * * *`)
- Política de concurrencia: Permite la ejecución simultánea de múltiples trabajos (`Allow`)
- Suspensión: No está suspendido (`False`)
- Límite de historial de trabajos exitosos: 3
- Límite de historial de trabajos fallidos: 1
- Plantilla del Pod: Contenedor usando la imagen `imagen_practica_sie_kubernetes`
- Última vez programada: Thu, 11 Apr 2024 10:36:00 -0700
- Trabajos activos: `mi-cronjob-28547615`, `mi-cronjob-28547616`

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

2. Programar la tarea con CronJob:

Para ver lo que ejecuta script.py tenemos que usar `kubectl logs`

```
mi-cronjob-28551406-n27mc    0/1    Pending    0          2m13s
mi-cronjob-28551407-sm69x    0/1    Pending    0          73s
mi-cronjob-28551408-8f5qr    0/1    Pending    0          13s
python-task-pod             0/1    CrashLoopBackOff  28 (25s ago)  45h
ubuntu@ubuntu:~/Desktop/kubernetes$ kubectl logs python-task-pod
Iniciando tarea de procesamiento de datos...
Procesando datos... (Estimado: 2 segundos)
Tarea de procesamiento de datos completada
ubuntu@ubuntu:~/Desktop/kubernetes$ kubectl logs mi-cronjob-28551408-8f5qr
```

Como se observa, script.py se ha ejecutado sin errores

2. LEVANTAR UN POD EN GKE CON CLOUD SCHEDULER QUE EJECUTE UNA TAREA EN PY

2. Programar la tarea con CronJob:

Si volvemos a ejecutarlo y esperamos un minuto, veremos que el pod “mi-cronjob-28551412-t4xf4” llega a los 60 segundos y el pod de python pasa al estado de “Running” y luego “Completed”

mi-cronjob-28551412-t4xf4	0/1	Pending	0	60s
mi-cronjob-28551413-52sk4	0/1	Pending	0	0s
python-task-pod	1/1	Running	29 (5m12s ago)	45h

mi-cronjob-28551412-t4xf4	0/1	Pending	0	61s
mi-cronjob-28551413-52sk4	0/1	Pending	0	1s
python-task-pod	0/1	Completed	29 (5m13s ago)	45h

USAR UN DEPLOYMENT EN VEZ DE UN POD

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mi-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mi-aplicacion
  template:
    metadata:
      labels:
        app: mi-aplicacion
    spec:
      containers:
        - name: mi-contenedor
          image: imagen_practica_sie_kubernetes
          imagePullPolicy: Never
          ports:
            - containerPort: 80
```

Pod:

- *Unidad básica de ejecución*: Un Pod es la unidad más básica de ejecución en Kubernetes.
- *Contenedor o conjunto de contenedores*: Un Pod puede contener uno o más contenedores que comparten el mismo espacio de red y almacenamiento.
- *Efímero por naturaleza*: Los Pods son efímeros y pueden ser eliminados y recreados en cualquier momento.
- *No es escalable automáticamente*: Si necesitas escalar la aplicación, debes crear múltiples Pods manualmente o mediante algún controlador como Deployment.
- *No tiene estrategias de actualización o rollback integradas*: No proporciona mecanismos integrados para realizar actualizaciones o rollbacks de manera controlada.

Deployment:

- *Manejo de Pods*: Un Deployment es un objeto de Kubernetes que maneja la creación y administración de múltiples réplicas de un Pod.
- *Escalabilidad automática*: Puedes escalar automáticamente el número de réplicas de los Pods administrados por un Deployment según la demanda utilizando el escalado automático o manual.
- *Estrategias de actualización y rollback*: Los Deployments proporcionan estrategias integradas para actualizar la versión de la aplicación de manera controlada y realizar rollbacks en caso de problemas.
- *Despliegue declarativo*: Se define de manera declarativa en un archivo YAML, lo que permite una fácil reproducción y gestión del estado deseado de la aplicación.
- *Monitoreo y manejo de errores*: Los Deployments supervisan constantemente el estado de los Pods y realizan automáticamente acciones para mantener el estado deseado, como la creación de nuevos Pods en caso de fallas.

USAR UN DEPLOYMENT EN VEZ DE UN POD

Comandos útiles

Aplicar el archivo YAML para crear el Deployment:

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

Este comando aplicará el archivo YAML para crear el Deployment en tu clúster de Kubernetes.

Verificar el estado del Deployment:

```
kubectl get deployments
```

Este comando muestra una lista de todos los Deployments en tu clúster y su estado actual.

Verificar los pods asociados con el Deployment:

```
kubectl get pods
```

Este comando muestra una lista de todos los pods asociados con el Deployment y su estado actual.

Describir detalles del Deployment:

```
kubectl describe deployment <nombre_del_deployment>
```

Proporciona detalles detallados sobre el Deployment, incluidos los eventos relacionados, las estrategias de actualización, etc.

Escalar el Deployment:

```
kubectl scale deployment <nombre_del_deployment> --  
replicas=<número_de_réplicas>
```

Este comando escala el número de réplicas del Deployment según el número especificado.

Actualizar el Deployment:

Puedes editar el archivo `deployment.yaml` para realizar actualizaciones en el Deployment, como cambiar la imagen del contenedor. Luego, puedes aplicar los cambios con `kubectl apply -f deployment.yaml`.

Eliminar el Deployment:

```
kubectl delete deployment <nombre_del_deployment>
```

Este comando eliminará el Deployment y todos los pods asociados con él.

Ver eventos del Deployment:

```
kubectl describe deployment <nombre_del_deployment>
```