KUBERNETES – ACTIVIDAD 7

Cristóbal Suárez Abad

OPTATIVA 2º ASIR

Contenido

1) Instalación de Minikube.

Nos dirigimos a la guía oficial:
https://minikube.sigs.k8s.io/docs/start/?arch=%2Flinux%2Fx86-64%2Fstable%2Fbinary+download

Seleccionamos el sistema operativo que vamos a usar. En nuestro caso Linux. Primero debemos descargarnos los binarios:

curl -LO
https://github.com/kubernetes/minikube/releases/latest/download/minikube-linux-amd64

```
root@debian:~# curl -LO https://github.com/kubernetes/minikube/releases/l
atest/download/minikube-linux-amd64
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  C
urrent
                                 Dload  Upload   Total   Spent    Left  S
peed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
    0
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
    0
  2  133M    2 2764k    0     0  5019k      0  0:00:27 --:--:--  0:00:27
 31  133M   31 41.8M    0     0  27.1M      0  0:00:04  0:00:01  0:00:03
 58  133M   58 77.9M    0     0  30.7M      0  0:00:04  0:00:02  0:00:02
 80  133M   80  107M    0     0  30.3M      0  0:00:04  0:00:03  0:00:01
100  133M  100  133M    0     0  30.6M      0  0:00:04  0:00:04 --:--:--
34.4M
```

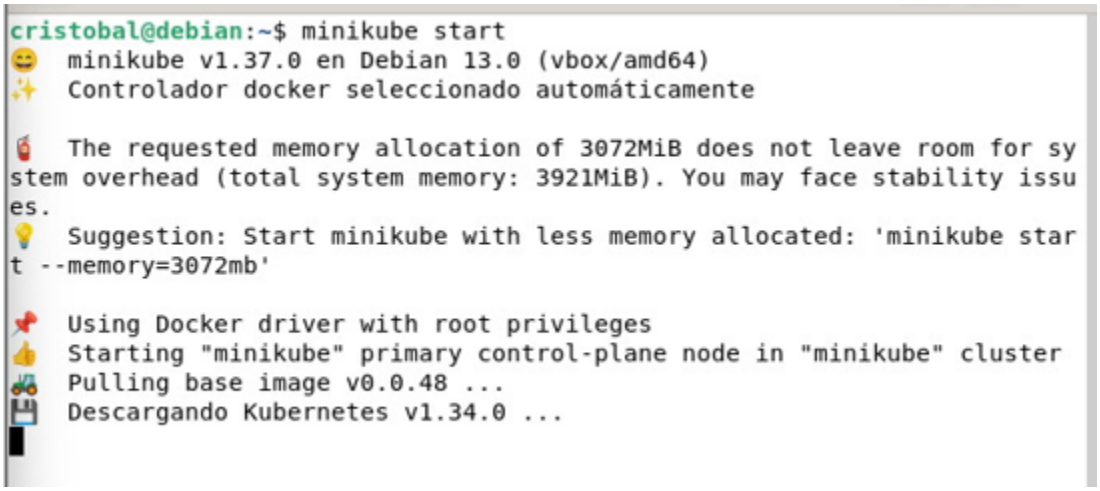
Y luego podemos instalarlo:

sudo install minikube-linux-amd64 /usr/local/bin/minikube && rm minikube-linux-amd64

```
root@debian:~# sudo install minikube-linux-amd64 /usr/local/bin/minikube
&& rm minikube-linux-amd64
```

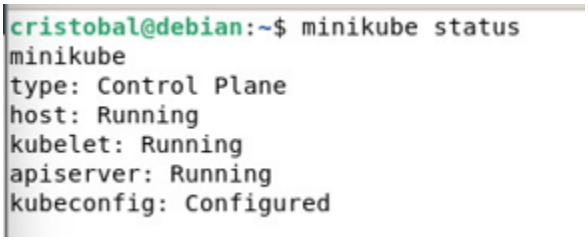
A continuación lo iniciamos:

minikube start



Comprobamos el estado:

minikube status



2) Instalamos kubectl:

En nuestro caso elegimos la versión para Linux:
https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/

De igual manera, primero nos descargamos los binarios:

curl -LO https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl

Después lo instalamos:

sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

Y por último comprobamos la versión:

kubectl version –client

```
root@debian:~# curl -LO "https://dl.k8s.io/release/$(curl -L -s https://d
l.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  C
urrent
                                 Dload  Upload   Total   Spent    Left  S
peed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--
100   138  100   138    0     0    857      0 --:--:-- --:--:-- --:--:--
  862
  8 57.7M    8 5123k    0     0  11.6M      0  0:00:04 --:--:--  0:00:04
 69 57.7M   69 40.2M    0     0  28.2M      0  0:00:02  0:00:01  0:00:01
100 57.7M  100 57.7M    0     0  29.8M      0  0:00:01  0:00:01 --:--:--
35.0M
root@debian:~# sudo install -o root -g root -m 0755 kubectl /usr/local/bi
n/kubectl
root@debian:~# kubectl version --client
Client Version: v1.34.1
Kustomize Version: v5.7.1
root@debian:~#
```

3) Verificación del clúster:

minikube status: nos ayuda a ver el estado del clúster.

kubectl get nodes: saber el estado de cada nodo.

minikube stop: parar el servicio de kubernetes.

```
cristobal@debian:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

cristobal@debian:~$ kubectl get nodes
NAME       STATUS   ROLES           AGE    VERSION
minikube   Ready    control-plane   4m6s   v1.34.0
cristobal@debian:~$ minikube stop
✋  Stopping node "minikube"  ...
🛑  Apagando "minikube" mediante SSH...
🛑  1 node stopped.
cristobal@debian:~$ minikube start
😄  minikube v1.37.0 en Debian 13.0 (vbox/amd64)
✨  Using the docker driver based on existing profile

🧯  The requested memory allocation of 3072MiB does not leave room for sy
stem overhead (total system memory: 3921MiB). You may face stability issu
es.
💡  Suggestion: Start minikube with less memory allocated: 'minikube star
t --memory=3072mb'

👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.48 ...
🔄  Restarting existing docker container for "minikube" ...
```

minikube delete: elimina el clúster y todos los recursos añadidos a él.

4) Conceptos básicos:

a) Despliegue de Pod:

La unidad más pequeña de Kubernetes, capaz de desplegar uno o más contenedores. Se usan archivos "**.yaml**" y se pueden definir red, almacenamiento y otros recursos (CPU, RAM, etc).

Ejemplo: Tipo "Pod", nombre del pod "my-pod" y se establece el contenedor (en este caso uno solo).

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
  - name: nginx-container
    image: nginx:1.21
    ports:
    - containerPort: 80
```

Para aplicarlo usamos: **kubectl apply -f NOMBREARCHIVO.yaml**

```
cristobal@debian:~/proyectos_kubernets$ kubectl apply -f 01.yaml
pod/my-pod created
cristobal@debian:~/proyectos_kubernets$
```

b) Implantaciones:

Implantaciones o Despliegue. Se usa para gestionar el ciclo de vida de los pods. Lo que hace es establecer el número deseado de réplicas de un pod y también gestiona las actualizaciones y retrocesos (upgrades and downgrades).

Ejemplo de archivo ".yaml" de despliegue: Tipo "Deployment", nombre del despliegue, cuantas réplicas, se establecen etiquetas y nombre e imagen que se va a usar para los contenedores. Muy importante la información que se encuentra en las "labels" puesto que es lo que más adelante permitirá identificar a los recursos.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
spec:
 replicas: 3
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:1.21
    ports:
    - containerPort: 80
```

Se instala de igual manera que el anterior: **kubectl apply -f deployment.yaml**

```
cristobal@debian:~/proyectos_kubernets$ kubectl apply -f 02.yaml
deployment.apps/nginx-deployment created
```

c) Servicios:

Permite dar acceso red estable a los Pods, aunque cambien de dirección IP. Esto se hace debido a la corta vida de los Pods. Si un nuevo Pod toma el lugar del anterior, los clientes tendrían que ir cambiando de IP cada vez. Para evitar eso se asocian los Pods que tienen una etiqueta determinada con una IP.

Ejemplo: Todos los Pods con la etiqueta "nginx" usarán el puerto 80. Se usa tipo "NodePort" porque "expone el Servicio en un puerto accesible fuera del clúster".

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
spec:
 selector:
  app: nginx
 ports:
 - protocol: TCP
  port: 80
  targetPort: 80
 type: NodePort
```

Aplicamos: **kubectl apply -f service.yaml**

```
cristobal@debian:~/proyectos_kubernets$ kubectl apply -f 03.yaml
service/nginx-service created
```

Puedes ver una descripción del servicio usando:

kubectl describe service nginx-service

```
cristobal@debian:~/proyectos_kubernets$ kubectl describe service nginx-se
rvice
Name:                     nginx-service
Namespace:                default
Labels:                   <none>
Annotations:              <none>
Selector:                 app=nginx
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.108.244.216
IPs:                      10.108.244.216
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  30884/TCP
Endpoints:                10.244.0.9:80,10.244.0.8:80,10.244.0.7:80
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
cristobal@debian:~/proyectos_kubernets$
```

5) Espacios de nombres.

Sirven para organizar y aislar recursos dentro de un clúster. Los agrupa.

Ejemplo:

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-namespace
```

Aplicamos: **kubectl apply -f namespace.yaml**

Ver listado de espacios: **kubectl get namespaces**

```
cristobal@debian:~/proyectos_kubernets$ kubectl apply -f namespace.yaml
namespace/my-namespace created
cristobal@debian:~/proyectos_kubernets$ kubectl get namespaces
NAME              STATUS   AGE
default           Active   33m
kube-node-lease   Active   33m
kube-public       Active   33m
kube-system       Active   33m
my-namespace      Active   7s
cristobal@debian:~/proyectos_kubernets$
```

Una vez que hemos creado el espacio, podemos crear recursos estableciendo el nombre del espacio de nombres en la sección de metadatos de tu recurso.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: my-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
```

image: nginx:1.21
ports:
- containerPort: 80

La aplicamos como las anteriores. Y si queremos ver los recursos asociados a este espacio de nombres usamos:
kubectl get deployment -n my-namespace

```
cristobal@debian:~$ kubectl get deployment -n my-namespace
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           90m
cristobal@debian:~$ S
```

6) Gestionar tu primera aplicación con Kubernetes.

Vamos a crear un servidor web nginx y exponerlo a la web. Lo escalaremos (crear réplicas) y actualizaremos con "actualización continua".

a) Crear un despliegue de aplicación web sencillo

Se hace un despliegue sencillo de un contenedor con una sola réplica.

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
spec:
 replicas: 1
 selector:
  matchLabels:
   app: nginx
 template:
  metadata:
   labels:
    app: nginx
  spec:
   containers:
   - name: nginx
    image: nginx:latest
    ports:
    - containerPort: 80
```

Lo aplicamos: **kubectl apply -f nginx-deployment.yaml**

```
cristobal@debian:~/proyectos_kubernets$ nano nginx-deployment.yaml
cristobal@debian:~/proyectos_kubernets$ kubectl apply -f nginx-deployment
.yaml
deployment.apps/nginx-deployment configured
cristobal@debian:~/proyectos_kubernets$
```

Comprobación: **kubectl get pods**

```
cristobal@debian:~/proyectos_kubernets$ kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
my-pod                              1/1     Running             0          24m
nginx-deployment-78764869f5-7m5lc   1/1     Running             0          17m
nginx-deployment-7d584c4448-f7xgs   0/1     ContainerCreating   0          9s
```

b) <u>Exponer la aplicación con un servicio.</u>

Vamos a usar un Servicio para exponer el servidor que acabamos de crear al mundo exterior. Usaremos un servicio tipo NodePort. Fijate que en la etiqueta “app” se pone el nombre del contenedor del Despliegue.

```
apiVersion: v1
kind: Service
metadata:
 name: nginx-service
spec:
 selector:
  app: nginx
 ports:
 - protocol: TCP
  port: 80
  targetPort: 80
 type: NodePort
```

Aplicamos: **kubectl apply -f nginx-service.yaml**

```
cristobal@debian:~/proyectos_kubernets$ nano nginx-service.yaml
cristobal@debian:~/proyectos_kubernets$ kubectl apply -f nginx-service.ya
ml
service/nginx-service unchanged
cristobal@debian:~/proyectos_kubernets$
```

Usamos minikube para obtener la URL del servidor web:

minikube service nginx-service –url

```
cristobal@debian:~/proyectos_kubernets$ minikube service nginx-service --
url
http://192.168.49.2:30884
```

c) <u>Escalar la aplicación.</u>

Vamos a crear replicas del Pod para que este pueda hacer frente ante una hipotética carga masiva de usuarios que entren en la página web. Como en nuestro caso ya lo tenemos creado, usaremos el comando:

kubectl scale deployment nginx-deployment --replicas=3

Este establece que sean 3 réplicas en el despliegue "**nginx-deployment**" que creamos anteriormente. Pero podríamos haberlo creado en el ".yaml" desde el principio.

Ahora usamos "**kubectl get pods**" para comprobar que se han creado los otros Pods (las réplicas).

```
cristobal@debian:~/proyectos_kubernets$ kubectl scale deployment nginx-deployment --replicas=3
deployment.apps/nginx-deployment scaled
cristobal@debian:~/proyectos_kubernets$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-pod                              1/1     Running   0          27m
nginx-deployment-7d584c4448-28j2c   1/1     Running   0          19s
nginx-deployment-7d584c4448-br2hl   1/1     Running   0          19s
nginx-deployment-7d584c4448-f7xgs   1/1     Running   0          3m21s
```

Para ver los “endpoints” a los que está conectado el servicio “nginx-service” que creamos previamente.

kubectl get endpoints nginx-service

```
cristobal@debian:~/proyectos_kubernets$ kubectl get endpoints nginx-service
Warning: v1 Endpoints is deprecated in v1.33+; use discovery.k8s.io/v1 EndpointSlice
NAME            ENDPOINTS                                           AGE
nginx-service   10.244.0.13:80,10.244.0.14:80,10.244.0.15:80   20m
cristobal@debian:~/proyectos_kubernets$ ^C
cristobal@debian:~/proyectos_kubernets$ kubectl get endpointslices
NAME                  ADDRESSTYPE   PORTS   ENDPOINTS                           AGE
kubernetes            IPv4          8443    192.168.49.2                        49m
nginx-service-cg4bz   IPv4          80      10.244.0.13,10.244.0.15,10.244.0.14   21m
cristobal@debian:~/proyectos_kubernets$
```

Funciona, pero se indica que está obsoleto. Mejor usar:

kubectl get endpointslices

d) <u>Actualizaciones continuas:</u>

Nos permite llevar a cabo la actualización de los pods de un clúster de un modo que el servicio no se vea interrumpido. En nuestro caso tenemos tres réplicas en funcionamiento, por lo tanto, irá actualizando un nodo cada vez sin parar el servidor. Además, comprobará que cada Pod está en buen estado, si no parará el proceso.

Vamos a cambiar nginx a la versión 1.23

kubectl set image deployment/nginx-deployment nginx=nginx:1.23

Podemos ver el estado del proceso con:

kubectl rollout status deployment/nginx-deployment

```
cristobal@debian:~/proyectos_kubernets$ kubectl set image deployment/nginx-deployment nginx=nginx:1.23
deployment.apps/nginx-deployment image updated
cristobal@debian:~/proyectos_kubernets$ kubectl rollout status deployment/nginx-deployment
Waiting for deployment "nginx-deployment" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 1 out of 3 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 2 out of 3 new replicas have been updated...
Waiting for deployment "nginx-deployment" rollout to finish: 1 old replicas are pending termination...
Waiting for deployment "nginx-deployment" rollout to finish: 1 old replicas are pending termination...
deployment "nginx-deployment" successfully rolled out
cristobal@debian:~/proyectos_kubernets$
```

7) Gestionar los recursos de Kubernetes.

Ver Pods: **kubectl get pods**

Ver Servicios: **kubectl get services**

Ver Despliegues: **kubectl get deployments**

```
cristobal@debian:~/proyectos_kubernets$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
my-pod                              1/1     Running   0          38m
nginx-deployment-74cc86b97c-kqckp   1/1     Running   0          6m2s
nginx-deployment-74cc86b97c-kx9v2   1/1     Running   0          6m7s
nginx-deployment-74cc86b97c-zln69   1/1     Running   0          6m29s
cristobal@debian:~/proyectos_kubernets$ kubectl get services
NAME            TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
kubernetes      ClusterIP   10.96.0.1        <none>        443/TCP        58m
nginx-service   NodePort    10.108.244.216   <none>        80:30884/TCP   30m
cristobal@debian:~/proyectos_kubernets$ kubectl get deployments
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   3/3     3            3           32m
cristobal@debian:~/proyectos_kubernets$
```

Para obtener más información sobre un recurso en concreto usamos “describe”:

kubectl describe deployment nginx-deployment

```
cristobal@debian:~/proyectos_kubernets$ kubectl describe deployment nginx-deployment
Name:                   nginx-deployment
Namespace:              default
CreationTimestamp:      Tue, 28 Oct 2025 09:18:15 +0100
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 3
Selector:               app=nginx
Replicas:               3 desired | 3 updated | 3 total | 3 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=nginx
  Containers:
   nginx:
    Image:         nginx:1.23
    Port:          80/TCP
    Host Port:     0/TCP
    Environment:   <none>
    Mounts:        <none>
  Volumes:         <none>
  Node-Selectors:  <none>
  Tolerations:     <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
```

También puedes ver el log de un Pod en concreto:

kubectl logs (NOMBRE DEL POD)

```
cristobal@debian:~/proyectos_kubernets$ kubectl logs nginx-deployment-74cc86b97c-kqckp
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/10/28 08:44:34 [notice] 1#1: using the "epoll" event method
2025/10/28 08:44:34 [notice] 1#1: nginx/1.23.4
2025/10/28 08:44:34 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2025/10/28 08:44:34 [notice] 1#1: OS: Linux 6.12.41+deb13-amd64
2025/10/28 08:44:34 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/10/28 08:44:34 [notice] 1#1: start worker processes
2025/10/28 08:44:34 [notice] 1#1: start worker process 29
2025/10/28 08:44:34 [notice] 1#1: start worker process 30
cristobal@debian:~/proyectos_kubernets$
```

Borrar recursos:

Borrar Pods: **kubectl delete pod <pod-name>**

```
cristobal@debian:~/proyectos_kubernets$ kubectl delete pod my-pod
pod "my-pod" deleted from default namespace
```

Si esto se hace en un Despliegue que tenga establecido un número de réplicas, ten en cuenta que otro Pod se generará automáticamente para sustituir al borrado. Abajo se ve como se borra el "**kqckp**" y acto seguid se crea uno nuevo, el "**z7pfh**".

```
cristobal@debian:~/proyectos_kubernets$ kubectl delete pod nginx-deployme
nt-74cc86b97c-
nginx-deployment-74cc86b97c-kqckp  nginx-deployment-74cc86b97c-zln69
nginx-deployment-74cc86b97c-kx9v2
cristobal@debian:~/proyectos_kubernets$ kubectl delete pod nginx-deployme
nt-74cc86b97c-kqckp
pod "nginx-deployment-74cc86b97c-kqckp" deleted from default namespace
cristobal@debian:~/proyectos_kubernets$ kubectl delete pod nginx-deployme
nt-74cc86b97c-
nginx-deployment-74cc86b97c-kx9v2  nginx-deployment-74cc86b97c-zln69
nginx-deployment-74cc86b97c-z7pfh
```

Para este caso se tendría que eliminar el Despliegue completo:

kubectl delete deployment nginx-deployment

```
cristobal@debian:~/proyectos_kubernets$ kubectl delete deployment nginx-d
eployment
deployment.apps "nginx-deployment" deleted from default namespace
cristobal@debian:~/proyectos_kubernets$ kubectl get pods
No resources found in default namespace.
cristobal@debian:~/proyectos_kubernets$
```

Si quisiéramos eliminar todo, usamos: **kubectl delete all --all**

```
cristobal@debian:~/proyectos_kubernets$ kubectl delete all --all
service "kubernetes" deleted from default namespace
service "nginx-service" deleted from default namespace
cristobal@debian:~/proyectos_kubernets$
```