

Ampliación Kubernetes

Tahir Farooq
cos

Contenido

Introducción y aprendiendo a trabajar con ficheros yaml.....	2
Escalado horizontal de pods(HPA)	9
Dashboard	17
Dashboard sin utilizar token	17
Dashboard con token de acceso	20
Conclusiones	27
Anexos	28
Bibliografía	30

Introducción y aprendiendo a trabajar con ficheros yaml

En esta guía, vamos a profundizar sobre el clúster Kubernetes. En primer lugar, debemos tener una instalación de clúster Kubernetes con dos nodos trabajadores y un máster, sobre el clúster debemos tener instalado una red de intercomunicación entre nodos como puede ser Flannel o Calico(mi caso). Esta instalación fue realizada con el módulo de Ansible.

En primer lugar, vamos a borrar el despliegue de nginx que hemos realizado mediante comandos, para poder crear el mismo despliegue a través de ficheros “.yaml”, de esta forma poder familiarizarnos con los descriptores de los ficheros yaml:

En primer lugar, vemos los Deployments que tenemos en la siguiente captura:

```
[root@my-master-1 ~]# kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
my-first-nginx      2/2     2             2           8d
```

Lo borramos y consultamos que se han borrado el deployment y los pods asociados a dicho deployment con los siguientes comandos:

```
[root@my-master-1 ~]# kubectl delete deployments my-first-nginx
deployment.apps "my-first-nginx" deleted
[root@my-master-1 ~]# kubectl get deployments
No resources found in default namespace.
[root@my-master-1 ~]# kubectl get pods
No resources found in default namespace.
```

Sin embargo, nos damos cuenta de que el servicio asociado a este deployment no se borra tras realizar el borrado del deployment, lo podemos consultar con el siguiente comando:

```
[root@my-master-1 ~]# kubectl get services
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
kubernetes          ClusterIP   10.233.0.1     <none>       443/TCP          16d
my-first-nginx      LoadBalancer 10.233.44.231 <pending>    80:30475/TCP     8d
```

Tenemos que borrar el servicio expresamente de nuestro clúster, como se aprecia en la siguiente captura:

```
[root@my-master-1 ~]# kubectl delete service my-first-nginx
service "my-first-nginx" deleted
[root@my-master-1 ~]# kubectl get services
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)          AGE
kubernetes          ClusterIP   10.233.0.1     <none>       443/TCP          16d
```

Ahora vamos a trabajar con los ficheros “yaml”, en primer lugar, vamos a crear un pod que despliega un servicio mongodb.

Vamos a crear el fichero mongodb.yaml con el siguiente contenido:

```
[root@my-master-1 ~]# cat mongodb.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mongodb
spec:
  containers:
  - image: mongo
    name: mongodb
```

Ahora vamos a desplegar este manifiesto para crear el pod de la siguiente manera:

```
[root@my-master-1 ~]# kubectl apply -f mongodb.yaml
pod/mongodb created
[root@my-master-1 ~]# kubectl get pods
NAME      READY   STATUS             RESTARTS   AGE
mongodb   0/1     ContainerCreating   0          7s
```

Vemos que el estado del pod es “No Ready”, ya que acabamos de crear el pod (“Container Creating”) y demora un poco de tiempo en iniciarse.

Ahora vamos a ejecutar dicho pod que todavía no sabemos en qué nodo se está ejecutando. Para ello, vamos a ejecutar el nodo con el comando “kubectl exec [Pod] [Comando]” como se ve en la siguiente captura.

```
[root@my-master-1 ~]# kubectl exec -it mongodb /bin/bash
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
root@mongodb:/# ls
bin  data  docker-entrypoint-initdb.d  home      lib      media  opt   root  sbin  sys  usr
root dev  etc                js-yaml.js  lib64    mnt    proc  run   srv   tmp  var
root@mongodb:/# exit
exit
```

Nos indica un warning al ejecutar el pod, quiere decir que la forma de ejecución será eliminada en una versión futura (“Deprecated”), ya que ahora debemos realizarlo añadiendo “--” antes del comando.

En la captura anterior, podemos observar que hemos ejecutado el pod mongodb con el parámetro “- it” para poder interactuar con el pod desde dentro. Nos salimos del pod haciendo “exit”.

Si queremos ver información relativa al pod, lo podemos ver mediante el comando “kubectl describe pod [Pod]”. En la siguiente captura lo podemos observar:

```
[root@my-master-1 ~]# kubectl describe pod mongodb
Name:         mongodb
Namespace:    default
Priority:      0
Node:         my-node-2/192.168.1.98
Start Time:   Wed, 20 Jan 2021 11:43:54 +0100
Labels:       <none>
Annotations:  cnf.projectcalico.org/podIP: 10.233.67.13/32
              cnf.projectcalico.org/podIPs: 10.233.67.13/32
Status:       Running
IP:           10.233.67.13
IPs:          IP: 10.233.67.13
Containers:
  mongodb:
    Container ID:  docker://e96423c706a0c233222ff7b7ec9072c93b489a2c4fa966789b34b5f3c71ca03
    Image:         mongo
    Image ID:      docker-pullable://mongo@sha256:001400644bfc27b5da634ee09b95b4129566e2d4dcc6d27bd403b10cff9191b
    Port:         <none>
    Host Port:    <none>
    State:        Running
      Started:    Wed, 20 Jan 2021 11:44:45 +0100
    Ready:        True
    Restart Count: 0
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-n4tj4 (ro)
Conditions:
  Type            Status
  Initialized      True
  Ready           True
  ContainersReady True
  PodScheduled     True
```

Podemos observar información relativa al pod como nombre :“mongodb”, la fecha de creación, la IP que tiene dicho POD: 10.233.67.13. Este pod solamente se puede acceder desde dentro del clúster ya que no tiene ningún servicio asociado para poder ser accedido de forma pública.

Dicho pod tiene un único contenedor : “mongodb”.

A continuación, podemos observar que fue asignado al nodo2.

Events:				
Type	Reason	Age	From	Message
Normal	Scheduled	84s	default-scheduler	Successfully assigned default/mongodb to my-node-2
Normal	Pulling	84s	kubelet	Pulling image "mongo"
Normal	Pulled	36s	kubelet	Successfully pulled image "mongo" in 47.466395134s
Normal	Created	34s	kubelet	Created container mongodb
Normal	Started	33s	kubelet	Started container mongodb

Ahora mismo estamos realizando pruebas de mongodb, pero si en un futuro fallara, podemos observar los logs del pod con el comando “kubectl logs mongodb”

```
[root@my-master-1 ~]# kubectl logs mongodb
{"t":{"$date":"2021-01-20T10:44:48.027+00:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"main","msg":"Automatically
disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2021-01-20T10:44:48.029+00:00"},"s":"W",  "c":"ASIO",    "id":22601,   "ctx":"main","msg":"No TransportL
ayer configured during NetworkInterface startup"}
{"t":{"$date":"2021-01-20T10:44:48.030+00:00"},"s":"I",  "c":"NETWORK",  "id":4648601, "ctx":"main","msg":"Implicit TCP
```

Pero también podemos verificar los logs de otra manera:

Si accedemos al nodo2 y hacemos “Docker ps -a”, veremos la imagen mongodb que acaba de crear:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
5789ef0d9f7d	mongo	"docker-entrypoint.s..."	24 seconds ago	Up 23 seconds
k8s_mongodb_mongodb_default_3bf254bc-c6c9-42b2-aa65-96e027eddb56_0				

Tras saber el contenedor docker que lo está corriendo, podemos meternos en la máquina docker (con los comandos de docker tales como docker attach) y observar los logs desde dentro del contenedor.

Sin embargo, esto ya no lo podremos hacer a partir de la versión 1.22 ya que Kubernetes anuncia oficialmente que a partir de la versión 1.22, va a dejar de utilizar Docker como Runtime subyacente. Quedando alternativas como **containerd (que es una parte de Docker) o CRI-O**. Algunas razones pueden ser debido a que Docker necesita tener un proceso de fondo corriendo siempre y eso consume recursos a diferencia de Podman que no precisa dicho proceso consumiendo recursos de fondo.

Podman es una alternativa bastante atractiva a Docker ya que los comandos de docker son idénticos a los de podman: “podman images” para ver las imágenes, “podman ps ” para listar los contenedores, “podman rm -F \$(podman ps -a -q) para matar todos los contenedores y eliminarlos , lanzamiento de contenedores, etc...

Por tanto, volviendo a Kubernetes, si un nodo no responde como debería en local, no podremos hacer ssh a dicho nodo para ver los logs de docker, porque Docker ya no existiría, únicamente podremos realizar “kubectl exec”.

Esta noticia se puede visualizar en estas referencias^{1 2}.

¹ <https://kubernetes.io/blog/2020/12/02/dont-panic-kubernetes-and-docker/>

² <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.20.md#dockershim-deprecation>

Esta característica se mostrará como deprecated, hasta su eliminación en la versión 1.22 de Kubernetes.

Volviendo a Kubernetes.

Ahora vamos a desplegar un deployment utilizando fichero “yaml”. Para ello, vamos a crear un fichero “yaml” con el siguiente contenido:

```
[root@my-master-1 ~]# cat deploymentnginx.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # Indica al controlador que ejecute 2 pods
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

En este deployment, vamos a ejecutar dos pods bajo el nombre de “nginx-deployment”. Tendrá un total de 2 réplicas, en el apartado “spec” podremos visualizar la información de los contenedores tales como la imagen de nginx que vamos a utilizar y el puerto de escucha.

Por tanto, vamos a crearlos en primer lugar con el comando “kubectl apply -f [archivo.yaml]”

```
[root@my-master-1 ~]# kubectl apply -f deploymentnginx.yaml
deployment.apps/nginx-deployment created
```

Vamos a verificar que contamos con dos pods:

```
[root@my-master-1 ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-66b6c48dd5-2bmnj   1/1     Running   0           62s
nginx-deployment-66b6c48dd5-z2gb6   1/1     Running   0           62s
```

Y también podemos observar que se ha creado el deployment:

```
[root@my-master-1 ~]# kubectl get deployment
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment 2/2      2             2           5m45s
```

Ahora vamos a modificar nuestro archivo para que en lugar de crear dos réplicas de pods, le indicamos que tiene que crear 4. Modificamos el fichero “deploymentnginx.yaml” en la línea que indica “replicas”, cambiamos de 2 a 4.

```
[root@my-master-1 ~]# cat deploymentnginx.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # Modificamos el numero de replicas
  template:
```

Volvemos a ejecutar el comando “kubectl apply -f [archivo.yaml]” como se ve en la siguiente captura:

```
[root@my-master-1 ~]# kubectl apply -f deploymentnginx.yaml
deployment.apps/nginx-deployment configured
```

Vamos a comprobar la cantidad de pods y podemos observar que se ha reconfigurado el número de pods, han pasado de 2 a 4.

```
[root@my-master-1 ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-66b6c48dd5-2bmj   1/1     Running   0           10m
nginx-deployment-66b6c48dd5-4r4bs  1/1     Running   0           49s
nginx-deployment-66b6c48dd5-wjkff  1/1     Running   0           49s
nginx-deployment-66b6c48dd5-z2gb6  1/1     Running   0           10m
[root@my-master-1 ~]# kubectl get deployments.apps
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment  4/4     4            4           10m
```

Si en el fichero “.yaml” indicamos que queremos 0 réplicas, y ejecutamos el comando “kubectl apply -f [archivo.yaml]” se destruyen automáticamente todos los pods

Vamos a verificar si se ha destruido el deployment y ver la cantidad de pods con el siguiente comando:

```
[root@my-master-1 ~]# kubectl get pods
No resources found in default namespace.
[root@my-master-1 ~]# kubectl get deployments.apps
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment  0/0     0            0           6m27s
```

Efectivamente, podemos afirmar que se han destruido todos los pods y el deployment cuenta con 0 pods disponibles para el usuario(Columna AVAILABLE).

El estado del deployment se queda de la siguiente manera(kubectl describe deployments [deployment]):

```
Containers:
  nginx:
    Image:      nginx:1.14.2
    Port:       80/TCP
    Host Port:  0/TCP
    Environment: <none>
    Mounts:      <none>
    Volumes:     <none>
Conditions:
  Type           Status  Reason
  ----           -
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  nginx-deployment-66b6c48dd5 (0/0 replicas created)
Events:
  Type     Reason             Age   From              Message
  ----     -
  Normal   ScalingReplicaSet  3m58s deployment-controller Scaled up replica set nginx-deployment-66b6c48dd5 to 4
  Normal   ScalingReplicaSet  3m32s deployment-controller Scaled down replica set nginx-deployment-66b6c48dd5 to 0
```

En la captura superior, en la fila final de eventos, podemos observar que el número de réplicas se ha escalado hacia abajo a 0.

Ahora vamos a crear un servicio mediante archivos yaml.

Tenemos de base este deployment de 3 pods con nginx:

```
[root@my-master-1 ~]# kubectl get deployments.apps
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment    3/3     3            3           6m45s
[root@my-master-1 ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-deployment-66b6c48dd5-7svw7   1/1     Running   0          6m59s
nginx-deployment-66b6c48dd5-9rxjr   1/1     Running   0          6m59s
nginx-deployment-66b6c48dd5-drgzg   1/1     Running   0          6m59s
```

Vamos a crear el fichero .yaml:

```
[root@my-master-1 ~]# cat servicenginx.yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
  type: LoadBalancer
```

Vamos a analizar por partes, la estructura de este fichero yaml:

En primer lugar, indicamos la versión y el tipo de despliegue que estamos realizando, en este caso kind="Service". Queremos desplegar un servicio.

En segundo lugar, empezamos a definir los metadatos de este servicio, tales como nombre y definimos dentro del apartado "labels" una etiqueta "app:nginx".

Por último, indicamos las especificaciones del servicio en el apartado "spec":

Los pods se definen mediante un selector, con la etiqueta app:nginx.

En el apartado de "ports" definimos la parte de la política de acceso. Finalmente en el apartado "type" definimos el acceso desde el exterior, le indicamos que se haga un balanceo de carga.

Vamos a aplicarlo:

```
[root@my-master-1 ~]# kubectl apply -f servicenginx.yaml
service/nginx created
```

Y consultamos si se ha creado el servicio:

```
[root@my-master-1 ~]# kubectl get service
NAME      TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
kubernetes ClusterIP    10.233.0.1     <none>         443/TCP          16d
nginx     LoadBalancer 10.233.50.196  <pending>      80:30757/TCP     6s
```


Si hacemos ssh al nodo1 y al nodo2, podemos observar que tenemos contenedores corriendo sobre dichos nodos.

En el nodo1:

```
[root@my-node-1 ~]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
864989267a48	295c7be07902	"nginx -g 'daemon of..."	18 minutes ago	Up 18 minutes		k8s_nginx_nginx-deployment-66b6c48dd5-dmgzg_default_0ebbb992-12d7-4443-8d82-c818235ae9ef_0

En el nodo2:

```
[root@my-node-2 ~]# docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
545540b0f37a	295c7be07902	"nginx -g 'daemon of..."	19 minutes ago	Up 19 minutes		k8s_nginx_nginx-deployment-66b6c48dd5-9rxjr_default_de92cf70-1a27-4ce3-ba3d-f301d9a54c56_0
7d894a88e1bc	295c7be07902	"nginx -g 'daemon of..."	19 minutes ago	Up 19 minutes		k8s_nginx_nginx-deployment-66b6c48dd5-7svu7_default_7d8f7edd-9a78-4ba3-b289-c4822730102_0
0f71d0f08fa2	k8s.gcr.io/pause:3.3	"/pause"	19 minutes ago	Up 19 minutes		k8s_POD_nginx-deployment-66b6c48dd5-9rxjr_default_de92cf70-1a27-4ce3-ba3d-f301d9a54c56_0

Hasta ahora, hemos desplegado servicios en el mismo espacio de nombre, vamos a ver como desplegar nuestros despliegues en otro espacio de nombre que no sea “por defecto”.

Creamos un Namespace con el siguiente comando, indicando el nombre que queramos:

#Kubectl create namespace “tf10”

Ahora creamos un pod indicando el namespace con el siguiente comando:

#kubectl run tf10 --image=nginx --port --namespace tf10

Ahora podemos observar los pods dentro de un determinado nombre de espacio como se ve en la siguiente captura:

#kubectl get pods --namespace tf10

```
[root@my-master-1 ~]# kubectl get pods --namespace tf10
```

NAME	READY	STATUS	RESTARTS	AGE
nginx	1/1	Running	0	56m

Si queremos eliminar un nombre de espacio, lo realizaremos con el siguiente comando:

#Kubectl delete namespace tf10.

```
[root@my-master-1 ~]# kubectl delete namespace tf10
```

namespace "tf10" deleted

Si queremos fijar un determinado espacio de nombre por defecto, lo realizamos con el siguiente comando:

#kubectl config set-context --current --namespace=tf10

Y si queremos volver al nombre de espacio por defecto, indicamos “--namespace=default”.

Escalado horizontal de pods(HPA)

Ahora vamos a realizar el auto escalado horizontal de pods (HPA).

Para ello, primero instalamos un programa que nos va a recoger las métricas que nosotros definamos en nuestro hpa, que más adelante definiremos.

HPA consulta la fuente de métricas periódicamente y determina si el escalado es requerido por un controlador en función de las métricas que obtiene. Anteriormente se utilizaba heapster, pero en las actuales versiones del clúster, éste ha quedado obsoleto, por tanto, vamos a instalar su sucesor que “metrics-server” que se puede descargar del repositorio oficial de github de Kubernetes³.

La versión actual es v0.4.2 y se pueden instalar a través del fichero “.yaml” que está subido en su propio repositorio, por razones de compatibilidad, he optado por instalar la versión 0.3.7 ya que la última versión no me recogía las métricas de uso de los pods y siempre mostraba <unknown>.

Tras instalar la versión v.0.3.7 que parecía que iba a ser la que iba a funcionar en mi clúster, me dio los mismos problemas que las posteriores versiones, tras varias consultas a repositorios y búsquedas en github me di cuenta de que el problema provenía de unos “flags” que se habían eliminado en estas versiones, por tanto tuve que modificar ligeramente el fichero “.yaml” que venía por defecto. Los cambios han sido añadir estos parámetros en el fichero de configuración:

En el contenedor “metrics-server” hay que añadir las líneas

- --kubelet-preferred-address-types=InternalIP
- --kubelet-insecure-tls

Finalmente el apartado del contenedor de “metrics-server” se queda de la siguiente manera:

```
spec:
  containers:
    - args:
      - --cert-dir=/tmp
      - --secure-port=4443
      - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
      - --kubelet-use-node-status-port
      - --kubelet-insecure-tls
```

Como hemos tenido que descargar el fichero para realizar la modificación, vamos a aplicarlo con el siguiente comando:

#kubectl apply -f componentes.yaml

Ahora que tenemos la versión correcta de “metrics-server” instalada, vamos a ver que los pods de “metrics-server” funcionan correctamente con el siguiente comando:

#kubectl get pods -n kube-system -l k8s-app=metrics-server

Y vemos la salida:

³ <https://github.com/kubernetes-sigs/metrics-server/releases/>

```
[root@my-master-1 ~]# kubectl get pods -n kube-system -l k8s-app=metrics-server
```

NAME	READY	STATUS	RESTARTS	AGE
metrics-server-9fcfc7879-zcqm4	1/1	Running	1	12h

Ahora vamos a crear un deployment y un servicio php-apache y vamos a someterlo a carga para ver que el “load-balancer” hace su función de subida y bajada de número de réplicas de pods en función de la demanda:

Ahora vamos a crear un deployment y un servicio con el siguiente fichero “yaml”:

```
[root@my-master-1 hpapache]# cat apache.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: php-apache
spec:
  selector:
    matchLabels:
      run: php-apache
  replicas: 1
  template:
    metadata:
      labels:
        run: php-apache
    spec:
      containers:
      - name: php-apache
        image: k8s.gcr.io/hpa-example
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 500m
          requests:
            cpu: 200m
```

En la imagen de la izquierda, podemos observar que describimos en primer lugar, un deployment con nombre “php-apache” con 1 réplica.

Cogemos para este ejemplo la imagen del repositorio oficial de Kubernetes: “k8s.gcr.io/hpa-example”.

Limitamos el uso de recursos de cpu a 500milli-cores, este parámetro puede ser cantidad de memoria RAM, número de peticiones que un servidor es capaz de soportar.

Ahora vamos a describir un servicio dentro de mismo archivo.yaml para dicho deployment. Tal y como se aprecia en la siguiente captura.

```
---
apiVersion: v1
kind: Service
metadata:
  name: php-apache
  labels:
    run: php-apache
spec:
  ports:
  - port: 80
  selector:
    run: php-apache
```

Vamos a crear el deployment y el servicio aplicando el fichero yaml que acabamos de generar con el siguiente comando:

#kubectl apply -f apache.yaml

```
[root@my-master-1 hpapache]# kubectl apply -f apache.yaml
deployment.apps/php-apache created
service/php-apache created
```

Ahora vamos a desplegar el horizontal pod autoscaler con el siguiente comando:

```
#kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

Aquí le indicamos que queremos desplegar el deployment con nombre “php-apache” y queremos que el número de pods se mantenga entre 1 y 10. HPA incrementará o disminuirá el número de pods para mantener la media de utilización de CPU que le indicamos= 50 %.

El resultado de dicho comando es el siguiente:

```
[root@my-master-1 hpapache]# kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
horizontalpodautoscaler.autoscaling/php-apache autoscaled
```

Ahora vamos a ver el funcionamiento.

Vamos a comprobar el número de réplicas actuales:

```
[root@my-master-1 hpapache]# kubectl get hpa
NAME                REFERENCE                TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
php-apache          Deployment/php-apache     0%/50%   1         10        1         35s
```

Podemos observar que actualmente el servicio está parado y tenemos una única réplica.

Si queremos ver las métricas de utilización de pods, lo podemos ver con el siguiente comando:

```
#kubectl top pod
```

```
[root@my-master-1 hpapache]# kubectl top pod
NAME                CPU(cores)  MEMORY(bytes)
php-apache-d4cf67d68-1bzcx  1m          5Mi
```

Vamos a consultar nuestro hpa para ver si está funcionando correctamente con el siguiente comando:

```
#kubectl describe hpa php-apache
```

```
[root@my-master-1 hpapache]# kubectl describe hpa php-apache
Name:                php-apache
Namespace:           default
Labels:              <none>
Annotations:         <none>
CreationTimestamp:   Wed, 10 Feb 2021 12:09:14 +0100
Reference:           Deployment/php-apache
Metrics:             ( current / target )
  resource cpu on pods  (as a percentage of request):  0% (1m) / 50%
Min replicas:       1
Max replicas:       10
Deployment pods:     1 current / 1 desired
Conditions:
  Type            Status  Reason                        Message
  ----            -
  AbleToScale     True    ScaleDownStabilized          recent recommendations were higher than current one, applying the high
est recent recommendation
  ScalingActive   True    ValidMetricFound              the HPA was able to successfully calculate a replica count from cpu re
source utilization (percentage of request)
  ScalingLimited  False   DesiredWithinRange            the desired count is within the acceptable range
Events:           <none>
```

En la anterior captura, podemos observar que está funcionando de manera correcta ya que se desea 1 pod y actualmente hay un único pod, vamos a realizar la comprobación al final de este ejercicio para ver la serie de eventos que han hecho incrementar o disminuir el número de pods.

Ahora vamos a incrementar la carga, vamos a crear un contenedor con un pod que mandará peticiones a nuestro servicio “php-apache”, lo realizamos con el siguiente comando:

```
# kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c
"while sleep 0.01; do wget -q -O- http://php-apache; done"
```

Este comando consulta al servicio apache que en lugar de devolver una página por defecto, se ha optado para que haga algo útil. Se ha creado un fichero “index.php” que realiza un cálculo computacional y tras realizarlo devuelve “OK!”:

```
<?php
    $x = 0.0001;

    for ($i = 0; $i <= 1000000; $i++) {

        $x += sqrt($x);

    }

    echo "OK!";

?>
```

Este fichero se ha insertado en la imagen Docker con el Dockerfile, por tanto al instanciar los contenedores, se guarda dicho fichero en el directorio `"/var/www/html"`:

```
FROM php:5-apache
COPY index.php /var/www/html/index.php
RUN chmod a+rx index.php
```

El resultado de ejecutar este comando al cabo de 15 minutos es el siguiente:

```
[root@my-master-1 ~]# kubectl run -i --tty load-generator --rm --image=busybox --restart=Never - /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

Desde otro terminal, para analizar el comportamiento de aumentar y disminuir la carga y el número de pods, puse a ejecutar el siguiente comando:

#kubectl get hpa -w (o --watch)

Este comando nos devuelve el estado del hpa a lo largo del tiempo.

El resultado ha sido el siguiente:

```
[root@my-master-1 hpapache]# kubectl get hpa -w
```

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	0%/50%	1	10	1	4m22s
php-apache	Deployment/php-apache	0%/50%	1	10	1	5m21s
php-apache	Deployment/php-apache	247%/50%	1	10	1	6m8s
php-apache	Deployment/php-apache	247%/50%	1	10	4	6m23s
php-apache	Deployment/php-apache	247%/50%	1	10	5	6m38s
php-apache	Deployment/php-apache	71%/50%	1	10	5	7m9s
php-apache	Deployment/php-apache	66%/50%	1	10	5	8m10s
php-apache	Deployment/php-apache	69%/50%	1	10	5	9m11s
php-apache	Deployment/php-apache	69%/50%	1	10	6	9m26s
php-apache	Deployment/php-apache	70%/50%	1	10	6	10m
php-apache	Deployment/php-apache	66%/50%	1	10	6	11m
php-apache	Deployment/php-apache	67%/50%	1	10	6	12m
php-apache	Deployment/php-apache	66%/50%	1	10	6	13m
php-apache	Deployment/php-apache	67%/50%	1	10	6	14m
php-apache	Deployment/php-apache	69%/50%	1	10	6	15m
php-apache	Deployment/php-apache	70%/50%	1	10	6	16m
php-apache	Deployment/php-apache	66%/50%	1	10	6	17m
php-apache	Deployment/php-apache	67%/50%	1	10	6	18m
php-apache	Deployment/php-apache	68%/50%	1	10	6	20m
php-apache	Deployment/php-apache	66%/50%	1	10	6	22m

Podemos observar el aumento de la carga y su ajuste con el escalado automático de pods en la captura anterior.

Ahora vamos a ver los pods que se están ejecutando en el nodo máster, y obtenemos el siguiente resultado:

```
[root@my-master-1 ~]# kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
load-generator	1/1	Running	0	19m
php-apache-d4cf67d68-42899	0/1	Pending	0	18m
php-apache-d4cf67d68-5c8z5	1/1	Running	0	18m
php-apache-d4cf67d68-h276b	0/1	Pending	0	15m
php-apache-d4cf67d68-kc8jr	1/1	Running	0	18m
php-apache-d4cf67d68-tklm	1/1	Running	0	18m
php-apache-d4cf67d68-wj2xz	1/1	Running	0	25m

Como se aprecia en la captura, se ha generado 6 pods para satisfacer las necesidades de las peticiones “php-apache”, pero existen dos pods que están en estado pendiente, esto es debido a mis limitaciones de memoria o cores de la máquina virtual en la que están corriendo dichos pods.

Si paramos de realizar las peticiones con Ctrl+C.

```
OK!OK!çOK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!OK!^C
E0210 12:37:26.826536 21415 v2.go:105] EOF
pod "load-generator" deleted
```

Podemos observar que al cabo de unos minutos se disminuye automáticamente el número de pods destruyendo los que no son necesarios, debido a la disminución de la carga. Se puede apreciar en la siguiente captura:

php-apache	Deployment/php-apache	67%/50%	1	10	6	28m
php-apache	Deployment/php-apache	0%/50%	1	10	6	29m
php-apache	Deployment/php-apache	0%/50%	1	10	6	33m
php-apache	Deployment/php-apache	0%/50%	1	10	1	34m

Ahora vamos a ver en detalle la descripción del hpa, lo podemos ver con el siguiente comando:

kubectl describe hpa php-apache

El resultado es el siguiente:

```
[root@my-master-1 ~]# kubectl describe hpa php-apache
Name: php-apache
Namespace: default
Labels: <none>
Annotations: <none>
CreationTimestamp: Wed, 10 Feb 2021 12:09:14 +0100
Reference: Deployment/php-apache
Metrics:
  resource cpu on pods (as a percentage of request): 0% (1m) / 50%
Min replicas: 1
Max replicas: 10
Deployment pods: 1 current / 1 desired
Conditions:
  Type           Status  Reason                        Message
  ----           -
  AbleToScale    True    ReadyForNewScale             recommended size matches current size
  ScalingActive  True    ValidMetricFound             the HPA was able to successfully calculate a replica count from cpu resource utilization (percentage of request)
  ScalingLimited True    TooFewReplicas               the desired replica count is less than the minimum replica count
Events:
  Type           Reason              Age             From              Message
  ----           -
  Normal         SuccessfulRescale   33m            horizontal-pod-autoscaler    New size: 4; reason: cpu resource utilization (percentage of request) above target
  Normal         SuccessfulRescale   32m            horizontal-pod-autoscaler    New size: 5; reason: cpu resource utilization (percentage of request) above target
  Normal         SuccessfulRescale   30m            horizontal-pod-autoscaler    New size: 6; reason: cpu resource utilization (percentage of request) above target
  Normal         SuccessfulRescale   5m23s          horizontal-pod-autoscaler    New size: 1; reason: All metrics below target
```

Podemos observar que hpa ha sido ejecutado en el nombre de espacio por defecto, las métricas que hemos definido son recursos cpu en cada pod. Actualmente el número de pods necesarios son 1 y únicamente se encuentra un único pod disponible.

Al final de la captura anterior, en el apartado de eventos, podemos analizar los sucesos que han ido ocurriendo en nuestro hpa, primero al aumentar la carga brutalmente a base de peticiones, hpa ha considerado aumentar a 4, el número de réplicas de pods, posteriormente dicha cantidad se ha incrementado a 5 y luego a 6. Finalmente, al cabo de unos 5 minutos, al no haber carga, hpa ha destruido los pods quedando actualmente un único pod que va a recibir la carga.

Finalmente si consultamos el número de pods disponibles, vemos que tenemos la siguiente cantidad de pods:

```
[root@my-master-1 ~]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
php-apache-d4cf67d68-wj2xz         1/1     Running   0          46m
[root@my-master-1 ~]#
```

Y si vemos de cuántos pods consta el deployment, también veremos que consta de un único pod:

```
[root@my-master-1 ~]# kubectl get deployment php-apache
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
php-apache    1/1     1            1           47m
```


En el anterior ejemplo, hemos lanzado el hpa a través de un comando, pero también es posible ejecutarlo a través de un fichero “.yaml”. Dicho fichero debe tener la siguiente estructura:

```
[root@my-master-1 hp]# cat nginx-hpa.yaml
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: nginx
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

También es posible realizar el escalado horizontal de pods en función de varias métricas, para ello definimos el fichero yaml con varios recursos como se aprecia en la siguiente figura:

```
[root@my-master-1 hp]# cat nginx-multiple-hpa.yaml
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
  - type: Resource
    resource:
      name: memory
      target:
        type: AverageValue
        averageValue: 100Mi
```

En la captura anterior, hemos optado por servidor web nginx que constará entre 1(mínimo) y 10(máximo) réplicas de pods que van a depender de recursos utilizados tanto del CPU como de la memoria.

Ahora vamos a crear 3 pods y vamos a analizar los recursos consumidos:


```
[root@my-master-1 hp]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-57cf88d87f-bkhv5             1/1     Running   0           3m4s
nginx-57cf88d87f-btsgx             1/1     Running   0           3m4s
nginx-57cf88d87f-sk5w              1/1     Running   0           3m4s
[root@my-master-1 hp]# kubectl top pod
NAME                                CPU(cores)   MEMORY(bytes)
nginx-57cf88d87f-bkhv5             0m           1Mi
nginx-57cf88d87f-btsgx             0m           1Mi
nginx-57cf88d87f-sk5w              0m           1Mi
[root@my-master-1 hp]# kubectl get hpa
NAME      REFERENCE          TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
nginx     Deployment/nginx    1387178666m/100Mi, 0%/50%   1         10        3           2m58s
[root@my-master-1 hp]#
```

En la captura anterior, podemos observar que se han generado 3 pods, que actualmente la carga se encuentra a 0% en todos ellos, ya que no hemos realizado ninguna petición al servidor. Y también podemos observar la cantidad de CPU en uso (medido en cores) y la cantidad de memoria consumida en bytes.

Al cabo de unos minutos, al no haber carga, hpa reduce automáticamente el número de réplicas de pods a 1, como se aprecia en la descripción de hpa:

```
Normal    SuccessfulRescale    105s    horizontal-pod-autoscaler    New size: 1; reason: All metrics below target
```

Si analizamos de nuevo los anteriores comandos, vemos que se han producido dichos cambios:

```
[root@my-master-1 hp]# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-57cf88d87f-btsgx             1/1     Running   0           10m
[root@my-master-1 hp]# kubectl top pod
NAME                                CPU(cores)   MEMORY(bytes)
nginx-57cf88d87f-btsgx             0m           1Mi
[root@my-master-1 hp]# kubectl get hpa
NAME      REFERENCE          TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
nginx     Deployment/nginx    1376256/100Mi, 0%/50%   1         10        1           10m
```

Dashboard

Ahora vamos a configurar el dashboard para analizar el uso y las métricas de nuestro clúster. En versiones anteriores, se utilizaba Heapster, pero en las nuevas versiones de Kubernetes, se utiliza metrics-server, lo mismo que utilizábamos para medir las métricas cuando realizábamos el escalado horizontal de pods en nuestro clúster.

Existen actualmente dos formas de realizar el despliegue de dashboards, una versión menos segura y que nos ofrece menos posibilidades pero podemos observar y analizar métricas de nuestro clúster con “http” sin utilizar ningún token. Y la otra forma más segura de analizar y observar las métricas generadas utilizando el token y visualizando los gráficos.

Dashboard sin utilizar token

Primero vamos a ver la forma de realizarlo sin token, esta forma está bastante restringida en cuanto a los ajustes y no nos permite ver algunos tipos de objetos, por ejemplo, no permite ver la página overview que nos proporciona información general del clúster.

Primero de todo, desplegamos el Dashboard UI por defecto con el siguiente comando:

#kubectl apply -f

<https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/recommended.yaml>

El resultado es el siguiente:

```
[root@my-master-1 hp]# kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/recommended.yaml
namespace/kubernetes-dashboard created
serviceaccount/kubernetes-dashboard created
service/kubernetes-dashboard created
secret/kubernetes-dashboard-certs created
secret/kubernetes-dashboard-csrf created
secret/kubernetes-dashboard-key-holder created
configmap/kubernetes-dashboard-settings created
role.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrole.rbac.authorization.k8s.io/kubernetes-dashboard created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
clusterrolebinding.rbac.authorization.k8s.io/kubernetes-dashboard created
deployment.apps/kubernetes-dashboard created
service/dashboard-metrics-scraper created
deployment.apps/dashboard-metrics-scraper created
```

Observamos que se han generado los artefactos necesarios para visualizar el dashboard.

Vamos a crear un pod bajo el nombre de espacio “kube-system” que va a recoger nuestras métricas. Vamos a desplegarlo con el siguiente comando:

#kubectl create -f

<https://raw.githubusercontent.com/kubernetes/dashboard/v1.10.1/src/deploy/alternative/kubernetes-dashboard.yaml>

Ahora vamos a comprobar que se ha generado el pod “Kubernetes-dashboard” y vemos que está listo:

```
[root@my-master-1 heapster]# kubectl get pods --namespace kube-system
NAME                                READY    STATUS    RESTARTS   AGE
kubernetes-dashboard-5d977bcb48-6lclf 1/1      Running   0           168m
```

Ahora vamos a exponer nuestro pod a través del comando “kubectl proxy” para que sea accedido desde el exterior.

Ahora desde un terminal, podemos ejecutar el siguiente comando para iniciar el dashboard:

```
# kubectl proxy --address 0.0.0.0 --accept-hosts '.*'
```

El usuario que viene por defecto no tiene privilegios administrativos, por tanto debemos dar de alta dicho usuario en nuestro clúster. Vamos a asignar el rol de cluster-admin.

Con el siguiente commando, vamos a generar un artefacto de tipo “clusterrolebinding” que sirve para gestionar los permisos de un grupo de usuarios a unos recursos determinados, creamos por tanto el siguiente rol:

```
#kubectl create clusterrolebinding kubernetes-dashboard --clusterrole=cluster-admin \
--serviceaccount=kube-system:kubernetes-dashboard
```

Ahora vamos a nuestro navegador en el host y nos conectamos al dashboard a través del siguiente comando:

<http://192.168.100.8:8001/api/v1/namespaces/kube-system/services/kubernetes-dashboard/proxy/#!/cluster?namespace=default>

Donde la IP inicial es la IP de la máquina del máster, puerto 8001.

A continuación se muestran algunas capturas del funcionamiento del dashboard y la captura del terminal que confirma los datos ya que se realiza la misma consulta.

Name	Labels	Cluster IP	Internal endpoints	External endpoints	Age
php-apache	run: php-apache	10.233.12.62	php-apache:80 TCP	-	18 hours
kubernetes	component: apiserver provider: kubernetes	10.233.0.1	kubernetes:443 TCP	-	7 days

```
root@my-master-1:~# kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.233.0.1    <none>         443/TCP     7d18h
php-apache   ClusterIP   10.233.12.62 <none>         80/TCP     18h
```

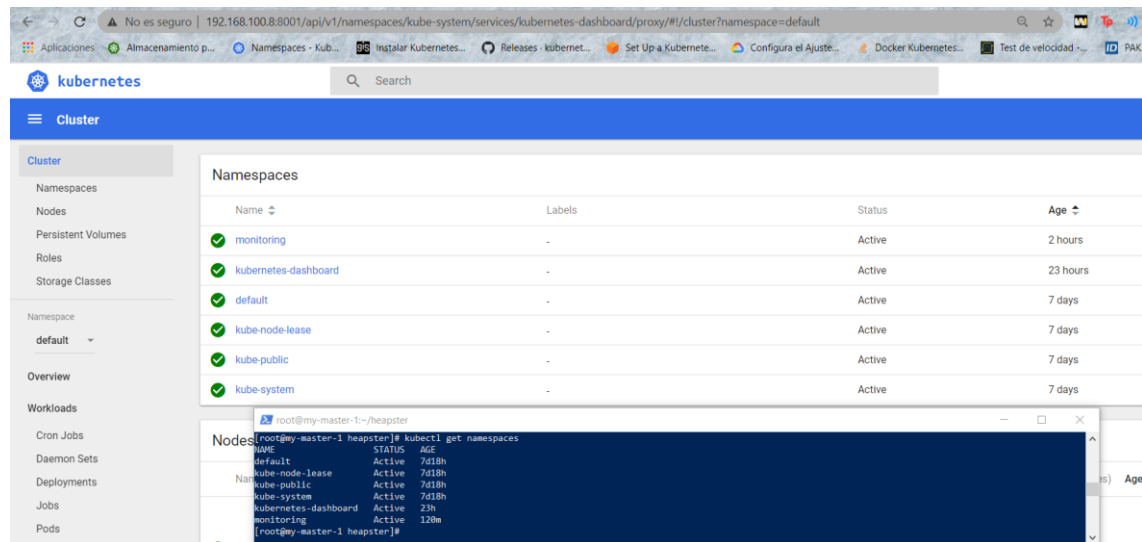
Ahora vamos a consultar los pods:

Name	Node	Status	Restarts	Age
nginx-57cf88d87f-4pbz4	my-node-2	Running	0	38 minutes
nginx-57cf88d87f-86kb7		Pending	0	38 minutes
nginx-57cf88d87f-fcb7c	my-node-1	Running	0	38 minutes
prometheus-prometheus-0	my-node-2	Running	1	an hour
prometheus-prometheus-1	my-node-2	Running	1	an hour

```
root@my-master-1:~# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-57cf88d87f-4pbz4              1/1     Running   0           38m
nginx-57cf88d87f-86kb7              0/1     Pending   0           38m
nginx-57cf88d87f-fcb7c              1/1     Running   0           38m
prometheus-prometheus-0             2/2     Running   1           76m
prometheus-prometheus-1             2/2     Running   1           79m
```

En la captura anterior, aparece un aviso en naranja que nos indica que debido a la insuficiencia de cpu, no se han podido crear pods, cosa que afirma la captura de la terminal, ya que aparecen pods con estado 0/0, es decir, no están corriendo.

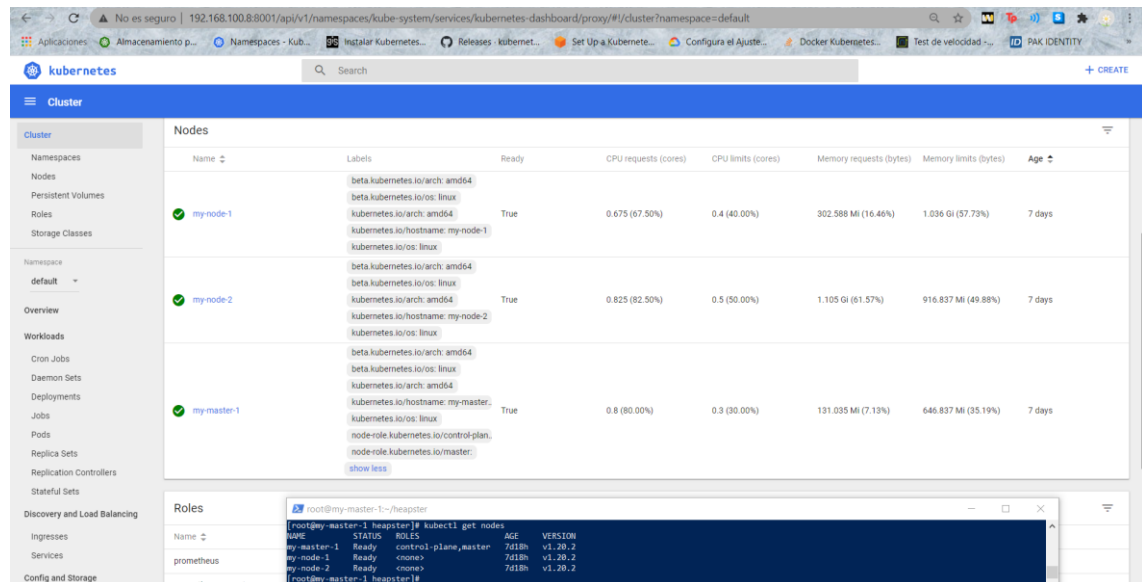
Ahora vamos a ver los nombres de espacios que tenemos actualmente en el clúster:



Name	Labels	Status	Age
monitoring	-	Active	2 hours
kubernet-dashboard	-	Active	23 hours
default	-	Active	7 days
kube-node-lease	-	Active	7 days
kube-public	-	Active	7 days
kube-system	-	Active	7 days

```
root@my-master-1:~# kubectl get namespaces
NAME                STATUS    AGE
default             Active    7d18h
kube-node-lease     Active    7d18h
kube-public         Active    7d18h
kubernet-dashboard  Active    23h
monitoring          Active    128m
```

Ahora vamos a consultar los nodos que están corriendo sobre nuestro clúster y sus correspondientes métricas tanto de peticiones CPU, límite de CPU, peticiones de memoria en bytes y el límite de memoria en bytes. También podemos observar los días que llevan iniciados:



Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
my-node-1	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64 kubernetes.io/hostname: my-node-1 kubernetes.io/os: linux	True	0.675 (87.50%)	0.4 (40.00%)	302.588 Mi (16.46%)	1.036 Gi (57.73%)	7 days
my-node-2	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64 kubernetes.io/hostname: my-node-2 kubernetes.io/os: linux	True	0.825 (82.50%)	0.5 (50.00%)	1.105 Gi (61.57%)	916.837 Mi (49.88%)	7 days
my-master-1	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/arch: amd64 kubernetes.io/hostname: my-master-1 node-role.kubernetes.io/control-plane: node-role.kubernetes.io/master:	True	0.8 (80.00%)	0.3 (30.00%)	131.035 Mi (7.13%)	646.837 Mi (35.19%)	7 days

```
root@my-master-1:~# kubectl get nodes
NAME                STATUS    ROLES    AGE    VERSION
my-master-1        Ready    control-plane,master    7d18h    v1.20.2
my-node-1          Ready    <none>    7d18h    v1.20.2
my-node-2          Ready    <none>    7d18h    v1.20.2
```

En la anterior captura, podemos observar un uso de CPU al 40% del nodo1, del 50% del nodo2 y del 30% del máster.

Dashboard con token de acceso

Ahora vamos a ver cómo realizar el mismo proceso pero generando un token de acceso y utilizando para acceder a las métricas de nuestro clúster.

Estando en local(Windows), primero generamos una sesión ssh hacia el nodo máster desde el host con el siguiente comando:

```
#ssh -L localhost:8001:127.0.0.1:8001 <user>@<master_public_IP>
```

Donde en <user> ponemos root y <master_public_IP> ponemos la IP que es 192.168.100.8.

Tal y como se ve en la siguiente captura:

```
PS C:\Users\Farooq> ssh -L localhost:8001:127.0.0.1:8001 root@192.168.100.8
root@192.168.100.8's password:
Last login: Thu Feb 11 14:01:58 2021 from 192.168.100.3
[root@my-master-1 ~]# kubectl get pods -A
```

Posteriormente, desplegamos los pods para medir las métricas del dashboard con el siguiente comando:

```
#kubectl apply -f
```

<https://raw.githubusercontent.com/kubernetes/dashboard/v2.0.0/aio/deploy/recommended.yaml>

Ahora vemos si se han generado los pods y en qué espacio de nombre se han generado con el siguiente comando:

```
#kubectl get pods -A
```

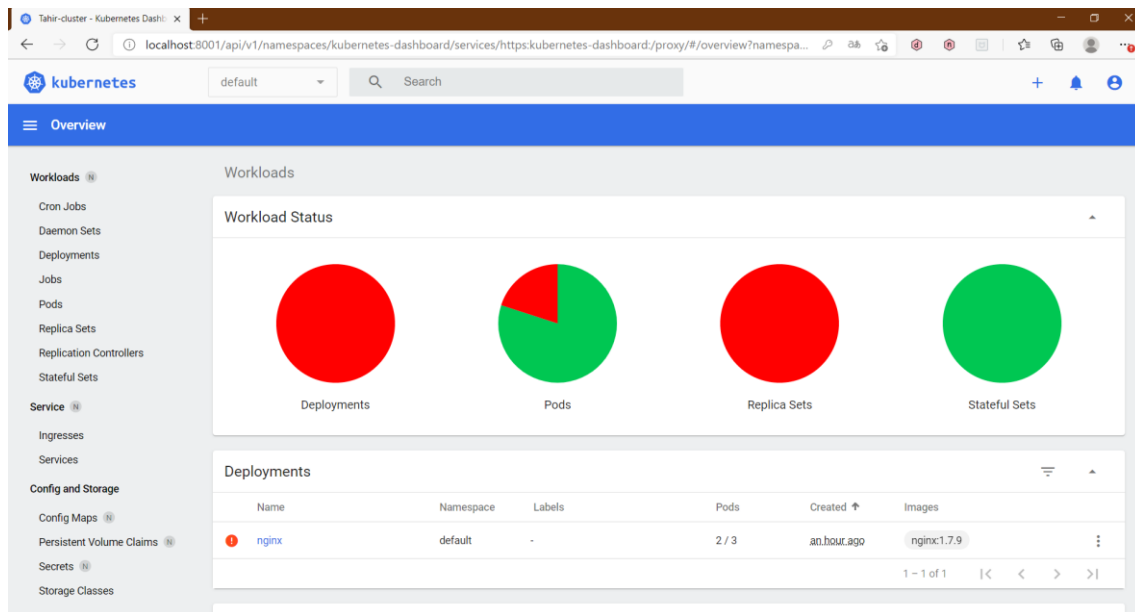
kubernetes-dashboard	dashboard-metrics-scraper-79c5968bdc-nbvzk	1/1	Running	0	122m
kubernetes-dashboard	kubernetes-dashboard-7448ffc97b-hrs2z	1/1	Running	0	122m

Vemos que hay un total de dos pods corriendo en el espacio de nombre “Kubernetes-dashboard”.

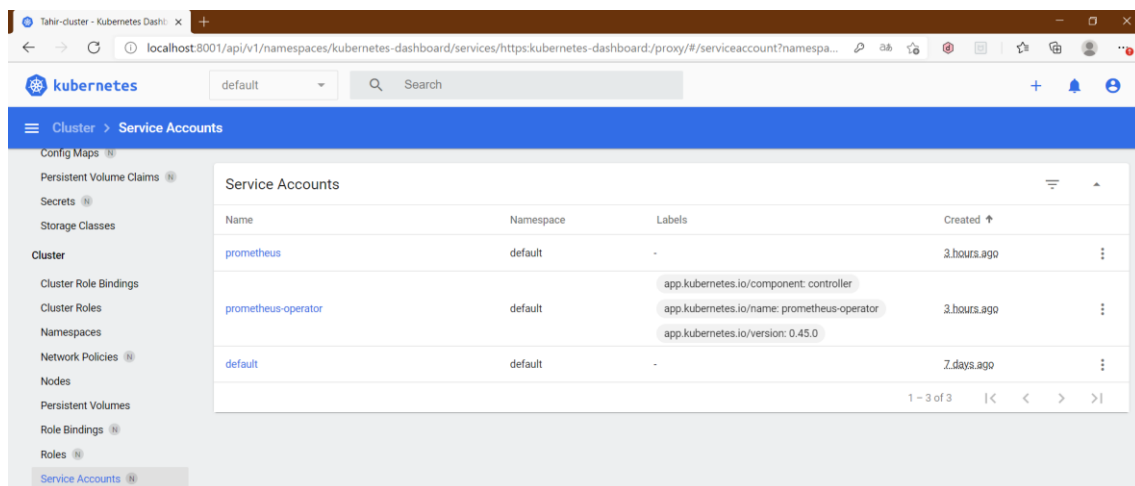
Ahora vamos a crear el usuario con el rol de cluster-admin para que tenga acceso a toda la información del clúster.

Antes lo hemos realizado a través de línea de comando, ahora vamos a realizar con el siguiente fichero yaml:

```
[root@my-master-1 dashboard]# cat archivo.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

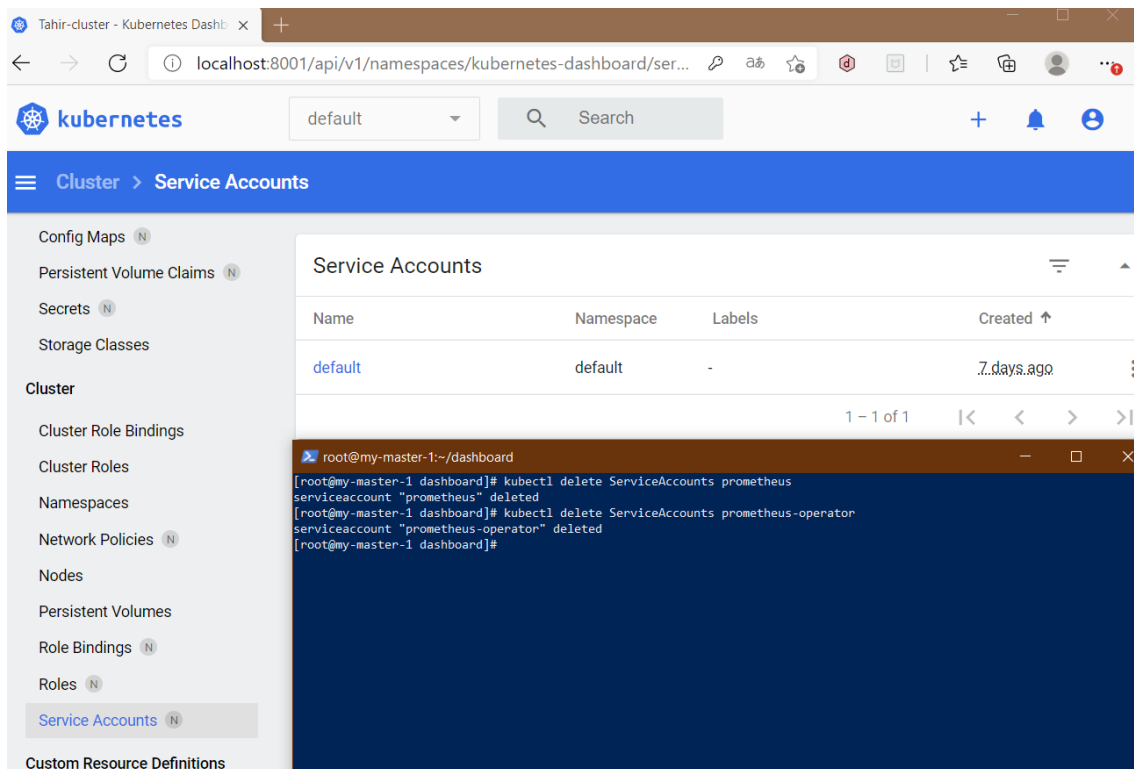



En la captura, anterior, podemos analizar que fallan los pods de nginx que fueron generados con el objeto replicaset(que está en rojo, por tanto falla) y el deployment asignado(que también falla). A continuación, veremos como solucionarlo.

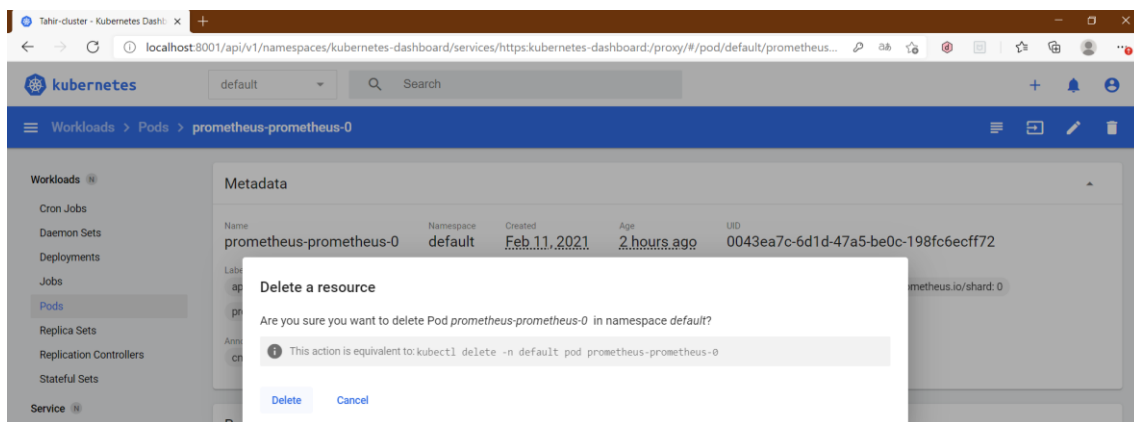


En la captura superior podemos observar las cuentas de servicios actualmente corriendo en nuestro clúster, podemos observar que tenemos prometheus, prometheus-operator y "default".

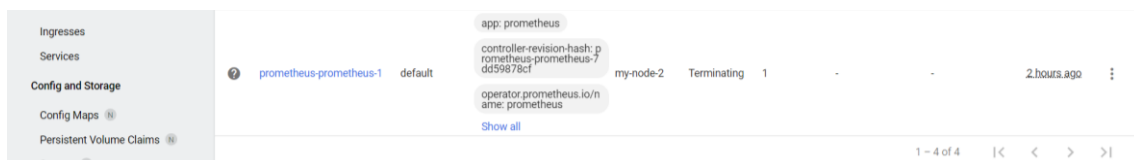
Ahora vamos a eliminar las dos cuentas de servicio con nombre prometheus y prometheus-operator a través de comandos y podemos observar que se han borrado también de la interfaz gráfica. Como se puede observar en la siguiente captura:



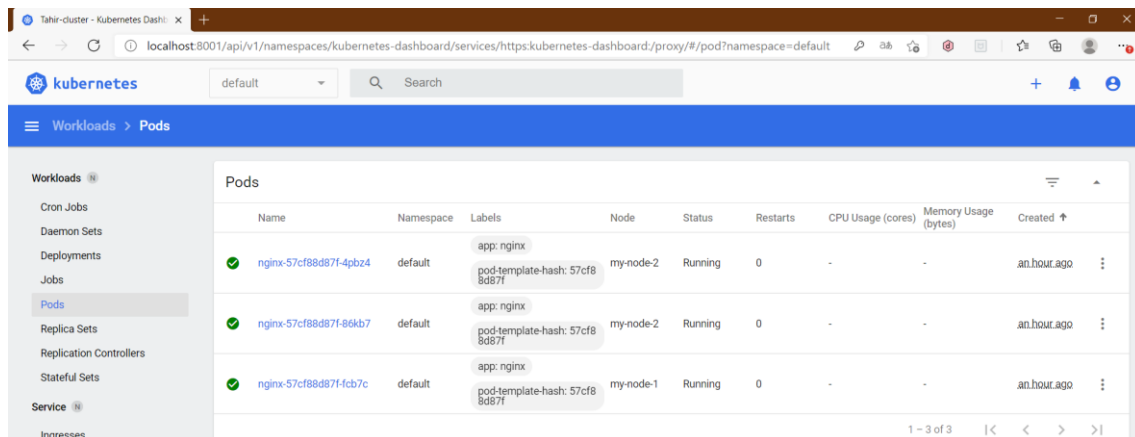
Pero, también podemos eliminar un recurso desde la interfaz pulsando sobre el emoticono que se encuentra a la derecha sobre la barra de menú (papelera). Como se aprecia en la siguiente captura:



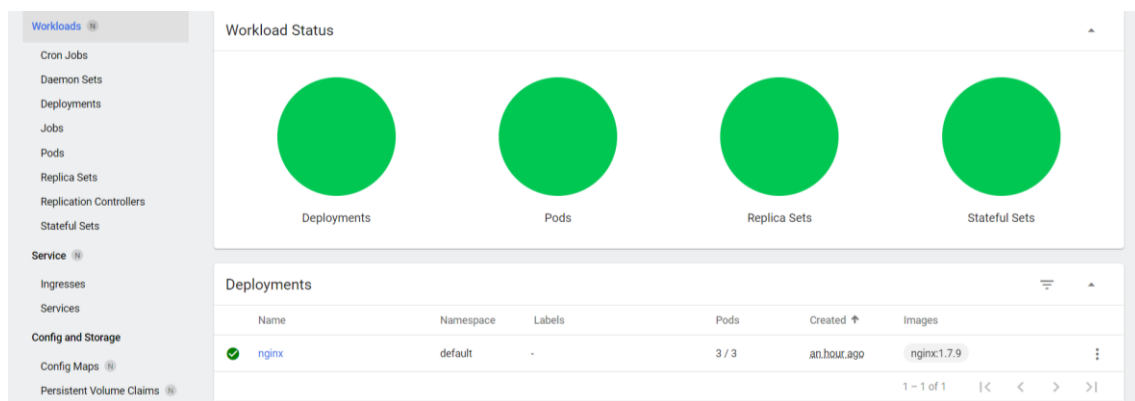
Vemos que al eliminar dicho pod, podemos observar la transición de su eliminación, de Running->Terminating para finalmente ser eliminado. Al eliminar estos pods, se liberan recursos que ocupaban.



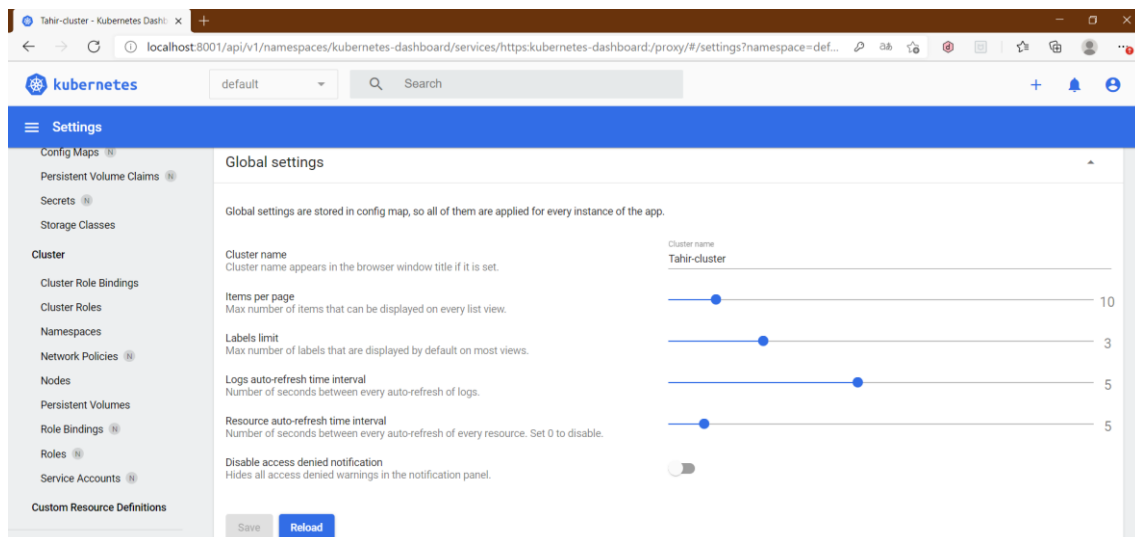
Tras eliminar los pods de prometheus, podemos observar que ya hay recursos suficientes para desplegar los pods de nginx, que se levantan de manera automática. Ahora podemos observar que tenemos los 3 pods corriendo en estado saludable como se ve en la siguiente captura:



Ahora los gráficos iniciales nos salen en verde:



Podemos ir a ajustes y cambiar el nombres del clúster, yo he querido llamar “Tahir-clúster” y podemos realizar modificaciones tales como intervalo de refresco, idioma, frecuencia de actualizaciones de logs:



Si queremos acceder a la información de un pod, podemos clicar sobre él y nos abre la interfaz completa tal y como se ve en la siguiente captura:

Workloads > Pods > nginx-57cf88d87f-4pbz4

Metadata

Name	Namespace	Created	Age	UID
nginx-57cf88d87f-4pbz4	default	Feb 11, 2021	an hour ago	81f9c11b-b276-4ea1-b43f-1dfb9d4910dd

Labels

- app: nginx
- pod-template-hash: 57cf88d87f

Annotations

- cni.projectcalico.org/podIP: 10.233.67.84/32
- cni.projectcalico.org/podIPs: 10.233.67.84/32

Resource information

Node	Status	IP	QoS Class	Restarts
my-node-2	Running	10.233.67.84	Burstable	0

Podemos cambiar ver los recursos por espacio de nombre:

Tahir-cluster - Kubernetes Dashb

localhost:8001/api/v1/namespaces/ku

kubernetes default

Si seleccionamos el espacio de nombre “kube-system” podemos ver los recursos con los que funciona Kubernetes.

Podemos filtrar los pods por nombre y por nombre de espacio, como se observa en la siguiente captura:

Workloads > Pods

Pods


Name	Labels	Node	Status	Restarts	CPU Usage (cores)	Memory Usage (bytes)	Created
calico-kube-controllers-847f479c5-qw3qj	k8s-app: calico-kube-controllers pod-template-hash: 847f479c5	my-node-1	Running	14	-	-	7.08s ago
calico-node-5bzt5	controller-revision-hash: 6d0c75c67c k8s-app: calico-node	my-node-1	Running	7	-	-	7.08s ago
calico-node-b1vhd	controller-revision-hash: 6d0c75c67c k8s-app: calico-node	my-master-1	Running	6	-	-	7.08s ago
calico-node-grjpt	controller-revision-hash: 6d0c75c67c k8s-app: calico-node	my-node-2	Running	6	-	-	7.08s ago

Podemos ver los demonios instalados:

Daemon Sets

Name	Labels	Pods	Created	Images
node-local-dns	addonmanager.kubernetes.io/mode: Reconcile k8s-app: kube-dns	3 / 3	7.08s ago	k8s.gcr.io/dns/k8s-dns-node-cache:1.16.0
calico-node	k8s-app: calico-node	3 / 3	7.08s ago	quay.io/calico/node:v3.16.6
kube-proxy	k8s-app: kube-proxy	3 / 3	7.08s ago	k8s.gcr.io/kube-proxy:v1.20.2

También podemos visualizar los servicios desplegados:

Services					
Name	Labels	Cluster IP	Internal Endpoints	External Endpoints	Created ↑
 kubelet-dashboard	k8s-app: kubelet-dashboard	10.233.23.72	kubelet-dashboard kube-system:80 TCP kubelet-dashboard kube-system:80 TCP	-	4.765018.890 Feb 11, 2021, 10:48:08 AM
1 - 1 of 1 < >					
Events					
There is nothing to display here No resources found.					

También podemos observar los pods cuando estamos realizando hpa:

Pods status		
Running	Pending	Desired
2	3	5

Si seguimos mirando, podemos observar que aumenta hasta 4 pods y ya no escala más debido a las limitaciones de mi máquina virtual:

Pods status			
Updated	Total	Available	Unavailable
6	6	4	2

Workload Status

Deployments

Pods

Replica Sets

Stateful Sets

Deployments

Name	Namespace	Labels	Pods	Created ↑	Images
<div><div></div>php-apache</div>	default	-	4 / 6	8 minutes ago	k8s.gcr.io/hpa-example

1 - 1 of 1

Conclusiones

Este trabajo me ha ayudado a entender el funcionamiento de los clústeres en local, siendo de gran interés la experiencia obtenida, resultado de trabajar con el clúster Kubernetes. Este trabajo me ha motivado para estudiar y entender la gestión de aplicaciones en la nube. Me ha gustado bastante trabajar de primera mano el escalado horizontal que es tremendamente útil cuando una aplicación o un sistema es sometido a carga.

Para un futuro trabajo, se podría ampliar instalando almacenamiento persistente de volúmenes dinámico en Kubernetes. También se opta por desplegar el clúster Kubernetes en la nube, utilizando objetos de tipo Ingress(que representan un mayor nivel de abstracción que un objeto de tipo servicio).

Para realizar el despliegue en la nube, se puede utilizar la plataforma de Google cloud y utilizar la aplicación implementada en la asignatura de SAD para realizar el despliegue de Kubernetes.

Anexos

En este apartado, voy a comentar errores hallados durante el despliegue de este trabajo.

El error hallado al lanzar el comando “ansible-playbook” ha sido la no generación de token, a continuación se detalla la captura del clúster junto:

```
root@ansible:~/kubespray
out", "-in", "/etc/kubernetes/ssl/apiserver.crt", "-checkip", "192.168.100.11", "delta": "0:00:00.022978", "end": "2021-02-02 20:27:18.001044", "item": "192.168.100.11", "rc": 0, "start": "2021-02-02 20:27:17.978066", "stderr": "", "stderr_lines": [], "stdout": "IP 192.168.100.11 does match certificate", "stdout_lines": ["IP 192.168.100.11 does match certificate"]}
ok: [my-master-1] => (item=192.168.100.11) => (ansible_loop_var=192.168.100.11, changed=false, cmd=[openssl, -x509, -noout, "-in", "/etc/kubernetes/ssl/apiserver.crt", "-checkhost", "my-master-1.cluster.local"], "delta": "0:00:00.019455", "end": "2021-02-02 20:27:23.935752", "item": "my-master-1.cluster.local", "rc": 0, "start": "2021-02-02 20:27:23.916307", "stderr": "", "stderr_lines": [], "stdout": "Hostname my-master-1.cluster.local does match certificate", "stdout_lines": ["Hostname my-master-1.cluster.local does match certificate"])
Tuesday 02 February 2021 20:27:24 +0100 (0:01:21.291) 0:30:43.992 *****
Tuesday 02 February 2021 20:27:24 +0100 (0:00:00.112) 0:30:44.105 *****
Tuesday 02 February 2021 20:27:24 +0100 (0:00:00.073) 0:30:44.178 *****
Tuesday 02 February 2021 20:27:24 +0100 (0:00:00.068) 0:30:44.247 *****
Tuesday 02 February 2021 20:27:24 +0100 (0:00:00.072) 0:30:44.319 *****
Tuesday 02 February 2021 20:27:24 +0100 (0:00:00.090) 0:30:44.410 *****
FAILED - RETRYING: Create kubeadm token for joining nodes with 24h expiration (default) (5 retries left).
FAILED - RETRYING: Create kubeadm token for joining nodes with 24h expiration (default) (4 retries left).
FAILED - RETRYING: Create kubeadm token for joining nodes with 24h expiration (default) (3 retries left).
FAILED - RETRYING: Create kubeadm token for joining nodes with 24h expiration (default) (2 retries left).
FAILED - RETRYING: Create kubeadm token for joining nodes with 24h expiration (default) (1 retries left).
```

Tras sucesivos intentos fallidos, se resetea el comando “ansible-playbook” con los siguientes comandos:

Solo vamos a lanzar reset.yml:

```
ansible-playbook --flush-cache -i inventory/mycluster/hosts.yml reset.yml --become -u root
```

y cluster.yml:

```
ansible-playbook --flush-cache -i inventory/mycluster/hosts.yml cluster.yml --become -u root
```

Otro error hallado al instalar las métricas de la versión v0.4.1 para visualizar el autoescalado horizontal del clúster, era que fallaban los propios pods de las métricas, como se observa en la siguiente captura(al final de la captura):

```
[root@my-master-1 hp]# kubectl get pods --namespace=kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
calico-kube-controllers-847f479bc5-gw5wj	1/1	Running	6	5d23h
calico-node-5q6z5	1/1	Running	3	6d
calico-node-b9xdn	1/1	Running	3	6d
calico-node-gnjpt	1/1	Running	3	6d
coredns-847f564ccf-5hgw7	1/1	Running	3	5d23h
coredns-847f564ccf-8fvbj	1/1	Running	3	5d23h
dns-autoscaler-b5c786945-fpqjs	1/1	Running	3	5d23h
kube-apiserver-my-master-1	1/1	Running	3	6d
kube-controller-manager-my-master-1	1/1	Running	7	6d
kube-proxy-b4ntj	1/1	Running	3	6d
kube-proxy-m8klr	1/1	Running	3	6d
kube-proxy-vmwvj	1/1	Running	3	6d
kube-scheduler-my-master-1	1/1	Running	6	6d
metrics-server-5d5c49f488-q7ht6	0/1	CrashLoopBackOff	6	6m59s
metrics-server-844d9574cf-d52px	0/1	CrashLoopBackOff	7	6m38s

Para solucionar esto, primero eliminamos los contenedores del deployment “metrics-server” con el siguiente comando:

```
#kubectl delete -n kube-system deployments.apps metrics-server
```

Posteriormente, instalamos la versión 0.3.7 de métricas con el siguiente comando:

Tras descargar este fichero:

<https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.3.7/components.yaml>

Realizamos los pertinentes cambios, ya explicados en el desarrollo de la memoria y aplicamos con el comando:

#kubectl apply -f components.yaml

El resultado es el correcto funcionamiento de los pods de “metrics-server”.

```
[root@my-master-1 metrics-server]# kubectl get pods --namespace=kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
calico-kube-controllers-847f479bc5-gw5wj	1/1	Running	6	6d
calico-node-5q6z5	1/1	Running	3	6d
calico-node-b9xdn	1/1	Running	3	6d
calico-node-gnjpt	1/1	Running	3	6d
coredns-847f564ccf-5hgw	1/1	Running	3	6d
coredns-847f564ccf-8fvbj	1/1	Running	3	6d
dns-autoscaler-b5c786945-fpqjs	1/1	Running	3	6d
kube-apiserver-my-master-1	1/1	Running	3	6d
kube-controller-manager-my-master-1	1/1	Running	7	6d
kube-proxy-b4ntj	1/1	Running	3	6d
kube-proxy-m8klr	1/1	Running	3	6d
kube-proxy-vmwvj	1/1	Running	3	6d
kube-scheduler-my-master-1	1/1	Running	6	6d
metrics-server-5f4b6b9889-b7xfs	1/1	Running	0	16s

Se puede consultar acerca del error en esta bibliografía⁴.

Comandos útiles:

kubectl get pods --all-namespaces → Para visualizar todos los pods ejecutados en todos los nombres de espacios.

#kubectl top pod → Para visualizar los recursos consumidos a nivel de pod. Pero es necesario tener instalado “metrics-server”.

#kubectl top nodes → Para visualizar los recursos consumidos a nivel de nodo.

#kubectl delete ClusterRoleBinding <nombre> → Este comando elimina el recurso de tipo ClusterRoleBinding con el nombre indicado. Un rol, clusterrol, rolbinding, clusterrolbinding son recursos que asignan un rol a un usuario o conjunto de usuarios. Asignando permisos bajo un espacio de nombre determinado. Esto se ha tenido que realizar para dar acceso al dashboard.

#kubectl delete -n [espacio-de-nombre] [tipo de objeto] [nombre objeto] → Este comando se utiliza para eliminar los objetos (de tipo Deployments, service, pod, clusterrolebinding, ...) en un determinado espacio de nombre.

⁴ <https://stackoverflow.com/questions/54106725/docker-kubernetes-mac-autoscaler-unable-to-find-metrics>

Bibliografía

Para realizar esta actividad, he consultado las siguientes fuentes bibliográficas:

<https://computingforgeeks.com/deploy-production-kubernetes-cluster-with-ansible/>

Kubernetes Cookbook Second Edition Practical solutions to container orchestration Hideto Saito, 2018

<https://ualmtorres.github.io/SeminarioKubernetes/#truepods>

<https://aws.amazon.com/es/premiumsupport/knowledge-center/eks-metrics-server-pod-autoscaler/>

<https://stackoverflow.com/questions/53725248/how-to-enable-kubeapi-server-for-hpa-autoscaling-metrics/53727101#53727101>

<https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/>

<https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/>

<https://upcloud.com/community/tutorials/deploy-kubernetes-dashboard/>

<https://onthedock.github.io/post/170716-mi-primera-app-en-kubernetes/>