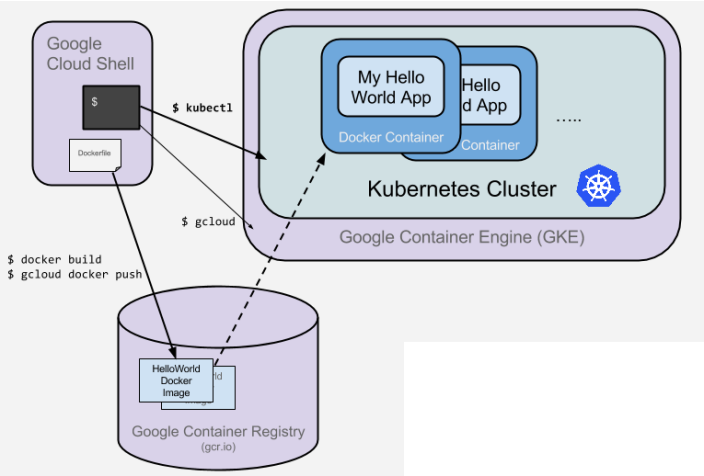
Lab Introducción a Kubernetes

Descripción general

El objetivo de este lab práctico es que hacer que un código desarrollado en un contenedor se ejecute en Kubernetes y concretamente en un clúster gestionado por Kubernetes Engine de Google Cloud. Para este lab, el código será una simple aplicación node.js que devolverá la cadena “Hello World” a peticiones que vengan de un navegador.

A continuación, tenemos un diagrama de las diversas piezas consideradas en este lab, lo usaremos como referencia a medida que avanzamos en el desarrollo del mismo.



Kubernetes es un proyecto de código abierto (disponible en kubernetes.io) que se puede ejecutar en distintos entornos, desde laptops hasta clústeres de múltiples nodos de alta disponibilidad, desde nubes públicas hasta implementaciones locales, desde máquinas virtuales hasta equipos físicos.

Para cumplir con el objetivo de este lab, utilizar un entorno administrado como Kubernetes Engine (una versión de Kubernetes alojada en Google que se ejecuta en Compute Engine) nos permite enfocarnos en la experimentación de Kubernetes que en la configuración de la infraestructura subyacente.

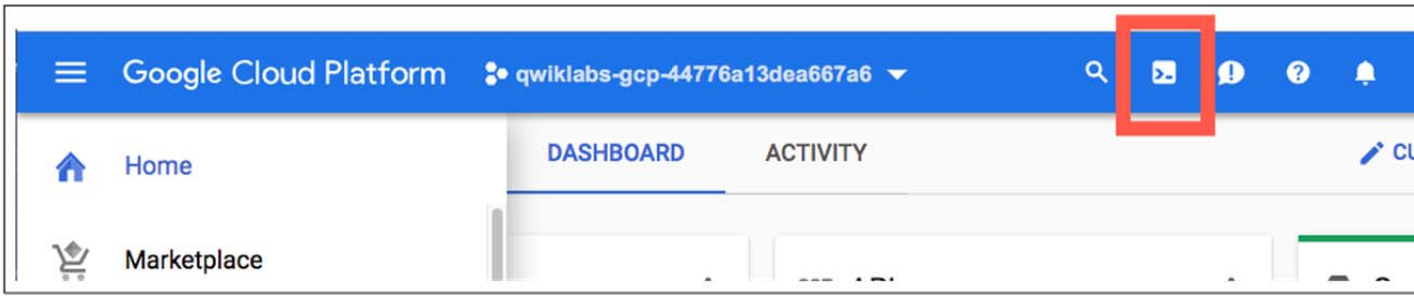
Actividades

- Crear un servidor Node.js
- Crear una imagen de contenedor de Docker
- Crear un clúster de contenedor
- Crear un pod de Kubernetes
- Escalar verticalmente sus servicios

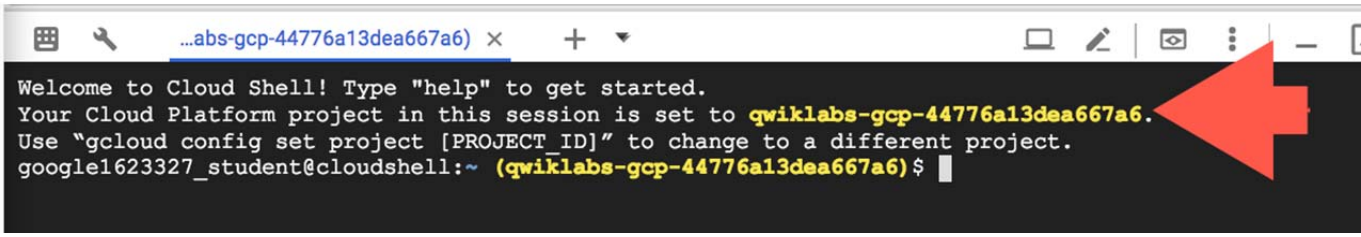
Activar Google Cloud Shell

Google Cloud Shell es una máquina virtual que cuenta con herramientas de desarrollo. Ofrece un directorio principal persistente de 5 GB y se ejecuta en Google Cloud. Google Cloud Shell proporciona acceso de línea de comandos a los recursos de GCP.

1. En GCP Console, en la barra de herramientas superior derecha, haga clic en el botón **Abrir Cloud Shell**.



Toma unos minutos aprovisionar los recursos y conectarse con el entorno. Cuando está conectado, ya está autenticado y el proyecto está configurado en su **PROJECT_ID** . Por ejemplo:



gcloud es la herramienta de línea de comandos para Google Cloud Platform. Viene preinstalada en Cloud Shell y es compatible con la función "tab-completion".

Podemos ver el nombre de nuestra cuenta activa con el comando:

```
$gcloud auth list
```

El ID de nuestro proyecto activo en Google cloud se puede obtener con este comando:

```
$gcloud config list project
```

Crear una aplicación Node.js

Con Cloud Shell, codificaremos un servidor Node.js simple para implementarlo en Kubernetes Engine:

```
$nano server.js
```

Añadimos el siguiente código:

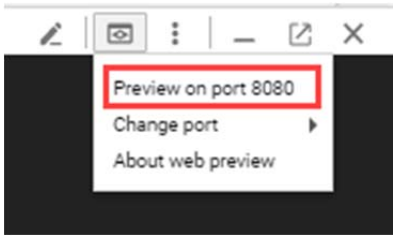
```
var http = require('http');
var handleRequest = function(request, response) {
  response.writeHead(200);
  response.end("Hello World!");
}
var www = http.createServer(handleRequest);
www.listen(8080);
```

Guardamos el archivo `server.js` (Ctrl O-Enter-Ctrl-X)

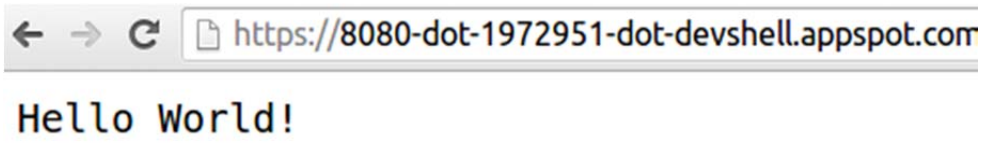
Ya que Cloud Shell tiene instalado el entorno node, ejecute este comando para iniciar el servidor web (el comando no muestra ningún resultado):

```
$node server.js
```

Utilice la función de vista previa web incorporada de Cloud Shell para abrir una nueva pestaña del navegador y enviar una solicitud a través de un proxy a la instancia que acaba de iniciar en el puerto 8080. Si no encuentra esta función en la barra de herramientas de Cloud Shell, haga clic en el **menú de navegación** para cerrar el panel izquierdo.



Se abrirá una pestaña nueva en el navegador en la que se mostrarán sus resultados:



Antes de continuar, vuelva a Cloud Shell y presione Ctrl+C para detener la ejecución del servidor de node.

Ahora deberá empaquetar esta aplicación en un contenedor de Docker.

Crear una imagen de contenedor de Docker

Ahora, generaremos un `Dockerfile` que describa la imagen que queremos crear. Las imágenes de contenedor de Docker pueden extenderse desde otras imágenes existentes, por lo que, para esta imagen, haremos una extensión desde una imagen de node existente.

Inicie el editor

$nano Dockerfile

Agregue este contenido:

```
FROM node:6.9.2
EXPOSE 8080
COPY server.js .
CMD node server.js
```

La "receta" para la imagen de Docker hará lo siguiente:

- Comenzará desde la imagen `node` que se encuentra en el hub de Docker.
- Expondrá el puerto 8080.
- Copiará el archivo `server.js` en la imagen.
- Iniciará el servidor web de node como ya hicimos manualmente.

Guarde este `Dockerfile`. Para ello, presione (Ctrl O-Enter-Ctrl-X)

Compile la imagen con el siguiente comando y ***reemplace `PROJECT_ID` por su ID de proyecto de GCP que se puede ver en la consola de GCP o ejecutando el comando (`$gcloud config list project`)***

$docker build -t gcr.io/PROJECT_ID/hello-node:v1 .

La compilación de la imagen docker tardará un tiempo. Una vez finalizada, probaremos la imagen localmente con el siguiente comando que ejecutará un contenedor de Docker como un daemon (sin interacción con el usuario) en el puerto 8080.

(reemplace `PROJECT_ID` por el ID de su proyecto de GCP)

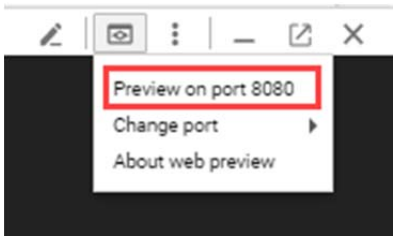
$docker run -d -p 8080:8080 gcr.io/PROJECT_ID/hello-node:v1

El resultado debería ser algo similar a :

```
$325301e6b2bffd1d0049c621866831316d653c0b25a496d04ce0ec6854cb7998
```

Que es la ID del contenedor que se está ejecutando.

Puede usar la función de vista previa web de Cloud Shell para ver los resultados:



O utilice `curl` desde la misma ventana de Cloud Shell:

$curl http://localhost:8080

Este es el resultado que debería ver:

```
Hello World!
```

A continuación, detenga el contenedor en ejecución.

Para encontrar el ID del contenedor de Docker, ejecutamos:

```
$docker ps
```

El resultado debería verse así:

```
CONTAINER ID        IMAGE                                  COMMAND
2c66d0efcbd4        gcr.io/PROJECT_ID/hello-node:v1    "/bin/sh -c 'node
```

Detenga el contenedor con el siguiente comando y ***reemplace CONTAINER ID*** por el valor que se obtuvo en el paso anterior (los dos primeros caracteres son suficientes):

```
$docker stop [CONTAINER ID]
```

El resultado de su consola debería verse así (ID de su contenedor):

```
2c66d0efcbd4
```

Ahora que la imagen funciona correctamente la guardaremos en Google Container Registry, un repositorio privado para sus imágenes de Docker accesible desde sus proyectos de Google Cloud.

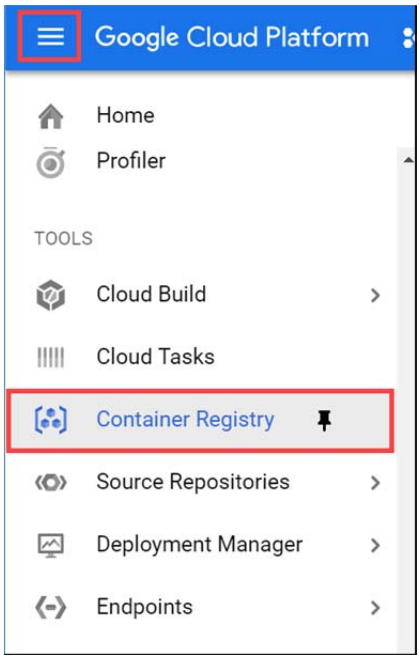
Ejecute los siguientes comandos y ***reemplace `PROJECT_ID` por su ID de proyecto*** de GCP

```
$gcloud auth configure-docker
$docker push gcr.io/PROJECT_ID/hello-node:v1
```

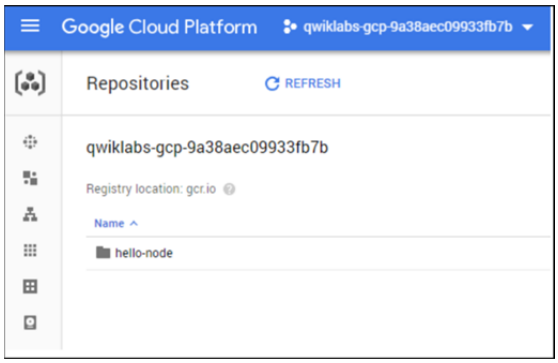
La operación podría tardar unos minutos en completarse.

```
The push refers to a repository [gcr.io/qwiklabs-gcp-6h281a111f098/hello-
node]
ba6ca48af64e: Pushed
381c97ba7dc3: Pushed
604c78617f34: Pushed
fa18e5ffd316: Pushed
0a5e2b2ddeaa: Pushed
53c779688d06: Pushed
60a0858edcd5: Pushed
b6ca02dfe5e6: Pushed
v1: digest:
sha256:8a9349a355c8e06a48a1e8906652b9259bba6d594097f115060acca8e3e941a2
size: 2002
```

La imagen del contenedor aparecerá en Console. Seleccione el **menú de navegación** y, luego, **Container Registry**.



Ahora posee una imagen de Docker disponible para todo el proyecto, a la que Kubernetes puede acceder y que puede utilizar deberíamos ver algo parecido a esto:



Crear un clúster de Kubernetes

Ahora que tenemos nuestra imagen docker en el repositorio podemos crear un clúster de Container Engine. Un clúster consiste en un servidor de API principal de Kubernetes (master) alojado por Google y en un conjunto de nodos trabajadores. Los nodos trabajadores son máquinas virtuales de Compute Engine.

Tenemos que asegurarnos de haber configurado nuestro proyecto con `gcloud` (***reemplace `PROJECT_ID` con su ID de proyecto de GCP*** .

```
$gcloud config set project PROJECT_ID
```

Cree un clúster con dos nodos n1-standard-1 (esto tardará unos minutos en completarse):

```
$gcloud container clusters create hello-world \
                --num-nodes 2 \
                --machine-type n1-standard-1 \
                --zone us-central1-a
```

Podemos ignorar las advertencias que aparecen cuando se construye el clúster.

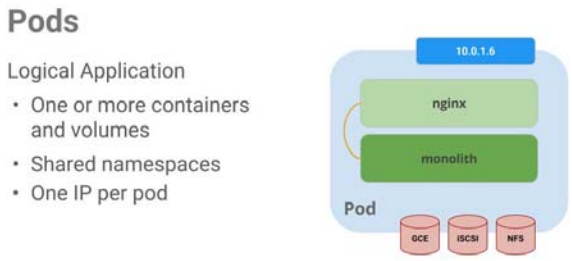
El resultado de Console debería ser similar al siguiente:

```
Creating cluster hello-world...done.
Created [https://container.googleapis.com/v1/projects/PROJECT_ID/zones/us-
central1-a/clusters/hello-world].
kubeconfig entry generated for hello-world.
NAME         ZONE           MASTER_VERSION  MASTER_IP       MACHINE_TYPE
STATUS
hello-world  us-central1-a  1.5.7           146.148.46.124  n1-standard-1
RUNNING
```

A partir de ahora podemos implementar nuestra propia aplicación en contenedores en el clúster de Kubernetes. Usaremos la línea de comandos `kubectl` (intérprete de comandos de kunernetes, que ya está configurada en su entorno de Cloud Shell).

Creación de un *pod*

Un **pod** de Kubernetes es un grupo de contenedores agrupados que comparten funciones administrativas y herramientas de redes. Puede tener uno o muchos contenedores.



En este lab usaremos un contenedor con la imagen de Node.js almacenada en nuestro Container Registry. Este entregará contenido en el puerto 8080.

Crearemos el pod con el comando `kubectl run` (***reemplace `PROJECT_ID` por su ID de proyecto de GCP*** .

```
$kubectl run hello-node \
    --image=gcr.io/PROJECT_ID/hello-node:v1 \
    --port=8080
```

Este es el resultado que debería ver:

```
deployment "hello-node" created
```

Como podemos ver, Kubernetes creó un objeto de **implementación**. Las implementaciones son la forma recomendada para crear y escalar pods. Aquí, una nueva implementación administra una única réplica de pod que ejecuta la imagen `hello-node:v1`.

Ejecute el siguiente comando para ver la implementación:

```
$kubectl get deployments
```

Este es el resultado que debería ver:

```
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-node   1         1         1            1           2m
```

Para ver el pod creado por la implementación, ejecute lo siguiente:

```
$kubectl get pods
```

Este es el resultado que debería ver:

```
NAME                          READY     STATUS    RESTARTS   AGE
hello-node-714049816-ztzrb    1/1       Running   0          6m
```

Ahora es un buen momento para revisar algunos comandos `kubectl` interesantes. Ninguno de estos cambiará el estado del clúster. La documentación completa está disponible aquí:

```
$kubectl cluster-info
$kubectl config view
```

Y para la solución de problemas:

```
$kubectl get events
$kubectl logs <pod-name>
```

Ahora haremos que nuestro ***pod*** sea accesible públicamente.

Permitir el tráfico externo

Según la configuración predeterminada, el pod únicamente es accesible mediante su IP interna dentro del clúster. Para hacer accesible el contenedor `hello-node` desde fuera de la red virtual de Kubernetes, tenemos que exponer el pod como un **servicio** de Kubernetes.

Desde Cloud Shell, puede exponer nuestro ***pod*** a Internet con el comando `kubectl expose` y el flag `--type="LoadBalancer"`. Esta marca es necesaria para la creación de una IP accesible de forma externa:

```
$kubectl expose deployment hello-node --type="LoadBalancer"
```

Este es el resultado que debería ver:

```
service "hello-node" exposed
```

La marca utilizada en este comando especifica que se usará el balanceador de cargas provisto por la infraestructura subyacente (en este caso, el balanceador de cargas de Compute Engine). *Tenga en cuenta que se expone directamente la implementación, no el pod*. Esto hará que el servicio resultante balancee la carga de tráfico en todos los *pods*

administrados por la implementación (en este caso, solo 1 *pod*, luego agregaremos más réplicas).

La instancia principal de Kubernetes crea el balanceador de cargas y las reglas de reenvío de Compute Engine relacionadas, los grupos de destino y las reglas de firewall para que el servicio se vuelva completamente accesible desde el exterior de Google Cloud Platform.

Para obtener la dirección IP del servicio que es accesible públicamente accesible, ejecutamosr:

```
$kubectl get services
```

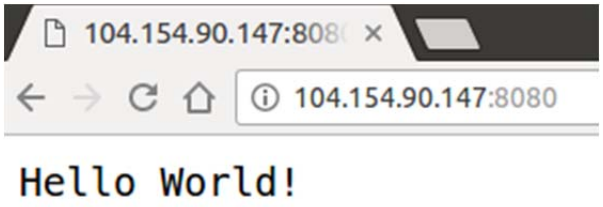
Este es el resultado que debería ver:

```
NAME         CLUSTER-IP     EXTERNAL-IP      PORT(S)    AGE
hello-node   10.3.250.149   104.154.90.147   8080/TCP   1m
kubernetes   10.3.240.1     <none>           443/TCP    5m
```

Se mencionan 2 direcciones IP para su servicio hello-node, y ambas entregan contenido al puerto `8080`. CLUSTER-IP corresponde a la IP interna que solo es visible dentro de su red virtual de Cloud, mientras que EXTERNAL-IP corresponde a la IP externa de balanceo de cargas.

La `EXTERNAL-IP` podría tardar varios minutos en estar disponible y visible. Si falta la `EXTERNAL-IP`, esperaremos unos minutos y probamos de nuevo con el comando anterior.

Ahora debería poder acceder al servicio dirigiendo su navegador a esta dirección:
`http://<EXTERNAL_IP>:8080`



Escalado vertical de nuestro servicio

Una de las características avanzadas que ofrece Kubernetes es la gran facilidad para escalar una aplicación. Supongamos que, de repente, necesitamos más capacidad para, en este caso atender peticiones desde un navegador, podemos decirle a kubernetes que cree una cantidad de réplicas para nuestro ***pod***:

```
$kubectl scale deployment hello-node --replicas=4
```

Este es el resultado que debería ver:

```
deployment "hello-node" scaled
```

Puede pedir una descripción de la implementación actualizada:

```
$kubectl get deployment
```

Este es el resultado que debería ver:

```
NAME         DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
hello-node   4         4         4            4           16m
```

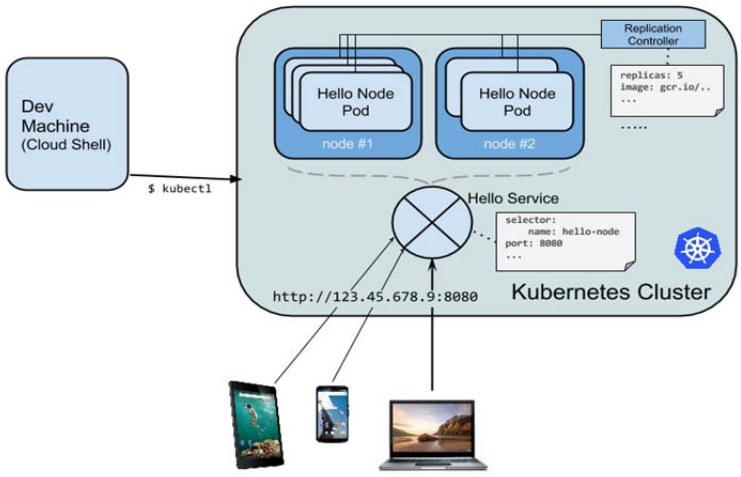
También se puede obtener una lista de todos los pods:

```
$kubectl get pods
```

Este es el resultado que debería ver:

```
NAME                          READY     STATUS    RESTARTS   AGE
hello-node-714049816-g4azy    1/1       Running   0          1m
hello-node-714049816-rk0u6    1/1       Running   0          1m
hello-node-714049816-sh812    1/1       Running   0          1m
hello-node-714049816-ztzrb    1/1       Running   0          16m
```

La siguiente figura representa un diagrama que resume el estado de nuestro clúster de Kubernetes, Podemos probar a conectarnos con el navegador y ver que nuestro servicio sigue respondiendo con la ventaja de que ahora el tráfico irá a cualquiera de las cuatro instancias dependiendo del balanceo de carga y todo gestionado por Kubernetes.



Una vez finalizada la práctica, ir a la consola de google (compute engine-virtual machines) y detener las instancias que se han utilizado para el cluster kubernetes

A entregar :

Resumen de 4 páginas sobre KUBERNETES, sus componentes principales y organización, con especial énfasis en Google Kubernetes Engine