MEDIAMARKT CLOUD ENGINEERING CHALLENGE

27/03/23

Miguel Estevan Moreno

Tabla de contenido

Prerrequisitos y herramientas

La solución se va a llevar a cabo en un sistema Linux con distribución Ubuntu 22.04, y las herramientas a utilizar van a ser:

- Docker.
- Terraform.
- Google Kubernetes Engine (GKE).
- Container Registry.

Tarea 1 – Generar imagen y subida a Container Registry

Para la generación de una imagen en base a un fichero Dockerfile se requiere alguna de las múltiples herramientas para trabajar con contenedores, como Podman, Buildah o Docker, siendo este último el escogido para resolver el reto. Respecto al repositorio de imágenes, se nos proporciona acceso a la nube pública de Google, el cual tiene un servicio de repositorio de imágenes llamado Container Registry. Para la generación de la imagen y subida al repositorio de imágenes se han seguido los siguientes pasos:

- Clonado de repositorio proporcionado para el reto.

  ```
  git clone https://github.com/nuwe-io/mms-cloud-skeleton.git
  ```

- Generación de la imagen con Docker.

  ```
  docker build -t devops-challenge:latest .
  ```

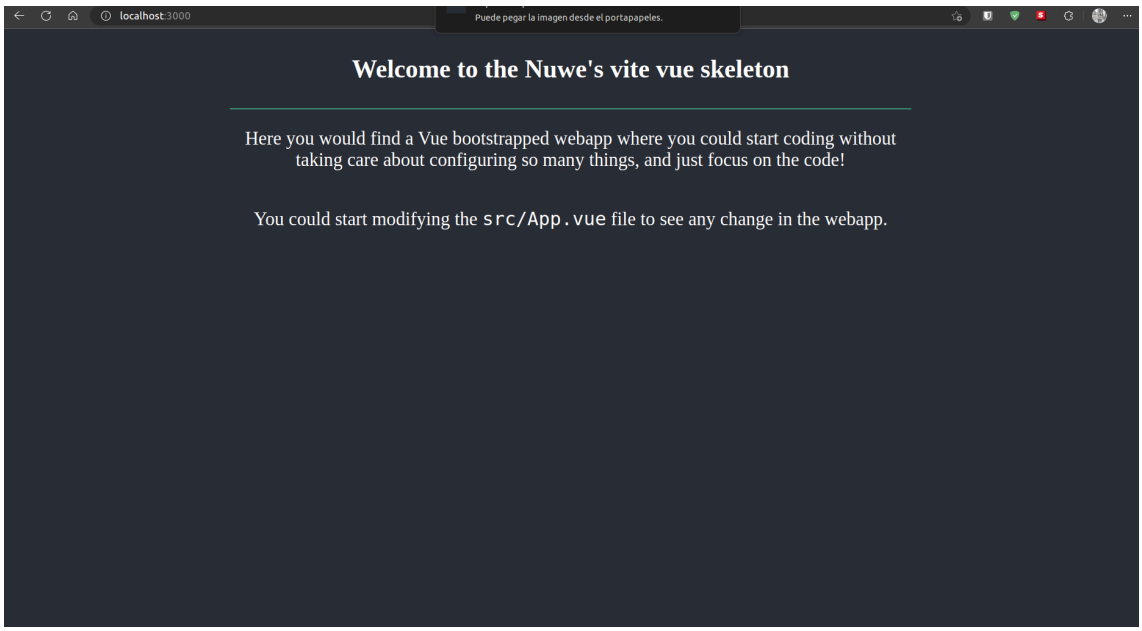
- Tras previo registro en el SDK de Google para poder subir imágenes, tageamos la imagen y la subimos al Container Registry de Google.

  ```
  docker tag devops-challenge:latest eu.gcr.io/<project-id>/devops-challenge:latest
  docker push eu.gcr.io/<project-id>/devops-challenge:latest
  ```

En caso de que se quiera comprobar en local que la imagen funciona, bastaría con ejecutar en local:

```
docker container run -it -p 3000:3000 <image-id>
```

Tras ejecutar este comando, abrir el navegador, y acceder a localhost:3000 para comprobar que aplicación levanta este contenedor.



Tarea 2 – Generar YAML de despliegue sobre Kubernetes

Docker Composer te permite la ejecución de varios contenedores mediante la definición de ficheros YAML. En este caso, el YAML resultante es muy sencillo al tratarse de un solo contenedor. El YAML se encuentra en:

- yaml/docker-compose/docker-compose.yaml

Para comprobar su ejecución basta con ejecutar:

```
docker compose up
```

Tarea 3 – Generación de los ficheros Terraform y YAML GKE

Definición de recursos desarrollados

Los recursos de infraestructura generados con Terraform son:

1. ***Namespace* de Kubernetes**: Espacio virtual nativo de Kubernetes para poder alojar no solo el despliegue de la aplicación, sino todos aquellos recursos de Kubernetes para vitaminar dicho despliegue.
2. **Despliegue de la aplicación**: El despliegue de la aplicación se ha vitaminado con dos características a destacar. La primera es que se ha establecido una política a la hora de actualizar el despliegue, de forma que durante la actualización del mismo siempre haya un *pod* ofreciendo servicio. Se ha indicado de forma que se tiene que actualizar solo el 25% de los pods sobre el número de pods deseados, de forma que no se actualizan todos a la vez. También se ha indicado que solo puede haber una indisponibilidad de los dos del 25%, de forma que siempre se garantice que por lo menos haya un pod en estado *running* dando servicio.

   La segunda característica clave son los *health checks*, los cuales van monitorizando la salud del pod, de forma que se puede llegar a recrear si lleva tiempo en un estado no deseado. El primero de los *health checks, el liveness probe*, comprueba cada 3 segundos si el *pod*, está levantado, sin tener porque estar en un estado *ready*, dándole un margen de 30 segundos una vez se levanta. El segundo de los *heath cheks*, el *readiness probe*, comprueba cada 10 segundos si es *pod* está en estado running, dándole un margen de 10 segundos una vez se empieza a desplegar el *pod*.
3. **Reserva de IP externa para *Ingress* nativo de GKE**: Se ha utilizado el recurso del provider de Google que nos permite reservar IP's globales, de forma que posteriormente se indicará esta IP pública para asociarla al balanceador expuesto a Internet, el cual se desplegará gracias a la definición indicada en el *ingress*, que explicaré a continuación.
4. **Servicio de Kubernetes**: Los servicios de Kubernetes son los encargados de reenviar tráfico hacia los pods. En este caso se ha escogido el servicio *ClusterIP*, de forma que tiene asociada una IP virtual interna, por lo que no es accesible desde internet. Para este reto será el recurso *ingress* de Kubernetes configurado

de forma que levante un balanceador nativo de Google expuesto a Internet, el cual recogerá las llamadas y las redirigirá de forma interna a este servicio, y posteriormente al *pod*.

5. ***Ingress* nativo de GKE**: Los recursos de tipo *ingress* son el punto de entrada a nuestras aplicaciones desplegadas en Kubernetes, nos permiten entrar mediante tráfico HTPPS por el navegador, además de permitirnos establecer el DNS por el cual entrar, TLS para tráfico HTTPS, o el enrutado a nivel de servicio en función del *path* indicado en la URL. En este caso, al no disponer de DNS ni certificado TLS, simplemente se ha indicado que levante un balanceador externo con la IP reservada anteriormente.
6. ***Horizontal Pod Autoscaler* (HPA)**: Este tipo de recursos de Kubernetes permiten escalar los *pods* de la aplicación en base a las métricas indicas en la definición del mismo, por ejemplo, que los pods de la aplicación escalen cuando lleguen al 80% uso de CPU, o cuando alcancen un alto porcentaje de uso de memoria. En este caso, se ha aplicado un HPA para que escalen al 70% del uso de CPU.
7. **Cuotas**: Cuando un *cluster* de Kubernetes se divide en *namespaces* es recomendable establecer cuotas a dichos *namespaces* para que los despliegues no consuman más recursos de los necesarios del nodo, y se haga un uso ineficiente de los recursos, ya que en nube pública se paga por uso, y prima que se haga un uso eficiente de todos los recursos. En este caso, solo se han establecido límite para las coutas y no *requests*, calculando aproximadamente el límite para 3 *pods*.
8. ***Limit Range* a nivel de contenedor**: A pesar de que este recurso de se ha dejado comentado en el código de Terraform que ya que no aportaba valor debido a que ya existen las cuotas, los *limits ranges* de Kubernetes te permiten establecer límites de infraestructura a nivel de los contenedores internos de un pod, o límites para un solo *pod*, mientras que las cuotas se aplican a nivel de *namespace*.

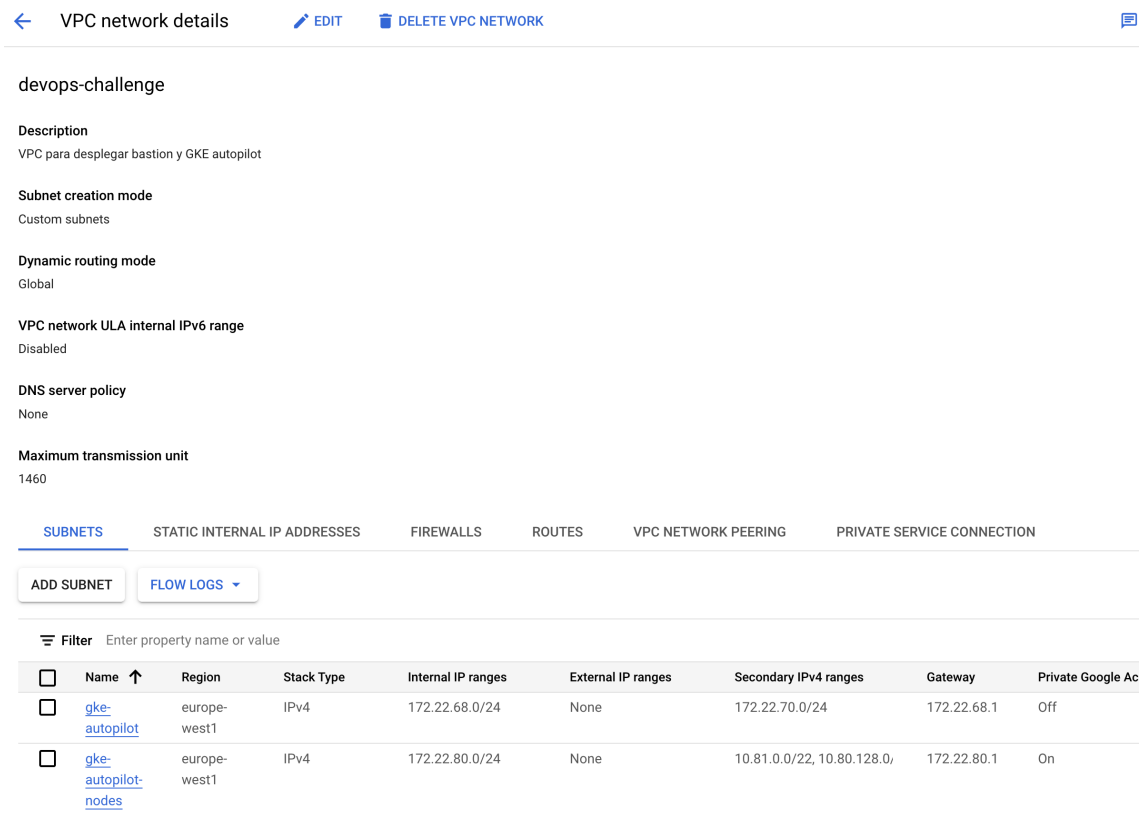
Además del desarrollo de estos recursos mediante Terraform, también se han implementado en ficheros YAML declarativos compatibles con Kubernetes, agrupados todos en un fichero kustomization.yaml. Este fichero kustomization tiene un comportamiento similar a Helm, y es que permite desplegar mediante una sola llamada

con el cliente de Kubernetes todos los ficheros .yaml indicados en su definición, además de poder configurar y desplegar *configmaps* complejos, o secretos o CRD's.

Despliegue de recursos

Para probar el despliegue de estos recursos, ya sea por Terraform o mediante el cliente de Kubernetes, se ha decidido desplegar un cluster de GKE Autopilot público, ya que para ello no se requieren muchos recursos de red, y su despliegue es relativamente rápido y sencillo, aunque las buenas prácticas recomienden un clúster privado para la VPC, además de, tener acotados los rangos de red que puedan atacar a dicho clúster, y creadas las reglas de firewall necesarias para interactuar con él y aplicadas con *network tags* de Google, por ejemplo.

En primer lugar, y además de ser una buena práctica, se ha borrado la VPC que viene por defecto para desplegar una limpia con aquellos recursos necesarios para el despliegue del clúster GKE Autopilot. Una vez borrado se ha creado la VPC *devops challenge*, dentro de la cual se han desplegado dos subredes, todo esto por la consola de GCP.



Una vez desplegados los recursos de red, se puede desplegar el clúster de GKE Autopilot público. Para ello, basta con indicar el rango de nodos en la configuración de red

solicitada, además de indicar la misma región en la que se ha desplegado las redes, dejando el resto de opciones por defecto. Una vez se tiene el clúster, basta con obtener las credenciales del mismo con el comando que te proporciona la consola de Google para conectarte a él, y probar el despliegue de los recursos desarrollados.

Para probar el despliegue con Terraform basta con entrar por terminal a la carpeta Terraform del repositorio de Github y ejecutar:

```
terraform init
terraform plan # Comprobar conf. de recursos
terraform apply
```

A la hora de trabajar con Terraform las buenas prácticas indican que se debe indicar un *backend* para securizar y garantizar el acceso al tfstate, fichero esencial para el flujo de trabajo con Terraform. Una buena opción para este reto podría ser un *bucket* de Google Cloud Storage (GCS) que tuviera acceso por una serie de cuentas de servicio con permisos, además de tener habilitado versionado en el *bucket*. Para este reto no se ha implementado ya que no se podían asignar permisos a las cuentas de servicios creadas.

Para probar el despliegue mediante el cliente de Kubernetes, basta que entrar en la carpeta yaml/gke por terminal del repositorio de Github y ejecutar:

```
kubectl apply -k .
```

Tarea 4 – Pregunta sobre IAM

A la hora de seguir el principio de menos privilegios para la asignación de permisos, ya sean a usuarios, grupos de usuarios, cuentas de servicio, etc, una alternativa puede ser crear un rol personalizado con los permisos individuales que requieren cada una de las partes, o investigar cual es el rol predefinido de Google que más se ajusta a las necesidades del usuario/s. El primero de los casos puede ser correcto para roles no críticos, pero se corre el riesgo de que Google cambie o depreque el nombre o algo relacionado con alguno de los permisos que forman parte del rol personalizado, y se pierda la función de este rol personalizado, por lo que se va a escoger la segunda opción.

Para ello, basta con leer la documentación de Google (recomendable escoger inglés, es la mejor mantenida), para comprobar que el rol predefinido que mejor se adapta para el equipo de DevOps es el rol *Kubernetes Engine Cluster Admin* (a nivel de API *roles/container.clusterAdmin*), ya que les ofrece un control total sobre la administración

del clúster, de forma que sea el equipo de desarrollo junto al de CI/CD el que mantenga las aplicaciones de dentro.

Respecto al rol a asociado al equipo financiero, se procede de la misma forma, buscamos en la documentación de Google qué rol sería el ideal para sus necesidades. En base a lo indicado en la documentación de Google, el rol que se escogería sería el de *Billing Account Administrator* (a nivel de API *roles/billing.admin*), ya que este no permite crear nuevas cuentas de facturación, pero si gestionar todo lo relacionado con la gestión de las cuentas de facturación de Google, gastos de proyectos, vinculación y desvinculación de proyectos, etc.