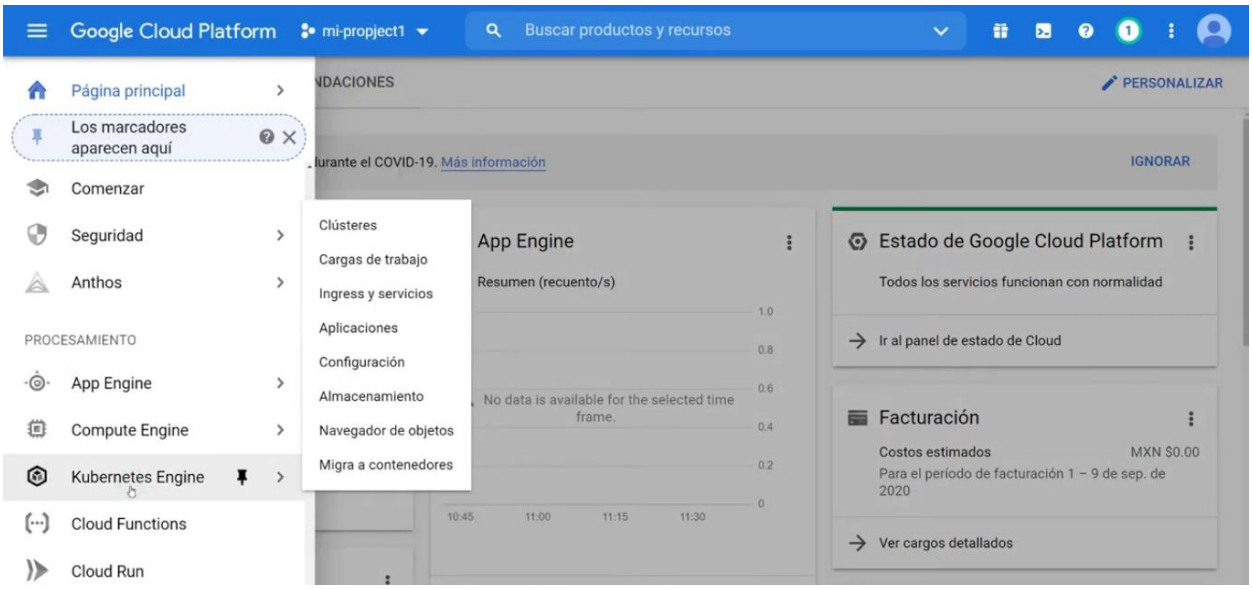
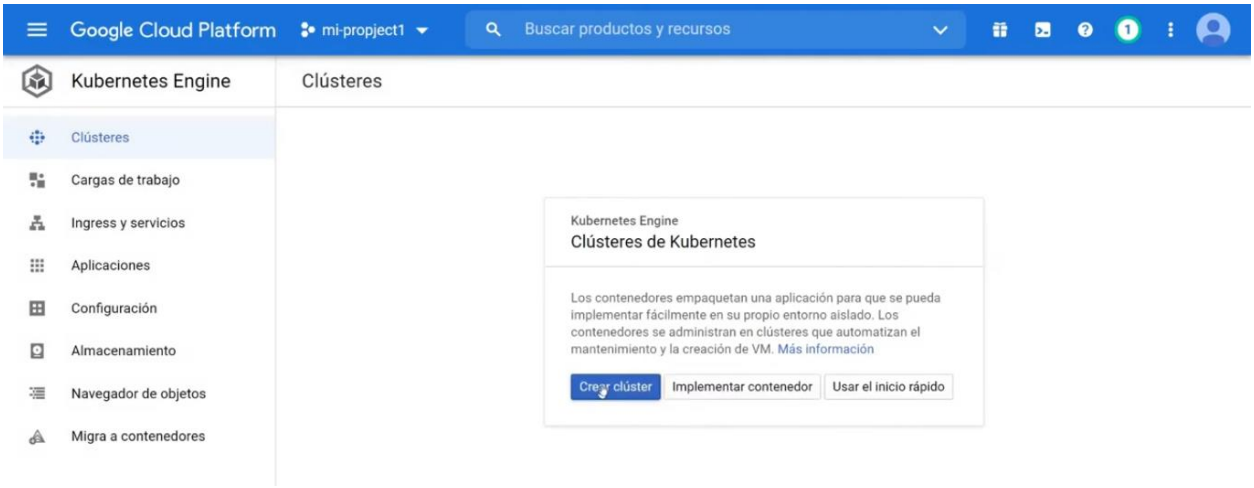
Manual de implementacion Kubernetes

Creación de Clúster en GCP

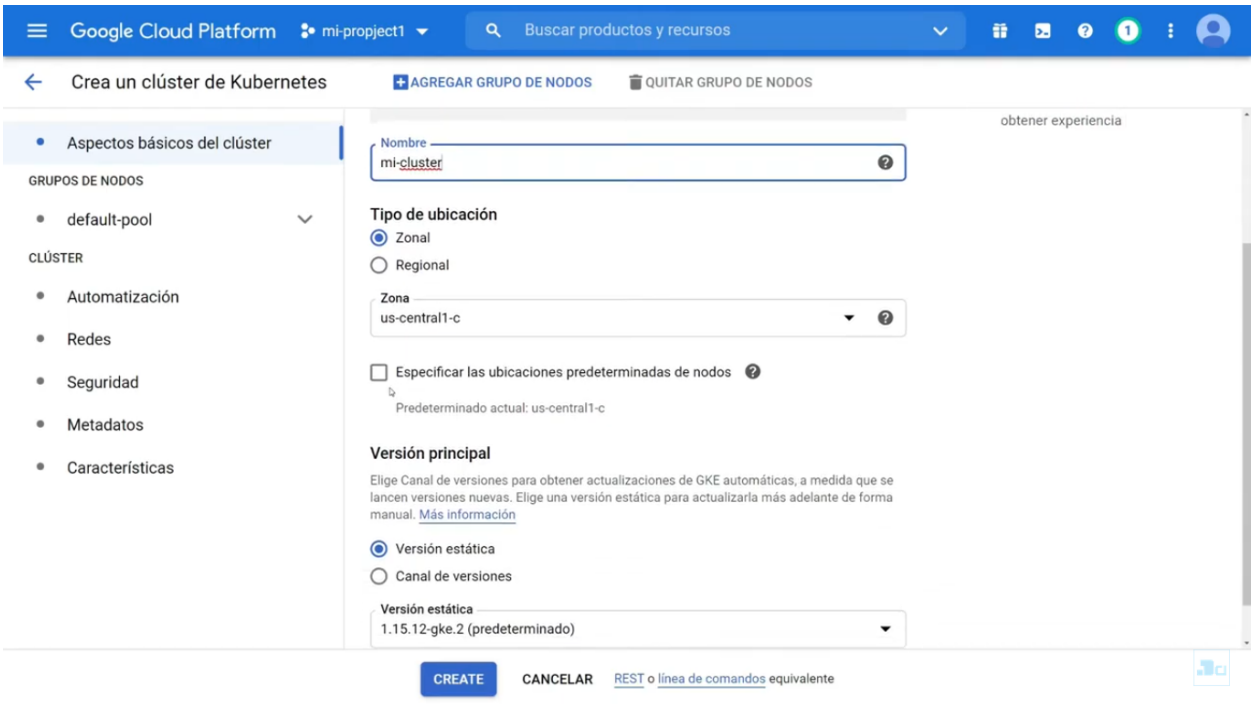
Primero nos dirigimos a la pagina de Google Cloud Plataform, y en el menú buscamos la opción de clústeres.



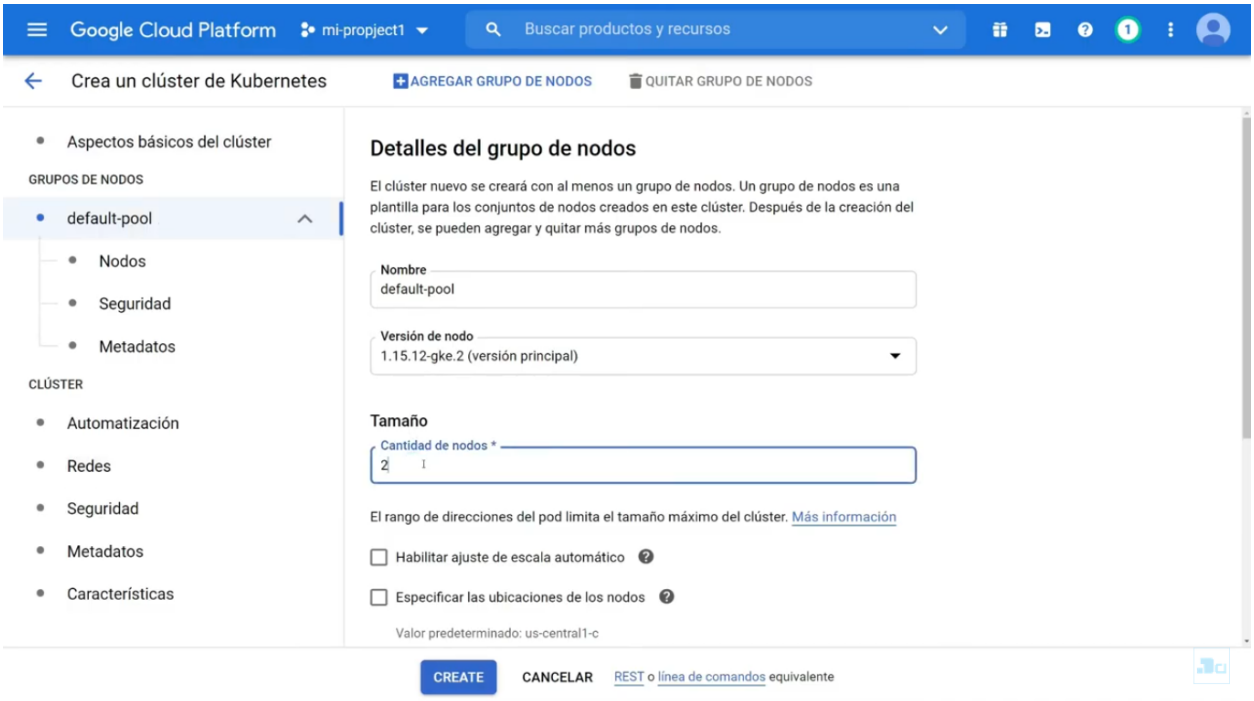
Una vez en la sección de kubernets engine, buscamos la opción crear clúster y lo seleccionamos, estos clusteres permiten alojar los pods que alojaran nuestros contenedores.



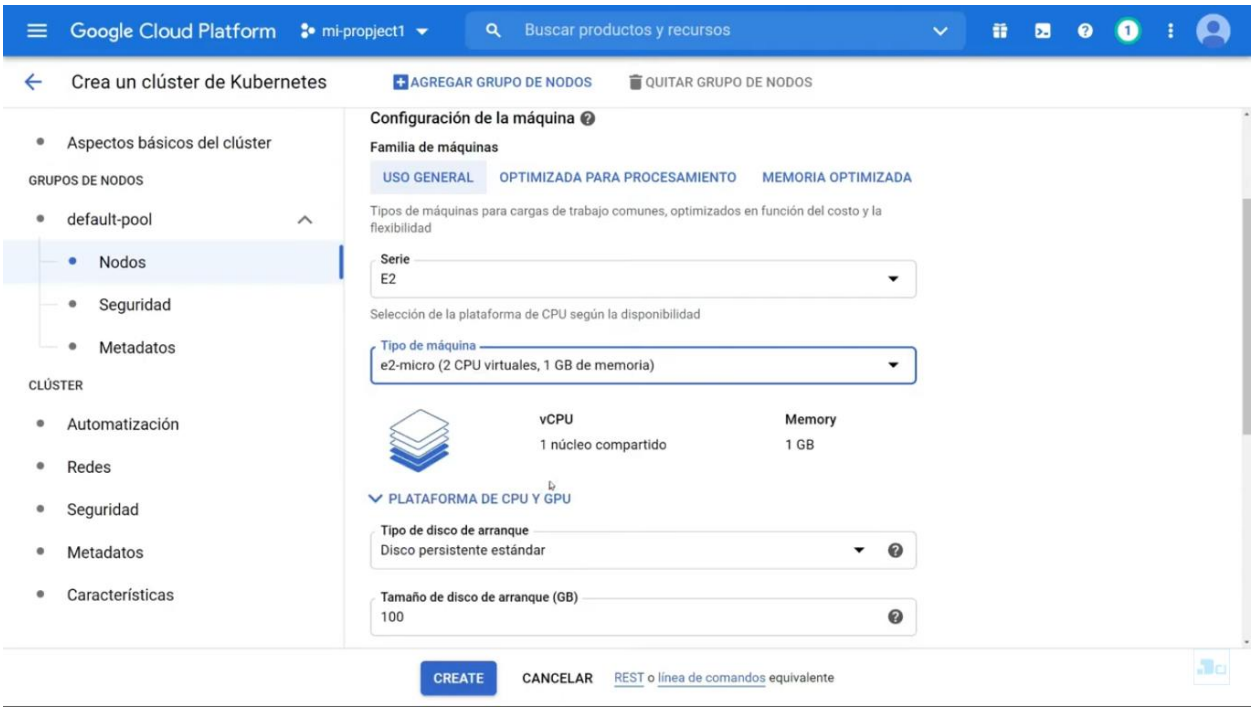
Al darle clic en crear, nos aparecerá un formulario donde debemos ingresar el nombre del clúster, lo demás se puede quedar con los valores por defecto



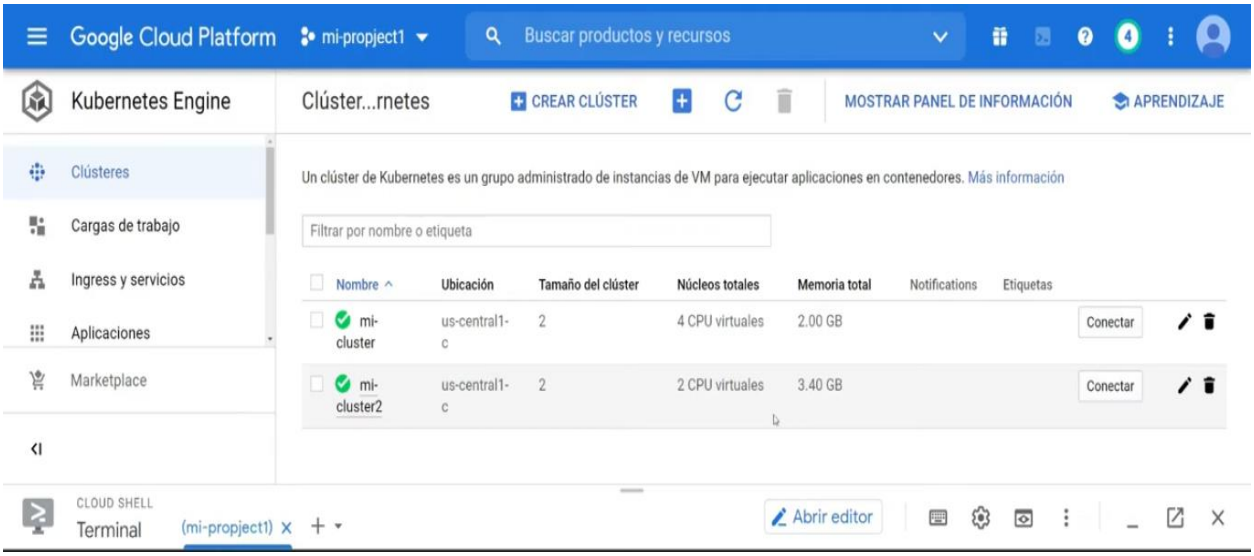
Nos dirigimos a la sección de la izquierda donde está la opción default-pool, para determinar la cantidad de nodos, para nuestro caso necesitamos 2 nodos para frontend.



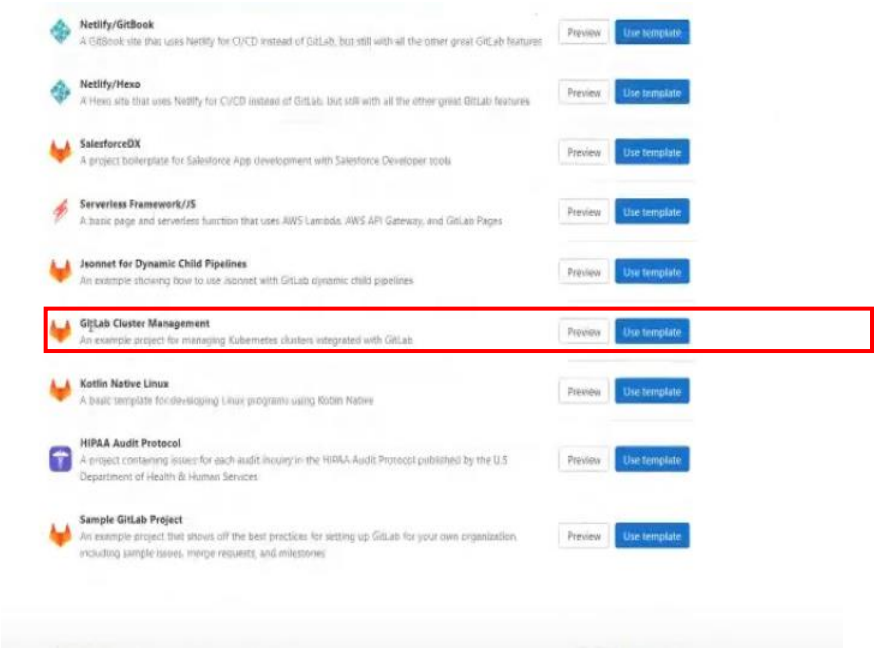
Ahora nos vamos a la opción de nodos, y vamos a elegir el tipo de maquina para nuestro clúster, por temas de costos elegimos la serie E2-micro que contiene 2 CPU virtuales y 1G de memoria. Seleccionamos el botón crear esperamos a que termine de levantar la máquina.



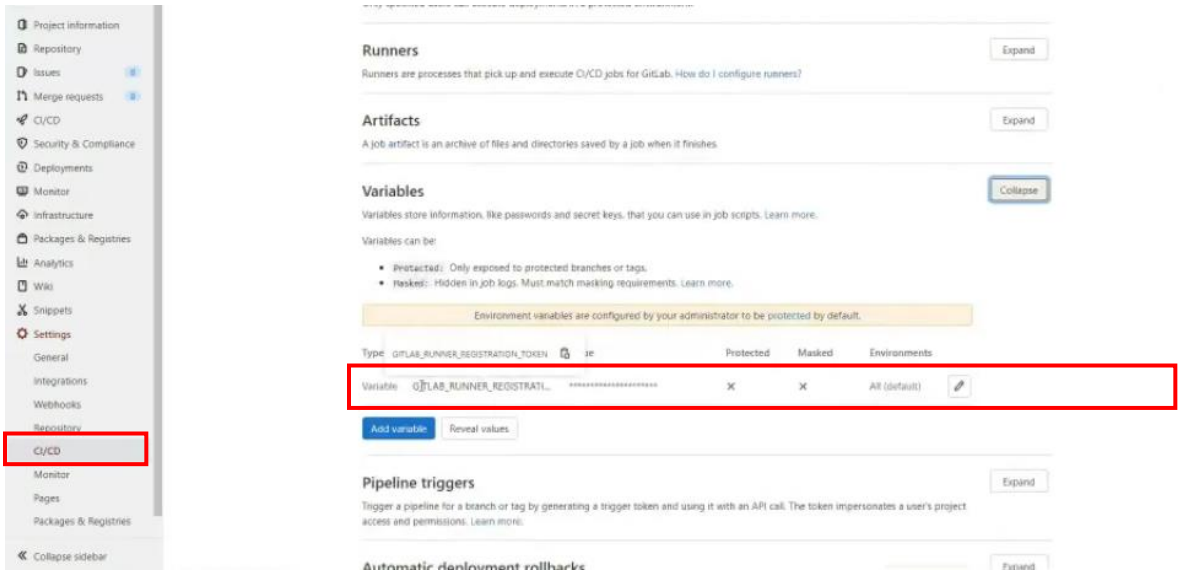
Para crear un nuevo clúster únicamente debemos seguir los pasos anteriores, el segundo clúster será para alojar el backen.



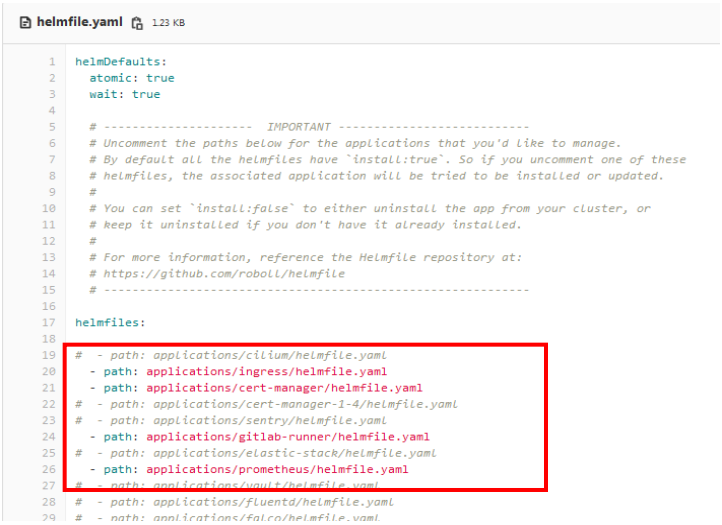
Agregar la plantilla de clúster manager a nuestro repositorio, esta plantilla se agrega cuando creamos un repositorio a partir de una plantilla desde gitlab o bien podemos agregar la plantilla después de haber creado el repositorio. Esta plantilla incluye archivos de configurados, para que gitlab puede interactuar con un clúster



Agregar el token de registro de gitlab runner, para eso nos dirigimos al menú de la izquierda en la opción Settings –> CI/CD -> Variables



Agregar el archivo helmfile.yaml al repositorio, este archivo es básicamente un gestor de paquetes, donde nosotros le decimos que paquetes queremos que instale. Este archivo es muy útil para cuando realizamos integraciones con CI/CD. En la siguiente imagen se recomiendan instalar los módulos se indican en el recuadro.



Una vez que tengamos estas configuraciones, debemos instalar lo siguiente en nuestra máquina de desarrollo:

Instalar gitlab-runner

curl -L "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash

sudo apt-get install gitlab-runnerp

Permitir que se ejecute kubectl en nuestro cluster

Instalar docker

sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io

Darle todos los privilegios al usuario de gitlab-runner, para evitar problemas de permisos

sudo nano /etc/pam.d/su
auth [success=ignore default=1] pam_succeed_if.so user = gitlab-runner
auth sufficient pam_succeed_if.so use_uid user ingroup gitlab-runner
sudo usermod -aG gitlab-runner ubuntu
su - gitlab-runner

auth [success=ignore default=1] pam_succeed_if.so user = gitlab-runner
auth sufficient pam_succeed_if.so use_uid user ingroup gitlab-runner
sudo usermod -aG gitlab-runner ubuntu
su - gitlab-runner
sudo usermod -aG gitlab-runner $USER
newgrp gitlab-runner

Instalar Gooble CP y Kuteclt, se recomienda hacerlo con el usuario de gitlab-runner.

1. Agrega el URI de distribución del SDK de Cloud como una fuente de paquete:

echo "deb [signed-by=/usr/share/keyrings/cloud.google.gpg] https://packages.cloud.google.com/apt cloud-sdk main" | sudo tee -a /etc/apt/sources.list.d/google-cloud-sdk.list

sudo apt-get install apt-transport-https ca-certificates gnupg

2. Importar la clave publica de GP

curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key --keyring /usr/share/keyrings/cloud.google.gpg add -

Instalar y actualizar el SDK de cloud

sudo apt-get update && sudo apt-get install google-cloud-sdk

3. Ejecutar gcloud init
gcloud init

Instalar kubectl desde el cluster

1.Descargar la ultima version

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

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

Una vez instalado tolas las herramientas y modificado los permisos para el usuario gitlab-runer para que pueda ejecutar kubeclt, ahora podemos configurar nuestro flujo pipeline en el archivo gitlab-ci.yml

Debemos asegurarnos de que kubectl sea ejecutado por el usuario gitlab-runer y no por el usuario por defecto de Ubuntu, para eso podemos ejecutar el comando whoami.

Flujo del stage de kubernetes

```
deploy-job-frontend:
  stage: kuberentes-stage
  only:
    - "main"
    - "develop"
    #- "feature/decorator"
  tags:
    - "ayd2_test"
  before_script:
    - cd ./angular-app
    - docker image rmi selvinlp/serviciofrontend:latest -f
    - docker pull selvinlp/serviciofrontend:latest
  script:
    #Debe estar dentro del directorio angular-app y ya deben estar creadas las imagenes para que funcione.
    - echo "Ejecutando deploy Frontend Kubernetes y LoadBalancer ... "
    - gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project ayd2-327723
    - kubectl get services
    - kubectl delete svc aydrive-front1
    #- kubectl delete svc aydrive-front2
    - kubectl delete deployments load-balancer-front-aydrive

    - kubectl apply -f load-balancer-frontAYD2.yaml
    - kubectl expose deployment load-balancer-front-aydrive --type=LoadBalancer --name=aydrive-front1
    #- kubectl expose deployment load-balancer-front-aydrive --type=LoadBalancer --name=aydrive-front2
    - kubectl get svc
  after_script:
    #- docker-compose push
    - cd ./
    - echo "Terminando deploy frontend ... "

deploy-job-backend:
```

Pasos:

1. Nos movemos al directorio angular-app

cd ./angular-app

2.Eliminamos todas las imágenes de los contenedores del frontend

docker image rmi selvinlp/serviciofrontend:latest -f

3.Descargamos las imágenes del frontend

docker pull selvinlp/serviciofrontend:latest

4.Nos conectamos al cluster que creamos al inicio

5.gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project ayd2-327723

6.Eliminamos el servicio y el deployment de nuestro frontend

kubectl delete svc aydrive-front1

kubectl delete deployments load-balancer-front-aydrive

7.Creamos el deployment para actualizar los cambios que se hayan hecho

kubectl apply -f load-balancer-frontAYD2.yaml

8.Volvemos a exponer el deployment del frontend, para que nos proporcione una ip publica

kubectl expose deployment load-balancer-front-aydrive --type=LoadBalancer --name=aydrive-front1

9.Mostramos la ip publica asignada

kubectl get svc

Aplicar el mismo procedimiento para el kubernetes-stage del backend

```
84
85  deploy-job-backend:
86    stage: kuberentes-stage
87    only:
88      - "main"
89      - "develop"
90      #- "feature/decorator"
91    tags:
92      - "ayd2_test"
93    before_script:
94      - docker-compose down -v --rmi all
95      - docker-compose pull
96      - cd ./backend
97    script:
98      #Debe estar dentro del directorio backend y ya deben estar creadas las imagenes para que funcione.
99      - echo "Ejecutando deploy backend Kubernetes y LoadBalancer ... "
00      - gcloud container clusters get-credentials cluster-2 --zone us-central1-c --project ayd2-327723
01      - kubectl get services
02      - kubectl delete svc aydrive-backend1
03      #- kubectl delete svc aydrive-backend2
04      - kubectl delete deployments load-balancer-backend-aydrive
05
06      - kubectl apply -f load-balancer-backendAYD2.yaml
07      - kubectl expose deployment load-balancer-backend-aydrive --type=LoadBalancer --name=aydrive-backend1 --load-balancer-ip=35.232.111.58
08      #- kubectl expose deployment load-balancer-backend-aydrive --type=LoadBalancer --name=aydrive-backend1 --load-balancer-ip=34.134.217.79
09      - kubectl get svc
10    after_script:
11      #- docker-compose push
12      - cd ./
13      - echo "Terminando deploy backend ... "
```