UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS

LAB. SOFTWARE AVANZADO
ING. EVEREST DARWIN MEDINILLA RODRIGUEZ
AUX. DIEGO MOLINA

DOCUMENTACIÓN PRÁCTICA 3

201830313 - DENILSON FLORENTÍN DE LEÓN AGUILAR
201931581 - JONATHAN MARCOS VALIENTE GONZÁLEZ

Índice

Contratos de Microservicios

Contrato del servicio de nombre (Name Service)

- **Endpoint**: `http://name-service:3002/name?name=juan`
- **Método HTTP**: GET
- **Parámetros de consulta**:
    - `name`: El nombre para el cual se busca obtener información.
- **Respuesta esperada**: json

```
{
  "name": "marcos",
  "count": 24,
  "age": 38,
  "country_id": "GT"
}
```

Contrato del servicio de género (Gender Service)

Endpoint para obtener género

- **Endpoint**: `http://gender-service:3001/gender?name=juan`
- **Método HTTP**: GET
- **Parámetros de consulta**:
    - `name`: El nombre para el cual se busca obtener el género.
- **Respuesta esperada**:

```
{
  "gender": "male",
  "count": 578,
  "name": "marcos",
  "country_id": "GT",
```

```
  "probability": 1
}
```

Contrato del servicio de Edad (Age Service)

Endpoint para obtener edad

- **Endpoint**: `http://gender-service:3000/age?name=juan`
- **Método HTTP**: GET
- **Parámetros de consulta**:
    - `name`: El nombre para el cual se busca obtener la edad.
- **Respuesta esperada**: json

```
{
  "age": 38,
  "count": 24,
  "name": "marcos",
  "country_id": "GT"
}
```

Estos contratos especifican los endpoints, métodos HTTP, parámetros de consulta y el formato esperado de las respuestas para los servicios de nombre, género y edad.

Tecnologías Utilizadas y Justificación:

1. **Node.js:** Se eligió Node.js debido a su eficiencia y facilidad para construir aplicaciones escalables y de alto rendimiento. Además, cuenta con una amplia comunidad de desarrolladores y un ecosistema de módulos robusto que facilita el desarrollo de microservicios.
2. **Kubernetes para microservicios:**
    - **Jobs:** Utilizados para ejecutar tareas cronológicas o de procesamiento por lotes de forma confiable.

- **Pods:** Permiten empaquetar, distribuir y escalar aplicaciones de manera eficiente, garantizando la disponibilidad y la escalabilidad horizontal.
- **Ingress:** Proporciona un enrutamiento de tráfico HTTP y HTTPS desde el exterior del clúster hacia los servicios dentro del clúster, lo que permite una gestión avanzada de rutas y equilibrado de carga.

3. **Contenedores:**
    - **Docker:** Escogido por su portabilidad y facilidad de uso en el empaquetado y despliegue de aplicaciones. Permite crear entornos de desarrollo consistentes y reproducibles.
    - **Artifact Registry de Google Cloud Service:** Ofrece un registro privado y seguro de imágenes de contenedores, permitiendo un almacenamiento confiable y escalable de las imágenes Docker.
4. **Proveedor: Google Cloud Platform (GCP):**
    - **Alta Flexibilidad:** GCP proporciona una amplia gama de servicios gestionados que se integran perfectamente con Kubernetes, lo que permite una flexibilidad máxima en la implementación y gestión de infraestructura en la nube.
    - **Documentación Completa:** La documentación detallada de GCP facilita la comprensión y la implementación de las mejores prácticas para el despliegue y la gestión de aplicaciones en la nube.
    - **Créditos Iniciales:** Los créditos iniciales ofrecidos por GCP permiten comenzar a utilizar la plataforma de forma gratuita, lo que resulta beneficioso para proyectos de desarrollo y pruebas iniciales.
    - **Configuración de Kubernetes:** GCP ofrece herramientas y servicios específicos para Kubernetes, simplificando la configuración y gestión del clúster Kubernetes.
5. **CI/CD con GitHub Actions:**
    - **Conexión y Despliegue de Workflow:** GitHub Actions se utilizó para automatizar el proceso de integración continua y entrega continua (CI/CD), lo que permite desplegar

automáticamente las aplicaciones en el clúster Kubernetes después de cada confirmación de código.

- **Secretes:** GitHub Actions ofrece un almacenamiento seguro para secretos como credenciales y tokens de acceso, garantizando que la información confidencial esté protegida durante el proceso de despliegue.

6. **GitHub para Ramas:**
   - GitHub proporciona un sistema de control de versiones sólido que facilita la colaboración en equipo y el seguimiento de cambios en el código fuente a través del uso de ramas.
7. **Herramientas para Debuguear:**
   - **Postman:** Utilizado para probar y depurar los endpoints de la API de forma rápida y sencilla, facilitando la validación del comportamiento esperado de la aplicación.
8. **Pruebas:**
   - **Unitarias:**
     - **Mocha:** Marco de pruebas para Node.js que proporciona una estructura flexible y fácil de usar para escribir y ejecutar pruebas unitarias.
     - **Sinon:** Herramienta de simulación y espionaje que facilita la escritura de pruebas unitarias al permitir la simulación de comportamientos y la verificación de interacciones entre componentes.
9. **Bash:**
   - Bash se utilizó para la automatización de tareas de administración del sistema y para la creación de scripts de implementación y configuración, lo que contribuye a la eficiencia y la consistencia en el flujo de trabajo de desarrollo.

Gestión de aplicaciones

Para realizar el despliegue de aplicaciones utilizando Kubernetes y Container Registry, necesitaremos seguir los siguientes pasos:

1. **Configurar el entorno de Kubernetes:**
    - Antes de comenzar, asegúrate de configurar el entorno de Kubernetes. Puedes hacerlo utilizando el comando:

```
gcloud container clusters get-credentials cluster-pr2 --zone
us-central1-c --project carbon-ray-415302
```

    - Este comando obtiene las credenciales del clúster de Kubernetes para que puedas interactuar con él desde tu entorno local.

2. **Configurar Docker para autenticación:**
    - Configura Docker para que pueda autenticarse con Container Registry utilizando el comando:

```
gcloud auth configure-docker us-east1-docker.pkg.dev
```

    - Esto permite a Docker autenticarse y empujar las imágenes de contenedor al registro de contenedor de Google.

3. **Construir las imágenes de Docker:**
    - Construye las imágenes de Docker para tus microservicios. Por ejemplo:

```
docker build -t us.gcr.io/carbon-ray-415302/node_microservice_age:v1 .
docker build -t us.gcr.io/carbon-ray-415302/node_microservice_gender:v1 .
docker build -t us.gcr.io/carbon-ray-415302/node_microservice_name:v1 .
```

    - Asegúrate de que estás en el directorio adecuado que contiene el Dockerfile de cada microservicio.

4. **Empujar las imágenes a Container Registry:**
    - Una vez que las imágenes de Docker estén construidas, empújalas a Container Registry:

```
docker push us.gcr.io/carbon-ray-415302/node_microservice_age:v1
docker push us.gcr.io/carbon-ray-415302/node_microservice_gender:v1
```

```
docker push us.gcr.io/carbon-ray-415302/node_microservice_name:v1
```

5. **Despliegue en Kubernetes:**
    - Cambia al directorio que contiene los archivos YAML de los despliegues y servicios de Kubernetes:

```
cd k8s
```

Aplica los archivos YAML de despliegue y servicio para cada microservicio:

```
kubectl apply -f deployment-age.yaml
kubectl apply -f deployment-gender.yaml
kubectl apply -f deployment-name.yaml
kubectl apply -f service-age.yaml
kubectl apply -f service-gender.yaml
kubectl apply -f service-name.yaml
```

    - Esto creará los despliegues y servicios en tu clúster de Kubernetes, haciendo que tus microservicios estén disponibles para su uso.

6. **Verificar el estado del despliegue:**
    - Puedes verificar el estado de los servicios y despliegues utilizando los siguientes comandos:

```
kubectl get svc
kubectl get deploy
```

    - Estos comandos te mostrarán información sobre los servicios expuestos y los despliegues en tu clúster de Kubernetes.

7. **Acceder a los microservicios:**
    - Una vez que los servicios estén desplegados, podrás acceder a ellos utilizando la dirección IP o el nombre de host del servicio junto con el puerto especificado en el archivo de servicio YAML. Por ejemplo:

```
34.41.171.142:3002/name?name=denilson
```

Siguiendo estos pasos, podrás construir tus imágenes de Docker, empujarlas a Container Registry y desplegarlas en tu clúster de Kubernetes de manera efectiva.

Uso de YAML en Kubernetes

Despliegues

```
# Despliegue para el microservicio de edad
apiVersion: apps/v1
kind: Deployment
metadata:
  name: age-deployment
  labels:
    app: age
spec:
  replicas: 1
  selector:
    matchLabels:
      app: age
  template:
    metadata:
      labels:
        app: age
    spec:
      containers:
        - name: age-container
          image:
us-east1-docker.pkg.dev/carbon-ray-415302/sa/node_microservice_age_${ENVIRONME
NT}:latest
          ports:
            - containerPort: 3000

# Despliegue para el microservicio de género
apiVersion: apps/v1
kind: Deployment
metadata:
  name: gender-deployment
```

```
  labels:
    app: gender
spec:
  replicas: 1
  selector:
    matchLabels:
      app: gender
  template:
    metadata:
      labels:
        app: gender
    spec:
      containers:
        - name: gender-container
          image:
us-east1-docker.pkg.dev/carbon-ray-415302/sa/node_microservice_gender_${ENVIRO
NMENT}:latest
          ports:
            - containerPort: 3001

# Despliegue para el microservicio de nombre
apiVersion: apps/v1
kind: Deployment
metadata:
  name: name-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: name
  template:
    metadata:
      labels:
        app: name
    spec:
      containers:
        - name: name
          image:
us-east1-docker.pkg.dev/carbon-ray-415302/sa/node_microservice_name_${ENVIRONM
ENT}:latest
          ports:
```

```
            - containerPort: 3002
```

Servicios

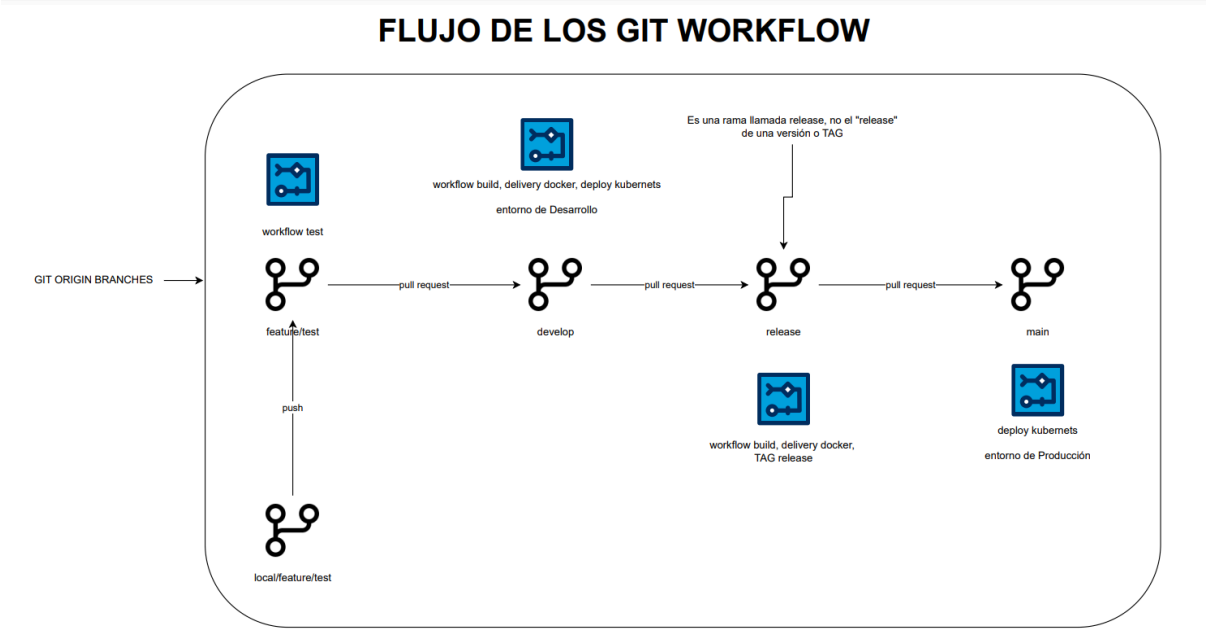
```
# Servicio para el microservicio de edad
apiVersion: v1
kind: Service
metadata:
  name: age-service
spec:
  selector:
    app: age
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000

# Servicio para el microservicio de género
apiVersion: v1
kind: Service
metadata:
  name: gender-service
spec:
  selector:
    app: gender
  ports:
    - protocol: TCP
      port: 3001
      targetPort: 3001

# Servicio para el microservicio de nombre
apiVersion: v1
kind: Service
metadata:
  name: name-service
spec:
  ports:
    - port: 3002
```

```
    targetPort: 3002
selector:
  app: name
type: LoadBalancer
```

Estos recursos de Kubernetes definen despliegues (Deployment) y servicios (Service) para tres microservicios diferentes: edad, género y nombre. Cada despliegue crea un conjunto de pods que ejecutan el contenedor del microservicio correspondiente, mientras que cada servicio expone esos pods para que puedan ser accedidos desde fuera del clúster de Kubernetes.

- **Despliegues (Deployment)**:
    - Cada despliegue especifica la imagen del contenedor que se usará, el número de réplicas (en este caso, 1), y los puertos que se expondrán dentro del contenedor.
    - Los despliegues aseguran que haya una cantidad específica de instancias del microservicio en funcionamiento en todo momento.
- **Servicios (Service):**
    - Cada servicio se encarga de enrutar el tráfico externo al conjunto de pods correspondiente.
    - Los servicios mapean un puerto en el clúster de Kubernetes al puerto expuesto por los pods de los despliegues.
    - En el caso del servicio de nombre, se especifica el tipo LoadBalancer, lo que hace que se proporciona un balanceador de carga para distribuir el tráfico externo entre los pods de nombre, si el entorno de Kubernetes lo soporta.

DEFINICIÓN DE CI/CD JOBS



FeatureCI.yaml - Feature/test

El archivo `featureCI.yaml` define el flujo de trabajo para la integración continua y entrega continua (CI/CD) de la característica "test". Aquí está la explicación detallada de cada sección:

Nombre del Flujo de Trabajo

```
name: SA_P3 CI
```

- Especifica el nombre del flujo de trabajo, que en este caso es "SA_P3 CI".

Evento que Dispara el Flujo de Trabajo

```
on:
  push:
    branches:
        - feature/test
```

- Define que este flujo de trabajo se ejecutará cuando haya un push en la rama "feature/test".

Definición de Jobs

```
jobs:
  build:
    runs-on: ubuntu-latest
```

- Define un job llamado "build" que se ejecutará en una máquina virtual con Ubuntu.

Estrategia de Matriz

```
strategy:
      matrix:
        node-version:
          - 21.7.0
```

- Define una matriz de versiones de Node.js para ejecutar el trabajo. En este caso, solo se ejecutará con la versión 21.7.0.

Pasos del Job

- **Clonar el Código:**

```
- name: Checkout and copy code to workflow
  uses: actions/checkout@v2
```

- Clona el repositorio de código en el flujo de trabajo.

Configurar Node.js:

```
- name: Configurar Node.js ${{ matrix.node-version }}
  uses: actions/setup-node@v2
  with:
    node-version: ${{ matrix.node-version }}
```

- Configura la versión específica de Node.js especificada en la matriz.

Instalar y Construir los Microservicios:

```
- name: Install && Build Microservice age
  run: |
    cd age_microservice/
    npm run build

- name: Install && Build Microservice gender
  run: |
    cd gender_microservice/
    npm run build

- name: Install && Build microservice name
  run: |
    cd name_microservice/
    npm run build
```

- Instala las dependencias y construye cada uno de los microservicios. Cada paso se ejecuta en su respectivo directorio.

Este archivo define un flujo de trabajo básico para construir y probar los microservicios de la característica "test" en un entorno de CI/CD. Cada vez que se realice un push en la rama "feature/test", se ejecutarán estos pasos para garantizar que los microservicios se construyan correctamente.

DevelopCI.yaml - Develop on Pull request

Nombre del Flujo de Trabajo

```
name: SA_P3 CI
```

- Define el nombre del flujo de trabajo como "SA_P3 CI".

Evento que Dispara el Flujo de Trabajo

```
on:
  pull_request:
    branches:
      - develop
```

- Establece que este flujo de trabajo se activará cuando se abra un pull request a la rama "develop".

Definición de Jobs

jobs:

```
build_stage:
   runs-on: ubuntu-latest
```

- Define un job llamado "build_stage" que se ejecutará en una máquina virtual con Ubuntu.

Estrategia de Matriz

```
    strategy:
      matrix:
        node-version:
          - 21.7.0
```

- Define una matriz de versiones de Node.js para ejecutar el trabajo. En este caso, solo se ejecutará con la versión 21.7.0.

Pasos del Job

- **Clonar el Código:**

```
- name: Checkout and copy code to workflow
  uses: actions/checkout@v2
```

-
  - Clona el repositorio de código en el flujo de trabajo.
- **Configurar Node.js:**

```
- name: Configure Node.js ${{ matrix.node-version }}
  uses: actions/setup-node@v2
  with:
    node-version: ${{ matrix.node-version }}
```

  - Configura la versión específica de Node.js especificada en la matriz.
- **Instalar y Construir los Microservicios:**

```
- name: Install and Build Microservice age
  run: |
    cd age_microservice/
    npm install
    npm run build

- name: Install and Build Microservice gender
  run: |
    cd gender_microservice/
    npm install
    npm run build

- name: Install and Build Microservice name
  run: |
    cd name_microservice/
    npm install
    npm run build
```

- Instala las dependencias y construye cada uno de los microservicios. Cada paso se ejecuta en su respectivo directorio.

Entorno de Desarrollo

- **Definir Variables de Entorno:**

```
env:
  REGION_REGISTRY: us-east1-docker.pkg.dev
  PROJECT_ID: carbon-ray-415302
  REPOSITORY: sa
  ENVIRONMENT: develop
```

- Define las variables de entorno necesarias, como la región del registro de contenedores, el ID del proyecto, el repositorio, y el entorno, que en este caso es "develop".

Sustituir el Nombre de la Imagen en la Configuración de Kubernetes:

```
- name: Replace image name in k8s config
  run: |
    ls
```

```
    for file in ./k8s/*.yaml; do
      envsubst '${ENVIRONMENT}' < "$file" > "${file%.yaml}.temp"
      mv "${file%.yaml}.temp" "$file"
    done
```

- Reemplaza el nombre de la imagen en los archivos de configuración de Kubernetes con la variable de entorno "ENVIRONMENT".

Construcción y Despliegue de Contenedores

- **Construcción y Envío de Contenedores a Artifact Registry:**

```
- name: Push Docker Image to Artifact Registry Microservice age
  env:
    GIT_TAG: ${{ steps.increment-git-tag.outputs.git-tag }}
    IMAGE_NAME_MS: node_microservice_age_${{ env.ENVIRONMENT }}
  run: |
    cd age_microservice/
    docker build -t $IMAGE_NAME_MS:latest .
    docker tag $IMAGE_NAME_MS:latest
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:latest
    docker tag $IMAGE_NAME_MS:latest
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:${{
steps.increment-git-tag.outputs.git-tag }}
    docker push $REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:latest
    docker push $REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:${{
steps.increment-git-tag.outputs.git-tag }}

- name: Push Docker Image to Artifact Registry Microservice gender
  env:
    GIT_TAG: ${{ steps.increment-git-tag.outputs.git-tag }}
    IMAGE_NAME_MS: node_microservice_gender_${{ env.ENVIRONMENT }}
  run: |
    cd gender_microservice/
    docker build -t $IMAGE_NAME_MS:latest .
    docker tag $IMAGE_NAME_MS:latest
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:latest
    docker tag $IMAGE_NAME_MS:latest
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:${{
steps.increment-git-tag.outputs.git-tag }}
    docker push $REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:latest
```

releaseCI.yaml - Release On pull request

Nombre del Flujo de Trabajo

```
name: SA_P3 CI
```

- Define el nombre del flujo de trabajo como "SA_P3 CI".

Evento que Dispara el Flujo de Trabajo

```
on:
  pull_request:
    branches:
      - release
```

- Establece que este flujo de trabajo se activará cuando se abra un pull request a la rama "release".

Definición de Jobs

```
jobs:
  build_stage:
    runs-on: ubuntu-latest
```

- Define un job llamado "build_stage" que se ejecutará en una máquina virtual con Ubuntu.

Estrategia de Matriz

```
strategy:
      matrix:
        node-version:
          - 21.7.0
```

- Define una matriz de versiones de Node.js para ejecutar el trabajo. En este caso, solo se ejecutará con la versión 21.7.0.

Pasos del Job

- **Clonar el Código:**

```
- name: Checkout and copy code to workflow
  uses: actions/checkout@v2
```

- Clona el repositorio de código en el flujo de trabajo.

- **Configurar Node.js:**

```
- name: Configure Node.js ${{ matrix.node-version }}
  uses: actions/setup-node@v2
  with:
    node-version: ${{ matrix.node-version }}
```

- Configura la versión específica de Node.js especificada en la matriz.

- **Instalar y Construir los Microservicios:**

```
- name: Install and Build Microservice age
  run: |
    cd age_microservice/
    npm install
    npm run build

- name: Install and Build Microservice gender
  run: |
    cd gender_microservice/
    npm install
    npm run build

- name: Install and Build Microservice name
  run: |
    cd name_microservice/
    npm install
    npm run build
```

- Instala las dependencias y construye cada uno de los microservicios. Cada paso se ejecuta en su respectivo directorio.

Entorno de Producción

- **Definir Variables de Entorno:**

```
env:
  REGION_REGISTRY: us-east1-docker.pkg.dev
```

```
  PROJECT_ID: carbon-ray-415302
  REPOSITORY: sa
  ENVIRONMENT: production
```

- Define las variables de entorno necesarias para el entorno de producción.

- **Configurar Docker con autenticación de Google Cloud**

```
- name: Configure Docker
  run: |
    gcloud auth configure-docker --quiet
    gcloud auth configure-docker us-east1-docker.pkg.dev --quiet
```

- Configura Docker para autenticarse con el registro de contenedores de Google Cloud.

- Construcción y Despliegue de Contenedores

  - **Construcción y Envío de Contenedores a Artifact Registry:**

    yaml

```
- name: Push Docker Image to Artifact Registry Microservice age
  env:
    GIT_TAG: ${{ steps.increment-git-tag.outputs.git-tag }}
    IMAGE_NAME_MS: node_microservice_age_${{ env.ENVIRONMENT }}
  run: |
    cd age_microservice/
    docker build -t $IMAGE_NAME_MS:latest .
    docker tag $IMAGE_NAME_MS:latest
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:latest
    docker tag $IMAGE_NAME_MS:latest
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:${{
steps.increment-git-tag.outputs.git-tag }}
    docker push
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:latest
    docker push
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:${{
steps.increment-git-tag.outputs.git-tag }}

- name: Push Docker Image to Artifact Registry Microservice gender
  env:
    GIT_TAG: ${{ steps.increment-git-tag.outputs.git-tag }}
    IMAGE_NAME_MS: node_microservice_gender_${{ env.ENVIRONMENT }}
  run: |
    cd gender_microservice/
```

```
    docker build -t $IMAGE_NAME_MS:latest .
    docker tag $IMAGE_NAME_MS:latest
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:latest
    docker tag $IMAGE_NAME_MS:latest
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:${{
steps.increment-git-tag.outputs.git-tag }}
    docker push
$REGION_REGISTRY/$PROJECT_ID/$REPOSITORY/$IMAGE_NAME_MS:latest
```

MainCI.yaml - Despliegue de Kubernetes

Nombre del Flujo de Trabajo

```
name: SA_P3 CI
```

- Define el nombre del flujo de trabajo como "SA_P3 CI".

Evento que Dispara el Flujo de Trabajo

```
on:
  pull_request:
    branches:
      - main
```

- Establece que este flujo de trabajo se activará cuando se abra un pull request a la rama "main".

Definición de Jobs

```
jobs:
  deploy_stage:
    runs-on: ubuntu-latest
```

- Define un job llamado "deploy_stage" que se ejecutará en una máquina virtual con Ubuntu.

Entorno de Producción

```
env:
```

```
      ENVIRONMENT: production
```

- Define la variable de entorno "ENVIRONMENT" con el valor "production".

Pasos del Job

- **Clonar el Código:**

```
- name: Checkout and copy code to workflow
  uses: actions/checkout@v2
```

- Clona el repositorio de código en el flujo de trabajo.

Configurar Entorno de Producción:

```
- name: Set environment variable
  run: echo "ENVIRONMENT=${{ env.ENVIRONMENT }}" >> $GITHUB_ENV
```

- Establece la variable de entorno "ENVIRONMENT" con el valor de producción.
- **Reemplazar el Nombre de la Imagen en la Configuración de Kubernetes:**

```
- name: Replace image name in k8s config
  run: |
    ls
    for file in ./k8s/*.yaml; do
      envsubst '${ENVIRONMENT}' < "$file" > "${file%.yaml}.temp"
      mv "${file%.yaml}.temp" "$file"
    done
```

- Reemplaza el nombre de la imagen en los archivos de configuración de Kubernetes con el valor del entorno de producción.

Despliegue en Kubernetes

- **Configurar Credenciales de Google Cloud Platform:**

```
- id: 'auth'
  uses: 'google-github-actions/auth@v2'
  with:
```

```
    credentials_json: '${{ secrets.GCP_CREDENTIALS }}'
```

- Configura las credenciales de Google Cloud Platform para autenticarse.

- **Configurar el SDK de Google Cloud:**

```
- name: 'Set up Cloud SDK'
  uses: 'google-github-actions/setup-gcloud@v2'
```

- Configura el SDK de Google Cloud para su uso en el flujo de trabajo.

- **Despliegue en el Clúster de GKE:**

```
- name: Deploy to GKE cluster
  uses: ameydev/gke-kubectl-action@master
  env:
    PROJECT_ID: carbon-ray-415302
    APPLICATION_CREDENTIALS: ${{ secrets.GCP_CREDENTIALS_BASE64 }}
    CLUSTER_NAME: cluster-production
    ZONE_NAME: us-central1-c
  with:
    args: apply -f k8s/
```

- Utiliza la acción `gke-kubectl-action` para desplegar los recursos en el clúster de Google Kubernetes Engine (GKE) utilizando los archivos de configuración en la carpeta `k8s`.

MANUAL DE INSTALACIÓN DE RUNNER SA_P3 CI

Descripción del Repositorio y Runner:

El repositorio SA_P3 CI contiene el código fuente de una aplicación de microservicios. Este manual te guiará a través de la instalación y configuración del runner asociado, que se encarga de automatizar la compilación, construcción y despliegue de los microservicios en entornos de desarrollo, pruebas y producción.

Requisitos Previos:

- Acceso a una cuenta de GitHub con privilegios de administrador en el repositorio.

- Credenciales válidas de Google Cloud Platform (GCP) si se despliega en un clúster de Kubernetes.
- Conocimientos básicos de Docker y Kubernetes.

Instalación del Runner:

1. Accede a la configuración del repositorio en GitHub.
2. Navega hasta la sección "Acciones" y selecciona "Configurar un runner propio".
3. Sigue las instrucciones proporcionadas para descargar, configurar y ejecutar el runner en tu entorno.

Configuración del Entorno:

- Este runner utiliza Node.js versión 21.7.0. Asegúrate de tener esta versión instalada en tu sistema.
- El archivo `GCP_CREDENTIALS` debe contener las credenciales de servicio de GCP para autenticación.

Configuración de Microservicios:

1. Clona el repositorio SA_P3 CI en tu entorno local.
2. Navega a cada uno de los microservicios (`age_microservice`, `gender_microservice`, `name_microservice`) y ejecuta `npm install` seguido de `npm run build`.

Despliegue a Kubernetes (K8s):

1. Configura las credenciales de GCP con el comando `gcloud auth activate-service-account --key-file=[KEY_FILE]`.

2. Ejecuta `gcloud container clusters get-credentials [CLUSTER_NAME] --zone [ZONE_NAME] --project [PROJECT_ID]` para configurar `kubectl` para tu clúster de Kubernetes.
3. Despliega las definiciones de recursos de Kubernetes ubicadas en la carpeta `k8s/` con el comando `kubectl apply -f [ARCHIVO_YAML]`.

Automatización:

- El runner automatiza la construcción de imágenes Docker, etiquetado de versiones y despliegue a través de GitHub Actions y scripts personalizados.

Pruebas:

- Después de cada despliegue, asegúrate de que los microservicios estén funcionando correctamente en su entorno respectivo (desarrollo, pruebas, producción).

Solución de Problemas:

- Si encuentras problemas durante la instalación o ejecución del runner, revisa los registros de GitHub Actions y verifica las configuraciones de GCP y Kubernetes.

Referencias:

- Documentación de GitHub Actions: https://docs.github.com/es/actions
- Documentación de Google Cloud Platform: https://cloud.google.com/docs