Tecnologías de Despliegue

Para el despliegue de la práctica, se optó por la combinación de Kubernetes y Github Actions por los siguientes motivos:

Kubernetes

Es una plataforma de orquestación de contenedores e imágenes que facilita la gestión, escalado y despliegue de aplicaciones en contenedores. Los microservicios de la práctica se construyeron individualmente en imágenes docker, por lo que hacer uso de Kubernetes era una decisión óptima y sencilla de aplicar puesto que ya existía un precedente de su uso en la práctica anterior.

De igual forma, Kubernetes es compatible con muchos proveedores de nube y entornos on-premise, lo que lo hace aún mejor para futuras aplicaciones y despliegues en diferentes entornos.

Por último, se continuó con Kubernetes por el escalado automático de los recursos, garantizando así optimización y disponibilidad de nuestras aplicaciones.

Github Actions

¿Por qué no? Si ya se utilizaba un repositorio de Github para guardar el código, era fácil decantarse por esta opción. Github Actions permite automatizar el flujo de trabajo de desarrollo, pruebas y despliegue en producción de los microservicios, permitiendo trabajar el CI y el CD de manera sencilla, mejorando la eficiencia en el desarrollo y reduciendo el tiempo de lanzamiento a producción de los microservicios.

Además, Actions es de fácil uso y configuración dentro de Github, además de no necesitar instalar programas externos o “runners” para poder utilizar el CI/CD, solamente con un par de configuraciones tendremos ya corriendo nuestros workflows.

Gestión de Aplicaciones

¿Cómo hacer un despliegue de aplicaciones en Kubernetes utilizando GIthub Actions?

Para hacer un despliegue sencillo utilizando Kubernetes y Github Actions, es primordial tener un repositorio Github para interactuar con ambas herramientas, además de poder llevar el versionamiento de nuestro proyecto.

También se necesita un clúster configurado de Kubernetes, en este caso utilizando GKE (Google Kubernetes Engine), el cual es un servicio de Kubernetes administrado por Google.

Para hacer el despliegue se hace uso de Github Actions, donde se crea un workflow dentro de la carpeta **".github/workflows"** dentro del repositorio, en la carpeta raíz.

Para configurar los despliegues se utilizaron archivos de extensión **".yml"**.

Uso de YAML

Deploy de ambiente de producción

Configuración

```yaml
name: production build and deploy
on:
  push:
    branches:
      - "main"
env:
  PROJECT_ID: ${{ secrets.GKE_PROJECT }}
  GKE_CLUSTER: cluster-1 # cluster name
  GKE_ZONE: us-central1-c # cluster zone
```

Job: push-images

```yaml
jobs:
  push-images:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      # Log in to Dockerhub
      - name: Log in to Dockerhub
        uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
        with:
          username: ${{ secrets.DOCKERHUB_USER }}
          password: ${{ secrets.DOCKERHUB_PASSWORD }}
      # Build and push the Docker image
      - name: Build and push docker image agify
        env:
          DOCKER_BUILDKIT: 1
        run: |
          cd practica3
          docker build -t epum/agify:latest -f agify.dockerfile .
          docker push epum/agify:latest
      # Build and push the Docker image
      - name: Build and push docker image genderize
        env:
          DOCKER_BUILDKIT: 1
        run: |
          cd practica3
          docker build -t epum/genderize:latest -f genderize.dockerfile .
          docker push epum/genderize:latest
      # Build and push the Docker image
      - name: Build and push docker image core
        env:
          DOCKER_BUILDKIT: 1
        run: |
          cd practica3
          docker build -t epum/main:latest -f main.dockerfile .
          docker push epum/main:latest
```

Descripción: Este job se encarga de hacer build y push de las imágenes Docker de las aplicaciones

Steps:

1. Checkout: Clona el repositorio
2. Iniciar Sesión en Dockerhub
3. Construir y subir la imagen de Docker de Agify
4. Construir y subir la imagen de Docker de Genderize
5. Construir y subir la imagen de Docker de Main.

Job: deploy

```yaml
deploy:
  needs: [push-images]
  runs-on: ubuntu-latest
  steps:
    - name: Checkout
      uses: actions/checkout@v4
    # Setup gcloud CLI
    - uses: google-github-actions/auth@v2
      with:
        credentials_json: ${{secrets.GKE_SA_KEY_JSON}}
    # Configure Docker to use the gcloud command-line tool as a credential
    # helper for authentication
    - run: |-
        gcloud --quiet auth configure-docker
    # Get the GKE credentials so we can deploy to the cluster
    - uses: google-github-actions/get-gke-credentials@db150f2cc60d1716e61922b832eae71d2a45938f
      with:
        project_id: ${{ env.PROJECT_ID }}
        cluster_name: ${{ env.GKE_CLUSTER }}
        location: ${{ env.GKE_ZONE }}
    # Apply the Kubernetes manifest for deployment
    - name: Deploy agify, genderize and main
      run: |
        cd practica3

        kubectl delete deployments --all -n microservices-201700761

        kubectl apply -f agify.yaml
        kubectl apply -f genderize.yaml
        kubectl apply -f main.yaml
```

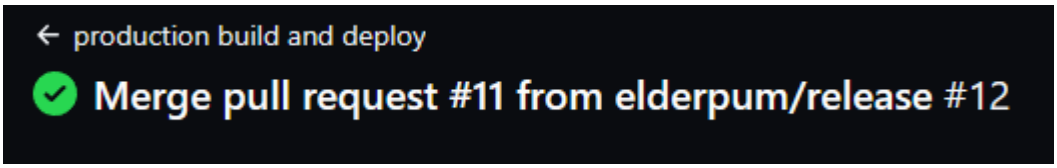
Descripción: Este job se encarga de desplegar las aplicaciones en el clúster de Kubernetes.

Dependencias: Depende del job de push-images

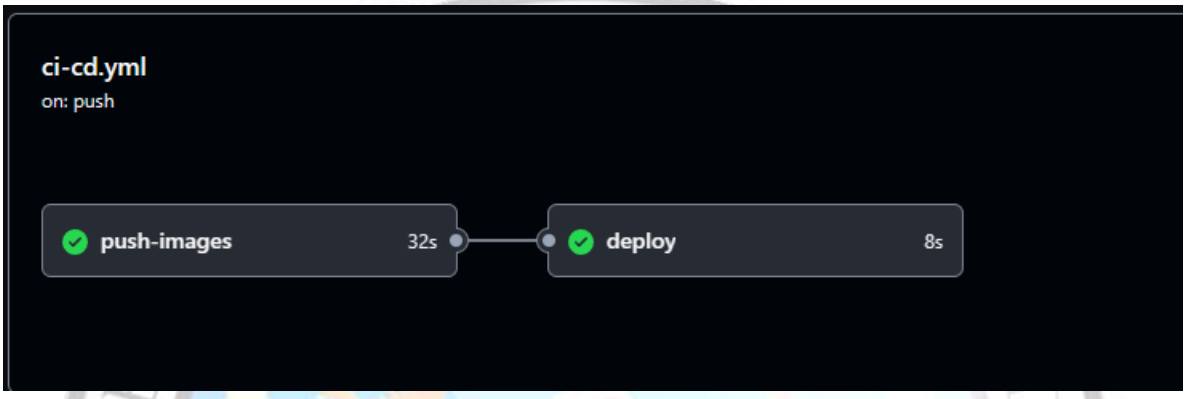
Steps:

1. Checkout: Clona el repositorio.
2. Configurar la CLI de gcloud con las credenciales
3. Configurar Docker para usar gcloud como helper de autenticación
4. Obtener credenciales de GKE
5. Desplegar agify, genderize y main

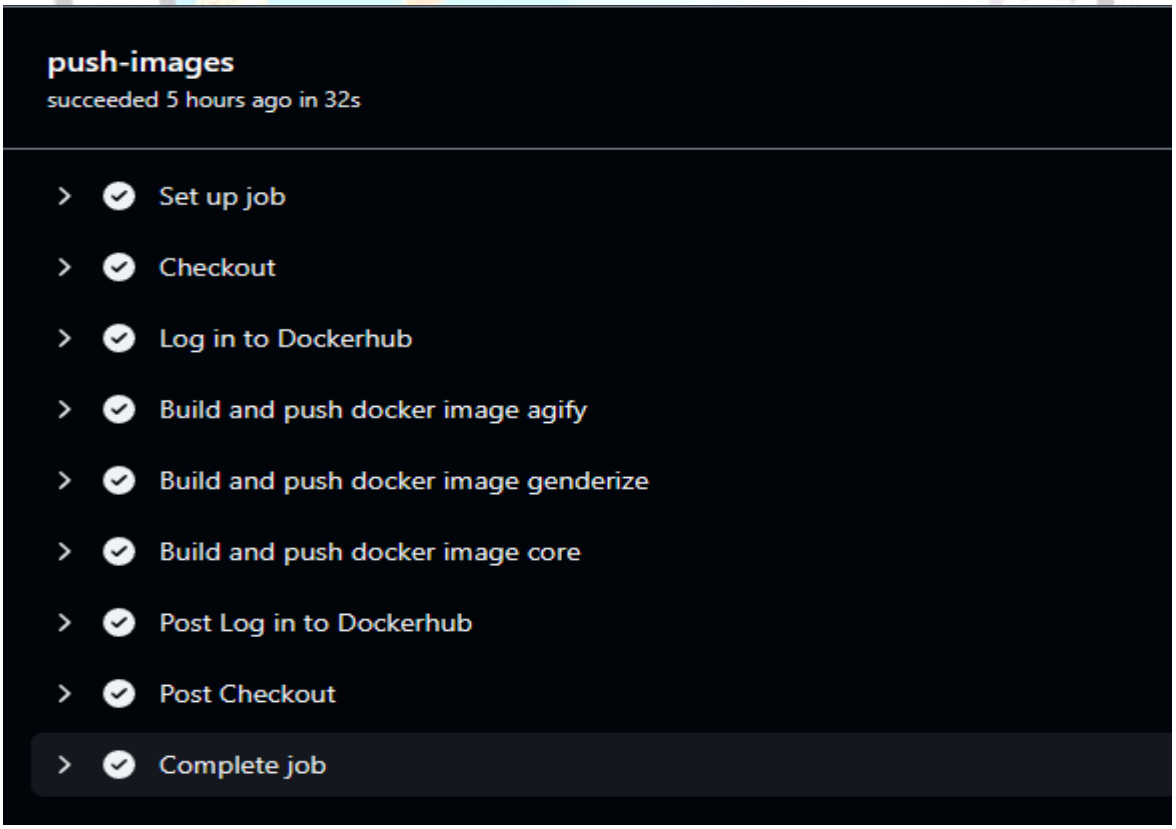
El pipeline se ejecuta en Github Actions:



Los Jobs se ejecutan de forma secuencial:



En cada Jobs se ejecutan los Steps:



Definición de CI/CD

El pipeline *feature.yml* se ejecuta al momento de empujar en la rama feature.

Stages y Jobs

1. **Stage 1:** Build and Test
    a. **Job 1:** build and test– Este job compila las imágenes Docker para los servicios “agify”, “genderize” y “main” desde el código fuente presente en las respectivas carpetas del repositorio. Cada job utiliza Docker para construir una imagen basada en Dockerfile presente en la práctica.

```yaml
jobs:
  build-and-test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2

      # Paso de construcción
      - name: Build
        run: |
          cd practica3
          docker build -t epum/agify:latest -f agify.dockerfile .
          docker build -t epum/genderize:latest -f genderize.dockerfile .
          docker build -t epum/main:latest -f main.dockerfile .
          echo "All images built successfully!"

      # Paso de prueba (puedes agregar tus propios pasos de prueba aquí)
      - name: Test
        run: |
          echo "Running tests..."
          echo "All tests passed successfully!"
```

El pipeline *release.yml* se ejecuta al momento de empujar en la rama release.

Stages y Jobs

1. **Stage 1:** Release-Delivery
    a. **Job 1:** release-delivery – Este job se encarga de compilar cada imagen de Docker mediante su correspondiente archivo Dockerfile, hacer testing de nuestros archivos, después se encarga de crear una tag nueva para el repositorio de Github y por último, se encarga de hacer un Push de nuestras imágenes compiladas hacia Dockerhub con el objetivo de poderlas utilizar después.

```yaml
jobs:
  release-delivery:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      # Log in to Dockerhub
      - name: Log in to Dockerhub
        uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
        with:
          username: ${{ secrets.DOCKERHUB_USER }}
          password: ${{ secrets.DOCKERHUB_PASSWORD }}

      # Paso de construcción
      - name: Build
        run: |
          cd practica3
          docker build -t epum/agify:latest -f agify.dockerfile .
          docker build -t epum/genderize:latest -f genderize.dockerfile .
          docker build -t epum/main:latest -f main.dockerfile .
          echo "All images built successfully!"

      # Paso de prueba (puedes agregar tus propios pasos de prueba aquí)
      - name: Test
        run: |
          echo "Running tests..."
          echo "All tests passed successfully!"

      # Crear y empujar etiquetas de versión
      - name: Create and push version tags
        run: |
          VERSION=$(date +%Y%m%d)-$(git rev-parse --short HEAD)
          echo "Version: $VERSION"
          git tag $VERSION
          git push origin $VERSION

      # Paso de entrega
      - name: Deliver
        run: |
          cd practica3
          docker push epum/agify:latest
          docker push epum/genderize:latest
          docker push epum/main:latest
          echo "Images delivered successfully!"
```

El pipeline *ci/cd.yml* se ejecuta al empujar en la rama main.

Stages y Jobs

1. **Stage 1:** push-images:
    a. **Job 1:** push-images – Este trabajo realiza acciones similares al pipeline de release. Construye las imágenes Docker para los servicios de agify, genderize y main, las etiqueta y empuja a DockerHub con su respectiva etiqueta.

```yaml
jobs:
  push-images:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      # Log in to Dockerhub
      - name: Log in to Dockerhub
        uses: docker/login-action@f4ef78c080cd8ba55a85445d5b36e214a81df20a
        with:
          username: ${{ secrets.DOCKERHUB_USER }}
          password: ${{ secrets.DOCKERHUB_PASSWORD }}
      # Build and push the Docker image
      - name: Build and push docker image agify
        env:
          DOCKER_BUILDKIT: 1
        run: |
          cd practica3
          docker build -t epum/agify:latest -f agify.dockerfile .
          docker push epum/agify:latest
      # Build and push the Docker image
      - name: Build and push docker image genderize
        env:
          DOCKER_BUILDKIT: 1
        run: |
          cd practica3
          docker build -t epum/genderize:latest -f genderize.dockerfile .
          docker push epum/genderize:latest
      # Build and push the Docker image
      - name: Build and push docker image core
        env:
          DOCKER_BUILDKIT: 1
        run: |
          cd practica3
          docker build -t epum/main:latest -f main.dockerfile .
          docker push epum/main:latest
```

2. **Stage 2:** deploy:
    a. **Job 2:** deploy – Realiza la conexión al clúster de Kubernetes ubicado en GKE haciendo uso de las credenciales (guardadas como Secrets en el repositorio) para finalmente, eliminar los deployments existentes dentro de nuestro clúster y actualizando con las imágenes

previamente buildeadas nuestro clúster, desplegando así finalmente nuestra aplicación en producción.

```yaml
deploy:
  needs: [push-images]
  runs-on: ubuntu-latest
  steps:
    - name: Checkout
      uses: actions/checkout@v4
    # Setup gcloud CLI
    - uses: google-github-actions/auth@v2
      with:
        credentials_json: ${{secrets.GKE_SA_KEY_JSON}}
    # Configure Docker to use the gcloud command-line tool as a credential
    # helper for authentication
    - run: |-
        gcloud --quiet auth configure-docker
    # Get the GKE credentials so we can deploy to the cluster
    - uses: google-github-actions/get-gke-credentials@db150f2cc60d1716e61922b832eae71d2a45938f
      with:
        project_id: ${{ env.PROJECT_ID }}
        cluster_name: ${{ env.GKE_CLUSTER }}
        location: ${{ env.GKE_ZONE }}
    # Apply the Kubernetes manifest for deployment
    - name: Deploy agify, genderize and main
      run: |
        cd practica3

        kubectl delete deployments --all -n microservices-201700761

        kubectl apply -f agify.yaml
        kubectl apply -f genderize.yaml
        kubectl apply -f main.yaml
```

Manual de Instalación del Runner

Introducción

Se necesita un repositorio de GitHub para crear y ejecutar un flujo de trabajo de GitHub Actions. En esta guía, se explicará cómo agregar un flujo de trabajo que demuestre alguna de las características esenciales de las GitHub Actions.

A continuación, se muestra cómo los jobs de GitHub Actions pueden activarse automáticamente, dónde se ejecutan y cómo pueden interactuar con el código del repositorio.

Crear el primer flujo de trabajo en Actions

Para comenzar, se debe clonar el repositorio remoto en algún ambiente local, esto para tener una mayor comodidad al momento de trabajar con los pipelines.

Se debe crear un directorio .github/workflows en el repositorio de GitHub si todavía no existe. Para que GitHub detecte los flujos de trabajo, este directorio debe tener ese nombre exacto.

En el directorio anteriormente creado, se debe crear un archivo con la extensión .yml o .yaml. Para este manual, el archivo a crear se llamará: github-actions-demo.yml.

En el anterior archivo, se debe pegar lo siguiente:

```yaml
name: GitHub Actions Demo
run-name: ${{ github.actor }} is testing out GitHub Actions 🚀
on: [push]
jobs:
  Explore-GitHub-Actions:
    runs-on: ubuntu-latest
    steps:
      - name: List files in the repository
        run: |
          ls ${{ github.workspace }}
      - run: echo "🍏 This job's status is ${{ job.status }}."
```

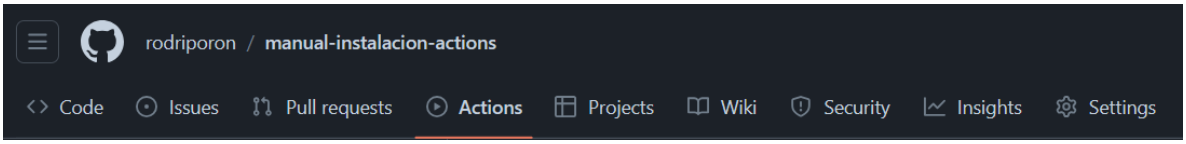
Este archivo el cual define un flujo de trabajo que realiza lo siguiente cuando se realiza algún push en el repositorio actual:

Ejecuta un trabajo que lista los archivos actuales en el repositorio, también lanza una salida en consola el cual indica el status del presente job.

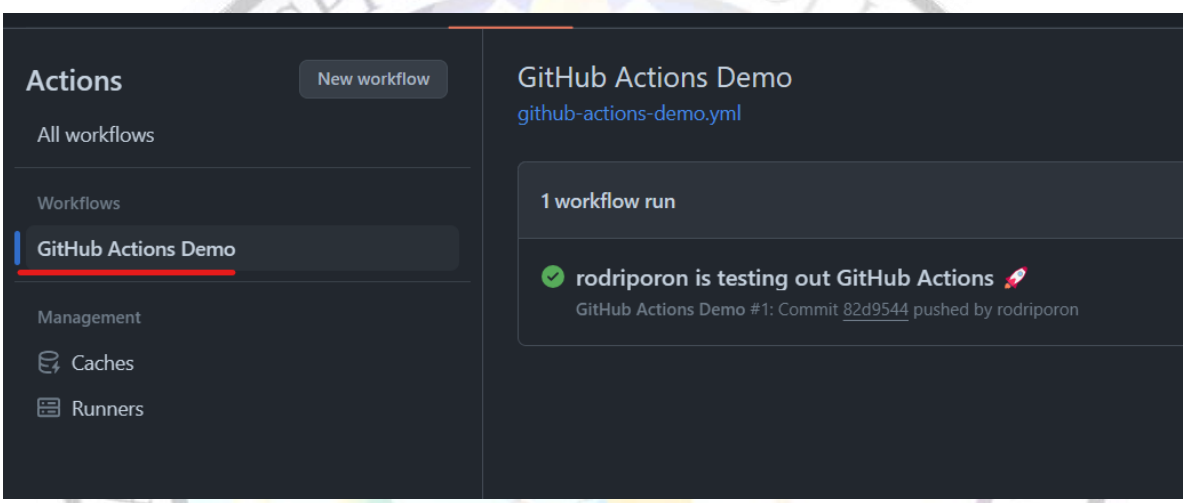
Para aplicar estos cambios, se debe crear un commit en el repositorio local y realizar un push al repositorio remoto para que el flujo de trabajo se pueda ejecutar.

Ver los Resultados del flujo de trabajo

En GitHub.com se debe navegar a la página principal del repositorio, página en la cual se podrá visualizar lo siguiente:



En la pestaña Actions se podrá ver el flujo de trabajo creado anteriormente de la siguiente forma:



En el cual se podrá ver todos los pasos ejecutados por el Job Explore-GitHub-Actions: