FaaSioFlex: Instalación del Entorno de Trabajo en una Máquina Virtual con Ubuntu

Chunlu Hu, Juan Amilcar Tito Tito, Carlos Caramagna, José Javier Gutiérrez Gil

Fecha: 24 de octubre de 2024

Índice

1. Preparación de la Máquina Virtual

Si no se tiene una maquina con ubuntu, lo ideal es montar una maquina virtual donde instalar el entorno de desarrollo con un sistema operativo linux, en nuestro caso:

Campo	**Valor**
Static hostname:	Ubuntu-Dockers
Icon name:	computer-laptop
Chassis:	laptop
Machine ID:	abcd1234efgh5678ijkl9012mnopqr
Boot ID:	1234abcd5678efgh9012ijklmnopqr
Operating System:	Ubuntu 22.04.2 LTS
Kernel:	Linux 6.8.0-47-generic
Architecture:	x86_64

Cuadro 1: Información del sistema utilizando laptop con Ubuntu

Campo	**Valor**
Static hostname:	Ubuntu-Dockers
Icon name:	computer-vm
Chassis:	vm
Machine ID:	abcd1234efgh5678ijkl9012mnopqr
Boot ID:	1234abcd5678efgh9012ijklmnopqr
Operating System:	Ubuntu 22.04.2 LTS
Kernel:	Linux 6.8.0-47-generic
Architecture:	x86_64

Cuadro 2: Información del sistema utilizando mv

- **Crear la Máquina Virtual (VM)**: Utiliza un hipervisor como VirtualBox o VMware.
- **Asignación de Recursos**: Asigna al menos 2 GB de RAM, 2 CPUs, y 20 GB de espacio en disco (estos valores pueden ajustarse según los requisitos del proyecto).
- **Descargar Ubuntu**: Descarga la última versión LTS de Ubuntu desde `https://ubuntu.com/download/desktop` (imagen ISO).

2. Instalación del Sistema Operativo

Sigue los pasos del asistente de instalación de Ubuntu y selecciona las opciones mínimas si se va a configurar el entorno manualmente.

3. Configuración Inicial del Sistema

- **Actualizar el Sistema**:

```
sudo apt update && sudo apt upgrade -y
```

- **Instalar Utilidades Básicas**:

```
sudo apt install -y build-essential curl git wget
```

- **Instalar Lenguajes de Programación**:
  - **Instalar Go**:

```
sudo apt install -y golang
```

  - **Node.js y npm**:

```
curl -fsSL https://deb.nodesource.com/setup_18.x |
    sudo -E bash -
sudo apt install -y nodejs
```

  - **Python y pip**:

```
sudo apt install -y python3 python3-pip
```

  - **Instalar TypeScript**:

```

sudo npm install -g typescript
```

- Para verificar la instalación de typescript:

```
        tsc -v
```

- **Instalar Visual Studio Code Visual Studio Code usando Snap**:

```
    #Verifiar canales snap
    snap info core
    # Instalar Visual Studio Code
sudo snap install --classic code
```

- **Instalar Visual Studio Code Visual Studio Code desde Microsfot**:

```
    #Agregar la clave GPG de Microsoft
    wget -qO- https://packages.microsoft.com/keys/
        microsoft.asc | sudo apt-key add -

    # Agregar el repositorio de Visual Studio Code
sudo add-apt-repository "deb [arch=amd64] https://
    packages.microsoft.com/repos/vscode stable main"
    # Actualizar los repositorios
    sudo apt update
    # Instalar Visual Studio Code
    sudo apt install code
```

- **Verificar dependencias**:

```
     sudo apt-get install -f
```

4. Opciones de Desarrollo

Desarrollar directamente en Ubuntu tiene varias ventajas, ya que estarás trabajando en el mismo entorno que utilizarás para la ejecución y despliegue de tu aplicación. Sin embargo, si tu sistema operativo principal es Windows, es recomendable crear una máquina virtual (MV) con Ubuntu. Este enfoque no solo te permite trabajar en un entorno nativo de Linux, sino que también asegura que la configuración de tu entorno de desarrollo coincida con el de producción, minimizando problemas de compatibilidad.

Para configurar esta máquina virtual, puedes usar herramientas como VirtualBox o VMware. Una vez que hayas instalado Ubuntu en la máquina virtual, sigue estos pasos:

- **Instalación del Entorno de Desarrollo**: Comienza instalando todas las herramientas necesarias, como Visual Studio Code, Git y, por supuesto, Docker. Puedes utilizar el terminal de Ubuntu para instalar Docker mediante los comandos correspondientes, asegurándote de seguir las recomendaciones de la documentación oficial de Docker.

- **Configuración de Docker**: Docker es esencial para la ejecución de contenedores que alojarán tus aplicaciones. Instalar Docker en tu máquina virtual te permitirá crear, ejecutar y gestionar contenedores de manera eficiente. También es importante instalar Docker Compose, que te ayudará a definir y ejecutar aplicaciones multicontenedor.

- **Integración con GitHub**: Es crucial que tu entorno de desarrollo esté integrado con un repositorio en GitHub. Esto permite mantener un control de versiones efectivo y colaborar con otros desarrolladores. Al hacer cambios en tu código, puedes hacer un commit y push a tu repositorio, lo que desencadenará la compilación y el despliegue automático de tus imágenes Docker en Docker Hub. Posteriormente, puedes descargar estas imágenes en tu máquina virtual usando comandos de Docker.

- **Pruebas y Despliegue**: Una vez que tu código está listo, puedes ejecutar los contenedores Docker en Ubuntu, facilitando así pruebas rápidas y ajustes en tu aplicación. Esto asegura que cualquier problema se resuelva en un entorno que simula la producción, reduciendo el riesgo de fallos al desplegar en un entorno real.

Además, al utilizar Visual Studio Code en tu máquina virtual Ubuntu, puedes aprovechar su amplia gama de extensiones, como la extensión Remote - SSH, que te permite trabajar de forma remota y editar archivos en la máquina virtual como si estuvieras trabajando localmente. Esto simplifica enormemente el flujo de trabajo, permitiendo que los desarrolladores se concentren en el desarrollo de aplicaciones sin preocuparse por las diferencias entre sistemas operativos.

el esquema de pasos a seguir con esta opción será: crear una máquina virtual con Ubuntu en tu sistema Windows y configurar adecuadamente tu entorno de

desarrollo no solo es una estrategia efectiva para asegurar la compatibilidad, sino que también optimiza el proceso de desarrollo y despliegue.

4.1. Desarrollo desde Windows

Si prefieres trabajar en tu máquina local con Windows, puedes utilizar Visual Studio Code como tu entorno de desarrollo integrado (IDE). Sin embargo, es fundamental que instales Docker Desktop para crear un entorno local que te permita desplegar y ejecutar contenedores Docker de manera eficiente.

4.1.1. Instalación de Docker Desktop

Docker Desktop es una aplicación que proporciona una interfaz gráfica y herramientas que facilitan la creación, gestión y ejecución de contenedores Docker. Esta herramienta es especialmente útil en sistemas operativos como Windows y macOS, ya que simplifica la complejidad de trabajar con Docker. Docker Desktop incluye características como:

- **Interfaz Gráfica de Usuario (GUI)**: Permite a los desarrolladores gestionar contenedores, imágenes y redes de forma visual, lo que hace que la administración sea más intuitiva.
- **Docker Compose**: Facilita la orquestación de múltiples contenedores mediante la definición de servicios en un archivo `docker-compose.yml`.
- **Integración con WSL 2 (Windows Subsystem for Linux)**: Proporciona una mejor experiencia de ejecución al permitir que los contenedores se ejecuten en un entorno Linux.

Para instalar Docker Desktop, simplemente visita el sitio oficial de Docker y sigue las instrucciones de instalación. Una vez instalado, podrás ejecutar contenedores en tu máquina local, lo que permitirá pruebas rápidas y ajustes en tu código sin necesidad de un entorno de producción.

4.1.2. Integración con GitHub

La integración del desarrollo con un repositorio en GitHub es esencial para mantener un flujo de trabajo organizado y eficiente. Cada vez que se realice un commit en el repositorio del proyecto, se ejecutará un proceso automatizado que compilará el código, creará nuevas imágenes Docker y las subirá a Docker Hub. El flujo de trabajo será el siguiente:

- Los desarrolladores realizan cambios en el código en su entorno local utilizando Visual Studio Code.
- Al realizar un commit en GitHub, un proceso automatizado (como GitHub Actions) se activa y compila el código.

- Se genera una nueva imagen Docker que se sube automáticamente a Docker Hub.
- Los desarrolladores pueden usar comandos de Docker para descargar la imagen actualizada desde Docker Hub a su entorno local.
- Finalmente, ejecutan la imagen descargada utilizando Docker Desktop para verificar que los cambios funcionan correctamente.

4.2. Transferencia de Archivos a la Máquina Virtual en Ubuntu

Si necesitas transferir archivos desde tu máquina local con Windows a un entorno de ejecución en una máquina virtual con Ubuntu, existen varias herramientas que pueden facilitar este proceso:

- **SSH/SFTP**: Puedes utilizar un cliente como WinSCP para transferir tus archivos a la máquina virtual a través de SSH. WinSCP proporciona una interfaz gráfica fácil de usar que te permite arrastrar y soltar archivos entre tu máquina local y la máquina virtual.
- **Remote - SSH en VS Code**: Esta extensión te permite conectar directamente a tu máquina virtual desde Visual Studio Code en Windows. Con esta opción, puedes editar archivos remotos como si estuvieras trabajando localmente, evitando el proceso manual de transferencia de archivos. Simplemente instala la extensión `Remote - SSH` desde la tienda de extensiones de Visual Studio Code y configura la conexión a tu máquina virtual.

Es importante asegurarte de que el código que desarrollas en Windows sea compatible con el entorno de Ubuntu, especialmente en lo que respecta a las versiones de software y dependencias. Esto ayudará a prevenir problemas de compatibilidad al desplegar tus aplicaciones en el entorno de producción.

5. Instalación de Herramientas Específicas del Entorno

- **Instalar Docker**:

```
# Instalar paquetes necesarios
sudo apt install -y apt-transport-https ca-certificates
     curl software-properties-common

# Agregar la clave GPG oficial de Docker
curl -fsSL https://download.docker.com/linux/ubuntu/gpg
     | sudo apt-key add -

# Agregar el repositorio de Docker
```

```
sudo add-apt-repository "deb [arch=amd64] https://
    download.docker.com/linux/ubuntu $(lsb_release -cs)
    stable"

# Actualizar los repositorios
sudo apt update

# Instalar Docker
sudo apt install -y docker-ce

# Habilitar e iniciar el servicio Docker
sudo systemctl enable --now docker

# Agregar tu usuario al grupo de Docker para ejecutar
    sin sudo
sudo usermod -aG docker $USER

# Reiniciar la terminal para aplicar los cambios
# Verificar la instalaci n
docker --version
```

Instalar Docker Compose:

```
sudo curl -L "https://github.com/docker/compose/
    releases/download/$(curl -s https://api.github.com/
    repos/docker/compose/releases/latest | grep -Po '"
    tag_name": "\K.*\d')/docker-compose-$(uname -s)-$(
    uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Verificar la instalaci n de Docker Compose
docker-compose --version
```

Instalar Kimori Docker: Sólo cuando el servicio funcione perfecamente en local.

```
# Agregar el repositorio de Kimori
curl -fsSL https://apt.kimori.io/kimori.key | sudo apt-
    key add -
sudo add-apt-repository "deb [arch=amd64] https://apt.
    kimori.io/ubuntu $(lsb_release -cs) stable"

# Actualizar los repositorios
sudo apt update

# Instalar Kimori
sudo apt install -y kimori

# Verificar la instalaci n de Kimori
kimori --version

    \subitem \textbf{Instalar kubectl (opcional, para
        gestionar cl steres)}:
    \begin{lstlisting}
# Instalar kubectl
sudo apt install -y kubectl
```

```
# Verificar la instalaci n de kubectl
kubectl version --client
```

Instalar Minikube (opcional, para entornos de desarrollo de Kubernetes):

```
# Descargar e instalar Minikube
curl -Lo minikube https://storage.googleapis.com/
    minikube/releases/latest/minikube-linux-amd64
sudo install minikube /usr/local/bin/

# Verificar la instalaci n de Minikube
minikube version
```

Instalar herramientas de monitoreo (opcional):

```
# Instalar Prometheus y Grafana para monitoreo (si es
    necesario)
# Este paso se debe ajustar seg n tus preferencias de
    monitoreo.
sudo apt install -y prometheus grafana

# Verificar la instalaci n de Prometheus y Grafana
prometheus --version
grafana-cli --version
```

6. Configuración de Redes y Acceso Remoto

- **Instalar y configurar OpenSSH para el acceso remoto**:

```
sudo apt install -y openssh-server
sudo systemctl enable --now ssh
```

- **Configuración de Firewall (UFW)**:

```
sudo ufw allow OpenSSH
sudo ufw enable
```

7. Clonar Repositorios de Proyectos y Configurar el Entorno de Trabajo

- **Ver Anexo A. Uso repositorio GitHub**:
- **Instalar Dependencias del Proyecto**: Según el proyecto, instala las dependencias:
  - **Go**: Ejecuta `go mod tidy` para instalar las dependencias.
  - **Node.js**: Ejecuta `npm install`.
  - **Docker Compose**: Ejecuta `docker-compose up -d`.

8. Guía para Desplegar una Aplicación en Google Cloud Run desde GitHub Actions

Esta guía explica cómo desplegar una aplicación 'Hola Mundo' escrita en Go en Google Cloud Run utilizando GitHub Actions. La configuración incluye el uso de Docker Hub para almacenar la imagen Docker, y Google Cloud para ejecutar la aplicación. Es un ejemnplo base de como ejecutar una app en Go sencilla sin Dex, nats ni apisix.Dichos módulos se integraran paso a paso en los subsiguientes ejemplos base.

8.1. Requisitos Previos

- Tener una cuenta de GitHub y un repositorio con el nombre `FaaSioFlex` que contenga el código fuente de la aplicación Go.
- Tener una cuenta de Docker Hub y haber creado un repositorio para almacenar la imagen.
- Tener una cuenta de Google Cloud y haber habilitado Google Cloud Run y el Container Registry.

8.2. A) Configuración de Google Cloud

8.2.1. Crear una Cuenta en Google Cloud

1. Regístrate en Google Cloud: `https://cloud.google.com/`.
2. Si es la primera vez, Google ofrece créditos gratuitos para nuevos usuarios. (Get $300 in free credits and free usage of 20+ products

8.2.2. Crear un Proyecto en Google Cloud

1. Accede a la Consola de Google Cloud: `https://console.cloud.google.com/`.
2. Crea un nuevo proyecto llamado `FaaSioFlex`.
3. Selecciona "Nuevo proyectoz asigna un nombre, por ejemplo, "FaaSioFlex".

8.2.3. Habilitar Servicios en Google Cloud

1. Habilita Google Cloud Run: `https://console.cloud.google.com/run`.
2. Habilita el Container Registry para almacenar imágenes Docker.

Configuración del Servicio

Un servicio expone un endpoint único y escala automáticamente la infraestructura subyacente para manejar las solicitudes entrantes.

Opciones de Implementación

- **Docker Hub**
  - Desplegar una revisión desde una imagen de contenedor existente
- **GitHub**
  - Desplegar continuamente desde un repositorio (fuente o función)
- **Funciones**
  - Usar un editor en línea para crear una vista previa de una función

Configuración de la Imagen de Contenedor

- **URL de la imagen de contenedor:**

  Se requiere la URL de la imagen de contenedor. Prueba un ejemplo: `us-docker.pkg.dev/cloudrun/container/hello`

- **¿Cómo construir un contenedor?**

Configuración del Servicio

- **Nombre del servicio:** `FaaSioFlex`
  El nombre del servicio debe comenzar con una letra y contener hasta 49 letras minúsculas, números o guiones.
- **URL del endpoint:**

  `https://FaaSioFlex-1077654616796.region.run.app`

- **Autenticación:**
  - **Permitir invocaciones no autenticadas:** Marca esta opción si estás creando una API pública o un sitio web.
  - **Requerir autenticación:** Gestiona usuarios autorizados con Cloud IAM.
- **Asignación de CPU y precios:**
  - La CPU solo se asigna durante el procesamiento de solicitudes. Se te cobrará por solicitud y solo cuando la instancia del contenedor procese una solicitud.
  - La CPU siempre está asignada. Se te cobrará por todo el ciclo de vida de la instancia del contenedor.
- **Escalado automático del servicio:**
  - Número mínimo de instancias: `0`
    Establecer en 1 para reducir los arranques en frío. Aprende más.

- **Control de ingreso:**
  - **Interno:** Permitir tráfico desde tu proyecto, VPC compartido y perímetro de controles de servicio de VPC. El tráfico desde otro servicio de Cloud Run debe ser enrutado a través de una VPC. Se aplican limitaciones. Aprende más.
  - **Todo:** Permitir acceso directo a tu servicio desde Internet.

8.2.4. Configurar la Cuenta de Servicio

1. Ve a "IAM & Admin >Cuentas de servicio" en la consola de Google Cloud.
2. Crea una cuenta de servicio llamada `faasioflex-deploy`.
3. Asigna el rol de "Editor" o "Administrador de Cloud Run".
4. Crea una clave JSON para la cuenta de servicio y descarga el archivo.

8.3. Configurar Docker Hub

1. Crea un repositorio en Docker Hub con el nombre `my-go-app`.
2. Genera un token de acceso en la configuración de tu cuenta de Docker Hub.
3. Añade las credenciales de Docker Hub como secretos en GitHub:
   - Ve a "Settings >Secrets and variables >Actions" en tu repositorio `FaaSioFlex`.
   - Añade dos nuevos secretos:
     - `DOCKER_USERNAME`: Tu nombre de usuario en Docker Hub.
     - `DOCKER_PASSWORD`: El token de acceso generado.

8.4. Configuración de GitHub Actions

8.5. Archivo `.github/workflows/deploy.yml`

Crea el archivo `deploy.yml` en el directorio `.github/workflows/` con el siguiente contenido:

```
name: Deploy to Google Cloud Run

on:
  push:
    branches:
      - main

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

```
    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Set up Docker Buildx
      uses: docker/setup-buildx-action@v3

    - name: Login to Docker Hub
      uses: docker/login-action@v3
      with:
        username: \${{ secrets.DOCKER_USERNAME }}
        password: \${{ secrets.DOCKER_PASSWORD }}

    - name: Build and Push Docker image
      run: |
        docker build -t my-go-app:latest .
        docker tag my-go-app:latest \${{ secrets.
            DOCKER_USERNAME }}/my-go-app:latest
        docker push \${{ secrets.DOCKER_USERNAME }}/my-go-
            app:latest

    - name: Authenticate to Google Cloud
      uses: google-github-actions/auth@v1
      with:
        credentials_json: \${{ secrets.
            GCP_SERVICE_ACCOUNT_KEY }}

    - name: Deploy to Cloud Run
      run: |
        gcloud config set project faasioflex
        gcloud run deploy faasioflex-app --image \${{
            secrets.DOCKER_USERNAME }}/my-go-app:latest --
            platform managed --region us-central1 --allow-
            unauthenticated
```

Código Fuente de la Aplicación Go

El código fuente de la aplicación "Hola Mundo" se encuentra en `src/go.main`, `src/go.mod` y `src/go.sum`.

src/go.main:

```
package main

import (
    "fmt"
    "net/http"
)

func helloWorld(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintln(w, 'Hola Mundo')
}

func main() {
```

```
    http.HandleFunc("/", helloWorld) // Configura la ruta
    fmt.Println("Servidor escuchando en http://localhost
        :8080")
    http.ListenAndServe(":8080", nil) // Inicia el servidor
}
```

src/go.mod

```
module hello-world

go 1.23
```

src/go.sum

```
module hello-world

go 1.23
```

8.6. Guía para Desplegar una Aplicación 'Hola Mundo' de Contenedores Docker en Fly.io y Heroku desde GitHub Actions

Esta documentación proporciona los pasos necesarios para desplegar una aplicación escrita en Go utilizando contenedores Docker en Fly.io y Heroku. Ambos servicios ofrecen un nivel gratuito que permite ejecutar aplicaciones sin necesidad de ingresar información de tarjeta de crédito. Esta guía proporciona pasos detallados para desplegar una aplicación 'Hola Mundo' escrita en Go utilizando contenedores Docker en Fly.io y Heroku, integrando GitHub Actions para la automatización del proceso de despliegue.

9. Estructura del Proyecto

Tu proyecto debe tener la siguiente estructura de archivos:

```
FaaSioFlex/

          .github/
                workflows/
                    deploy.yml

          src/
                go.mod
                go.sum
                main.go

          Dockerfile
```

10. Paso 1: Crear el Archivo `Dockerfile`

Crea un archivo llamado `Dockerfile` en el directorio raíz de tu proyecto con el siguiente contenido:

```
# Dockerfile

# Usar una imagen base de Go
FROM golang:1.20 AS builder

# Establecer el directorio de trabajo
WORKDIR /app

# Copiar los archivos necesarios
COPY src/go.mod ./
COPY src/go.sum ./
COPY src/ ./

# Construir la aplicaci n
RUN go build -o hello-world .

# Usar una imagen m s ligera para la ejecuci n
FROM alpine:latest
WORKDIR /root/
COPY --from=builder /app/hello-world .

# Exponer el puerto
EXPOSE 8080

# Comando para ejecutar la aplicaci n
CMD ["./hello-world"]
```

11. Paso 2: Crear la Aplicación

Crea un archivo `main.go` en el directorio `"src/"` con el siguiente contenido:

```
package main

import (
    "fmt"
    "net/http"
)

func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(w, " Hola , Mundo!")
}

func main() {
    http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

12. Paso 3: Configuración de GitHub Actions

Crea un archivo llamado `deploy.yml` en el directorio `.github/workflows/` con el siguiente contenido:

```
name: Deploy to Fly.io and Heroku

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1

      - name: Build Docker image
        run: |
          docker build -t faasioflex:latest .

      - name: Log in to Fly.io
        uses: flyctl-actions/login@v1
        with:
          api_token: ${{ secrets.FLY_API_TOKEN }}

      - name: Deploy to Fly.io
        run: |
          flyctl deploy --remote-only --app faasioflex

      - name: Log in to Heroku
        run: |
          echo "${{ secrets.HEROKU_API_KEY }}" | docker
              login --username=_ --password-stdin registry.
              heroku.com

      - name: Deploy to Heroku
        run: |
          heroku container:push web --app faasioflex
          heroku container:release web --app faasioflex
```

12.1. Variables de Entorno en GitHub

Asegúrate de agregar las siguientes variables de entorno a tu repositorio en GitHub:

```
    FLY_API_TOKEN: Token de API de Fly.io.
    HEROKU_API_KEY: Token de API de Heroku.
```

12.2. Despliegue de mi app Go

Esta documentación proporciona los pasos necesarios para desplegar una aplicación escrita en Go utilizando contenedores Docker en Fly.io, Heroku y GitHub Codespaces. Ambos servicios ofrecen un nivel gratuito que permite ejecutar aplicaciones sin necesidad de ingresar información de tarjeta de crédito.

A)Despliegue de mi app Go con GitHub Codespaces

¿Qué es GitHub Codespaces?

GitHub Codespaces es un entorno de desarrollo en la nube que permite a los desarrolladores trabajar en un entorno de desarrollo integrado (IDE) basado en la nube. Está diseñado para ser completamente personalizable y reproducible, lo que facilita la configuración y la colaboración en proyectos. Con Codespaces, se puede iniciar un entorno preconfigurado con todas las dependencias del proyecto en cuestión de segundos, lo cual reduce el tiempo de configuración y mejora la productividad.

¿Para qué sirve?

Codespaces se utiliza principalmente para facilitar el desarrollo de software, permitiendo que los desarrolladores trabajen en cualquier lugar y desde cualquier dispositivo. Es especialmente útil para proyectos que dependen de configuraciones complejas o requieren múltiples herramientas y dependencias. Con la capacidad de ejecutar contenedores y servicios en el entorno Codespaces, los desarrolladores pueden probar y ejecutar aplicaciones directamente en la nube sin necesidad de instalar todas las herramientas localmente.

Uso de GitHub Codespaces con Docker

Para desplegar una aplicación utilizando Docker dentro de un Codespace, es necesario instalar Docker en el entorno y configurarlo para ejecutar contenedores. El uso de contenedores permite aislar la aplicación y sus dependencias, lo que facilita el despliegue en diferentes entornos de desarrollo, pruebas y producción.

En la siguiente configuración de `deploy.yml`, se muestran los pasos para preparar el entorno Codespaces, instalar Docker y desplegar la aplicación Go. Además, se incluye el uso de `ngrok` para exponer el servicio localmente y acceder a la aplicación de forma remota.

Despliegue de la aplicación desde GitHub con GitHub Actions

El siguiente archivo de configuración de GitHub Actions (`deploy.yml`) describe los pasos necesarios para ejecutar la aplicación Go en un entorno Codespaces:

```
run_in_codespace:
    runs-on: ubuntu-latest
    needs: build

    steps:
      - name: Checkout Code
        uses: actions/checkout@v3

      - name: Set up Codespaces environment
        run: |
          set -e  # Detener en el primer error

          # Limpieza de la cach  y eliminaci n de listas
          sudo apt-get clean
          sudo rm -rf /var/lib/apt/lists/*
          sudo apt-get update

          # Comprobar y eliminar containerd si est
              instalado
          if dpkg -l | grep -q containerd; then
            sudo apt-get remove --purge -y containerd
                containerd.io || true
            sudo apt-get purge -y containerd || true
            sudo apt-get autoremove -y || true
          fi

          # Eliminar cualquier paquete que pueda causar
              conflictos
          sudo apt-get autoremove -y || true
      - name: Update package list
        run: |
          sudo apt-get update
      - name: Instalar dependencias necesarias
        run: |
          # Instalar dependencias necesarias ; containerd.
              io
          set -e
          sudo apt-get install -y  docker.io

      - name: Iniciar el servicio Docker
        run: |
          # Iniciar el servicio Docker
          sudo systemctl start docker || true
          sudo systemctl enable docker || true  #
              Aseg rate de que Docker se inicie en el
              arranque

          # Agregar el usuario actual al grupo docker (en
              GitHub Actions, este paso no es necesario)
          # El siguiente comando no tiene efecto, as  que
              se omite
          # sudo usermod -aG docker $USER

      - name: Check installed packages
```

```
        run: |
          dpkg -l | grep containerd

      - name: Check sources list
        run: |
          cat /etc/apt/sources.list
          ls /etc/apt/sources.list.d/

      - name: Fix broken packages
        run: |
          sudo apt-get update
          sudo apt-get upgrade -y
          sudo apt-get install -f

      - name: Clean APT cache
        run: |
          sudo apt-get clean
          sudo rm -rf /var/lib/apt/lists/*
          sudo apt-get update
          sudo apt-get install -y docker.io

      - name: Run Docker container in Codespaces
        run: |
          docker run -d -p 8080:8080 --name my-go-app ${{
              secrets.DOCKER_USERNAME }}/my-go-app:latest

      - name: Install ngrok
        run: |
          curl -s https://ngrok-agent.s3.amazonaws.com/
              ngrok.asc | sudo tee /etc/apt/trusted.gpg.d/
              ngrok.asc >/dev/null
          echo "deb https://ngrok-agent.s3.amazonaws.com
              buster main" | sudo tee /etc/apt/sources.list.
              d/ngrok.list
          sudo apt-get update
          sudo apt-get install -y ngrok

      - name: Run ngrok
        run: |
          nohup ngrok http 8080 > ngrok.log &
          sleep 5
          # Extraer la URL p blica de ngrok
          NGROK_URL=$(curl -s http://localhost:4040/api/
              tunnels | jq -r .tunnels[0].public_url)
          echo "La URL p blica de ngrok es: $NGROK_URL"
        env:
          NGROK_AUTHTOKEN: ${{ secrets.NGROK_AUTHTOKEN }}


      - name: Test the application
        run: |
          # Usar la URL p blica para probar
          curl -s "$NGROK_URL" || echo "La aplicaci n no
              respondi  correctamente"
```

```
      - name: Stop and Remove Docker container
        if: always()
        run: |
          docker stop my-go-app
          docker rm my-go-app
```

Uso de ngrok

El uso de `ngrok` permite exponer la aplicación que se ejecuta en el entorno Codespaces a través de una URL pública accesible desde cualquier lugar. Esto es útil para pruebas y demostraciones, ya que no requiere configuraciones adicionales de red ni cambios en el firewall.

Para usar `ngrok`, se debe crear una cuenta en `https://ngrok.com/` y obtener un `authtoken`, el cual debe ser configurado en los secretos del repositorio de GitHub para ser usado en el archivo de configuración de GitHub Actions. La configuración anterior muestra cómo instalar `ngrok`, ejecutar el servicio y obtener la URL pública generada automáticamente.

Con esta configuración, el despliegue en GitHub Codespaces y el uso de `ngrok` permiten probar y acceder a la aplicación Go desde cualquier ubicación sin la necesidad de un servidor web tradicional o configuraciones complejas.

B) Despliegue en Fly.io

Paso 1: Crear una Cuenta en Fly.io

- Visita `https://fly.io/` y haz clic en "Sign Up"para crear una cuenta.

Paso 2: Instalar la CLI de Fly

- Sigue las instrucciones de instalación para la CLI de Fly desde la documentación oficial: `https://fly.io/docs/getting-started/installing-flyctl/`.

Paso 3: Crear un Nuevo Proyecto

- Abre la terminal y ejecuta el siguiente comando para iniciar el proceso de creación de un nuevo proyecto:

```
fly launch
```

- Sigue las instrucciones en pantalla para crear y configurar tu aplicación.

Paso 4: Desplegar la Aplicación

- Asegúrate de que tu Dockerfile esté presente en el directorio raíz de tu proyecto.
- Despliega tu aplicación ejecutando:

```
fly deploy
```

- Espera a que el despliegue se complete y anota la URL proporcionada para acceder a tu aplicación.

C) Despliegue en Heroku

12.2.1. Paso 1: Crear una Cuenta en Heroku

- Visita `https://www.heroku.com/` y haz clic en "Sign Up"para crear una cuenta.

Paso 2: Instalar la CLI de Heroku

- Descarga e instala la CLI de Heroku desde `https://devcenter.heroku.com/articles/heroku-cli`.

Paso 3: Crear un Nuevo Proyecto

- Abre la terminal y autentícate en Heroku:

```
heroku login
```

- Crea un nuevo proyecto:

```
heroku create faasioflex
```

Paso 4: Desplegar la Aplicación

- Asegúrate de que tu Dockerfile esté presente en el directorio raíz de tu proyecto.
- Despliega tu aplicación utilizando el siguiente comando:

```
heroku container:push web --app faasioflex
```

- Luego, libera la imagen con:

```
heroku container:release web --app faasioflex
```

- Espera a que el proceso de despliegue se complete y anota la URL proporcionada para acceder a tu aplicación.

12.3. Despliegue ejemplo base con APISIX

Utilizando Docker Compose, puedes crear un archivo `docker-compose.yml` para desplegar APISIX:

```
version: '3.7'
services:
  apisix:
    image: apache/apisix:latest
    restart: always
    ports:
```

```
      - "9080:9080"   # Puerto HTTP
      - "9443:9443"   # Puerto HTTPS
    environment:
      - APISIX_ADMIN_KEY=12345   # Cambia esta clave por
          seguridad
    networks:
      - apisix-net

networks:
  apisix-net:
    driver: bridge
```

Levanta el contenedor de APISIX:

```
docker-compose up -d
```

12.4. Ejemplos Sencillos

Una vez que APISIX esté en funcionamiento, puedes realizar pruebas de rutas. Aquí tienes un ejemplo de cómo agregar una ruta en APISIX:

```
curl -X POST http://127.0.0.1:9180/apisix/admin/routes/1 -H
    "X-API-Key: 12345" -d '{
    "uri": "/hello",
    "upstream": {
        "type": "roundrobin",
        "nodes": {
            "127.0.0.1:8000": 1
        }
    }
}'
```

Verifica el despliegue accediendo a la URL configurada.

A. Anexo: Instalación y Configuración de Herramientas de Desarrollo

Este anexo proporciona detalles sobre la instalación y configuración de herramientas complementarias necesarias para el desarrollo y despliegue del entorno de trabajo.

A.1. Creación de un Repositorio en GitHub

Para gestionar el código fuente del proyecto, es recomendable utilizar GitHub. Sigue estos pasos para crear un repositorio:

1. **Crear una cuenta en GitHub**: Si no tienes una cuenta, dirígete a `https://github.com` y regístrate.
2. **Crear un nuevo repositorio**:
   - Haz clic en el botón *New* en la parte superior derecha.

- Asigna un nombre a tu repositorio y elige si quieres que sea público o privado.
- Opcionalmente, agrega una descripción y selecciona un archivo `README` y una licencia.
- Haz clic en *Create repository*.

3. **Clonar el repositorio en tu máquina local**:

```
git clone https://github.com/tu_usuario/
    nombre_del_repositorio.git
```

Cambia `tu_usuario` y `nombre_del_repositorio` por los valores correspondientes.

4. **Configurar el repositorio local**:

```
cd nombre_del_repositorio
git config --global user.name "Tu Nombre"
git config --global user.email "tu_email@ejemplo.com"
```

A.2. Crear una Cuenta de GitHub para el Proyecto

1. **Registro de una nueva cuenta de GitHub:**
   - Ve a GitHub y selecciona la opción para registrarte.
   - Usa el correo `jogugi@postgrado.upv.es` para crear la cuenta.
   - Sigue los pasos para configurar la cuenta y verificar el correo electrónico.
2. **Configuración inicial:**
   - Una vez que la cuenta esté creada, completa la configuración básica del perfil, como agregar una foto y una biografía que indique que es la cuenta oficial del proyecto FaaSioFlex.

A.3. Crear un Repositorio para el Proyecto

1. **Acceder a la cuenta recién creada:**
   - Inicia sesión en GitHub con la cuenta `XXXXXX@postgrado.upv.es`.
2. **Crear un nuevo repositorio:**
   - Ve a la pestaña de Repositoriosz selecciona "Nuevo repositorio".
   - Dale un nombre descriptivo al repositorio, por ejemplo, `FaaSioFlex`.
   - Puedes agregar una descripción, como "Proyecto de microservicios para FaaS basado en Docker, NATS, APISIX, y Dex".
   - Selecciona "Público.º "Privado"según prefieras.

- Agrega un archivo README si lo deseas.

3. **Configurar el repositorio:**
   - Agrega un archivo `.gitignore` para excluir archivos no deseados del control de versiones. Por ejemplo, puedes usar la plantilla de `.gitignore` para proyectos de Golang o Docker.
   - Puedes crear una estructura básica de carpetas para organizar el código, como `src/`, `docs/`, `configs/`, etc.

A.4. Agregar a los Colaboradores del Proyecto

1. **Invitar a los miembros del equipo:**
   - Ve a la configuración del repositorio (Settings) y selecciona "Manage Access".
   - Haz clic en Ïnvite a collaboratorz agrega los correos electrónicos:
     - `jattitti@posgrado.upv.es` (Juan Amilcart Tito Tito)
     - `ccarama@posgrado.upv.es` (Carlos Caramagna)
     - `chu@posgrado.upv.es` (Chunlu Hu)
     - `jogugi@postgrado.upv.es` (José Javier Gutiérrez Gil)

2. **Configurar permisos:**
   - Asigna los permisos adecuados a cada colaborador (por ejemplo, "Write"para que puedan hacer commits y enviar pull requests).

A.5. Configurar el Repositorio para el Despliegue

1. **Subir el código base:**
   - Una vez que el repositorio esté configurado, puedes subir el código existente o comenzar un nuevo desarrollo desde cero.
   - Usa Git para clonar el repositorio y subir el código:

```
git clone https://github.com/<nombre_usuario>/
    FaaSioFlex.git
cd FaaSioFlex
git add .
git commit -m "Initial commit"
git push origin main
```

2. **Configurar GitHub Actions (opcional):**
   - Si deseas automatizar el despliegue, puedes configurar GitHub Actions para ejecutar tests, hacer builds de Docker, o desplegar automáticamente en un servidor.

Opcional: Organización en GitHub

(a) Crear una organización

1. **Acceder a GitHub**:
   - Inicia sesión en tu cuenta de GitHub. Si no tienes una cuenta, necesitarás crear una.
2. **Navegar a las organizaciones**:
   - Ve a la sección de .ºrganizaciones". Puedes hacerlo haciendo clic en tu foto de perfil en la esquina superior derecha y seleccionando "Your organizations"(Tus organizaciones) en el menú desplegable.
3. **Crear una nueva organización**:
   - Haz clic en el botón "New organization"(Nueva organización).
4. **Seleccionar un plan**:
   - GitHub ofrece diferentes planes para organizaciones (gratuitos y de pago). Elige el que mejor se adapte a tus necesidades.
5. **Completar la información de la organización**:
   - **Nombre de la organización**: Introduce un nombre único para tu organización.
   - **Correo electrónico de contacto**: Proporciona un correo electrónico válido para recibir notificaciones y actualizaciones.
   - **Opciones de configuración**: Puedes optar por crear la organización con repositorios públicos o privados, dependiendo de tus necesidades.
6. **Invitar a miembros**:
   - Puedes invitar a otros usuarios de GitHub a unirse a tu organización en esta etapa. Proporciona sus nombres de usuario de GitHub o direcciones de correo electrónico.
7. **Revisar y crear**:
   - Revisa la información que has proporcionado y haz clic en Çreate organization"(Crear organización).
8. **Configurar tu organización**:
   - Una vez creada la organización, puedes configurarla más a fondo. Esto incluye la personalización del perfil, la creación de equipos, y la configuración de repositorios.

(b) Administrar la organización

- **Agregar o eliminar miembros**:
  - Ve a la pestaña "People"(Personas) para administrar los miembros de la organización. Puedes agregar nuevos miembros, asignar roles y permisos, o eliminar a aquellos que ya no necesiten acceso.
- **Crear equipos**:
  - Organiza a los miembros en equipos con diferentes niveles de acceso y responsabilidades. Esto ayuda a gestionar mejor los permisos y la colaboración.
- **Configurar repositorios**:
  - Crea repositorios dentro de la organización y ajusta la visibilidad y las configuraciones de acceso según sea necesario.

Conectar GitHub con Docker Hub

Pasos para Conectar GitHub con Docker Hub

1. **Crear una cuenta en Docker Hub**:
   - Si no tienes una cuenta en Docker Hub, ve a `https://hub.docker.com` y regístrate.
2. **Crear un repositorio en Docker Hub**:
   - Inicia sesión en Docker Hub y haz clic en "Create Repository".
   - Dale un nombre a tu repositorio y ajusta las configuraciones (público o privado) según tus necesidades.
3. **Generar un token de acceso en Docker Hub**:
   - Ve a tu perfil de Docker Hub y selecciona "Account Settings" >"Security".
   - Crea un nuevo token de acceso y copia el token generado.
4. **Configurar GitHub Secrets**:
   - Ve a tu repositorio de GitHub y haz clic en "Settings" >"Secrets and variables" >"Actions".
   - Crea nuevos secretos para almacenar tus credenciales de Docker Hub:
     - `DOCKER_USERNAME`: tu nombre de usuario de Docker Hub.
     - `DOCKER_PASSWORD`: el token de acceso que generaste anteriormente.

5. **Crear un archivo de flujo de trabajo (workflow)**:

   - En tu repositorio de GitHub, crea un directorio llamado `.github/workflows` si no existe.
   - Crea un archivo YAML en este directorio (por ejemplo, `docker-build.yml`) con el siguiente contenido:

```
name: Docker Build and Push

on:
  push:
    branches:
      - main  # Cambia esto si usas otra rama

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Log in to Docker Hub
      run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u "${{ secrets.DOCKER_USERN

    - name: Build Docker image
      run: docker build -t yourusername/your-repo-name:latest .

    - name: Push Docker image
      run: docker push yourusername/your-repo-name:latest
```

A.6. Instalación y Configuración de Visual Studio Code

Visual Studio Code (VS Code) es un entorno de desarrollo integrado (IDE) ligero y ampliamente utilizado para la implementación de servicios con diversos lenguajes de programación, incluyendo Go y TypeScript.

- **Instalar VS Code**:

  ```
  sudo snap install --classic code
  ```

  Este comando instalará Visual Studio Code utilizando el gestor de paquetes `snap`.

- **Configuración de Extensiones**:

  Para un entorno de desarrollo óptimo, se recomienda instalar las siguientes extensiones en VS Code:

- **Go**: Para el desarrollo en Go, instala la extensión oficial *Go* proporcionada por Microsoft.
- **TypeScript**: VS Code tiene soporte nativo para TypeScript, pero se puede instalar la extensión *TypeScript Hero* para funciones avanzadas.
- **Docker**: La extensión *Docker* ayuda a gestionar contenedores y archivos `Dockerfile` directamente desde VS Code.
- **REST Client**: Útil para probar APIs REST sin salir del editor.

- **Configuración del Entorno de Desarrollo**:

  1. **Configurar el proyecto para Go**:
     - Abre una carpeta de proyecto y ejecuta `go mod init` para inicializar un módulo Go.
     - Configura las opciones de depuración y pruebas en VS Code con el archivo `launch.json`.
  2. **Configurar el proyecto para TypeScript**:
     - Crea un archivo `tsconfig.json` para configurar las opciones de compilación de TypeScript.
     - Usa el terminal integrado de VS Code para ejecutar comandos de compilación como `tsc` y gestionar dependencias con `npm`.

Con estas configuraciones, VS Code se convierte en una herramienta eficaz para el desarrollo de servicios en Go y TypeScript.

A.7. Uso de Git para el repositorio FaaSioFlex

A.7.1. Introducción a Git

Git es un sistema de control de versiones distribuido que permite a múltiples desarrolladores colaborar en proyectos de manera eficiente. Facilita la gestión de cambios en el código fuente a lo largo del tiempo, permitiendo que los equipos trabajen simultáneamente en diferentes características sin conflictos.

Este documento proporciona una guía completa sobre cómo utilizar Git para el repositorio FaaSioFlex, incluyendo comandos comunes y ejemplos prácticos de todas las operaciones posibles.

A.7.2. Operaciones posibles con Git

A continuación se presentan las operaciones más comunes que se pueden realizar con Git en el repositorio FaaSioFlex:

Clonar el repositorio

Para clonar el repositorio FaaSioFlex en tu máquina local, utiliza el siguiente comando:

```
git clone https://github.com/FaaSioFlex/FaaSioFlex.git
```

Verificar el estado del repositorio

Para verificar el estado de los archivos en tu repositorio, utiliza:

```
git status
```

Añadir archivos al índice

Para añadir archivos al índice (staging area) antes de confirmar cambios, utiliza:

```
git add <nombre-del-archivo>
```

Para añadir todos los archivos modificados:

```
git add .
```

Confirmar cambios

Para confirmar los cambios añadidos al índice, utiliza:

```
git commit -m "Descripci n de los cambios realizados"
```

Ver el historial de commits

Para ver el historial de commits en el repositorio, utiliza:

```
git log
```

Actualizar el repositorio local

Para actualizar tu repositorio local con los últimos cambios del repositorio remoto, utiliza:

```
git pull
```

Enviar cambios al repositorio remoto

Para enviar tus cambios confirmados al repositorio remoto, utiliza:

```
git push origin <nombre-de-la-rama>
```

Crear una nueva rama

Para crear una nueva rama, utiliza:

```
git branch <nombre-de-la-rama>
```

Cambiar de rama

Para cambiar a otra rama existente, utiliza:

```
git checkout <nombre-de-la-rama>
```

Fusionar ramas

Para fusionar cambios de otra rama en la rama actual, utiliza:

```
git merge <nombre-de-la-rama>
```

Eliminar una rama

Para eliminar una rama que ya no es necesaria, utiliza:

```
git branch -d <nombre-de-la-rama>
```

Visualizar diferencias entre versiones

Para ver las diferencias entre los archivos en tu área de trabajo y el último commit, utiliza:

```
git diff
```

Revertir cambios

Para revertir cambios en un archivo específico y volver al último commit, utiliza:

```
git checkout -- <nombre-del-archivo>
```

Etiquetar commits

Para etiquetar un commit específico, utiliza:

```
git tag <nombre-de-la-etiqueta>
```

Ver las etiquetas existentes

Para ver todas las etiquetas en el repositorio, utiliza:

```
git tag
```

Eliminar una etiqueta

Para eliminar una etiqueta, utiliza:

```
git tag -d <nombre-de-la-etiqueta>
```

Crear un stash de cambios

Para guardar cambios temporales que no deseas confirmar todavía, utiliza:

```
git stash
```

Aplicar un stash

Para aplicar los cambios guardados en el stash, utiliza:

```
git stash apply
```

Eliminar un stash

Para eliminar el último stash aplicado, utiliza:

```
git stash drop
```

Buscar en el historial de commits

Para buscar un término en el historial de commits, utiliza:

```
git log --grep="<término-buscado>"
```

Ver el gráfico del historial de commits

Para visualizar el historial de commits de forma gráfica, utiliza:

```
git log --oneline --graph --all
```