Entregable 1. Pipeline CI/CD

Prácticas Continuas

Máster Universitario en Ingeniería del Software

Autores:

Hernán Salambay Roldán
Alejandro Montoya Toro
Pedro Nicolás Gomariz
Aurora Hervás López
Dongyue Yu

18 de Diciembre de 2024

Facultad de Informática
UMU

Índice

1. Introducción

En este documento se detalla la aplicación de los conocimientos adquiridos en la asignatura Prácticas Continuas en el proyecto grupal SolidarianID. La implementación de estos conocimientos ha dado lugar a la creación de un pipeline de Integración Continua y Despliegue Continuo, que ha permitido automatizar diversas etapas del proyecto, optimizando así el flujo de trabajo.

El documento se organiza en las siguientes secciones: Selección de herramientas, Plan de Integración y Despliegue Continuos, Enlace al repositorio, Conclusiones y Bibliografía.

2. Selección de herramientas

En este apartado se detallan las tecnologías y herramientas propuestas en la asignatura, junto con la justificación de su inclusión o descarte para el proyecto grupal. Se explican las razones por las cuales se seleccionaron ciertas herramientas y las consideraciones que llevaron a no utilizar otras.

2.1. Vagrant

Vagrant es una herramienta de software que permite crear y gestionar entornos de desarrollo virtualizados de manera eficiente y reproducible. Diseñada para simplificar el trabajo de desarrolladores y equipos, proporciona una capa de abstracción sobre soluciones de virtualización como VirtualBox, VMware o proveedores en la nube. A través de un archivo de configuración llamado Vagrantfile, es posible definir el sistema operativo, las dependencias, las configuraciones de red y otras especificaciones del entorno, asegurando que todos los desarrolladores trabajen en un entorno idéntico, independientemente del sistema operativo anfitrión.

El principal objetivo de Vagrant es facilitar la configuración, distribución y mantenimiento de entornos de desarrollo, reduciendo problemas de compatibilidad y eliminando el clásico dilema de "funciona en mi máquina, pero no en la tuya". Con comandos sencillos como *vagrant up* (para iniciar un entorno) o *vagrant destroy* (para eliminarlo), Vagrant automatiza la creación y administración de máquinas virtuales. Es una solución altamente profesional que promueve consistencia en los entornos de desarrollo, mejora la productividad y reduce el tiempo dedicado a la configuración manual de infraestructuras.

Aunque esta herramienta ofrece numerosas ventajas, no se ha considerado adecuada para el proyecto. En su lugar, se ha optado por Docker, ya que, al tratarse de una aplicación web basada en microservicios, resultó más lógico y sencillo utilizar Docker en lugar de Vagrant. La principal ventaja de Docker es que permite definir contenedores que se ejecutan de manera más nativa en la máquina local, además de ser una opción más ligera y eficiente para la ejecución de aplicaciones, en comparación con las máquinas virtuales, lo que optimiza tanto el rendimiento como la gestión de recursos.

2.2. Docker

Docker es una plataforma de software que permite crear, ejecutar y gestionar aplicaciones en contenedores. Los contenedores son entornos ligeros y portátiles que encapsulan una aplicación y todas sus dependencias, asegurando que esta funcione de manera consistente en cualquier sistema, ya sea en desarrollo, pruebas o producción. A diferencia de las máquinas virtuales, los contenedores comparten el kernel del sistema operativo anfitrión, lo que los hace más eficientes en términos de recursos y velocidad de inicio.

Docker utiliza un archivo de configuración llamado Dockerfile para definir cómo se construye un contenedor, especificando el sistema base, las dependencias necesarias y las configuraciones. Además, permite orquestar múltiples contenedores con herramientas como Docker Compose, ideal para gestionar aplicaciones complejas que requieren diferentes servicios. Es ampliamente utilizado en la industria debido a su capacidad para facilitar la implementación de software, mejorar la escalabilidad y garantizar entornos homogéneos en cualquier infraestructura, desde equipos locales hasta la nube.

Esta herramienta ha sido fundamental en el desarrollo de la aplicación, permitiendo tanto la realización de pruebas en local como la gestión del despliegue en una plataforma en la nube. Al estar SolidarianID compuesto por múltiples microservicios, Docker ha simplificado la creación de imágenes consistentes, utilizadas posteriormente durante el proceso de despliegue. Para ello, se diseñó un Dockerfile base común para todos los componentes de la aplicación, que luego fue complementado mediante Docker Compose, aplicando configuraciones específicas para cada contenedor.

Además de utilizar Docker para generar las imágenes, se ha empleado Docker Compose para definir un entorno de despliegue local, lo que permite verificar la

funcionalidad y el correcto funcionamiento de la aplicación. Para ello, se crearon los archivos docker-compose-dev.yml y docker-compose-prod.yml en cada uno de los componentes de la aplicación (frontend y backend). Estos archivos configuran y lanzan un contenedor para cada imagen de los componentes de la aplicación. El archivo docker-compose-dev.yml se utiliza para ejecutar la aplicación en modo desarrollo, mientras que docker-compose-prod.yml está diseñado para el despliegue en modo producción.

2.3. Git y GitHub Actions

GitHub Actions

Git es un sistema de control de versiones distribuido que permite a los desarrolladores rastrear y gestionar los cambios en el código fuente de manera eficiente. Es ampliamente utilizado para coordinar el trabajo en equipo, ya que facilita la colaboración, la fusión de cambios y el mantenimiento de un historial detallado de las modificaciones en un proyecto. Git ayuda a gestionar ramas, revertir cambios y experimentar sin riesgo, lo que lo convierte en una herramienta esencial para el desarrollo de software.

GitHub Actions, por otro lado, es una plataforma de automatización integrada en GitHub que permite crear flujos de trabajo para realizar tareas como integración continua (CI) y entrega continua (CD). Con GitHub Actions, los desarrolladores pueden configurar pipelines automatizados que ejecuten pruebas, construyan aplicaciones y desplieguen software en diferentes entornos. Estos flujos se definen mediante archivos YAML, lo que brinda flexibilidad y potencia para gestionar procesos repetitivos y mejorar la eficiencia del desarrollo. Juntas, Git y GitHub Actions ofrecen una solución robusta para gestionar código y automatizar su ciclo de vida.

Ambas herramientas han sido fundamentales en el desarrollo de la aplicación. Git se utilizó para permitir que cada desarrollador trabajara de manera independiente, generando su propio código y sincronizándolo con el repositorio central compartido, lo que facilitó la colaboración y el control de versiones. Por su parte, GitHub Actions se empleó para implementar el pipeline de CI/CD, definiendo una serie de etapas automatizadas que abarcan el proceso de construcción y despliegue de la aplicación. Este pipeline será explicado con mayor detalle en la sección Plan de Integración y Despliegue Continuos.

2.4. Ansible

Ansible es una herramienta de automatización de código abierto que permite gestionar configuraciones, aprovisionar infraestructura y orquestar tareas en múltiples sistemas de manera sencilla y eficiente. Es conocida por su enfoque sin agentes, ya que se comunica con los servidores gestionados a través de SSH, eliminando la necesidad de instalar software adicional en ellos. Esto lo hace liviano y fácil de integrar en entornos existentes.

Ansible utiliza archivos de configuración en formato YAML, llamados playbooks, donde se definen las tareas y configuraciones de forma declarativa. Es ideal para automatizar tareas repetitivas, como la instalación de software, la gestión de usuarios y la configuración de sistemas, además de escalar fácilmente en entornos complejos. Gracias a su simplicidad y su filosofía de ïnfraestructura como código", Ansible es una herramienta ampliamente utilizada en DevOps para garantizar consistencia, reducir errores y mejorar la productividad en la administración de infraestructura.

Para el proyecto SolidarianID, se ha decidido no utilizar Ansible como herramienta de automatización, ya que el despliegue de la aplicación se realizará en Amazon AWS utilizando EC2 (Elastic Compute Cloud). AWS EC2 permite lanzar y gestionar instancias virtuales, proporcionando un mayor control sobre la infraestructura y adaptándose mejor a los requerimientos específicos del proyecto.

3. Plan de Integración y Despliegue Continuos

En esta sección se describe el Plan de Integración y Despliegue Continuos (CI/CD) implementado para el proyecto SolidarianID. Se detallan las herramientas y procesos utilizados para automatizar la integración del código, la ejecución de pruebas y el despliegue de la aplicación en entornos Cloud. El objetivo de este plan es garantizar un flujo de trabajo eficiente y automático que permita facilitar un despliegue rápido y seguro de nuevas versiones del proyecto.

3.1. Descripción del pipeline CI/CD

El pipeline definido para SolidarianID lo encontramos en la figura 1. Como podemos ver, está compuesto por un total de 4 etapas, las cuales se detallan a continuación:

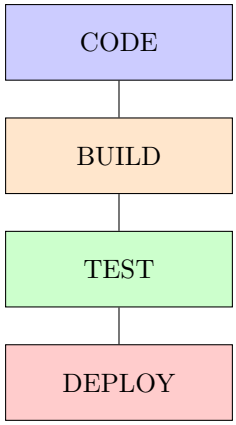

Figura 1: Pipeline de Integración y Despliegue Continuos

1. **CODE**: Esta etapa agrupa los procesos relacionados con la ejecución de Lint y Prettier sobre los ficheros de código del proyecto. Esta etapa es muy importante dentro de un equipo de desarrolladores, ya que garantiza la consistencia en la forma de escribir el código, además de asegurar que este se encuentre correctamente formateado y tabulado.

2. **BUILD**: En esta segunda etapa se realiza el compilado y construcción de la aplicación. Durante esta etapa se generarán las imágenes de cada uno de los componentes de la aplicación, lo cual será necesario para más tarde guardarlas en un repositorio ubicado en la nube.

3. **TEST**: Aquí se ejecutan todos los tipos de test de la aplicación. Esta etapa permite asegurar que el código hace lo que se espera de él, aún cuando se realice algún cambio en el código abarcado por los test.

4. **DEPLOY**: El objetivo final de cualquier aplicación web es ser desplegada y utilizada por los usuarios. Esta última etapa agrupa los procesos relacionados con desplegar la aplicación en la infraestructura de Amazon Web Services, el cual ha sido el Cloud con el que hemos trabajado durante las prácticas.

Para implementar este pipeline y aprovechar al máximo las ventajas de la integración y el despliegue continuos, hemos utilizado la herramienta GitHub

Actions. Gracias a esta herramienta, hemos configurado un workflow de ejecución automática que se activa con cada *git push* realizado en la rama *develop* del repositorio. Este workflow, denominado *ci-cd-workflow.yml*, consta de un único job llamado *code-build-test-deploy*, el cual agrupa todos los procesos necesarios para garantizar la integración y el despliegue continuos.

A continuación, se detalla el código implementado en GitHub Actions para cada etapa del pipeline:

1. Preparación inicial (pre-CODE): Antes de iniciar la etapa de código, copiamos el repositorio, configuramos el entorno de Node.js, y realizamos la instalación de dependencias tanto para el backend como para el frontend:

```
- uses: actions/checkout@v4
- name: Use Node.js ${{ matrix.node-version }}
  uses: actions/setup-node@v4
  with:
    cache-dependency-path: ./backend/package-lock.json
    node-version: ${{ matrix.node-version }}
    cache: "npm"
- name: Install backend dependencies
  working-directory: ./backend
  run: npm ci
- name: Install frontend dependencies
  working-directory: ./frontend
  run: npm ci
```

2. Etapa CODE: En esta etapa se ejecutan los comandos configurados en el archivo package.json para analizar el código con Lint y formatearlo con Prettier en ambas partes del proyecto (backend y frontend):

```
# CODE
- name: Run prettier in backend files
  working-directory: ./backend
  run: npm run prettier:check
- name: Run prettier in frontend files
  working-directory: ./frontend
  run: npm run prettier:check
- name: Run eslint in backend files
  working-directory: ./backend
  run: npm run lint
- name: Run eslint in frontend files
  working-directory: ./frontend
  run: npm run lint
```

3. Etapa BUILD: Durante la etapa de construcción, se generan los artefactos tanto para el backend como para el frontend. También se construyen las imágenes Docker de los componentes de la aplicación, se etiquetan con la ubicación del repositorio ECR y se almacenan de forma persistente.

```
# BUILD
- name: Build the backend
  working-directory: ./backend
  run: npm run build:communities-ms &&
    npm run build:users-ms &&
    npm run build:api-gateway &&
    npm run build:statistics-ms
- name: Build the frontend
  working-directory: ./frontend
  run: npm run build
- name: Build docker network
  run: docker network create --driver bridge solidarianid-network
- name: Build images
  run: docker compose -f docker-compose-app.yml build
- name: List images
  run: docker images -a
- name: Tag images
  run:
    docker tag solidarianid-communities-ms
    ${{ vars.ECR_REPOSITORY }}:communities-ms &&
    docker tag solidarianid-users-ms
    ${{ vars.ECR_REPOSITORY }}:users-ms &&
    docker tag solidarianid-api-gateway
    ${{ vars.ECR_REPOSITORY }}:api-gateway &&
    docker tag solidarianid-statistics-ms
    ${{ vars.ECR_REPOSITORY }}:statistics-ms &&
    docker tag solidarianid-frontend
    ${{ vars.ECR_REPOSITORY }}:frontend
```

4. Etapa TEST: En esta etapa, se ejecutan todas las pruebas de integración (e2e) y unitarias del backend de la aplicación:

```
# TEST
- name: Run backend tests
  working-directory: ./backend
  run: npm run test &&
    npm run test:e2e
```

Para poder ejecutar las pruebas de integración, es necesario definir contenedores dentro del workflow con todas las bases de datos de la aplicación:

```
services:
  mongodb:
    image: mongo:latest
    ports:
      - 27017:27017
    env:
      MONGO_HOST: localhost
      MONGO_DB: solidarianid-test

  postgres:
    image: postgres:latest
    ports:
      - 5432:5432
    env:
      POSTGRES_USER: solidarianid
      POSTGRES_PASSWORD: solidarianid
      POSTGRES_DB: solidarianid-test

  zookeeper:
    image: confluentinc/cp-zookeeper
    ports:
      - "2181:2181"
    env:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    image: confluentinc/cp-kafka
    ports:
      - "9092:9092"
    env:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://localhost:9092
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

5. Etapa DEPLOY: Finalmente, desplegamos la aplicación en AWS. Para ello, iniciamos sesión en el servicio ECR utilizando las credenciales proporcionadas en el laboratorio de AWS y subimos las imágenes al repositorio denominado *solidarianid*. Como paso final del pipeline, nos conectamos vía SSH a la instancia EC2 configurada para el despliegue de la aplicación y ejecutamos un script de reinicio. Este script detiene los contenedores en ejecución, actualiza las imágenes y relanza los servicios para garantizar que se ejecuten las versiones más recientes.

```
# DEPLOY
- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v3
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
    aws-region: ${{ vars.AWS_REGION }}
    aws-session-token: ${{ secrets.AWS_SESSION_TOKEN }}

- name: Login to Amazon ECR
  id: login-ecr
  uses: aws-actions/amazon-ecr-login
  @62f4f872db3836360b72999f4b87f1ff13310f3a
- name: Push images to Amazon ECR
  run: docker push ${{ vars.ECR_REPOSITORY }}:communities-ms &&
    docker push ${{ vars.ECR_REPOSITORY }}:users-ms &&
    docker push ${{ vars.ECR_REPOSITORY }}:statistics-ms &&
    docker push ${{ vars.ECR_REPOSITORY }}:api-gateway &&
    docker push ${{ vars.ECR_REPOSITORY }}:frontend

- name: Deploy application in EC2 instance
  uses: appleboy/ssh-action@v0.1.7
  with:
    host: ${{ vars.EC2_HOST }}
    username: ${{ vars.EC2_USERNAME }}
    key: ${{ secrets.EC2_SSH_KEY }}
    port: 22
    script_stop: true
    script: |
      docker compose -f docker-compose-bbdd.yml stop
      docker compose -f docker-compose-app.yml stop

      docker pull 052405837810.dkr.ecr.us-east-1.amazonaws.com
      /solidarianid:communities-ms
      docker pull 052405837810.dkr.ecr.us-east-1.amazonaws.com
      /solidarianid:users-ms
      docker pull 052405837810.dkr.ecr.us-east-1.amazonaws.com
      /solidarianid:api-gateway
      docker pull 052405837810.dkr.ecr.us-east-1.amazonaws.com
      /solidarianid:statistics-ms
      docker pull 052405837810.dkr.ecr.us-east-1.amazonaws.com
      /solidarianid:frontend

      docker compose -f docker-compose-bbdd.yml up --build -d
      docker compose -f docker-compose-app.yml up --build -d
```

3.2. Configuración en Amazon Web Services

Para desplegar la aplicación en la infraestructura de Amazon Web Services (AWS), ha sido necesario configurar una instancia EC2. Para ello, accedimos a la plataforma AWS desde el laboratorio utilizado durante las prácticas y navegamos al servicio EC2. Una vez allí, seleccionamos la opción *Lanzar instancia* y configuramos los siguientes parámetros:

1. Asignamos el nombre de la instancia como *solidarianid-app* y seleccionamos Ubuntu Server como la imagen de máquina (AMI) (figura 2).

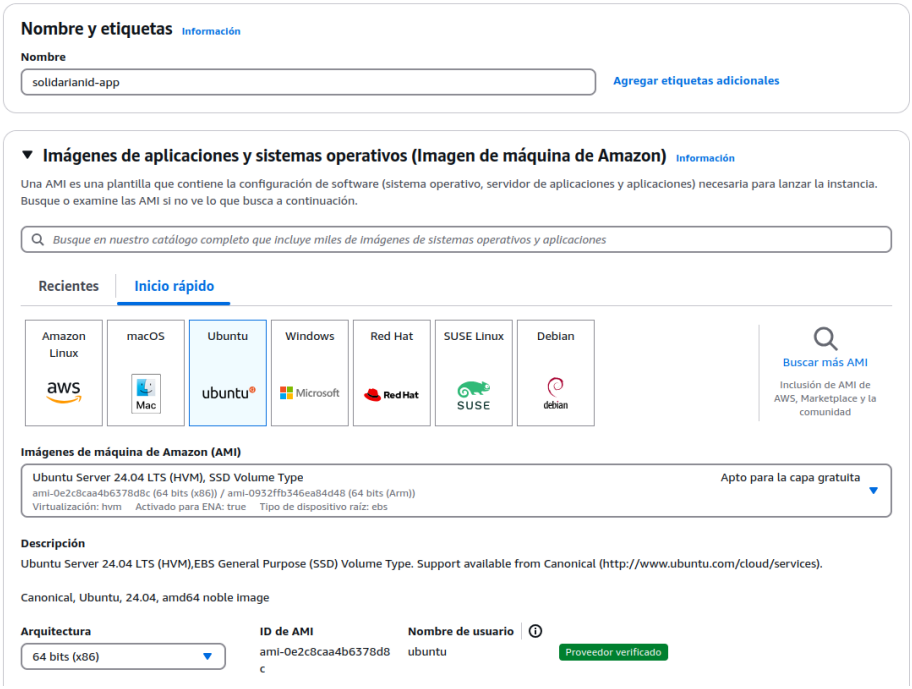

Figura 2: Configuración de nombre y AMI de la instancia EC2

2. Optamos por un tamaño de instancia medio, para garantizar un equilibrio entre rendimiento y velocidad, y generamos un par de claves para habilitar el acceso por SSH. (figura 3)

3. Finalmente, seleccionamos el grupo de seguridad *default* y configuramos 30 GB de almacenamiento para la máquina. El grupo de seguridad se ajustó para permitir todo tipo de tráfico hacia la instancia, asegurando que los clientes puedan acceder a los contenedores y que la máquina cuente con conectividad a internet para instalar los paquetes necesarios. (figura 4)

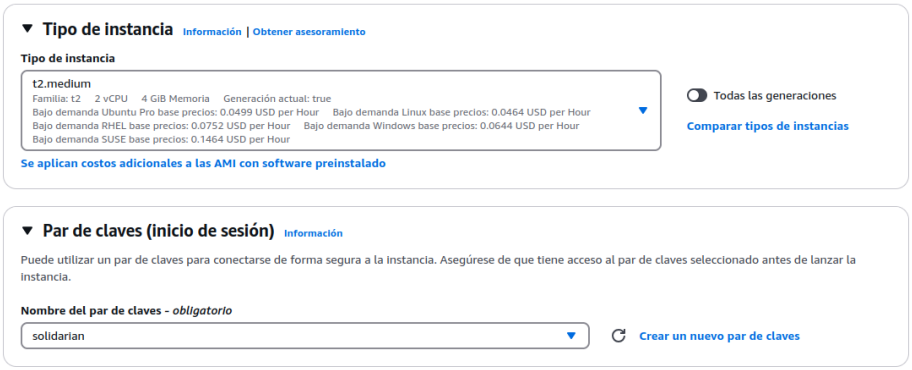

Figura 3: Configuración de tamaño de instancia y par de claves de la instancia EC2

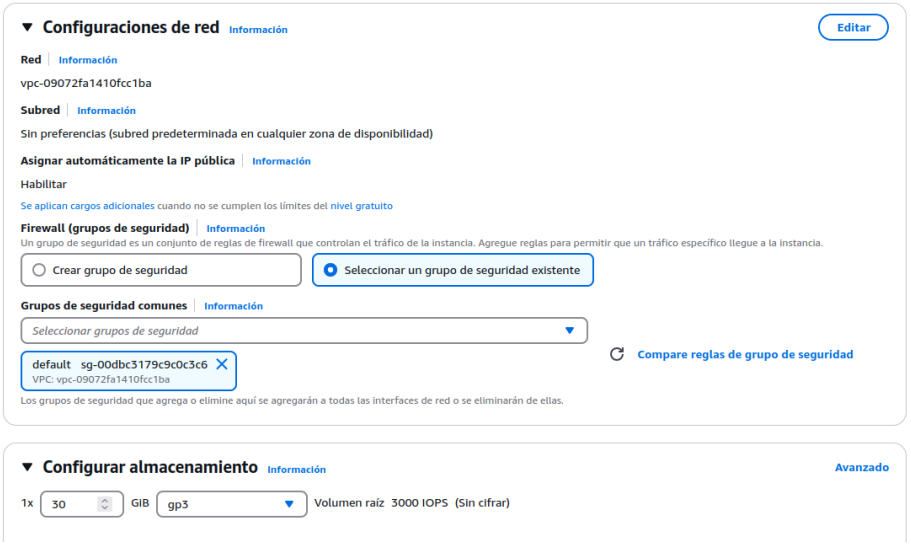

Figura 4: Configuración del grupo de seguridad y almacenamiento de la instancia EC2

4. Enlace al repositorio

El enlace al repositorio de SolidarianID del Grupo 1 es el siguiente:

`https://github.com/MISUM-G1/solidarianid.git`

5. Conclusiones

La aplicación de los contenidos de la asignatura Prácticas Continuas al proyecto SolidarianID ha sido fundamental para comprender y poner en práctica los principios de DevOps, permitiéndonos experimentar de primera mano cómo estas metodologías fomentan la colaboración entre los equipos de desarrollo y operaciones. A través de este proceso, hemos adquirido una valiosa experiencia en el uso de diversas herramientas y tecnologías clave, como Docker, GitHub Actions y AWS, que nos han permitido automatizar de manera eficiente los procesos de integración y despliegue continuos. Esta experiencia ha sido clave para optimizar el flujo de trabajo, mejorar la calidad del software y garantizar una entrega continua y confiable de la aplicación, lo que nos ha proporcionado una comprensión más profunda de cómo se aplican las prácticas modernas de desarrollo en entornos reales.

También queríamos aprovechar este apartado para resaltar que SolidarianID se ha desplegado haciendo uso de una única instancia EC2, no obstante existen métodos mucho más escalables y automatizados que ofrecen un rendimiento mejor, como sería un despliegue con ECS o EKS. La elección del despliegue en una única instancia se ha justificado por la sencillez y la cantidad de recursos que existían por la red para hacer un despliegue similar.

6. Bibliografía

La bibliografía utilizada para la redacción de este documento se ha basado principalmente en los contenidos de la asignatura Prácticas Continuas.