

# Practica Creativa 2 Grupo 32

Guillermo Peláez Cañizáres y Marcos Rosado González

[Repositorio GitHub](#)

## Bloque 1: Despliegue de la aplicación en máquina virtual pesada

### 1. Preparación de la Máquina Virtual en Google Cloud

Lo primero es preparar la Instancia de la Máquina Virtual donde se va a arrancar el Bloque 1. En esta máquina es donde se ejecutará el script de python `bloque1.py` para arrancar la aplicación en la IP pública de la Máquina Virtual.

#### Configuración de la máquina

Nombre * bloque1	
Región * europe-southwest1 (Madrid)	Zona * Cualquiera
La región es permanente	Google elegirá una zona en tu nombre, lo que maximizará la disponibilidad de las VMs. La zona es permanente.

#### Sistema operativo y almacenamiento

Nombre	bloque1
Tipo	Disco persistente balanceado nuevo
Tamaño	10 GB
Programa de instantáneas ?	No se seleccionó ningún programa
Tipo de licencia ?	Gratis
Imagen	Ubuntu 20.04 LTS

CAMBIAR

# Redes

## Firewall ?

Agrega etiquetas y reglas de firewall para permitir determinados tipos de tráfico de red desde Internet

- ☒ Permitir tráfico HTTP
- ☒ Permitir tráfico HTTPS
- ☐ Permitir las verificaciones de estado del balanceador de cargas

Etiquetas de red



Nombre de host



Configura un nombre de host personalizado para esta instancia o conserva el nombre predeterminado. La selección es permanente

Una vez creada, esperar unos minutos a que arranque la Maquina Virtual. Aqui se puede ver la IP externa de la Maquina y acceder a su terminal mediante SSH. Falta configurar una norma en el Firewall para que funcione nuestra app en el puerto 9080.

Google Cloud

Instancias de VM

INSTANCIAS OBSERVABILIDAD PROGRAMAS DE LAS INSTANCIAS

Instancias de VM

Filtro Ingresar el nombre o el valor de la propiedad

Estado	Nombre	Zona	Recomendaciones	En uso por	IP interna	IP externa	Conectar
	bloque1	europa-southwest1-c			10.204.0.12 (nic0)	34.175.65.115 (nic0)	SSH

Acciones relacionadas

- Iniciar/Reanudar
- Detener
- Suspender
- Restablecer
- Borrar
- Crear un grupo basado en esta VM VISTA PREVIA
- Ver detalles de la red
- Crea una nueva imagen de máquina
- Ver registros
- Ver supervisión

SHOW

←

Crea una regla de firewall

Las reglas de firewall controlan el tráfico saliente o entrante a una instancia. Según la configuración predeterminada, se bloquea el tráfico que entra desde el exterior de tu red. [Más información](#)

Nombre \*

http-allow-port-9080

Se permiten letras minúsculas, números y guiones

Descripción

Registros

Activar los registros de firewall puede generar una gran cantidad de registros y aumentar los costos en Logging. [Más información](#)

☐ Sí
 ☒ Desactivado

Red \*

default

Prioridad \*

1000

COMPARAR

La prioridad puede ser de 0 a 65535

Dirección del tráfico

☒ Entrada
 ☐ Salida

Acción en caso de coincidencia

☒ Permitir
 ☐ Rechazar

Destinos

Todas las instancias de la red

←

Crea una regla de firewall

Filtro de origen

Rangos de IPv4

Rangos de IPv4 de origen \*

0.0.0.0/0

Segundo filtro de origen

Ninguno

Filtro de destino

Ninguno

Protocolos y puertos

☐ Permitir todo
 ☒ Protocolos y puertos especificados

☒ TCP
 

Puertos

9080

P. ej., 20, 50-60

☐ UDP
 

Puertos

P. ej., todos

☐ SCTP
 

Puertos

P. ej., 20, 50-60

☐ Otra
 

Protocolos

Separa múltiples protocolos con comas, p. ej., ah, icmp

A continuación falta configurar la MV para que pueda ejecutar el script. Para ello ejecutar los siguientes comandos:

Actualizar paquetes:

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

Instalar Python3:

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

Instalar Git:

```
sudo apt install -y git
```

Con esto ya está configurada la MV y puede arrancar el script de la aplicación monolítica. Ahora es posible clonar desde un repositorio de GitHub los scripts necesarios para arrancar la aplicación. Como ultimo paso creamos un directorio para el bloque 1 y darle permisos de escritura y lectura.

```
mkdir bloque1
```

```
chmod 777 bloque1
```

## 2.Desarrollo del Script del Bloque 1

### Descripción del Script

El script realiza las siguientes funciones principales:

1. Construir la aplicación ( `build` ).
2. Iniciar la aplicación ( `start` y `startport` ).
3. Borrar el entorno ( `delete` ).

### Código del Script

El siguiente es el contenido del script `bloque1.py` :

```
from subprocess import call
import os, sys

# Función para construir la aplicación monolítica
def build(port='9080'):
    # Clona el repositorio e instala dependencias
    call(['git', 'clone', 'https://github.com/CDPS-ETSIT/practica_creativa2.git'])
    call(['sudo', 'apt-get', 'update'])
    call(['sudo', 'apt-get', 'install', '-y', 'python3-pip'])
    os.chdir('practica_creativa2/bookinfo/src/productpage')

    # Modifica "requirements.txt" para arreglar conflictos
    requirements = 'requirements.txt'
    with open(requirements, 'r') as file:
        lines = file.readlines()
    with open(requirements, 'w') as file:
        for line in lines:
            if line.startswith('requests=='):
                file.write('requests\n')
            else:
                file.write(line)

    # Instala los requisitos y corrige problemas de compatibilidad
    call(['pip3', 'install', '-r', 'requirements.txt'])
    call(['pip', 'install', '--upgrade', 'json2html'])

    # Configura variables de entorno
    os.environ['GROUP_NUM'] = '32'
```

```

# Modifica "productpage_monolith.py" para incluir el número de grupo
call(['mv', 'productpage_monolith.py', 'productpage_monolith_tmp.py'])
with open('productpage_monolith_tmp.py', 'r') as fin,
open('productpage_monolith.py', 'w') as fout:
    for line in fin:
        if 'flood_factor = 0' in line:
            fout.write(line)
            fout.write(os.linesep + 'groupNumber = 0 if
(os.environ.get("GROUP_NUMBER") is None) else
int(os.environ.get("GROUP_NUMBER"))' + os.linesep)
        elif 'def front():' in line:
            fout.write(line)
            fout.write('    group = groupNumber' + os.linesep)
        elif '\productpage.html\',' in line:
            fout.write(line)
            fout.write('    group=group,' + os.linesep)
        else:
            fout.write(line)
    call(['rm', '-f', 'productpage_monolith_tmp.py'])

# Modifica la plantilla HTML para incluir el número de grupo
os.chdir('templates')
call(['mv', 'productpage.html', 'productpage_tmp.html'])
with open('productpage_tmp.html', 'r') as fin, open('productpage.html', 'w')
as fout:
    for line in fin:
        if '{% block title %}Simple Bookstore App{% endblock %}' in line:
            fout.write(line.replace('{% block title %}Simple Bookstore App{%
endblock %}', '{% block title %}Grupo{{ group }}{% endblock %}'))
        else:
            fout.write(line)
    call(['rm', '-f', 'productpage_tmp.html'])

print("App built properly")

# Función para iniciar la aplicación
def start(port='9080'):
    os.chdir('practica_creativa2/bookinfo/src/productpage')
    call(['python3', 'productpage_monolith.py', port])

# Función para borrar el entorno
def delete():
    call(['rm', '-rf', 'practica_creativa2'])

# Comandos del script
param = sys.argv
if param[1] == "build":
    build()
elif param[1] == "delete":

```

```
delete()
elif param[1] == "start":
    start()
elif param[1] == "startport":
    start(param[2])
else:
    print("Unknown command")
```

## Uso del Script

bloque1.py tiene 4 comandos

Construir la aplicación

```
python3 bloque1.py build
```

Iniciar la aplicación (Puerto por defecto 9080)

```
python3 bloque1.py start
```

Iniciar la aplicación en un puerto seleccionado

```
python3 bloque1.py startport <puerto>
```

Borrar la aplicación

```
python3 bloque1.py delete
```

---

## 3. Arranque en la Nube de la Aplicación

Ahora que ya están preparados la MV y el script, se puede ejecutar el script en la consola de la MV. Para visualizar la web de la aplicación basta con visitar `http://<IP-EXTERNA>:9080`, desde cualquier dispositivo.

Cambiar de directorio

```
cd bloque1
```

Arrancar la aplicación

```
python3 bloque1.py build
```

Ahora la aplicación es visible en `http://<IP-EXTERNA>:9080`

---

## 2. Bloque 2: Despliegue de una aplicación monolítica usando docker

### 1. Creación del Dockerfile

Lo primero es desarrollar el `Dockerfile` que define el contenedor donde se va a arrancar la aplicación. El script de python es idéntico al del Bloque 1, solamente cambia el nombre de `bloque1.py` a `bloque2.py` y el puerto de 9080 a 5060.

El contenido del `Dockerfile`

```
# Usar Python como base
FROM python:3.7.7-slim

# Instalar herramientas que el script necesita
RUN apt-get update && apt-get install -y git sudo

# Configurar el directorio de trabajo (el "lugar donde trabajará el contenedor")
WORKDIR /app

# Copiar el script `bloque2.py` dentro del contenedor
COPY bloque2.py /app/

# Establecer la variable de entorno de GROUP_NUMBER
ENV GROUP_NUMBER=32

# Definir el puerto donde exponer la aplicación
EXPOSE 5060

# Definir el comando que se ejecutará cuando el contenedor inicie
CMD ["python3", "bloque2.py", "build"]
```

Este `Dockerfile` debe estar en el mismo directorio que el script `bloque2.py`

---

## 2. Creación y Carga del Contenedor en Google Cloud

Construir el contenedor a partir de la información del `Dockerfile`:

```
docker build -t product-page/32 .
```

Arrancar el contenedor en la maquina local (Opcional, para ver que funciona):

```
docker run product-page/32
```

Iniciar sesión con la cuenta de google cloud (Hace falta el Google Cloud CLI SDK):

```
gcloud auth login
```

Establecer el ID del proyecto de trabajo (Tiene que haberse creado en Google Cloud antes, en mi caso "creativa-2-445510" ):

```
gcloud config set project creativa-2-445510
```

Etiquetar el contenedor con el formato necesario para Google Cloud:

```
docker tag product-page/32 gcr.io/creativa-2-445510/product-page:latest
```

Habilitar el Container Registry para Google Cloud:

```
gcloud services enable containerregistry.googleapis.com
```

Hacer push del contenedor hacia Google Cloud:

```
docker push gcr.io/creativa-2-445510/product-page:latest
```

---

### 3. Arranque del Docker en Google Cloud





Ahora ya esta subido el contenedor de Docker construido en el paso anterior a Google Cloud. Para este bloque se va a arrancar el contenedor en *Cloud Run*.

Hacemos click en los 3 puntos del contenedor en *Implementar en Cloud Run*. Configuramos con los parámetros marcados. Las opciones *Permitir invocaciones sin autenticar*, *el Puerto 5060* y *los 2 GiB de memoria* son necesarias.





Cada servicio expone un extremo único y ajusta automáticamente la escala de la infraestructura subyacente para controlar las solicitudes entrantes. No se puede cambiar el nombre del servicio ni la región más adelante.

  Artifact Registry    Docker Hub	 GitHub	 Funciones
<input checked="" type="radio"/> Implementar una revisión desde una imagen de contenedor	<input type="radio"/> Implementar continuamente a partir de un repositorio (de origen o de función)	<input type="radio"/> Usar un editor directo para crear una función <a href="#">VISTA PREVIA</a>

URL de la imagen del contenedor  
gcr.io/creativa-2-445510/product-page:latest

[SELECCIONAR](#)

#### REALIZAR PRUEBAS CON UN CONTENEDOR DE MUESTRA

Debe detectar las solicitudes HTTP en \$PORT y no depender del estado local. [¿Cómo se compila un contenedor?](#)

## Configurar

Nombre del servicio \*

product-page-32

Región \*

europa-southwest1 (Madrid)

[¿Cómo se selecciona la región?](#)

## URL del extremo ?

https://product-page-32-898779070225.europa-southwest1.run.app

## Autenticación \*

- ☒ Permitir invocaciones sin autenticar  
Marca esta opción si estás creando una API pública o un sitio web.
- ☐ Autenticación obligatoria  
Administrar los usuarios autorizados con Cloud IAM.

## ^ Editar contenedor



## URL de la imagen del contenedor

gcr.io/creativa-2-445510/product-page:latest

## Puerto de contenedor

5060

Las solicitudes se enviarán al contenedor de este puerto. Recomendamos detectar en \$PORT, en lugar de en este número específico.

## CONFIGURACIÓN

## VARIABLES Y SECRETOS

## ACTIVACIONES DE VOLÚMENES

Nombre del contenedor: product-page-1 [EDIT](#)

## Comando de contenedor

Deja el campo en blanco para usar el comando de punto de entrada definido en la imagen de contenedor.

## Argumentos de contenedor

Argumentos pasados al comando del punto de entrada.

## Recursos

## Memoria

2 GiB

Es la memoria para asignar a cada instancia de este contenedor

## CPU

1

Es la cantidad de CPU virtuales asignadas a cada instancia de este contenedor

☐ GPU [VISTA PREVIA](#)

Access to GPU requires quota. Request a [GPU quota](#) to start using Cloud Run with GPUs.

Tras hacer click en *Crear* y esperar unos minutos, el contenedor arranca y la pagina web de la aplicación se puede visualizar en la dirección URL que aparece en la consola de *Cloud Run*

Google Cloud | Creativa 2 | Buscar (/) recursos, documentos, productos y más

Cloud Run | Detalles del servicio | IMPLEMENTAR Y EDITAR UNA NUEVA REVISIÓN | CONFIGURAR LA IMPLEMENTACIÓN CONTINUA | MÁS INFORMACIÓN

product-page-32 | Región: europe-southwest1 | URL: https://product-page-32-898779070225.europe-southwest1.run.app | Cantidad mínima de instancias del servicio: 0

MÉTRICAS | SLO | REGISTROS | REVISIONES | ACTIVADORES | REDES | SEGURIDAD | YAML

Revisiones | ADMINISTRAR EL TRÁFICO

Filtro | Filtrar revisiones

Nombre	Tráfico	Implementada ↓	Etiquetas de revisión	Acciones
product-page-32-00001-bdr	100% (a la versión más reciente)	hace 3 minutos	+	⋮

product-page-32-00001-bdr

Implementado por rosadgonzalezmarcos2@gmail.com con Cloud Console

CONTENEDORES | VOLUMES | REDES | SEGURIDAD | YAML

General

Facturación: Basada en solicitudes

Aumento de CPU de inicio: Habilitado

Simultaneidad: 80

Tiempo de espera de la solicitud: 300 seconds

Entorno de ejecución: Predeterminada

Ajuste de escala automático

Cantidad máxima de instancias de revisión: 100

Imagen: gcr.io/creativa-2-445510/product-page@sha256:739a... | Puerto: 5060

Compilación: (no hay información disponible sobre la compilación)

Fuente: (no hay información disponible sobre el origen)

Comandos y argumentos: (punto de entrada del contenedor)

Límite de CPU: 1

Límite de memoria: 2GiB

Variables de entorno (0): Ninguno

Activaciones de volúmenes (0)

Verificaciones de estado (1)

product-page-32-898779070225.europe-southwest1.run.app/productpage?u=normal

Aplicaciones | moodle | WhatsApp | Webmail UPM Alum... | UPM - CAS - Centra... | Netflix | Inicio - OneDrive | Reddit - Dive into a... | ChatGPT

BookInfo Sample | Sign in

### The Comedy of Errors

Summary: [Wikipedia Summary](#): The Comedy of Errors is one of **William Shakespeare's** early plays. It is his shortest and one of his most farcical comedies, with a major part of the humour coming from slapstick and mistaken identity, in addition to puns and word play.

[Book Details](#)

[Book Reviews](#)

Type:  
paperback  
Pages:  
65  
Publisher:  
Courier Corporation  
Language:  
English  
ISBN-10:  
9780486424613  
ISBN-13:  
9780486424613

## Bloque 3: Segmentación de una aplicación monolítica en microservicios utilizando docker-compose

### 1. Creación de la Instancia de MV en Google Cloud

Crear una Instancia de MV en Google Cloud con el nombre `bloque3` de manera idéntica a como se hizo en el primer paso de la solución del [Bloque 1](#). Lo único nuevo es al final, cuando se instalan los diferentes paquetes, hay que instalar también `Docker-Compose` y darle acceso como super usuario para no obtener futuros errores:

```
sudo apt install docker-compose
```

```
sudo usermod -aG docker $USER
```

---

## 2. Creación de los ficheros necesarios

Hay que crear varios archivos `Dockerfile` y un archivo `Docker-Compose` que orqueste a estos últimos. Todos los servicios, excepto `reviews`, tendrán un `Dockerfile` propio que hay que crear.

Siguiendo las instrucciones del enunciado, los ficheros son los siguientes:

- `productpage_dockerfile`

```
# Usar la imagen base del enunciado
FROM python:3.7.7-slim

# Establecer la variable de entorno de GROUP_NUMBER
ENV GROUP_NUMBER=32

# Definir el puerto donde exponer la aplicación
EXPOSE 9080

# Instalar herramientas que el script necesita, como `git` y `sudo`
RUN apt-get update && apt-get install -y git sudo
RUN sudo apt-get install -y python3-pip
RUN git clone https://github.com/CDPS-ETSIT/practica_creativa2.git
RUN apt-get update

# Configurar el directorio de trabajo (el "lugar donde trabajará el contenedor")
WORKDIR /practica_creativa2/bookinfo/src/productpage

# Arregla el problema de requests
RUN pip3 install -r requirements.txt
RUN pip3 install --upgrade requests

# Definir el comando que se ejecutará cuando el contenedor inicie
CMD ["python3", "productpage.py", "9080"]
```

**Importante:** Notar como se instalan mas dependencias desde el `Dockerfile` desde `requirements.txt` y se arregla directamente el problema con el paquete `requests`. Esto es asi porque ya no ejecutamos el script de los bloques anteriores, pero las dependencias se deben instalar igualmente.

- details\_dockerfile

```
# Usar la imagen base del enunciado
FROM ruby:2.7.1-slim

# Exponer por el puerto 9080
EXPOSE 9080

# Copiar 'details.rb' al directorio solicitado
COPY practica_creativa2/bookinfo/src/details/details.rb /opt/microservices/

# Establecer el directorio de trabajo
WORKDIR /opt/microservices/

# Declarar las variables de entorno que piden
ENV SERVICE_VERSION=v1
ENV ENABLE_EXTERNAL_BOOK_SERVICE=true

# Arrancar el microservicio
CMD ["ruby", "details.rb", "9080"]
```

- ratings\_dockerfile

```
# Usar la imagen base del enunciado
FROM node:12.18.1-slim

# Exponer por el puerto 9080
EXPOSE 9080

# Copiar los ficheros al directorio solicitado
COPY practica_creativa2/bookinfo/src/ratings/package.json /opt/microservices/
COPY practica_creativa2/bookinfo/src/ratings/ratings.js /opt/microservices/

# Establecer el directorio de trabajo
WORKDIR /opt/microservices/

# Declarar las variables de entorno solicitadas
ENV SERVICE_VERSION=v1

# Instalar las dependencias
RUN npm install

#Arrancar el microservicio
CMD [ "node", "ratings.js", "9080" ]
```

No hay que crear un Dockerfile para el microservicio reviews ya que este se basara en un Dockerfile dado dentro del directorio practica\_creativa2/bookinfo/src/reviews/reviews-

wlpcfg . Esto ultimo se realiza dentro de las instrucciones del archivo `compose.yaml` :

- `compose.yaml`

```
version: '3.3'

services:
  productpage:
    build:
      context: .
      dockerfile: productpage_dockerfile
    image: productpage/32
    container_name: productpage-32
    ports:
      - '9080:9080'
    environment:
      - GROUP_NUMBER=32
    volumes:
      - productpage-vol:/home/rosadogonzalezmarcos2/bloque3/volumes/productpage

  details:
    build:
      context: .
      dockerfile: details_dockerfile
    image: details/32
    container_name: details-32
    ports:
      - '9080'
    environment:
      - ENABLE_EXTERNAL_BOOK_SERVICE=true
      - SERVICE_VERSION=v1
    volumes:
      - details-vol:/home/rosadogonzalezmarcos2/bloque3/volumes/details

  reviews:
    build:
      context: practica_creativa2/bookinfo/src/reviews/reviews-wlpcfg
    image: reviews/32
    container_name: reviews-32
    ports:
      - '9080'
    environment:
      - ENABLE_RATINGS=true
      - STAR_COLOR=red
      - SERVICE_VERSION=v1
    volumes:
      - reviews-vol:/home/rosadogonzalezmarcos2/bloque3/volumes/reviews

  ratings:
```

```

build:
  context: .
  dockerfile: ratings_dockerfile
image: ratings/32
container_name: ratings-32
ports:
  - '9080'
environment:
  - SERVICE_VERSION=v1
volumes:
  - ratings-vol:/home/rosadogonzalezmarcos2/bloque3/volumes/ratings

volumes:
  productpage-vol:
  details-vol:
  reviews-vol:
  ratings-vol:

```

Servicio	Imagen	Puerto Expuesto	Variables de Entorno	Vc M
productpage	productpage/32	9080:9080	GROUP_NUMBER=32	pr vc
details	details/32	9080	ENABLE_EXTERNAL_BOOK_SERVICE =true	de
reviews	reviews/32	9080	ENABLE_RATINGS=true, STAR_COLOR=red	re
ratings	ratings/32	9080	SERVICE_VERSION=v1	ra

### 3. Arranque de la aplicación multicontenedor en Google Cloud

Lo primero de todo es pasar los ficheros que hemos creado al directorio `bloque3` en la instancia de la MV que se ha creado para este bloque. Tambien es recomendable crear un pequeño script `bloque3.py` que ejecutara los comandos de docker necesarios y los requeridos por el enunciado para arrancar la aplicación.

- `bloque3.py`

```

from subprocess import call
import os, sys

def build():

```

```

# Clonar el repositorio
os.system('git clone https://github.com/CDPS-ETSIT/practica_creativa2.git')
os.system('sudo apt install -y docker-compose')

# Construir la aplicación Reviews
os.chdir('practica_creativa2/bookinfo/src/reviews')
os.system('docker run --rm -u root -v "$(pwd)":/home/gradle/project -w
/home/gradle/project gradle:4.8.1 gradle clean build')

# Cambiar de directorio al proyecto principal
os.chdir('/home/rosadogonzalezmarcos2/bloque3')
os.system('pwd') # Mostrar la ruta actual para verificación

# Ejecutar los comandos Docker Compose
os.system('sudo docker-compose -f compose.yaml build')
os.system('sudo docker-compose -f compose.yaml up')

def start():
    os.system('sudo docker-compose -f compose.yaml up')

def startdetached():
    os.system('sudo docker-compose -f compose.yaml up -d')

def stop():
    os.system('sudo docker-compose -f compose.yaml stop')

def delete():
    os.system('sudo docker-compose -f compose.yaml down')
    os.system('sudo rm -rf practica_creativa2/')

param = sys.argv

# Comandos del script
if param[1] == "build":
    build()
elif param[1] == "start":
    start()
elif param[1] == "startdetached":
    startdetached()
elif param[1] == "stop":
    stop()
elif param[1] == "delete":
    delete()
else:
    print("Unknown command")

```



Comando	Descripción
build	Clona, compila y levanta los contenedores.
start	Inicia los contenedores en modo interactivo.
startdetached	Inicia los contenedores en segundo plano.
stop	Detiene sin eliminar los contenedores.
delete	Detiene y elimina los contenedores.

Ahora ya con todos los archivos en el directorio `bloque3` de la instancia MV de Google Cloud basta con ejecutar el siguiente comando para arrancar la aplicación.

```
python3 bloque3.py build
```

La aplicación será visible en `http://<IP-EXTERNA>:9080`

---

## 4. Mejoras

Para no tener que modificar manualmente el archivo `compose.yaml` para cambiar entre las distintas versiones de la aplicación se pueden implementar mejoras en el script `bloque3.py`. Concretamente añadir una función `state_version()` y modificar la función `build()` para que incluya a la anterior.

```
def build():
    # Si se pasa una versión como argumento, actualiza el archivo
    compose.yaml
    if len(sys.argv) > 2:
        version = sys.argv[2]
        state_version(version)
    else: print("No version specified. Using the current configuration in
compose.yaml.")

    # Clonar el repositorio si no existe
    if not os.path.exists("practica_creativa2"):
        os.system('git clone https://github.com/CDPS-
ETSIT/practica_creativa2.git')
        os.system('sudo apt install -y docker-compose')

    # Construir la aplicación Reviews
    os.chdir('practica_creativa2/bookinfo/src/reviews')
    os.system('docker run --rm -u root -v "$(pwd)":/home/gradle/project -w
/home/gradle/project gradle:4.8.1 gradle clean build')
```

```

# Cambiar de directorio al proyecto principal
os.chdir('/home/rosadogonzalezmarcos2/bloque3')
os.system('sudo docker-compose -f compose.yaml build')
os.system('sudo docker-compose -f compose.yaml up')

def state_version(version):
    # Define los valores de entorno según la versión
    env_config = {
        "v1": {
            "ENABLE_RATINGS": "false",
            "STAR_COLOR": "black",
            "SERVICE_VERSION": "v1"
        },
        "v2": {
            "ENABLE_RATINGS": "true",
            "STAR_COLOR": "black",
            "SERVICE_VERSION": "v2"
        },
        "v3": {
            "ENABLE_RATINGS": "true",
            "STAR_COLOR": "red",
            "SERVICE_VERSION": "v3"
        }
    }

    # Verifica que la versión sea válida
    if version not in env_config:
        print(f"Error: Version '{version}' not supported. Choose from v1,
v2, or v3.")
        return

    # Modifica el archivo compose.yaml
    with open("compose.yaml", "r") as file:
        compose = yaml.safe_load(file)

    compose["services"]["reviews"]["environment"] = [
        f"ENABLE_RATINGS={env_config[version]['ENABLE_RATINGS']}",
        f"STAR_COLOR={env_config[version]['STAR_COLOR']}",
        f"SERVICE_VERSION={env_config[version]['SERVICE_VERSION']}"
    ]

    with open("compose.yaml", "w") as file:
        yaml.dump(compose, file)

    print(f"Environment variables for 'reviews' updated to {version}")

```

De esta manera es posible arrancar la aplicación y elegir la version sin modificar manualmente `compose.yaml`

# Bloque 4: Despliegue de una aplicación basada en microservicios utilizando Kubernetes

## 1. Creación de los ficheros necesarios

Hay que crear los archivos `.yaml` necesarios para definir los pods y servicios de nuestra aplicación. **Importante:** hemos optado por subir a DockerHub las imágenes correspondientes al [Bloque 3](#), en un repositorio público nuestro para garantizar que las imágenes este disponible en Google Cloud. Estas imágenes son `marcoosrg/<servicio>-32`.

Siguiendo las instrucciones del enunciado:

- `details.yaml`

```
apiVersion: v1
kind: Service
metadata:
  name: details
  labels:
    app: details
    service: details
spec:
  type: ClusterIP
  ports:
    - port: 9080
      name: http
  selector:
    app: details
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: details-v1
  labels:
    app: details
    version: v1
spec:
  replicas: 3 # Factor replicación 3
  selector:
    matchLabels:
      app: details
      version: v1
```

```

template:
  metadata:
    labels:
      app: details
      version: v1
  spec:
    containers:
      - name: details
        image: marcoosrg/details-32:latest
        imagePullPolicy: IfNotPresent
        ports:
          - containerPort: 9080
        securityContext:
          runAsUser: 1000
---

```

- productpage.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: productpage
  labels:
    app: productpage
    service: productpage
spec:
  type: LoadBalancer
  ports:
    - port: 9080
      name: http
      protocol: TCP
      targetPort: 9080
  selector:
    app: productpage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: productpage-v1
  labels:
    app: productpage
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: productpage
      version: v1
  template:

```

```

metadata:
  labels:
    app: productpage
    version: v1
spec:
  containers:
    - name: productpage
      image: marcoosrg/productpage-32:latest
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 9080
      securityContext:
        runAsUser: 1000

```

- ratings.yaml

```

apiVersion: v1
kind: Service
metadata:
  name: ratings
  labels:
    app: ratings
    service: ratings
spec:
  ports:
    - port: 9080
      name: http
  selector:
    app: ratings
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ratings-v1
  labels:
    app: ratings
    version: v1
spec:
  replicas: 2 # Factor replicación 2
  selector:
    matchLabels:
      app: ratings
      version: v1
  template:
    metadata:
      labels:
        app: ratings
        version: v1

```

```
spec:
  containers:
    - name: ratings
      image: marcoosrg/ratings-32:latest
      imagePullPolicy: IfNotPresent
      ports:
        - containerPort: 9080
      securityContext:
        runAsUser: 1000
```

- reviews-svc.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: reviews
  labels:
    app: reviews
    service: reviews
spec:
  ports:
    - port: 9080
      name: http
  selector:
    app: reviews
```

- reviews-v1-deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: reviews-v1
  labels:
    app: reviews
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: reviews
      version: v1
  template:
    metadata:
      labels:
        app: reviews
        version: v1
    spec:
      containers:
```

```

- name: reviews
  image: marcoosrg/reviews-v1-32:latest
  imagePullPolicy: IfNotPresent
  env:
    - name: LOG_DIR
      value: "/tmp/logs"
  ports:
    - containerPort: 9080
  volumeMounts:
    - name: tmp
      mountPath: /tmp
    - name: wlp-output
      mountPath: /opt/ibm/wlp/output
  securityContext:
    runAsUser: 1000
volumes:
  - name: wlp-output
    emptyDir: {}
  - name: tmp
    emptyDir: {}

```

## 2. Despliegue de la aplicación en Google Kubernetes Engine

Ahora que ya se tienen definidos los archivos, podemos desplegar la aplicación en GKE, para que sea visible desde una IP externa y poder ver la aplicación desde cualquier dispositivo. Para poder hacer esto hacemos lo siguiente:

1. Creamos un cluster en Kubernetes Engine desde la consola de Google Cloud con el nombre y region apropiados ( `europa-southwest1` ):

<input type="checkbox"/> Estado	Nombre ↑	Ubicación	Nivel ?	Cantidad de nodos	CPU virtuales totales	Memoria total
<input checked="" type="checkbox"/>	<a href="#">bloque4</a>	europa-southwest1	Estándar		0	0 GB

2. Nos conectamos al cluster clickando en los 3 puntos, ya sea en Cloud Shell directamente o en la Consola de VSCode:

### Acceso a la línea de comandos

Configura el acceso a la línea de comandos de [kubecti](#) ejecutando el siguiente comando:

```
$ gcloud container clusters get-credentials bloque4 --region europa-southwest1 --project creativa-2-445510
```

[EJECUTAR EN CLOUD SHELL](#)

3. Desde el directorio donde estén todos los ficheros `.yaml` que hemos creado aplicamos los despliegues:

```
kubectl apply -f productpage.yaml
kubectl apply -f details.yaml
kubectl apply -f ratings.yaml
kubectl apply -f reviews-svc.yaml
kubectl apply -f reviews-v1-deployment.yaml
```

4. Tras esperar unos instantes, todos los pods y services se habrán creado y la aplicación será visible en la IP Externa del servicio `productpage` :

```
upm@vnxlab:~/Desktop/Creativa2/bloque4$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/details-v1-5d96c47d67-lqrh4	1/1	Running	0	2m12s
pod/details-v1-5d96c47d67-n2r7j	1/1	Running	0	2m12s
pod/details-v1-5d96c47d67-ndxvk	1/1	Running	0	2m12s
pod/productpage-v1-54b649b67c-fw9b	1/1	Running	0	4m45s
pod/ratings-v1-79c7c8b895-62ngx	1/1	Running	0	2m2s
pod/ratings-v1-79c7c8b895-788x8	1/1	Running	0	2m2s
pod/reviews-v1-584b5bcd4f-tv9wp	1/1	Running	0	107s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/details	ClusterIP	34.118.228.186	<none>	9080/TCP	2m12s
service/kubernetes	ClusterIP	34.118.224.1	<none>	443/TCP	13m
service/productpage	LoadBalancer	34.118.231.121	34.175.23.128	9080:30526/TCP	4m46s
service/ratings	ClusterIP	34.118.233.220	<none>	9080/TCP	2m3s
service/reviews	ClusterIP	34.118.231.148	<none>	9080/TCP	115s

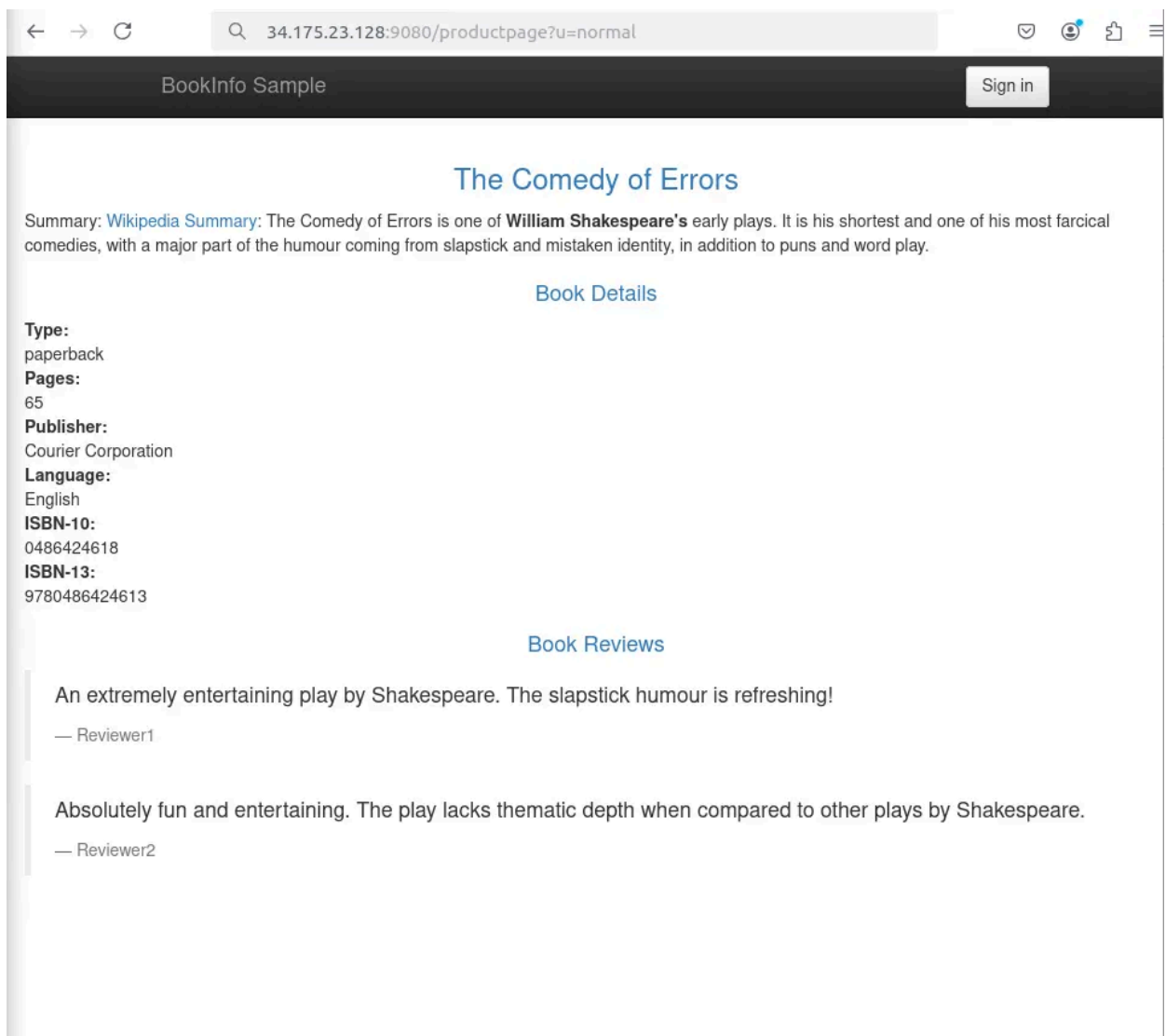
NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/details-v1	3/3	3	3	2m12s
deployment.apps/productpage-v1	1/1	1	1	4m46s
deployment.apps/ratings-v1	2/2	2	2	2m3s
deployment.apps/reviews-v1	1/1	1	1	107s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/details-v1-5d96c47d67	3	3	3	2m12s
replicaset.apps/productpage-v1-54b649b67c	1	1	1	4m45s

5. Ahora la aplicación será visible en la IP Externa mencionada y visible desde cualquier dispositivo en la dirección `http://<IP_EXTERNA>:9080`





### 3. Destrucción del escenario

Para poder limpiar y reestablecer el escenario, sin borrar el cluster de Kubernetes de Google Cloud, basta con ejecutar:

```
kubectl delete --all deployments && kubectl delete --all pods && kubectl delete --all services
```

### 4. Mejoras

Para automatizar el despliegue hemos creado el script `bloque4.py` :

```
import os
import sys
```

```

def build(version):
    os.system('gcloud container clusters get-credentials bloque4 --region europe-
southwest1 --project creativa-2-445510')
    os.system('kubectl apply -f productpage.yaml')
    os.system('kubectl apply -f ratings.yaml')
    os.system('kubectl apply -f details.yaml')
    os.system('kubectl apply -f reviews-svc.yaml')
    os.system(f'kubectl apply -f reviews-{version}-deployment.yaml')

def delete():
    os.system('kubectl delete --all deployments && kubectl delete --all pods &&
kubectl delete --all services')

param = sys.argv

if len(param) < 2:
    print("Usage: python3 bloque4.py [build|delete] [version]")
    sys.exit(1)

command = param[1]

if command == "build":
    if len(param) < 3:
        print("Please specify a version (e.g., v1, v2, v3)")
        sys.exit(1)
    version = param[2]
    build(version)
elif command == "delete":
    delete()
else:
    print("Unknown command")

```

De esta manera basta con escribir los siguientes comandos para crear y borrar el escenario:

```

python3 bloque4.py build v1
python3 bloque4.py delete

```

---