Índice

1. Estudio del problema y análisis del sistema

1.1 Introducción

En un mundo donde el desarrollo de software exige cada vez mayor velocidad, calidad y escalabilidad, surge DevOps como una filosofía transformadora que rompe las barreras tradicionales entre los equipos de desarrollo (Dev) y operaciones (Ops). Nacido a finales de la década de 2000 como una respuesta al clásico conflicto entre lanzar rápido y mantener la estabilidad, DevOps unifica la cultura, las prácticas y las herramientas necesarias para entregar software de forma continua, fiable y automatizada

Este proyecto se centra dentro de esa visión moderna del ciclo de vida del software. Su objetivo es desarrollar un servicio web contenerizado, alineado con los principios de DevOps, en el que tanto la aplicación como su

infraestructura estén diseñadas para ser fácilmente desplegables, reproducibles y escalables

El servicio consiste en una aplicación web sencilla alojada en un contenedor, creada con tecnologías ligeras y eficientes. Para contenerizarla, se utiliza Podman, una alternativa a Docker que respeta los principios de seguridad y compatibilidad con OCI. La aplicación se conecta a una base de datos SQL gestionada en Amazon RDS, aprovechando las capacidades escalables y seguras del entorno cloud de AWS

Todo el sistema es orquestado mediante Kubernetes, desplegado sobre Amazon EKS (Elastic Kubernetes Service). Este entorno permite gestionar los contenedores con alta disponibilidad, balanceo de carga, control declarativo y autoscaling, demostrando así la relevancia de Kubernetes como columna vertebral de la infraestructura DevOps moderna

Este proyecto no solo persigue una implementación funcional, sino que también representa un ejercicio integral de prácticas DevOps, abarcando desde la contenerización, automatización, gestión de configuraciones y despliegue en la nube, hasta la creación de un sistema robusto y replicable

1.2 Finalidad

Este proyecto tiene como objetivo desarrollar y desplegar un servicio web contenerizado con Podman y gestionado en AWS mediante Kubernetes, siguiendo principios DevOps

Busca demostrar cómo construir una arquitectura moderna, escalable y desacoplada, donde la aplicación y la base de datos estén separadas y desplegadas en la nube, facilitando el aprendizaje práctico de herramientas clave como EKS, RDS y la contenerización

1.3 Requisitos

Requisitos del Sistema Una vez implementado, el sistema ofrecerá los siguientes servicios:

Interfaz web accesible desde el navegador, permitiendo al usuario final interactuar con un formulario para introducir datos (correo, usuario, contraseña y teléfono)

Validación de datos del formulario en tiempo real, evitando la inserción de información inválida o malformada

Inserción segura de datos en una base de datos SQL (MySQL) alojada en Amazon RDS

Contenerización de la aplicación y despliegue en AWS mediante Kubernetes (EKS), facilitando su mantenimiento, escalado y actualización

Alta disponibilidad y balanceo de carga, garantizando que el servicio siga activo ante fallos de uno de los nodos

Separación entre lógica de aplicación y almacenamiento de datos, mejorando la seguridad, la modularidad y la mantenibilidad del sistema

Acceso controlado a la base de datos, permitiendo conexiones únicamente desde los recursos autorizados del clúster de Kubernetes

2. Recursos

2.1 Recursos hardware

Los recursos de hardware son bastante escasos puesto que usamos principalmente recursos en la nube por los que pagas por su uso, disponibilidad y tiempo de computación así que cualquier cosa que pueda correr un sistema operativo que soporte el SO, Podman, eksctl, kubectl, AWS CLI y si desarrollas en local que pueda soportar también la aplicación y dependencias metidas en el contenedor

Recursos mínimos si usas Windows 10:

- 8 GB RAM
- 2 o mas núcleos de procesador
- Conexión a Internet
- Disco duro 128 GB
- Gráficos integrados del procesador

2.2 Recursos software

Para este proyecto se necesitan:

- Una cuenta IAM de AWS
- AWS CLI (Para el manejo de AWS o usar la consola web)
- eksctl (Para crear el clúster en AWS o usar la consola web)
- kubectl (Para el manejo y uso de Kubernetes)
- Podman (Para el desarrollo del entorno de ejecución de la aplicaciones, también existe Podman Desktop)

Y un sistema operativo que pueda soportar estas aplicaciones y binarios, por ejemplo Windows 10, 11 y bastantes distribuciones usadas de linux como podría ser Ubuntu, RHEL etc

3. Planificación

3.1 Planificación temporal

Aquí en la planificación quiero hacer una aclaración, este proyecto ha tenido muchísimo mas investigación que parte práctica realmente, marcaré la estimación que me hize al principio y después pondré una aproximación un poco mas real del tiempo transcurrido

Apartado	Estimación	Realidad
Podman	11 horas	18 horas
Kubernetes	8 horas	16 horas
AWS	24 horas	34 horas

Apartado	Estimación	Realidad
Total	43 horas	68 horas

No he esclarecido cuantas horas hay de formación pero el balance general es un 70%-80% de formación y el resto de tiempo práctico

Esto es debido a que ninguna de las tecnologías usadas en este proyecto son totalmente nuevas para mi y al tener que defender lo que estoy haciendo me obliga a comprenderlo

3.2 Planificación económica

Se excluye el precio de tener un ordenador normal en casa, el coste de la luz y el internet

Podman - Gratis

Kubernetes - Gratis

Formación / Documentación que he adquirido yo - Gratis

AWS:

💰 Estimación de Costes Mensuales (24/7)

(Está en dólares porque así están las fuentes oficiales de precios)

EKS (control plane) Precio fijo por clúster $73.00/mes

EC2 nodos EKS 2x t3.micro (24/7) (~$8.20/mes c/u) $16.40 (Estos son los 2 nodos metidos en el clúster)

RDS MySQL t3.micro, 20 GB almacenamiento $15.10 (Tener activa la base de datos)

RDS Storage 20 GB x $0.115/GB $2.30 (Almacenamiento de la base de datos)

Elastic Load Balancer Uso básico + tráfico ~$20.00 (Balanceador de carga)

ECR (imágenes Docker) 500 MB – 1 GB ~$0.10 (Repositorio de imágenes de Docker)

Total con uso 24/7 (sin Free Tier):

$127–130 USD/mes

Esto costaría esta infraestructura suponiendo que no tienes un tráfico desmedido

Además esto solo sería si quieres desarrollar y mantener tu la infraestructura, si queremos contratar a gente para que mantenga la infraestructura (Ingeniero DevOps), y gestione la página web (FrontEnd developer) y la base de datos (BackEnd developer) los 3 Senior nos costaría como empleador un aproximado de:

- DevOps 6,700 – 9,600 €/mes
- FrontEnd 5,400 – 8,300 €/mes
- BackEnd 5,000 – 7,900 €/mes

Coste total: 17,217 - 25,930 €/mes

Obviamente si contratas gente para ocuparte de estas labores la infraestructura de AWS también aumentará y por lo tanto aumentará el coste

4. Desarrollo

Explicación de conceptos mas usados

¿Qué es Podman?

Podman es una herramienta nativa de Linux sin servicios, de código abierto diseñada para encontrar, construir, compartir y lanzar aplicaciones usando contenedores e imágenes de contenedores

Podman te da una CLI similar a la de otras herramientas de contenerización como docker

Podman usa un runtime que sigue el estandar OCI (Open Container Iniciative) para gestionar y crear los contenedores en vez de un demonio central

Los contenedores de podman pueden ser arrancados con o sin privilegios además de que maneja el ecosistema entero de contenedores:

- Pods
- Contenedores
- Imágenes de contenedores
- Volúmenes de contenedores

Te permite crear, arrancar y mantener los contenedores y sus imágenes en un entorno de producción

¿Qué es un contendor?

Es una unidad ligera, portable y auto-suficiente que une una aplicación con sus dependencias, librerías y entorno de ejecución para asegurarnos que la aplicación se ejecuta correctamente a lo largo de diferentes entornos

Aisla las aplicaciones del sistema manteniendo su propio sistema de archivos mientras usan el kernel del host lo cual contribuye a su eficiencia y facilidad de uso

A diferencia de las máquinas virtuales que ocupan una cantidad de espacion considerable, llegado a ocupar varios GB de espacio los contenedores pueden manejar mas aplicaciones ocupado muchísimo menos

Esto lo hace eficiente para desarrollo o pruebas

Para crear un contenedor tienes que partir de una imagen que es una plantilla la cual contiene las instrucciones o aplicaciones necesarias para hacer un contenedor

Lo que yo haré para este proyecta será crear una pod con dos contenedores uno con la aplicación web de Python Flask y otro con la base de datos

Una vez creados conectarlos y que la app de python le envíe los datos a la BBDD

La de la aplicación web será creada con un Dockerfile personalizado a partir de una imagen de Python3.13 metido en un Alpine Linux

La base de datos será un RDS de AWS

¿Qué es una pod o cápsula?

La pod es un grupo de uno o mas contenedores que comparten recursos y pueden comunicarse entre sí a través de Localhost, agrupand los contenedores en un namespace de Linux que comparten recursos específicos permitiéndome fusionar variedad de aplicaciones y son gestionadas a través de la CLI

¿Qué es un Dockerfile o Containerfile?

Es un archivo que sirve como una plantilla con una serie de instrucciones que se ejecutan de manera consecutiva para crear una imagen con la cual luego podremos construir un contenedor

¿Qué es Kubernetes?

Es una plataforma portable, extensible y de código abierto de orquestación de contenedores para administrar cargas de trabajo y servicios facilitando la automatización

¿Qué es AWS?

AWS es una plataforma de servicios en la nube ofrecida por Amazon que permite a individuos, empresas y organizaciones acceder a recursos informáticos como servidores, bases de datos, almacenamiento, redes y herramientas de desarrollo a través de internet, pagando solo por lo que utilizan

Con las 3 aplicaciones ya instaladas empezaremos a desarrollar

Código app Python Flask

La app web lo haremos de **Python Flask** que es un framework de Python para crear aplicaciones web con un número de líneas de código reducido basado en WSGI

```
# Importamos al .py el Flask, para las templates y el conector a la BBDD
import os
from flask import Flask, render_template, request
import re
import mysql.connector
from mysql.connector import Error

app = Flask(__name__)

# Acceso a la BBDD
db_config = {
    'host': os.getenv('DB_HOST', 'localhost'),  # Lee la variable de entorno
DB_HOST (o 'localhost' como valor por defecto)
    'user': os.getenv('DB_USER', 'root'),       # Lee la variable de entorno
DB_USER (o 'root' como valor por defecto)
    'password': os.getenv('DB_PASSWORD'),  # Lee la variable de entorno
DB_PASSWORD
    'database': os.getenv('DB_NAME', 'usuarios')  # Lee la variable de entorno
```

```
    DB_NAME
}

# Función para validar los datos del formulario
def validate_form(email, username, password, phone):
    email_regex = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
    phone_regex = r'^\d{10}$'
    if not re.match(email_regex, email):
        return "Correo electrónico no válido."
    if len(username) < 3:
        return "El nombre de usuario debe tener al menos 3 caracteres."
    if len(password) < 6:
        return "La contraseña debe tener al menos 6 caracteres."
    if not re.match(phone_regex, phone):
        return "El número de teléfono debe tener 10 dígitos."
    return None

# Función para insertar los datos del formulario en la base de datos
def insert_user(email, username, password, phone):
    try:
        # Conectar a la base de datos
        connection = mysql.connector.connect(**db_config)

        if connection.is_connected():
            cursor = connection.cursor()
            # Consulta SQL para insertar el usuario
            insert_query = """INSERT INTO usuarios (email, usuario, pwd, phone)
                              VALUES (%s, %s, %s, %s)"""
            # Ejecutar la consulta
            cursor.execute(insert_query, (email, username, password, phone))
            connection.commit()
            cursor.close()
            return None
    except Error as e:
        return f"Error al insertar datos: {e}"
    finally:
        if connection.is_connected():
            connection.close()

@app.route('/', methods=['GET', 'POST'])
def home():
    message = ""

    if request.method == 'POST':
        email = request.form.get('email')
        username = request.form.get('username')
        password = request.form.get('password')
        phone = request.form.get('phone')

        validation_error = validate_form(email, username, password, phone)
        if validation_error:
            message = validation_error
        else:
            insert_error = insert_user(email, username, password, phone)
```

```
            if insert_error:
                message = insert_error
            else:
                message = f"Registro exitoso para {username}!"
    # Aquí renderiza el HTML
    return render_template('index.html', message=message)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

Contenedores y pod - Podman

Empezaremos con la parte de Podman para todo el tema de contenerización

Este es el dockerfile de la app de python flask

```
FROM python:3.13-alpine

WORKDIR /app

COPY . /app

RUN apk update && apk upgrade
RUN pip install --no-cache-dir -r requirements.txt
RUN pip install --upgrade pip

EXPOSE 8080

ENTRYPOINT ["python"]
CMD ["app.py"]
```

Vamos a explicar el dockerfile:

- **FROM** dicta la imagen base sobre la que se va a construir tu imagen personalizada en este caso un Linux Alpine sobre el cual está instalado Python
- **WORKDIR** crea un directorio si no existe y le hace un cd para entrar a el en este caso nos dirige a /app
- **COPY {fuente} {destino}** copia todo lo que le digas hacia donde le digas, si pones . copiará todo lo que esté en la carpeta del dockerfile y sus hijos y con un archivo .dockerignore funciona como un .gitignore, solo pones el archivo que quieras que ignore en este caso copia todo lo que tenga en el directorio del Dockerfile a la carpeta /app
- **RUN** Ejecuta un comando cuando se crea la imagen y se pasa a la shell del sistema (CMD, /bin/bash, /bin/sh) en este caso instala todas las dependencias de Python que hayamos puesto en el archivo requirements.txt, esto se hace porque así solo tendremos que añadir la dependencia al .txt y no habría que hacer nada mas y también actualizarán el Python

```
flask==3.1.0
cryptography
mysql-connector-python
```

- **EXPOSE** Expone el puerto que has designado del contenedor a los que se puedan conectar al contenedor, esto lo haremos solo en local para las pruebas, después en vez de exponer el puerto del contenedor expondremos el puerto de la pod
- **ENTRYPOINT** Sirve para ejecutar un ejecutable en el contenedor cuando arranca, por ejemplo si quiero saber los servicios de windows que están corriendo puede servir en este caso te ejecuta Python

```
ENTRYPOINT [“Powershell”, “Get-Services”]
```

- **CMD** Pasa un argumento al comando del entrypoint, si pongo MYSQL me mostrará todos los servicios de MYSQL en este caso dentro de python me ejecuta la aplicación web

```
CMD [“MySQL”]
```

Para crear la imagen del Dockerfile es con el siguiente comando

```
PS D:\Projects> podman build -t flask_app .\PFG\
STEP 1/9: FROM python:3.13-alpine
STEP 2/9: WORKDIR /app
--> Using cache 3a2563be92ac47103c1bf59d5e4d5c329a2ea33ac0b4f2e8dfe6f9e5564e2552
--> 3a2563be92ac
STEP 3/9: COPY . /app
--> a56b1e2e0ea1
STEP 4/9: RUN apk update && apk upgrade
fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/community/x86_64/APKINDEX.tar.gz
v3.21.3-264-gb0ede8eacde [https://dl-cdn.alpinelinux.org/alpine/v3.21/main]
v3.21.3-267-gb1b14b7bf27 [https://dl-cdn.alpinelinux.org/alpine/v3.21/community]
OK: 25398 distinct packages available
(1/2) Upgrading libffi (3.4.6-r0 -> 3.4.7-r0)
(2/2) Upgrading tzdata (2025a-r0 -> 2025b-r0)
OK: 10 MiB in 28 packages
--> d23610eda503
STEP 5/9: RUN pip install --no-cache-dir -r requirements.txt
Installing collected packages: pycparser, mysql-connector-python, MarkupSafe,
itsdangerous, click, blinker, Werkzeug, Jinja2, cffi, flask, cryptography
Successfully installed Jinja2-3.1.6 MarkupSafe-3.0.2 Werkzeug-3.1.3 blinker-1.9.0
cffi-1.17.1 click-8.1.8 cryptography-44.0.2 flask-3.1.0 itsdangerous-2.2.0 mysql-
connector-python-9.2.0 pycparser-2.22
WARNING: Running pip as the 'root' user can result in broken permissions and
conflicting behaviour with the system package manager, possibly rendering your
system unusable.It is recommended to use a virtual environment instead:
```

```
https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know
what you are doing and want to suppress this warning.

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: pip install --upgrade pip
--> 683d5d11a78b
STEP 6/9: RUN pip install --upgrade pip
Requirement already satisfied: pip in /usr/local/lib/python3.13/site-packages
(24.3.1)
Collecting pip
  Downloading pip-25.0.1-py3-none-any.whl.metadata (3.7 kB)
Downloading pip-25.0.1-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 50.9 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 24.3.1
    Uninstalling pip-24.3.1:
      Successfully uninstalled pip-24.3.1
Successfully installed pip-25.0.1
WARNING: Running pip as the 'root' user can result in broken permissions and
conflicting behaviour with the system package manager, possibly rendering your
systema.io/warnings/venv. Use the --root-user-action option if you know what you
are do--> 55da837126e7
STEP 7/9: EXPOSE 8080
--> d36295992f34
STEP 8/9: ENTRYPOINT ["python"]
--> bdd93767efda
STEP 9/9: CMD ["app.py"]
COMMIT flask_app
--> 18ee5d98cd04
18ee5d98cd042fcc2a63d27a6873e106aeacd01309b06b49d807f21ecf46ee21
```

Parámetros:

- -t le dará un nombre a la imagen

Como puedes ver va paso a paso, esto te ayuda para que veas que comando falla siguiendo el orden establecido en el Dockerfile

Creamos la pod

```
PS D:\Projects> podman pod create -p 8080:8080
b3474e41f547662191e7cc88215b609f7a5b4e09458c50add9e7f9deef4fb538
PS D:\Projects> podman pod ps
POD ID        NAME             STATUS      CREATED        INFRA ID      # OF
CONTAINERS
b3474e41f547  thirsty_babbage  Created     6 seconds ago  93258c27af2e  1
PS D:\Projects> podman pod start thirsty_babbage
thirsty_babbage
PS D:\Projects> podman pod ps
POD ID        NAME             STATUS      CREATED             INFRA ID      # OF
CONTAINERS
```

```
b3474e41f547  thirsty_babbage  Running     About a minute ago  93258c27af2e  1
PS D:\Projects>
```

Parámetro:

- -p mapea el puerto 8080 de la pod al puerto 8080 del pc, esto solo sería en local, a la hora de hacer el despliegue el mapeo del puerto se pondría en el archivo.yaml

Lo que he hecho ha sido crearla mapeando el puerto 8080 y arrancarla

```
PS D:\Projects> podman run -d --pod thirsty_babbage localhost/flask_app
39c362a105e71de8282cce665fd783af3a4189046d99a21d9d8d68515d0c62b5
PS D:\Projects>
```

Parámetros:

- -d sirve para que no te agarre la shell desde donde lo has ejecutado
- --pod para meterlo en la pod que indicas

Y generamos un archivo .YAML para kubernetes de la pod

```
PS D:\Projects> podman generate kube -f flaskapp_bbdd.yaml thirsty_babbage
PS D:\Projects>
```

Este archivo sirve como un archivo de configuración a la hora de querer levantar la pod y sus contenedores y que estén completamente funcionales con el comando **podman play kube {archivo}**

Esto es lo que se usará próximamente en kubernetes en vez de ir creando la estructura poco a poco, pero también es necesario ver como se forma la estructura, el .yaml lo explicaré mas adelante

```
PS D:\Projects> podman pod ps
b3474e41f547  thirsty_babbage  Running     4 days ago  93258c27af2e  3
PS D:\Projects> podman pod rm -f b34
PS D:\Projects> podman ps -a
CONTAINER ID  IMAGE       COMMAND     CREATED     STATUS      PORTS       NAMES
PS D:\Projects> podman play kube .\PFG\flaskapp_bbdd.yaml
Pod:
b590b3160c4a3d058fc709dab1a62d900140992aa861b8c881aa7c2ffb997a31
Containers:
c4a449f42c65e16146bc5c50dc01166999164914059dd856f0ebe97abb9dbdc5
ff8545c877aa331353075791f902300d76ff72a93f54f81d1adb0d343b4b0d82

PS D:\Projects> podman pod ps
POD ID        NAME            STATUS      CREATED        INFRA ID      # OF
CONTAINERS
b590b3160c4a  thirstybabbage  Running     6 seconds ago  79d8b984af3e  3
```

```
PS D:\Projects> podman ps -a
CONTAINER ID  IMAGE                                       COMMAND     CREATED
STATUS        PORTS                                         NAMES
79d8b984af3e  localhost/podman-pause:5.4.1-1741651200               8 seconds ago
Up 8 seconds  0.0.0.0:8080->8080/tcp                        b590b3160c4a-infra
c4a449f42c65  localhost/flask_app:latest                  app.py      8 seconds ago
Up 8 seconds  0.0.0.0:8080->8080/tcp                        thirstybabbage-
nostalgicyonath
PS D:\Projects>
```

Como podrás ver he borrado la pod y por lo tanto se han borrado sus contenedores y al usar el comando anteriormente mencionado he podido volver a crear la misma estructura funcional que tenía antes

Ya que Kubernetes no es un software como tal y es una plataforma de código abierto el despliegue con Kubernetes hay que hacerlo en AWS con su servicio de Kubernetes (Elastic Kubernetes Service) o EKS

AWS y Kubernetes

Así que procederemos a usar AWS, primero nos crearemos la cuenta del usuario ROOT y el usuario IAM

El **usuario IAM** es una identidad que creas en AWS y representa la persona o aplicación que interactúa con los servicios y recursos de aws, por defecto no tiene permisos pero es la mejor práctica si los asignas correctamente, funciona como si creases un usuario en un Active Directory y le asignases unos permisos para que pueda hacer unas cosas en concreto

Será la única cosa que crearé desde la interfaz gráfica de AWS, el resto lo haré con la CLI

Para crearlo nos dirigimos al panel IAM y arriba a la derecha en "Alias de cuenta" lo configuramos, en mi caso pondré hdsm

En el buscador de la izquierda ponemos "Crear Persona", escribimos el nombre de usuario en mi caso HugoIAM y le damos acceso de usuario a la consola de admin, le asignamos una contraseña y ya estaría creado

Le asignamos todos los permisos relacionados con la consola de administración y ver el precio ya que aunque si que es cierto que pierde un poco la gracia de usar un usuario distinto al root seguirá sin poder igualar los poderes del usuario root puesto que el máximo de permisos de un usuario IAM es de 10 permisos

Y ya estaría creado, ahora solo quedaría loguear desde la CLI

```
PS D:\Projects> aws configure
```

```
PS D:\Projects> aws configure
AWS Access Key ID [None]: ********************
AWS Secret Access Key [None]: ********************
Default region name [None]: eu-west-2
Default output format [None]: json
PS D:\Projects>
```

Este comando sirve para configurar tu CLI haciéndole saber a AWS cual es el usuario con el que quieres realizar las acciones, obviamente solo si tiene permisos

Te pedirá una serie de cosas, la primera que te pide es la "AWS Access Key ID" y después la "Secret Access Key", se saca en Consola IAM AWS

Cliqueas el usuario al que te quieres loguear en **Persona** -> **Credenciales de seguridad** -> **Claves de acceso** (En caso de no teneras genérala marcango para la CLI) y cópialas y guárdalas en algún txt

Elegimos la región, eu-west-2 es Londres

Y el formato de salida de los comandos, json es el de por defecto

Ahora crearemos el repositorio ECR para subir la imagen de la app de flask

El ECR es un servicio de registro de contenedores administrado por AWS, es como si subieras una imagen al repositorio de docker.io pero lo haces al de aws

Creamos el repositorio

```
PS D:\Projects> aws ecr create-repository --repository-name flask_app
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:eu-west-
2:362107691285:repository/flask_app",
        "registryId": "362107691285",
        "repositoryName": "flask_app",
        "repositoryUri": "362107691285.dkr.ecr.eu-west-2.amazonaws.com/flask_app",
        "createdAt": "2025-05-01T19:44:10.336000+02:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": false
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

Explicación del comando:

- aws ecr: Señalamos el servicio de AWS a usar, en este caso ECR (Elastic Container Registry)
- create-repository: Creamos el repositorio donde meteremos la imagen
- --repository-name: Le asignamos un nombre al repositorio

La salida del comando te indica cosas importantes como:

El id del repositorio (repositoryArn), fecha de creación (createdAt), el nombre del repositorio (repositoryName)

Nos loguearemos desde podman y ECR para subir la imagen de flask

```
PS D:\Projects> aws ecr get-login-password --region eu-west-2 | podman login --
username AWS --password-stdin 362107691285.dkr.ecr.eu-west-2.amazonaws.com
Login Succeeded!
```

Etiquetamos la imagen de Podman local para que apunte a los repositorios de ECR

```
PS D:\Projects> podman tag flask_app:latest 362107691285.dkr.ecr.eu-west-
2.amazonaws.com/flask_app:latest
PS D:\Projects>
```

Y subimos la imagen a ECR

```
PS D:\Projects> podman push 362107691285.dkr.ecr.eu-west-
2.amazonaws.com/flask_app:latest
Getting image source signatures
Copying blob
sha256:feee10fa80a53d66b02e8688ebdf927b438b2e353c68e7d1c7e2de5d498b72e0
Copying blob
sha256:53d9c097c68dce02089ccfbac33654973e6d57f0ced0b8ecbd9e8ab6ebdb86ec
Copying blob
sha256:08000c18d16dadf9553d747a58cf44023423a9ab010aab96cf263d2216b8b350
Copying blob
sha256:052b772c7a047144fe9764df3fe0bc683abd5b2d212494aef4054f5a91f36b16
Copying blob
sha256:336e6a290569e2248d824031e4508cc6b6198e244de975b12299b3d16ae9b243
Copying blob
sha256:dd70ab34691f9ee6fe89993aa09f20316a8ffa51485d443435837fe5fb55de6e
Copying blob
sha256:563c4ac185a75496520c9f2c365fbc15b43aaae7134bad98aae4a1303e501edc
Copying blob
sha256:d80611d4948249c153e5be773b2ff05ba5b3d8defb32f33304538ec4adf6c3d1
Copying config
sha256:98a43e15ff8ca6479df5dfb6dd048f52d8e87d39fe2c666c988c216e4a94ef9c
Writing manifest to image destination
PS D:\Projects>
```

Comprobamos que la imagen haya sido subida correctamente

```
PS D:\Projects> aws ecr list-images --repository-name flask_app
{
    "imageIds": [
        {
            "imageDigest":
"sha256:879bd3f07fc950f7f54a7123ab07144e197b34eec287fe6f70a8127fd44af5c1",
```

```
            "imageTag": "latest"
        }
    ]
}

PS D:\Projects>
```

Y también crearemos la bbdd con el servicio de Amazon RDS

RDS es un servicio que te deja tener bbdd relacionales en la nube, automatiza tareas como provisionamiento de hardware, preparacion de la bbdd, parcheo y copias de seguridad

```
RDS permite 6 motores de bases de datos

    - Amazon Aurora

    - PostgreSQL

    - MySQL

    - MariaDB

    - Oracle Database

    - Microsoft SQL Server

Aurora

    Es una bbdd relacional compatible con MySQL y PostgreSQL 5 veces mas rápido
que esos motores estándar, te ayuda a reducir los costes quitando I/O innecesarios
mientras se asegura de que tus recursos se mantienen disponibles
```

```
PS D:\Projects> aws rds create-db-instance --db-instance-identifier flaskapp-db --
db-instance-class db.t3.micro --engine mysql --master-username root --master-user-
password ********** --allocated-storage 20 --vpc-security-group-ids sg-
01b079e6d1ac65356 --db-name usuarios --region eu-west-2
{
    "DBInstance": {
        "DBInstanceIdentifier": "flaskapp-db",
        "DBInstanceClass": "db.t3.micro",
        "Engine": "mysql",
        "DBInstanceStatus": "creating",
        "MasterUsername": "root",
        "DBName": "usuarios",
        "AllocatedStorage": 20,
        "PreferredBackupWindow": "03:30-04:00",
        "BackupRetentionPeriod": 1,
        "DBSecurityGroups": [],
```

```
            "VpcSecurityGroups": [
                {
                    "VpcSecurityGroupId": "sg-01b079e6d1ac65356",
                    "Status": "active"
                }
            ],
            "DBParameterGroups": [
                {
                    "DBParameterGroupName": "default.mysql8.0",
                    "ParameterApplyStatus": "in-sync"
                }
            ],
            "DBSubnetGroup": {
                "DBSubnetGroupName": "default",
                "DBSubnetGroupDescription": "default",
                "VpcId": "vpc-0f2fc9edd28642656",
                "SubnetGroupStatus": "Complete",
                "Subnets": [
                    {
                        "SubnetIdentifier": "subnet-0ed1593d38a9c1551",
                        "SubnetAvailabilityZone": {
                            "Name": "eu-west-2c"
                        },
                        "SubnetOutpost": {},
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-0b8aa002bcdbf323e",
                        "SubnetAvailabilityZone": {
                            "Name": "eu-west-2a"
                        },
                        "SubnetOutpost": {},
                        "SubnetStatus": "Active"
                    },
                    {
                        "SubnetIdentifier": "subnet-063db8cee86183226",
                        "SubnetAvailabilityZone": {
                            "Name": "eu-west-2b"
                        },
                        "SubnetOutpost": {},
                        "SubnetStatus": "Active"
                    }
                ]
            },
            "PreferredMaintenanceWindow": "sun:04:41-sun:05:11",
            "PendingModifiedValues": {
                "MasterUserPassword": "****"
            },
            "MultiAZ": false,
            "EngineVersion": "8.0.41",
            "AutoMinorVersionUpgrade": true,
            "ReadReplicaDBInstanceIdentifiers": [],
            "LicenseModel": "general-public-license",
            "OptionGroupMemberships": [
```

```
            {
                "OptionGroupName": "default:mysql-8-0",
                "Status": "in-sync"
            }
        ],
        "PubliclyAccessible": true,
        "StorageType": "gp2",
        "DbInstancePort": 0,
        "StorageEncrypted": false,
        "DbiResourceId": "db-LUA3BMXZ5S76GXBVMMROKH5VFY",
        "CACertificateIdentifier": "rds-ca-rsa2048-g1",
        "DomainMemberships": [],
        "CopyTagsToSnapshot": false,
        "MonitoringInterval": 0,
        "DBInstanceArn": "arn:aws:rds:eu-west-2:362107691285:db:flaskapp-db",
```

Explicación del comando:

aws rds create-db-instance: Crea una instancia de base de datos en Amazon RDS

Parámetros:

- --db-instance-identifier flaskapp-db: Nombre único para identificar la instancia de base de datos
- --db-instance-class db.t3.micro: Tipo de instancia. db.t3.micro es una clase pequeña usada para hacer pruebas
- --engine mysql: El motor de base de datos a usar, se especifica MySQL
- --master-username root: Usuario administrador de la base de datos
- --master-user-password **********: Contraseña del usuario administrador, mínimo de 8 carácteres
- --allocated-storage 20: Espacio de almacenamiento asignado en GB (20 GB)
- --vpc-security-group-ids sg-01b079e6d1ac65356: El ID del grupo de seguridad (security group) que permitirá o restringirá el acceso a la base de datos
- --db-name usuarios: Nombre de la base de datos que se creará automáticamente en la instancia

--region eu-west-2: Región en la que se desplegará la instancia, eu-west-2 es Londres

Creamos la tabla

Creamos el cluster de Kubernetes con eksctl

¿Por qué con eksctl y no kubectl?

eksctl es la herramienta que te crea el cluster en AWS y con kubectl lo gestionas

```
C:\Users\Duke>eksctl create cluster --name demo-flask-cluster --region eu-west-2 -
-nodes 2 --node-type t3.small --managed
2025-05-01 21:01:29 [ℹ]  eksctl version 0.207.0
```

```
2025-05-01 21:01:29 [ℹ]  using region eu-west-2
2025-05-01 21:01:30 [ℹ]  setting availability zones to [eu-west-2b eu-west-2c eu-
west-2a]
2025-05-01 21:01:30 [ℹ]  subnets for eu-west-2b - public:192.168.0.0/19
private:192.168.96.0/19
2025-05-01 21:01:30 [ℹ]  subnets for eu-west-2c - public:192.168.32.0/19
private:192.168.128.0/19
2025-05-01 21:01:30 [ℹ]  subnets for eu-west-2a - public:192.168.64.0/19
private:192.168.160.0/19
2025-05-01 21:01:30 [ℹ]  nodegroup "ng-627ab3ad" will use "" [AmazonLinux2/1.32]
2025-05-01 21:01:30 [ℹ]  using Kubernetes version 1.32
2025-05-01 21:01:30 [ℹ]  creating EKS cluster "demo-flask-cluster" in "eu-west-2"
region with managed nodes
2025-05-01 21:01:30 [ℹ]  will create 2 separate CloudFormation stacks for cluster
itself and the initial managed nodegroup
2025-05-01 21:01:30 [ℹ]  if you encounter any issues, check CloudFormation console
or try 'eksctl utils describe-stacks --region=eu-west-2 --cluster=demo-flask-
cluster'
2025-05-01 21:01:30 [ℹ]  Kubernetes API endpoint access will use default of
{publicAccess=true, privateAccess=false} for cluster "demo-flask-cluster" in "eu-
west-2"
2025-05-01 21:01:30 [ℹ]  CloudWatch logging will not be enabled for cluster "demo-
flask-cluster" in "eu-west-2"
2025-05-01 21:01:30 [ℹ]  you can enable it with 'eksctl utils update-cluster-
logging --enable-types={SPECIFY-YOUR-LOG-TYPES-HERE (e.g. all)} --region=eu-west-2
--cluster=demo-flask-cluster'
2025-05-01 21:01:30 [ℹ]  default addons metrics-server, vpc-cni, kube-proxy,
coredns were not specified, will install them as EKS addons
2025-05-01 21:01:30 [ℹ]
2 sequential tasks: { create cluster control plane "demo-flask-cluster",
    2 sequential sub-tasks: {
        2 sequential sub-tasks: {
            1 task: { create addons },
            wait for control plane to become ready,
        },
        create managed nodegroup "ng-627ab3ad",
    }
}
2025-05-01 21:01:30 [ℹ]  building cluster stack "eksctl-demo-flask-cluster-
cluster"
2025-05-01 21:01:30 [ℹ]  deploying stack "eksctl-demo-flask-cluster-cluster"
2025-05-01 21:02:00 [ℹ]  waiting for CloudFormation stack "eksctl-demo-flask-
cluster-cluster"
2025-05-01 21:09:33 [ℹ]  creating addon: metrics-server
2025-05-01 21:09:34 [ℹ]  successfully created addon: metrics-server
2025-05-01 21:09:34 [!]  recommended policies were found for "vpc-cni" addon, but
since OIDC is disabled on the cluster, eksctl cannot configure the requested
permissions; the recommended way to provide IAM permissions for "vpc-cni" addon is
via pod identity associations; after addon creation is completed, add all
recommended policies to the config file, under `addon.PodIdentityAssociations`,
and run `eksctl update addon`
2025-05-01 21:09:34 [ℹ]  creating addon: vpc-cni
2025-05-01 21:09:35 [ℹ]  successfully created addon: vpc-cni
2025-05-01 21:09:35 [ℹ]  creating addon: kube-proxy
```

```
2025-05-01 21:09:35 [ℹ]  successfully created addon: kube-proxy
2025-05-01 21:09:35 [ℹ]  creating addon: coredns
2025-05-01 21:09:36 [ℹ]  successfully created addon: coredns
2025-05-01 21:11:37 [ℹ]  building managed nodegroup stack "eksctl-demo-flask-
cluster-nodegroup-ng-627ab3ad"
2025-05-01 21:11:37 [ℹ]  deploying stack
2025-05-01 21:14:08 [ℹ]  waiting for the control plane to become ready
2025-05-01 21:14:09 [✔]  saved kubeconfig as "C:\\Users\\Duke\\.kube\\config"
2025-05-01 21:14:09 [ℹ]  no tasks
2025-05-01 21:14:09 [✔]  all EKS cluster resources for "demo-flask-cluster" have
been created
2025-05-01 21:14:09 [ℹ]  nodegroup "ng-627ab3ad" has 2 node(s)
2025-05-01 21:14:09 [ℹ]  node "ip-192-168-48-47.eu-west-2.compute.internal" is
ready
2025-05-01 21:14:09 [ℹ]  node "ip-192-168-81-142.eu-west-2.compute.internal" is
ready
2025-05-01 21:14:09 [ℹ]  waiting for at least 2 node(s) to become ready in "ng-
627ab3ad"
2025-05-01 21:14:09 [ℹ]  nodegroup "ng-627ab3ad" has 2 node(s)
2025-05-01 21:14:09 [ℹ]  node "ip-192-168-48-47.eu-west-2.compute.internal" is
ready
2025-05-01 21:14:09 [ℹ]  node "ip-192-168-81-142.eu-west-2.compute.internal" is
ready
2025-05-01 21:14:09 [✔]  created 1 managed nodegroup(s) in cluster "demo-flask-
cluster"
2025-05-01 21:14:09 [✖]  kubectl not found, v1.10.0 or newer is required
2025-05-01 21:14:09 [ℹ]  cluster should be functional despite missing (or
misconfigured) client binaries
2025-05-01 21:14:09 [✔]  EKS cluster "demo-flask-cluster" in "eu-west-2" region
is ready

C:\Users\Duke>
```

Una vez hemos creado el cluster cambiamos a kubectl y comprobamos que se hayan creado los nodos

```
PS D:\Projects> kubectl get nodes
NAME                                          STATUS   ROLES    AGE     VERSION
ip-192-168-48-47.eu-west-2.compute.internal   Ready    <none>   3m24s   v1.32.1-
eks-5d632ec
ip-192-168-81-142.eu-west-2.compute.internal   Ready    <none>   3m24s   v1.32.1-
eks-5d632ec
```

Ejecutamos el yaml con toda la pod

```
PS D:\Projects> kubectl apply -f PFG/flaskapp_bbdd.yaml
pod/flaskapp-bbdd created
PS D:\Projects>
```

Comprobamos que se haya creado la pod

```
PS D:\Projects> kubectl get pods
NAME            READY   STATUS    RESTARTS   AGE
flaskapp-bbdd   0/2     Pending   0          2m12s
PS D:\Projects>
```

5. Conclusiones finales

5.1 Cumplimiento de los requisitos fijados

El cumplimiento con los requisitos fijados en el anteproyecto es total

Está el servicio web metido en un contenedor

Lo he desplegado en AWS

Y con Kubernetes le estoy proporcionando mejoras como balanceador de carga y alta disponibilidad

5.2 Propuestas de mejora

Automatización de tareas con Terraform

6. Guías

6.1 Guía de uso

6.2 Guía de instalación

Empezaremos con la instalación del software:

Instalación Podman

Iremos a https://podman.io/ y tenemos 2 opciones de descarga.

La Desktop y la CLI, yo descargaré la Desktop porque así instala la CLI también.

Cuando ejecutemos el .exe nos dará la opción de descargar 3 dependencias e instalaremos las 3.

Te pedirá que instales WSL o Windows HyperV, el motivo de esto es que para crear un contenedor de Linux el sistema de contenerización necesita acceso al Kernel de Linux y lo que hace el WSL es justamente instalarte un Kernel de Linux

Instalación Kubernetes

Ahora instalaremos Kubernetes que usaremos Kubectl para la command line tool

Para kubectl nos dirigimos a (Kubernetes)[kubernetes.io] en el apartado de documentación a la derecha Tasks -> Install tools y ejecutamos el siguiente comando

```
curl.exe -LO "https://dl.k8s.io/release/v1.32.0/bin/windows/amd64/kubectl.exe"
```

El .exe que se nos ha descargado lo movemos a una carpeta en C: llamada kube

En Panel de Control -> Sistema y Seguridad -> Sistema -> Configuración avanzada de Sistema -> Variables de entorno -> Path -> Añades la ruta del .exe de kube que en mi caso es C:\kube

Instalación Hyper-V

Además tendremos que usar Hyper-V que es el hipervisor que mejor funciona con Kubernetes

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Hyper-V -All
```

Instalación AWS CLI y EKSCTL

Ponemos lo siguiente en la cmd

```
msiexec.exe /i https://awscli.amazonaws.com/AWSCLIV2.msi /qn
```

Y confirmamos su instalación con:

```
C:\Users\Duke>aws --version
aws-cli/2.27.5 Python/3.13.2 Windows/10 exe/AMD64
```

Para la ctl de eks

Accedes a este enlace Instalación eksctl

Descargas el AMD64

Y añades el binario al PATH

7. Referencias bibliográficas

Curso Podman

Curso AWS Cloud Practicioner Oficial

Instalación Kubernetes

Instalación eksctl

Documentación Kubernetes

Documentación Python Flask

Documentación Dockerfile

Documentación AWS (La instalación de AWS CLI también está incluida aquí)