Obligatorio

Programación Para DevOps

2025

Guillermo Larrea N.º 326256

Fernando Sosa N.º 182835

Grupo: M4A

Docente: Christian Gastón Huelmo Rodríguez

Contenido

1. Declaración de autoría

Trabajo ideado y redactado por Guillermo Larrea y Fernando Sosa.

2. Entrega en GitHub

Todo el contenido de la entrega se encuentra GitHub y se podra acceder por el siguiente enlace: https://github.com/FernandoSosa-123/DevOps-Obligatorio-2025

3. Contenido del README

DevOps-Obligatorio-2025

Contenido:

1- Introduccion
Banco Riendo estuvo analizando los beneficios de adoptar un modelo de nube híbrida por requerimientos del negocio la adopción deber ser lo más acelerada posible por lo tanto nos encargamos de las siguientes tareas:

1- Un script en Bash que cree los usuarios contenidos en un archivo pasado como parámetro, con información que especifica la Shell por defecto, directorio home, comentario y si se va a crear o no el directorio home si no existe. El script incluye las siguientes opciones e informa, para cada usuario contenido en el archivo pasado como parámetro, el resultado de la creación (si fue creado con éxito o no) y crear todos los usuarios con una contraseña pasada como parámetro de la opción.

2- Un script en Python en donde se deberá automatizar el despliegue de la aplicación de recursos humanos en donde se alojan información sensible como son los nombres, email y salario de los empleados actuales de la empresa con los siguientes requerimientos: Este despliegue considere todas las medidas de seguridad para evitar filtraciones, todos los cambios deberán ser trazables mediante un repositorio de GitHub y la documentacion se encontrara en el README en la cual se deberá ver reflejado la descripción del proyecto, requerimientos, modo de uso, etc.

2- Script de bash

El siguiente script ej1_crea_usuarios.sh se tendrá que ejecutar con permisos sudo, también se encuentra un ejemplo del archivo que contendrá los usuarios a crear.

Ejemplo de uso: ej1_crea_usuarios.sh [-i] [-c contraseña] usuarios.example

El script crea los usuarios indicados en el archivo usuarios.example pasado como parámetro, definiendo para cada usuario (según el contenido del archivo pasado como parámetro) los campos de shell por defecto, directorio home, comentario y si se va a crear o no el directorio home si no existe. En caso que alguno de esos campos no esté definido para algún usuario en el archivo usuarios.example, se creara el usuario usando los valores por defecto que utilice el comando que crea los usuarios; useradd.
La información de los usuarios, contenida en el archivo pasado como parámetro, estará separada por ":" y tendrá una sintaxis de la forma:
Nombre de usuario: Comentario: Directorio home: crear dir home si no existe (SI/NO): Shell por defecto
Si el script recibe el modificador -i (desplegar información de la creación de los usuarios), para cada usuario a crear, contenido en el archivo usuarios.example, se desplegará información del resultado de la creación o intento de creación del mismo. Después del listado de información de cada usuario, y con una línea vacía de separación, se desplegará el total de usuarios creados con éxito.
Si el script recibe el modificador -c, seguido de un texto que se considerará como una contraseña, el script asignará esa contraseña a cada uno de los usuarios creados. Si no se recibe este modificador (no se define una contraseña para los usuarios), se tomará (en este aspecto) el comportamiento por defecto del comando que se use para crear los usuarios; useradd.
En caso de errores, el script terminara con un código de retorno distinto de 0 y diferente para cada posible error encontrado (como por ejemplo archivo inexistente, que no sea un archivo regular o no se tengan permisos de lectura sobre él, sintaxis incorrecta del archivo pasado como parámetro -donde alguna de sus líneas no contenga exactamente 4 campos separados por ":"-, parámetros incorrectos -como no recibirse la contraseña al usarse el modificador -c o usarse modificadores inválidos-, cantidad de parámetros incorrectos, y cualquier otro error que se produzca), y desplegará un mensaje de error adecuado por la salida estándar de errores.

2.1- Ejemplo de uso

Suponga que el archivo Usuarios, que está en el directorio corriente de trabajo, contiene la siguiente información (los campos del archivo son Nombre de usuario: Comentario: Directorio home: crear el directorio home si no existe (SI/NO): Shell por defecto):

pepe:Este es mi amigo pepe:/home/jose:SI:/bin/bash

papanatas:Este es un usuario trucho:/trucho:NO:/bin/sh

elmaligno::::/bin/el_maligno

Y se ejecuta el comando:
ej1_crea_usuarios.sh -i -c "123456" usuarios.example

Entonces se crean con éxito los usuarios pepe y elmaligno pero no se puede crear el usuario papanatas, desplegara por la salida estándar la información siguiente (por la opción -i):
Si se pueden crear con éxito los usuarios pepe y elmaligno pero no se puede crear el usuario papanatas, se deberá (aparte de crear los usuarios pepe y elmaligno con contraseña 123456) desplegar
por la salida estándar la información siguiente (por la opción -i):

Usuario pepe creado con éxito con datos indicados:

Comentario: Este es mi amigo pepe

Dir home: :/home/jose

Asegurado existencia de directorio home: SI

Shell por defecto: /bin/bash

ATENCION: el usuario papanatas no pudo ser creado

Usuario elmaligno creado con éxito con datos indicados:

Comentario: < valor por defecto >

Dir home: < valor por defecto >

Asegurado existencia de directorio home: < valor por defecto >

Shell por defecto: /bin/el_maligno

Se han creado 2 usuarios con éxito.

3- Script de python

El script principal para desplegar la aplicación es deploy_app.py

Este script despliega automáticamente una aplicación web PHP con apache en AWS, creando y configurando todos
los recursos necesarios AWS que se detallarán a continuacion:

1) Importa librerías y variables de entorno desde el archivo .env

2) Crea y devuelve un cliente EC2 autenticado con las credenciales del .env

3) Crea el key pair en AWS y evita duplicados, además almacena el .pem localmente

4) Crea el Security Group para EC2, si existe obtiene su ID

5) Gestiona el Security Group de la base de datos (crea o reutiliza el existente)

6) Abre puertos (22/80) en EC2 y permite a EC2 acceder a DB (3306)

7) Crea cliente RDS para administrar instancias de base de datos

8) Consulta RDS y devuelve el endpoint público de la DB

9) Crea la instancia RDS o usa la existente y devuelve su endpoint

10) Crea y devuelve un cliente S3 con credenciales AWS

11) Crea un cliente EC2 tipo resource para manipular instancias

12) crear o reutilizar una instancia EC2 en AWS y preparar la aplicación web para que quede disponible

13) Genera el script de Bash al inicializar la instancia

13.1) actualiza sistema e instala paquetes

13.2) configura e inicia servicios apache y php

13.3) copia aplicacion desde el s3

13.4) Importar base de datos a RDS

13.5) Crear archivo .env con la configuración

13.6) Configurar permisos

13.7) Reinicia los servicios para aplicar cambios

3.1- Requerimientos para desplegar la aplicacion
a) Sera necesario contar con los servicios aws y las Credenciales STS (sean temporales o fijas)

b) Se debera contar con los archivos de la App y el archivo init_db.sql

c) Tendra que crear un archivo .env tomando de ejemplo el .env.example con todas las credenciales a usar

d) Tendra que crear un ambiente virtual .venv de python donde instalara las librerias necesarias desde el
archivo requirements.txt para ejecutar la aplicacion deploy_app.py

3.2 Como crear ambiente virtual python
verificar primero si esta python instalado python3 --version

instalacion python y modulo ambiente virtual
sudo apt update

sudo apt install python3

sudo apt install python3-venv

creacion de ambiente virtual
python3 -m venv .venv

Activacion del entorno
source .venv/bin/activate

Desactivacion del entorno
deactivate

Instalacion de Dependencias
Activar tu entorno virtual
source .venv/bin/activate

Opcion 1: Instalar desde requirements.txt
pip install -r requirements.txt

Opcion 2 Instalar manualmente
pip install boto3

pip install dotenv

4- GITHUB
GitHub es la plataforma que usaremos para gestionar este proyecto, facilitando el trabajo colaborativo,
permitiendo crear ramas, revisar cambios, integrar código y mantener un historial del desarrollo.
4.1- Instalación y configuración inicial
Instalar Git
sudo apt install git-all
Generar clave SSH
ssh-keygen -t ed25519 -C "tu_email@example.com"
Copiar clave y agregar en GitHub
cat ~/.ssh/id_ed25519.pub
Pegar en GitHub → Settings → SSH and GPG keys
Verificar conexión
ssh -T git@github.com
Clonar repositorio git
git clone git@github.com:FernandoSosa-123/DevOps-Obligatorio-2025.git

Creacion de nueva rama
git checkout -b feature/mirama

subir rama por primera vez
git add .

git commit -m"Descripcion de cambios"

git push --set-upstream origin feature/mirama

en github abrir pull request y aprobar el merge main
4.2- Flujo de trabajo diario
Hacer cambios en archivos
git checkout feature/mirama

git add . (todos los archivos)

git add nombre_archivo (un solo archivo)

git status (verifica los cambios a subir)

git commit -m "Descripción de los cambios"

git push

en github abrir pull request y aprobar el merge main
Flujo copiar main a mi rama y subir rama
git checkout main

git fetch

git pull

git checkout feature/mirama

git merge main

git push
Flujo copiar mi rama a la main
git checkout main

git pull

git merge feature/mirama

git push
4.3- Archivos importantes
.gitignore: Archivos que Git ignorará

.env.example: Template de variables de entorno

.env: Variables locales (NUNCA subir a Git)

.venv/ Entorno virtual de python (NUNCA subir a git)

test_deploy_app.py Script temporal para testeo (NUNCA subir a git)

5. Script de BASH

```
#!/bin/bash

# ------------------------------------------------------------
# Descripción: Crea usuarios desde un archivo con formato definido.
# Autor: Guillermo Larrea, Fernando Sosa y ChatGPT (GPT-5)
# ------------------------------------------------------------

#Codigos de error, se manejan como variables permitiondo que estos sean cambiados de ser nesesario de una manera sencilla
error_formato=1
error_arch=2
error_perm=3

#Variables a usar
desplegar_info=0
```

```
contrase=""
archivo_usuarios=""
comprobacion_usuario=0

#Funcion que señala error de uso y muestra su correcto funcionamiento
mostrar_uso() {
   echo "Modo de uso: $0 [-i] [-c contraseña] archivo_usuarios" >&2
   exit $error_formato
}

#Con el comando "getopts" nos permitira prosesar modificadores nesesarios "-i" y "-c" con una
contraseña
while getopts ":ic:" opcion; do
   case "$opcion" in
      i) desplegar_info=1
            #controlo si nesesito desplegar la informacion de creacion de usuario
            ;;
      c) contrase="$OPTARG"
            #getopts pone automaticamente el argumento que continua del modificadores "-c"
            #de una opción en la variable OPTARG que sera la contaseña de los usuarios
            ;;
      :) echo "Error de uso, la opción -$OPTARG requiere un argumento." >&2
            #La opcion ":" en un "case" usando "getopts" señala que no axiste un argumento despues
del "-c"
            mostrar_uso
            ;;
      \?) echo "Error de uso, opción inválida -$OPTARG" >&2
            mostrar_uso
            ;;
   esac
done

#getopts solo procesa opciones y sus argumentos, sin tomar en cuenta argumentos posicionales
(argumentos sin - o --)
#Esto puede generar error si los usuarios introducen argumentos extra
#Esto se solucionara con el comando "shift" y la variable de "getopts" "OPTIND", que mueve los
modificadores dado por el usario
shift $((OPTIND - 1))

#Verificar que se recibió un archivo
if [ $# -ne 1 ]; then
   echo "Error de uso, falta el archivo de usuarios" >&2
   mostrar_uso
fi

archivo_usuarios="$1"
```

```
#Verifico que el archivo existe
if [ ! -e "$archivo_usuarios" ]; then
   echo "Error de uso, el archivo '$archivo_usuarios' no existe." >&2
   exit $error_arch
fi
#Verifico que es un archivo
if [ ! -f "$archivo_usuarios" ]; then
   echo "Error de uso, '$archivo_usuarios' no es un archivo regular." >&2
   exit $error_arch
fi
#Verifico que tiene permiso de lectura
if [ ! -r "$archivo_usuarios" ]; then
   echo "Error de uso, no tiene permisos de lectura sobre '$archivo_usuarios'." >&2
   exit $error_perm
fi

#Proceso archivo línea a línea

#Cuento usarios creados
total_ok=0

#con "IFS" denoto el separador ":" que se usara en el formato del archivo y se le asigna a las
variables
while IFS=":" read -r usuario comentario dir_home crear_home shell_def; do
   #Cheque con el usuario exite como variable y si no existe con "continue" se omite los comandos
restantes dentro del bucle
      #para la iteración actual y pasa al siguiente iteración del bucle.
   [ -z "$usuario" ] && echo "ATENCION: usuario sin valor NO pudo ser creado" && echo "" &&
continue

      #Comprobamos si ya existe un usario creado con un "grep" silencioso, esto lo comprobamos
con un "for" para usar el comando "continue" para el while
      if grep -q "^${usuario}:" /etc/passwd; then
            comprobacion_usuario=1
      else
            comprobacion_usuario=0
      fi

      #Si existe un usuario, de una manera simiar nos saltaimos el bucle e informamos de esto
      [ "$comprobacion_usuario" -eq 1 ] && echo "ATENCION: usuario ya existe" && echo "" &&
continue

      #convertimos a "crear_home" en minuscula si o si, para poder comprobar en cualquier caso
que alla un "no"
   prueba_de_home="${crear_home,,}"
```

```
   #Similar al caso anterior utilizo un "test" verificando si el valor de crear_home es un "no" y si lo es saltamos los comandos restantes y se pasa de bucle
   [ ! -d "$dir_home" ] && [ "$prueba_de_home" = "no" ] && echo "ATENCION: el usuario $usuario no pudo ser creado, el directorio: $dir_home no existe" && echo "" && continue

   #Verifco los valores de las variables, si estos estan vacios los cambio con ":-" por "valor por defecto"
   comentario=${comentario:-"<valor por defecto>"}
   dir_home=${dir_home:-"<valor por defecto>"}
   crear_home=${crear_home:-"<valor por defecto>"}
   shell_def=${shell_def:-"/bin/bash"}


   #se construira el comando "useradd", para esto se asignara como una variable y dependiendo el contenido del archivo con los usarios se armara
   cmd="useradd"
   #Se susitura los valores quitados del archivo por cada uno de los valores por defecto de ser el caso
   [ "$comentario" != "<valor por defecto>" ] && cmd+=" -c \"$comentario\""
   [ "$dir_home" != "<valor por defecto>" ] && cmd+=" -d \"$dir_home\""
   [ "$shell_def" != "<valor por defecto>" ] && cmd+=" -s \"$shell_def\""

   #Crear directorio home si aplica
   if [ "$prueba_de_home" = "si" ]; then
      cmd+=" -m"
   fi

   cmd+=" $usuario"

   #Luego de construir el comando "useradd" y todas sus opciones, con "eval"
      #que permite usar un string como si fuera un comando se implementara al usuario
   eval $cmd 2>/dev/null
   if [ $? -eq 0 ]; then
      #Comprobamos que el comando "useradd" sea exitoso y sumamos una ejecucion de este
           ((total_ok++))

      #Asignar contraseña si corresponde con el comando "chpasswd" y envaiamos sus posibles errores a "/dev/null"
      if [ -n "$contrase" ]; then
         echo "$usuario:$contrase" | chpasswd 2>/dev/null
      fi

      if [ $desplegar_info -eq 1 ]; then
               #Desplegamos la informacion de creacion del usuario
         echo "Usuario $usuario creado con éxito con datos indicados:"
```

```
            echo "    Comentario: $comentario"
            echo "    Dir home: $dir_home"
            echo "    Asegurado existencia de directorio home: $crear_home"
            echo "    Shell por defecto: $shell_def"
            echo ""
        fi
    else
        if [ $desplegar_info -eq 1 ]; then
                #Damos un mesaje de error por si no se puede crear el usuario
            echo ""
                    echo "ATENCIÓN: el usuario $usuario no pudo ser creado"
            echo ""
        fi
    fi

done < "$archivo_usuarios"
#A el "for" cargamos el archivo que se le paso
if [ $desplegar_info -eq 1 ]; then
    echo "Se han creado $total_ok usuarios con éxito."
fi
#Final de cuenta de usuarios creados
```

6. Script de PYTHON

```
# ------------------------------------------------------------
# Descripción: Este script automatiza el despliegue completo de una aplicación web en AWS
# Autor: Guillermo Larrea, Fernando Sosa y ChatGPT (GPT-5)
# ------------------------------------------------------------

import os
import boto3
import time

from dotenv import load_dotenv

# Cargar las variables del archivo .env y creo variables globales
load_dotenv()

key_name = os.getenv('KEY_NAME')
aws_access_key_id = os.getenv('AWS_ACCESS_KEY_ID')
aws_secret_access_key = os.getenv('AWS_SECRET_ACCESS_KEY')
aws_session_token = os.getenv('AWS_SESSION_TOKEN')
aws_region = os.getenv('AWS_REGION', 'us-east-1')
aws_image_id = os.getenv('AWS_IMAGE_ID')
aws_instance_type = os.getenv('AWS_INSTANCE_TYPE', 't2.micro')
aws_s3_name = os.getenv('AWS_S3_NAME')
```

```
aws_ec2_name = os.getenv('AWS_EC2_NAME')
sg_ec2_name = os.getenv('SG_EC2_NAME')
sg_db_name = os.getenv('SG_DB_NAME')
db_identifier = os.getenv('DB_IDENTIFIER')
db_instance_class = os.getenv('DB_INSTANCE_CLASS')
db_engine = os.getenv('DB_ENGINE')
db_username = os.getenv('DB_USER_NAME')
db_password = os.getenv('DB_PASSWORD')
db_name = os.getenv('DB_NAME')
app_user = os.getenv('APP_USER')
app_pass = os.getenv('APP_PASS')

# Crea y devuelve un cliente EC2 autenticado con las credenciales del .env
def crear_cliente_ec2():
    ec2 = boto3.client(
        'ec2',
        aws_access_key_id=aws_access_key_id,
        aws_secret_access_key=aws_secret_access_key,
        aws_session_token=aws_session_token,
        region_name=aws_region
    )
    return ec2                                          #devuelve el cliente

# Crea el key pair en AWS y evita duplicados, ademas almacena el .pem localmente
def crear_par_de_claves(ec2):
    try:
        key_pair = ec2.create_key_pair(KeyName=key_name)            #crea el key pair en aws
        with open(f'{key_name}.pem', 'w') as file:                  #guarda el .pem localmente
            file.write(key_pair['KeyMaterial'])
            os.chmod(f'{key_name}.pem', 0o400)                      #asigna permisos solo lectura
        print(f"Par de claves creado y guardado como {key_name}.pem")
    except ec2.exceptions.ClientError as e:
        if 'InvalidKeyPair.Duplicate' in str(e):                    #verifica si ya existe
            print(f"La clave {key_name} ya existe")
        else:
            raise

# Crea el Security Group para EC2, si existe obtiene su ID
def crear_grupo_seguridad_ec2(ec2):
    try:
        response = ec2.create_security_group(GroupName=sg_ec2_name, Description="Grupo de
seguridad para mi ec2")
        sg_ec2_id = response['GroupId']                             #Obtengo el ID del sg
        print(f"Grupo de seguridad EC2 creado con el id: {sg_ec2_id}")
        return sg_ec2_id                                    #devuelvo el ID del sg
    except ec2.exceptions.ClientError as e:
        if 'InvalidGroup.Duplicate' in str(e):                      #si ya existe el grupo
            print(f"El grupo de seguridad {sg_ec2_name} ya existe")
            response = ec2.describe_security_groups(
```

```
                Filters=[                       #busco el SG existente
                  {
                    'Name': 'group-name',
                    'Values': [sg_ec2_name]
                  }
                ])
        if response['SecurityGroups']:                  #Valido si la busqueda devolvio algo
            for sg in response['SecurityGroups']:
                sg_ec2_id = sg['GroupId']               #Extraigo ID del grupo existente
                print(f"    ID del grupo de seguridad: {sg_ec2_id}")
                return sg_ec2_id                        #devuelvo el id encontrado
        else:
            print("No se encontraron grupos de seguridad con los filtros especificados.")
    else:
        raise

# Gestiona el Security Group de la base de datos (crea o reutiliza el existente)
def crear_grupo_seguridad_db(ec2):
    try:
        response = ec2.create_security_group(GroupName=sg_db_name, Description="Grupo de
seguridad para la base de datos")
        sg_db_id = response['GroupId']                  #Obtengo el ID del SG creado
        print(f"Grupo de seguridad DB creado con el id: {sg_db_id}")
        return sg_db_id                                 #Devuelvo el ID
    except ec2.exceptions.ClientError as e:
        if 'InvalidGroup.Duplicate' in str(e):          #Si el SG ya existe
            print(f"El grupo de seguridad de db {sg_db_name} ya existe")
            response = ec2.describe_security_groups(
                Filters=[                               #Busco el SG existente
                    {
                        'Name': 'group-name',
                        'Values': [sg_db_name]
                    }
                ])
            if response['SecurityGroups']:              #verifico resultados
                for sg in response['SecurityGroups']:
                    sg_db_id = sg['GroupId']            #Obtengo el ID del SG existente
                    print(f"    ID del grupo de seguridad: {sg_db_id}")
                    return sg_db_id                     #Devuelvo el ID
            else:
                print("No se encontraron grupos de seguridad con los filtros especificados.")
        else:
            raise

# Abre puertos (22/80) en EC2 y permite a EC2 acceder a DB (3306)
def crear_reglas_de_seguridad(ec2, sg_ec2_id, sg_db_id):    #Crea reglas para los grupos
    try:
        ec2.authorize_security_group_ingress(
            GroupId=sg_ec2_id,                          #En el SG de EC2 agrego reglas
```

```
            IpPermissions=[
                {
                    'IpProtocol': 'tcp',
                    'FromPort': 22,
                    'ToPort': 22,
                    'IpRanges': [{'CidrIp': '0.0.0.0/0'}]
                },                              #Habilito SSH desde cualquier IP
                {
                    'IpProtocol': 'tcp',
                    'FromPort': 80,
                    'ToPort': 80,
                    'IpRanges': [{'CidrIp': '0.0.0.0/0'}]
                }                               #Habilito HTTP desde cualquier IP
            ]
        )
    except ec2.exceptions.ClientError as e:
        if 'InvalidPermission.Duplicate' in str(e):     #Si ya existen aviso en pantalla
            print("Las reglas de seguridad de ec2 ya existen")
        else:
            raise

    try:
        ec2.authorize_security_group_ingress(
            GroupId=sg_db_id,                   #En el SG de la DB agrego regla
            IpPermissions=[
                {
                    'IpProtocol': 'tcp',
                    'FromPort': 3306,
                    'ToPort': 3306,
                    'UserIdGroupPairs': [{'GroupId': sg_ec2_id}]
                }                               #Habilito 3306 solo desde mi EC2
            ]
        )
    except ec2.exceptions.ClientError as e:
        if 'InvalidPermission.Duplicate' in str(e):     #Si ya existen aviso en pantalla
            print("Las reglas de seguridad de db ya existen")
        else:
            raise

# Crea cliente RDS para administrar instancias de base de datos
def crear_cliente_rds():
    rds = boto3.client(
        'rds',                          #Servicio RDS
        aws_access_key_id=aws_access_key_id,
        aws_secret_access_key=aws_secret_access_key,
        aws_session_token=aws_session_token,
        region_name=aws_region
    )
    return rds                          # Devuelve cliente RDS
```

```
# Consulta RDS y devuelve el endpoint público de la DB
def obtener_endpoint_rds(rds):
    """Obtener el endpoint de la base de datos RDS"""
    try:                                #Consulta la instancia buscando primera coincidencia
        response = rds.describe_db_instances(DBInstanceIdentifier=db_identifier)
        db_instance = response['DBInstances'][0]
        return db_instance['Endpoint']['Address']    #Devuelve endpoint
    except Exception as e:
        print(f"Error obteniendo endpoint de RDS: {e}")
        return None

# Crea la instancia RDS o usa la existente y devuelve su endpoint
def crear_base_de_datos(rds, sg_db_id):
    try:
        response = rds.create_db_instance(
            DBInstanceIdentifier=db_identifier,    #nombre
            AllocatedStorage=20,                   #tamaño en GB
            DBInstanceClass=db_instance_class,     #tipo de instancia
            Engine=db_engine,                      #motor de db
            MasterUsername=db_username,            #usuario admin
            MasterUserPassword=db_password,        #contraseña admin
            VpcSecurityGroupIds=[sg_db_id]         #SG asignado a la DB
        )
        print("Instancia RDS creada")
        print("Esperando que quede disponible ...")
        waiter = rds.get_waiter('db_instance_available')       #espera que este lista
        waiter.wait(DBInstanceIdentifier=db_identifier)
        print("Ahora la db ya está pronta")
        endpoint = obtener_endpoint_rds(rds)                   #obtengo endpoint
        if endpoint:
            print(f"Endpoint de la base de datos: {endpoint}")
            return endpoint
        return None
    except rds.exceptions.ClientError as e:
        if 'DBInstanceAlreadyExists' in str(e):                #si ya existe la DB
            print("La instancia de base de datos ya existe")
            endpoint = obtener_endpoint_rds(rds)               #obtengo endpoint existente
            if endpoint:
                print(f"Endpoint de la base de datos existente: {endpoint}")
                return endpoint
            return None
        else:
            raise

# Crea y devuelve un cliente S3 con credenciales AWS
def crear_cliente_s3():
    s3_client = boto3.client(
        's3',
```

```
        aws_access_key_id=aws_access_key_id,
        aws_secret_access_key=aws_secret_access_key,
        aws_session_token=aws_session_token,
        region_name=aws_region
    )
    return s3_client          #Devuelve cliente s3

# Sube archivos de la app al bucket S3, omitiendo existentes
def subir_app_a_s3(s3_client):
    try:
        s3_client.create_bucket(Bucket=aws_s3_name)       #Creo el bucket s3
        print(f"Bucket {aws_s3_name} creado")
    except (s3_client.exceptions.BucketAlreadyExists,
            s3_client.exceptions.BucketAlreadyOwnedByYou): #si ya existe
        print(f"El bucket {aws_s3_name} ya existe")

    archivos_a_incluir = [
        'app.css', 'app.js', 'config.php', 'index.html',
        'index.php', 'init_db.sql', 'login.css', 'login.html',
        'login.js', 'login.php'
    ]                                       #Archivos que se subiran

    for archivo in archivos_a_incluir:
        if os.path.exists(archivo):                 #Confirma si estan los archivos
            try:                                    #Busca si ya existen en la s3
                s3_client.head_object(Bucket=aws_s3_name, Key=archivo)
                print(f"    {archivo} ya existe en S3, omitiendo")
                continue
            except:
                pass                                #si el archivo no existe lo sube

            s3_client.upload_file(archivo, aws_s3_name, archivo)
            print(f"    {archivo} subido a S3")
        else:
            print(f"    {archivo} no existe, omitiendo")   #si el archivo no lo tengo localmente

# Crea un cliente EC2 tipo resource para manipular instancias
def crear_cliente_ec2_resource():
    ec2_resource = boto3.resource(
        'ec2',                          #Servicio ec2 en modo resource
        aws_access_key_id=aws_access_key_id,
        aws_secret_access_key=aws_secret_access_key,
        aws_session_token=aws_session_token,
        region_name=aws_region
    )
    return ec2_resource                 #Devuelve resource de ec2

# crear o reutilizar una instancia EC2 en AWS y preparar la aplicación web para que quede
disponible
```

```
def crear_instancia_ec2(ec2_resource, sg_ec2_id, db_endpoint):

    user_data_script = generar_user_data(db_endpoint)    # Ejecuta script de Bash al iniciar EC2

    existing_instances = list(ec2_resource.instances.filter(
        Filters=[                                # Busca instancias creadas
            {'Name': 'tag:Name', 'Values': [aws_ec2_name]},
            {'Name': 'instance-state-name', 'Values': ['running', 'stopped', 'pending']}
        ]
    ))

    if existing_instances:
        instance = existing_instances[0]              #Usa la instancia existente
        instance_id = instance.id                     #Id de la instancia
        print(f"La instancia EC2 '{aws_ec2_name}' ya existe con ID: {instance_id}")
        instance.reload()                             #Actualiza datos
        ip_publica = instance.public_ip_address       #Obtengo IP publica y la muestro
        print(f"App disponible en http://{ip_publica}/login.php")
        return instance_id                            #Retorno la id

    try:
        instances = ec2_resource.create_instances(
            ImageId=aws_image_id,                     #AMI usada
            MinCount=1,
            MaxCount=1,                               #1 sola instancia
            InstanceType=aws_instance_type,           #tipo
            KeyName=key_name,                         #Par de claves
            SecurityGroupIds=[sg_ec2_id],             #SG asignado
            UserData=user_data_script,                #Script de inicializacion
            TagSpecifications=[
                {
                    'ResourceType': 'instance',
                    'Tags': [{'Key': 'Name', 'Value': aws_ec2_name}]  #Nombre en tags
                }
            ]
        )
        instance_id = instances[0].id
        print("Instancia creada con ID:", instance_id)
        print("    Esperando a que inicie instancia")
        instance = instances[0]
        instance.wait_until_running()                 #Espera estado de "running"
        instance.reload()                             #actualiza info

        print("Instancia iniciada, desplegando la APP")  #Espera 40s mientras despliega app
        time.sleep(40)

        ip_publica = instance.public_ip_address
        print(f"APP en: http://{ip_publica}/login.php")
        return instance_id
```

```
    except Exception as e:
        print(f"Error creando instancia EC2: {e}")       #Muestra si surge un error
        raise

# Genera el script de Bash al inicializar la instancia
def generar_user_data(db_endpoint):
    return f"""#!/bin/bash

export AWS_ACCESS_KEY_ID={aws_access_key_id}
export AWS_SECRET_ACCESS_KEY={aws_secret_access_key}
export AWS_SESSION_TOKEN={aws_session_token}
export AWS_REGION={aws_region}
export S3_BUCKET={aws_s3_name}

#actualiza sistema e instala paquetes
sudo dnf clean all
sudo dnf makecache
sudo dnf -y update
sudo dnf -y install httpd php php-cli php-fpm php-common php-mysqlnd mariadb105 mariadb105-
server-utils.x86_64

#configura e inicia servicios
sudo systemctl enable --now httpd
sudo systemctl enable --now php-fpm
sudo systemctl restart httpd php-fpm

#copia aplicacion desde el s3
aws s3 cp s3://{aws_s3_name}/ /var/www/html --recursive
mv /var/www/html/init_db.sql /var/www

#Archivo de prueba
echo "<?php phpinfo(); ?>" | sudo tee /var/www/html/info.php

# Importar base de datos a RDS
mysql -h {db_endpoint} -u {db_username} -p{db_password} < /var/www/init_db.sql

# Crear archivo .env con la configuración
tee /var/www/.env > /dev/null <<EOF
DB_HOST={db_endpoint}
DB_NAME={db_name}
DB_USER={db_username}
DB_PASS={db_password}

APP_USER={app_user}
APP_PASS={app_pass}
EOF

# Configurar permisos seguros
sudo chown apache:apache /var/www/.env
```

```
sudo chmod 600 /var/www/.env

# Configurar permisos generales
sudo chown -R apache:apache /var/www/html
sudo chmod -R 755 /var/www/html

# Reiniciar servicios para aplicar cambios
sudo systemctl restart httpd php-fpm
"""

if __name__ == "__main__":
    cliente_ec2 = crear_cliente_ec2()
    crear_par_de_claves(cliente_ec2)
    sg_ec2_id = crear_grupo_seguridad_ec2(cliente_ec2)
    sg_db_id = crear_grupo_seguridad_db(cliente_ec2)
    crear_reglas_de_seguridad(cliente_ec2, sg_ec2_id, sg_db_id)
    cliente_rds = crear_cliente_rds()
    db_endpoint = crear_base_de_datos(cliente_rds, sg_db_id)
    cliente_s3 = crear_cliente_s3()
    subir_app_a_s3(cliente_s3)
    ec2_resource = crear_cliente_ec2_resource()
    instance_id = crear_instancia_ec2(ec2_resource, sg_ec2_id, db_endpoint)
```