

# Universidad ORT Uruguay

## Facultad de Ingeniería: Escuela de Tecnología

# PROGRAMACION PARA DEVOPS

**Marcelo Sosa N°141855**

**Hernán Pintos N°151357**

**Grupo: N4A**

**Docentes: Leonardo Genta - Guillermo Ferres**

## Declaración de autoría

Nombres de los autores: Marcelo Sosa  
Hernán Pintos

Declaración de autoría Nosotros, **Marcelo Sosa** y **Hernán Pintos**, declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Aseguramos, bajo nuestra entera responsabilidad, que:

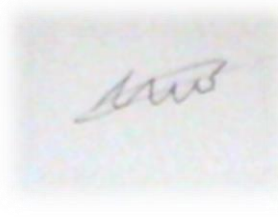
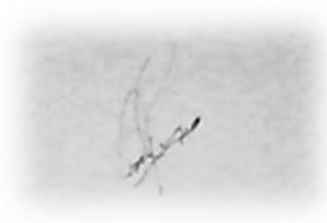
- La obra fue producida en su totalidad mientras realizábamos la carrera ANALISTA EN INRAESTRUCTURA INFORMATICA;- Cuando hemos consultado trabajos publicados por otros, lo hemos atribuido con claridad;- Cuando hemos citado obras de otros autores, hemos indicado las fuentes. Con excepción de dichas citas, la obra es enteramente nuestra;- En la obra, hemos acusado recibo de las ayudas recibidas;- Cuando la obra se basa en trabajo realizado juntamente con otros no pertenecientes al equipo, hemos explicado claramente que parte fue contribuida por dichos terceros, y que parte fue contribuida por nosotros;- Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto los casos en que se han realizado las aclaraciones correspondientes.



Hernán Pintos



Marcelo Sosa



### REPOSITORIO GIT

<https://github.com/ortdevops2025/obligatorio2025>

---

## **INDICE**

## **SECCION I - OBJETIVOS**

El objetivo de este documento es dar cumplimiento a las tareas solicitadas por los docentes en la materia de programación para DevOps

Esto se realizará cumpliendo los siguientes ítems:

Creación de script de Bash que cumpla el primer requerimiento de la letra

Creación de script de Python que cumpla requerimientos de AWS pedidos para la segunda parte de la letra del obligatorio

Documento de registros de ejecución e información

Reflexiones sobre los desafíos planteados

Para lograr estos objetivos, utilizaremos los conocimientos y herramientas obtenidos en la materia a lo largo del semestre.

## SECCION II – USO DE GIT Y RECURSOS

Para la realización del trabajo obligatorio se utilizó GIT para control de cambios y trabajo en equipo.

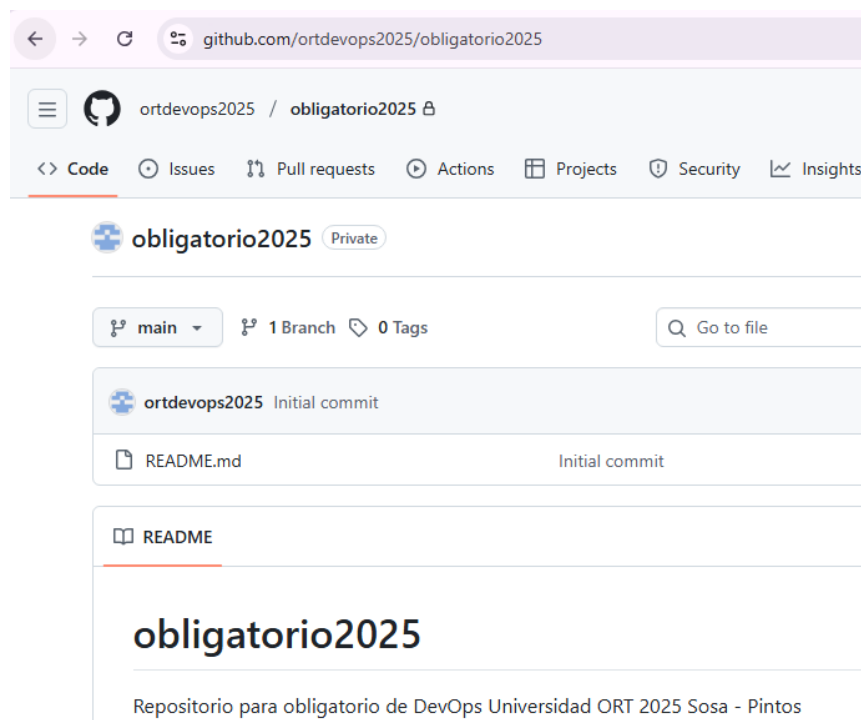
Si bien todos los documentos están en el formato solicitado en el repositorio de GIT creado con la finalidad de la realización de este trabajo, se incluye en este documento, en las siguientes secciones, el código de ambos scripts, pruebas de uso y contenido de archivos de ejemplo así como también el ambiente en donde las pruebas fueron realizadas.

Se utilizó también GITHUB Desktop para la carga de documentación y mantenimiento de archivos.

### REPOSITORIO DE GIT

Para la realización de este trabajo obligatorio, el siguiente repositorio de GIT fue creado:

<https://github.com/ortdevops2025/obligatorio2025>



Para evitar filtraciones o acceso no permitido, este repositorio fue puesto como privado, solo los usuarios del obligatorio y los docentes tendrán acceso al mismo para la realización del trabajo y posteriormente la defensa del mismo.



### SECCION III – SCRIPT DE BASH

Para la realización y pruebas de este script fue utilizada la máquina virtual de CentOS 8 proporcionada en el curso con algunas actualizaciones.

Version de CentOS:

```
[alumno@asl final]$ cat /etc/centos-release
CentOS Linux release 8.5.2111
[alumno@asl final]$
```

Se muestra a continuación el script que esta adjunto en el repositorio.

```
#####

#!/bin/bash

flaginfo=0      # Variable para -i vale distinto de 0 se muestra flaginforamcion
password=""     # Variable para -c que guarda el password
archivousu=""   # archivo con la lista de usuarios
contadorusuok=0 # Contador de usuarios creados OK

#####
#####Comprobacion de Parametros#####
#####

if [ "$1" = "-i" ] && [ "$2" = "-c" ] #Verifica el orden de los parametros i y c
then
    flaginfo=1
    password="$3"
    archivousu="$4"
else
    if [ "$1" = "-c" ] && [ "$3" = "-i" ] #Verifica segundo orden posible de los parametros i y c
    then
        flaginfo=1
        password="$2"
        archivousu="$4"
    else
        if [ "$1" = "-i" ] #Verifica si se utilizo solo el parametro de informacion
        then
            flaginfo=1
            archivousu="$2"
        else
            if [ "$1" = "-c" ] #Verifica si se utilizo solo el parametro de password
            then
                password="$2"
                archivousu="$3"
            else
                archivousu="$1" #Procsa el archivo si no se utilizaron parametros
            fi
        fi
    fi
fi

#####
#####VALIDAR archivousu #####
#####

if [ ! $# -ge 1 ] #Verifica que se usen al menos un parametro en el script
then
    echo "Debe proporcionar al menos un parametro"
```

```

        exit 2
fi

if [ ! -f "$sarchivousu" ] #Verifica que el archivo sea regular
then
    echo "$sarchivousu no existe o no es regular" >&2
    exit 3
fi

if [ ! -r "$sarchivousu" ] #Verifica que el archivo tenga read only
then
    echo "$sarchivousu no tiene permiso de lectura" >&2
    exit 4
fi

if [ "$(id -u)" -ne 0 ] #IAG Geneardo, verifica si es root el usuario al momento de ejecucion
then
    echo "Este script debe ejecutarse como root" >&2
    exit 5
fi

#####
#####DESEMPAQUETADO DE archivousu#####
#####

IFS=$'\n' #Defino separador de lineas como el salto de linea

for i in $(cat "$sarchivousu")
do

#verificador de campos de linea para que sean 5

    campos=$(echo "$i" | tr -cd ':' | wc -c) #Cuenta los delimitadores : para verificar sintaxis de la linea

    if [ "$campos" -ne 4 ]
    then
        echo "Sintaxis incorrecta del $sarchivousu pasado como parametro" >&2
        echo "La linea $i no contiene exactamente 5 campos separados por : " >&2

    else

        #Asigna valores a las variables

        USUARIO=$(echo "$i" | cut -d: -f1)
        COMENTARIO=$(echo "$i" | cut -d: -f2)
        HOME=$(echo "$i" | cut -d: -f3)
        CREARHOME=$(echo "$i" | cut -d: -f4 | tr '[:lower:]' '[:upper:]')
        SHELL=$(echo "$i" | cut -d: -f5-)

        # Comprobacion de existencia de usuario

        if id "$USUARIO" >/dev/null 2>&1
        then
            echo "ATENCION: el usuario $USUARIO ya existe"
            continue
        fi

        # Valores por defecto

        [ -z "$SHELL" ] && SHELL="/bin/bash"
        [ -z "$HOME" ] && HOME="/home/$USUARIO"
        [ -z "$CREARHOME" ] && CREARHOME="SI"

        # Creacion del usuario

```

```

if [ "$CREARHOME" = "SI" ]
then
    useradd -m -c "$COMENTARIO" -d "$HOME" -s "$SHELL" "$USUARIO" >/dev/null 2>&1
else
    useradd -M -c "$COMENTARIO" -d "$HOME" -s "$SHELL" "$USUARIO" >/dev/null 2>&1
fi

if [ $? -eq 0 ] #verifica que el comando de agregar usuario haya sido exitoso
then
    if [ -n "$password" ]
    then
        echo "$USUARIO:$password" | chpasswd 2>/dev/null
    fi

    contadorusuok=$((contadorusuok + 1)) #Sumo uno si se agrego el usuario correctamente

    if [ "$flaginfo" -eq 1 ]; then #Muestra informacion si se uso el parametro -i

        echo "Usuario $USUARIO creado con éxito con datos indicados:"
        echo "Comentario: ${COMENTARIO:-<valor por defecto>}"
        echo "Dir home: ${HOME:-<valor por defecto>}"
        echo "Asegurado existencia de directorio home: ${CREARHOME:-<valor por defecto>}"
        echo "Shell por defecto: ${SHELL:-<valor por defecto>}"
        echo

    fi
else
    echo "ATENCION: el usuario $USUARIO no pudo ser creado"
fi
fi
done < "$Sarchivousu"

#####
###INFORMACION A MOSTRAR###
#####

if [ "$flaginfo" -eq 1 ]; then
    echo "Se han creado $contadorusuok usuarios con éxito."
fi

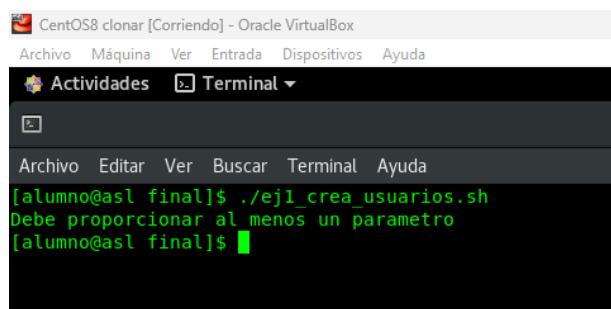
```

#####

## PRUEBA DE USO:

Se procede a mostrar ejemplos de uso y los resultados obtenidos incluidos algunos errores testeados:

### 1 – Script sin Parametros:



En este ejemplo, al no colocarse ningún parámetro el script devuelve un resultado de error indicando que el script necesita al menos un parámetro para funcionar (el archivo)



## 2 – Archivo no regular

```

Archivo  Editar  Ver  Buscar  Terminal  Ayuda
[alumno@asl final]$ ./ej1_crea_usuarios.sh sinarchivo
sinarchivo no existe o no es regular
[alumno@asl final]$

```

3 – Uso normal con un archivo de usuarios que tiene 3 usuarios validos para agregar y uno que ya existe:

```

pepe:Este es mi amigo pepe:/home/jose:SI:/bin/bash
terran:Jim_Raynor:/home/raynor:NO:/bin/sh
elmaligno:::/bin/el_maligno
root:::/bin/root

```

```

[alumno@asl final]$ sudo ./ej1_crea_usuarios.sh -i -c "123456" usuarios
[sudo] password for alumno:
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

Usuario terran creado con éxito con datos indicados:
Comentario: Jim_Raynor
Dir home: /home/raynor
Asegurado existencia de directorio home: NO
Shell por defecto: /bin/sh

Usuario elmaligno creado con éxito con datos indicados:
Comentario: <valor por defecto>
Dir home: /home/elmaligno
Asegurado existencia de directorio home: SI
Shell por defecto: /bin/el_maligno

ATENCION: el usuario root ya existe
Se han creado 3 usuarios con éxito.
[alumno@asl final]$

```

En este caso, se deberán de agregar tres usuarios, pepe, Jim Raynor y el maligno.

El usuario root ya existe por lo que el comando debería avisar de esta situación.

Se utilizan los dos parámetros opcionales también -i y -c con un password 123456

Se observa en la imagen el resultado obtenido.

Archivo /etc/passwd mostrando los nuevos usuarios agregados

```

cockpit-wsinstance:x:977:975:User for cockpit-ws instances:/none
flatpak:x:976:974:User for flatpak system helper:/:/sbin/nologin
rngd:x:975:973:Random Number Generator Daemon:/var/lib/rngd:/sbin
pruebal:x:1001:1001:Primera prueba:/home/prueba:/bin/bash
terran:x:1003:1003:Jim_Raynor:/home/raynor:/bin/sh
pepe:x:1004:1004:Este es mi amigo pepe:/home/jose:/bin/bash
elmaligno:x:1005:1005::/home/elmaligno:/bin/el_maligno
[alumno@asl final]$

```

## SECCION IV – SCRIPT DE PYTHON

Para la creacion de este script y su ejecución, se deben de satisfacer ciertos requerimientos especiales relacionados con el ambiente en donde el script se va a utilizar.

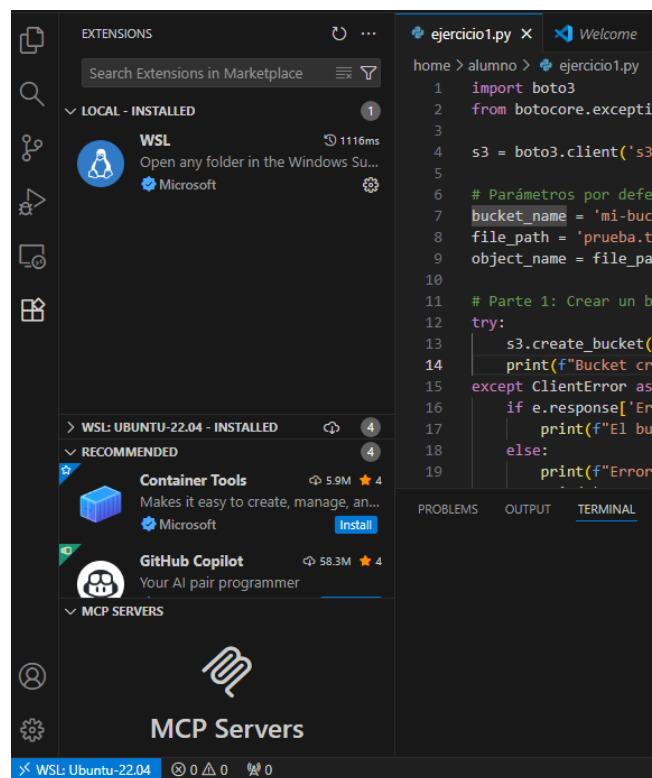
En este caso, se configuro para el desarrollo una instancia de WSL en Windows 11 utilizando la imagen de Ubuntu 22 proporcionada por el repositorio de WSL.

Se instalo el siguiente software una vez configurado el ambiente de WSL:

- Python3
- Boto3
- Pip3
- AWS CLI
- Git

```
alumno@PCW11: ~  
alumno@PCW11:~$ pip3 show boto3  
Name: boto3  
Version: 1.40.68  
Summary: The AWS SDK for Python  
Home-page: https://github.com/boto/boto3  
Author: Amazon Web Services  
Author-email:  
License: Apache-2.0  
Location: /home/alumno/.local/lib/python3.10/site-packages  
Requires: botocore, jmespath, s3transfer  
Required-by:  
alumno@PCW11:~$ aws --version  
aws-cli/2.31.31 Python/3.13.9 Linux/6.6.87.2-microsoft-standard-WSL2 exe/x86_64.ubuntu.22  
alumno@PCW11:~$
```

Para el desarrollo de los script, se utilizo Visual Studio Code con los plugins de Boto3, WSL y Python instalados.



## **Configuración de GIT**

Se utilizo la versión de GIT para Windows Desktop y también la versión GIT en WSL Ubuntu. Para la configuración de GIT en WSL se procedió a la clonación inicial del repositorio:

```
git clone https://github.com/ortdevops2025/obligatorio2025.git
```

NOTA: PARA LA EJECUCION DE LOS SCRIPTS, SE TOMA EN CUENTA QUE LOS ARCHIVOS ESTAN EN LA MAQUINA DONDE SE CORRERAN LOS MISMOS, POR LO TANTO EL REPOSITORIO DEBE SER CLONADO A LA MISMA COMO SE MUESTRA A CONTINUACION.

La carpeta de aplicación esta ya contenida dentro del repositorio Oligatorio2025.git

## **INSTRUCCIONES DE EJECUCION PYTHON**

Los siguientes pasos son necesarios para el correcto despliegue y ejecución de la aplicación.

Tener AWS CLI configurado / credenciales disponibles en la máquina donde ejecute los scripts.

Suponiendo que se ejecute este proyecto desde una maquina de LINUX, se debe tener el AWS CLI instalado y confiogurado, Python 3, Pip, Boto3.

Adicionalmente debemos contar con la KEY de EC2 para el acceso de SSH si es necesario y un rol que nos permita acceder al S3.

Para ello, como ejemplo en Ubuntu AWS el cual fue utilizado ejecutamos:

```
Sudo apt update  
Sudo apt upgrade  
sudo apt install python3  
sudo apt install python3 python3-pip -y  
pip3 install boto3  
sudo apt install unzip  
pip3 install --upgrade --user awscli
```

Comprobacion de versiones usando como ejemplo las utilizadas:

```
apt show awscli | grep Version  
aws-cli/2.13.24 Python/3.11.5 Linux/6.8.0-31-generic exe/x86_64.ubuntu.22 prompt/off
```

Tambien debemos instalar GIT para poder clonar el repositorio:

```
sudo apt install git
```

Configuramos el AWS CLI (En caso de ser necesario, con las credenciales del ambiente a utilizar)

Se utiliza el comando

```
aws configure
```

Se pedirán los siguientes campos los cuales salen de la sesión de AWS a utilizar:

AWS Access Key ID

AWS Secret Access Key

AWS Session Token

Region Name: us-east-1 (si corresponde esa region)

Default Format: json

A continuación se ve un ejemplo de la configuración:

```
alumno@PCW11:~$ aws configure
AWS Access Key ID [*****VTFU]: ASIAXSYCAGFDJQ3YUD5R
AWS Secret Access Key [*****IoLC]: gh/130wAOq1ZsTu4nmXJzCZ6BDG3I80Tzwa0Tkyj
AWS Session Token [*****XYnP]: IQoJb3JpZ2luX2VjEF8aCXVzLXdlc3QzMjI0YVYyNDcwIgzEry7A6KgGMOcD4F4qkgIvldVfGEyXTM7jsRIw+ck+9bycLZGuZC00BY8wLhBpSBBz1AUVDem0sJtVwptStGXqDPLWA1
fn8SqAaUCrcYZn7syo1oLy8cwZKzUCCCGQARoMNTIxHzA10DYyNDcwIgzEry7A6KgGMOcD4F4qkgIvldVfGEyXTM7jsRIw+ck+9bycLZGuZC00BY8wLhBpSBBz1AUVDem0sJtVwptStGXqDPLWA1
sycY8aghlwaYsLfghkH/7F3t0Fbo6Pg0Buh6qhJhAU/TS17xwWkj6s+VVIYLq8fyTHAQuBgdKYaUIDBH/vzz7w02LVv3tFRT8X1LEZSM8hb7LI2pg5oZrk54e4kZerNs8a0MwTeJG5Yyny6qXDMB
vejTORhGYqxwBYn5SAdLFxPqz3NckVyTOTE17VtmW308qskvWgEQ8tyD/HoCQfuzvjWYEVHfKZ3em8UjqdSSMT9f3sWTC5nFkvUtBu9Lsr/RGLYppEafZcYhPtgC/4hQ+yQuEuTnvnC9tC8FMK
j2zsgG0pmB4L8/TLsvDHI8paHMHqTG2K0SDt35bGLIEtE9xIqLU8TOWV2az3jQzZyFz7WHHrVsr9748xbjZFnmVXYHGyQwLnuqJMj8R622mPrCcHyLcXIRNmiLQC/hxEJX1RXb25xeCCKCvHwT0
/3KsmhGcQW7DuY3rNYb0iJCGAIUCgoooqphPteVm0o3fJ8QdGAr53p9YkvYLhBcclqQX2sU
Default region name [us-east-1]: us-east-1
Default output format [json]: json
```

En caso de que debamos alterar algún valor, se puede correr el comando nuevamente o acceder al archivo de configuración el cual esta en texto plano en la ubicación:

```
/home/usuario/.aws/credentials
```

Instalación de agente de SSM de AWS en Ubuntu

```
sudo snap install amazon-ssm-agent --classic
```

Una vez teniendo todo configurado, clonamos el repositorio siguiente:

```
git clone https://github.com/ortdevops2025/obligatorio2025.git
```

## PASOS PARA DESPLIEGUE DE APLICACIÓN USANDO SCRIPTS:

Una vez clonado el repositorio e instalado las dependencias de la sección anterior, se debe proceder al despliegue de la instalación.

NOTA: Todos los comandos se deben ejecutar desde la carpeta /obligatorio2025, la cual es la raíz del repositorio clonado.

Desde el Directorio del repositorio clonado, ejecutar el script de instalacion y despliegue de la aplicación:

**python3 obligatorio.py**

```
alumno@PCW11:~/obligatorio2025$ python3 experimental.py
Bucket creado: obligatorio2025qwertyuiop
Archivos subidos correctamente.
Ingresar password maestro de la base:
Obligatorio2025
SG EC2 ya existe: sg-06b0be5f05155fd67
SG RDS ya existe: sg-05f277b4eb7a7ff12
La instancia RDS ya existe.
Esperando a que RDS esté listo...
RDS listo.
Endpoint RDS: app-mysql.cxzqtmaexaa8.us-east-1.rds.amazonaws.com
Instancia EC2 creada: i-050cfec16adf02ae7
Esperando a que EC2 este listo
EC2 esta listo, ya puede navegar en el sitio
SG listos y asignados a EC2 y RDS
Navegue a la IP pública del EC2 cuando esté disponible
```

Output de Creacion de Bucket y subida de archiv

Password de la Base de Datos ingresado

Creacion de Security Groups

Creacion de la base de RDS

Creacion de instancia EC2

Salida final

## PRUEBA DE USO

Una vez finalizado el script, y la instancia de AWS correctamente iniciada, se podrá probar la aplicación ingresando al siguiente link:

**[http://\(IP PUBLICA DE INSTANCIA\)/login/php](http://(IP PUBLICA DE INSTANCIA)/login/php)**

Se utiliza el log in:

**admin**  
**admin123**

Se debe desplegar correctamente la información de la aplicación:

| ID | Nombre      | Email                   | Sueldo    | Puesto            | Acciones        |
|----|-------------|-------------------------|-----------|-------------------|-----------------|
| 1  | Juan Perez  | juan@example.com        | 55,000.00 | Desarrollador     | Editar Eliminar |
| 2  | Ana Gomez   | ana@example.com         | 60,000.00 | Analista          | Editar Eliminar |
| 3  | Carlos Ruiz | carlos.ruiz@example.com | 48,000.00 | Tester            | Editar Eliminar |
| 4  | Maria Lopez | maria.lopez@example.com | 70,000.00 | Lider de Proyecto | Editar Eliminar |

---

## **SECCION VII – RESULTADOS**

**DIFICULTADES ENCONTRADAS**

PROBLEMA DE USAR COMANDO QUE FUNCIONE EN UBUNTU Y CENTOS

PROBLEMA CON EL PASSWORD SOLUCIONADO CON IAG III

SE LLENABA DE USUARIOS EL SISTEMA OPERATIVO ASI QUE SE HIZO UN SCRIPT QUE REALIZARA LA ACCION CONTRARIA A AGREGAR

BASE DE DATOS NO FUNCIONABA PORQUE EL NOMBRE ERA DIFERENTE

INTENTAR SUBIR \$ ARCHIVOS DE PYTHON EN VEZ DE HACERLO EN UNO SOLO

## ANEXO I – Uso de Inteligencia Artificial Generativa

Para la realizacion de este trabajo, el uso de **IAG** fue utilizado EXCLUSIVAMENTE para la creacion de gráficos específicos o conjunto de comandos básicos con motivo de verificación.

A continuacion, se detalla la lista de prompts utilizados.

### IAG 1 – Sugerencia de estructura

<https://chatgpt.com/>

Prompt: Sugerir estructura estructura en Bash usando if que retorne ultimo comando y analice si es 0. Si es igual a 0 dejar libre para colocar un comando una accion, sino, devolver un mensaje de exito.

#### Estructura sugerida

```
bash

# Ejecutar algún comando
comando_que_deseas_ejecutar

# Evaluar el resultado del comando anterior
if [ $? -eq 0 ]; then
    # Acción a realizar si fue exitosa (colocar aquí)
    # Ejemplo: echo "Acción ejecutada correctamente."
else
    echo "El comando no se ejecutó correctamente."
fi
```

### IAG 2 – Sugerencia de estructura

<https://chatgpt.com/>

#### Descripción:

chpasswd (abreviatura de *change password*) permite establecer contraseñas para uno o varios usuarios de forma no interactiva, leyendo pares `usuario:contraseña` desde la entrada estándar. Es ideal para usarlo en scripts, ya que no necesita interacción manual (a diferencia de `passwd`).

#### Ejemplo dentro de un script

```
bash

#!/bin/bash

USUARIO="juan"
CLAVE="1234"

# Crear usuario
useradd "$USUARIO"

# Asignar contraseña automáticamente
echo "$USUARIO:$CLAVE" | chpasswd
```

Prompt: Comando en linux que sirva para colocar un apssword automaticamente dentro de un script de bash junto a useradd

### IAG 3 – Sugerencia de estructura

<https://chatgpt.com/>

```
if [ "$CREAR_HOME" = true ]; then
    # Crear usuario con su directorio home personalizado
    echo "Creando usuario $USUARIO con su directorio home..."
    $CMD --home "/home/$USUARIO" --shell "$SHELL_DEF" --gecos "$COMENTARIO" --disabled-password "$US
else
```

Prompt: Ejemplo de bloque en Bash que cree usuarios con el comando adduser, compatible en ubuntu y centos.

Suponer una variable true apra crear un home y un false entregaria un home por defecto.

Agregar en el ejemplo, el comentario de usuario

Utilizar if y else preferentemente



## IAG 4 – Sugerencia de estructura

<https://chatgpt.com/>

Prompt: Como asignar valores por defecto a \$SHELL o \$HOME

### 3 Método clásico con condicional [ -z ]

bash


```
[ -z "$SHELL" ] && SHELL="/bin/bash"
[ -z "$HOME" ] && HOME="/home/$USUARIO"
```

#### Qué hace

- Comprueba si la variable está vacía (-z → "zero length").
- Si lo está, asigna el valor indicado.

## IAG 5 – Orden de deploy

### Orden recomendado de los scripts

| Nº | Script                     | Descripción  |
|----|----------------------------|--|
| 1  | <code>create_sg.py</code>  | Crea los Security Groups necesarios (uno para EC2 y otro opcional para RDS).   |
| 2  | <code>create_s3.py</code>  | Crea el bucket S3 y sube los archivos de la aplicación (código fuente, <code>init_db.sql</code> , etc.).   |
| 3  | <code>create_rds.py</code> | Despliega la base de datos RDS (por ejemplo, MySQL).   |
| 4  | <code>create_ec2.py</code> | Lanza la instancia EC2 (capa web), con un <i>user-data</i> que: <ul style="list-style-type: none"> <li>• Instala Apache o Nginx</li> <li>• Descarga la app desde S3</li> <li>• Configura <code>config.php</code> o variables de entorno con el endpoint del RDS</li> <li>• Importa el <code>.sql</code> </li> </ul> |

<https://chatgpt.com/>

Prompt: En que orden debería implementar los scripts en python con boto3 para deployar una aplicación en dos capas, la idea principal es tener, uno para ec2, otro para base de datos RDS, otro de S3 y uno para security groups

## IAG 6 – Subida de S3 a EC2


<https://chatgpt.com/>

Prompt: como puedo subir todos los archivos de un S3 a un directorio de una instancia EC2 usando Python y boto3

### 💡 Tip adicional

Si vas a hacerlo frecuentemente, considera usar el comando nativo de AWS CLI (más eficiente):

bash

 Copiar código

```
aws s3 sync s3://mi-bucket/datos/ /home/ec2-user/descargas_s3/
```

## IAG 6 – Variable de entorno para password

<https://chatgpt.com/>

Prompt: Como agregar una variable de entorno  
RDS\_ADMIN\_PASSWORD en  
Python para ejecutar en linux

```
python

import boto3
import os

print('Ingresar password maestro de la base:')
RDS_ADMIN_PASSWORD = input().strip()

# Guardar en variable de entorno
os.environ['RDS_ADMIN_PASSWORD'] = RDS_ADMIN_PASSWORD
```

```
try:
    response = ec2.create_security_group(
        GroupName=rds_sg_name,
        Description='Permitir tráfico desde EC2 hacia RDS'
    )
    rds_sg_id = response['GroupId']
    print(f"SG RDS creado: {rds_sg_id}")

    ec2.authorize_security_group_ingress(
        GroupId=rds_sg_id,
        IpPermissions=[
            {
                'IpProtocol': 'tcp',
                'FromPort': 3306,
                'ToPort': 3306,
                'UserIdGroupPairs': [{'GroupId': sg_id}]
            }
        ]
    )
```

## IAG 7 – RDS Security Group

<https://chatgpt.com/>

Prompt: Optimizar el bloque de Script dado para el  
security group de RDS para el puerto 3306

## IAG 8 – Wait en instancias

<https://chatgpt.com/>

Prompt: Ejemplo para bloque de espera en RDS o EC2 para que la instancia este lista, usar get\_waiter

```
print("Esperando a que RDS esté listo...")
waiter = rds.get_waiter('db_instance_available')
waiter.wait(DBInstanceIdentifier=DB_INSTANCE_ID)
print("RDS listo.")

# Obtener endpoint
rds_info = rds.describe_db_instances(DBInstanceIdentifier=DB_INSTANCE_ID)
RDS_ENDPOINT = rds_info['DBInstances'][0]['Endpoint']['Address']
print(f"Endpoint RDS: {RDS_ENDPOINT}")
```

IAG 9 – Subida de S3 a EC2<https://chatgpt.com/>

Prompt: Bloque en bash que sirva para esperar que una conexión a una base de datos sea satisfactoria con RDS

```
for i in {1..20}; do
    mysql -h {RDS_ENDPOINT} -u {DB_USER} -p{DB_PASS} -e "SELECT 1;" {DB_NAME} && break
    sleep 5
done
```

IAG 9 – Subida de S3 a EC2<https://chatgpt.com/>

Prompt: Generar banner que tenga el símbolo y y el texto BASH con algún elemento discreto característico de starcraft

## **ANEXO II**

---

## **BIBLIOGRAFIA**