Documentación del Obligatorio

Programación para DevOps

Diego García Lacuesta N.º 331576

Carlos Pérez Nº 179303

Grupo: N4A

Docente: Leonardo Genta
Asistente de docente: Guillermo Ferradas

Índice

Tabla de contenido

Datos del grupo

- Diego García Lacuesta.
- Nro. Estudiante: 331576.

- Carlos Pérez Borgarelli.
- Nro. Estudiante: 179303.

Objetivo del trabajo

El objetivo del siguiente trabajo es desarrollar soluciones automatizadas y seguras en base a dos scripts, uno conforme a Bash y otro en Python para el Banco Riendo, empresa que se encuentra analizando los beneficios de adoptar un modelo de nube hibrida por requerimientos del negocio. Dichas soluciones deben de ser fácilmente mantenidas y auditadas. Se busca aplicar en un caso realista los conceptos de automatización, scriptación en Linux, buenas prácticas de seguridad y trabajo colaborativo con control de versiones. A partir de este propósito general, se desprenden los siguientes objetivos en específico.

1) Automatizar la gestión de usuarios en Linux mediante Bash.

Desarrollar un script en Bash denominado ej1_crea_usuarios.sh, dicho script tendrá la capacidad de leer un archivo de entrada con la información de los usuarios a crear y, en base a estos datos, se invocará de forma adecuada los comandos del sistema, entre ellos useradd o adduser.

Deberá permitir configurar para cada usuario atributos como comentarios, directorio home, creación o no del directorio en caso de no existir y la Shell por defecto, manejando los valores por defecto cuando la información esté incompleta.

Se incorporará al script opciones adicionales (por ejemplo -i y -c contraseña) para:

- Informar detalladamente el resultado de la creación de cada usuario.
- Asignar una contraseña común a los usuarios creados.

Se deberá de gestionar correctamente situaciones de error, entre ellos si el archivo es inexistente, no regular, sin permisos de lectura, errores de sintaxis o de parámetros, entre otros. Utilizando códigos de retorno diferenciados y mensajes claros por la salida de errores.

2) Automatizar el despliegue de la aplicación de RRHH mediante scripts en Python

El diseño y desarrollo del script en Python deberá de automatizar el despliegue de la aplicación de recursos humanos del Banco Riendo, encargada de almacenar información sensible de los empleados, ejemplo, nombres, emails y salarios.

Se incluirá en el proceso de despliegue todas las medidas de seguridad razonables para minimizar riesgos de filtración de datos sensibles, tales como una correcta configuración de permisos, separación de componentes, manejo seguro de credenciales y registros de actividad.

3) Garantizar trazabilidad y buenas prácticas mediante el uso de GitHub

Se mantendrá todo el código del trabajo obligatorio, es decir, script en Bash y Python, archivos auxiliares y documentación) en un repositorio de GitHub, asegurando que todos los cambios sean trazables.

Ejercicio 1 – Script en Bash

Descripción general

Para el primer ejercicio se desarrolló un script en Bash llamado ej1_crea_usuarios.sh, cuyo objetivo es automatizar la creación de usuarios en un sistema Linux a partir de un archivo de texto con su información básica.

El script permite:

- Leer un archivo de entrada con los datos de cada usuario.
- Validar que el archivo sea correcto (exista, sea regular y tenga permisos de lectura).
- Verifica que cada línea del archivo tenga una sintaxis correcta y adecuada.
- Crea usuarios utilizando el comando useradd, pasando solo los parámetros necesarios.

Opcionales:

- Se mostrará información detallada sobre el resultado de la creación de cada usuario.
- Se asignará una misma contraseña a todos los usuarios creados.

Todo el proceso está pensado para ser robusto y trazable, utilizando códigos de error diferenciados para los distintos tipos de problemas que puedan ocurrir durante su ejecución. Adicionalmente el script está diseñado para ejecutarse únicamente con privilegios de root o mediante sudo. La creación de usuarios en el sistema requiere permisos de administrador.

Formato de archivo de usuarios

El archivo de entrada contiene una línea por usuario con el siguiente formato con los campos separados por " : ":

Usuario:comentario:directorio_home:crear_home(SI/NO):shell

- Usuario: nombre de cuenta (Obligatorio)
- Comentario: descripción o nombre completo (opcional)
- Directorio_home: ruta del home (Opcional)
- Crear_home (SI/NO): Indica si se debe de crear el home (SI/si/Si = -m, NO/no/No = M).
- Shell: ruta completa de la shell (por ejemplo /bin/bash); si se indica, el script verificará que exista.

  Comentario: Líneas vacías y líneas que comienzan con # se consideran comentarios y se ignoran.

Parámetros y opciones del script

ej1_crea_usuarios.sh [-i] [-c contraseña] Archivo_con_los_usuarios_a_crear

- **-i** : modo informativo, muestra para cada usuario si fue creado, si ya existía o si hubo errores. Al final indica el total de usuarios creados.
- **-c contraseña** : asigna una misma contraseña a todos los usuarios creados utilizando chpasswd. Si se usa -c sin indicar contraseña, el script finalizará con error de parámetro incorrecto.
- **Archivo de usuarios**: parámetro obligatorio. Solo se permite un archivo; si se pasan varios, se retorna el error de número de parámetros.

Validaciones y manejo de errores

Antes de crear usuarios, el script verifica:

1) Que se ejecute como root ($EUID == 0).
2) Que se haya indicado exactamente un archivo.
3) Que el archivo: Existe (parámetro -e)
4) Sea un archivo regular (parámetro -f)
5) Tenga los permisos de lectura (parámetro -r)

Durante el procedimiento, se controla que cada línea tenga exactamente 5 campos. Si falta el nombre de usuario o la cantidad de campos es incorrecta, se marca error de sintaxis.

Comprobaciones adicionales:

- Si el usuario ya existe, se omite su creación (informando en modo -i)
- Si se especifica una Shell que no existe, entonces no se crea el usuario y se marca error de creación.

Por último, el script utiliza códigos de salida diferenciados del 1 al 7 para indicar el tipo de error: archivo inexistente, no regular, sin permisos, sintaxis incorrecta, parámetros inválidos, número de parámetros incorrectos u otros errores. Si todo funciona correctamente, retorna un 0.

Funcionamiento general y ejemplo de uso

Para cada línea válida, el script:

1) Separa los campos (usuario, comentario, home_dir, crear_home, Shell).
2) Construye dinámicamente las opciones de useradd (por ejemplo -c, -d, -m / -M, -s)
3) Llama a useradd con esos parámetros.
4) Si se indicó -c contraseña, asigna la contraseña con chpasswd.
5) Lleva un contador de usuarios_creados y, en modo -i, muestra mensajes descriptivos.

Ruta absoluta del script en Bash:

/home/usuario/Obligatorio-DevOps-2025/ej1_crea_usuarios.sh

Ejemplo de ejecución script en Bash:

- sudo chmod a+x ./ej1_crea_usuarios.sh
- sudo . /ej1_crea_usuarios.sh -i -c [contraseña a elegir] ./Usuarios

- Con el siguiente comando se generan los usuarios definidos en el archivo usuarios.txt, se informa el resultado de cada uno y se le asigna la contraseña indicada.

Ejercicio 2 – Script en Python

Descripción de la aplicación en Recursos Humanos

La solución propuesta contempla una aplicación web de Recursos Humanos (RRHH) que gestiona información sensible de los empleados de Banco Riendo, como nombres, correos electrónicos y salarios.

A nivel lógico, la aplicación se compone de:

- Un front-end web servido por Apache y PHP sobre una instancia EC2 de AWS.
- Una base de datos MySQL alojada en un servicio gestionado de Amazon RDS.
- Un archivo de configuración .env que contiene los parámetros de conexión a la base de datos y credenciales de la aplicación.

El script de despliegue realiza las siguientes acciones:

1) Prepara el servidor web (Apache y PHP) sobre la instancia EC2, incluyendo la instalación de paquetes y la configuración de PHP-FPM.
2) Crea o reutiliza la instancia de base de datos RDS (MySQL) y obtiene su endpoint para conectarla a la aplicación.
3) Configura los Security Groups necesarios para permitir:

- Acceso HTTP (Puerto 80) hacia la instancia EC2.
- Acceso MySQL (Puerto 3306) desde la instancia EC2 hacia la instancia RDS.

4) Descarga y desplegar el código de la aplicación, desde un archivo descargable .ZIP publicado en GitHub Releases, lo copia a /var/www/html, genera el archivo /var/www/.env y ejecuta el script SQL de inicialización de la base de datos, si está disponible.

- De esta forma, el script se centra en el aprovisionamiento seguro de la infraestructura y en dejar el entorno listo para que la aplicación de RRHH sea desplegada correctamente.

Objetivos del script de despliegue:

1) Crear y configurar la instancia EC2 y su Security Group (SG-EC2):

- Localiza la VPC por defecto de la región us-east-1
- Crea o reutiliza un Security Group llamado SG-EC2 asociado a una VPC.
- Añade una regla de entrada para HTTP (puerto 80) desde cualquier IP (0.0.0.0/0), permitiendo el acceso web a la aplicación.
- Añade una regla de salida para HTTPS (puerto 443), necesaria para que la instancia pueda comunicarse hacia internet (por ejemplo, con SSM o para descargas).
- Asocia este Security Group a la instancia EC2, sin eliminar los grupos que ya atuvieran asignados.

2) Crea o reutiliza la instancia de base de datos RDS(MySQL):

Comprueba si existe una instancia RDS con el identificador RDS-Base-De-Datos, si ya existe, muestra su estado, pide por consola (de forma oculta) la contraseña del usuario administrador y la reutiliza para la conexión. Si no existe, solicita al usuario que ingrese una contraseña para el administrador de la base de datos, valida que no esté vacia y que tenga al menos 8 caracteres, y crea una instancia RDS con:

- Motor MySQL
- Nombre de la base de datos: demo_db.
- Usuario administrador: admin.
- PubliclyAccesible=False, evitando que la base de datos quede expuesta directamente a internet.
- BackupRetentionPeriod=0 (Sin backups automáticos configurados por el script).

- Espera a que la instancia RDS quede en estado disponible y obtiene su endpint, que luego se utilizará en la aplicación.

3) **Configuración del servidor web en la instancia EC2:**

- Lanza una instancia EC2 con:

- AMI ami-06b21ccaeff8cd686 (Amazon Linux 2023 en us-east-1).
- Tipo: t2.micro.
- Perfil IAM: LabInstanceProfile.

- Un script de user data que:

- Actualiza paquetes del sistema.
- Instala Apache, PHP, PHP-FPM y el cliente de MariaDB/MySQL.
- Configura la integración entre Apache y PHP-FPM.
- Crea un archivo de prueba /var/www/html/info.php para verificar la correcta ejecución de PHP.

4) Configurar el acceso entre EC2, RDS y desplegar aplicación:

- Crea o reutiliza un Security Group específico para RDS (SG-RDS-RDS-Base-De-Datos).
- Añade una regla de entradad que permite MySQL (puerto 3306) únicamente desde el Security Group de la instancia EC2 (SG-EC2), evitando abrir la base a todo internet.
- Asocia este Security Group a la instancia RDS.

5) Utilización de AWS Systems Manager (SSM):
- Descargar el ZIP de la aplicación desde GitHub Releases.
- Descomprimiéndolo en /home/ssm-user/app.
- Localiza el directorio de la aplicación (Obligatorio-main*) y copia su contenido a /var/www/html.
- Mueve el script init_db.sql fuera del directorio público a /var/www/**init**_db.sql
- Crea el archivo de configuración /var/www/.env con los datos de conexión en la base de datos y credenciales iniciales de la aplicación.

- Ejecuta init_db.sql sobre la base de datos RDS (Si existe).
- Ajusta permisos y renicia los servicios httpd y php-fpm.

- Al finalizar, el script muestra un resumen con los identificadores de la instancia EC2, la instancia RDS, el endpoint de RDS y la URL de acceso a la aplicación (index.php) y a la página de información de PHP (info.php).

Medidas de seguridad implementadas

El script incorpora varias medidas de seguridad orientadas a proteger tanto la base de datos como las credenciales de la aplicación:

A) **Manejo seguro de credenciales**: La contraseña del usuario administrador de la base de datos (RDS_ADMIN_PASSWORD) no está hardcodeada en el script y cuando se inicia RDS ya existente o se va a crear una nueva, el script solicita la contraseña mediante getpass, de forma que no se muestra en pantalla mientras se escribe y no queda registrada en texto plano dentro del código fuente.

B) **Configuración segura de la base de datos RDS**: Al crear la instancia RDS MySQL, se aplican las siguientes decisiones de seguridad:

- **PubliclyAccessible=False:** La base de datos no es accesible directamente desde Internet, sólo desde la red privada de la VPC.

- El script crea un Security Group dedicado a RDS y añade una regla de entrada que permite únicamente el tráfico MySQL (3306) desde el Security Group asociado a la instancia EC2. De esta forma, el acceso a la base de datos queda limitado a la capa de aplicación, y no se abre cualquier IP externa.

C) **Protección al archivo de configuración (.env) y del script SQL:**

- El archivo .env se genera en /var/www/.env, es decir, fuera del directorio público /var/www/html, esto evita que el archivo con credenciales pueda descargarse directamente vía HTTP.
- Se ajustan propietario y permisos del archivo de configuración:

- Chown apache:apache /var/www/.env
- Chmod 600 /var/www/.env, solo el usuario que ejecuta Apache puede leer el archivo; ningún otro usuario del sistema tiene acceso de lectura o escritura.

- El script init_db_sql se mueve desde /var/www/html a /var/www/init_db.sql, también fuera del webroot para evitar que quede expuesto a través del servidor web.
- Para ejecutar el SQL de inicialización, se crea temporalmente un fichero de configuración para el cliente mysql en /tmp con usuario, contraseña y host y se elimina al finalizar, reduciendo la exposición de las credenciales.

D) Buenas prácticas operativas:

- Uso de un perfil IAM (LabInstanceProfile) para la instancia EC2, que permite gestionar permisos a nivel de rol en lugar de usar claves de acceso estáticas en el código.
- Se utiliza AWS Systems Manager (SSM) para ejecutar comandos remotos en la instancia (descarga de la aplicación, configuración final, ejecución del SQL, etc.), evitando el uso de SSH directo como canal principal de administración.
- Se separan claramente las tareas de aprovisionamiento de infraestructura (EC2, RDS, Security Groups) de la configuración de la aplicación (descarga del ZIP, despliegue de archivos, .env, importación de la base de datos).

Flujo general de ejecución

Flujo general de ejecución

El flujo principal del script de despliegue es el siguiente:

1. **Inicio y descubrimiento de la VPC por defecto**

- Muestra mensajes informativos de inicio de despliegue.
- Localiza la VPC por defecto en la región us-east-1.

2. **Creación de la instancia EC2**

- Lanza la instancia con la AMI, tipo, perfil IAM y script de user_data.
- Espera a que la instancia alcance el estado de salud correcto (instance_status_ok).

3. **Creación y asociación del Security Group de EC2 (SG-EC2)**

- Crea o reutiliza el Security Group para EC2.
- Configura la regla de entrada HTTP (puerto 80) desde Internet.
- Configura una regla de salida HTTPS (puerto 443) para permitir tráfico saliente (por ejemplo, para SSM o descargas).
- Recupera los Security Groups actuales de la instancia EC2, añade SG-EC2 a la lista y actualiza la instancia para que quede asociada a todos ellos.

4. **Gestión de la instancia RDS**

- Intenta describir la instancia RDS RDS-Base-De-Datos.
- Si existe, muestra su estado y pide la contraseña del administrador por consola.
- Si no existe, solicita una nueva contraseña válida al usuario y crea la instancia RDS con los parámetros definidos (MySQL, demo_db, admin, sin acceso público, sin backups automáticos).
- Espera a que la instancia esté disponible y vuelve a consultarla para obtener el endpoint de conexión.

5. **Creación y configuración del Security Group de RDS (SG-RDS-RDS-Base-De-Datos)**

- Crea o reutiliza el Security Group de RDS.
- Añade una regla de entrada que permite MySQL (3306) únicamente desde el Security Group de la instancia EC2 (SG-EC2).
- Modifica la instancia RDS para que utilice este Security Group.

6. **Descarga y extracción del código de la aplicación mediante SSM**

- Define la URL pública al ZIP de la aplicación en GitHub Releases.
- Envía un comando vía SSM para:
    - Eliminar cualquier contenido previo en /home/ssm-user/app.
    - Crear ese directorio.
    - Descargar el ZIP con curl.
    - Descomprimirlo y mostrar el contenido.

7. **Configuración final de la aplicación en la instancia EC2 (SSM)**
   A través de otro comando SSM, ejecuta un script Bash que:

- Localiza el directorio real de la aplicación (obligatorio-main*).
- Crea los directorios /var/www y /var/www/html si no existen.
- Mueve el contenido de la aplicación a /var/www/html.
- Mueve init_db.sql a /var/www/init_db.sql, fuera del webroot.
- Elimina README.md del directorio público, si existe.
- Genera /var/www/.env con:
    - DB_HOST = endpoint de RDS
    - DB_NAME = nombre de la base
    - DB_USER y DB_PASS = credenciales de RDS
    - APP_USER y APP_PASS = credenciales iniciales de la aplicación
- Se asegura de que el cliente mysql esté instalado y ejecuta init_db.sql contra la base de datos RDS, manejando el caso en que las tablas ya existan.
- Ajusta los permisos de /var/www/html para que pertenezcan al usuario apache.
- Reinicia los servicios httpd y php-fpm.

8. **Resumen e instrucciones finales**

- Obtiene la IP pública de la instancia EC2.
- Muestra por pantalla:
    - ID de la instancia EC2.
    - IP pública.
    - ID de la instancia RDS.
    - Endpoint de RDS.
    - URL de acceso a la aplicación (http://IP_PUBLICA/index.php).
    - URL de la página de información de PHP (http://IP_PUBLICA/info.php).

Ruta absoluta del Script en Python:

/home/usuario/Obligatorio-DevOps-2025/ej2_despliegue_rh.py

Ejemplo de ejecución script en Python:

- Sudo chmod a+x ej2_despliegue_rh.py (De ser necesario)
- Python3 ej2_despliegue_rh.py

Repositorio de GitHub

El código completo de los scripts de Bash y Python, archivos auxiliares y documentación se encuentra en el siguiente repositorio:

- **URL clickeable al repositorio**: (https://github.com/Diegogar8/Obligatorio-DevOps-2025)

Archivo README:

El archivo readme contiene:

- Descripción del proyecto
- Requisitos / Requerimientos
- Modo de uso
- Archivos auxiliares

Anexos

Anexo A – Código del script en Bash:

```
#!/bin/bash

# Script para crear usuarios en el sistema a partir de un archivo de texto.
# Permite:
#  - Mostrar info detallada de cada creación (-i)
#  - Asignar una misma contraseña a todos (-c contraseña)
#  - Leer un archivo con formato: usuario:comentario:directorio_home:crear_home(SI/NO):shell

# Códigos de error (para salir con distintos códigos según el problema)
ERROR_ARCHIVO_NO_EXISTE=1       # El archivo de usuarios no existe
ERROR_ARCHIVO_NO_REGULAR=2      # El archivo no es regular
ERROR_SIN_PERMISOS_LECTURA=3    # No hay permisos de lectura sobre el archivo
ERROR_SINTAXIS_ARCHIVO=4        # Error de formato/sintaxis en alguna línea del archivo
ERROR_PARAMETRO_INCORRECTO=5    # Error en el uso de opciones (parámetros incorrectos)
ERROR_NUMERO_PARAMETROS=6       # Error en la cantidad/lógica de parámetros
ERROR_OTRO=7                    # Otros errores (genérico)

# Variables principales
ARCHIVO_USUARIOS=""             # Ruta del archivo con la lista de usuarios
PASSWORD=""                     # Contraseña a asignar si se usa -c
MOSTRAR_INFO=false              # Indica si se muestra info detallada (-i)
usuarios_creados=0              # Contador de usuarios creados

# Verificar que el script se ejecute como root (uid 0)
if [[ "$EUID" -ne 0 ]]; then
   echo "Error: Este script debe ejecutarse como root (usar sudo)" >&2
   exit $ERROR_OTRO
fi

# Flag para saber si ya se especificó un archivo de usuarios
tiene_archivo=false

# Bucle para procesar todos los parámetros recibidos ($@)
while [[ $# -gt 0 ]]; do
   case $1 in
      -i)
         # Opción -i: activar la salida informativa
         MOSTRAR_INFO=true
         shift
         ;;
      -c)
         # Opción -c: siguiente parámetro debe ser la contraseña
         if [[ -z "$2" ]]; then
            echo "Error: La opción -c requiere un argumento (contraseña)" >&2
            exit $ERROR_PARAMETRO_INCORRECTO
         fi
         PASSWORD="$2"  # Guardamos la contraseña
         shift 2        # Consumimos '-c' y el valor de la contraseña
         ;;
      -*)
```

```
            # Cualquier otra opción que empiece con '-' se considera inválida
            echo "Error: Modificador inválido: $1" >&2
            exit $ERROR_PARAMETRO_INCORRECTO
            ;;
        *)
            # Cualquier cosa que no empiece con '-' se toma como archivo de usuarios
            if [[ "$tiene_archivo" == true ]]; then
                # Si ya teníamos un archivo, significa que pasaron más de un error
                echo "Error: Se especificó más de un archivo" >&2
                exit $ERROR_NUMERO_PARAMETROS
            fi
            ARCHIVO_USUARIOS="$1"  # Guardamos el nombre del archivo
            tiene_archivo=true     # Marcamos que ya se especificó archivo
            shift                  # Consumimos este parámetro
            ;;
    esac
done

# Verificar que se haya pasado un archivo de usuarios
if [[ "$tiene_archivo" == false ]]; then
    echo "Error: Debe especificar un archivo con los usuarios a crear" >&2
    echo "Uso: $0 [-i] [-c contraseña] Archivo_con_los_usuarios_a_crear" >&2
    echo "" >&2
    echo "Opciones:" >&2
    echo "  -i              Muestra información sobre la creación de cada usuario" >&2
    echo "  -c contraseña    Asigna la contraseña especificada a todos los usuarios creados" >&2
    echo "  Archivo         Archivo con la información de usuarios (obligatorio)" >&2
    echo "" >&2
    echo "Formato del archivo (campos separados por :):" >&2
    echo "  usuario:comentario:directorio_home:crear_home(SI/NO):shell" >&2
    exit $ERROR_NUMERO_PARAMETROS
fi

# Verificar que el archivo exista
if [[ ! -e "$ARCHIVO_USUARIOS" ]]; then
    echo "Error: El archivo '$ARCHIVO_USUARIOS' no existe" >&2
    exit $ERROR_ARCHIVO_NO_EXISTE
fi

# Verificar que sea un archivo regular
if [[ ! -f "$ARCHIVO_USUARIOS" ]]; then
    echo "Error: '$ARCHIVO_USUARIOS' no es un archivo regular" >&2
    exit $ERROR_ARCHIVO_NO_REGULAR
fi

# Verificar permisos de lectura
if [[ ! -r "$ARCHIVO_USUARIOS" ]]; then
    echo "Error: No se tienen permisos de lectura para '$ARCHIVO_USUARIOS'" >&2
    exit $ERROR_SIN_PERMISOS_LECTURA
fi

# Variables para control de lectura de líneas y errores
num_linea=0            # Lleva el número de línea actual del archivo
errores_sintaxis=false  # Indica si hubo errores de formato
errores_creacion=false  # Indica si hubo errores al crear usuarios

# Leer el archivo línea por línea
```

```
# IFS= read -r linea   → lee la línea respetando espacios
# || [[ -n "$linea" ]] → asegura leer la última línea aunque no termine en '\n'
while IFS= read -r linea || [[ -n "$linea" ]]; do
    num_linea=$((num_linea + 1))  # Incrementar número de línea

    # Quitar espacios en blanco al inicio y al final de la línea
    linea_trimmed=$(echo "$linea" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')

    # Saltar líneas vacías o que comiencen con '#'
    if [[ -z "$linea_trimmed" ]] || [[ "$linea_trimmed" =~ ^# ]]; then
        continue
    fi

    # Contar cuántos ':' tiene la línea.
    # Si tiene 4 ':' → 5 campos: usuario:comentario:home:crear_home:shell
    num_campos=$(echo "$linea_trimmed" | tr -cd ':' | wc -c)

    if [[ $num_campos -ne 4 ]]; then
        echo "Error: Línea $num_linea no contiene exactamente 5 campos separados por ':'" >&2
        echo "  Línea: $linea_trimmed" >&2
        errores_sintaxis=true
        continue
    fi

    # Separar la línea en variables usando ':' como separador
    IFS=':' read -r usuario comentario home_dir crear_home shell <<< "$linea_trimmed"

    # Validar que el nombre de usuario no esté vacío
    if [[ -z "$usuario" ]]; then
        echo "Error: Línea $num_linea - El nombre de usuario no puede estar vacío" >&2
        errores_sintaxis=true
        continue
    fi

    # Verificar si el usuario ya existe en el sistema
    if id "$usuario" &>/dev/null; then
        if [[ "$MOSTRAR_INFO" == true ]]; then
            echo "Usuario '$usuario' (línea $num_linea): Ya existe, se omite"
        fi
        continue  # No lo crea de nuevo, sigue con la siguiente línea
    fi

    # Array para ir armando los parámetros a pasar a useradd
    args=()

    # Si hay comentario, agregar opción -c "comentario"
    if [[ -n "$comentario" ]]; then
        args+=(-c "$comentario")
    fi

    # Si hay directorio home (campo no vacío), agregar opción -d
    if [[ -n "$home_dir" ]]; then
        args+=(-d "$home_dir")
    fi

    # Según el valor SI/NO, decidir si crear o no el home (-m o -M)
    if [[ "$crear_home" == "SI" ]] || [[ "$crear_home" == "si" ]] || [[ "$crear_home" == "Si" ]]; then
```

```
    args+=(-m)   # Crear el home
elif [[ "$crear_home" == "NO" ]] || [[ "$crear_home" == "no" ]] || [[ "$crear_home" == "No" ]]; then
    args+=(-M)   # No crear el home
fi

# Si el campo de shell no está vacío, verificar que el archivo exista
if [[ -n "$shell" ]]; then
    if [[ ! -f "$shell" ]]; then
        # La shell no existe, marcamos error pero no detenemos todo el script
        if [[ "$MOSTRAR_INFO" == true ]]; then
            echo "Usuario '$usuario' (línea $num_linea): ERROR - Shell '$shell' no existe"
        fi
        errores_creacion=true
        continue
    fi
    # Si existe, se agrega la opción -s
    args+=(-s "$shell")
fi

# Finalmente, agregar el nombre de usuario como último argumento
args+=("$usuario")

# Intentar crear el usuario con useradd y los argumentos armados
if useradd "${args[@]}" 2>/dev/null; then
    # Si se creó correctamente y se definió una contraseña general
    if [[ -n "$PASSWORD" ]]; then
        # Asignar contraseña con chpasswd
        if ! echo "$usuario:$PASSWORD" | chpasswd 2>/dev/null; then
            # Si no se pudo asignar la contraseña, marcar el error
            if [[ "$MOSTRAR_INFO" == true ]]; then
                echo "Usuario '$usuario' (línea $num_linea): Creado pero no se pudo asignar contraseña"
            fi
            errores_creacion=true
            continue
        fi
    fi

    # Mensaje de éxito si corresponde mostrar info
    if [[ "$MOSTRAR_INFO" == true ]]; then
        echo "Usuario '$usuario' (línea $num_linea): Creado exitosamente"
    fi
    # Incrementar contador de usuarios creados
    usuarios_creados=$((usuarios_creados + 1))
else
    # useradd falló por algún motivo (uid, grupos, etc.)
    if [[ "$MOSTRAR_INFO" == true ]]; then
        echo "Usuario '$usuario' (línea $num_linea): ERROR - No se pudo crear"
    fi
    errores_creacion=true
fi
```

```
# Redirección final: todo el while lee desde el archivo de usuarios
done < "$ARCHIVO_USUARIOS"

# Al final, si se pidió info, mostrar el total de usuarios creados
if [[ "$MOSTRAR_INFO" == true ]]; then
    echo ""
    echo "Total de usuarios creados exitosamente: $usuarios_creados"
fi

# Decidir código de salida según los errores ocurridos
if [[ "$errores_sintaxis" == true ]]; then
    # Hubo al menos un error de formato en el archivo
    exit $ERROR_SINTAXIS_ARCHIVO
elif [[ "$errores_creacion" == true ]]; then
    # Hubo problemas al crear usuarios (shell no existe, useradd falló, etc.)
    exit $ERROR_OTRO
fi

# Si todo fue bien, devolver 0 (éxito)
exit 0
```

Anexo B – Código de script en Python:

```
#!/usr/bin/env python3

import boto3                 # SDK de AWS para Python (para usar EC2, RDS, SSM, etc.)
import time                  # Para esperas entre comandos SSM
import getpass
from botocore.exceptions import ClientError    # Excepción específica de errores de AWS

# ---------------------------
# CONSTANTES DE CONFIGURACIÓN
# ---------------------------
REGION = 'us-east-1'

ec2 = boto3.client('ec2', region_name=REGION)
ssm = boto3.client('ssm', region_name=REGION)
rds = boto3.client("rds", region_name=REGION)

INSTANCE_NAME = "rh-app-web"
SG_EC2_NAME = "SG-EC2"
DB_INSTANCE_ID = "RDS-Base-De-Datos"
SG_RDS_NAME = "SG-RDS-RDS-Base-De-Datos"
DB_NAME = "demo_db"
DB_USERNAME = "admin"
DB_PASSWORD = None
# ---------------------------
# === Creación de instancia EC2 Y EL USER DATA ===
# ---------------------------
user_data = """#!/bin/bash
sudo dnf clean all
sudo dnf makecache
sudo dnf update -y
# Instalar web + php + mariadb + utilidades
sudo dnf install httpd php php-cli php-fpm php-common php-mysqlnd mariadb105 -y
sudo systemctl enable --now httpd
sudo systemctl enable --now php-fpm
echo '<FilesMatch \\.php$>
  SetHandler "proxy:unix:/run/php-fpm/www.sock|fcgi://localhost/"
</FilesMatch>' > /etc/httpd/conf.d/php-fpm.conf
sudo mkdir -p /var/www/html
echo "<?php phpinfo(); ?>" > /var/www/html/info.php
sudo systemctl restart httpd php-fpm
"""

print("=" * 60)
print("Lanzando la creación de la infraestructura EC2 y RDS en AWS")
print("Despliegue de aplicación de RRHH.")
print("=" * 60)

# Se Identificará la VPC predeterminada para asociar los Security Groups
vpcs = ec2.describe_vpcs(Filters=[{'Name': 'isDefault', 'Values': ['true']}])
if not vpcs['Vpcs']:
    print("No se pudo localizar la VPC por defecto. Verifica la región y las credenciales de AWS.")
    exit(1)
VPC_ID = vpcs['Vpcs'][0]['VpcId']
```

```
print(f"[INFO] Usando VPC por defecto: {VPC_ID}")

# Iniciar una instancia EC2 con el perfil de instancia (Instance Profile) LabRole
print("\n[*] Creando instancia EC2...")
response = ec2.run_instances(
    ImageId='ami-06b21ccaeff8cd686',  # Amazon Linux 2023 en us-east-1
    MinCount=1,
    MaxCount=1,
    InstanceType='t2.micro',
    IamInstanceProfile={'Name': 'LabInstanceProfile'},
    UserData=user_data,
    TagSpecifications=[
        {
            'ResourceType': 'instance',
            'Tags': [{'Key': 'Name', 'Value': INSTANCE_NAME}]
        }
    ]
)

instance_id = response['Instances'][0]['InstanceId']
print(f"[+] Instancia desplegada correctamente. ID: {instance_id}, tag '{INSTANCE_NAME}'")
print("[*] Monitoreando hasta que la instancia esté en estado 'running'...")
print("[*] El arranque suele tomar entre 5 y 8 minutos.")

# Esperar a que la instancia esté en estado running
ec2.get_waiter('instance_status_ok').wait(InstanceIds=[instance_id])
print("[+] Instancia EC2 lista y en estado 'running'.")


# ----------------------------
# CREAR SECURITY GROUP PARA EC2
# ----------------------------
print("\n[*] Preparando el Security Group para la instancia EC2...")
sg_name = SG_EC2_NAME
try:
    response = ec2.create_security_group(
        GroupName=sg_name,
        Description="Permitir trafico web desde cualquier IP",
        VpcId=VPC_ID
    )
    sg_id = response["GroupId"]
    print(f"[+] Security Group de EC2 creado correctamente. ID: {sg_id}")
except ClientError as e:
    error_code = e.response["Error"]["Code"]
    if error_code == "InvalidGroup.Duplicate":
        print("[*] El Security Group ya existe. Obteniendo su ID...")
        sg_id = ec2.describe_security_groups(
            Filters=[
                {'Name': 'group-name', 'Values': [sg_name]},
                {'Name': 'vpc-id', 'Values': [VPC_ID]}
            ]
        )["SecurityGroups"][0]["GroupId"]
        print(f"[+] Security Group EC2 encontrado. ID: {sg_id}")
    else:
        raise
```

```
# ----------------------------------------------------------------------
# Verificar y agregar la regla HTTP (puerto 80) en caso de ser necesario
#----------------------------------------------------------------------
try:
    ec2.authorize_security_group_ingress(
        GroupId=sg_id,
        IpPermissions=[
            {
                "IpProtocol": "tcp",
                "FromPort": 80,
                "ToPort": 80,
                "IpRanges": [{"CidrIp": "0.0.0.0/0"}]
            }
        ]
    )
    print("[+] Regla HTTP (puerto 80) aplicada correctamente al Security Group EC2.")
except ClientError as e:
    error_code = e.response["Error"]["Code"]
    if error_code == "InvalidPermission.Duplicate":
        print("[*] La regla HTTP ya existe. Continuando...")
    else:
        raise
#-----------------------------------------------------
# Agregar regla de SALIDA HTTPS (puerto 443) para SSM
#-----------------------------------------------------
try:
    ec2.authorize_security_group_egress(
        GroupId=sg_id,
        IpPermissions=[
            {
                "IpProtocol": "tcp",
                "FromPort": 443,
                "ToPort": 443,
                "IpRanges": [{"CidrIp": "0.0.0.0/0"}]
            }
        ]
    )
    print("[+] Permiso de tráfico saliente HTTPS (443) habilitado para SSM.")
except ClientError as e:
    error_code = e.response["Error"]["Code"]
    if error_code == "InvalidPermission.Duplicate":
        print("[*] La regla HTTPS saliente ya existe. Continuando...")
    else:
        # Los SGs por defecto ya permiten todo el tráfico saliente,
        # así que este error es esperado
        print("[*] Tráfico saliente ya permitido por defecto.")
#----------------------------------------------------------------------
# AÑADIR EL SG A LA INSTANCIA EC2 MANTENIENDO LOS SGs EXISTENTES
#----------------------------------------------------------------------
print(f"\n[*] Actualizando la instancia {instance_id} para incluir el SG {sg_id}...")
try:
    # Listar los SGs actualmente vinculados a la instancia
    current_instance = ec2.describe_instances(InstanceIds=[instance_id])
    current_sgs = current_instance['Reservations'][0]['Instances'][0]['SecurityGroups']
    current_sg_ids = [sg['GroupId'] for sg in current_sgs]

    # Incluir el nuevo SG en la lista de asociados si todavía no está incluido
```

```
    if sg_id not in current_sg_ids:
        current_sg_ids.append(sg_id)

    # Vincular la instancia con todos los Security Groups definidos
    ec2.modify_instance_attribute(
        InstanceId=instance_id,
        Groups=current_sg_ids
    )
    print(f"[+] Asociación del SG {sg_id} con la instancia realizada con éxito.")
except ClientError as e:
    print("[ERROR] Error inesperado al asociar el SG:")
    raise
#------------------------------------------------
# MANEJO DE LA INSTANCIA RDS (CREAR U OBTENER EXISTENTE)
#------------------------------------------------
print(f"\n[*] Buscando instancia RDS '{DB_INSTANCE_ID}'...")

rds_exists = False

try:
    resp = rds.describe_db_instances(DBInstanceIdentifier=DB_INSTANCE_ID)
    rds_instance = resp["DBInstances"][0]
    rds_exists = True
    print(f"[+] RDS existe (estado: {rds_instance['DBInstanceStatus']}).")
    print("[*] Ingrese la contraseña anterior para continuar:")
    DB_PASS = getpass.getpass().strip()
    DB_PASSWORD = DB_PASS

except ClientError as e:
    if e.response['Error']['Code'] == 'DBInstanceNotFound':
        print("[*] RDS no existe. Creando nueva instancia...")
    else:
        raise

if not rds_exists:
    while True:
        DB_PASS = getpass.getpass('\n[*] Ingresa la contraseña del admin RDS: ').strip()
        if not DB_PASS:
            print('[!] La contraseña no puede estar vacía. Intenta nuevamente.')
            continue
        if len(DB_PASS) < 8:
            print('[!] La contraseña debe tener al menos 8 caracteres. Intenta nuevamente.')
            continue
        break

    DB_PASSWORD = DB_PASS

    rds.create_db_instance(
        DBName=DB_NAME,
        DBInstanceIdentifier=DB_INSTANCE_ID,
        AllocatedStorage=20,
        DBInstanceClass="db.t3.micro",
        Engine="mysql",
        MasterUsername=DB_USERNAME,
        MasterUserPassword=DB_PASS,
        PubliclyAccessible=False,
        BackupRetentionPeriod=0
```

```
    )
    print("[*] Esperando a que RDS esté disponible...")
    print("[*] Tiempo estimado: 5-10 minutos.")
    rds.get_waiter("db_instance_available").wait(DBInstanceIdentifier=DB_INSTANCE_ID)
    rds_instance = rds.describe_db_instances(DBInstanceIdentifier=DB_INSTANCE_ID)["DBInstances"][0]
    print("[+] Instancia RDS creada y disponible.")
#---------------------------------------------
# CREAR SECURITY GROUP PARA RDS
#---------------------------------------------
print("\n[*] Iniciando configuración del Security Group específico para RDS...")
rds_sg_name = SG_RDS_NAME
try:
    resp = ec2.create_security_group(
        GroupName=rds_sg_name,
        Description="SG para RDS que permite acceso MySQL desde EC2",
        VpcId=VPC_ID
    )
    rds_sg_id = resp["GroupId"]
    print(f"[+] Security Group RDS creado: {rds_sg_id}")
except ClientError as e:
    code = e.response["Error"]["Code"]
    if code == "InvalidGroup.Duplicate":
        sgs = ec2.describe_security_groups(
            Filters=[
                {'Name': 'group-name', 'Values': [rds_sg_name]},
                {'Name': 'vpc-id', 'Values': [VPC_ID]}
            ]
        )["SecurityGroups"]
        if not sgs:
            raise
        rds_sg_id = sgs[0]["GroupId"]
        print(f"[+] Security Group RDS existente: {rds_sg_id}")
    else:
        raise
#-------------------------------------------------------------------------------
# Habilitar ingreso al SG de RDS desde el SG de EC2 en el puerto MySQL (3306)
#-------------------------------------------------------------------------------
try:
    ec2.authorize_security_group_ingress(
        GroupId=rds_sg_id,
        IpPermissions=[
            {
                "IpProtocol": "tcp",
                "FromPort": 3306,
                "ToPort": 3306,
                'UserIdGroupPairs': [{'GroupId': sg_id}]
            }
        ]
    )
    print(f"[+] Regla de ingreso MySQL (puerto 3306) aplicada al SG de RDS para conexiones desde EC2.")
except ClientError as e:
    if e.response["Error"]["Code"] == "InvalidPermission.Duplicate":
        print("[*] La regla MySQL ya existe en el SG RDS. Continuando...")
    else:
        raise
```

```
#----------------------------------------------------------------
# Esperar a que la instancia RDS esté disponible antes de modificarla
#----------------------------------------------------------------
print(f"\n[*] Chequeando disponibilidad de la instancia RDS {DB_INSTANCE_ID}...")
waiter = rds.get_waiter('db_instance_available')
try:
    rds.describe_db_instances(DBInstanceIdentifier=DB_INSTANCE_ID)
    waiter.wait(DBInstanceIdentifier=DB_INSTANCE_ID)
    print(f"[+] Instancia RDS disponible.")
except ClientError as e:
    code = e.response['Error']['Code']
    if code == 'DBInstanceNotFoundFault':
        print(f"[ERROR] La instancia RDS '{DB_INSTANCE_ID}' no existe.")
    else:
        print(f"[ERROR] Error inesperado: {e}")
    raise
#-------------------------------------------------------------------------------
# Consultar nuevamente la instancia RDS para asegurarse de tener el endpoint actual
#-------------------------------------------------------------------------------
rds_instance = rds.describe_db_instances(DBInstanceIdentifier=DB_INSTANCE_ID)["DBInstances"][0]
endpoint = rds_instance['Endpoint']['Address']
print(f"[+] Endpoint RDS: {endpoint}")
#-------------------------------------------------
# Asignar el Security Group correspondiente a la instancia RDS
#-------------------------------------------------
print("\n[*] Actualizando la instancia RDS para usar el nuevo Security Group...")
try:
    rds.modify_db_instance(
        DBInstanceIdentifier=DB_INSTANCE_ID,
        VpcSecurityGroupIds=[rds_sg_id],
        ApplyImmediately=True
    )
    print(f"[+] SG RDS {rds_sg_name} asociado a la instancia RDS.")
except ClientError as e:
    print("[ERROR] Error al asociar el SG a la instancia RDS:")
    raise

print(f"\n[+] Instancia RDS {DB_INSTANCE_ID} configurada correctamente.")
#-----------------------------------------------------
# === CONFIGURAR LA APLICACIÓN EN LA INSTANCIA EC2 ===
#-----------------------------------------------------
PUBLIC_ZIP_URL = "https://github.com/Diegogar8/Obligatorio-DevOps-2025/releases/download/v1.0/obligatorio-
main.zip"

print("\n[*] Iniciando descarga y extracción del archivo ZIP desde GitHub Release...")
download_and_extract_cmds = [
    "sudo rm -rf /home/ssm-user/app",
    "sudo mkdir -p /home/ssm-user/app",
    f"curl -L {PUBLIC_ZIP_URL} -o /home/ssm-user/app/app.zip",
    "sudo unzip -o /home/ssm-user/app/app.zip -d /home/ssm-user/app/",
    "echo '[CHECK] Contenido /home/ssm-user/app:'",
    "ls -la /home/ssm-user/app"
]
```

```
def send_ssm_and_wait_simple(instance_id, commands, timeout=300, comment=""):
    """Envía comandos via SSM y espera el resultado."""
    resp = ssm.send_command(
        InstanceIds=[instance_id],
        DocumentName="AWS-RunShellScript",
        Parameters={"commands": commands},
        Comment=comment
    )
    cmd_id = resp['Command']['CommandId']
    elapsed = 0
    while elapsed < timeout:
        try:
            inv = ssm.get_command_invocation(CommandId=cmd_id, InstanceId=instance_id)
        except ClientError as e:
            msg = str(e)
            if "InvocationDoesNotExist" in msg or "ThrottlingException" in msg:
                time.sleep(2)
                elapsed += 2
                continue
            else:
                raise

        status = inv.get('Status')
        if status in ['Success', 'Failed', 'Cancelled', 'TimedOut']:
            return status, inv.get('StandardOutputContent', ''), inv.get('StandardErrorContent', '')
        time.sleep(2)
        elapsed += 2
    return 'Timeout', '', 'Timeout esperando descarga'


dl_status, dl_out, dl_err = send_ssm_and_wait_simple(
    instance_id, download_and_extract_cmds, comment="download-extract"
)
print(f"\n[Descarga ZIP] Estado: {dl_status}")
if dl_status != 'Success':
    print('[!] Advertencia: problemas durante la descarga/extracción.')
    if dl_err:
        print(f"    Error: {dl_err}")
#-------------------------------------
# COMANDOS SSM FINALES
#-------------------------------------
print("\n[*] Realizando tareas finales de configuración en EC2...")
print("    (copiar archivos, generar .env, importar SQL, ajustar permisos y reiniciar)")


def send_ssm_and_wait(instance_id, commands, timeout=600, poll=3, comment=""):
    """
    Envía comandos via SSM (commands: list with a single big script is recommended)
    Espera resultado y maneja InvocationDoesNotExist y throttling.
    Retorna (status, stdout, stderr).
    """
    resp = ssm.send_command(
        InstanceIds=[instance_id],
        DocumentName="AWS-RunShellScript",
        Parameters={"commands": commands},
        Comment=comment
```

```
    )
    cmd_id = resp["Command"]["CommandId"]
    elapsed = 0
    backoff = 1
    while elapsed < timeout:
        try:
            inv = ssm.get_command_invocation(CommandId=cmd_id, InstanceId=instance_id)
        except ClientError as e:
            msg = str(e)
            if "InvocationDoesNotExist" in msg or "ThrottlingException" in msg:
                time.sleep(backoff)
                elapsed += backoff
                backoff = min(backoff * 2, 10)
                continue
            else:
                raise
        status = inv.get("Status")
        if status in ("Success", "Failed", "Cancelled", "TimedOut"):
            return status, inv.get("StandardOutputContent", ""), inv.get("StandardErrorContent", "")
        time.sleep(poll)
        elapsed += poll
    return "Timeout", "", "SSM command timed out"


shell_script = f"""#!/bin/bash
set -euo pipefail
echo "[SSM SCRIPT] inicio: $(date)"

REALDIR="$(find /home/ssm-user/app -maxdepth 1 -type d -name 'obligatorio-main*' | head -n1)"
if [ -z "$REALDIR" ]; then
  echo "[ERROR] No se encontró ninguna carpeta obligatorio-main* en /home/ssm-user/app"
  exit 2
fi
#-----------------------------------
# Variables principales del script
#-----------------------------------
EXTRACTED="$REALDIR"
RDS_ENDPOINT="{endpoint}"
DB_NAME="{DB_NAME}"
DB_USER="{DB_USERNAME}"
DB_PASS="{DB_PASSWORD}"
#-------------------------------------------------
# 1) Verificar y crear los directorios necesarios
#-------------------------------------------------
sudo mkdir -p /var/www
sudo mkdir -p /var/www/html
#-----------------------------------------------------------------------
# 2) Copiar y mover el contenido, incluyendo archivos ocultos, si los hay
#-----------------------------------------------------------------------
if [ -d "$REALDIR" ]; then
    shopt -s dotglob nullglob
    if [ -z "$(ls -A "$REALDIR")" ]; then
        echo "[WARN] Directorio $REALDIR vacío, nada para mover"
    else
        sudo mv "$REALDIR"/* /var/www/html/ || true
    fi
    shopt -u dotglob nullglob
```

```
else
   echo "[ERROR] REALDIR ($REALDIR) no existe"
   exit 2
fi
#--------------------------------------------------------------------------
# 3) Proteger init_db.sql moviéndolo fuera del webroot, si el archivo existe
#--------------------------------------------------------------------------
if [ -f /var/www/html/init_db.sql ]; then
  sudo mv /var/www/html/init_db.sql /var/www/init_db.sql
else
  echo "[WARN] init_db.sql no encontrado en /var/www/html"
fi
#-------------------------------------------------------------------
# 4) Eliminar README del directorio web en caso de estar presente
#-------------------------------------------------------------------
if [ -f /var/www/html/README.md ]; then
  sudo rm -f /var/www/html/README.md
fi
#-------------------------------------------------------------------
# 5) Crear el archivo .env con datos sensibles de manera segura
#-------------------------------------------------------------------
sudo bash -c 'cat > /var/www/.env << "EOF"
DB_HOST={endpoint}
DB_NAME={DB_NAME}
DB_USER={DB_USERNAME}
DB_PASS={DB_PASSWORD}
APP_USER=admin
APP_PASS=admin123
EOF'
sudo chown apache:apache /var/www/.env || true
sudo chmod 600 /var/www/.env
#-------------------------------------------------
# 6) Asegurar cliente mysql instalado
#-------------------------------------------------
if ! command -v mysql >/dev/null 2>&1; then
  echo "[INFO] mysql no encontrado, instalando cliente..."
  if command -v dnf >/dev/null 2>&1; then
    sudo dnf -y install mariadb105 || sudo dnf -y install mariadb
  elif command -v yum >/dev/null 2>&1; then
    sudo yum -y install mariadb
  else
    echo "[ERROR] No hay gestor de paquetes conocido (dnf/yum)."
    exit 3
  fi
  echo "[OK] mysql client instalado"
else
  echo "[OK] mysql client ya presente"
fi
#--------------------------------------------------------------------------
# 7) Lanzar el script init_db.sql sobre la base de datos RDS (si existe)
#--------------------------------------------------------------------------
if [ -f /var/www/init_db.sql ]; then
   TMPCNF="/tmp/.mycred.$$"
   cat > "$TMPCNF" <<EOF
[client]
user={DB_USERNAME}
password={DB_PASSWORD}
```

```
host={endpoint}
EOF
    chmod 600 "$TMPCNF"
    echo "[INFO] Ejecutando init_db.sql en $RDS_ENDPOINT..."
    set +e
    mysql --defaults-extra-file="$TMPCNF" {DB_NAME} < /var/www/init_db.sql 2> /tmp/mysql_err.$$
    rc=$?
    set -e
    MYSQL_ERR_OUT="$(cat /tmp/mysql_err.$$ || true)"
    rm -f /tmp/mysql_err.$$ "$TMPCNF"
    if [ $rc -ne 0 ]; then
        if echo "$MYSQL_ERR_OUT" | grep -qi 'already exists'; then
            echo "[WARN] Tablas existentes detectadas, continuando: $MYSQL_ERR_OUT"
        else
            echo "[ERROR] mysql código $rc: $MYSQL_ERR_OUT"
            exit $rc
        fi
    else
        echo "[OK] init_db.sql ejecutado correctamente"
    fi
else
    echo "[WARN] /var/www/init_db.sql no existe; salto ejecución SQL"
fi
#------------------------------------------------------------------------
# 8) Alinear permisos finales de archivos y directorios de la aplicación
#------------------------------------------------------------------------
sudo chown -R apache:apache /var/www/html || true
#----------------------------
# 9) Reiniciar servicios
#----------------------------
sudo systemctl restart httpd || {{ echo '[ERROR] fallo restart httpd'; systemctl status httpd --no-pager || true; exit 4; }}
sudo systemctl restart php-fpm || {{ echo '[ERROR] fallo restart php-fpm'; systemctl status php-fpm --no-pager || true;
exit 5; }}

echo "[SSM SCRIPT] fin: $(date)"
"""
#------------------------------------------------------------
# Lanzar este bloque como un único comando en la terminal
#------------------------------------------------------------
status, out, err = send_ssm_and_wait(instance_id, [shell_script], timeout=1200, comment="deploy-full-script")
```

```
#-----------------------------------------------
# Obtener la IP pública de la instancia EC2
#-----------------------------------------------
resp = ec2.describe_instances(InstanceIds=[instance_id])
EC2_public_ip = resp['Reservations'][0]['Instances'][0].get('PublicIpAddress')

if not EC2_public_ip:
    print("\n[ERROR] No fue posible obtener la IP pública de la instancia EC2.")
    print("        Verifica que la instancia tenga una IP pública asociada y esté en estado 'running'.")
else:
    print("\n" + "=" * 60)
    print("   DESPLIEGUE FINALIZADO CORRECTAMENTE   ")
    print("=" * 60)
    print(f"\n[+] Instancia EC2      : {instance_id}")
    print(f"[+] IP pública EC2     : {EC2_public_ip}")
    print(f"[+] Instancia RDS      : {DB_INSTANCE_ID}")
    print(f"[+] Endpoint RDS       : {endpoint}")

    print("\n>>> URL de la aplicación")
    print(f"    http://{EC2_public_ip}/index.php")

    print("\n>>> Página de información de PHP:")
    print(f"    http://{EC2_public_ip}/info.php")
    print("=" * 60)
```

Anexo C – Capturas de pantalla

- Ejecución exitosa del script Bash

```
cperez@Albivioleta-PC:/home/Obligatorio-DevOps-2025$ sudo chmod a+x ./ej1_crea_usuarios.sh
cperez@Albivioleta-PC:/home/Obligatorio-DevOps-2025$ sudo ./ej1_crea_usuarios.sh -i -c Password01 ./Usuario
Usuario 'pepe' (línea 1): Creado exitosamente
Usuario 'papanatas' (línea 2): Creado exitosamente
Usuario 'elmaligno' (línea 3): ERROR - Shell '/bin/el_maligno' no existe
Usuario 'usuario_tontopico' (línea 4): ERROR - Shell '/bin/jiji' no existe

Total de usuarios creados exitosamente: 2
cperez@Albivioleta-PC:/home/Obligatorio-DevOps-2025$
```

- Ejecución exitosa del script en Python.

```
cperez@Albivioleta-PC:~/Obligatorio-DevOps-2025$ python3 ej2_despliegue_rh.py
============================================================
Lanzando la creación de la infraestructura EC2 y RDS en AWS
Despliegue de aplicación de RRHH.
============================================================
[INFO] Usando VPC por defecto: vpc-033dae0e3eda2f0a9

[*] Creando instancia EC2...
[+] Instancia desplegada correctamente. ID: i-09e55353d6430b589, tag 'rh-app-web'
[*] Monitoreando hasta que la instancia esté en estado 'running'...
[*] El arranque suele tomar entre 5 y 8 minutos.
[+] Instancia EC2 lista y en estado 'running'.

[*] Preparando el Security Group para la instancia EC2...
[+] Security Group de EC2 creado correctamente. ID: sg-08d0ef66fb57fdec8
[+] Regla HTTP (puerto 80) aplicada correctamente al Security Group EC2.
[+] Permiso de tráfico saliente HTTPS (443) habilitado para SSM.

[*] Actualizando la instancia i-09e55353d6430b589 para incluir el SG sg-08d0ef66fb57fdec8...
[+] Asociación del SG sg-08d0ef66fb57fdec8 con la instancia realizada con éxito.

[*] Buscando instancia RDS 'RDS-Base-De-Datos'...
[*] RDS no existe. Creando nueva instancia...

[*] Ingresa la contraseña del admin RDS:
[*] Esperando a que RDS esté disponible...
[*] Tiempo estimado: 5-10 minutos.
[+] Instancia RDS creada y disponible.

[*] Iniciando configuración del Security Group específico para RDS...
[+] Security Group RDS creado: sg-0728990e755710049
[+] Regla de ingreso MySQL (puerto 3306) aplicada al SG de RDS para conexiones desde EC2.

[*] Chequeando disponibilidad de la instancia RDS RDS-Base-De-Datos...
[+] Instancia RDS disponible.
[+] Endpoint RDS: rds-base-de-datos.crcuws0ogoog.us-east-1.rds.amazonaws.com

[*] Actualizando la instancia RDS para usar el nuevo Security Group...
[+] SG RDS SG-RDS-RDS-Base-De-Datos asociado a la instancia RDS.

[+] Instancia RDS RDS-Base-De-Datos configurada correctamente.

[*] Iniciando descarga y extracción del archivo ZIP desde GitHub Release...

[Descarga ZIP] Estado: Success

[*] Realizando tareas finales de configuración en EC2...
    (copiar archivos, generar .env, importar SQL, ajustar permisos y reiniciar)

============================================================
   DESPLIEGUE FINALIZADO CORRECTAMENTE
============================================================

[+] Instancia EC2      : i-09e55353d6430b589
[+] IP pública EC2     : 54.175.37.67
[+] Instancia RDS      : RDS-Base-De-Datos
[+] Endpoint RDS       : rds-base-de-datos.crcuws0ogoog.us-east-1.rds.amazonaws.com

>>> URL de la aplicación
    http://54.175.37.67/index.php

>>> Página de información de PHP:
    http://54.175.37.67/info.php
============================================================
```