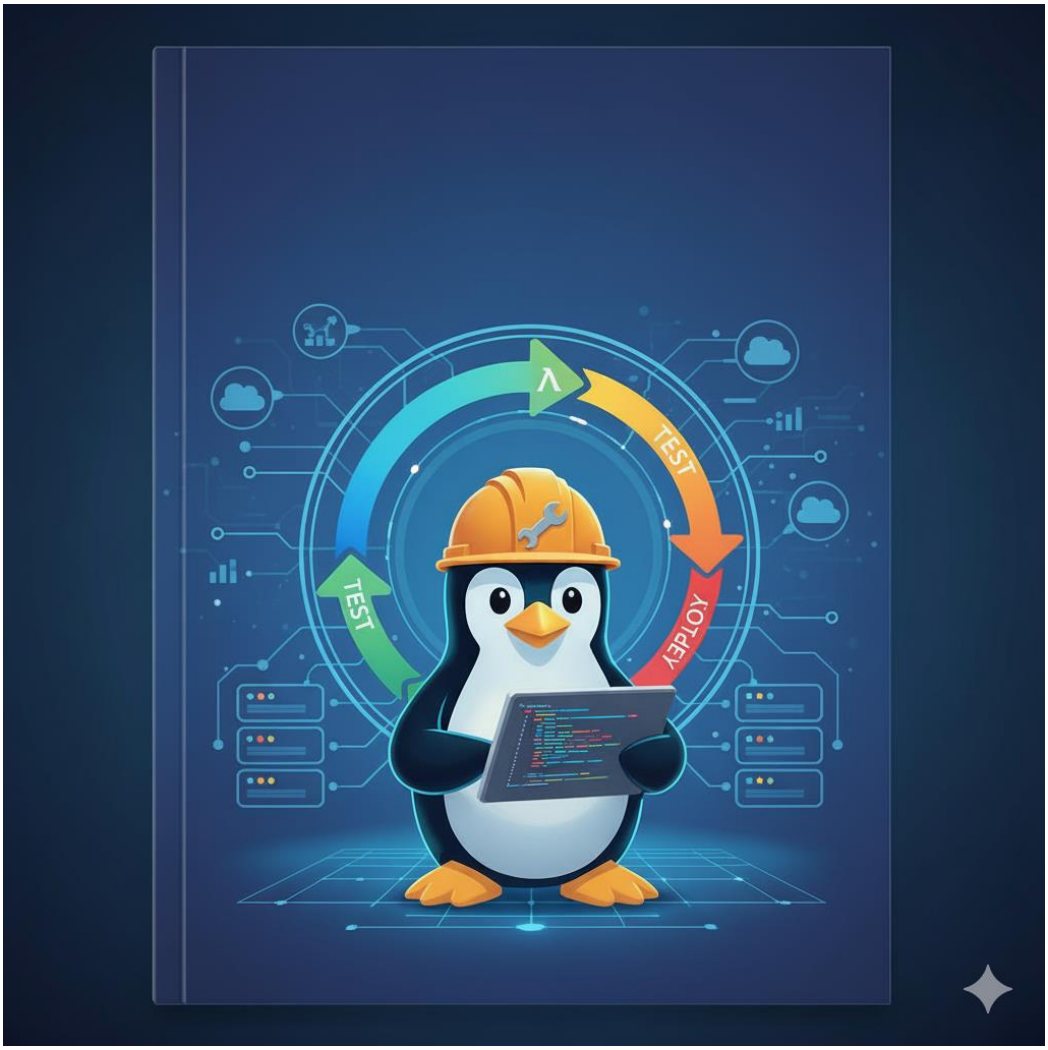
Obligatorio – Programación para DevOps – Agosto 2025



Nombre: Franco Larrosa Barreiro

Número de estudiante: 241688

Materia: Programación para DevOps

Indice

Ejercicio 1 Bash:

Código:

El código se encuentra en un archivo, en la carpeta .zip entregada. La carpeta .zip contiene el repositorio, junto a este documento.

Nombre del archivo: *ObligatorioPPDO/ScriptBash/crearUsers.sh*

Se opta por no pegar el código en este documento debido a que queda mal formateado.

Se recomienda abrir el archivo con notepad++ para que el formato no se cambie, o directamente copiarlo/clonarlo del repositorio de GitHub al equipo linux.

Repositorio: *https://github.com/franc0lb/ObligatorioPPDO*

Ubicación del archivo en el repositorio: *ObligatorioPPDO/ScriptBash/crearUsers.sh*

Explicación:

Primera parte – chequeo de parámetros y sintaxis de la lista:

Chequeo de parámetros:

La primera parte del script consiste en primero chequear que se corra como root/sudo, luego chequear que los parámetros recibidos sean correctos, se empieza por chequear si los parámetros recibidos son mínimo 1 y máximo 4. Se pone como máximo 4 porque dependiendo de las combinaciones de modificadores se podría dar el caso de que se reciban 4 parámetros, por ejemplo: `./crearUsers.sh archivo_lista -c $contraseña -i`

Ya que el script se armó para aceptar los modificadores en diferente orden, en el ejemplo anterior tendríamos 4 parámetros, 1 es la lista, el otro es -c el otro es la contraseña y el otro es -i.

Se pide mínimo un parámetro porque se requiere que se pase el archivo de la lista siempre.

Luego se chequea que el parámetro $1 exista, sea un archivo regular y se tenga permisos de lectura sobre el mismo.

Chequeo de sintáxis de la lista:

Se definen 2 variables las cuales corresponden a expresiones regulares.

Se utiliza wc -l para contar las líneas del archivo, y luego se usa grep con una expresión regular que coincide con líneas que contengan 4 campos separados por ":" y que el campo 1 no esté vacío, con grep -c puedo contar la cantidad de líneas válidas, y comparar las líneas totales con las líneas válidas, si no coinciden quiere decir que hay errores de sintaxis, hay líneas que no cumplen que existan 4 campos. Se muestra un mensaje en pantalla en caso de dar error.

En caso de que las líneas totales coincidan con las líneas validas se procede a volver a evaluar la lista usando otro regex, ya que el primer regex (contenido en la variable $regexCampos) se usó para detectar que haya 4 campos separados por 2 puntos, ahora hay que evaluar la sintaxis de los campos de forma más granular.

Esto se hace usando la misma lógica anterior, pero en este caso con otro regex, que evalúa que el campo 1 no contenga espacios, que tenga caracteres válidos y que no contenga 2 puntos, que el campo 3 no contenga espacios, contenga caracteres válidos , empiece por "/" y no contenga 2 puntos, que el campo 4 pueda ser *"si" "no" "SI" "NO"* y que no contenga 2 puntos, y que el campo 5 contenga caracteres válidos, que no tenga espacios , que empiece con "/" y no contenga 2 puntos, o que estos campos estén vacíos. También revisa que el campo 2 pueda estar vacío o contener cualquier carácter excepto 2 puntos, este es el campo de comentario por ende puede haber espacios, se puede dar el caso de que el comentario tenga espacios sin ningún carácter, lo cual se considera válido.

Si las líneas totales no coindicen con las válidas entonces se muestra en pantalla un error con todas las causas posibles.

En caso de no haber dado error los chequeos de sintaxis, ahora se chequea que el usuario no se repita, creando una variable que haga un cut del campo 1, luego un sort que me ordena por orden alfabético, y un uniq -d que me da los campos repetidos. En caso de haber usuarios repetidos se muestra un error indicando que hay usuarios repetidos.

Justificación de la rigurosidad del chequeo de sintaxis:

Todos estos chequeos de sintaxis se podrían hacer más simples, pero me pareció correcto hacerlo más granular, para evitar la mayor cantidad de errores antes de que el script empiece a crear los usuarios. También se usó el modificador -n para grep, lo que permite indicar el número de línea que no es válida, haciendo más fácil el identificar los errores.

Se hizo más hincapié en el chequeo de sintaxis de la lista que en hacer el script más simple, pero creo que es una buena idea porque asegura que el script creará la mayor cantidad de usuarios posibles con éxito, frente a una versión donde el chequeo de sintaxis sea más laxo, que llevaría a una mayor cantidad de usuarios que no serían creados por existir errores en la sintaxis.

Segunda parte - chequeo de modificadores:

Una vez se chequeó la lista hay que chequear qué modificadores (-i y -c) se aplicaron al script en caso de existir y el órden.

Las combinaciones posibles son:

```
-c $contraseña
```

```
-i
-ci $contraseña
-ic $contraseña
-c $contraseña -i
-i -c $contraseña
```

Para eso se crean 2 variables que funcionan como booleanos, una para definir si se ingresó el modificador de contraseña y otra para definir si se ingresó el modificador de mostrar log por pantalla.

También se crea una variable que va a contener la contraseña que se pasó como parámetro.

Se hacen varios if para ir detectando cada una de las combinaciones posibles y su órden.

Para los casos donde se cumple que exista -c se toma al siguiente parámetro como la contraseña, es decir cualquier cosa que se ponga luego de -c será la contraseña, incluido el uso de símbolos y de cualquier carácter, por ejemplo, si se ejecuta "`./crearUsers.sh $lista -c -i`" se toma a "`-i`" como la contraseña, no como un modificador, porque lo que venga luego del `-c` es la contraseña siempre.

IMPORTANTE: Para evitar problemas a la hora de asignar la contraseña se recomienda ponerla entre comillas, en caso contrario podría haber errores, si una contraseña contiene espacios no se tomaría en cuenta lo que haya luego del primer espacio, ya que la otra parte se consideraría un parámetro y daría error.

Tercera parte - creación de los usuarios:

Luego de que se verificaron los parámetros se deben crear los usuarios.

Se hace un for que va a iterar sobre la lista y sus campos, y se crean variables que corresponden a cada uno de sus campos usando cut.

Luego con varios if se detecta si la variable correspondiente al campo está vacía o no, en caso de estar vacía se asignan los valores por defecto, en este caso son valores que yo creé y asigné en variables.

Valores por defecto para el campo home, comentario, creación del home si/no y shell por defecto:

```
opdefcom="Comentario por defecto"
```

`opdefhome="/home/$c1"`

`opdefcrearhome="-m"`

`opdefshell="/bin/bash"`

Si el home no fue asignado por defecto se asigna `/home/$nombre_de_usuario`

Si el comentario no fue asignado se asigna `"Comentario por defecto"`

Si el crear o no el home no fue asignado se asigna `-m` que al ejecutarlo con el comando `useradd` significa que se cree el home, por ende, la opción por defecto es que el home sea creado.

Si el campo de shell por defecto no fue asignado se asigna a bash.

Los if funcionan de la siguiente forma: Si las variables correspondientes a los campos están vacías se sustituye su valor por el contenido de las variables que se setearon por defecto según la que corresponda, si las variables no están vacías entonces no se sustituye su contenido.

Luego se pasa a detectar si el usuario ya existe, con el comando id, en ese caso el usuario no será creado. Se crea una variable "$yaexiste" que pasa a tener el valor de "$c1", osea el nombre del usuario.

Luego se hace un if/elif para crear los usuarios. En el if se evalúa que "$yaexiste" no es igual a "$c1", osea que el usuario no esté ya creado, y se evalúa si el modificador -c está activo o no, y crea los usuarios asignando la contraseña pasada como parámetro.

En el elif se evalúa que el usuario no exista, y que el modificador -c no exista tampoco, por ende, se crean los usuarios sin asignar ninguna contraseña.

En ambos casos se agrega 1 a una variable que sirve como contador para poder saber el total de usuarios creados.

Mostrar mensajes por pantalla:

Para evaluar si hay que mostrar o no mensajes de la creación de usuarios por pantalla se evalúa que el usuario haya sido creado y que el modificador -i esté activo.

También se detecta en un if si el /home del usuario existe o no. Y se muestran los datos del usuario creado.

IMPORTANTE: Si el directorio home ya existía no se crea de nuevo, useradd se ejecutará con el modificador -m igualmente en caso de que el campo 4 sea “SI” o “si” o esté vacío (se aplica la opción por defecto), pero no va a volver a crear el home ni modificar sus datos, pero no hay nada

en el script que detecte antes de crear al usuario si el home existe, la detección de la existencia del home se hace luego y se muestra por pantalla los datos del usuario creado.

En caso de que el usuario no se haya podido crear se muestra un error en pantalla. Pero si un exit, el script sigue funcionando.

Luego se hace un echo de la cantidad de usuarios creados (esto se hace siempre con o sin modificador -i).

Uso de inteligencia artificial:

Se utilizó inteligencia artificial para mejorar la redacción del documento, y para la redacción y formato del archivo README.md en el repositorio.

Se utilizó también para la creación de la expresión regular.

Prompts:

¿Como bloquear los espacios en una expresión regular?

¿Como puedo en mi expresión regular hacer que en un campo se deba comenzar con "/" y que luego pueda contener una serie de caracteres?

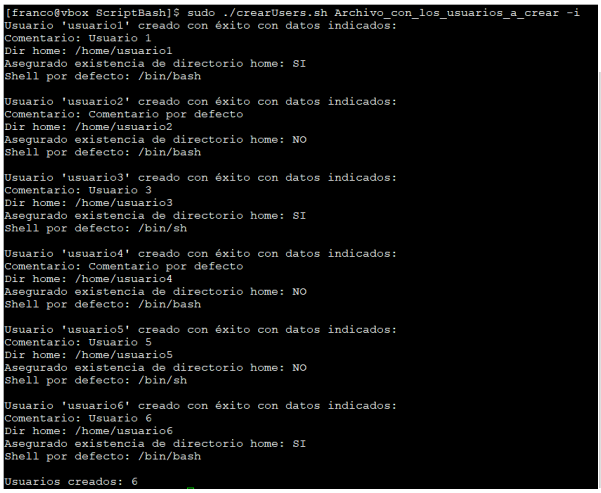
¿Podrías mejorar la redacción y estructura?

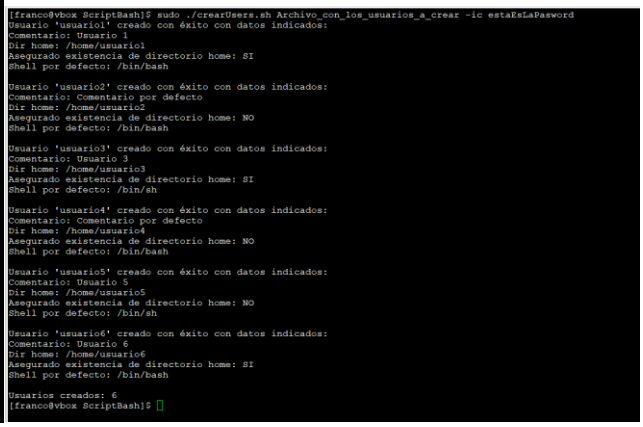
¿Podrías mejorar este archivo readme y formatearlo con markdown?

Testing:

Se adjuntan screenshots testeando el script de varias formas diferentes.

Creación exitosa de usuarios:






```
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear -c estaEsLaPasword
Usuarios creados: 6
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
```

```
[franco@vbox ScriptBash]$ cat Archivo_con_los_usuarios_a_crear
usuario1:Usuario 1:/home/usuario1:si:/bin/bash
usuario2::/home/usuario2:no:/bin/bash
usuario3:Usuario 3:/home/usuario3:si:/bin/sh
usuario4::/home/usuario4:no:/bin/bash
usuario5:Usuario 5:/home/usuario5:no:/bin/sh
usuario6:Usuario 6:/home/usuario6:si:/bin/bash
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear
Usuarios creados: 6
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$ tail -10 /etc/passwd
chrony:x:987:986:chrony system user:/var/lib/chrony:/sbin/nologin
dnsmasq:x:986:985:Dnsmasq DHCP and DNS server:/var/lib/dnsmasq:/usr/sbin/nologin
tcpdump:x:72:72::/:/sbin/nologin
franco:x:1000:1000:franco:/home/franco:/bin/bash
usuario1:x:1001:1001:Usuario 1:/home/usuario1:/bin/bash
usuario2:x:1002:1020:Comentario por defecto:/home/usuario2:/bin/bash
usuario3:x:1003:1003:Usuario 3:/home/usuario3:/bin/sh
usuario4:x:1004:1004:Comentario por defecto:/home/usuario4:/bin/bash
usuario5:x:1005:1005:Usuario 5:/home/usuario5:/bin/sh
usuario6:x:1006:1006:Usuario 6:/home/usuario6:/bin/bash
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$ ls /home | grep usuario
usuario1
usuario3
usuario6
[franco@vbox ScriptBash]$
```

Errores en sintaxis de la lista:

```
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear

Se detectaron errores en la sintaxis de la lista
Errores detectados: 3
Revise la sintaxis de la lista

Recuerde que el nombre del usuario no puede contener espacios ni caracteres raros
Recuerde que el home del usuario no puede contener espacios ni caracteres raros
Recuerde que el shell del usuario no puede contener espacios ni caracteres raros
Recuerde que el shell y el home en caso de existir deben empezar con '/'

Lineas invalidas:
3:usuario3:Usuario 3:/home/usuario3:si:/bi n/sh
4:usuario4::/ho me/usuario4:no:/bin/bash
6:usua rio6:Usuario 6:/home/usuario6:si:/bin/bash

[franco@vbox ScriptBash]$
```

```
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear
Error de sintaxis, hay usuarios repetidos en la lista
Lineas no validas:

6:usuario6:Usuario 6:/home/usuario6:si:/bin/bash
7:usuario6:Usuario 6:/home/usuario6:si:/bin/bash
[franco@vbox ScriptBash]$
```

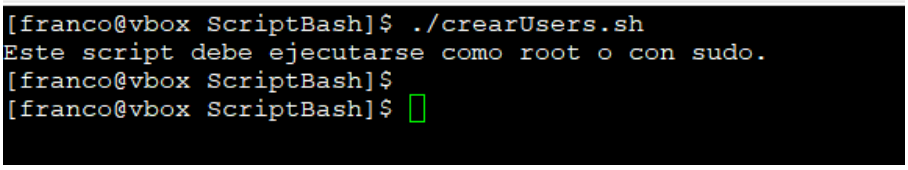
Usuario ya existe:

```
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear
Usuarios creados: 0
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear -i
ATENCIÓN: el usuario 'usuario1' no fue creado porque ya existe

ATENCIÓN: el usuario 'usuario2' no fue creado porque ya existe

ATENCIÓN: el usuario 'usuario3' no fue creado porque ya existe

ATENCIÓN: el usuario 'usuario4' no fue creado porque ya existe

ATENCIÓN: el usuario 'usuario5' no fue creado porque ya existe

ATENCIÓN: el usuario 'usuario6' no fue creado porque ya existe

Usuarios creados: 0
```

Errores por modificadores inválidos:

```
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear -c
Luego del modificador '-c' debe asignar la contraseña por defecto para los usuarios
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear -ic
Luego del modificador '-c' debe asignar la contraseña por defecto para los usuarios
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear -ci
Luego del modificador '-c' debe asignar la contraseña por defecto para los usuarios
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear -i aa
El parametro 3: 'aa' no es un modificador valido
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear -x
El parámetro 2: '-x' no es un modificador válido
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear -u
El parámetro 2: '-u' no es un modificador válido
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$ sudo ./crearUsers.sh Archivo_con_los_usuarios_a_crear -icu
El parámetro 2: '-icu' no es un modificador válido
[franco@vbox ScriptBash]$
[franco@vbox ScriptBash]$
```

Error por no ejecutar como root:



Ejercicio 2 - Python y AWS:

Se creó un script en Python con boto3 que sirve para automatizar el deploy de una aplicación de recursos humanos en AWS, accesible desde internet.

Código:

El código se encuentra accesible en el repositorio de github. O también en la carpeta .zip entregada, la cual contiene tanto este documento como todo el repositorio.

Repositorio: *https://github.com/franc0lb/ObligatorioPPDO/tree/main*

Ubicación del archivo en el repositorio: *ScriptAWS/ScriptAWS.py*

Requisitos:

Sistema operativo
Se necesita un equipo o máquina virtual con un sistema operativo Linux.

Paquetes necesarios
El equipo debe contar con lo siguiente instalado:

- pip
- boto3
- awscli
- python3.6 o superior

Instalación de paquetes
Los siguientes comandos corresponden a una distribución basada en Red Hat (por ejemplo, CentOS). Si se utiliza otra distribución, se deben adaptar los comandos.

```
dnf install pip

dnf install python3
```

```
pip install boto3
dnf install awscli
```

Configuración de credenciales de AWS

Se debe configurar el acceso a AWS. Ejecutar:

```
aws configure
```

Luego ingresar las credenciales correspondientes.
En AWS Academy, estas credenciales se encuentran en la sección **AWS Details**. Es importante configurar la región como **us-east-1**.

Variables de entorno:

Se deben configurar 2 variables de entorno previo a ejecutar el script, que corresponden a la contraseña del usuario admin para la base de datos y del usuario admin para la aplicación.

Comandos:

```
export RDS_ADMIN_PASSWORD=*******
export RDS_APP_PASSWORD=*******
```

Ejecución del script

Ejecutar el script con:

```
python ObligatorioPPDO/ScriptAWS/ScriptAWS.py
```

También puede abrirse en Visual Studio Code y ejecutarse desde allí, asegurándose de tener instalado el módulo de Python.

Explicación:

1 - librerías y chequeos:

Al principio del script se definen las librerías necesarias.

Luego se chequean que existan 2 variables de entorno, que corresponden a la contraseña del usuario admin de la base de datos y del usuario admin de la aplicación.

En caso de no existir se lanza error y se detiene el script, ya que no tiene sentido ejecutar el resto del script si no se tienen las contraseñas necesarias.

En los requisitos se explicó que estas contraseñas se guardan en variables de entorno previo a ejecutar el script.

2 - bucket S3 y archivos:

Este bloque del script automatiza el proceso de descargar un repositorio desde GitHub, extraer archivos específicos y subirlos a un bucket de Amazon S3. A continuación se detalla cada etapa del proceso.

Los archivos de la app se encuentran alojados en mi respositorio.

Se definen variables para el nombre del bucket, la carpeta local donde estarán alojados los archivos, el nombre de la carpeta remota en el bucket S3 donde se subirán los archivos, y una variable que contiene el URL para descargar en formato zip mi repositorio.

Se descarga el repositorio y se extrae en una carpeta local temporal.

Luego el script construye la ruta hacia la subcarpeta donde se encuentran los archivos que deben surbirse al S3 y los lista por pantalla.

Después crea el bucket S3, y con un for recorre todos los archivos previamente listados y los sube uno por uno al bucket.

3 - creación de instancia EC2 y ejecución de comandos dento de la instancia:

En esta parte el script primero crear la instancia EC2 y le agrega un tag para identificarla.

Cuando la instancia está en estado running se procede a ejecutar comandos para instalación de paquetes necesarios, inicialización de servicios y asignación de permisos. También para la descarga de los archivos que se subieron al bucket S3 y moverlos a las carpetas correspondientes.

Se hace un while que lo que hace es esperar el resultado de los comandos. Y fuera del bucle con un print se muestra el resultado.

4 - creación de security groups:

En esta parte se crean 2 security groups, uno para la instancia EC2 y el otro para la instancia RDS.

Se crea el security group para EC2 con el nombre web-sg-boto3, y se habilita tráfico entrante desde internet hacia el puerto 80, para que nos podamos conectar la instancia EC2 donde va a estar corriendo la app.

Se crea el security group para RDS con el nombre rds-mysql-sg, y se habilita el tráfico desde el sg de EC2 hacia el puerto 3306, para que la instancia EC2 tenga conexión con la base de datos.

Luego se asocia el secuirty group de EC2 al instance ID de la instancia EC2.

5 - Creación de la base RDS:

Luego se crea la base de datos. Se definen los parámetros de la base de datos: nombre de la base, nombre de la instancia, usuario de la app, usuario de admin de la base, y contraseñas (obtenidas previamente por variables de entorno). También se asocia el security group previamente creado para RDS a la instancia.

Luego se espera a que la base esté activa y se extrae la dirección del endpoint, que será usada por la aplicación para conectarse a la base.

Luego se ejecutan comandos dentro de la instancia EC2, para generar el archivo .env dentro del servidor web y se ejecuta un archivo SQL que contiene la creación de tablas, usuarios y datos iniciales.

Uso de inteligencia artificial:

Se utilizó inteligencia artificial para mejorar la redacción del documento, y para la redacción y formato del archivo README.md en el respositorio.

Se utilizó también para la creación del script y para entender cómo funciona boto3 y las distintas librerías, comandos y estructuras que se puede armar en el script.

Prompts:

¿Podrías mejorar la redacción y estructura?

¿Podrías mejorar este archivo readme y formatearlo con markdown?

¿Como puedo crear un for para subir archivos a un bucket S3?

¿Como puedo ejecutar comandos dentro de una instancia EC2 y cómo puedo saber si los comandos se ejecutaron exitosamente?

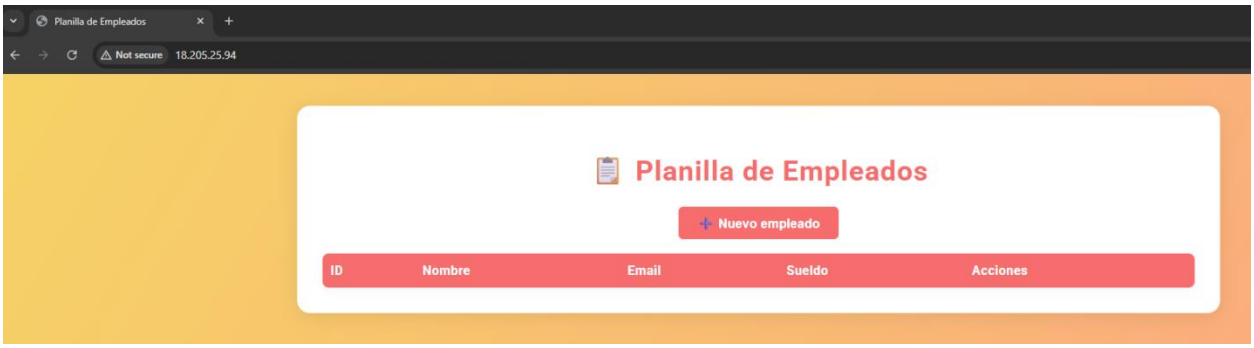
¿Como puedo descargar los archivos de una carpeta alojada en un bucket S3 a mi instancia EC2, alojarlos en una carpeta en concreto, excluir uno de los archivos y luego mover ese archivo a otra carpeta?

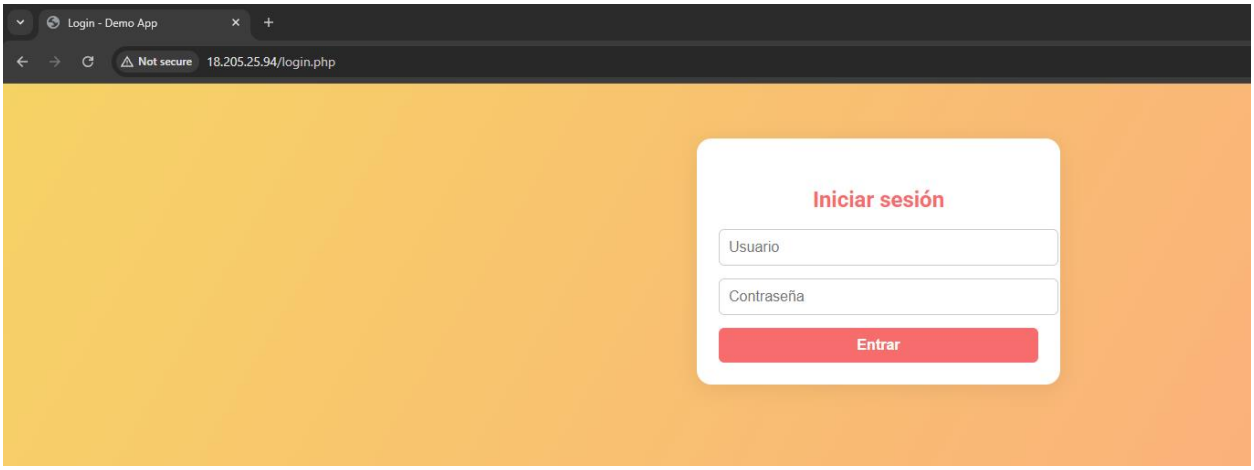
¿Como puedo saber si los comandos se ejecutaron correctamente?

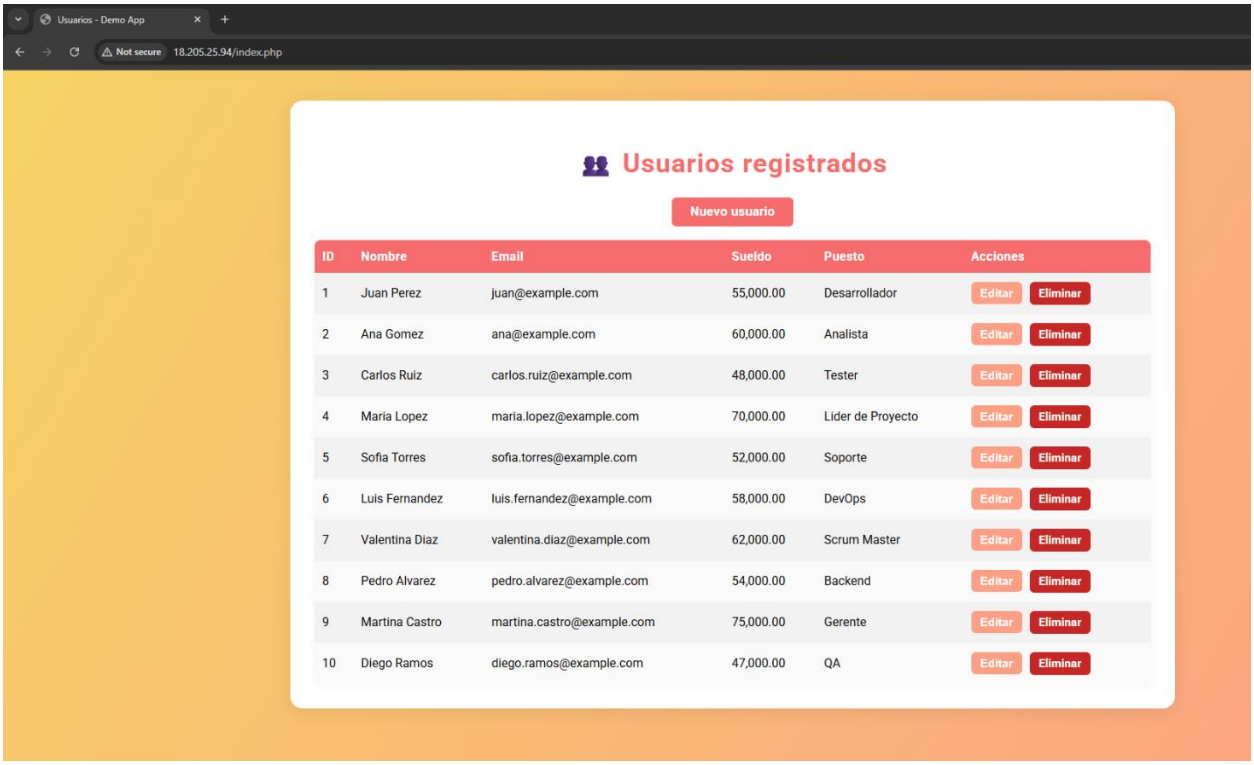
¿Como puedo asociar el instance ID de una instancia EC2 a un security group?

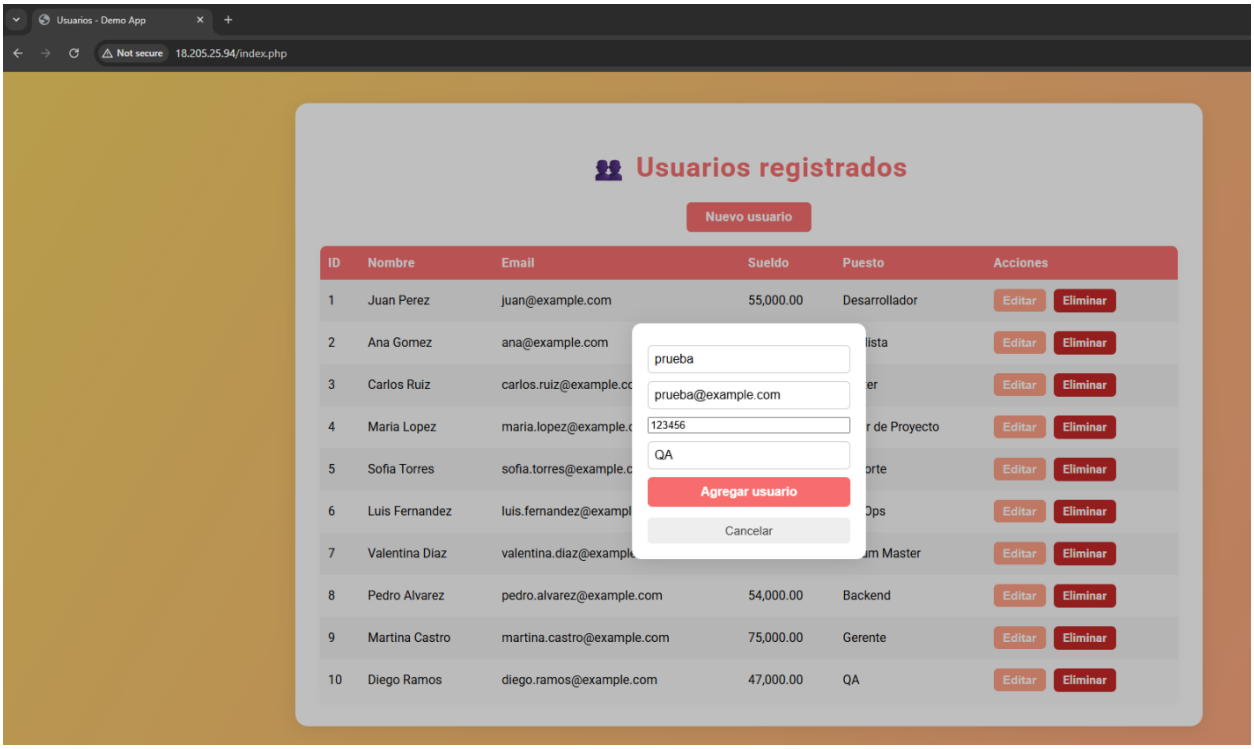
¿Cómo puedo obtener el endpoint de mi instancia RDS?

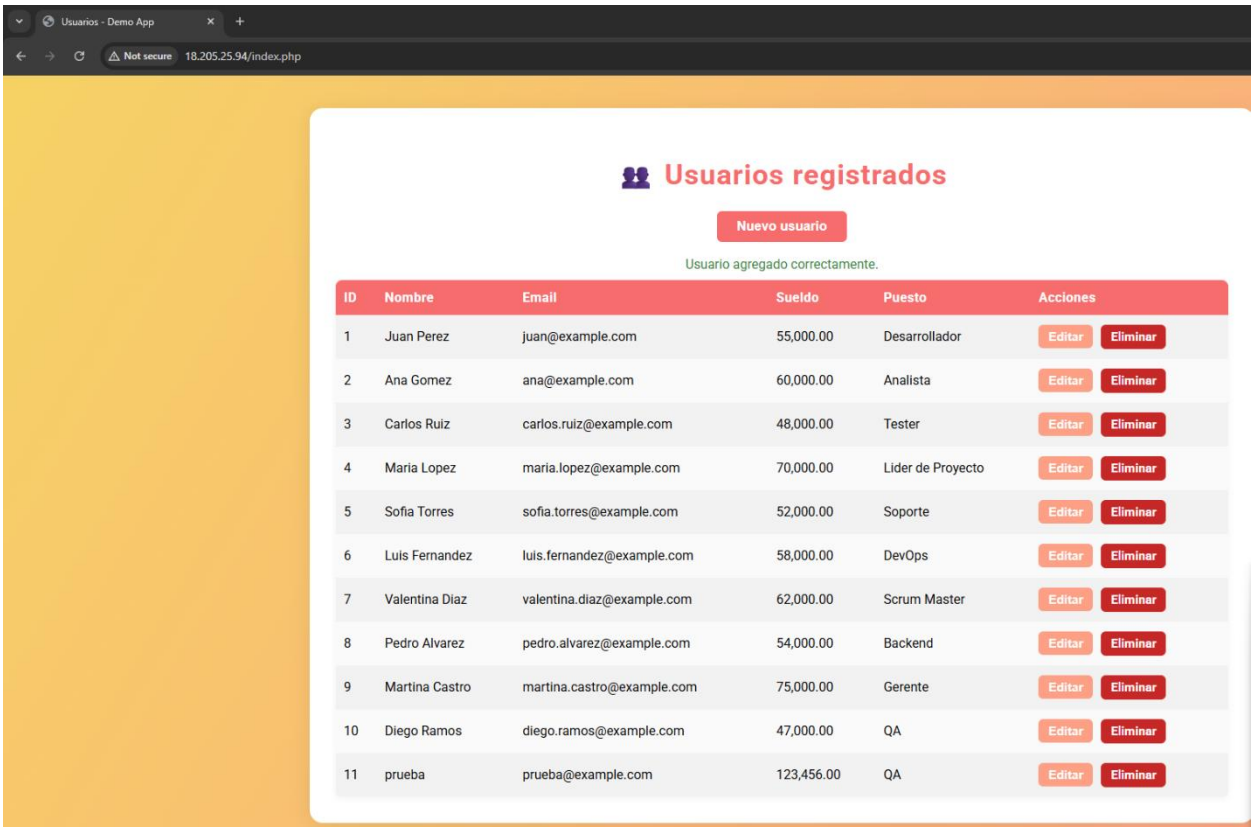
Testing:






Usuarios - Demo App
Not secure
18.205.25.94/index.php
Usuarios registrados
Nuevo usuario
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
5 Sofia Torres sofia.torres@example.com 52,000.00 Soporte Editar Eliminar
6 Luis Fernandez luis.fernandez@example.com 58,000.00 DevOps Editar Eliminar
7 Valentina Diaz valentina.diaz@example.com 62,000.00 Scrum Master Editar Eliminar
8 Pedro Alvarez pedro.alvarez@example.com 54,000.00 Backend Editar Eliminar
9 Martina Castro martina.castro@example.com 75,000.00 Gerente Editar Eliminar
10 Diego Ramos diego.ramos@example.com 47,000.00 QA Editar Eliminar


Usuarios - Demo App
Not secure
18.205.25.94/index.php
Usuarios registrados
Nuevo usuario
prueba
prueba@example.com
123456
QA
Agregar usuario
Cancelar


Usuarios - Demo App
Not secure
18.205.25.94/index.php
Usuarios registrados
Nuevo usuario
Usuario agregado correctamente.
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
5 Sofia Torres sofia.torres@example.com 52,000.00 Soporte Editar Eliminar
6 Luis Fernandez luis.fernandez@example.com 58,000.00 DevOps Editar Eliminar
7 Valentina Diaz valentina.diaz@example.com 62,000.00 Scrum Master Editar Eliminar
8 Pedro Alvarez pedro.alvarez@example.com 54,000.00 Backend Editar Eliminar
9 Martina Castro martina.castro@example.com 75,000.00 Gerente Editar Eliminar
10 Diego Ramos diego.ramos@example.com 47,000.00 QA Editar Eliminar
11 prueba prueba@example.com 123,456.00 QA Editar Eliminar