Universidad ORT Uruguay

Facultad de Ingeniería: Escuela de Tecnología

PROGRAMACION PARA DEVOPS

Marcelo Sosa Nº141855

Hernán Pintos Nº151357

Grupo: N4A

Docentes: Leonardo Genta - Guillermo Ferradas

Declaracion de autoría

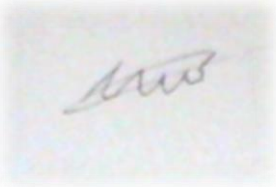
Nombres de los autores: Marcelo Sosa
Hernán Pintos

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.

Volver al índice

SECCION II – USO DE GIT Y RECURSOS

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

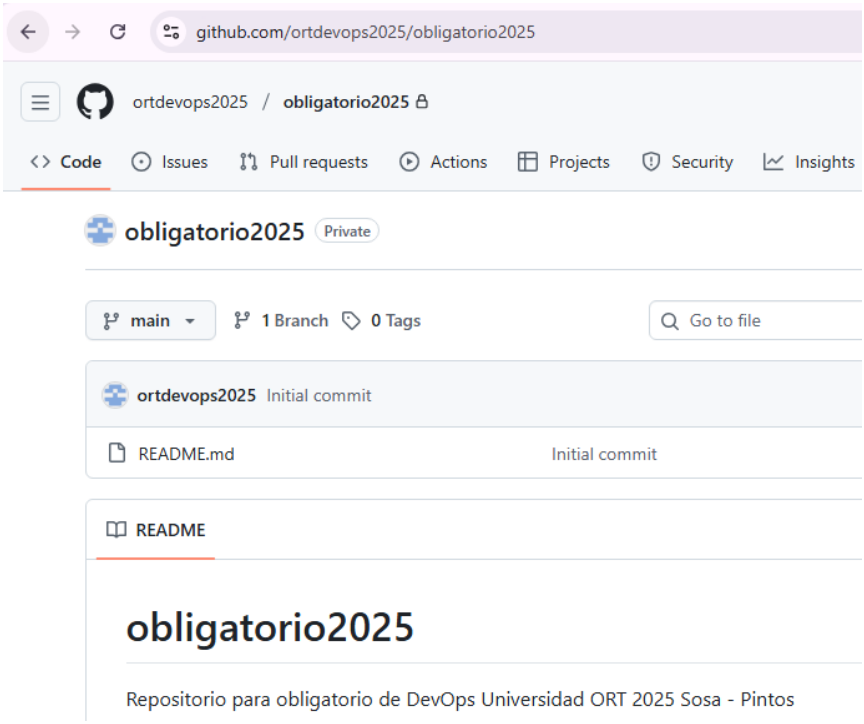
Si bien todos los documentos están en el formato solicitado en el repositorio de GIT creado con la finalidad de la realización de este trabajo, se incluye en este documento, en las siguientes secciones, el código de ambos scripts, pruebas de uso y contenido de archivos de ejemplo, así como también el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

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

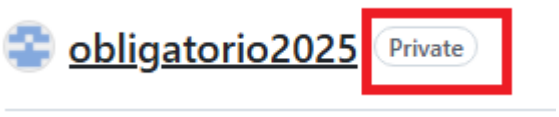
REPOSITORIO DE GIT

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 de este.



Volver al índice

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.

Versión 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
#Fecha Modificado 24/11/2025

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 utilizó solo el parametro de informacion
            then
                  flaginfo=1
                  archivousu="$2"
            else
                  if  [ "$1" = "-c" ] #Verifica si se utilizó solo el parametro de password
                  then
                        password="$2"
                        archivousu="$3"
                  else
                        archivousu="$1" #Procsa el archivo si no se utilizron 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 "$archivousu" ] #Verifica que el archivo sea regular
then
	echo "$archivousu no existe o no es regular" >&2
	exit 3
fi

if [ ! -r "$archivousu" ] #Verifica que el archivo tenga read only
then
	echo "$archivousu 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 $(cat "$archivousu")
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 $archivousu 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 #el comando id retorna 0 si es exitoso o distinto de 0 si no lo es
		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
```

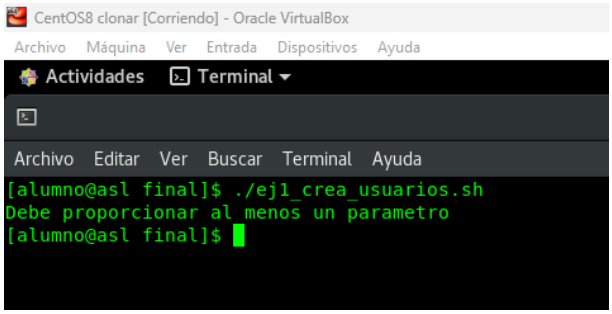
```
        if [ "$CREARHOME" = "SI" ]
        then
                # con -m se crea el directorio home si no existe
                useradd -m -c "$COMENTARIO" -d "$HOME" -s "$SHELL" "$USUARIO" >/dev/null 2>&1
        else
                # con -M se evita la creacion del directorio home
                useradd -M -c "$COMENTARIO" -d "$HOME" -s "$SHELL" "$USUARIO" >/dev/null 2>&1
        fi

        if [ $? -eq 0 ] #verifica que el comando de agregar usuario haya sido exitoso, (exit 0)
        then
                if [ -n "$password" ] # se chequea que la variable password no este vacia
                then
                        # se establece la contraseña del usuario usando chpasswd, ocultando errores
                        # chpasswd asigna la contraseña al usuario leida desde la entrada estandar
                        echo "$USUARIO:$password" | chpasswd 2>/dev/null
                fi

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

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

                        echo "Usuario $USUARIO creado con éxito con datos indicados:"
                        # echo -e "\t" para lograr identacion
                        echo -e "\tComentario: ${COMENTARIO:-<valor por defecto>}"
                        echo -e "\tDir home: ${HOME:-<valor por defecto>}"
                        echo -e "\tAsegurado existencia de directorio home: ${CREARHOME:-<valor por defecto>}"
                        echo -e "\tShell por defecto: ${SHELL:-<valor por defecto>}"
                        echo
                fi
        else
                echo "ATENCION: el usuario $USUARIO no pudo ser creado"
        fi
    fi
#done < "$archivousu"
done

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

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

Volver al índice

PRUEBA DE USO:

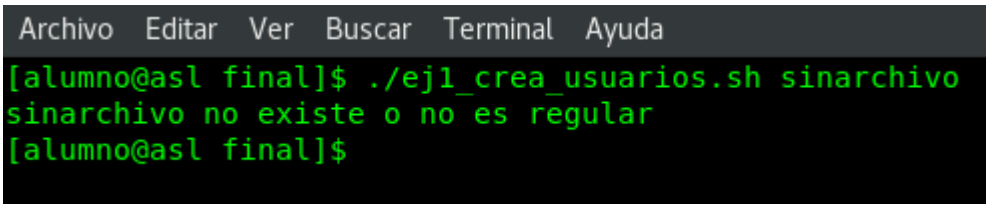
Se procede a mostrar ejemplos de uso y los resultados obtenidos incluidos algunos errores testeados:

1 – Script sin Parámetros:



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



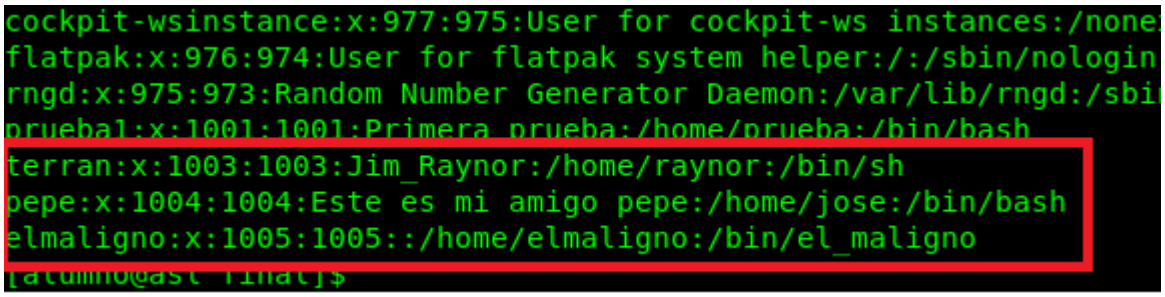
3 – Uso normal con un archivo de usuarios que tiene 3 usuarios válidos para agregar y uno que ya existe:

```
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



Volver al índice

SECCION IV – SCRIPT DE PYTHON

Para la creación 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.

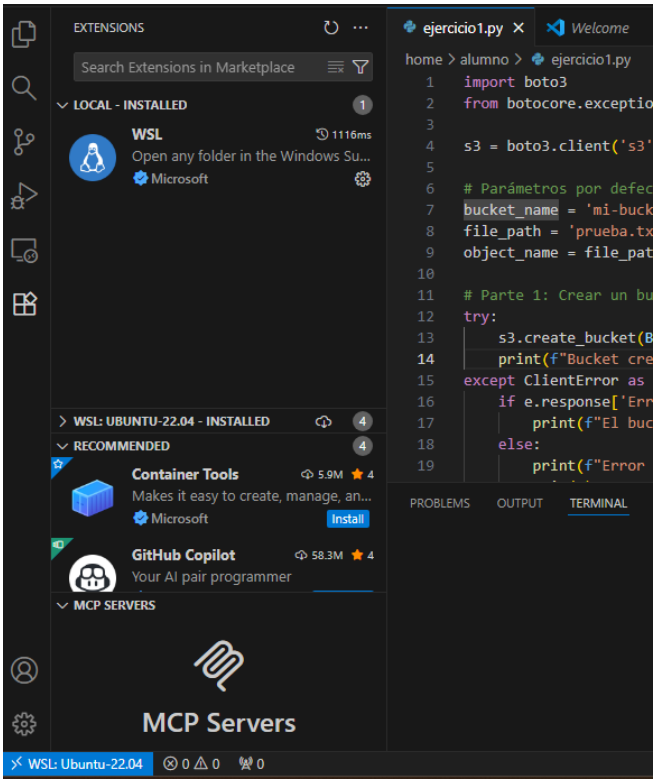
En este caso, se configuro para el desarrollo una instancia de WSL en Windows 11 utilizando la imagen de Ubuntu 22 proporcionada por el repositorio de WSL.

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 del script, se utilizó Visual Studio Code con los plugins de Boto3, WSL y Python instalados.



Volver al índice

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

Volver al índice

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 ejecutes los scripts.

Suponiendo que se ejecute este proyecto desde una máquina de LINUX, se debe tener el AWS CLI instalado y configurado, 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.

```
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
```

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

También 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:

En caso de que debamos alterar algún valor, se puede correr el comando nuevamente o acceder al archivo de configuración el cual está 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

Volver al índice

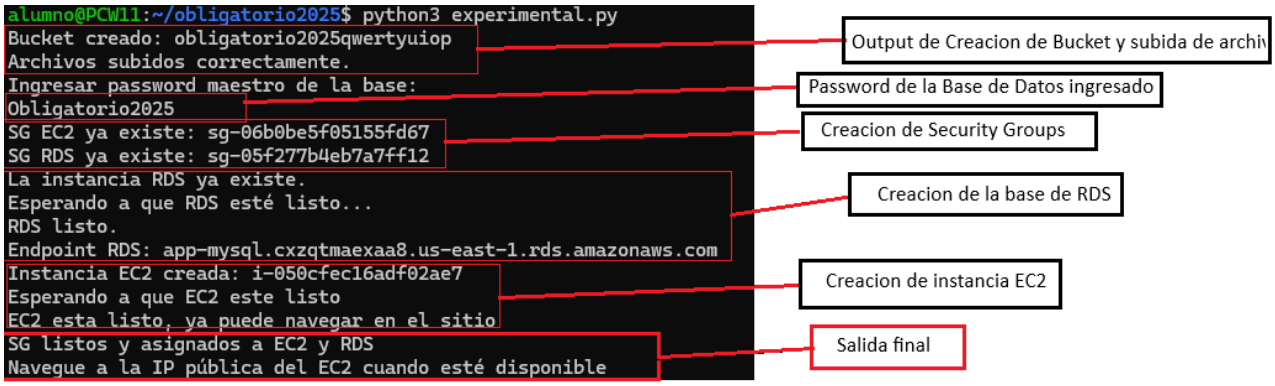
PASOS PARA DESPLIEGUE DE APLICACION USANDO SCRIPTS:

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 instalación y despliegue de la aplicación:

python3 obligatorio.py



Volver al índice

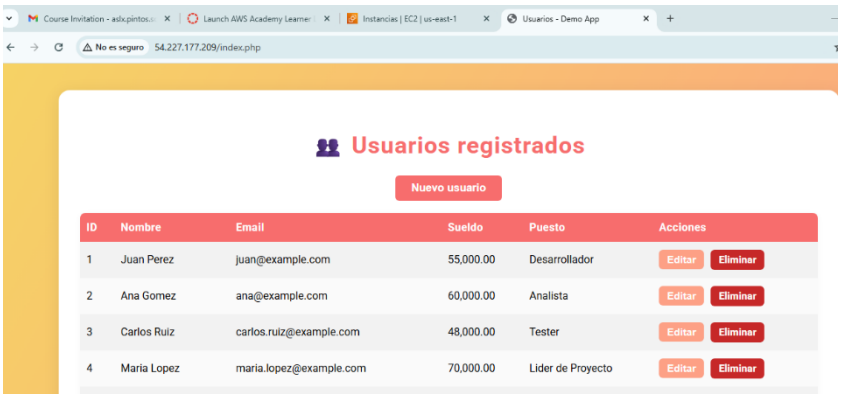
PRUEBA DE USO

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

Se utiliza el log in:
admin
admin123

Se debe desplegar correctamente la información de la aplicación:



Volver al índice

SCRIPT DE PYTHON

```
import boto3
from botocore.exceptions import ClientError

import os
import random

# SECCION S3 - Creacion de bucket y copia de archivos
s3 = boto3.client('s3')

# se randomiza el nombre de bucket
aleatorio = str(random.randint(0, 999999))


bucket_name = 'obligatorio2025enusnh' + aleatorio


# Crea el bucket si no exsite
# En caso de existir y que sea nuestro, entra en el except y continua la ejecucion del codigo
try:
    s3.create_bucket(Bucket=bucket_name)
    print(f"Bucket creado: {bucket_name}")
except ClientError as e:
    # siendo "e" la variable donde se guarda el tipo de error ocurrido
    if e.response['Error']['Code'] == 'BucketAlreadyOwnedByYou':
        print(f"El bucket {bucket_name} ya existe y es tuyo.")
    else:
        print(f"Error creando bucket: {e}")
        exit(1)

# Subir los archivos de la aplicacion
try:
    s3.upload_file('aplicacion/app.css', bucket_name, 'app.css')
    s3.upload_file('aplicacion/app.js', bucket_name, 'app.js')
    s3.upload_file('aplicacion/config.php', bucket_name, 'config.php')
    s3.upload_file('aplicacion/index.html', bucket_name, 'index.html')
    s3.upload_file('aplicacion/index.php', bucket_name, 'index.php')
    s3.upload_file('aplicacion/init_db.sql', bucket_name, 'init_db.sql')
    s3.upload_file('aplicacion/login.css', bucket_name, 'login.css')
    s3.upload_file('aplicacion/login.html', bucket_name, 'login.html')
    s3.upload_file('aplicacion/login.js', bucket_name, 'login.js')
    s3.upload_file('aplicacion/login.php', bucket_name, 'login.php')

    print("Archivos subidos correctamente.")
except ClientError as e:
    # siendo "e" la variable donde se guarda el tipo de error ocurrido
    # se muestra en pantalla el error ocurrido al subir los archivos
    print(f"Error subiendo archivo: {e}")

# CREACION de variables de base de datos

# Solicitud al usuario del password maestro de la base de datos
# El password se guardara en la variable de entorno para ser usado de manera segura
# print("Ingresar password maestro de la base: ")

RDS_ADMIN_PASSWORD = input("Ingresar password maestro de la base: ")

while len(RDS_ADMIN_PASSWORD) < 8 :
    print("El password debe ser minimo 8 caracteres")
    RDS_ADMIN_PASSWORD = input("Ingresar password maestro de la base: ") #Esto hace que no sea necesario hardcodear el
password en el codigo

os.environ["RDS_ADMIN_PASSWORD"] = RDS_ADMIN_PASSWORD
```

```
# Se definen las variables de configuraion de la base de datos
# Se randomiza nombre de instancia de DB

DB_INSTANCE_ID = "appw-mysql"
DB_NAME = "demo_db"  #Nombre sacado del archivo sql de ejemplo de la aplicacion, si es diferente no funciona sin modificar el .sql
DB_USER = "admin"
DB_PASS = RDS_ADMIN_PASSWORD

#Se comienzan a crear las instancias de EC2 y RDS
rds = boto3.client('rds')

ec2 = boto3.client('ec2')


####################################
#Creacion de Grupos de Seguridad####
####################################

# Se define nombre del security group que usaremos para la instancia ec2 y RDS
# se randomiza nombres de security groups

sg_name = "web-sg-boto3"
sg_rds_name = "rds-sg-boto3"

# Grupo de Seguridad de EC2
try:
    # Se intenta crear el security group para ec2
    response = ec2.create_security_group(
        GroupName=sg_name,
        Description="Permitir trafico web desde cualquier IP"
    )
    # Se guarda el id del nuevo security group
    sg_ec2 = response["GroupId"]

    # Se abre el puerto 80 (http)
    ec2.authorize_security_group_ingress(
        GroupId=sg_ec2,
        IpPermissions=[{
            'IpProtocol': 'tcp',
            'FromPort': 80,
            'ToPort': 80,
            'IpRanges': [{'CidrIp': '0.0.0.0/0'}] # Se permite acceso desde cualquier IP
        }]
    )
    print(f"SG EC2 creado: {sg_ec2}")
except ClientError as e:
    # En caso de que el security group ya exista, aws devuelve en el except "InvalidGroup.Duplicate"
    if "InvalidGroup.Duplicate" in str(e):
        # En este caso entonces solo se toma el id del grupo existente en lugar de crear uno nuevo
        sg_ec2 = ec2.describe_security_groups(GroupNames=[sg_name])['SecurityGroups'][0]['GroupId']
        print(f'SG EC2 ya existe: {sg_ec2}")
    else:
        # En el caso de otro errore se detiene la ejecucion
        raise

# SG RDS
try:
    # Se intenta crear el security group para rds
    response = ec2.create_security_group(
        GroupName=sg_rds_name,
        Description="ACCESSSQL"
    )
    sg_rds = response["GroupId"]
```

```
    # Se abre el puerto 3306 (MariaDB) solo para el security group del ec2
    ec2.authorize_security_group_ingress(
        GroupId=sg_rds,
        IpPermissions=[{
            "IpProtocol": "tcp",
            "FromPort": 3306,
            "ToPort": 3306,
            "UserIdGroupPairs": [{"GroupId": sg_ec2}]
        }]
    )
    print(f"SG RDS creado: {sg_rds}")
except ClientError as e:
    if "InvalidGroup.Duplicate" in str(e):
        # En este caso entonces solo se toma el id del grupo existente en lugar de crear uno nuevo
        sg_rds = ec2.describe_security_groups(GroupNames=[sg_rds_name])["SecurityGroups"][0]["GroupId"]
        print(f"SG RDS ya existe: {sg_rds}")
    else:
        # En el caso de otro errore se detiene la ejecucion
        raise

#################################
##Creacion de instancia de DB RDS
#################################

try:
    rds.create_db_instance(
        DBInstanceIdentifier=DB_INSTANCE_ID,
        AllocatedStorage=20,
        DBInstanceClass='db.t3.medium',
        Engine='mariadb',
        MasterUsername=DB_USER,
        MasterUserPassword=DB_PASS,
        DBName=DB_NAME,
        VpcSecurityGroupIds=[sg_rds],
        PubliclyAccessible=False,
        BackupRetentionPeriod=0
    )
    print("Creando instancia de RDS")
except rds.exceptions.DBInstanceAlreadyExistsFault:
    print("La instancia RDS ya existe")

# Esperar a que RDS esté disponible
print("Esperando RDS para continuar")
waiter = rds.get_waiter('db_instance_available')
waiter.wait(DBInstanceIdentifier=DB_INSTANCE_ID)
print("RDS creado y en linea")

# 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}")

#####################################
##Procesamiento de instancia de EC2##
#####################################

#Bloque de User data con comandos para instalar las dependencias y preparacion de la aplicacion
user_data = f"#!/bin/bash

dnf clean all
dnf makecache
dnf -y update

dnf -y install httpd php php-cli php-fpm php-common php-mysqlnd mariadb105

systemctl enable --now httpd
```

```
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

aws s3 sync s3://{bucket_name} /var/www/html/

mv /var/www/html/init_db.sql /var/www/init_db.sql

#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

mysql -h {RDS_ENDPOINT} -u {DB_USER} -p{DB_PASS} {DB_NAME} < /var/www/init_db.sql

cat << 'EOF' > /var/www/.env
DB_HOST={RDS_ENDPOINT}
DB_NAME={DB_NAME}
DB_USER={DB_USER}
DB_PASS={DB_PASS}

APP_USER=admin
APP_PASS=admin123
EOF

chown apache:apache /var/www/.env
chmod 600 /var/www/.env

chmod -R 755 /var/www/html
chown -R apache:apache /var/www/html

systemctl restart httpd
systemctl restart php-fpm
'''

response = ec2.run_instances(
    ImageId='ami-06b21ccaeff8cd686',
    InstanceType='t2.micro',
    MinCount=1,
    MaxCount=1,
    SecurityGroupIds=[sg_ec2],
    IamInstanceProfile={'Name': 'LabInstanceProfile'},
    UserData=user_data
)

instance_id = response['Instances'][0]['InstanceId']
print(f"Instancia EC2 creada: {instance_id}")

ec2.create_tags(Resources=[instance_id], Tags=[{'Key': 'Name', 'Value': 'webserver-devops'}])

print("Esperando a que EC2 este listo")
waiter = ec2.get_waiter('instance_running')
waiter.wait(InstanceIds=[instance_id])

print("EC2 esta listo, ya puede navegar en el sitio")
print("SG listos y asignados a EC2 y RDS")
print("Navegue a la IP pública del EC2 cuando esté disponible")
```

Volver al índice

SECCION VII – RESULTADOS

Luego de la creación de ambos scripts, así como de la documentación requerida, pudimos sacar las siguientes conclusiones y resultados de nuestro trabajo obligatorio.

Sección Bash:

Se logro crear un script de bash que satisficiere por completo los requerimientos del trabajo obligatorio.
El script logra agregar usuarios con o sin el uso de los parámetros opcionales haciendo varias verificaciones, algunas agregadas por iniciativa del equipo como por ejemplo la verificación de correr el script como root.

Se realizaron las pruebas de uso pertinentes y se adjuntaron en este mismo documento, se analizaron varias alternativas y se opto siempre por aquella que el equipo estuviese de acuerdo en dejar como solución final.

Sección Python:

Se consiguió realizar un script de Python para la realización de la tarea de publicación de una aplicación web con su correspondiente base de datos.
Si bien en un principio se optó por scripts independientes para cada sección de este despliegue, se opto gracias a la guía del Profesor, una aproximación de un solo script que contuviese todas las actividades a realizar, esto simplifico varias tareas y aumento la seguridad en ciertos aspectos como el manejo del passowrd maestro de la base de datos.

Conclusiones:

Las conclusiones, fueron ampliamente favorables según nuestro propio criterio ya que pudimos lograr el completo y correcto funcionamiento, hasta donde pudimos probar, de nuestros códigos.
Logramos superar varios retos que se nos impusieron a la hora de desarrollar ambos scripts gracias a la guía docente y la investigación propia.

Volver al índice

DIFICULTADES ENCONTRADAS

Compatibilidad de comandos

Si bien no fue una gran dificultad, se intentó que el script de bash, pudiese funcionar tanto en CentOS como en UBUNTU debido a que deseábamos que fuese universal.
Si bien la mayoría de los comandos son iguales, pueden darse algunas diferencias en algunas estructuras, creemos que esto fue cubierto en el código satisfactoriamente.

Password por defecto

Se necesito consultar mediante IAG, un ejemplo de comando para poder agregar un password por defecto en caso de que este no sea colocado por el usuario al momento de ejecutar el script.

Reutilización de ambientes de pruebas en BASH

Al lograr nuestras primeras pruebas exitosas en BASH, aconteció el problema de que el SO se nos llenaba de usuarios de prueba, esto fue solucionado reutilizando la lógica del script principal, más compacto, en el que pasándole como parámetro el archivo de usuarios, este removía los mismos.

Problema en nombre de base de datos – Python

La base de datos de la aplicación tenía un nombre especifico que debía ser el mismo que el utilizado en a la configuración, esto retraso el funcionamiento del script hasta que pudimos abrir el .sql e identificar el problema.

Fraccionamiento de Script de Python

Este quizás fue el mayor obstáculo encontrado, se intentó en una primera instancia, crear un script separado por cada tarea de Python a realizar (Ec2, RDS, etc) pero traía enormes dificultades a la hora de lidiar con variables que debían se reutilizadas, por ejemplo, el nombre de la base de datos o el password de esta.
Se opto por un script único en un orden específico para solucionar el problema.

Volver al índice

ANEXO I – Uso de Inteligencia Artificial Generativa

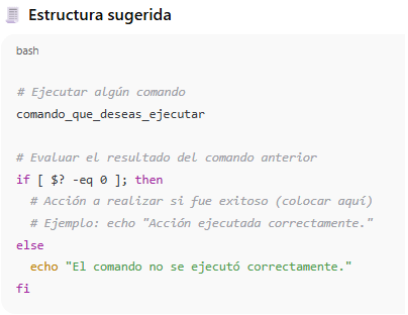
Para la realización de este trabajo, el uso de IAG fue utilizado EXCLUSIVAMENTE para la creación de gráficos específicos o conjunto de comandos básicos con motivo de verificación.

A continuación, 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 acción, sino, devolver un mensaje de éxito.

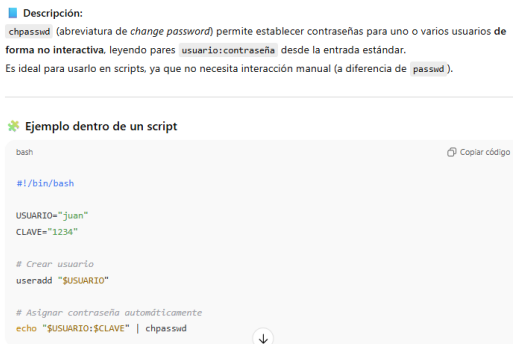


Estructura sugerida

```bash
# Ejecutar algún comando
comando_que_deseas_ejecutar

# Evaluar el resultado del comando anterior
if [ $? -eq 0 ]; then
  # Acción a realizar si fue exitoso (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 password automáticamente dentro de un script de bash junto a useradd

IAG 3 – Sugerencia de estructura

https://chatgpt.com/

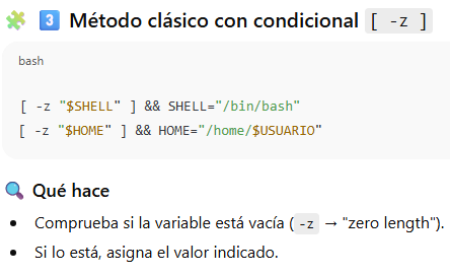
```bash
if [ "$CREAR_HOME" = true ]; then
  # Crear usuario con su directorio home personalizado
  echo "Creando usuario $USUARIO con su directorio home..."
  $CMD --home "/home/$USUARIO" --shell "$SHELL_DEF" --gecos "$COMENTARIO" --disabled-password "$US
else
```

Prompt: Ejemplo de bloque en Bash que cree usuarios con el comando adduser, compatible en ubuntu y centos.
Suponer una variable true para crear un home y un false entregaría 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



IAG 5 – Orden de deploy

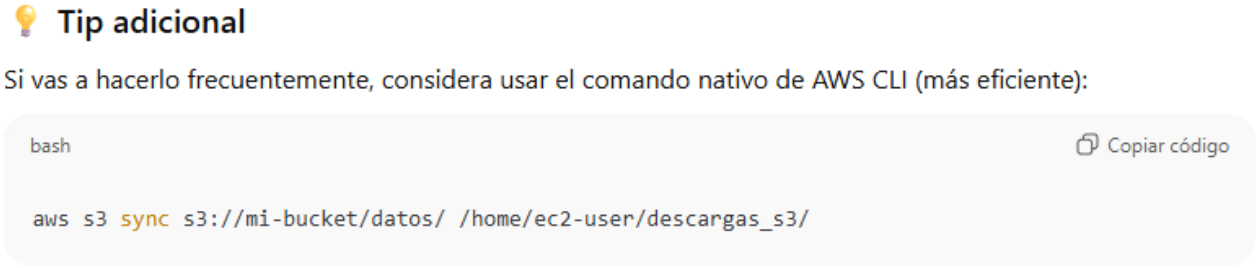
https://chatgpt.com/

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



IAG 7 – 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 8 – RDS Security Group

https://chatgpt.com/

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

IAG 9 – 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 10 – 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 11 – Logos

https://chatgpt.com/

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

IAG 12 – Formato de README.MD

https://chatgpt.com/

Prompt: (se copia el contenido del readme.md) Generar formato para README.MD de github sin modificar las imágenes insertadas o modificar el codigo fuente de BASH o PYTHON comentado. No utilizar iconos, solo formatear de manera profesional estándar el mismo.

Volver al índice

BIBLIOGRAFIA

Material General

Material de curso de la Materia Programación para DevOps de la carrera Analista en infraestructura informática.Facultad de ingeniería,Universidad ORT Uruguay.

https://aulas.ort.edu.uy/course/view.php?id=4143

Material específico de Python

Repositorio de Github provisto por el docente Guillermo Ferradas

https://github.com/ORT-AII-ProgramacionDevOps

Volver al índice