



## **Mantenimiento de usuarios y grupos en diferentes servidores**

---

### **Administración de sistemas operativos**



## Índice

1. Introducción .....	3
2. Preparación del entorno.....	4
3. Exportar usuarios y grupos locales de un servidor.....	5
4. . Conectarse desde PowerShell a otro servidor EC2 por SSH.....	6
5. Replicar usuarios y grupos en EC2 .....	8
6. Mantenimiento automático mediante tarea programada. ....	10
7. Ampliación soporte para múltiples servidores EC2.....	14
8. Crear un menú para facilitar la interacción .....	17



## 1. Introducción

En este trabajo se va a explicar cómo administrar usuarios y grupos en diferentes servidores, especialmente en servidores EC2 de Amazon Web Services (AWS). La idea es automatizar el proceso de gestión, para que no sea necesario hacerlo manualmente cada vez, lo que puede ser lento y propenso a errores.

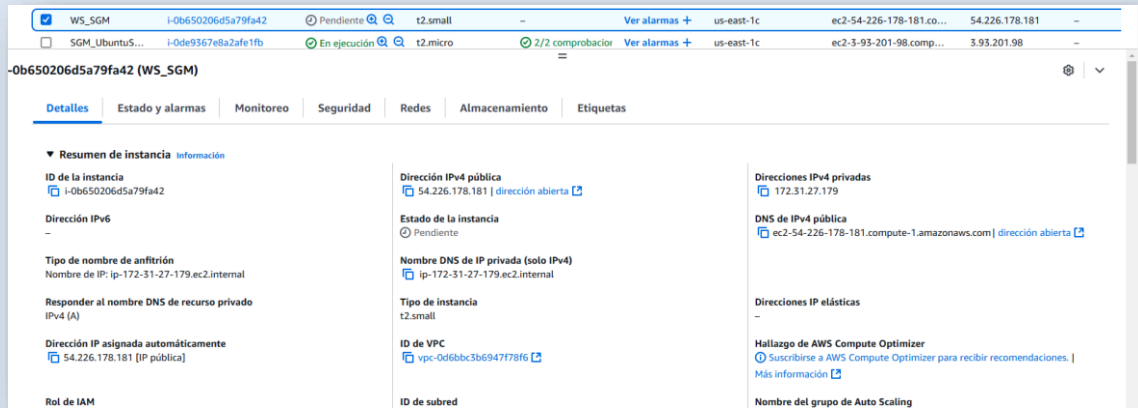
Primero, se mostrará cómo exportar los usuarios y grupos de un servidor local y luego replicarlos en un servidor EC2. Después, se explicará cómo crear un script que mantenga ambos servidores sincronizados, asegurando que siempre tengan los mismos usuarios y grupos. También se configurará un mantenimiento automático usando tareas programadas, para que todo el proceso se haga sin intervención manual.

Finalmente, se incluirá un menú interactivo para facilitar la ejecución de las tareas, y se añadirá una opción para gestionar múltiples servidores EC2 de forma más sencilla.

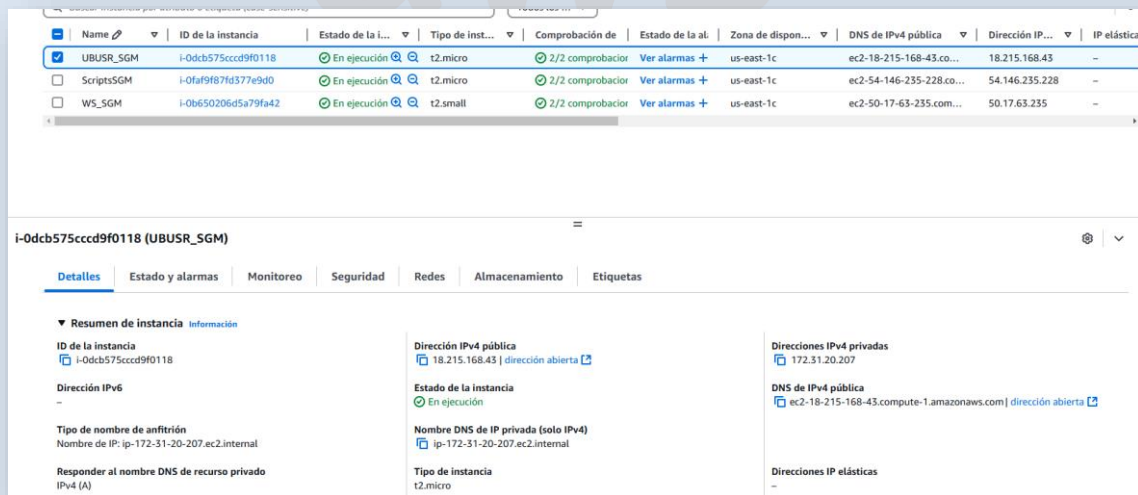
Para la práctica se ha usado un Windows server 2022 de aws, así como un Ubuntu Server. La configuración varía levemente en algunos pasos, pero nada muy complicado.

## 2. Preparación del entorno

El primer paso fue crear un Windows Server desde aws desde el que haremos toda la acción.



También un Ubuntu Server que es el que recibirá todos los usuarios y grupos.



### 3. Exportar usuarios y grupos locales de un servidor

Lo primero que he hecho es añadir algunos usuarios y grupos para darle algo mas de vidilla a la práctica.

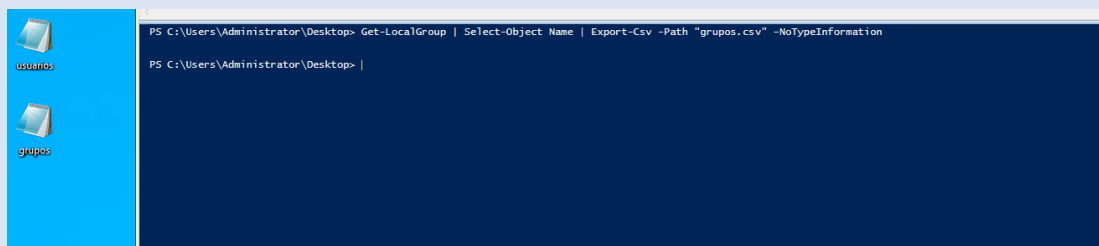
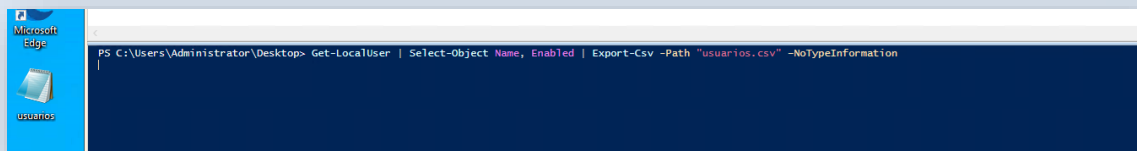
```
PS C:\Users\Administrator> get-localuser
```

Name	Enabled	Description
Administrator	True	Built-in account for administering the computer/domain
Batman	True	?
DefaultAccount	False	A user account managed by the system.
Fulanito	True	Si
Guest	False	Built-in account for guest access to the computer/domain
Harry	True	El que hará magia en el server
Sergio	True	Admin de la practica
WDAGUtilityAccount	False	A user account managed and used by the system for Windows Defender Application Guard scenarios.

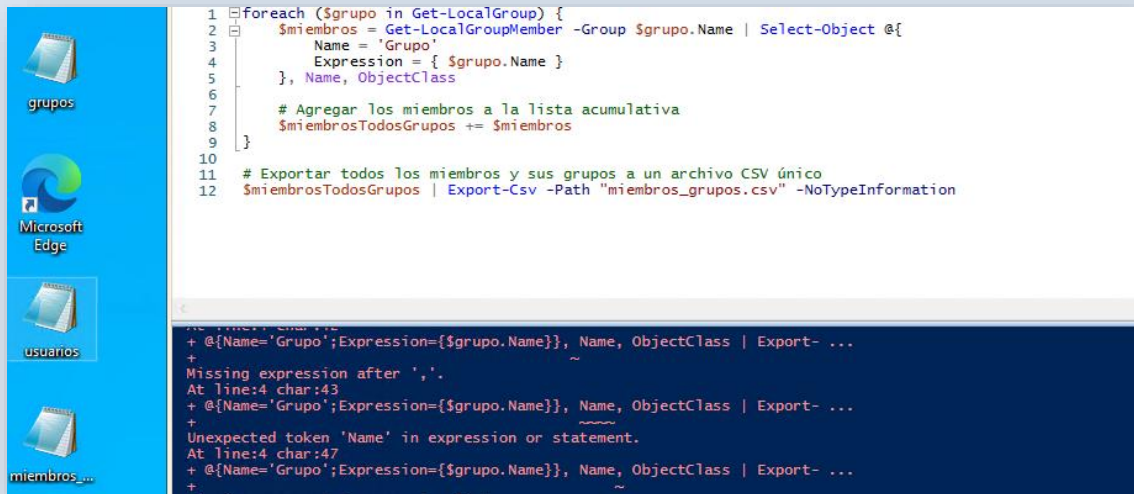
```
PS C:\Users\Administrator> get-localgroup
```

Name	Description
ASIR	Estudiantes
Hogwarts	Magico
Access Control Assistance Operators	Members of this group can remotely query authorization attributes and permissions for resources on this computer.
Administrators	Administrators have complete and unrestricted access to the computer/domain
Backup Operators	Backup Operators can override security restrictions for the sole purpose of backing up or restoring files
Certificate Service DCOM Access	Members of this group are allowed to connect to Certification Authorities in the enterprise
Cryptographic Operators	Members are authorized to perform cryptographic operations.
Device Owners	Members of this group can change system-wide settings.
Distributed COM Users	Members are allowed to launch, activate and use Distributed COM objects on this machine.
Event Log Readers	Members of this group can read event logs from local machine
Guests	Guests have the same access as members of the Users group by default, except for the Guest account which is further restricted
Hyper-V Administrators	Members of this group have complete and unrestricted access to all features of Hyper-V.
IIS_IUSRS	Built-in group used by Internet Information Services.
Network Configuration Operators	Members of this group can have some administrative privileges to manage configuration of networking features
Performance Log Users	Members of this group may schedule logging of performance counters, enable trace providers, and collect event traces both locally ...
Performance Monitor Users	Members of this group can access performance counter data locally and remotely
Power Users	Power Users are included for backwards compatibility and possess limited administrative powers
Print Operators	Members can administer printers installed on domain controllers
RDS Endpoint Servers	Servers in this group run virtual machines and host sessions where users RemoteApp programs and personal virtual desktops run. Thi...
RDS Management Servers	Servers in this group can perform routine administrative actions on servers running Remote Desktop Services. This group needs to b...
RDS Remote Access Servers	Servers in this group enable users of RemoteApp programs and personal virtual desktops access to these resources. In Internet-Faci...
Remote Desktop Users	Members in this group are granted the right to logon remotely
Remote Management Users	Members of this group can access WMI resources over management protocols (such as WS-Management via the Windows Remote Management ...
Replicator	Supports file replication in a domain
Storage Replica Administrators	Members of this group have complete and unrestricted access to all features of Storage Replica.
System Managed Accounts Group	Members of this group are managed by the system.
Users	Users are prevented from making accidental or intentional system-wide changes and can run most applications

Ahora con un par de comandos y un script (podría haber sido un script pero bueno) meteremos los usuarios.



Y con el siguiente script podremos añadir los usuarios a sus grupos.



```

1 foreach ($grupo in Get-LocalGroup) {
2     $miembros = Get-LocalGroupMember -Group $grupo.Name | Select-Object @{
3         Name = 'Grupo'
4         Expression = { $grupo.Name }
5     }, Name, ObjectClass
6
7     # Agregar los miembros a la lista acumulativa
8     $miembrosTodosGrupos += $miembros
9 }
10
11 # Exportar todos los miembros y sus grupos a un archivo CSV único
12 $miembrosTodosGrupos | Export-Csv -Path "miembros_grupos.csv" -NoTypeInformation

```

```

+ @{Name='Grupo';Expression={$grupo.Name}}, Name, ObjectClass | Export- ...
+
Missing expression after ','.
At line:4 char:43
+ @{Name='Grupo';Expression={$grupo.Name}}, Name, ObjectClass | Export- ...
+
Unexpected token 'Name' in expression or statement.
At line:4 char:47
+ @{Name='Grupo';Expression={$grupo.Name}}, Name, ObjectClass | Export- ...
+

```

#### 4. . Conectarse desde PowerShell a otro servidor EC2 por SSH

Ahora debemos instalar el módulo de ssh remoto. Como el comando proporcionado en la práctica no funcionaba (no sé si por temas de que aws es diferente), tuve que instalar el siguiente módulo para poder hacerlo:

```

PS C:\Users\Administrator\Desktop> Get-WindowsCapability -Online | Where-Object Name -like 'OpenSSH*'

Name : OpenSSH.Client~0.0.1.0
State : Installed
Name : OpenSSH.Server~0.0.1.0
State : NotPresent

PS C:\Users\Administrator\Desktop> Add-WindowsCapability -Online -Name OpenSSH.Server~0.0.1.0

Path      :
Online    : True
RestartNeeded : False

PS C:\Users\Administrator\Desktop> Start-Service sshd

PS C:\Users\Administrator\Desktop> Set-Service -Name sshd -StartupType Automatic

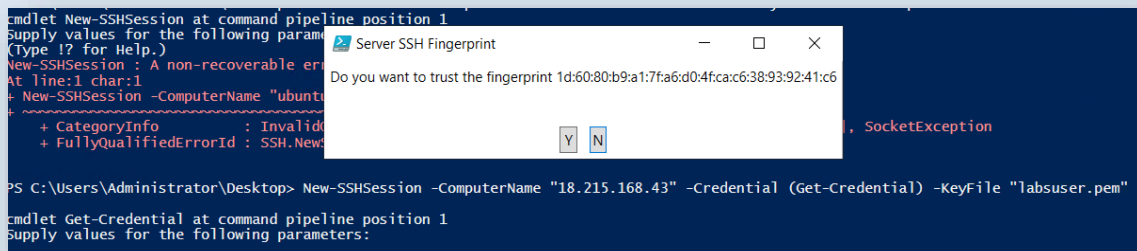
PS C:\Users\Administrator\Desktop>

```

```
PS C:\Users\Administrator\Desktop> Install-Module -Name Posh-SSH -Force -AllowClobber

PS C:\Users\Administrator\Desktop> Import-Module Posh-SSH
```

Ahora que al fin tenemos el módulo, podremos hacer el comando “New-SSHSession”, pero en este caso no necesitamos poner nombre de usuario o nos dará error.



The screenshot shows a PowerShell terminal window with the following text:

```
cmdlet New-SSHSession at command pipeline position 1
Supply values for the following parameters:
(Type ? for Help.)
New-SSHSession : A non-recoverable error has occurred.
At line:1 char:1
+ New-SSHSession -ComputerName "ubuntu"
+ ~~~~~
+ CategoryInfo          : InvalidOperation (FullQualifiedErrorId: SSH.NewSessionException), SocketException

PS C:\Users\Administrator\Desktop> New-SSHSession -ComputerName "18.215.168.43" -Credential (Get-Credential) -KeyFile "labsuser.pem"
```

Overlaid on the terminal is a "Server SSH Fingerprint" dialog box. It contains the text: "Do you want to trust the fingerprint 1d:60:80:b9:a1:7fa6:d0:4fcac6:38:93:92:41:c6". At the bottom are two buttons labeled "Y" and "N".

```
PS C:\Users\Administrator\Desktop> New-SSHSession -ComputerName "18.215.168.43" -Credential (Get-Credential) -KeyFile "labsuser.pem"

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:

SessionId Host Connected
-----
0 18.215.168.43 True
```

## 5. Replicar usuarios y grupos en EC2

Con el siguiente script, podremos crear todos los usuarios y grupos, así como al grupo al que pertenecen. Hubo que modificarlo un poco para que metiese los usuarios dentro de los grupos, pero acabó funcionando. Por ese motivo este script incluye el del punto siguiente, detectando los grupos que faltan.

```

1 #crear usuarios que faltan en el EC2
2 $localusuarios = Import-Csv -Path "usuarios.csv"
3 $ec2usuarios = Invoke-SSHCommand -SessionId 0 -Command "cut -d: -f1 /etc/passwd" |
4 Select-Object Output
5 $usuariosFaltantes = $localusuarios | where-Object { $ec2usuarios -notcontains $_.Name }
6 foreach ($usuario in $usuariosFaltantes) {
7     Invoke-SSHCommand -SessionId 0 -Command "sudo useradd $(($usuario.Name))"
8 }
9 #crear grupos que faltan en el EC2
10 $localgrupos = Import-Csv -Path "grupos.csv"
11 $ec2grupos = Invoke-SSHCommand -SessionId 0 -Command "cut -d: -f1 /etc/group" | Select-Object Output
12 $gruposFaltantes = $localgrupos | where-Object { $ec2grupos -notcontains $_.Name }
13 foreach ($grupo in $gruposFaltantes) {
14     Invoke-SSHCommand -SessionId 0 -Command "sudo groupadd $(($grupo.Name))"
15 }
16 #añadimos usuarios faltantes a grupos
17 foreach ($miembro in $miembros) {
18     Invoke-SSHCommand -SessionId 0 -Command "sudo usermod -a -G $(($miembro.Grupo))"
19     $(($miembro.Name))
20 }
21
Host : 18.215.168.43
Output : {}
ExitStatus : 127

Host : 18.215.168.43
Output : {}
ExitStatus : 127

Host : 18.215.168.43
Output : {}
ExitStatus : 127

Host : 18.215.168.43
Output : {}
ExitStatus : 127

Host : 18.215.168.43
Output : {}
ExitStatus : 127
  
```

Ahora, si nos vamos al ubuntu server, podremos comprobar qu todo funcionó.

```

ubuntu@UbuSGM: ~
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/usr/sbin/nologin
systemd-timesync:x:996:996:systemd Time Synchronization:/usr/sbin/nologin
dhcpd:x:100:65534:DHCP Client Daemon,,:/usr/lib/dhcpd:/bin/false
messagebus:x:101:101::/nonexistent:/usr/sbin/nologin
syslog:x:102:102::/nonexistent:/usr/sbin/nologin
systemd-resolve:x:991:991:systemd Resolver:/usr/sbin/nologin
uidd:x:103:103:/run/uidd:/usr/sbin/nologin
tss:x:104:104:TPM software stack,,:/var/lib/tpm:/bin/false
sshd:x:105:65534:/run/ssh:/usr/sbin/nologin
pollinate:x:106:1:/var/cache/pollinate:/bin/false
tcpdump:x:107:108:/nonexistent:/usr/sbin/nologin
landscape:x:108:109:/var/lib/landscape:/usr/sbin/nologin
fwupd-refresh:x:990:990:Firmware update daemon:/var/lib/fwupd:/usr/sbin/nologin
polkitd:x:989:989:User for polkitd:/usr/sbin/nologin
ec2-instance-connect:x:109:65534:/nonexistent:/usr/sbin/nologin
chrony:x:110:112:Chrony daemon,,:/var/lib/chrony:/usr/sbin/nologin
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
Administrator:x:1001:1001:/home/Administrator:/bin/sh
Batman:x:1002:1002:/home/Batman:/bin/sh
DefaultAccount:x:1003:1003:/home/DefaultAccount:/bin/sh
Fulanito:x:1004:1004:/home/Fulanito:/bin/sh
Guest:x:1005:1005:/home/Guest:/bin/sh
Harry:x:1006:1006:/home/Harry:/bin/sh
Sergio:x:1007:1007:/home/Sergio:/bin/sh
WDAGUtilityAccount:x:1008:1008:/home/WDAGUtilityAccount:/bin/sh
ubuntu@UbuSGM: ~$
  
```



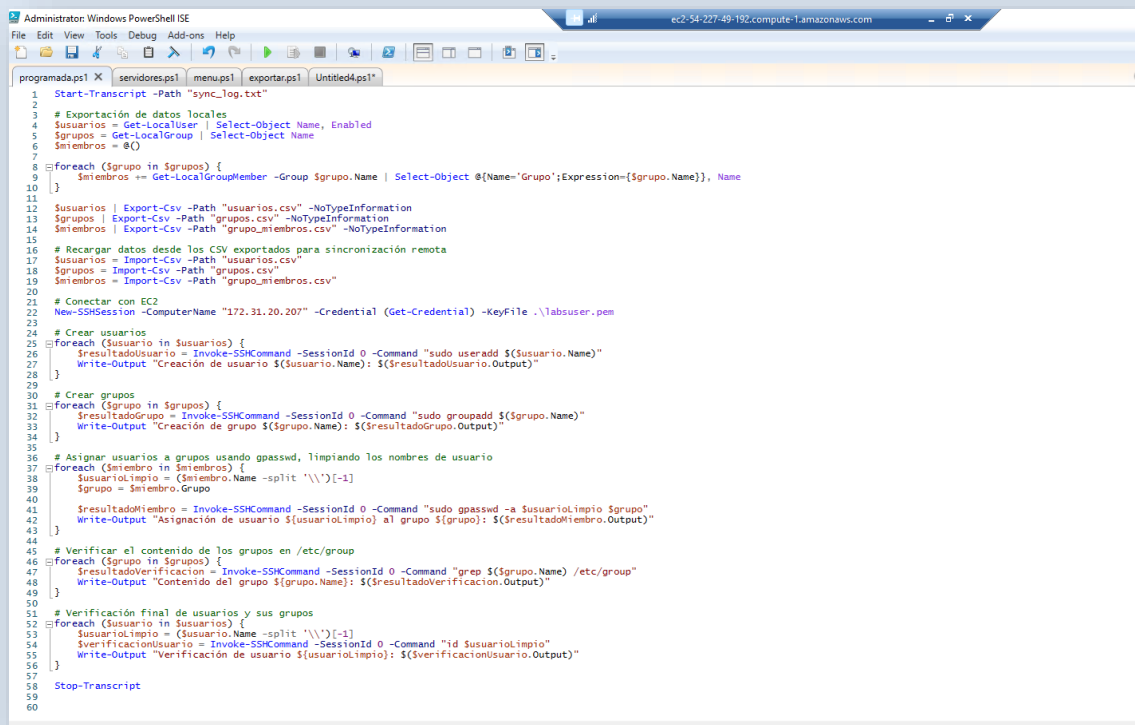
```
ubuntu@UbusrSGM: ~  
systemd-resolve:x:991:  
uidd:x:103:  
tss:x:104:  
lxd:x:105:ubuntu  
_ssh:x:106:  
rdma:x:107:  
tcpdump:x:108:  
landscape:x:109:  
fwupd-refresh:x:990:  
polkitd:x:989:  
admin:x:110:  
netdev:x:111:  
_chrony:x:112:  
ubuntu:x:1000:  
Administrator:x:1001:  
Batman:x:1002:  
DefaultAccount:x:1003:  
Fulanito:x:1004:  
Guest:x:1005:  
Harry:x:1006:  
Sergio:x:1007:  
WDAGUtilityAccount:x:1008:  
ASIR:x:1009:  
Hogwarts:x:1010:  
Administrators:x:1011:  
Guests:x:1012:  
IIS_IUSRS:x:1013:  
Replicator:x:1014:  
Users:x:1015:  
ubuntu@UbusrSGM:~$
```

```
ubuntu@UbusrSGM:~$ sudo getent group Hogwarts  
Hogwarts:x:1010:Batman,Harry  
ubuntu@UbusrSGM:~$ sudo getent group ASIR  
ASIR:x:1009:Sergio  
ubuntu@UbusrSGM:~$ sudo getent group Users  
Users:x:1015:Batman,Fulanito,Harry,Sergio  
ubuntu@UbusrSGM:~$
```

## 6. Mantenimiento automático mediante tarea programada.

Para este apartado de la práctica se configurará una tarea programada en el servidor local para exportar usuarios y sincronizarlos con EC2 todos los días a las 10:00 am. Debe dejar registro en un log. Tenemos que crear un script donde exportamos y sincronizamos.

El script que creará el log será el siguiente:



```

1 Start-Transcript -Path "sync_log.txt"
2
3 # Exportación de datos locales
4 $usuarios = Get-LocalUser | Select-Object Name, Enabled
5 $grupos = Get-LocalGroup | Select-Object Name
6 $miembros = @()
7
8 foreach ($grupo in $grupos) {
9     $miembros += Get-LocalGroupMember -Group $grupo.Name | Select-Object @{Name="Grupo"; Expression={$grupo.Name}}, Name
10 }
11
12 $usuarios | Export-Csv -Path "usuarios.csv" -NoTypeInformation
13 $grupos | Export-Csv -Path "grupos.csv" -NoTypeInformation
14 $miembros | Export-Csv -Path "grupo_miembros.csv" -NoTypeInformation
15
16 # Recargar datos desde los CSV exportados para sincronización remota
17 $usuarios = Import-Csv -Path "usuarios.csv"
18 $grupos = Import-Csv -Path "grupos.csv"
19 $miembros = Import-Csv -Path "grupo_miembros.csv"
20
21 # Conectar con EC2
22 New-SSHSession -ComputerName "172.31.20.207" -Credential (Get-Credential) -KeyFile .\labsuser.pem
23
24 # Crear usuarios
25 foreach ($usuario in $usuarios) {
26     $resultadoUsuario = Invoke-SSHCommand -SessionId 0 -Command "sudo useradd $($usuario.Name)"
27     Write-Output "Creación de usuario $($usuario.Name): $($resultadoUsuario.Output)"
28 }
29
30 # Crear grupos
31 foreach ($grupo in $grupos) {
32     $resultadoGrupo = Invoke-SSHCommand -SessionId 0 -Command "sudo groupadd $($grupo.Name)"
33     Write-Output "Creación de grupo $($grupo.Name): $($resultadoGrupo.Output)"
34 }
35
36 # Asignar usuarios a grupos usando gpasswd, limpiando los nombres de usuario
37 foreach ($miembro in $miembros) {
38     $usuarioLimpio = ($miembro.Name -split "\\")[1]
39     $grupo = $miembro.Grupo
40     $resultadoMiembro = Invoke-SSHCommand -SessionId 0 -Command "sudo gpasswd -a $usuarioLimpio $grupo"
41     Write-Output "Asignación de usuario $($usuarioLimpio) al grupo $($grupo): $($resultadoMiembro.Output)"
42 }
43
44 # Verificar el contenido de los grupos en /etc/group
45 foreach ($grupo in $grupos) {
46     $resultadoVerificacion = Invoke-SSHCommand -SessionId 0 -Command "grep $($grupo.Name) /etc/group"
47     Write-Output "Contenido del grupo $($grupo.Name): $($resultadoVerificacion.Output)"
48 }
49
50 # Verificación final de usuarios y sus grupos
51 foreach ($usuario in $usuarios) {
52     $usuarioLimpio = ($usuario.Name -split "\\")[1]
53     $verificacionUsuario = Invoke-SSHCommand -SessionId 0 -Command "id $usuarioLimpio"
54     Write-Output "Verificación de usuario $($usuarioLimpio): $($verificacionUsuario.Output)"
55 }
56
57 Stop-Transcript
58
59
60
  
```

Una vez que tenemos el script y verificamos que se ejecuta sin errores, nos debería dar un log como el siguiente:

The screenshot shows a Windows PowerShell console window with a script being executed. The script is titled "programadaps1 X" and is located in the "servidores.ps1" directory. The script is a PowerShell script that creates a local user and groups on a Windows system.

The script content is as follows:

```

1 Start-Transcript -Path "sync_log.txt"
2
3 # Exportación de datos locales
4 $usuarios = Get-LocalUser
5 $grupos = Get-LocalGroup
6 $entornos = @(Get-Childitem -Path "HKLM:\Software\Microsoft\Windows\CurrentVersion\GroupPolicy\{00000000-0000-0000-0000-000000000000}\{00000000-0000-0000-0000-000000000000}" -ErrorAction SilentlyContinue)
7
8 # Crear usuario
9 $usuario = "Administrator"
10 $password = "P@ssw0rd123456789"
11 $usuario | ForEach-Object {
12     $user = Get-LocalUser -Name $_
13     if ($?) {
14         $user.Password = $password
15     }
16 }
17
18 # Crear grupo
19 $grupo = "Administrators"
20 $grupo | ForEach-Object {
21     $group = Get-LocalGroup -Name $_
22     if ($?) {
23         $group.FullControl = "FullControl"
24     }
25 }
26
27 # Verificar
28 $usuario | ForEach-Object {
29     $user = Get-LocalUser -Name $_
30     if ($?) {
31         $user.Password = $password
32     }
33 }
34
35 # Verificar
36 $grupo | ForEach-Object {
37     $group = Get-LocalGroup -Name $_
38     if ($?) {
39         $group.FullControl = "FullControl"
40     }
41 }
42
43 Stop-Transcript

```

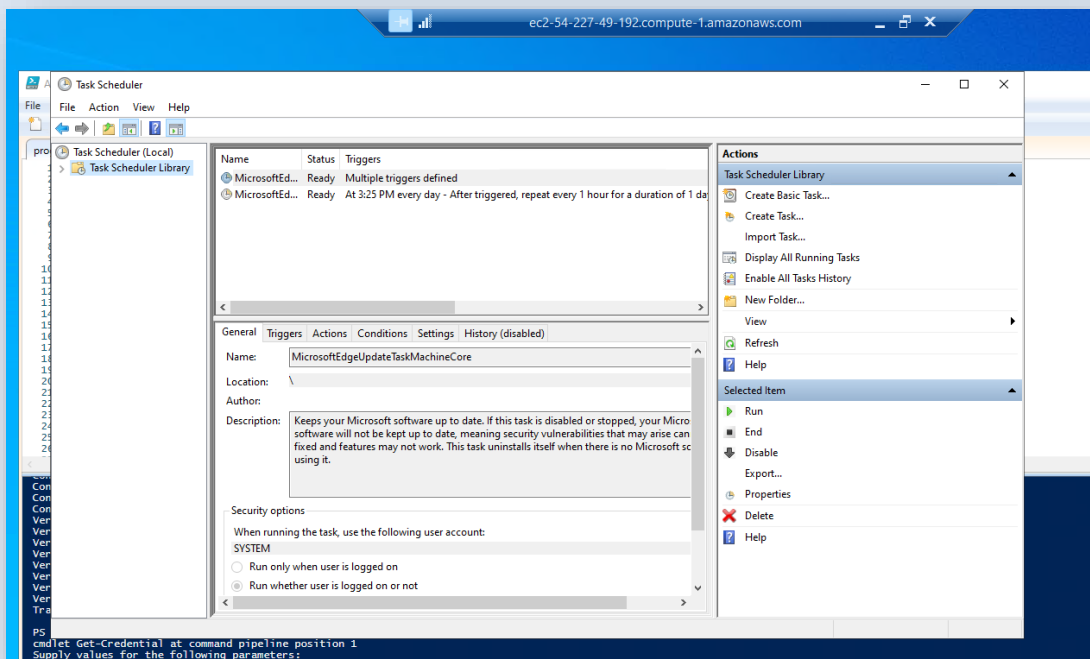
The output of the script is as follows:

```

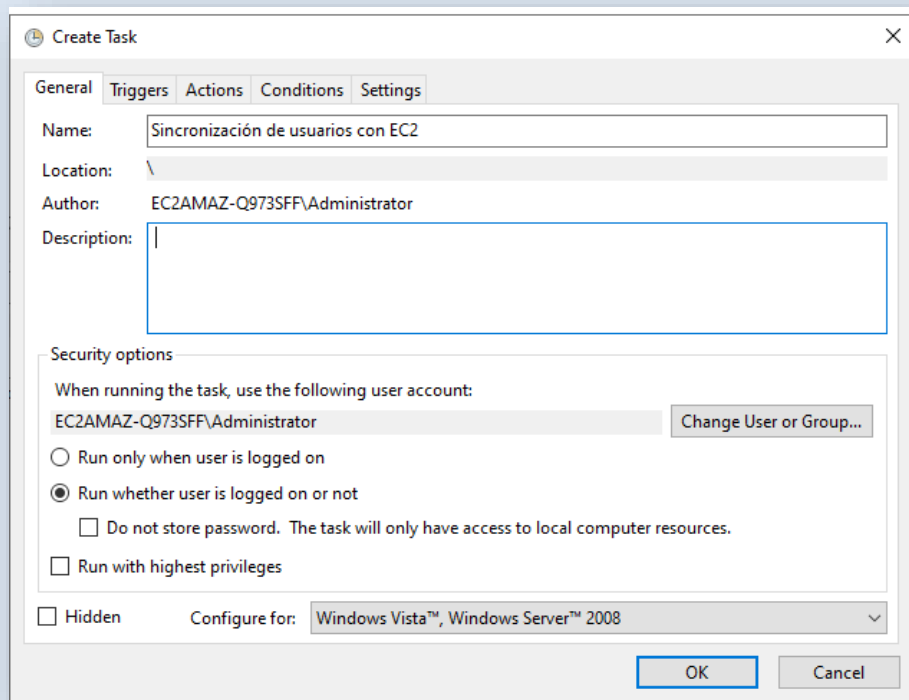
*****
Windows PowerShell transcript start
Start time: 20241205153500
Username: EC2AMAZ-Q9735FF\Administrator
RunAs User: EC2AMAZ-Q9735FF\Administrator
Configuration Name:
Machine: EC2AMAZ-Q9735FF (Microsoft Windows NT 10.0.20348.0)
Host Application: C:\Windows\system32\WindowsPowerShell\v1.0\PowerShell_TSE.exe
Process ID: 5104
PSVersion: 5.1.20348.2849
PSEdition: Desktop
PSCompatibleVersions: 1.0, 2.0, 3.0, 4.0, 5.0, 5.1.20348.2849
BuildVersion: 10.0.20348.2849
CLRVersion: 4.0.30319.42000
WSManStackVersion: 3.0
PSRemotingProtocolVersion: 2.3
SerializationVersion: 1.1.0.1
*****
Creación de usuario Administrator:
Creación de usuario Batman:
Creación de usuario DefaultAccount:
Creación de usuario Fulanito:
Creación de usuario Guest:
Creación de usuario Harry:
Creación de usuario Sergio:
Creación de usuario WDMUtilityAccount:
Creación de grupo ASIR:
Creación de grupo Hogwarts:
Creación de grupo Access Control Assistance Operators:
Creación de grupo Administrators:
Creación de grupo Backup Operators:
Creación de grupo Certificate Service DCOM Access:
Creación de grupo Cryptographic Operators:
Creación de grupo Device Owners:
Creación de grupo Distributed COM Users:
Creación de grupo Event Log Readers:
Creación de grupo Guests:

```

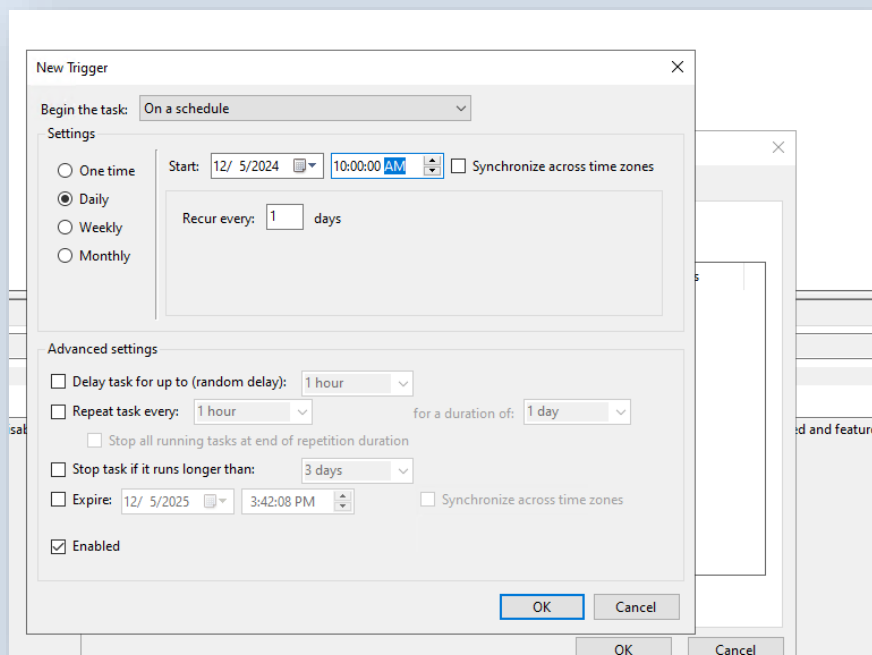
Ahora que sabemos que nos funciona, el siguiente paso será ir al programador de tareas.



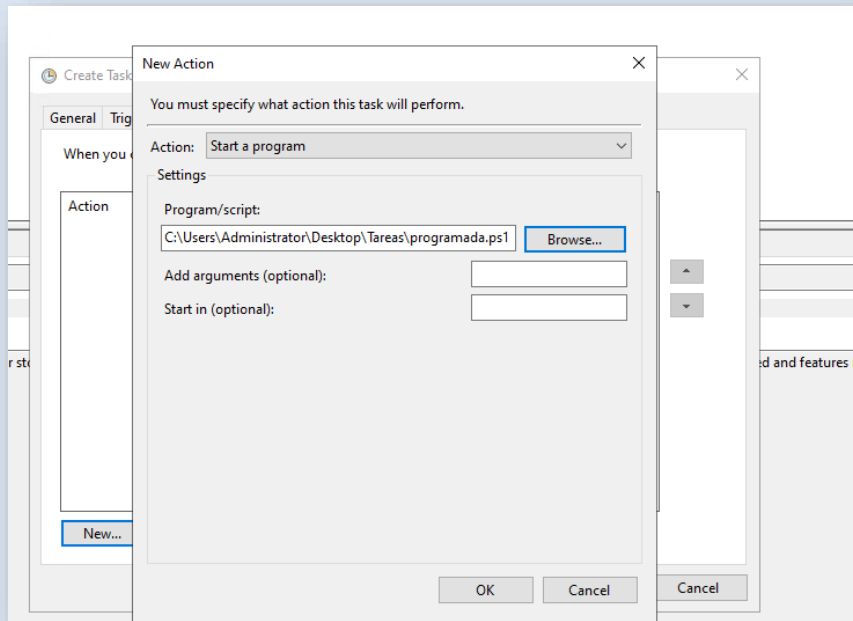
Crearemos la tarea con su nombre y la ejecutaremos estemos o no logueados.



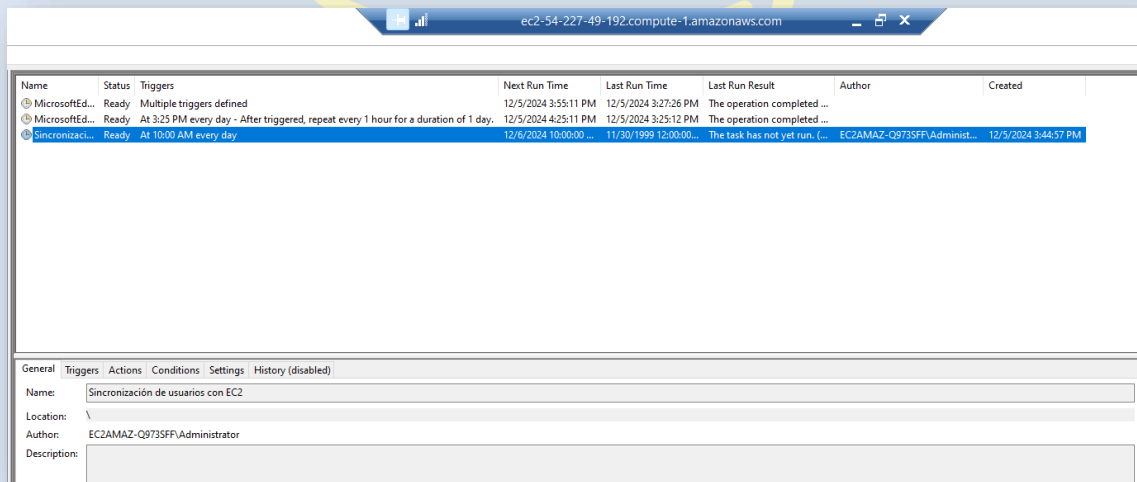
Haremos que se ejecute todos los días a las 10:00.



Le asignamos el script que hemos creado:

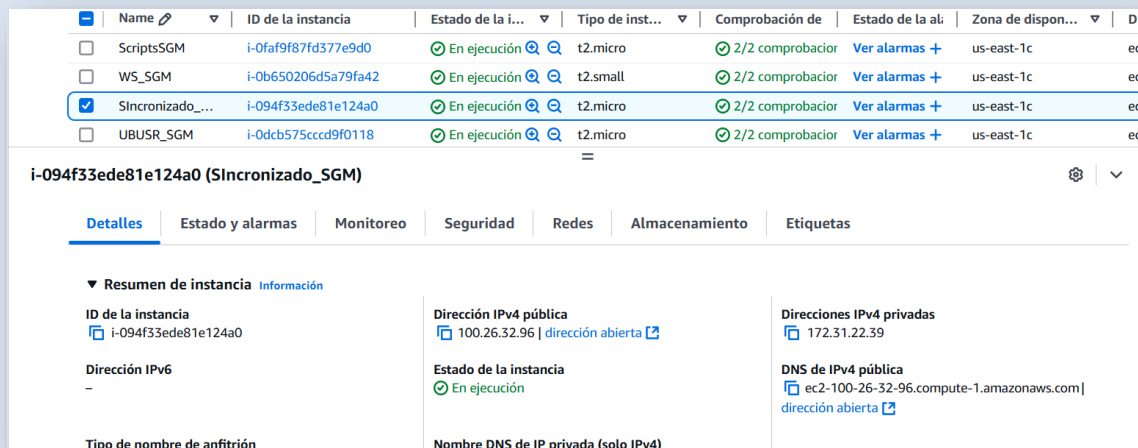


Y listo, ya tendríamos la tarea para que se ejecute todos los días a las 10.



## 7. Ampliación soporte para múltiples servidores EC2

En esta parte de la práctica automatizaremos para que podamos añadir los usuarios a todas las instancias EC2 que queramos. En mi caso he usado la instancia de antes y otra más que he creado como prueba:



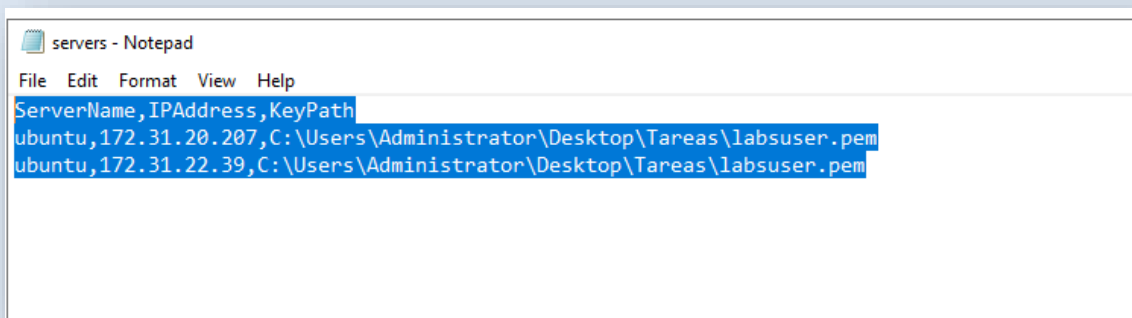
The screenshot shows the AWS Management Console. At the top, there is a table of EC2 instances:

Name	ID de la instancia	Estado de la i...	Tipo de inst...	Comprobación de	Estado de la ali	Zona de dispon...	DI
ScriptsSGM	i-0faf9f87fd377e9d0	En ejecución	t2.micro	2/2 comprobaci...	Ver alarmas +	us-east-1c	ec
WS_SGM	i-0b650206d5a79fa42	En ejecución	t2.small	2/2 comprobaci...	Ver alarmas +	us-east-1c	ec
Sincronizado_...	i-094f33ede81e124a0	En ejecución	t2.micro	2/2 comprobaci...	Ver alarmas +	us-east-1c	ec
UBUSR_SGM	i-0dcb575cccd9f0118	En ejecución	t2.micro	2/2 comprobaci...	Ver alarmas +	us-east-1c	ec

Below the table, the details for the instance **i-094f33ede81e124a0 (Sincronizado\_SGM)** are shown. The 'Resumen de instancia' section includes:

- ID de la instancia:** i-094f33ede81e124a0
- Dirección IPv4 pública:** 100.26.32.96 | dirección abierta
- Direcciones IPv4 privadas:** 172.31.22.39
- Estado de la instancia:** En ejecución
- DNS de IPv4 pública:** ec2-100-26-32-96.compute-1.amazonaws.com | dirección abierta

Lo siguiente será meter un archivo .csv los servidores. En mi caso usé la IP privada por ser un windows de aws.

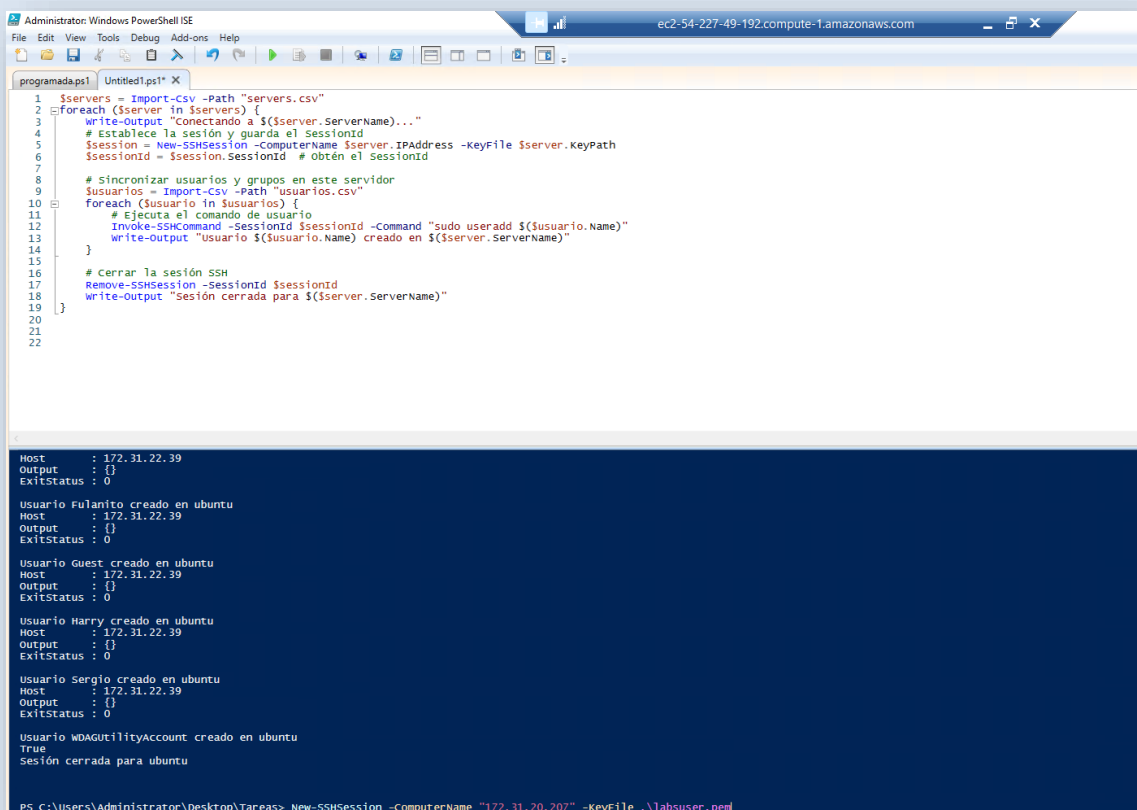


The screenshot shows a Notepad window titled 'servers - Notepad'. The text inside is as follows:

```
File Edit Format View Help
ServerName,IPAddress,KeyPath
ubuntu,172.31.20.207,C:\Users\Administrator\Desktop\Tareas\labsuser.pem
ubuntu,172.31.22.39,C:\Users\Administrator\Desktop\Tareas\labsuser.pem
```

Ahora falta meter el script para automatizar esta tarea. No conseguí que no me pidiese el login a la hora de ejecutarlo, supongo que porque cada vez que hago el 'New-SSHSession' me pide la contraseña sí o sí. Tampoco modificando en ubuntu para que no me pida login por ssh.

Pero ejecutando el siguiente script, aún contraseña, se ejecutó correctamente.



```
1 $servers = Import-Csv -Path "servers.csv"
2 foreach ($server in $servers) {
3     write-output "Conectando a $($server.ServerName)..."
4     # Establece la sesión y guarda el sessionId
5     $session = New-SSHSession -ComputerName $server.IPAddress -KeyFile $server.KeyPath
6     $sessionId = $session.SessionId # Obtén el SessionId
7
8     # Sincronizar usuarios y grupos en este servidor
9     $usuarios = Import-Csv -Path "usuarios.csv"
10    foreach ($usuario in $usuarios) {
11        # Ejecuta el comando de usuario
12        Invoke-SSHCommand -SessionId $sessionId -Command "sudo useradd $($usuario.Name)"
13        write-output "Usuario $($usuario.Name) creado en $($server.ServerName)"
14    }
15
16    # Cerrar la sesión SSH
17    Remove-SSHSession -SessionId $sessionId
18    write-output "Sesión cerrada para $($server.ServerName)"
19 }
20
21
22
```

```
Host      : 172.31.22.39
Output    : {}
ExitStatus: 0

Usuario Fulanito creado en ubuntu
Host      : 172.31.22.39
Output    : {}
ExitStatus: 0

Usuario Guest creado en ubuntu
Host      : 172.31.22.39
Output    : {}
ExitStatus: 0

Usuario Harry creado en ubuntu
Host      : 172.31.22.39
Output    : {}
ExitStatus: 0

Usuario Sergio creado en ubuntu
Host      : 172.31.22.39
Output    : {}
ExitStatus: 0

Usuario wDAGUtilityAccount creado en ubuntu
True
Sesión cerrada para ubuntu

PS C:\Users\Administrator\Desktop\Tareas> New-SSHSession -ComputerName "172.31.20.207" -KeyFile .\labsuser.pem
```

Si nos vamos a la instancia que acabamos de crear, podemos ver que se crearon los usuarios también:

```
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
_apt:x:42:65534::/nonexistent:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:998:998:systemd Network Management:/:/usr/sbin/nologin
systemd-timesync:x:996:996:systemd Time Synchronization:/:/usr/sbin/nologin
dhcpcd:x:100:65534:DHCP Client Daemon,,,:/usr/lib/dhcpcd:/bin/false
messagebus:x:101:101::/nonexistent:/usr/sbin/nologin
syslog:x:102:102::/nonexistent:/usr/sbin/nologin
systemd-resolve:x:991:991:systemd Resolver:/:/usr/sbin/nologin
uidd:x:103:103:/run/uidd:/usr/sbin/nologin
tss:x:104:104:TPM software stack,,,:/var/lib/tpm:/bin/false
sshd:x:105:65534:/run/sshd:/usr/sbin/nologin
pollinate:x:106:1:/var/cache/pollinate:/bin/false
tcpdump:x:107:108:/nonexistent:/usr/sbin/nologin
landscape:x:108:109:/var/lib/landscape:/usr/sbin/nologin
fwupd-refresh:x:990:990:Firmware update daemon:/var/lib/fwupd:/usr/sbin/nologin
polkitd:x:989:989:User for polkitd:/:/usr/sbin/nologin
ec2-instance-connect:x:109:65534:/nonexistent:/usr/sbin/nologin
chrony:x:110:112:Chrony daemon,,,:/var/lib/chrony:/usr/sbin/nologin
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
Administrator:x:1001:1001:/home/Administrator:/bin/sh
Batman:x:1002:1002:/home/Batman:/bin/sh
DefaultAccount:x:1003:1003:/home/DefaultAccount:/bin/sh
Fulanito:x:1004:1004:/home/Fulanito:/bin/sh
Guest:x:1005:1005:/home/Guest:/bin/sh
Harry:x:1006:1006:/home/Harry:/bin/sh
Sergio:x:1007:1007:/home/Sergio:/bin/sh
WDAGUtilityAccount:x:1008:1008:/home/WDAGUtilityAccount:/bin/sh
ubuntu@ip-172-31-22-39:~$
```

**i-094f33ede81e124a0 (Sincronizado\_SGM)**

PublicIPs: 100.26.32.96 PrivateIPs: 172.31.22.39



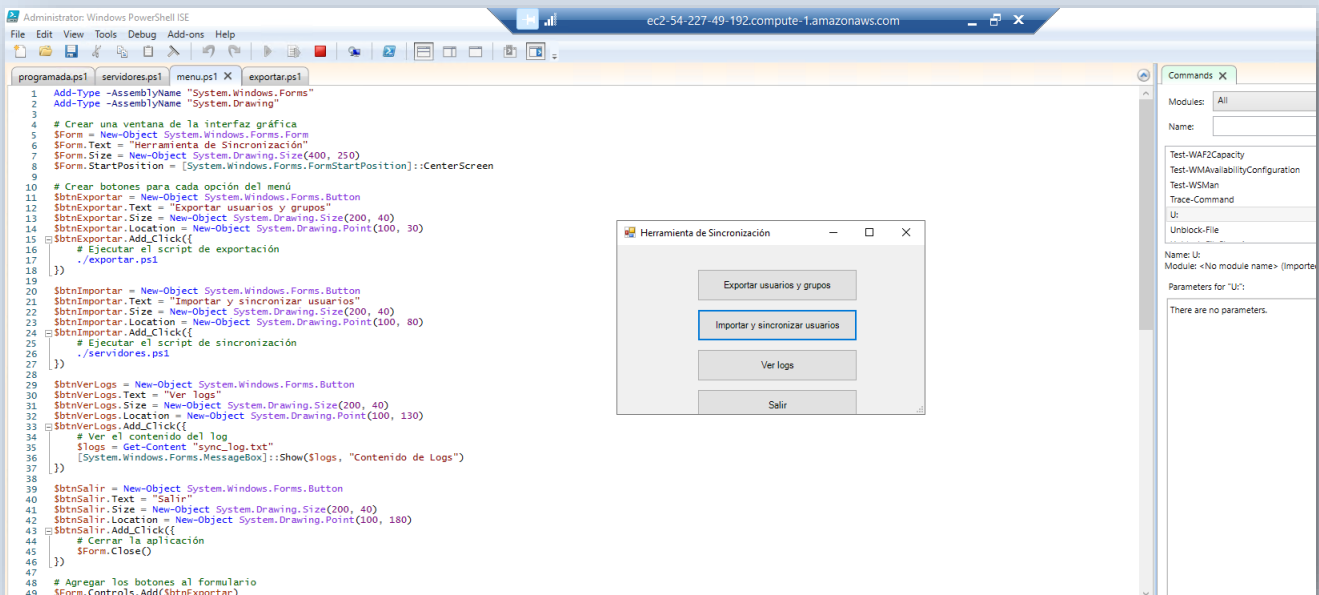
## 8. Crear un menú para facilitar la interacción

Para este paso se crearon un par de scripts para tener un menú que recoja lo anterior y ejecutarlo de forma eficiente. El primer script era este:

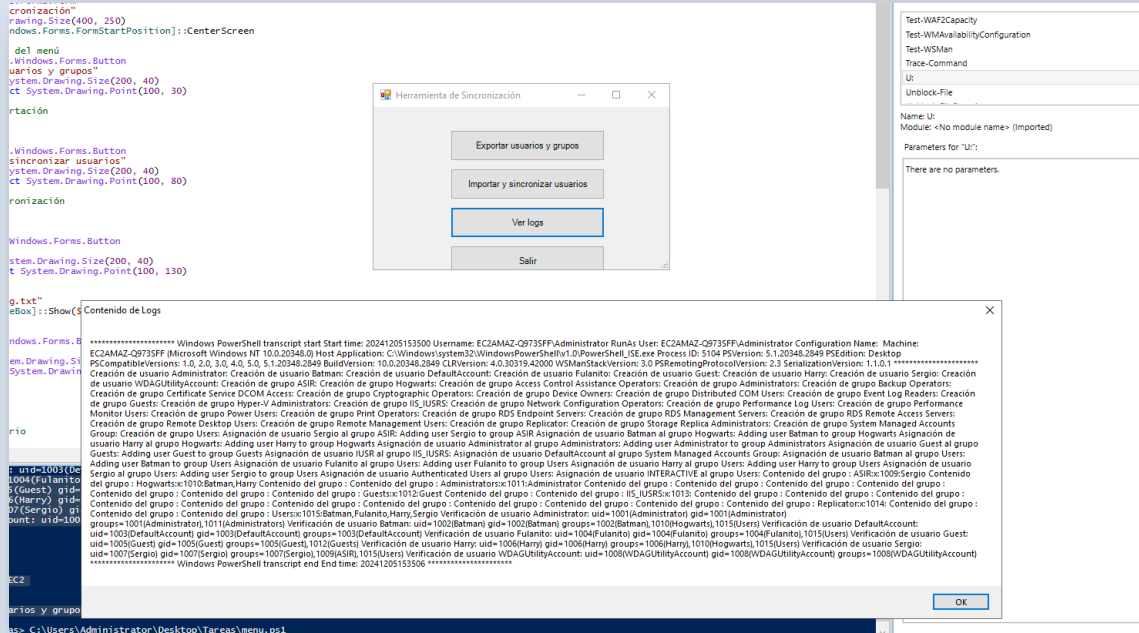
```

1 Function Mostrar-Menu {
2     Write-Host "1. Exportar usuarios y grupos"
3     Write-Host "2. Importar y sincronizar usuarios en EC2"
4     Write-Host "3. Ver logs"
5     Write-Host "4. Salir"
6 }
7 while ($true) {
8     Mostrar-Menu
9     $opcion = Read-Host "selecciona una opción"
10    switch ($opcion) {
11        1 { ./exportar.ps1 }
12        2 { ./servidores.ps1 }
13        3 { Get-Content "sync_log.txt" | Out-Host }
14        4 { break }
15        default { Write-Host "opción no válida." }
16    }
17 }
  
```

Pero como era muy soso, se le añadió interfaz gráfica para que se vea así:



Si por ejemplo ejecutamos para ver los logs, nos abre una ventana con el contenido de los logs.



Y si queremos sincronizar usuarios y grupos, seleccionamos la segunda opción, aunque tendremos que meter usuario y contraseña por cada instancia.

