ASO-IES CELIA VIÑAS
2º ASIR

IMPLEMENTACIÓN SERVIDOR **MINECRAFT**

Propuesta realizada por Fabián Alcaide López

Índice

1. Planteamiento del proyecto

Minecraft es un videojuego de construcción y sandbox desarrollado por Mojang Studios y comprada por Microsoft en la actualidad. Se caracteriza por tener un código sólido con elementos procedurales compuestos por bloques cúbicos o vóxeles donde los jugadores gestionar los recursos naturales del mundo y desarrollar diferentes técnicas para explorar los diferentes biomas. Un juego que ofrece un entorno de creación y supervivencia multijugador alojada en diferentes servidores, ahí será donde nosotros planteemos nuestro proyecto.

Mojang ofrece diferentes herramientas para poder descargar las dependencias para poder desplegar un servidor junto a diferentes tutoriales para securizar y personalizar los diferentes parámetros del servidor, por lo que nuestro proyecto se va a basar en crear un script de bash para automatizar el despliegue, su administración y realizar copias de seguridad programadas.

Como requisitos previos debemos contar con:

- Un sistema operativo basado en linux.
- Acceso a permisos de administrador (sudo).
- Conexión a internet para descargar dependencias.
- IP pública en caso de querer un servidor público dedicado.

En nuestro caso vamos a contar con el desarrollo en una instancia de cloud computing en AWS EC2, una solución versátil para este tipo de casos, que nos permite el lanzamiento de diferentes máquinas, gestión de puertos y uso de ips públicas elásticas.

Para el script de bash, vamos a usar diferentes estructuras y programación tanto estructurada como modular que hemos visto en clase, comandos como sed, crontab, etc.. para poner en un caso real este lenguaje.

2. Desarrollo del proyecto

2.1. Montaje del entorno

Primero vamos a crear el entorno en que vamos a trabajar, vamos a trabajar en un ubuntu server 24.04 creado en AWS EC2, para ello entramos en nuestro laboratorio de AWS y lanzamos la instancia con las siguientes características (figura 1).

Figura 1

Una vez la tenemos, solicitamos la ip elástica, y la apuntamos, en este caso 52.70.71.115. Además debemos crear un grupo de seguridad para nuestra máquina, en él

vamos a especificar el puerto 22 y el puerto 25565, este último será obligatorio, ya que es necesario para el socket de conexión de los servidores de Minecraft (figura 2).

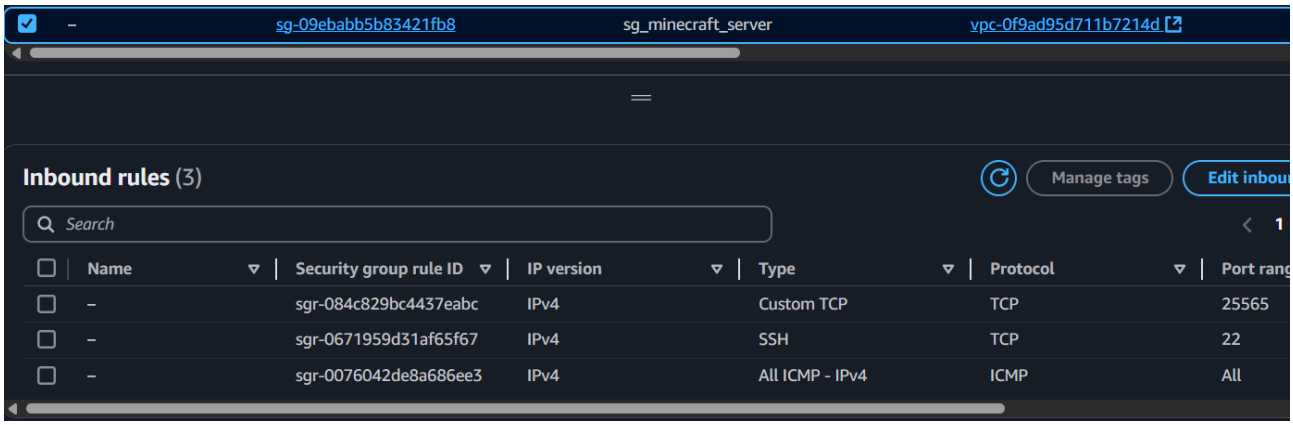

Figura 2

Y después, aunque de manera opcional, podemos crear un repositorio en github donde alojar el script, en nuestro caso adjuntamos el enlace:

falclop/servidor_minecraft_AWS: Despliegue de un servidor de minecraft en AWS

2.2. Montaje del script

Una vez tenemos el entorno activo vamos a crear el script. Siguiendo las normas de estructuración de scripts, vamos a comenzar nuestro archivo con las líneas #!/bin/bash y source .env, esta última hace referencia a que las **variables** que se trabajan en el script se tomarán de un archivo externo, en este caso un .env, que no se sube al repositorio pero que adjuntamos un ejemplo .env.example (figura 3).

Figura 3

Aquí especificamos el directorio donde crear nuestro servidor, el directorio de copias de seguridad, la ruta del servicio y la url de las dependencias del servidor de Minecraft.

Una vez tenemos estos archivos preparados vamos a ir analizando nuestro script.

2.2.1. Comandos a usar

tar – para comprimir los diferentes archivos que queramos.

echo – para lanzar por pantalla lo que elijamos.

> /dev/null – mandar la salida del comando a nulo

2>&1 - mandar la salida del error al mismo sitio que la salida de éxito, es decir a nulo.

date – para lanzar fechas.

grep – para crear filtros.

crontab – para crear tareas programadas.

read – para tomar inputs del usuario por teclado.

apt – el gestor de paquetes de linux (debian y ubuntu).

mkdir – para creación de directorios.

sleep – para parar la ejecución de un comando o script x segundos.

sed – para sustitución, adición o reemplazo de patrones.

chmod – para gestionar permisos.

cat – para mostrar el contenido de algo y trabajar con él.

EOF – “end of line”, con este termino definimos cual será el final del texto a usar.

systemctl – comando para gestionar procesos del sistema en segundo plano.

rm – para borrar archivos.

clear – limpia la pantalla.

2.2.2. Montaje de funciones

Como en toda programación modular, el uso de funciones para acotar posible código susceptible de repetición es esencial, por lo que en el nuestro la ejecución de diferentes funciones permite aligerar nuestro código.

Vamos a crear tres funciones:

hacer_backup():

```
# Función para crear una copia de seguridad del servidor de Minecraft
hacer_backup() {
    if [[ ! -d "$BACKUP_DIR" ]]; then
        mkdir -p ${BACKUP_DIR}
    fi
    tar -czvf ${BACKUP_DIR}/backup-$(date +%F_%H:%M).tar.gz ${MINECRAFT_DIR} > /dev/null 2>&1
    echo -e "\nBackup completado en $BACKUP_DIR"
}
```

Figura 4

Primero verificamos que el directorio no existe, si es así creamos el directorio, luego comprimimos el contenido del directorio.

programa_backup():

```
# Función para programar con crontab un copia de seguridad en caliente del servidor a las 3:00 AM
programar_backup() {
    BACKUP_CRON="tar -czvf ${BACKUP_DIR}/backup-$(date +%F_%H:%M).tar.gz ${MINECRAFT_DIR}"
    if crontab -l 2>/dev/null | grep -q "${BACKUP_CRON}"; then
        echo "Ya hay una copia de seguridad programada en cron."
    else
        (crontab -l 2>/dev/null; echo "0 3 * * * ${BACKUP_CRON}") | crontab -
        echo "¡Backup programado a las 3:00 AM todos los días!"
    fi
}
```

Figura 5

Definimos en un variable el comando de tar y hacemos una comprobación, si la tarea programada con ese nombre existe, nos avisa, si no, la crea con crontab.

Menu() y ayuda():

```
menu() {
    echo -e "1. Instalar servidor\n2. Desinstalar servidor\n3. Crear copia de seguridad\n0. Salir del programa\nPulse h para ayuda"
}

ayuda() {
    echo ""
    echo "Menú de ayuda de instalador de servidor de Minecraft."
    echo "Para elegir la opción que quiera pulse 1 2 o 3. 0 para salir del programa."
    echo "Páselo bien!!!"
    echo ""
}
```

Figura 6

Unas sencillas funciones para que el usuario vea el menú y tenga una ayuda de como funciona el programa.

2.2.3. Comprobación de sudo

Para esta comprobación vamos a hacer uso del EUID, el identificador de usuario efectivo, este podemos verlo lanzando echo $EUID, una variable de entorno que controla el identificador del usuario que esta actuando en este momento. Si nos fijamos en una prueba práctica al lanzar el comando siendo un usuario nos dará como valor el código del usuario, mientras que si lo lanzamos siendo root siempre nos devolverá 0 (figura 7-8).

```
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$ sudo su
root@ip-172-31-26-237:/home/ubuntu/servidor_minecraft_AWS/scripts# echo
 $EUID
0
root@ip-172-31-26-237:/home/ubuntu/servidor_minecraft_AWS/scripts# exit
exit
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$ echo $EUID
1000
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$
```

Figura 7

```
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS$ cat /etc/passwd | grep -E "root|ubuntu"
root:x:0:0:root:/root:/bin/bash
ubuntu:x:1000:1000:Ubuntu:/home/ubuntu:/bin/bash
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS$
```

Figura 8

Y para nuestro script haremos un sencilla comprobación sobre el EUID condicionando la respuesta a que si el EUID del ejecutante es diferente de 0 salta con error 1 (figura 9-10), y llevamos el comando a la salida de errores.

```
if [[ $EUID -ne 0 ]]; then
    echo "Este script debe ejecutarse con sudo o como root." >&2
    exit 1
fi
```

Figura 9

```
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$ ./start.sh
Este script debe ejecutarse con sudo o como root.
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$
```

Figura 10

2.2.4. Cuerpo del script

El funcionamiento del script se basa en un menú interactivo que se encuentra dentro de un bucle **while**, en este se comparará la opción elegida, y mientras no sea ni **h**, ni **0**, se repetirá selección (figura 11).

```
menu
echo -e "\nSeleccione una opción: "
read OP
while [[ $OP != [h] || $OP != [0] ]]; do
```

Figura 11

Dentro del while, tendremos el menú principal dentro de un **case**, una estructura de control basada en en opción múltiple. Aquí podremos elegir entre **1:** instalar el servidor, **2:** desinstalarlo, **3:** realizar la copia de seguridad, **h:** enseñar la ayuda o **0:** salir del programa.

Opción 1: instalar el servidor

Como componente principal, será el más largo y contendrá todo el sistema de instalación (figura 12).

```
case $OP in
    1)
        clear
        echo -e "\nInstalando servidor..."
        # Actualización de repositorios
        echo -e "\nActualizando repositorios..."
        # Usamos aquí la salida del comando a /dev/null, para que el usuario no vea la salida del comando
        # los errores 2> los lanzamos al mismo lugar que 1, es decir a /dev/null
        apt update && apt upgrade -y > /dev/null 2>&1

        # Instalar java y sus dependencias
        echo -e "\nInstalando dependencias de java..."
        apt install openjdk-21-jre -y > /dev/null 2>&1

        # Creamos directorio para server Minecraft y descargamos el server
        echo -e "\nCreando directorio del servidor..."
        mkdir -p ${MINECRAFT_DIR} && cd ${MINECRAFT_DIR}
        wget "${SERVER_JAR_URL}" -O server.jar
        java -Xmx1024M -Xms1024M -jar server.jar nogui > /var/log/server_mine.log 2>&1 &
        sleep 10
        # Aceptamos el EULA del servidor
        echo "eula=true" > eula.txt
```

Figura 12

Primero actualizamos repositorios y los paquetes con update y upgrade. A continuación descargamos las dependencias de java con openjdk-21 (jdk hace referencia al *java developer kit*, una serie de paquetes que tienen tanto el runtime enviroment como las dependencias del lenguaje). Una vez instalado, se creará el directorio donde se alojará el servidor, nuestro caso será /opt/ y con wget, descargaremos los archivos del server que irán en el formato .jar, compilados de java.

Y ahora se ejecutará con java y los comandos para esta versión -Xmx1024M con -jar para especificar que es un archivo compilado y con la opción nogui para evitar que salga la interfaz gráfica. Este comando lanzará por primera vez el servidor creando los archivos de configuración y ejecución del servidor, además todo el contenido del comando lo lanzaremos a un log personalizado **server.log** tanto si es error como si no y con el **&** mantendremos el comando en segundo plano.

Haremos un **sleep** para esperar que se ejecute el servidor y creamos un archivo **eula.txt con el valor eula=true**. Esto hará que el programa entiende que aceptamos la condiciones de uso.

```
# Configuramos las propiedades de server
echo -e "\nPrepárese para colocar las propiedades!..."
sed -i "s/online-mode=true/online-mode=false/" server.properties
echo -e "\nCantidad de jugadores (máx. 10): "
read PLY
while (( PLY > 10 || PLY < 1 )) ; do
    read -p "\nDebe estar entre 1 y 10. Introduzca cantidad de jugadores: " PLY
done
sed -i "s/max-players=20/max-players=${PLY}/" ${MINECRAFT_DIR}/server.properties
echo -e "\n¿Qué dificultad quiere [0=Pacífico, 1=Fácil, 2=Normal, 3=Difícil]? "
read DIF
case $DIF in
    0|1|2|3) sed -i "s/difficulty=.*/difficulty=${DIF}/" ${MINECRAFT_DIR}/server.properties
    ;;
    *) echo -e "\nOpción inválida. Usando dificultad predeterminada (Normal)."
    ;;
esac
echo -e "\nEscriba el mensaje de bienvenida: "
read MESG
sed -i "s/motd=.*/motd=${MESG}/" ${MINECRAFT_DIR}/server.properties
```

Figura 13

Ahora irán apareciendo las diferentes opciones que tenemos, número de jugadores, la dificultad del servidor, el mensaje de bienvenida (figura 13).

```
            # Convertimos el server en servicio
            echo -e "\nCreando servicio!..."
            touch ${SERVICE_FILE}
            chmod 777 ${SERVICE_FILE}
                        # Crear servicio systemd
            cat <<EOF > "${SERVICE_FILE}"
[Unit]
Description=Servidor de Minecraft
After=network.target

[Service]
User=root
WorkingDirectory=${MINECRAFT_DIR}
ExecStart=/usr/bin/java -Xmx1024M -Xms1024M -jar ${MINECRAFT_DIR}/server.jar nogui
Restart=on-failure

[Install]
WantedBy=multi-user.target
EOF
            systemctl daemon-reload
            # Aquí los errores los lanzamos al log personalizados
            systemctl enable minecraft.service > /var/log/server_mine.log 2>&1
            systemctl restart minecraft.service > /var/log/server_mine.log 2>&1
            echo -e "\nServidor de Minecraft instalado y en ejecución."
            exit 0
        ;;
```

Figura 14

La última parte del proceso se basará en la creación de un servicio específico para la ejecución desde servidor, de este modo hacemos automatizamos el arranque del servidor. Esto además lo hacemos creando el archivo mediante un cat con EOF y la estructura de un servicio. Con EOF determinamos que todo lo que vaya después del cat hasta la próxima vez que salga EOF, se ejecuta todo.

Opción 2: desinstalar el servidor

```
2)
    clear
    echo -e "\nDesinstalando servidor..."
    systemctl stop minecraft.service > /var/log/server_mine.log 2>&1
    systemctl disable minecraft.service > /var/log/server_mine.log 2>&1
    rm -rf "${MINECRAFT_DIR}" "${SERVICE_FILE}"
    apt purge openjdk-21-jre -y > /dev/null 2>&1
    systemctl daemon-reload
    echo -e "\nServidor y dependencias eliminados."
    exit 0
;;
```

Figura 15

La opción 2 desinstala el servidor, siguiendo la secuencia de parar el servidor, borrar el servicio, borrar el directorio y desinstalar las dependencias de java (figura 15).

Opción 3: crear o programar copias de seguridad

```
3)
    clear
    echo -e "\nCreando copia de seguridad..."
    hacer_backup
    echo -e "\n¿Quiere programar la copia de seguridad [S/N]?"
    read B_OP
    if [[ ${B_OP} == [sS] ]]; then
        programar_backup
    fi
    echo -e "\nSaliendo del programa..."
    exit 0
;;
```

Figura 16

Pulsando 3 accedemos al submenú de las copias de seguridad, en este caso una vez que termine la primera copia de seguridad, nos preguntará si queremos progamar dicha copia, si aceptamos se lanza la función **programar_backup**.

Opción h y 0: ayuda y salir

```
0)
    clear
    echo "Saliendo del programa..."
    exit 0
;;
h)
    clear
    ayuda
    echo -e "\nSeleccione una opción: "
    read OP
;;
*)
    clear
    echo -e "\nOpción inválida"
    ayuda
    echo -e "\nSeleccione una opción: "
    read OP
;;
```

Figura 17

Las últimas opciones, nos permiten salir del bucle y del menú o lanzar la función de ayuda (figura 17).

4. Funcionamiento y comprobaciones

Comprobado la construcción del script, vamos a lanzarlo. Cuando lo ejecutamos como sudo veremos el menú principal (figura 18).

Figura 18

Y veremos el menú que podemos ejecutar, pulsamos la primera opción.

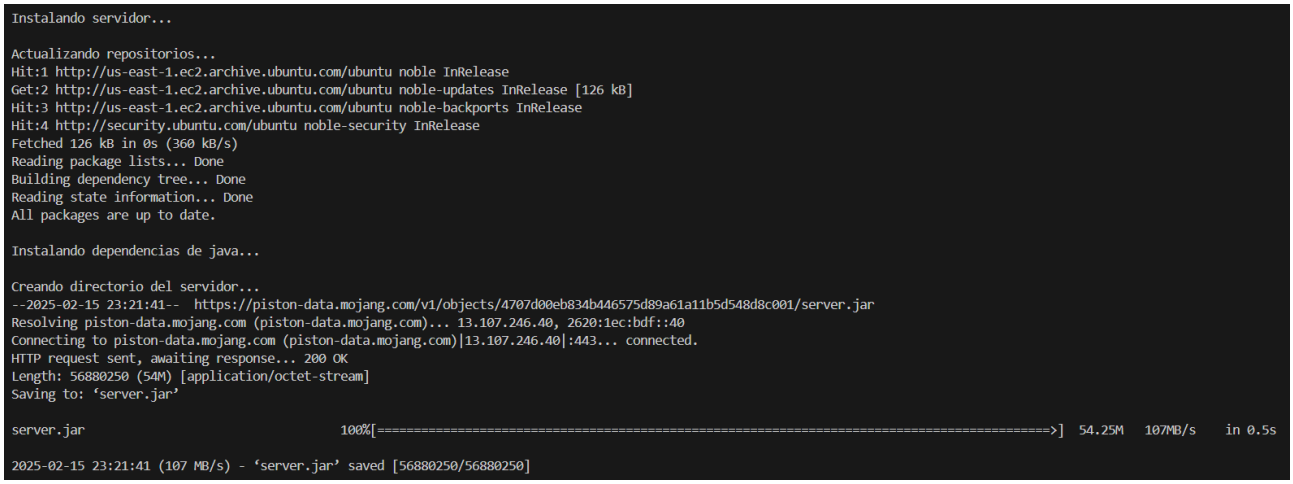

Figura 19

Se instalan las dependencias y colocamos las opciones (figura 20).

```
2025-02-15 23:21:41 (107 MB/s) - ‘server.jar’ saved [56880250/56880250]


Prepárese para colocar las propiedades!...

Cantidad de jugadores (máx. 10):
2

¿Qué dificultad quiere [0=Pacífico, 1=Fácil, 2=Normal, 3=Difícil]?
0

Escriba el mensaje de bienvenida:
Bienvenido al servidor de prueba!

Creando servicio!...

Servidor de Minecraft instalado y en ejecución.
```

Figura 20

Y comprobamos que el servicio se haya creado con systemctl minecraft.service (figura 21).

```
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$ sudo systemctl status minecraft.service
● minecraft.service - Servidor de Minecraft
     Loaded: loaded (/etc/systemd/system/minecraft.service; enabled; preset: enabled)
     Active: active (running) since Sat 2025-02-15 23:22:34 UTC; 1min 46s ago
   Main PID: 3189 (java)
      Tasks: 33 (limit: 4676)
     Memory: 1.0G (peak: 1.0G)
        CPU: 55.208s
     CGroup: /system.slice/minecraft.service
             └─3189 /usr/bin/java -Xmx1024M -Xms1024M -jar /opt/minecraft-server/server.jar nogui

Feb 15 23:23:03 ip-172-31-26-237 java[3189]: [23:23:03] [Worker-Main-1/INFO]: Preparing spawn area: 2%
Feb 15 23:23:03 ip-172-31-26-237 java[3189]: [23:23:03] [Worker-Main-1/INFO]: Preparing spawn area: 2%
Feb 15 23:23:04 ip-172-31-26-237 java[3189]: [23:23:04] [Worker-Main-1/INFO]: Preparing spawn area: 2%
Feb 15 23:23:04 ip-172-31-26-237 java[3189]: [23:23:04] [Worker-Main-1/INFO]: Preparing spawn area: 2%
Feb 15 23:23:05 ip-172-31-26-237 java[3189]: [23:23:05] [Worker-Main-1/INFO]: Preparing spawn area: 2%
Feb 15 23:23:05 ip-172-31-26-237 java[3189]: [23:23:05] [Worker-Main-1/INFO]: Preparing spawn area: 2%
Feb 15 23:23:06 ip-172-31-26-237 java[3189]: [23:23:06] [Worker-Main-1/INFO]: Preparing spawn area: 75%
Feb 15 23:23:06 ip-172-31-26-237 java[3189]: [23:23:06] [Server thread/INFO]: Time elapsed: 8023 ms
Feb 15 23:23:06 ip-172-31-26-237 java[3189]: [23:23:06] [Server thread/INFO]: Done (21.831s)! For help, type "help"
Feb 15 23:24:06 ip-172-31-26-237 java[3189]: [23:24:06] [Server thread/INFO]: Server empty for 60 seconds, pausing
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$
```

Figura 21

Comprobamos también que se haya creado el log (figura 22).

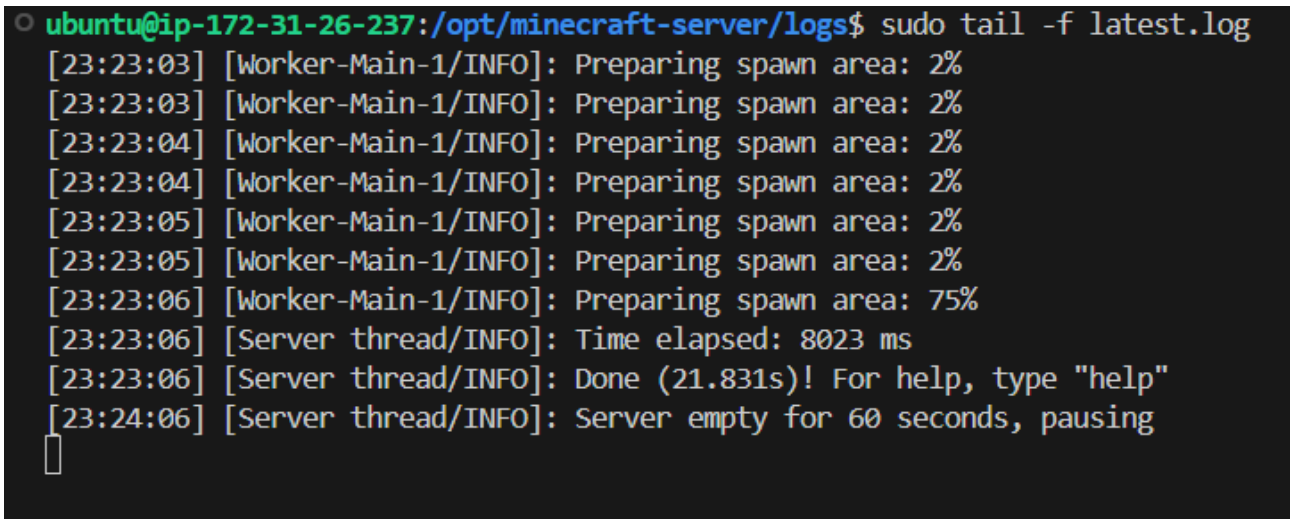

Figura 22

Con esto ya tendríamos nuestro servidor operativo.

Si pulsamos las demás opciones vemos el resultado, al pulsar el 2, se eliminan todas las dependencias del servidor (figura 23).

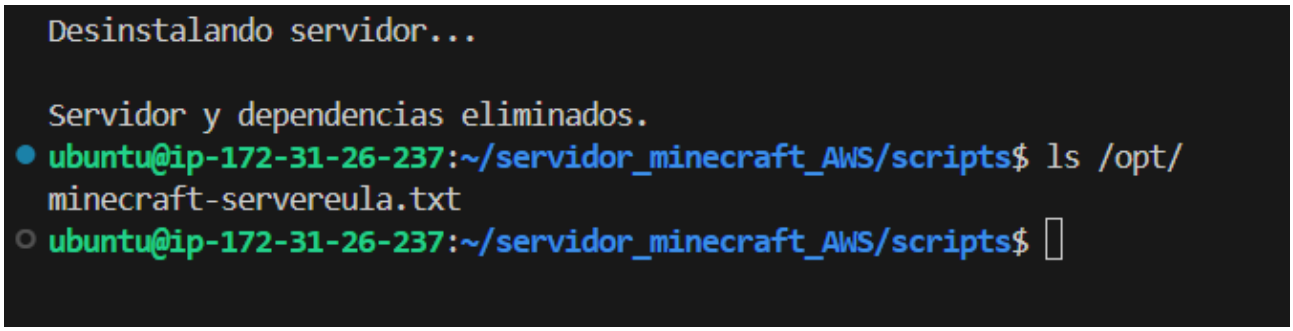

Figura 23

Y al pulsar el 3 nos nos creará un directorio y una copia de seguridad (figura 24-25) y además la vamos a programar.

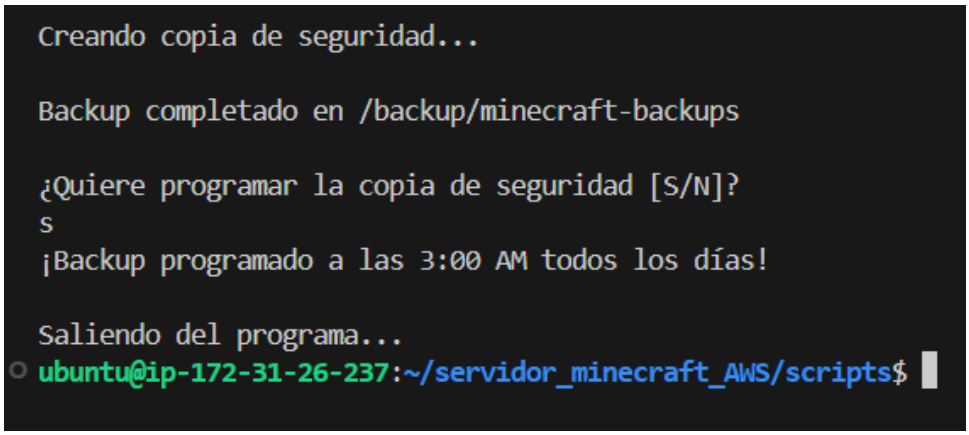

Figura 24

```
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$ ls /backup/minecraft-backups/
backup-2025-02-15_23:33.tar.gz
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$
```

Figura 25

Y comprobamos con crontab -l que tengamos la tarea creada (figura 26).

```
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$ sudo crontab -l
0 3 * * * tar -czvf /backup/minecraft-backups/backup-2025-02-15_23:31.tar.gz /opt/minecraft-server
```

Figura 26

Finalmente, para poder comprobar el funcionamiento del servidor abrimos el launcher de Minecraft y colocamos la ip del servidor y entramos (figura 27).

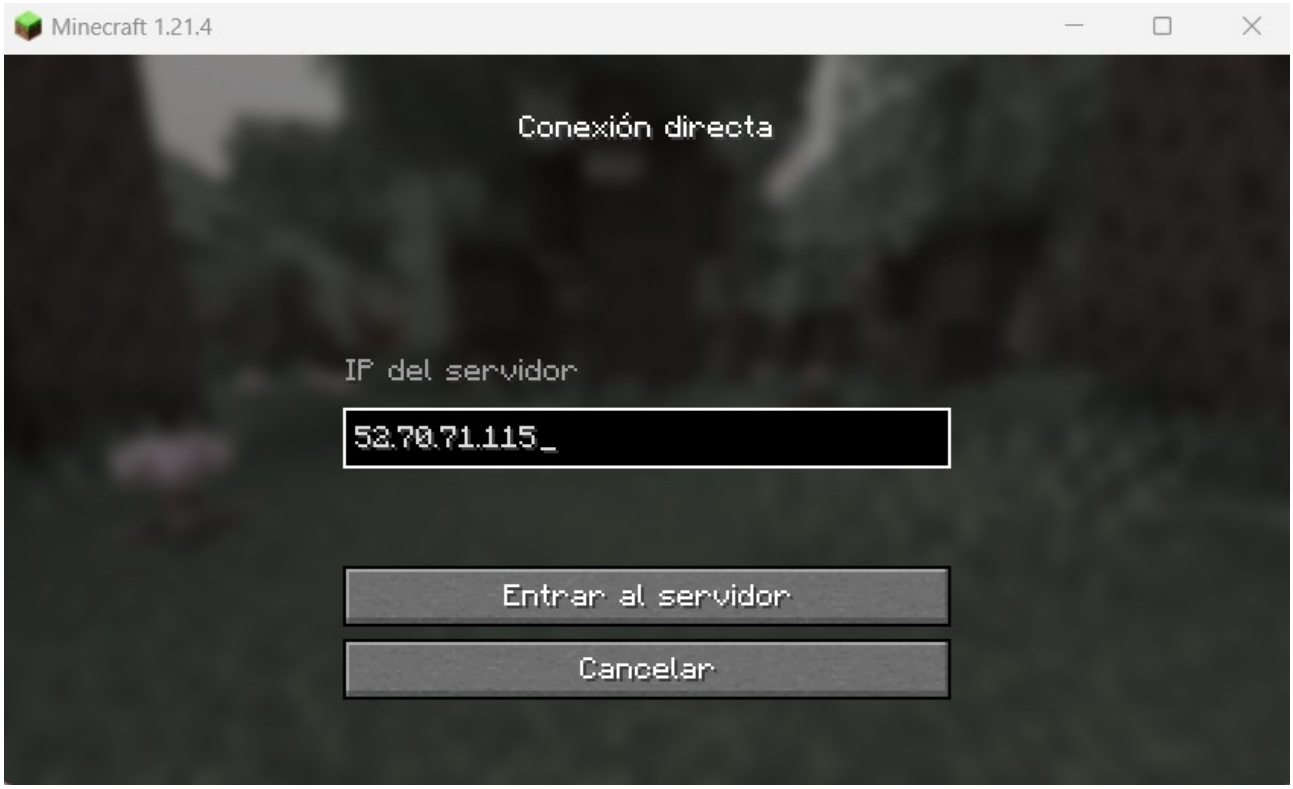

Figura 27

Ahora dentro del juego ya podemos jugar tranquilamente (figura 28), pero debemos tener en cuenta que este es un servidor sencillo y para grandes alojamientos debemos tener en cuenta la escalabilidad del proyecto.

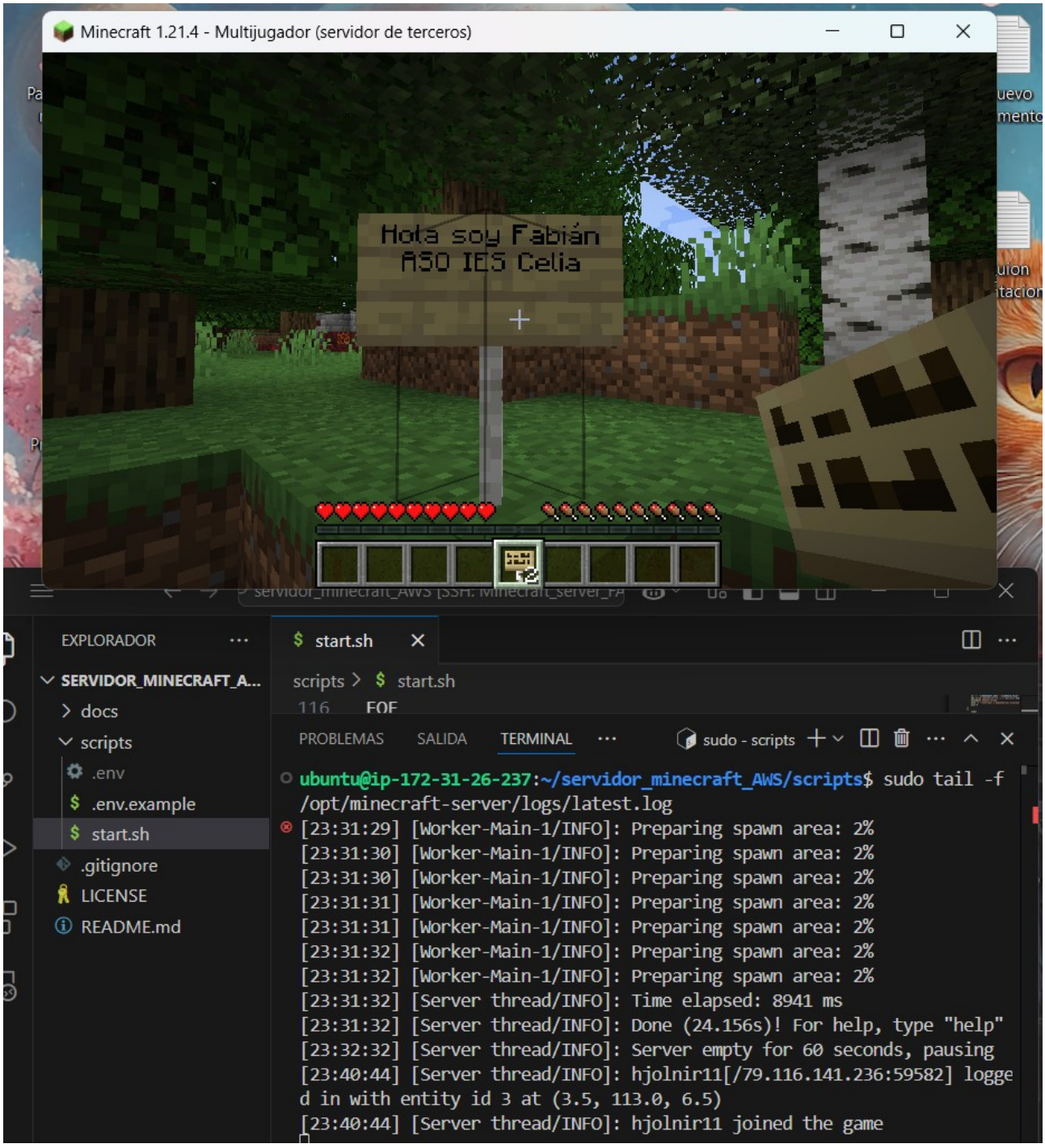
Minecraft 1.21.4 - Multijugador (servidor de terceros)
Hola soy Fabián
ASO IES Celia
EXPLORADOR
start.sh
SERVIDOR_MINECRAFT_A...
docs
scripts
.env
.env.example
start.sh
.gitignore
LICENSE
README.md
scripts > start.sh
PROBLEMAS
SALIDA
TERMINAL
sudo - scripts
ubuntu@ip-172-31-26-237:~/servidor_minecraft_AWS/scripts$ sudo tail -f /opt/minecraft-server/logs/latest.log
[23:31:29] [Worker-Main-1/INFO]: Preparing spawn area: 2%
[23:31:30] [Worker-Main-1/INFO]: Preparing spawn area: 2%
[23:31:30] [Worker-Main-1/INFO]: Preparing spawn area: 2%
[23:31:31] [Worker-Main-1/INFO]: Preparing spawn area: 2%
[23:31:31] [Worker-Main-1/INFO]: Preparing spawn area: 2%
[23:31:32] [Worker-Main-1/INFO]: Preparing spawn area: 2%
[23:31:32] [Worker-Main-1/INFO]: Preparing spawn area: 2%
[23:31:32] [Server thread/INFO]: Time elapsed: 8941 ms
[23:31:32] [Server thread/INFO]: Done (24.156s)! For help, type "help"
[23:32:32] [Server thread/INFO]: Server empty for 60 seconds, pausing
[23:40:44] [Server thread/INFO]: hjolnir11[/79.116.141.236:59582] logged in with entity id 3 at (3.5, 113.0, 6.5)
[23:40:44] [Server thread/INFO]: hjolnir11 joined the game

Figura 28

5. Conclusiones

Como hemos visto durante el periodo de instalación, bash es un lenguaje potente que nos permite hacer grandes operaciones siguiendo patrones sencillos y comandos de forma estructurada, además gracias a aplicar dicho lenguaje no solamente a los ejercicios, si no a un contexto real de implantación de programas podemos comprobar su verdadera potencia.

Todo ello nos lleva a comprender el potencial de administración que nos ofrece bash a la hora de automatizar y gestionar tareas con tanta importancia en los sistemas operativos como la creación de servicios, la ejecución de programas en segundo plano, la escalada de permisos en diferentes archivos y la creación de diferentes archivos y la personalización de los mismos.

6. Referencias

Download server for Minecraft | Minecraft – Descargar servidor de Minecraft

Tutorial:Setting up a server – Minecraft Wiki - Tutorial para creación de script y configuraciones

falclop/servidor_minecraft_AWS: Despliegue de un servidor de minecraft en AWS – Repositorio personal del programa