

Recuperar archivos LOG de MVs

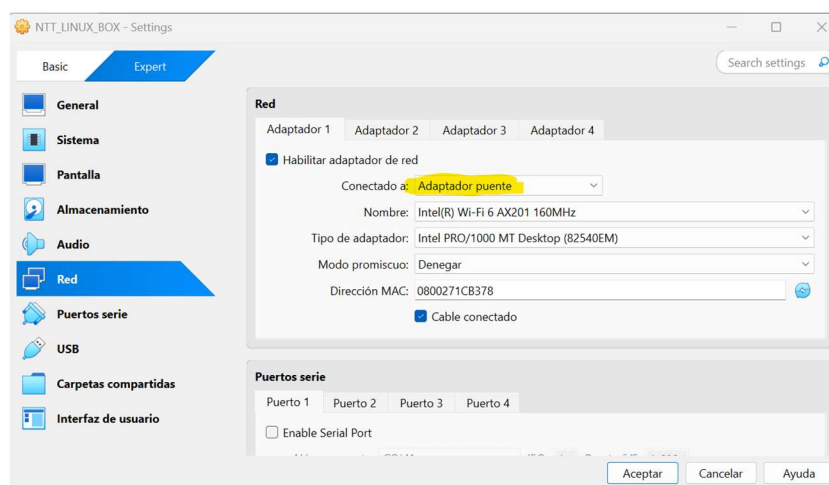
El objetivo de esta POC era realizar una pipeline en Jenkins que nos permitiera simular una recuperación de archivos log de diferentes máquinas / servidores con un filtrado previo. Esto lo haremos mediante el protocolo SSH, que nos permite conectarnos de manera segura a las máquinas. Además, usaremos el protocolo SCP para recuperar o copiar los archivos de los servidores a nuestra pipe.

CONFIGURACIÓN MAQUINA VIRTUAL

Como obviamente no disponemos de acceso a las máquinas del equipo, lo que haremos será simularlas con una máquina virtual (VM). Para esta POC he utilizado la **VirtualBox** de **Oracle**. Se puede descargar de forma gratuita en internet. Una vez lo tengamos descargado y abierto, antes deberemos descargarnos el sistema operativo **Ubuntu**, ya que usaremos Linux para esta VM. Y dentro de VirtualBox, crearemos una nueva máquina virtual donde instalaremos este sistema operativo que tenemos descargado.

Antes de iniciar la VM, un paso importante para que todo esto funcione es la configuración de la máquina, ya que si no conseguimos conectarnos a internet, será imposible acceder a esta mediante SSH con otro dispositivo.

Para ello, iremos a la configuración de la máquina, y vamos al apartado de red, donde veremos que por defecto está en NAT. A pesar de que con NAT nos podemos conectar a internet no nos sirve, ya que es invisible para los otros dispositivos de la red. Por lo tanto, no nos proporcionará una IP a la que podamos conectarnos. Para ello cambiaremos de NAT a **Bridged Networking**, que permite la comunicación entre la VM y los dispositivos de la red.



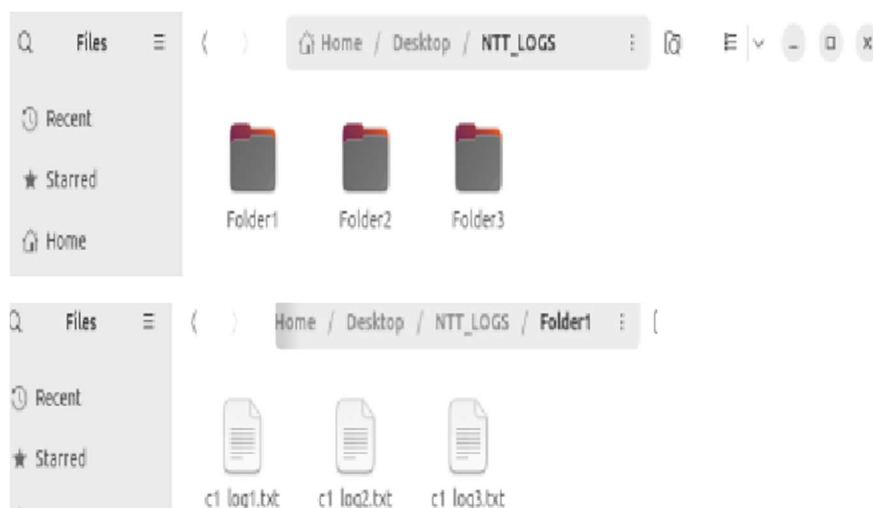
Aquí cabe destacar que este tipo de conexión está bloqueada en la red de la oficina de NTT DATA, o al menos no encontré la forma de hacerlo funcionar en esa red. Pero si lo hacemos desde una red personal debería funcionar sin mucho inconveniente.

Ahora ya podemos iniciar la VM y ver si tenemos conexión a internet. Si es así, lo siguiente es comprobar que tenemos una IP de red a la que podemos acceder desde otros dispositivos. Para ellos abriremos una terminal y escribiremos:

ip addr show

Esto nos mostrara nuestra IP si es que tenemos, por ejemplo si estuviéramos en NAT, tan solo veríamos la IP local. Ahora desde otro dispositivo podemos hacer un ping a esta dirección IP para comprobar que podemos llegar a la VM. Si es así podemos continuar.

Ahora, dentro de los archivos crearemos una jerarquía de carpetas que simularan las distintas maquinas:



NTTData_Logs:

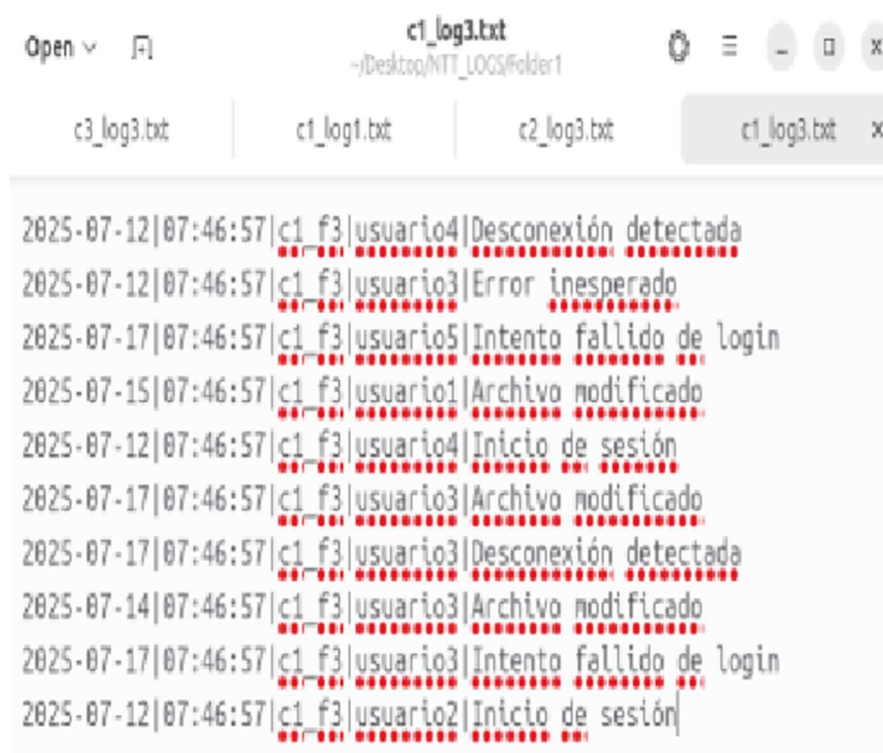
- Folder1
 - Logs1.txt
 - Logs2.txt
 - Logs3.txt
- Folder2
 - Logs1.txt
 - Logs2.txt
 - Logs3.txt
- Folder3
 - Logs1.txt
 - Logs2.txt
 - Logs3.txt

Cada Folder representa un servidor distinto con sus respectivos archivos de logs. Y cada archivo tiene el siguiente formato:

2025-07-12|07:46:57|c1_f3|usuario4|Desconexión detectada

Fecha | hora | carpetaN_archivoN | usuario | mensaje

Por lo tanto este línea se ha recuperado de la primera maquina (c1) y es del archivo log3 (f3).



```
Open v F1 c1_log3.txt ~/Desktop/NTT_LOGS/Folder1
c3_log3.txt c1_log1.txt c2_log3.txt c1_log3.txt x

2025-07-12|07:46:57|c1_f3|usuario4|Desconexión detectada
2025-07-12|07:46:57|c1_f3|usuario3|Error inesperado
2025-07-17|07:46:57|c1_f3|usuario5|Intento fallido de login
2025-07-15|07:46:57|c1_f3|usuario1|Archivo modificado
2025-07-12|07:46:57|c1_f3|usuario4|Inicio de sesión
2025-07-17|07:46:57|c1_f3|usuario3|Archivo modificado
2025-07-17|07:46:57|c1_f3|usuario3|Desconexión detectada
2025-07-14|07:46:57|c1_f3|usuario3|Archivo modificado
2025-07-17|07:46:57|c1_f3|usuario3|Intento fallido de login
2025-07-12|07:46:57|c1_f3|usuario2|Inicio de sesión
```

CONEXIÓN SSH

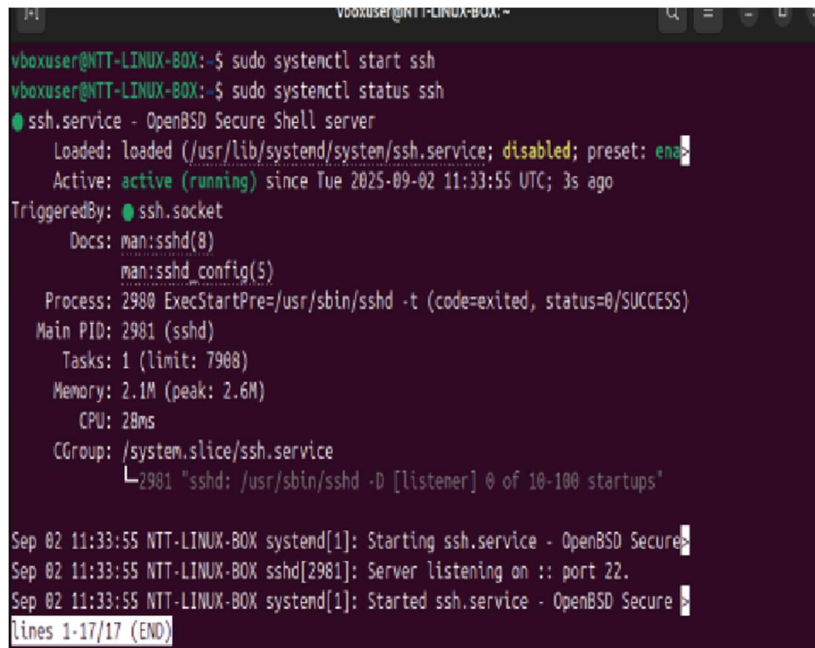
Una vez tenemos la jerarquía de carpetas creada, vamos a comprobar que podemos conectarnos con SSH a esta VM. Para ello primero tenemos que instalarlo en la VM, dentro de la terminal escribiremos:

sudo apt install openssh-server

una vez instalado podemos utilizar los siguientes comandos

`sudo systemctl start ssh` – Para iniciar el servidor ssh

`sudo systemctl status ssh` – Para comprobar el estado del servidor y logs.



```
vboxuser@NTT-LINUX-BOX:~$ sudo systemctl start ssh
vboxuser@NTT-LINUX-BOX:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: enabled)
   Active: active (running) since Tue 2025-09-02 11:33:55 UTC; 3s ago
     TriggeredBy: ● ssh.socket
     Docs: man:sshd(8)
           man:sshd_config(5)
    Process: 2980 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 2981 (sshd)
      Tasks: 1 (limit: 7968)
     Memory: 2.1M (peak: 2.6M)
        CPU: 28ms
    CGroup: /system.slice/ssh.service
            └─2981 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Sep 02 11:33:55 NTT-LINUX-BOX systemd[1]: Starting ssh.service - OpenBSD Secure
Sep 02 11:33:55 NTT-LINUX-BOX sshd[2981]: Server listening on :: port 22.
Sep 02 11:33:55 NTT-LINUX-BOX systemd[1]: Started ssh.service - OpenBSD Secure
lines 1-17/17 (END)
```

Ahora desde nuestro otro dispositivo donde usaremos la pipe de Jenkins, abriremos una terminal y trataremos de conectarnos mediante SSH a la VM.

Para esto escribimos: `ssh usuario@IP`

`ssh vboxuser@ip_VM`

Nos pedirá la contraseña para acceder y al ponerla podremos recuperar los archivos mediante scp si quisiéramos. Ahora hemos comprobado que podemos conectarnos, pero aun tenemos un problema. Nos pide la contraseña cada vez que queremos conectarnos y para la pipe sera un inconveniente, ya que no podemos estar poniendo la contraseña cada vez que queramos conectar la pipe.

Para ellos existen la **keys** podemos desde nuestro dispositivo generar una key ssh publica que podemos usar para conectarnos a la VM sin la necesidad de la contraseña.

Para ello haremos lo siguiente, desde el powershell de nuestro dispositivo ejecutaremos el siguiente comando:

`ssh-keygen -t rsa -b 4096 -C "tu_correo@example.com"`

Esto generara una key que quedara guardada en nuestro ordenador, ahora deberemos copiarla a los archivos de la VM para que pueda verificar que somos nosotros a través de la key y no nos pida la contraseña. Conectándonos así de forma automática.

Para ello desde la misma terminal en la que estamos usaremos el siguiente comando:

ssh-copy-id user@remote_host – En nuestro caso user y IP es vboxuser@IP_VM

Una vez hecho esto tendremos que reiniciar el servidor ssh de la VM. Y después cuando lo volvamos a encender deberíamos poder conectarnos sin la necesidad de introducir la contraseña.

Una vez tenemos todo esto, estamos casi listos para empezar la pipe de Jenkins.

CONFIGURACIÓN REPOSITORIO

Por último, uno de los requisitos que de esta POC era poder diferenciar las Folders de la VM como si fueran distintas máquinas o servidores, esto es porq en el mundo real estas máquinas tendrán IPs distintas que deberemos conocer para poder conectarnos de manera independiente a cada una, no se encuentran todas bajo la misma IP. En mi POC solo tengo una máquina virtual y por lo tanto una sola IP en donde se encuentran todas las carpetas.

Aun así lo que hemos hecho es tener un archivo externo donde podemos tener preconfiguradas todas las IPs de cada máquina y su usuario, para conectarnos fácilmente a estas mediante SSH. En la pipe las recuperaremos y nos conectaremos a cada una de manera independiente.

El archivo que yo tengo es el siguiente: **nombre maquina | usuario@IP_maquina**

	Code	Blame
1	Folder1	vboxuser@172.20.10.10
2	Folder2	vboxuser@172.20.10.10
3	Folder3	vboxuser@192.92.19.1

Como podemos ver, la ultima IP esta mal a propósito, para comprobar que falla y se conectan de manera independiente a cada Folder.

Esta configuracion sera un simple .txt que tendremos en el repositorio que queramos, Jenkins clonara este repositorio y recuperar este archivo para utilizar su configuración.

Una vez tenemos todo esto, ya lo tenemos todo listo para poder crear la pipe de Jenkins.

PIPELINE DE JENKINS

A continuación añadiré el código de la pipe de Jenkins, esta contiene comentarios sobre lo que hace cada parte del código.

En resumen los pasos son los siguientes.

- Clonamos el repositorio
- Recuperamos el archivo de configuraciones
- Nos conectamos a cada carpeta de manera independiente y recuperamos los archivos que hayamos seleccionado en los parámetros mediante SCP.
- Aplicamos los filtros a los archivos que hemos recuperado.
- Se escribe el resultado en consola y en un archivo .txt llamado result.

Para mas claridad, adjunto una captura de los posibles parámetros que podemos aplicar al filtrado de logs:

Pipeline SSH_Conexion

Esta ejecución requiere parámetros adicionales:

☒ FILTER_DATE

Wether to filter by date or not.

☒ FILTER_TIME

Wether to filter by time or not.

☒ FILTER_WORDS

Wether to filter by key words or not.

DATE

format: yyyy-mm-dd

2025-07-13

FROM_TIME

format: hh:mm

00:00

TO_TIME

format: hh:mm

23:59

KEY_WORDS

format: word1, word2, word3, ... wordn

usuario1, usuario2

☐ JOIN_WORDS

If checked, only the lines with all the keywords will show.

FILE_NAME

If empty, will filter all files. Otherwise, will filter the given name only

☒ FILTER_ALL

Filter from all VM

VM_SELECTION

Select from which VMs will the logs be filtered.

☐ Folder1

☐ Folder2

☐ Folder3

 Ejecución

Cancel

Los 3 primeros booleanos indican si queremos filtrar o no por esos parámetros, si están seleccionados aplicara el filtro de los strings de abajo.

Ademas en las keywords tambien tenemos un booleano que nos permite buscar por frases concretas. Por ejemplo si buscamos usuario1, conexión. Y marcamos el JOIN_WORDS, se filtraran solo las líneas del usuario1 que el mensaje contenga la palabra “conexión”.

Por ultimo tenemos el filtrado de archivos y maquinas. Por ejemplo FILE_NAME nos permite buscar una archivo especifico o que contenga ese palabra. Si ponemos “log1” tan solo recuperara en el SCP los archivos que coincidan con esa palabra.

Y ya para finalizar, tenemos 4 booleanos mas. Si tenemos marcado FILTER_ALL nos conectaremos una por una a todas las maquinas que tenemos en la lista y filtrara los parámetros. Pero si lo desmarcamos, podemos seleccionar las maquinas concretas a las que queremos conectarnos. Por ejemplo solo la 1 y la 2. Esta lista se puede extender dentro de la configuración de la pipe.

CODIGO PIPE JENKINS:

Para una lectura más sencilla, quizás es preferible verlo desde Git-Hub: [código](#)

```
pipeline {
    agent any

    parameters{
        booleanParam(
            defaultValue: true,
            description: 'Wether to filter by date or not.',
            name: 'FILTER_DATE'
        )

        booleanParam(
            defaultValue: true,
            description: 'Wether to filter by time or not.',
            name: 'FILTER_TIME'
        )

        booleanParam(
            defaultValue: true,
            description: 'Wether to filter by key words or not.',
            name: 'FILTER_WORDS'
```

)

```
validatingString(  
  name: 'DATE',  
  description: 'format: yyyy-mm-dd',  
  regex: /^\\s*[0-9-]+\\s*$/,  
  failedValidationMessage: 'Use numbers and -',  
  defaultValue: '2025-07-13'  
)
```

```
validatingString(  
  name: 'FROM_TIME',  
  description: 'format: hh:mm',  
  regex: /^\\s*[0-9:]+\\s*$/,  
  failedValidationMessage: 'Use numbers and :',  
  defaultValue: '00:00'  
)
```

```
validatingString(  
  name: 'TO_TIME',  
  description: 'format: hh:mm',  
  regex: /^\\s*[0-9:]+\\s*$/,  
  failedValidationMessage: 'Use numbers and :',  
  defaultValue: '23:59'  
)
```

```
validatingString(  
  name: 'KEY_WORDS',  
  description: 'format: word1, word2, word3, ... wordn',  
  regex: /.*/,  
  defaultValue: 'usuario1, usuario2'  
)
```

//TRUE --> queremos filtrar las palabras que contengan todas las keywords.

//FALSE --> queremos filtrar las palabras que contengan al menos UNA keyword.

```
booleanParam(  
  defaultValue: false,  
  description: 'If checked, only the lines with all the keywords will show.',  
  name: 'JOIN_WORDS'
```



```

    )

    validatingString(
        name: 'FILE_NAME',
        description: 'If empty, will filter all files. Otherwise, will filter the given name only',
        regex: /^[A-Za-z0-9_-]*$/, //PERMIT ALL
        defaultValue: ""
    )

    booleanParam(
        defaultValue: true,
        description: 'Filter from all VM',
        name: 'FILTER_ALL'
    )
}

environment{
    WP_NAME = 'logs' //NOMBRE DEL WORKSPACE QUE UTILIZAREMOS AQUI EN JENKINS
    OUTPUT_F = 'Result.txt' //NOMBRE DEL ARCHIVO FINAL CON EL RESULTADO DE LOS LOGS
    VM_HOST = 'vboxuser' //USUARIO DE LA VM, PARA CONECTARNOS MEDIANTE SSH
    VM_IP = '172.20.10.10' //IP DE LA VM, SERVIDOR SSH
    LOGS_PATH = '/home/vboxuser/Desktop/NTT_LOGS/' //PATH A LOS LOGS DE LA VM
    REPO_URL = 'https://github.com/RubenMailloBaena/JenkinsPipelineTest.git'
    BRANCH = 'main'
    FILE_DIR = 'git-repo/Jenkins_VM_Logs/tools'
    CONFIG_NAME = 'VM_config.txt'
    CONFIG_ARRAY = ""
}

stages {

    //CREAMOS EL WORKSPACE EN JENKINS, DONDE GUARDAREMOS TODOS LOS ARCHIVOS.
    stage('Create Workspace'){
        steps{
            script{
                cleanWs()
                powershell'''
                if(Test-Path -Path "$env:WP_NAME"){
                    Remove-Item -Path "$env:WP_NAME" -Recurse -Force
                }
            '''
        }
    }
}

```

```

    }

    mkdir "$env:WP_NAME"

    mkdir "git-repo"

    ""

  }

}

}

```

//MEDIANTE SSH, NOS CONECTAMOS A LA VM. RECUPERAMOS TODOS LOS ARCHIVOS CON LOS LOGS.

```

stage('Get logs from VM to WorkSpace') {

  steps {

    script {

      withCredentials([sshUserPrivateKey(credentialsId: 'ssh', keyFileVariable: 'PK')]) {

        //Clonamos el repositorio de Git

        dir('git-repo'){

          git branch: "${env:BRANCH}", url: "${env:REPO_URL}"

        }

        //Obtenemos el archivo con las configuraciones SSH de cada maquina, Folder en este caso.

        def configsFile = ""

        dir("${FILE_DIR}"){

          if(fileExists("${CONFIG_NAME})){

            configsFile = readFile("${CONFIG_NAME}")

          } else {

            error "NO CONFIG FILE FOUND."

          }

        }

      }

    }

  }

}

```

```

//Creamos un mapa con las distintas configuraciones de cada maquina.

//Por ejemplo en el archivo nos llega como Folder1 | vboxuser@172.20.10.10

//Y el resultado nos quedara: [Folder1:[user:'vboxuser', ip:'172.20.10.10']].

//De esta manera podemos usar las configuraciones de cada uno mas adelante.

def VMconfigs = []

configsFile.readLines().each { newLine ->

  def line = newLine.trim()

  if (!line) return //Si no hay ninguna linea, salimos.

```

```

def parts = line.split(/\//)

def folder = parts[0].trim() //Guardamos la priemra parte: FolderN

def userAtIp = parts[1].trim() //Y la parte de configuracion

def ui = userAtIp.split('@') //Volvemos a separar el user y la ip

//Y gurdamos el resultado en el mapa.
VMconfigs[folder] = [user: ui[0].trim(), ip: ui[1].trim()]
}
echo "Config SSH por folder: ${VMconfigs}"

//Aqui determinamos de que carpetas recuperaremos los logs segun los parametros que hayamos seleccionado.
//Si teniamos marcado el FILTER ALL, Obtendremos todas las configuraciones del mapa creado.
//En caso contrario solo añadiremos a targetFolders las que hemos seleccionado en parametros.
def targetFolders = []
if (params.FILTER_ALL) {
    targetFolders = VMconfigs.keySet().toList().sort()
} else {
    if (!params.VM_SELECTION.trim()){
        error "No Folder Selected!"
    }
    targetFolders = params.VM_SELECTION.split(',').collect { it.trim() }
}

//Construimos un script ssh para ejecutar mas adelante con todo el contenido necesario.
//Aqui en esta primera parte, creamos un agente ssh, que nos servira para establecer la conexion.
def scpScript = """
    eval \$(ssh-agent -s)
    trap "ssh-agent -k" EXIT
    ssh-add "$PK"
    """.stripIndent()

//Por cada Folder que estemos buscando, obtenemos su configuracion del mapa creado anteriormente.
targetFolders.each { folder ->
    def cfg = VMconfigs[folder]
    if (cfg == null) {
        echo "WARN: No SSH config found for ${folder}."
    }
}

```

```

        return
    }

    //guardamos el usuario e IP de cada Foler.

    def user = cfg.user

    def ip = cfg.ip

    // Si filtramos archivos especificos con FILE_NAME, copiamos solo los archivos que cuadren con los que
    buscamos

    def VMFolder = "${env.LOGS_PATH}${folder}/"

    def fileName = (params.FILE_NAME?.trim())

    ? "${VMFolder}*${params.FILE_NAME}*"

    : VMFolder

    //Si hay, lo añadimos a la ruta que buscamos, si no simplemente buscaremos la carpeta.

    //Añadimos al script la parte donde recuperamos los archivos necesarios para mas adelante filtrarlos.

    //Lo hacemos mediante scp. Nos conectamos y buscamos la ruta que hemos contruido.

    scpScript += """

        scp -o StrictHostKeyChecking=no -r "${user}@${ip}:${fileName}" "${env.WP_NAME}/" \

        || echo "error while getting files ${user}@${ip}:${fileName}"

    """.stripIndent() //Esta ultima funcion elimina las comillas "" para que al ejecutar no interfieran
    }

    //Aqui finalmente ejecutamos el script sh, donde se recuperarn todos los archivos.

    sh scpScript

    }

    }

    }

    }

    //RECORREMOS TODOS LOS ARCHIVOS DE LOGS, LINEA POR LINEA APLICANDO LOS FILTROS DE LOS PARAMETROS.

    stage('Filter Logs') {

        steps {

            dir("${WP_NAME}") {

                script {

                    //pillamos los parametros del usuario.

                    def result = []

                    def dateParam = params.DATE

                    def fromTimeParam = params.FROM_TIME

                    def toTimeParam = params.TO_TIME

                    def keywords = params.KEY_WORDS.split(',').collect { it.trim() }

```

```

//recorremos todos los archivos de logs que tenemos, por carpetas y txts.
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        def filePath = "c${i}_log${j}.txt"
        def folderPath = "Folder${i}/${filePath}"

        def readFrom = "${folderPath}"

        //En caso de que hayamos filtrado por archivos especificos, no recuperaremos las carpetas enteras.
        //Entonces si no esta la carpeta que buscamos, ni el archivo respectivo, pasamos a la siguiente iteracion.
        if(!fileExists("${folderPath}")){ //Si la carpeta que buscamos no existe
            if(!fileExists("${filePath}")){ //Si existe en archivos en vez de carpetas, al filtrar por archivo
                continue
            }
            readFrom = "${filePath}"
        }
        def content = readFile(readFrom)

        //separamos el contenido linea por linea y lo recorremos.
        content.split("\n").each { line ->

            //separamos el contenido de la linea por partes
            //formato --> date|time|info|user|message
            def parts = line.split("\\|")

            def logDate = parts[0]
            def logTime = parts[1]
            def logUser = parts[3]
            def logMessage = parts[4]

            //Si no queremos filtrar por fecha pasamos directamente, por el contrario comprobamos que coincida
            if (!params.FILTER_DATE || logDate == dateParam) {

                //Lo mismo con el tiempo, si queremos filtrarlo miramos que este dentro del rango.
                if (!params.FILTER_TIME || (logTime >= fromTimeParam && logTime <= toTimeParam)) {

                    //Para ver si pasa el filtro o no
                    boolean shouldAddLine = true

```

```

def lineToCheck = logUser + logMessage

//Si queremos filtrar por palabras, miramos si queremos que todas esten dentro de la misma linea
//O si por el contrario nos sirve con que la linea contenga una de las keywords para que sea valida
if(params.FILTER_WORDS){
  if(params.JOIN_WORDS){
    //si alguna de las keywords no esta en la linea, no es valida, no la añadimos al resultado
    for(def keyword : keywords){
      if(!line.toLowerCase().contains(keyword.toLowerCase())){
        shouldAddLine = false
        break
      }
    }
  } else {
    //Si alguna de las keywords esta en la linea, lo añadimos al resultado.
    shouldAddLine = false
    for(def keyword : keywords){
      if(line.toLowerCase().contains(keyword.toLowerCase())){
        shouldAddLine = true
        break
      }
    }
  }
}
if (shouldAddLine) {
  result.add(line)
}
}
}
}

//Escribimos el archivo de salida con el contenido del resultado y lo guardamos en el workspace antes creado.
writeFile file: "${env.OUTPUT_F}", text: result.join("\n")

//Imprimimos por consola el resultado.
def fileContent = readFile("${env.OUTPUT_F}")
echo "LOGS OBTAINED FROM FILTERS ---> \n${fileContent}"
}

```

}
}
}
}
}