Tercer Entregable Virtualización de Sistemas.

En este repositorio, **fork del original**, concretamente en la rama main, creada a partir de la rama master mediante un **git checkout -b main**, realizaremos el entregable tercero de la asignatura Virtualización de Sistemas, que abarca las tecnologías de **Terraform**, **Sistema de Control de Versiones SCV** y **Jenkins**.

Autores

- Álvaro Pérez Vargas **EHDEPELUCHE**
- Mª Elena Vázquez Rodríguez **elenavaz**

Índice

Creación de la imagen de Jenkins

Jenkins está de por sí disponible como imagen para **docker** pero no incluye el plug-in de **Blue Ocean** por lo que vamos a hacer una instalación personalizada de Jenkins en docker y, para ello, haremos uso de un **Dockerfile** desde el que crearemos nuestra imagen personalizada.

El fichero **Dockerfile** es el siguiente:

```
FROM jenkins/jenkins:2.479.2-jdk17

USER root

RUN apt-get update && apt-get install -y lsb-release

RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc \
    https://download.docker.com/linux/debian/gpg

RUN echo "deb [arch=$(dpkg --print-architecture) \
    signed-by=/usr/share/keyrings/docker-archive-keyring.asc] \
    https://download.docker.com/linux/debian \
    $(lsb_release -cs) stable" > /etc/apt/sources.list.d/docker.list

RUN apt-get update && apt-get install -y docker-ce-cli

USER jenkins

RUN jenkins-plugin-cli --plugins "blueocean docker-workflow"
```

A continuación una explicación línea a línea del fichero:

- **FROM** jenkins/jenkins:2.479.2-jdk17 indica que queremos usar la imagen **jenkins** del usuario **jenkins**, concretamente la que tiene la etiqueta **2.479.2-jdk17**.
- **USER** root nos indica que todas las demás líneas a continuación se ejecutarán como si fueran usuario root, es decir, con **privilegios elevados**.
- **RUN** apt-get update && apt-get install -y lsb-release realiza dos acciones: primero **actualiza el repositorio de apt-get** y posteriormente instala el paquete de **lsb-release**.
- **RUN** curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc https://download.docker.com/linux/debian/gpg descarga un **archivo de clave gpg** desde la URL indicada y lo guarda en el directorio **keyrings** con el nombre de **docker-archive-keyring.asc**.
- **RUN** echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.asc] https://download.docker.com/linux/debian $(lsb_release -cs) stable" > /etc/apt/sources.list.d/docker.list con **echo** añadimos una línea en el **archivo de fuentes** con la url de docker para debian. Obtiene una versión o otra dependiendo de la arquitectura del ordenador que esté ejecutando la línea.
- **RUN** apt-get update && apt-get install -y docker-ce-cli vuelve a actualizar las fuentes e **instala docker**.
- **USER** jenkins nos indica que todas las demás líneas a continuación se ejecutarán como si fueran **usuario jenkins**, que es el que tiene acceso por defecto a la orden de la siguiente línea.
- **RUN** jenkins-plugin-cli --plugins "blueocean docker-workflow" mediante el uso de **jenkins-plugin-cli** instalamos el plugin que necesitamos, es decir, **blueocean** y **docker-workflow**.

Tras realizar la **build** o construcción del Dockerfile arriba descrito mediante la orden:

```
docker build -t myjenkins-blueocean .
```

tendremos una imagen llamada **myjenkins-blueocean** creada a partir de la imagen original de jenkins pero con las adaptaciones hechas en el Dockerfile. A continuación veremos el despliegue de dos contenedores, uno a partir de la imagen ahora creada, y otra a partir de la imagen de docker **dind**.

Despliegue de los contenedores mediante Terraform

Para poder desplegar ambos contenedores, el de **dind** y el de **myjenkins-blueocean**, de forma más sencilla y automatizando la creación y configuración de redes y volúmenes haremos uso de un fichero de terraform llamado **Despliegue.tf**.

Aquí el contenido del fichero:

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.1"
    }
  }
}
```

```
provider "docker" {
  host = "npipe:////./pipe/docker_engine"
}

resource "docker_network" "jenkins" {
  name = "jenkins"
}

resource "docker_volume" "jenkins_docker_certs" {
  name = "jenkins-docker-certs"
}

resource "docker_volume" "jenkins_data" {
  name = "jenkins-data"
}

resource "docker_container" "jenkins_docker" {
  name  = "jenkins-docker"
  image = "docker:dind"
  privileged = true
  restart = "no"
  networks_advanced {
    name = docker_network.jenkins.name
    aliases = ["docker"]
  }
  env = [
    "DOCKER_TLS_CERTDIR=/certs"
  ]
  volumes {
    volume_name    = docker_volume.jenkins_docker_certs.name
    container_path = "/certs/client"
  }
  volumes {
    volume_name    = docker_volume.jenkins_data.name
    container_path = "/var/jenkins_home"
  }
  ports {
    internal = 2376
    external = 2376
  }
  storage_opts = {
    "overlay2.override_kernel_check" = "1"
  }
}

resource "docker_container" "jenkins_blueocean" {
  name  = "jenkins-blueocean"
  image = "myjenkins-blueocean"
  restart = "on-failure"
  networks_advanced {
    name = docker_network.jenkins.name
  }
  env = [
    "DOCKER_HOST=tcp://docker:2376",
```

```
    "DOCKER_CERT_PATH=/certs/client",
    "DOCKER_TLS_VERIFY=1"
  ]
  volumes {
    volume_name    = docker_volume.jenkins_data.name
    container_path = "/var/jenkins_home"
  }
  volumes {
    volume_name    = docker_volume.jenkins_docker_certs.name
    container_path = "/certs/client"
    read_only      = true
  }
  ports {
    internal = 8080
    external = 8080
  }
}
```

A continuación una explicación por bloques del fichero:

- Bloque **terraform {}**: Incluye el bloque **required_providers {}** que indica a terraform que queremos usar docker como su provider.
- Bloque **provider "docker" {}**: Indica la dirección del host Docker al que terraform se va a conectar, en este caso **"npipe:////./pipe/docker_engine"**.
- Bloques **resource ... {}**: Son los bloques de mayor importancia ya que indican los recursos que va a gestionar terraform. Nosotros utilizaremos cinco:
  - Bloque **resource "docker_network" "jenkins" {}**: declara la red **jenkins**, a la que da de nombre jenkins.
  - Bloque **resource "docker_volume" "jenkins_docker_certs" {}**: declara el volumen **jenkins_docker_certs** al que da el nombre jenkins_docker_certs.
  - Bloque **resource "docker_volume" "jenkins_data" {}**: declara el volumen **jenkins_data** al que da el nombre jenkins_data.
  - Bloque **resource "docker_container" "jenkins_docker" {}**: es el bloque que define el contenedor de docker que va a ejecutar las órdenes de jenkins. Definimos un nombre para el contenedor (**jenkins-docker**), la imagen que debe usar (**docker:dind**), se le asigna permisos elevados (**privileged = true**) y se indica que no se reinicie con **restart = "no"**. Además se le asigna la red anteriormente creada bajo el alias de **docker**, los volúmenes anteriormente creados, que se ligan con las correspondientes carpetas, se crea una variable llamada **env** que especifica en este caso el directorio donde va a almacenar docker los certificados TLS y se promociona el puerto **2376** para que el contenedor de jenkins pueda hablar con el de docker por ese puerto. Finalmente se define la variable **storage_opts** con la clave **"overlay2.override_kernel_check"** con el valor de **"1"** o true, habilitando una opción específica de overlay2, que es el controlador de almacenamiento.
  - Bloque **resource "docker_container" "jenkins_blueocean" {}**: es el bloque que define el contenedor de jenkins. Definimos un nombre para el contenedor (**jenkins-blueocean**), la imagen que debe usar (**myjenkins-blueocean**), que es la que creamos anteriormente mediante el Dockerfile y se le indica que solo se reinicie si falla con **restart = "on-failure"**. Al igual que con el contenedor de docker se le asigna la red común creada anteriormente, se declara una variable

env en la que definimos dónde encontrar al contenedor de docker, dónde almacenar los certificados y que use la capa de verificación TLS al interactuar con docker mediante **"DOCKER_TLS_VERIFY=1"**, se asignan los volúmenes con su ruta y, en el caso del volumen de los certificados, indicamos que sea solo de lectura con **read_only = true** y para terminar promocionamos el puerto **8080** para poder acceder a jenkins desde **localhost:8080**.

Podemos hacer que los contenedores se pongan a correr ejecutando en una terminal en el directorio en el que se encuentra el fichero **Despliegue.tf** la siguiente orden:

```
terraform init
terraform apply
```

que inicializan un directorio de trabajo terraform y aplican los cambios descritos en **Despliegue.tf**, creando los recursos pertinentes tras recibir una confirmación desde teclado.

Ahora debemos hacer la instalación de jenkins. Para ello vamos a usar cualquier navegador y vamos a entrar a la dirección **localhost:8080** para empezar la instalación.

Configuración de Jenkins

Cuando desplegamos los contenedores y accedemos a **localhost:8080** encontramos una la pantalla de jenkins, que nos solicita una contraseña de administrador. Para ver dicha contraseña vamos a ver los logs del contenedor de jenkins. Para ello vamos a ejecutar en la terminal la siguiente orden:

```
docker logs jenkins-blueocean
```

cuya salida incluye algo similar a lo siguiente:

```
(...)

*************************************************************

Jenkins initial setup is required. An admin user has been created and a password
generated.
Please use the following password to proceed to installation:

8ccada3616454f5094f4c7efd9249d7f

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************

(...)
```

De ahí, podemos copiar la contraseña e introducirla en el navegador con jenkins.

Tras meter la contraseña en el navegador y pulsar en siguiente vamos a elegir el tipo de instalación. En nuestro caso hemos escogido los plugins sugeridos y, tras esperar a que se instalen, podremos pasar a introducir el usuario administrador, modificar la URL de jenkins, si queremos, y finalmente podemos empezar a usar jenkins.

Creación del Pipeline

Una vez estamos en la pantalla principal de jenkins, para crear nuestro pipeline haremos click en la opción de **Nueva Tarea**, le damos el nombre de **Entregable3-VS** e indicamos que se trata de un **Pipeline** haciendo click en el recuadro pertinente.

Guardamos los cambios y pasamos a la siguiente pestaña en la que podemos añadir una descripción de nuestro pipeline y, en el apartado de **Pipeline > Definition** indicamos que queremos que tome el script de pipeline desde un **SCM**, hacemos click en **git** e incluímos el enlace a nuestro repositorio: GitHub e indicamos que queremos que use la rama **main** en vez de **master** para, finalmente, indicar que el fichero se encuentra en la ruta **docs/jenkinsfile**.

En nuestro caso, hemos usado el fichero Jenkinsfile proporcionado en el ejercicio ya que el que usaba el tutorial/demo de python no estaba configurado para usar **dind** como agente. El fichero **Jenkinsfile** en cuestión es el siguiente:

```
pipeline {
    agent none
    options {
        skipStagesAfterUnstable()
    }
    stages {
        stage('Build') {
            agent {
                docker {
                    image 'python:3.12.0-alpine3.18'
                }
            }
            steps {
                sh 'python -m py_compile sources/add2vals.py sources/calc.py'
                stash(name: 'compiled-results', includes: 'sources/*.py*')
            }
        }
        stage('Test') {
            agent {
                docker {
                    image 'qnib/pytest'
                }
            }
            steps {
                sh 'py.test --junit-xml test-reports/results.xml
sources/test_calc.py'
            }
            post {
                always {
                    junit 'test-reports/results.xml'
```

```
                }
            }
        }
        stage('Deliver') {
            agent any
            environment {
                VOLUME = '$(pwd)/sources:/src'
                IMAGE = 'cdrx/pyinstaller-linux:python2'
            }
            steps {
                dir(path: env.BUILD_ID) {
                    unstash(name: 'compiled-results')
                    sh "docker run --rm -v ${VOLUME} ${IMAGE} 'pyinstaller -F
add2vals.py'"
                }
            }
            post {
                success {
                    archiveArtifacts "${env.BUILD_ID}/sources/dist/add2vals"
                    sh "docker run --rm -v ${VOLUME} ${IMAGE} 'rm -rf build dist'"
                }
            }
        }
    }
}
```

A continuación una explicación en profundidad del fichero **Jenkinsfile** arriba descrito:

- Para empezar, el Jenkinsfile se compone de **tres etapas** junto con una **configuración general**:
    - **Configuración general**: es el primer bloque, en el que indicamos que no habrá un agente por defecto para todo el pipeline mediante **agent any** y qué, si alguna etapa marca el build como inestable, las etapas siguientes serán omitidas mediante el **options { skipStagesAfterUnstable() }**.
    - **Etapa Build**: durante el build se usará el agente de docker de **python:3.12.0-alpine3.18** y se definen dos pasos a realizar durante la etapa:
        - El primero de estos pasos, **sh 'python -m py_compile sources/add2vals.py sources/calc.py'**, se encarga de la compilación de los ficheros add2vals y calc.
        - El segundo, **stash(name: 'compiled-results', includes: 'sources/*.py')**, guarda esos archivos compilados para las etapas posteriores.
    - **Etapa Test**: en esta etapa indicamos que el agente docker a utilizar será **qnib/pytest** y se define un paso a realizar junto con una subetapa de post-ejecución con un paso más:
        - En pasos a realizar en la etapa de test tenemos la orden **sh 'py.test --junit-xml test-reports/results.xml sources/test_calc.py'**, que ejecuta las pruebas unitarias con pytest y genera un reporte en formato jUnit XML.
        - En la etapa de post ejecución se indica, mediante **always { junit 'test-reports/results.xml' }**, que siempre que se realicen los tests se publiquen los resultados

en formato de jUnit.

- **Etapa Deliver**: en esta última etapa no especificamos un agente específico de docker, pero sí hacemos algunas declaraciones sobre el entorno:

  - Definimos un volumen mediante **VOLUME = '$(pwd)/sources:/src'** y montamos la carpeta de sources en el direcotrio src del contenedor.
  - Indicamos, mediante **IMAGE = 'cdrx/pyinstaller-linux:python2'**, qué imagen vamos a utilizar para hacer el empaquetado del código.

  y cuenta además con un par de pasos (declarados en **steps {...}**) y un par de acciones a realizar tras la ejecución (declaradas en **post {...}**):

  - Para las pasos a realizar primero cambiamos al directorio especificado por env.BUILD.ID mediante **dir(path: env.BUILD_ID) {...}** y, una vez dentro se realizan dos acciones:
    - Primero, mediante **unstash(name: 'compiled-results')** recuperamos los archivos compilados que guardamos en la etapa de Build.
    - Segundo, y mediante **sh "docker run --rm -v ${VOLUME} ${IMAGE} 'pyinstaller -F add2vals.py'"**, ejecutamos pyinstaller dentro de un contenedor Docker, empaquetando el script add2vals en un ejecutable.
  - En la fase de post ejecución, en caso de éxito (**success {...}**), vamos a realizar dos acciones:
    - Primero, mediante **archiveArtifacts "${env.BUILD_ID}/sources/dist/add2vals"**, archiva el ejecutable generado.
    - Segundo, con **sh "docker run --rm -v ${VOLUME} ${IMAGE} 'rm -rf build dist'"**, ordenamos que se limpien los directorios de build y dist dentro del contenedor docker.

Ejecución del fichero Jenkinsfile

Para ejecutar el fichero de **Jenkinsfile**, si hemos seguido correctamente los pasos, bastará con hacer click en el botón **"Construir ahora"** y, tras esperar algunos instantes (algo más si es la primera vez, ya que tiene que descargar las imágenes docker) veremos por pantalla el resultado exitoso de la ejecución del pipeline descrito por el Jenkinsfile.

Ejecución del artefacto

Si el resultado del test ha sido exitoso, veremos que se ha generado el artefacto **add2vals**, que es el ejecutable del programa. Descargamos el ejecutable haciendo click sobre el y, desde una terminal, navegamos al directorio en el que se encuentra para hacer un

```
chmod u+x ./add2vals
```

con lo que le damos al usuario permisos para ejecutar el ejecutable add2vals. Para hacer la ejecución haremos un

```
./add2vals 5 8
```

y, si todo ha ido bien, tendremos por pantalla el resultado de realizar la suma de ambos valores.