ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

IMPLEMENTACIÓN DE CONTENEDORES ALOJADOS EN LA NUBE

IMPLEMENTACIÓN DE UN LENGUAJE DE PROGRAMACIÓN INTERPRETADO MEDIANTE CONTENEDORES ALOJADOS EN LA NUBE

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR EN REDES Y TELECOMUNICACIONES

ALEX ROBERTO RIVADENEIRA AGUALONGO

DIRECTOR: GABRIELA KATHERINE CEVALLOS SALAZAR

Quito, agosto 2022

CERTIFICACIONES

Yo, ALEX ROBERTO RIVADENEIRA AGUALONGO declaro que el trabajo de integración curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

ALEX ROBERTO RIVADENEIRA AGUALONGO

alex.rivadeneira@epn.edu.ec

alex_r1965@hotmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por ALEX ROBERTO RIVADENEIRA AGUALONGO, bajo mi supervisión.

GABRIELA KATHERINE CEVALLOS SALAZAR

DIRECTORA

gabriela.cevalloss@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

ALEX ROBERTO RIVADENEIRA AGUALONGO

DEDICATORIA

Este proyecto integrador lo dedico primeramente tanto a mis padres celestiales como a mis padres terrenales ya que fueron un pilar fundamental en mi formación personal y académica, así como al resto de mi entorno familiar como lo son mis dos hermanas y mi abuelita, entre todos ellos forjaron en mí un gran hombre lleno de valores y virtudes que me están llevando a finalizar de manera exitosa mi carrera profesional.

Alex Roberto Rivadeneira Agualongo

AGRADECIMIENTO

Principalmente le agradezco a Dios por brindarme el regalo de la vida y la sabiduría para lograr finalizar mis estudios de manera satisfactoria.

A mis padres Edgar y Miriam quienes fueron mi pilar fundamental para terminar este proceso de mi educación ya que siempre tuve su apoyo incondicional.

A mi abuelita María que, a pesar de ya no estar en este mundo supo enseñarme el valor del amor hacia los demás sin esperar nada a cambio.

A mis dos hermanas, que siempre están para mí cuando más lo necesito dándome apoyo moral y emocional.

A la Escuela Politécnica Nacional, que me brindó la oportunidad de formarme tanto de forma personal como profesional en esta prestigiosa casa de estudios.

Finalmente, agradezco a todos los profesores de mi facultad ESFOT por compartir conmigo sus conocimientos y experiencias, con el fin de añadir un aporte significativo en mi vida personal y profesional, así como agradecerle a la Ing. Gabriela Cevallos por su apoyo y dirección en el desarrollo de este trabajo de titulación.

Alex Roberto Rivadeneira Agualongo

ÍNDICE DE CONTENIDOS

RESUMEN

Este proyecto de titulación, “Implementación de un lenguaje de programación interpretado mediante contenedores alojados en la nube”, consta de dos contenedores alojados en la nube de Google, el primer contenedor tiene el editor “Vim” y el lenguaje de programación *Python* 3, el segundo contenedor posee tanto un “*Integrated Development Environment* (IDE)” llamado “RStudio” así como de su lenguaje base R. El propósito de la implementación de dichos contenedores cargados en la nube es brindar un entorno de programación preparado para desarrollar aplicaciones en *Python* 3 y un entorno para el análisis de datos en R.

En la primera sección se describe de forma general la implementación de los contenedores alojados en la nube. Además, se presentan los objetivos, alcance y marco teórico. En el marco teórico se menciona una pequeña introducción a los contenedores, las diferencias entre contenedores y máquinas virtuales y el análisis de tres lenguajes de programación interpretados más representativos en la industria de desarrollo.

La segunda sección consta de la metodología usada para el desarrollo de este proyecto, con el fin de dar cumplimiento al objetivo general del mismo.

En la tercera sección, se detallan los pasos utilizados para cumplir con los objetivos del proyecto, inicialmente se investigó todo lo referente al funcionamiento de los contenedores, se comparó varias herramientas DevOps. Además, se comparó varios proveedores de nube que poseen plataformas para alojar contenedores. Finalmente se creó, ejecutó y verificó el funcionamiento de los contenedores dentro del proveedor de nube seleccionado.

En las dos últimas secciones se presentan conclusiones de los resultados obtenidos y recomendaciones basadas en la experiencia de lo realizado. Por último, se anexa el certificado de originalidad, enlaces de videos demostrativos y códigos fuente.

PALABRAS CLAVE: *Docker*, Docker Hub, Contenedor, *Python*, Lenguaje R.

ABSTRACT

The Project "Implementation of a programming language interpreted by cloud-hosted containers", consists in two containers hosted in the google cloud, the first one has the editor VIM and the programming language Python 3, the second one has both an Integrated Development Environment (IDE) "RStudio" and its R-based language. The purpose of this implementation is to give a programming environment prepared to develop applications in Python 3 or R.

In the first section is described in general the implementation of the cloud-hosted containers. In addition are presented the objectives, scope, and theoretical framework. A little introduction is given about the cloud-hosted containers, the differences between containers and virtual machines and the analysis of the most interpreted programming languages in the development industry.

In the second section is explained the methodology used for the development of this project, to accomplish the general objective of the project.

In the third section, the steps are detailed to accomplish de project objectives, at the beginning was investigated all the referent to the operation of the containers, many DevOps tools were compared. Then, many cloud suppliers that possess platforms to host containers were compared. Was created, executed, and verified the operation of the containers inside the cloud supplier chosen.

The last two sections present conclusions, results and recommendations obtained, based on the experience of what has been done. At the end, is annexed the certificate of authenticity, links of demonstrative videos and source codes.

KEYWORDS: *Docker, Docker Hub, Container, Python, R Language.*

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

El presente proyecto conlleva una forma diferente de contextualizar las redes de información y la implementación de servicios, procesos y aplicativos. Con el fin de realizar una automatización de microservicios, procesos de *software*, servidores y aplicaciones, se ejecutará el despliegue de un lenguaje de programación interpretado mediante un contenedor alojado en la nube.

Para implementar los contenedores se procederá a analizar algunas herramientas de DevOps con el fin de seleccionar a la más adecuada para ejecutar el presente proyecto. Además, para disminuir los costos en cuanto al *hardware* requerido y con el fin de modernizar la implementación de contenedores se pretende alojar en la nube a los mismos, donde se analizará entre las opciones de nube sus características, ventajas y desventajas seleccionando el proveedor apropiado para este fin.

1.1 Objetivo general

Implementar diferentes servicios mediante contenedores alojados en la nube.

1.2 Objetivos específicos

- Analizar algunas herramientas de DevOps para realizar contenedores.
- Analizar diferentes proveedores de nube para alojar contenedores.
- Desarrollar los contenedores y alojarlos en la nube seleccionada.
- Realizar pruebas de funcionamiento y verificar los resultados finales.

1.3 Alcance

El presente proyecto consiste en analizar algunas herramientas DevOps que permitan realizar contenedores, se seleccionará a la más adecuada con el fin de implementar un contenedor con un lenguaje de programación interpretado. Además, se procederá a seleccionar de diferentes proveedores de nube al más idóneo para alojar al contenedor. Finalmente se realizarán pruebas de funcionamiento y verificación de los resultados obtenidos.

1.4 Marco Teórico

Introducción a contenedores

La construcción de contenedores es una técnica antigua que se vienen aplicando en los sistemas operativos Linux desde los años 80; Sin embargo, no fue una tecnología muy utilizada en esa época. Esta tecnología de la contenedorización se volvió popular en el año 2013, debido al nuevo formato de contenedor que publicó *Docker* [1].

Un contenedor es un paquete de *software* que resulta de la unión de librerías, dependencias y el código de la aplicación, que se encapsulan para ejecutarse en cualquier entorno, tales como Linux o *Windows* y su funcionamiento será el mismo, sea que se encuentre en la nube o en una máquina local. Esto se logra gracias a que los contenedores utilizan un modo de virtualización del sistema operativo (SO) utilizado para separar procesos, administrar memoria y almacenamiento de disco a la que acceden dichos procesos [2].

Los contenedores usan una parte del almacenamiento del núcleo del SO, así como también la red del equipo anfitrión en donde se ejecutarán. Son considerados como elementos mínimos necesarios para que una aplicación pueda ser ejecutada como un proceso. En un SO distribuido los contenedores separan las aplicaciones entre ellas. Estas aplicaciones se ejecutan inicialmente en un contenedor anfitrión y este se ejecuta en un SO. Finalmente, los contenedores brindan en el periodo de vida de la aplicación las siguientes utilidades: escalabilidad, portabilidad, aislamiento y rapidez [3].

Máquinas Virtuales y Contenedores

Una máquina virtual (VM) es una técnica para desarrollar un ambiente virtualizado. Las VM se empezaron a usar desde la década de los 70 y se consideran el cimiento de la informática en la nube. Una computadora física permite ejecutar varias máquinas virtuales con diferentes sistemas operativos de cliente o servidor asignando discos, memoria y procesador a cada una de ellas, utilizando un *software* que les permite ejecutarse denominado hipervisor [4].

Las VM son más rápidas y portables que los servidores físicos, pero necesitan el mismo tiempo de inicio y administración [1]. La Tabla 1.1 describe algunas diferencias entre contenedores y máquinas virtuales.

Tabla 1.1 Diferencias entre contenedores y máquinas virtuales [1]

Contenedores	Máquinas Virtuales
Livianos y con un tiempo de arranque rápido, ya que inicializa solo con el núcleo del SO.	Pesados y con un tiempo de ejecución lento, debido a que ejecutan todo el SO.
Portables; es decir, se pueden distribuir entre varios equipos y tienen el mismo comportamiento desde cualquier lugar donde se corra el contenedor.	No son portables, debido a que estos requieren necesariamente de un Hipervisor para su funcionamiento.
Permiten el desacoplamiento y la ejecución de los componentes de una aplicación, posibilitando el trabajo en una arquitectura construida en microservicios.	No permiten desacoplar los componentes de una determinada aplicación, debido a que ejecuta toda la aplicación en una sola VM.
No se crearon para ejecutar aplicaciones que necesiten estado, por ejemplo, el correo electrónico tiene varios estados como son: enviado, rechazado, en espera.	Se pueden ejecutar aplicaciones que necesiten estado.
Virtualizan lo estrictamente necesario para la ejecución de una aplicación.	Virtualizan tanto el SO, sus redes y almacenamiento; es decir, el servidor en su totalidad.

Después de analizar la comparación de las características entre contenedores y máquinas virtuales, se puede decir que el uso de contenedores es sumamente importante para el desarrollo de microservicios. Debido a que son ligeros y rápidos, ya que no llevan consigo el peso del sistema operativo en su totalidad. Además, permiten la optimización de recursos de un servidor, debido a que la cantidad de contenedores que se pueden alojar en un servidor es mayor a la cantidad de máquinas virtuales.

En cuanto al uso de los contenedores en la nube se puede decir que la mayoría de los proveedores de nube facilitan la ejecución de contenedores de manera directa en sus servidores sin la intervención de una VM para este fin, además los contenedores son portables dentro de la nube; es decir, una vez creado el contenedor es posible ejecutarlo en diversos servidores [5].

Lenguajes de Programación interpretados

Un lenguaje de programación interpretado es un tipo de lenguaje que utiliza un intérprete que se encarga de ejecutar de línea en línea la aplicación desarrollada, a continuación, se procede a analizar los más representativos [6].

Python

Python es un lenguaje interpretado y compilado, desarrollado por el holandés Guido van Rossum a fines de los años 80. Tiene instrucciones similares al idioma inglés por lo que su código es muy legible.

Entre los usos que se le puede dar a *Python* por ser de propósito general se tiene: creación de aplicaciones multimedia, herramientas de DevOps, campos de la educación, implementación de IoT, desarrollo de aplicaciones Web y móviles. Entre sus características más representativas se tiene:

- Tipado dinámico; es decir, asigna automáticamente el tipo a una variable el momento de ejecución según el valor asignado, si se asignara otro valor con diferente tipo la variable cambiaria de tipo.
- Multiplataforma, debido a que se lo puede encontrar en varios SO, por ejemplo: Linux, *Windows* o Mac.
- Orientado a objetos, que es una forma de programar las aplicaciones y se enfoca a utilizar clases y objetos relacionados con el mundo real [7].
- Lenguaje interpretado de alto nivel; es decir, no se necesita conocer el manejo de memoria, la forma o la organización del código [8].

Como ventajas *Python* se lo puede obtener de forma gratuita, fácil de usar y entender, compatible con varias plataformas, tiene una gran cantidad de librerías desarrolladas por la comunidad y se puede extender con C++, C.

Algunas desventajas de utilizar *Python* son que requiere una gran cantidad de memoria, pueden existir errores al momento de la ejecución, la versión 2 y la versión 3 no son compatibles y tiene algunas librerías incompletas [9].

Lenguaje R

Es un entorno para análisis y gráficos estadísticos. Brinda una gran cantidad de métodos estadísticos entre estos se tiene el no lineal, lineal, pruebas, series temporales, distribución, unión y métodos gráficos. Una de las características más importante es su manera sencilla de construir redes elaboradas con una alta difusión, que contienen símbolos y fórmulas matemáticas en el sitio que se requiera.

R es un *software* libre bajo lineamientos de la Licencia Pública General (GLP). Es interpretado y ejecutado en una gran cantidad de plataformas, tales como: *Windows*, Mac y Linux [10]. Entre sus características más representativas se tiene:

- Utilización y acopio de datos.
- Tiene operadores para utilizar en arreglos y matrices.
- Existen muchas herramientas creadas para analizar datos.
- Se puede imprimir o visualizar los gráficos que tiene R, para realizar un análisis de datos.
- Lenguaje práctico y simple que tiene sentencias de condición, bucles, funciones creadas por el usuario que pueden ser recursivas y funciones de ingreso/salida de datos [10].

Java

Lenguaje multiplataforma de propósito general, con orientación a objetos, veloz, confiable y seguro. Java fue creado por "*Sun Microsystems"* y lanzado en el año 1995, desde ahí es uno de los que más se utiliza en la industria de desarrollo, debido a que se puede ejecutar en varias plataformas con algunos sistemas operativos, tales como: *Windows*, Linux y Mac.

Es un lenguaje que no depende de la plataforma en donde se ejecutará; es decir, se construye el código de la aplicación una única vez pudiendo después ejecutarse en otro ordenador con otro sistema operativo. Para lograr esta portabilidad, Java requiere un ambiente de ejecución llamado "Java *Runtime Environment* (JRE)". Java está en todas partes desde ordenadores, dispositivos móviles, IoT hasta centro de datos; principalmente es el más usado para crear aplicaciones de tipo cliente-servidor [11]. Entre sus características más representativas se tiene:

- Lenguaje robusto y orientado a objetos.
- Sencillo de aprender.
- Es usado para el desarrollo de aplicaciones académicas y empresariales.
- Lenguaje seguro.
- Portátil de alto rendimiento.
- Multihilo, dinámico y distribuido.
- Lenguaje interpretado y compilado a la vez [12].

Después de realizar el estudio y el análisis de los tres lenguajes de programación interpretados más representativos en la industria de desarrollo como son: *Python*, R y Java, se procede a elegir tanto el lenguaje *Python* como R para el desarrollo del presente trabajo integrador. Debido a que han tendido gran acogida por su facilidad de uso y su amplia documentación. Además, entre los usos más importantes de estos lenguajes en la actualidad están las herramientas DevOps, microservicios, *BigData*, inteligencia artificial y análisis de datos.

2 METODOLOGÍA

La metodología que fue usada para esta investigación es la experimental, puesto que se analizó y comparó tanto las herramientas de DevOps como los proveedores de nube para la creación y ejecución de los contenedores. Además, se realizaron pruebas de funcionamiento de las imágenes de lenguajes de programación interpretados para posteriormente construir las nuevas imágenes de contenedores. Finalmente, fueron subidas y ejecutadas en la nube con lo que se cumplieron los objetivos establecidos. Los pasos de esta metodología se evidencian en la Figura 2.1.

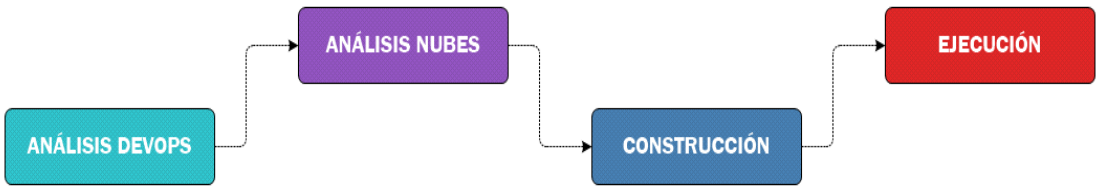

Figura 2.1 Metodología del trabajo integrador

El primer paso de análisis DevOps, permitió buscar la herramienta para crear contenedores, por lo tanto, se realizó el análisis y comparación de tres herramientas existentes en el mercado, tales como: "*Docker*", "Linux *Containers* (LXC)" y "Podman". De las cuales se eligió "*Docker*", por ser la herramienta más usada en el mercado de desarrollo de contenedores.

El segundo paso de análisis de nubes, se realiza la elección de un proveedor de nube que cumpla con las condiciones tanto funcionales como económicas para la implementación de los contendores. Para lo cual, se comparó y analizó tres proveedores de nube, tales como: "*Google Cloud Platform* (GCP)", "*Amazon Web Services* (AWS)" y "Microsoft Azure". De estas se seleccionó a GCP para que aloje a los contenedores de dos lenguajes de programación interpretados *Python* y R.

Para el tercer paso se construyeron dos proyectos en la nube GCP, en el primer proyecto se construyó y se subió la imagen del contendor de *Python* en la nube mediante un archivo “Dockerfile”; para el segundo proyecto se descargó y se subió la imagen del contenedor de “RStudio”.

En el cuarto y último paso de ejecución y comprobación de resultados, se demuestra el funcionamiento adecuado de ambos contenedores en la nube, con la ayuda de sencillos programas desarrollados en los lenguajes de *Python* y R respectivamente.

3 RESULTADOS

El presente proyecto le proporciona al programador un entorno de desarrollo para crear aplicaciones desde la nube sin tener que preocuparse de la instalación del lenguaje y de sus editores. En el caso del contenedor del lenguaje *Python* está instalado en la versión 3 y se le agregó el editor “Vim”, en el cual el programador podrá editar el código de su aplicación. Para el contenedor del lenguaje R está instalado en la versión 4.2.1 juntamente con el IDE “RStudio”, mismo que posee funciones que facultan ampliar la capacidad del análisis de datos.

El desarrollo de este proyecto se basó entorno a cuatro fases esenciales: análisis y comparación de herramientas DevOps para realizar contenedores y de proveedores de nube, construcción de dos contenedores con los lenguajes interpretados de *Python* y R basados en imágenes obtenidas de “Docker Hub”. Finalmente, la ejecución y comprobación del funcionamiento de los contendores en la nube de GCP.

3.1 Análisis de herramientas DevOps para realizar contenedores

La tecnología DevOps agrupa tanto a los desarrolladores (Dev) como a las operaciones (Ops), cuyo objetivo es la integración de estos dos equipos compartiendo la cooperación y el compromiso de optimizar el tiempo de respuesta a los clientes, logrando los objetivos propuestos de manera rápida y añadiendo certeza en las aplicaciones creadas [13]. A continuación, se procede a analizar las herramientas DevOps más representativas.

Docker

Es un formato de contenedor de *software* libre que admite el despliegue de aplicaciones para su ejecución y envió. *Docker* permite empacar una aplicación y ejecutarla en un medio separado conocido como contenedor. Con *Docker* se puede desunir la aplicación

de su infraestructura con el fin de que sea más rápida la prueba, entrega y puesta en producción de la aplicación [14].

Docker creó un archivo con un modelo descriptivo llamado “Dockerfile”, con el cual se crean los contenedores y se los puede mantener de forma fácil y simple. Cada línea de un archivo “Dockerfile” crea una capa que versiona los cambios realizados en la imagen (plantilla con instrucciones que permiten crear un contenedor) del contenedor [15].

LXC

Es un *software* libre con una interfaz para administrar las funciones de control del núcleo de Linux, mediante una “API” y varias herramientas que permiten a los clientes de Linux instaurar y gestionar fácilmente contenedores. El propósito de LXC es desarrollar un ambiente similar a una instalación modelo de Linux, pero sin separar el *kernel* [16].

Podman

Es un mecanismo originario de Linux, de *software* libre que permite la creación e implementación de aplicaciones usando contenedores, brinda una interfaz tipo “*Comand Line Interface* (CLI)”. Además, cuenta con un cliente remoto que se comunica a su “API-REST” para administrar los contenedores. Podman utiliza la librería libpod para manejar todo el ecosistema de los contendores [17].

En la Tabla 3.1 se compara algunas características entre *Docker*, LXC y Podman.

Tabla 3.1 Comparación entre *Docker*, LXC y Podman [15] [16] [18][1]

Característica	***Docker***	**LXC**	**Podman**
Facilidad de uso en la nube	Simple, debido a que no es necesario tener conocimientos en Linux. Y se encuentra instalado y listo para usar en varios proveedores de nube	Complejo, usa las funciones del núcleo de Linux para crear entornos virtuales. Para utilizarlo en la nube hay que instalarlo y configurarlo.	Posee comandos similares a los de *Docker*. Para utilizarlo en la nube hay que instalarlo y configurarlo.
Uso del equipo anfitrión	Mínimo, ya que tiene una capa llamada *Docker Engine* que lo hace portable.	Mayor dependencia, ya que comparte el núcleo del equipo anfitrión.	Mínimo, ya que tiene una capa llamada Podman *Engine.*

[1] Continúa en la siguiente página

Característica	***Docker***	**LXC**	**Podman**
Velocidad de arranque	Rápido, porque solo requiere del *kernel* del SO.	Rápido, porque no requiere empacar todo el SO.	Más rápido, porque no ejecuta el servicio para gestionar contenedores
Seguridad	Vulnerable, debido a que se ejecuta como administrador; sin embargo, cuenta con herramientas externas que lo hacen más seguro.	Seguro, cuenta con políticas y perfiles administrables de seguridad.	Seguro, debido a que cuenta con un registro de usuarios.
Enfoque de contenedorización	Aplicaciones	Sistema operativo	Aplicaciones
Documentación	Tiene la documentación más extensa y mejor explicada disponible en Internet	Tiene poca documentación y muy técnica.	Tiene poca documentación disponible en Internet
Repositorio de imágenes	Cuenta con su propio repositorio comunitario más grande del mundo para imágenes de contenedores.	Usa el repositorio de imágenes de *Docker*	Usa el repositorio de imágenes de *Docker*
Ejecución	Utiliza el comando **docker run** que realiza una búsqueda local de la imagen, caso contrario la descarga del repositorio configurado, crea un contenedor editable y lo inicia.	Usa el comando **lxc-start** para ejecutar el contenedor previamente creado con el comando **lxc-create.**	Tiene el comando **podman run** similar a **docker run**, pero con la integración de systemd.

En base al análisis de las tres herramientas de DevOps más representativas en el mercado para realizar contenedores, se determinó el uso de la herramienta *Docker* para la realización de los contenedores desarrollados en este proyecto. *Docker* es la herramienta más usada por los proveedores de nube para la implementación de contenedores, además cuenta con su propio repositorio digital “Docker Hub”, siendo este de uso general tanto para LXC como para Podman debido a que estos carecen de

dicho servicio. *Docker*, LXC y Podman son herramientas útiles para contenedores; sin embargo, LXC y Podman están enfocados a trabajar con VM ya que son sistemas originarios de Linux, es por esto que *Docker* es la herramienta de DevOps más usada para trabajar con contenedores en la nube.

3.2 Análisis de proveedores de nube para alojar contenedores

El concepto de nube permite responder activamente a la carga del sistema para repartir los servicios de acuerdo con el uso que se le da en la actualidad, lo que disminuye el costo de propiedad y mejora el aprovechamiento de los recursos. Las nubes se han convertido en el paradigma actual de la utilidad informática, debido a que el rendimiento de las aplicaciones es proporcional al número de recursos que brinda la nube. Por lo tanto, con frecuencia las nubes pueden indirectamente incrementar el rendimiento de las aplicaciones [19]. A continuación, se procede a analizar los proveedores de nube más representativos.

GCP

Esta plataforma fue comercializada en el año 2008, fue programada en *Python*, Java y C++. Esta nube cuenta con varios servicios, tales como: "*Google Container Registry"*, "*Google Cloud Storage"*, "*Google App Engine"*, entre otros [20]. "*Container Registry"* es utilizado para subir imágenes de *Docker* a la nube de Google [21].

La ventaja de utilizar GCP es que el soporte para trabajar en contenedores es notable; además, es flexible para incorporar servicios de otros proveedores de nube. Mientras que la desventaja es que cuenta con menos servicios que AWS y Azure; por otro lado, no cuenta con un buen soporte para resolver problemas de empresas [22].

AWS

La plataforma de Amazon fue creada en el año 2006, cuenta con más de 175 servicios en la nube, tales como: "*Amazon Storage Service", "AWS App Runner*", *"Amazon Elastic Container Registry",* entre otros. "*Elastic Container Registry"* es usado para almacenar y administrar imágenes de contenedores [23].

La ventaja de usar AWS es que tiene variedad y una alta disponibilidad en sus servicios; además, es considerado el proveedor de nube más confiable y seguro. Por el contrario, la desventaja de usar AWS son sus limitaciones cuando se trata de implementar

funcionalidades en varias nubes; por otra parte, es necesario adquirir el soporte de desarrollo empresarial [22].

Microsoft Azure

Es una plataforma de servicios en la nube públicos y privados que fue lanzada en el año 2008, cuenta con varios servicios, tales como: "*Azure App Service*", "*Azure Disk Storage*", "*Azure Container Registry*", entre otros. "*Azure Container Registry*" sirve para registrar y administrar imágenes de contenedores [24].

La ventaja de usar Azure es que tiene un buen soporte para implementar funcionalidades en diferentes nubes; además, cuenta con una sencilla migración e incorporación de todos los servicios de Microsoft que existan en una empresa. En cambio, la desventaja de usar Azure es que está principalmente enfocado a empresas y tiene menos servicios que AWS [22].

En la Tabla 3.2 se compara las principales características entre GCP, AWS y Azure.

Tabla 3.2 Comparación entre GCP, AWS y Azure [25] [26] [27][2]

Característica	**GCP**	**AWS**	**Azure**
Número de regiones donde existe el servicio	23	78	54
Servicios de contenedores	Ofrece 10 servicios entre los más representativos: 1) "*Container Registry*" 2) "*Cloud Run*" 3) "*Cloud Build*"	Ofrece 11 servicios entre los más representativos: 1) "*Amazon Elastic Container Registry*" 2) "*AWS App Container*"	Ofrece 8 servicios entre los más representativos: 1) "*Container Registry*" 2) "*Azure Container Apps*"
Tiempo de Gratuidad	Crédito de 300$ por 90 días gratis y sin ningún compromiso de tiempo de uso.	Uso gratuito por 1 año con un compromiso de tiempo de uso de 1 a 3 años.	Crédito de 200$ por 30 días gratis y con un compromiso de tiempo de uso de 1 a 3 años.

[2] Continúa en la siguiente página

Característica	GCP	AWS	Azure
Lenguajes de Programación Compatibles	Java, *Python*, Ruby, Go, Node.js, .NET, R.	Java, PHP, *Python*, Ruby, JavaScript, Go, TypeScript, CoffeeScript.	Java, *Python*, JavaScript, .NET, Go.
Bases de datos soportadas	Cloud Spanner, Cloud SQL, Cloud Datastore	Neptune, DynamoDB, RDS, Aurora, Redshift.	Cosmos DB, PostgreSQL, MySQL, SQL, Redis Cache.

En base al análisis de los tres proveedores de nube más representativos en el mercado, se determinó la utilización de la nube de GCP para la implementación y el alojamiento de los contenedores creados en este proyecto. Dicha nube brinda sólidos servicios y soporte para el trabajo con contenedores, además tiene una estrecha relación con la herramienta de DevOps escogida para la creación de los contenedores, *Docker*. Finalmente, y una de las más importantes referencias para trabajar con dicha nube es el tiempo de gratuidad que ofrece el proveedor de GCP, con respecto a los otros proveedores y el no condicionamiento del tiempo de uso del servicio después de la prueba gratuita.

3.3 Implementación de contenedores y almacenamiento en la nube de GCP

Inicialmente para implementar los contenedores tanto de *Python* como de R fue necesario registrarse en el repositorio digital de *Docker* llamado “Docker Hub”, como se presenta en la Figura 3.1. Esto es debido a que en dicha plataforma se buscarán las imágenes para la construcción de los contenedores. Este repositorio permite subir o descargar imágenes de contenedores, según la necesidad del usuario.

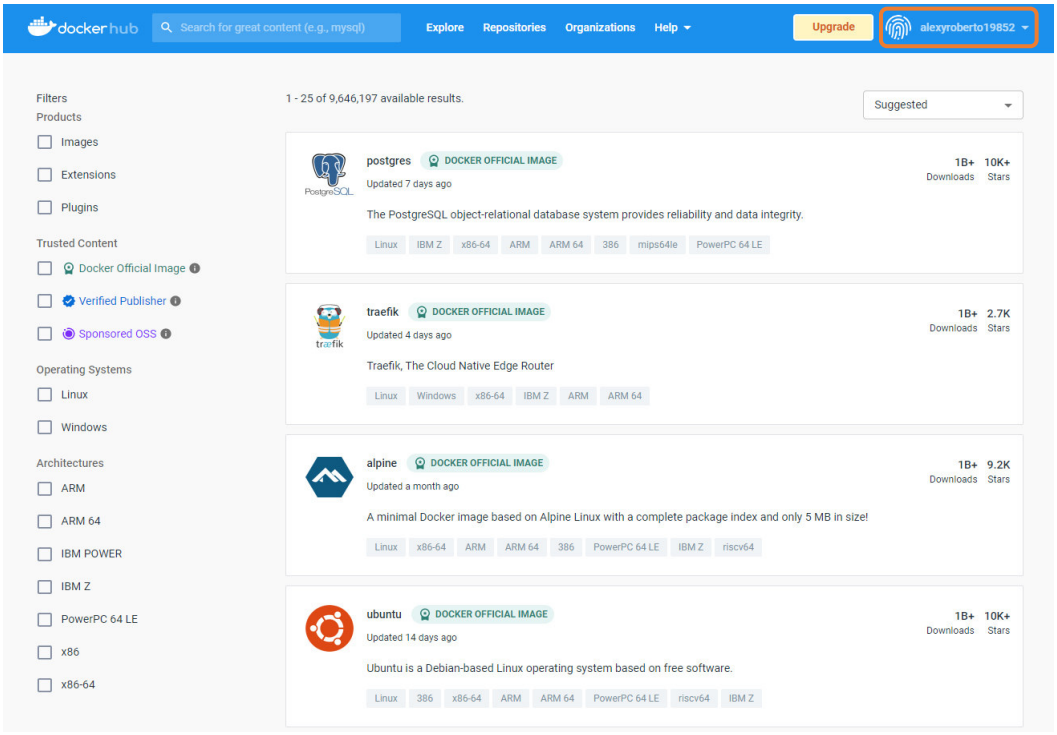

Figura 3.1 Registro en “Docker Hub”

Después de tener una cuenta en el repositorio de “Docker Hub”, también se creó una cuenta en el proveedor de nube GCP, para alojar los contenedores de *Python* y R, revisar la Figura 3.2. Para esto se llenó todos los datos solicitados por dicho proveedor, además fue necesario ingresar una tarjeta de débito para el registro, que pudo ser también de crédito; sin embargo, no existió ningún costo por el lapso de 90 días que duró la prueba gratuita.

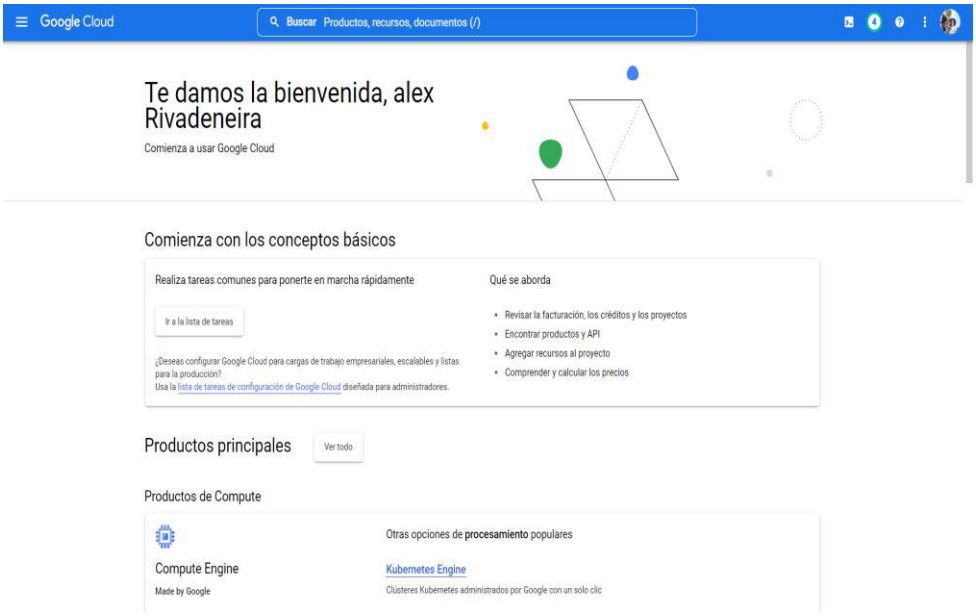

Figura 3.2 Registro en GCP

Implementación del contenedor de *Python*

Una vez completados los pasos anteriores, se implementó el contenedor de *Python*, para lo cual se creó un nuevo proyecto dando clic en la opción “Selecciona un proyecto”, después de esto se desplegó una ventana, donde se seleccionó en el botón “PROYECTO NUEVO”, esto se aprecia en la Figura 3.3. En la ventana proyecto nuevo se ingresa el nombre del proyecto llamado “*Python*” y se selecciona crear; observar la Figura 3.4.

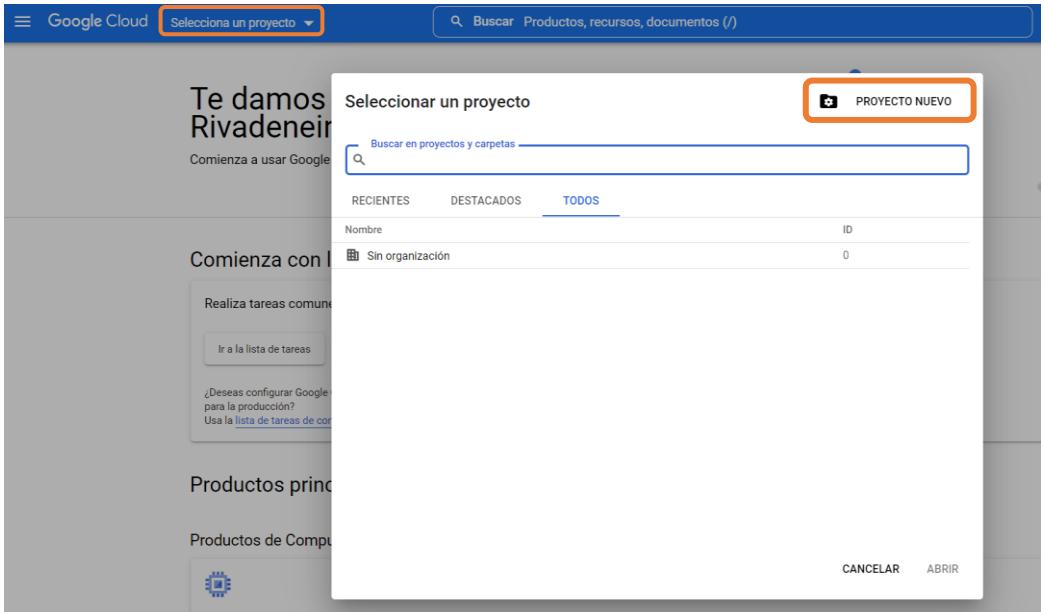

Figura 3.3 Nuevo proyecto para *Python*

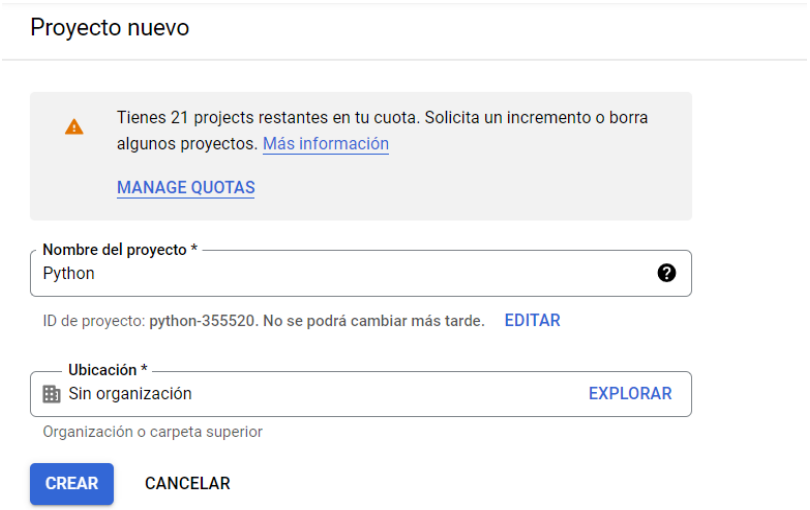

Figura 3.4 Proyecto con el nombre *Python*

Una vez que se ha creado y seleccionado el proyecto, se observa información importante acerca del mismo como su ID y nombre; además de la facturación del servicio; sin embargo, este último dato no se encuentra habilitado debido a que es una prueba gratuita.

Para registrar la imagen del contenedor de *Python* se utilizó la opción imágenes del servicio "*Container Registry*" que está dentro del menú de navegación de "*Google Cloud*", lo cual se indica en la Figura 3.5, en donde primero se habilitó la API de registro de contenedores de Google. Esto se visualiza en la Figura 3.6.

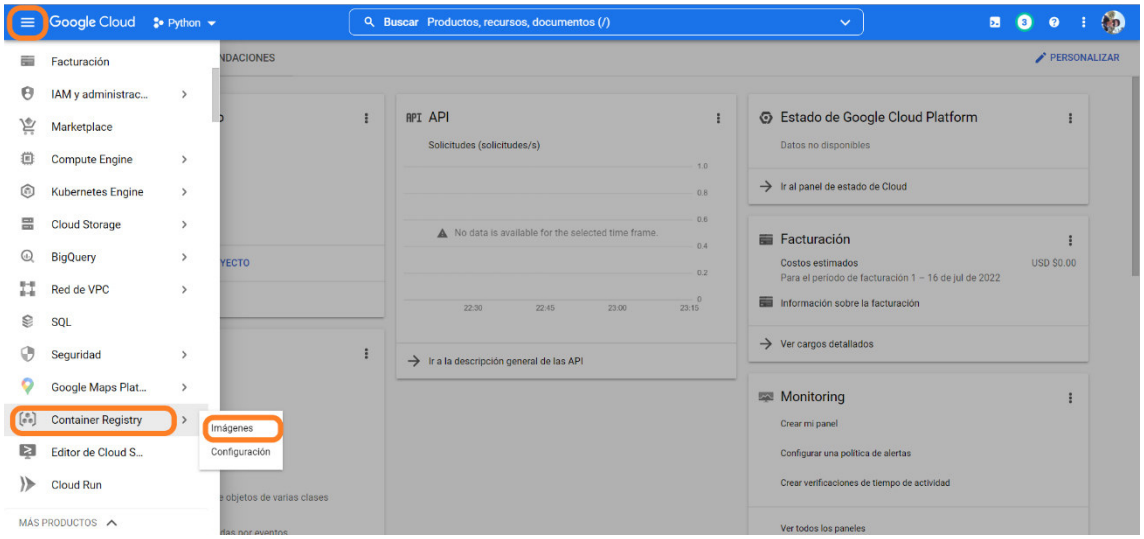

Figura 3.5 Despliegue de servicios de GCP

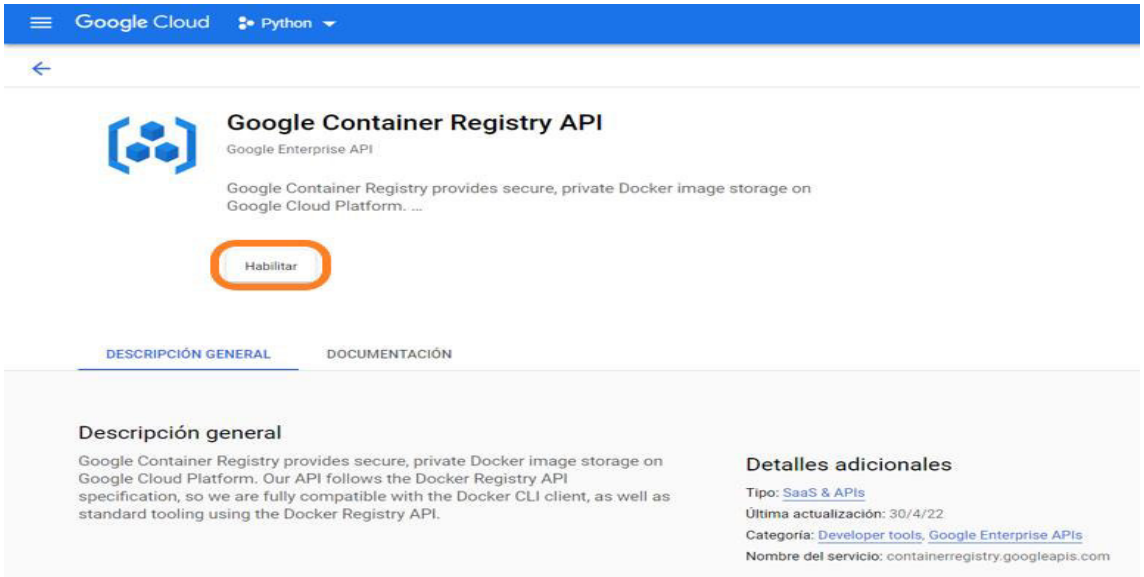

Figura 3.6 Habilitación de la API de Google

En la Figura 3.7 se ve el despliegue de la interfaz de repositorios donde se listan las imágenes de los contenedores registrados en la nube.

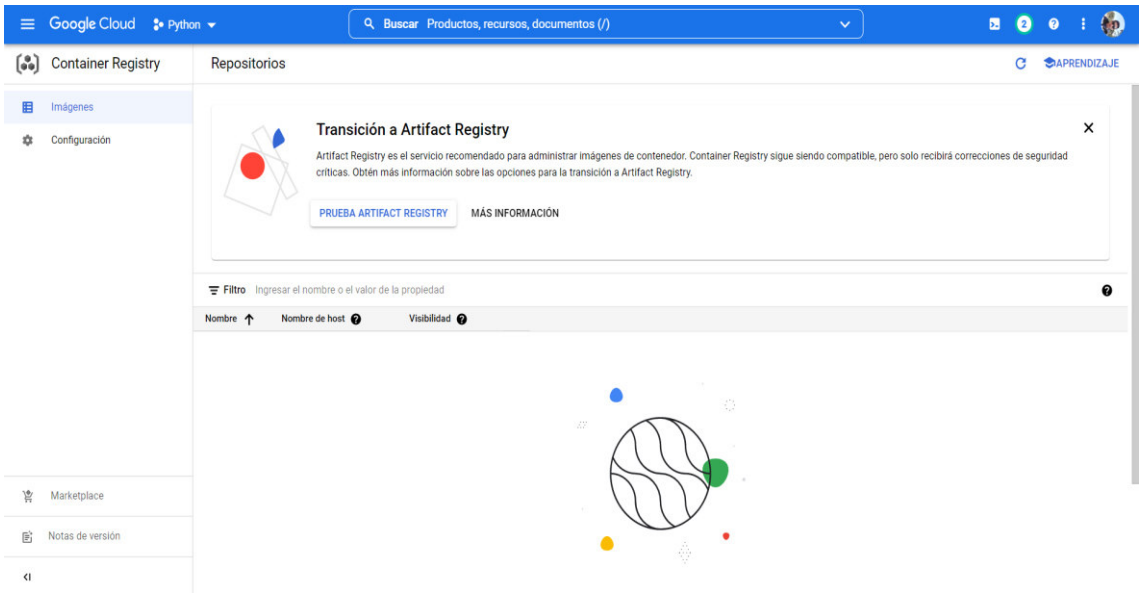

Figura 3.7 Registro de imágenes

Después es importante activar el intérprete de comandos seleccionando el botón "Activar *Cloud Shell*", que está en el menú superior, para posteriormente abrir el editor de *Cloud Shell* seleccionando la opción "Abrir editor", la Figura 3.8 muestra lo enunciado.

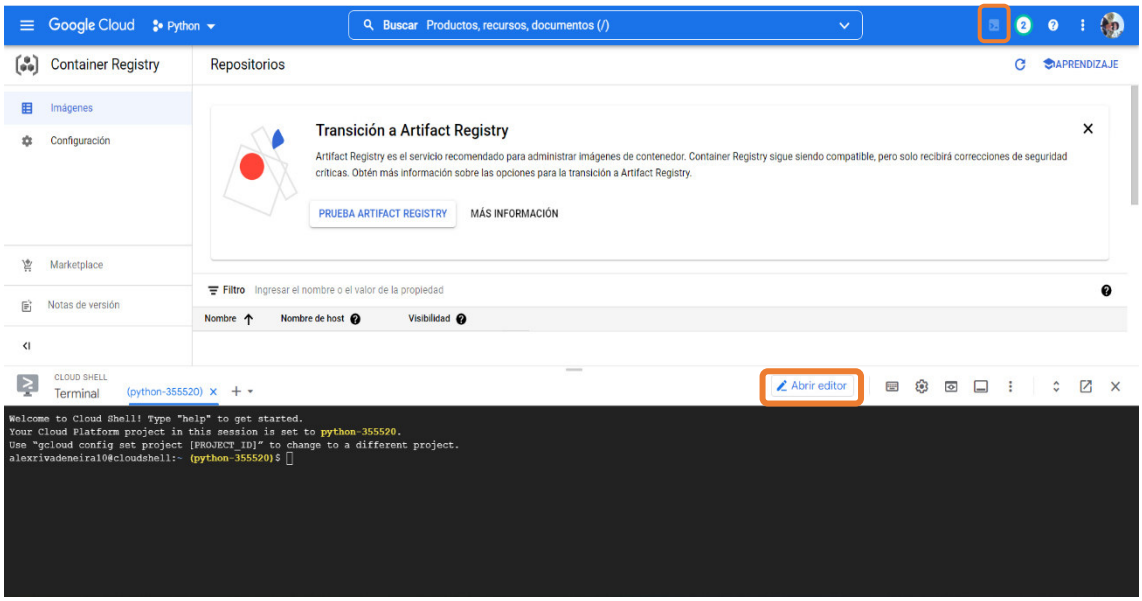

Figura 3.8 Activación de *Cloud Shell*

La Figura 3.9 presenta la creación de un archivo nuevo dentro del editor de *Cloud Shell*.

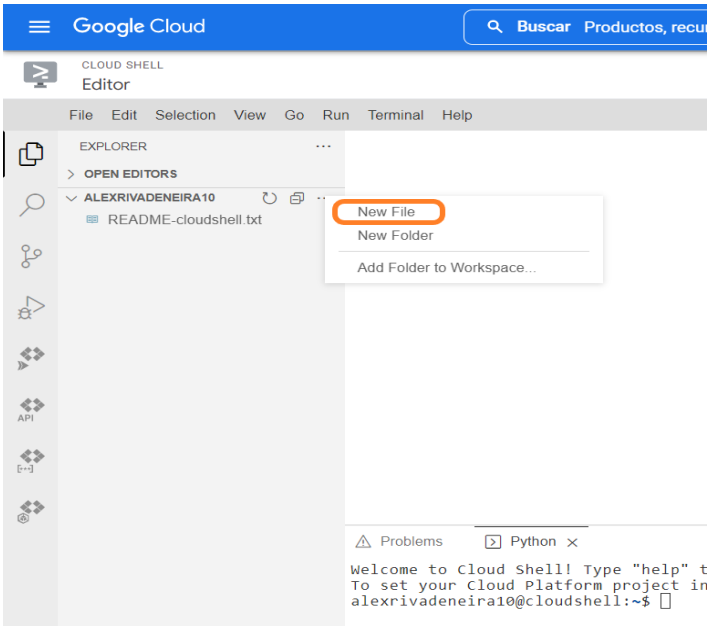

Figura 3.9 Creación de Archivo nuevo en *Cloud Shell*

Después se escribió las instrucciones del archivo que se visualiza en la Figura 3.10 y se las describe a continuación:

- FROM: Descarga la imagen de *Python* con la versión 3.9 de Slim búster obtenida del repositorio de “Docker Hub”, ver Figura 3.11.
- WORKDIR: Crea un directorio de trabajo app dentro de los directorios /usr/src, si no los encuentra, los crea.
- RUN: Ejecuta los comandos detallados a continuación, “apt-get update” para actualizar los repositorios del servidor y “apt-get install -y vim” instala el editor vim, el -y confirma la pregunta del desea continuar.
- COPY: copia todo lo que tiene dentro de la carpeta app; es decir, el ejemplo.py al contenedor.
- CMD: Ejecuta el ejemplo.py en el lenguaje *Python*.

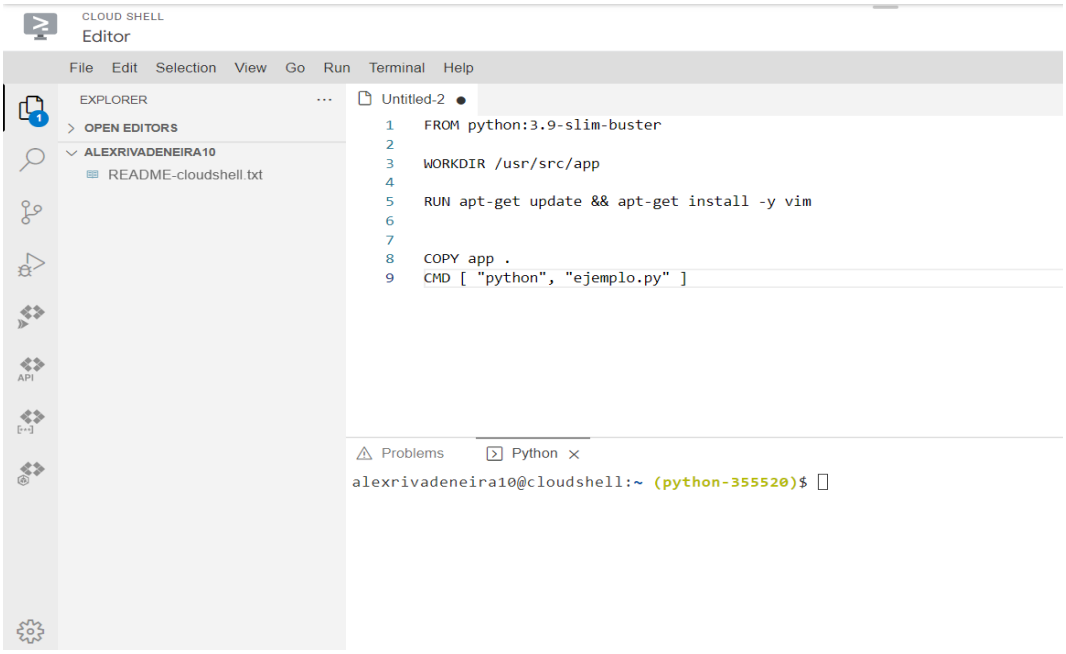

Figura 3.10 Contenido del archivo nuevo en *Cloud Shell*

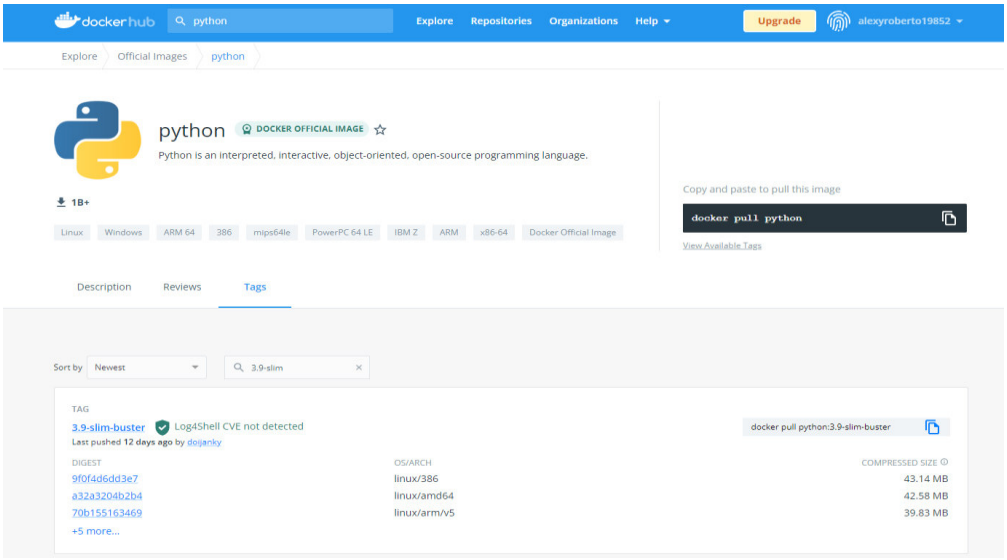

Figura 3.11 Imagen de la plantilla oficial de *Python*

La Figura 3.12 muestra el momento que se guardó este archivo presionando las teclas Ctrl + S con el nombre de “Dockerfile” y se presionó el botón *Save*.

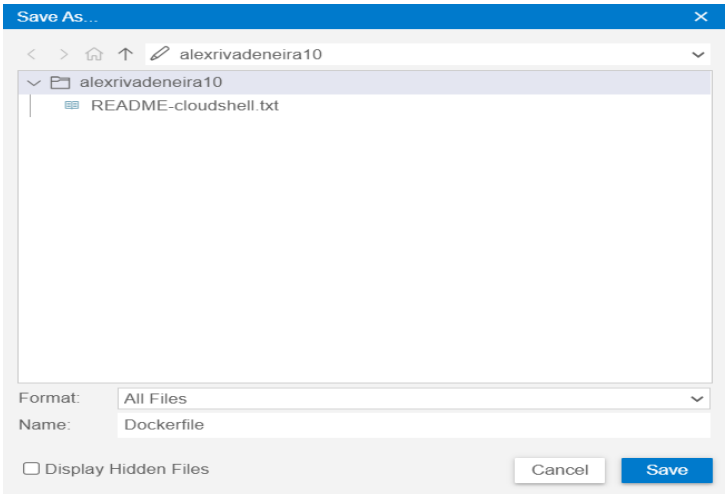

Figura 3.12 Nombrar archivo

Una vez creado el archivo “Dockerfile”, se creó una nueva carpeta, según se aprecia en la Figura 3.13. En la ventana “*New Folder”* se escribió el nombre “app” y se presionó en OK, esto se indica en la Figura 3.14.

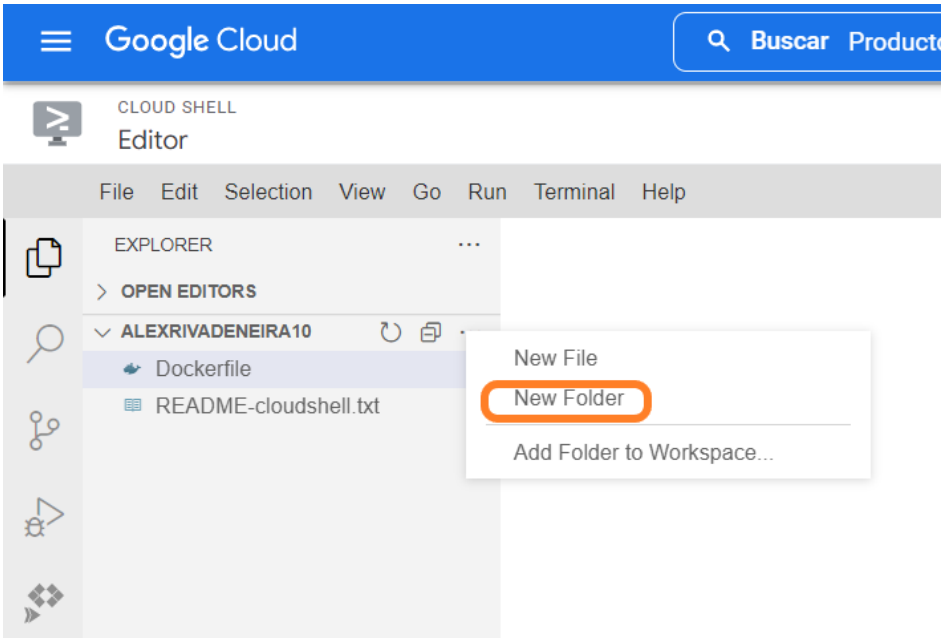

Figura 3.13 Creación nueva carpeta en *Cloud Shell*

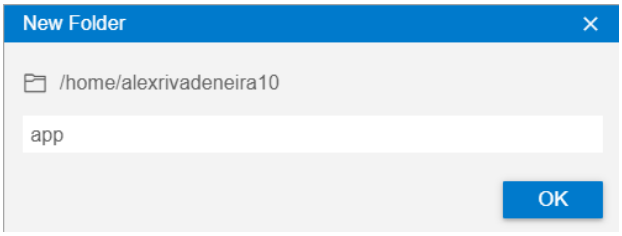

Figura 3.14 Nombrar Carpeta

Dentro de la carpeta “app” se creó un archivo nuevo con el contenido ilustrado en la Figura 3.15.

- En la primera línea tiene una instrucción “input” para el ingreso de datos, valor que se asigna a la variable “nombre” de tipo carácter.
- En la segunda línea se tiene el condicional “if” que compara el nombre ingresado con la cadena de texto “alex”, además contiene la función “lower()”, que convierte las letras ingresadas en minúsculas; si lo comparado es verdadero imprimirá en pantalla “Tesis de Alex Rivadeneira”.
- Caso contrario “else” imprimirá en pantalla Bienvenido, seguido del nombre ingresado por consola con letras mayúsculas gracias a la función “upper()”. Se guardó presionando las teclas Ctrl + S con el nombre de “ejemplo.py”.

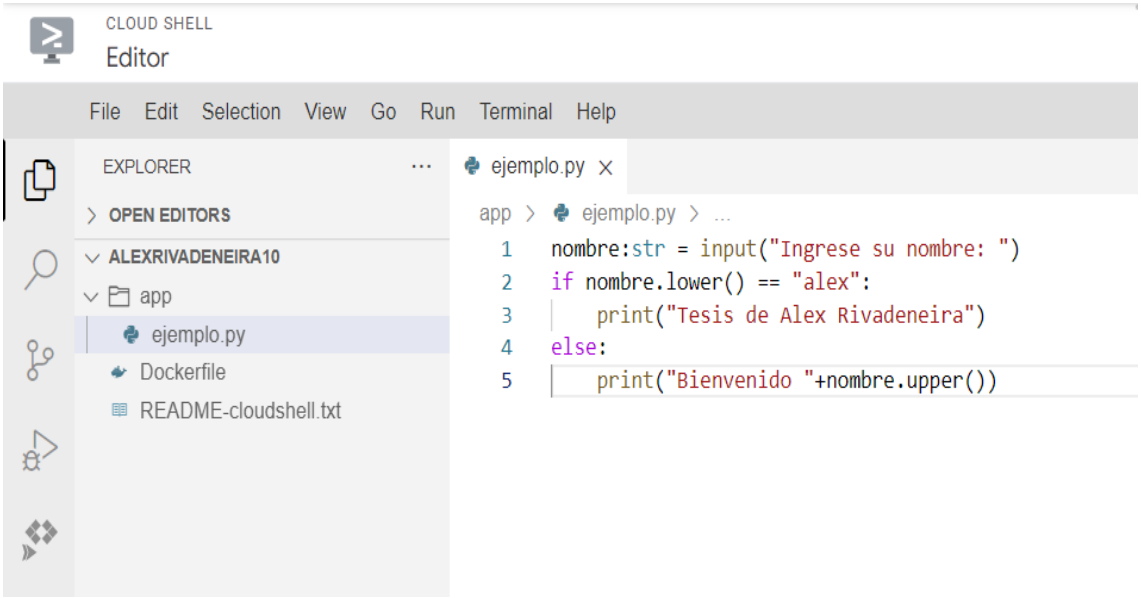

Figura 3.15 Ejemplo *Python*

Para finalizar con la implementación del contenedor de *Python*, se procedió a construir la nueva imagen del contenedor con el nombre “lenguaje-*Python*” y la versión 3.9 mediante el comando **docker build -t lenguaje-*Python*:3.9 .** (con este punto al final del comando se indica que esta instrucción se realice en el archivo actual). Esto se presenta en la Figura 3.16, cabe indicar que dicho comando es utilizado para crear una imagen de forma automática a partir del archivo “Dockerfile”.

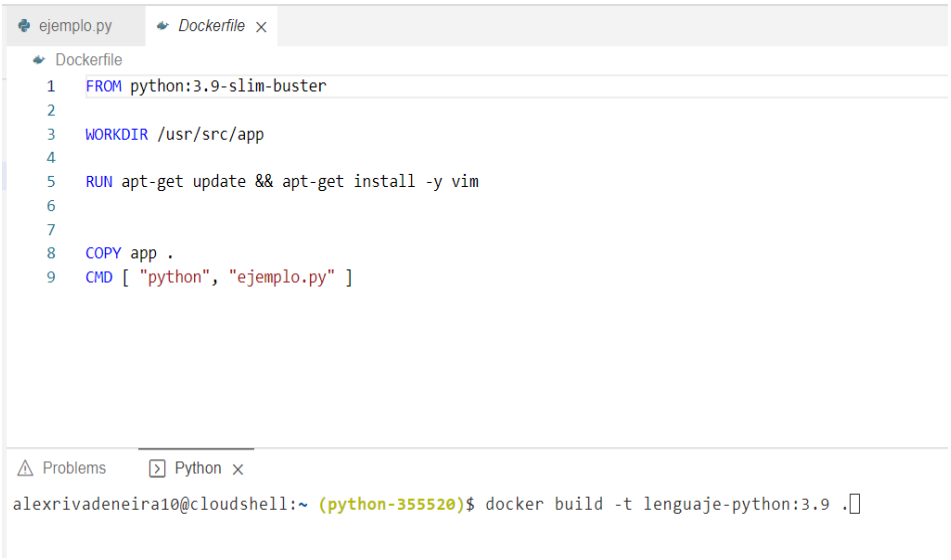

Figura 3.16 Construcción de nueva imagen

Por último, se verificó que la imagen esté creada correctamente con el comando **docker images**, esto se evidencia en la Figura 3.17; esta nueva imagen está compuesta tanto del lenguaje de *Python*, del editor "Vim" y del ejemplo escrito en *Python*; todo esto desde el archivo "Dockerfile".

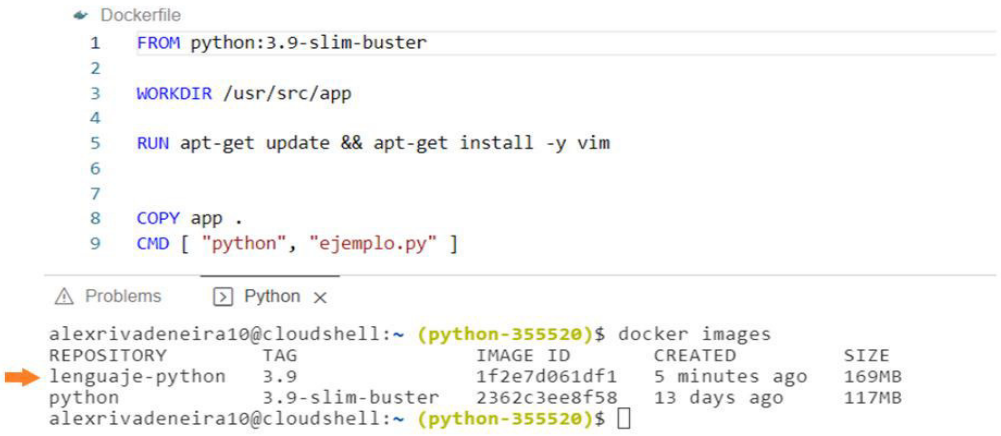

Figura 3.17 Verificación de la imagen de *Python* construida

Alojamiento en la nube del contenedor de *Python*

Una vez construida la nueva imagen del contenedor, se procedió a crear una etiqueta de dicha imagen con el comando **docker tag lenguaje-*Python*:3.9 gcr.io/*Python*-355520/*Python*-nube:3.9**, para subirla al registro de contenedores, mirar la Figura 3.18. Donde "gcr.io" es un repositorio que cambiará según la ubicación del host. Por ejemplo,

existe gcr.eu el cual está ubicado en Europa, el más usado para Latinoamérica es gcr.io; "*Python*-355520" es el ID del proyecto; "*Python*-nube:3.9" es el nombre de la imagen para la nube. Para verificar la creación de la etiqueta se ejecutó el comando **docker images**.

Figura 3.18 Creación de Etiqueta para la nube

En la Figura 3.19 se visualiza como se subió la imagen etiquetada a la nube de GCP con el comando **docker push gcr.io/*Python*-355520/*Python*-nube:3.9**. Se comprobó que la imagen está alojada en la nube dentro del registro de contendores refrescando la página de GCP, lo que se presenta en la Figura 3.20.

```
alexrivadeneira10@cloudshell:~ (python-355520)$ docker images
REPOSITORY                          TAG               IMAGE ID       CREATED          SIZE
lenguaje-python                     3.9               1f2e7d061df1   57 minutes ago   169MB
gcr.io/python-355520/python-nube    3.9               1f2e7d061df1   57 minutes ago   169MB
python                              3.9-slim-buster   2362c3ee8f58   13 days ago      117MB
alexrivadeneira10@cloudshell:~ (python-355520)$ docker push gcr.io/python-355520/python-nube:3.9
The push refers to repository [gcr.io/python-355520/python-nube]
19394ac56224: Preparing
5a225ae93bff: Preparing
5a225ae93bff: Pushed
0e6b356fb909: Layer already exists
30a5917ac819: Layer already exists
a9ebae26583c: Layer already exists
7ba8fd3fc230: Layer already exists
12cbeae46147: Layer already exists
3.9: digest: sha256:d5e6b9373ae6f80245986b55825009421a239e9eaaadc30b25d8a81a791236bf size: 1996
alexrivadeneira10@cloudshell:~ (python-355520)$
```

Figura 3.19 Subir imagen de *Python* a la nube de GCP

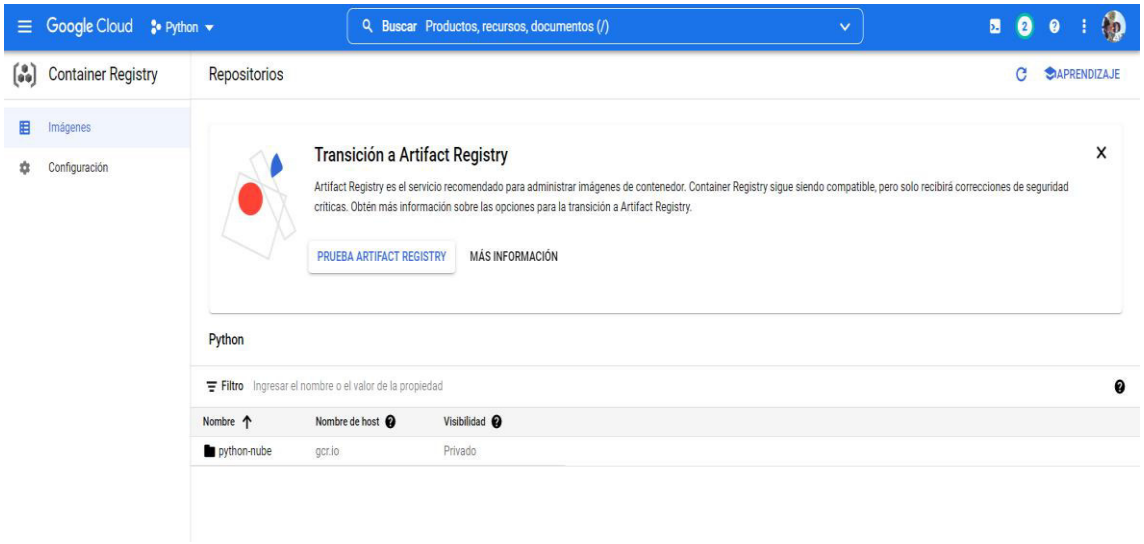

Figura 3.20 Verificación de imagen de *Python* en GCP

Implementación del contenedor de R

Para la implementación del contenedor de R se creó un nuevo proyecto dando clic en la opción “Selecciona un proyecto”, después de esto se desplegó una ventana, donde se seleccionó en el botón “PROYECTO NUEVO”, observar Figura 3.21. En la Figura 3.22 se puede apreciar el nuevo proyecto con el nombre “Lenguaje-R”.

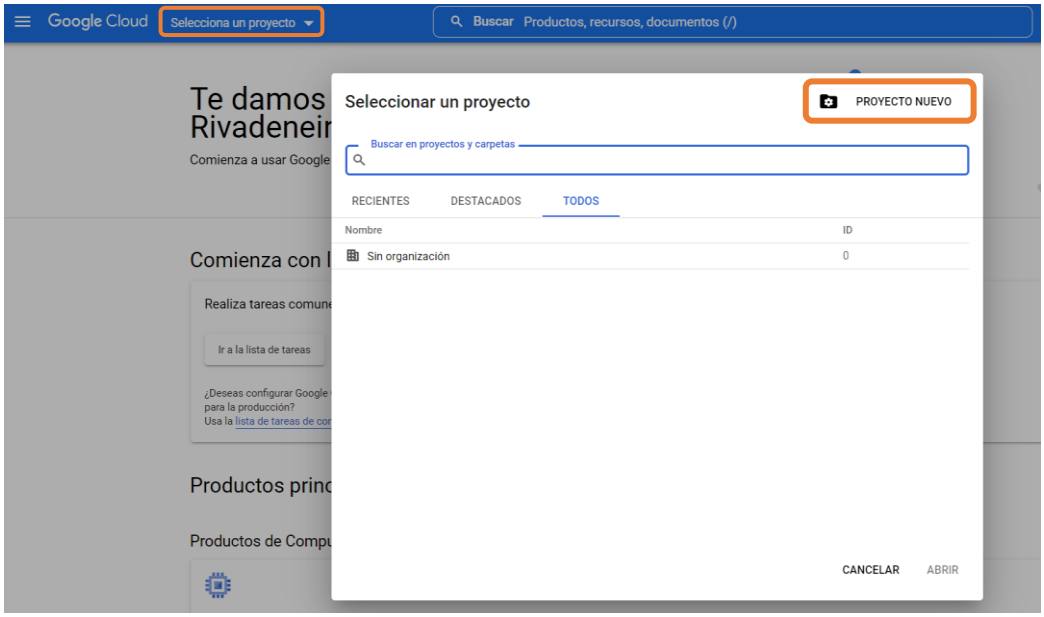

Figura 3.21 Creación del nuevo proyecto lenguaje R

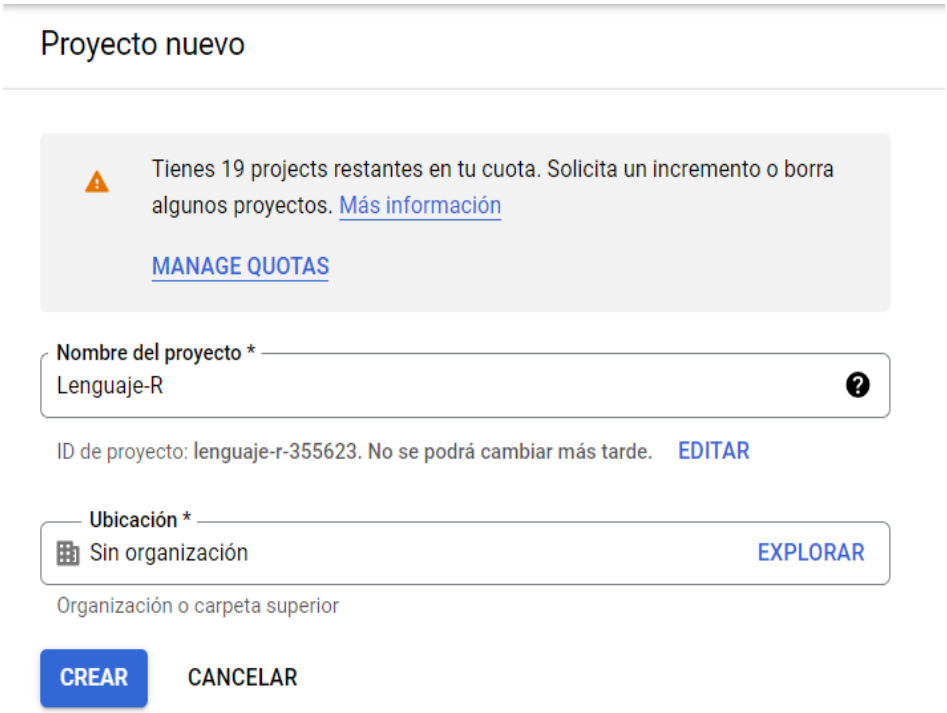

Figura 3.22 Nombre del proyecto Lenguaje-R

Dentro del menú de navegación se encuentran varias herramientas que ofrece GCP, para este caso se usó el registro de contenedores y en la opción imágenes se habilitó la API de registro de contenedores de Google. Después se activó el intérprete de comandos seleccionando el botón “Activar *Cloud Shell*”, como se realizó en el proyecto anterior.

En la Figura 3.23, se presenta la búsqueda y selección de la imagen de rstudio del repositorio “Docker Hub”.

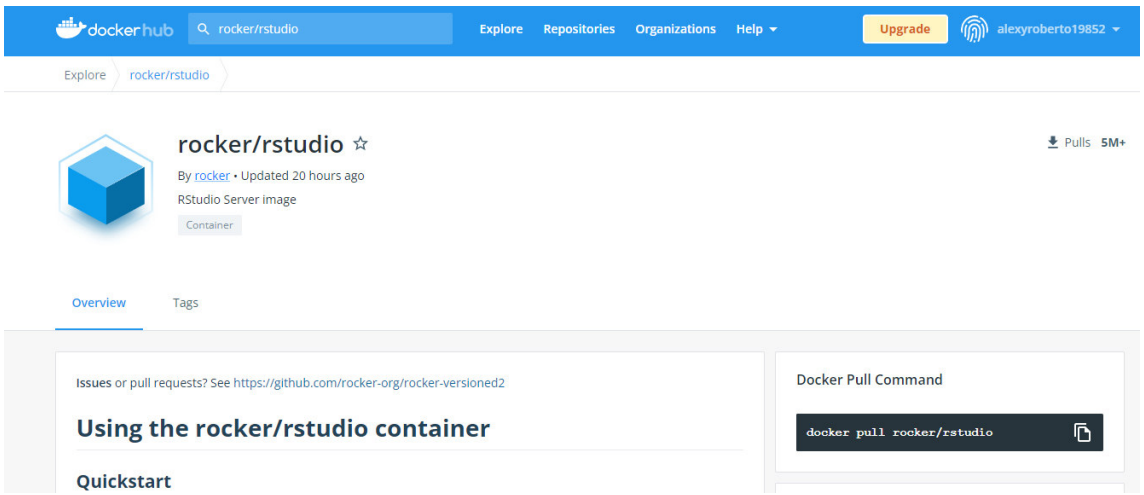

Figura 3.23 Imagen rstudio en “Docker Hub”

Una vez seleccionada la imagen, se descargó con el comando **docker pull rocker/rstudio**, esto se indica en la Figura 3.24.

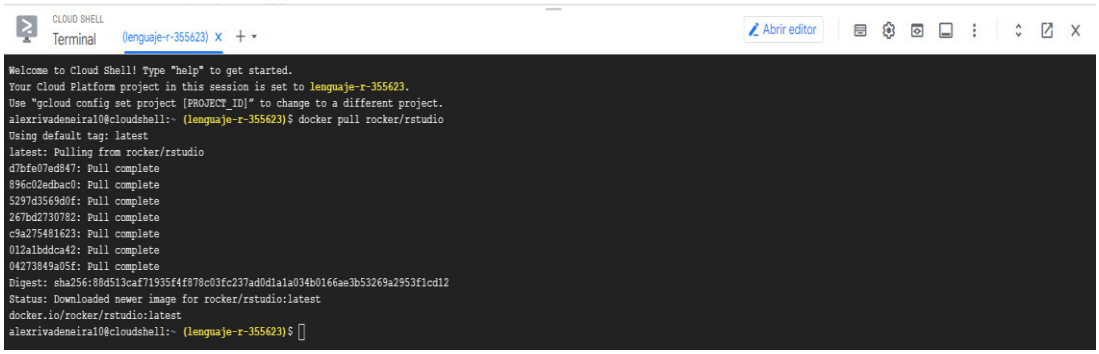

Figura 3.24 Descarga imagen rstudio

Finalmente, en la Figura 3.25 se ve la comprobación de la imagen descargada con el comando **docker images**.

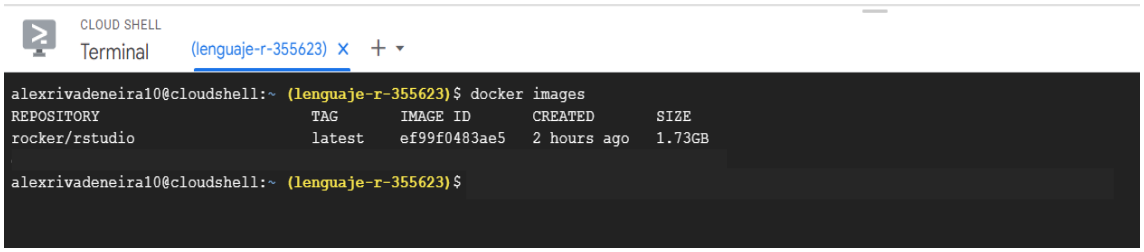

Figura 3.25 Comprobación imagen rstudio descargada

Alojamiento en la nube del contenedor de R

La Figura 3.26 presenta la creación de una etiqueta de la imagen rocker/rstudio con el comando **docker tag rocker/rstudio:latest gcr.io/lenguaje-r-355623/r-nube:latest**, para subirla al registro de contenedores; donde “lenguaje-r-355623” es el ID del proyecto y “r-nube:latest” es el nombre de la imagen.

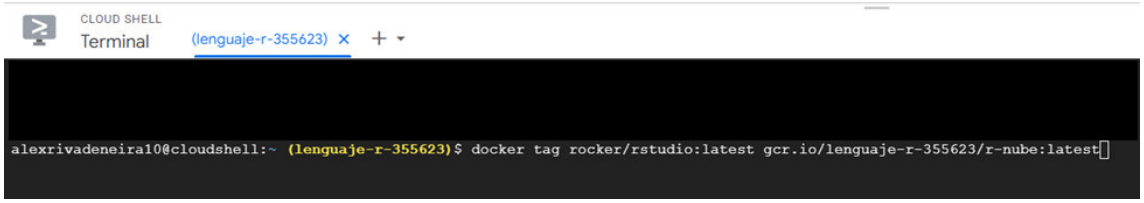

Figura 3.26 Creación de etiqueta rstudio

Por último, se subió esta imagen a la nube de GCP mediante el comando **docker push gcr.io/lenguaje-r-355623/r-nube:latest**, como se expone en la Figura 3.27 y se comprobó la imagen alojada en la nube dentro del registro de contenedores refrescando la página de GCP, ver Figura 3.28.

```
alexrivadeneira10@cloudshell:~ (lenguaje-r-355623)$ docker push gcr.io/lenguaje-r-355623/r-nube:latest
The push refers to repository [gcr.io/lenguaje-r-355623/r-nube]
092e64c86002: Pushed
cffe9dc5cd31: Pushed
6961c2c4dbbb: Pushed
c1889d29e0c8: Pushed
8bb1aa6a6cff: Layer already exists
4edddafacf4d: Layer already exists
af7ed92504ae: Layer already exists
latest: digest: sha256:3a73138267f74ccf51a42d98a22fc556459392ee9ffdda8c5157686249a85785 size: 1793
alexrivadeneira10@cloudshell:~ (lenguaje-r-355623)$
```

Figura 3.27 Subir imagen rstudio a GCP

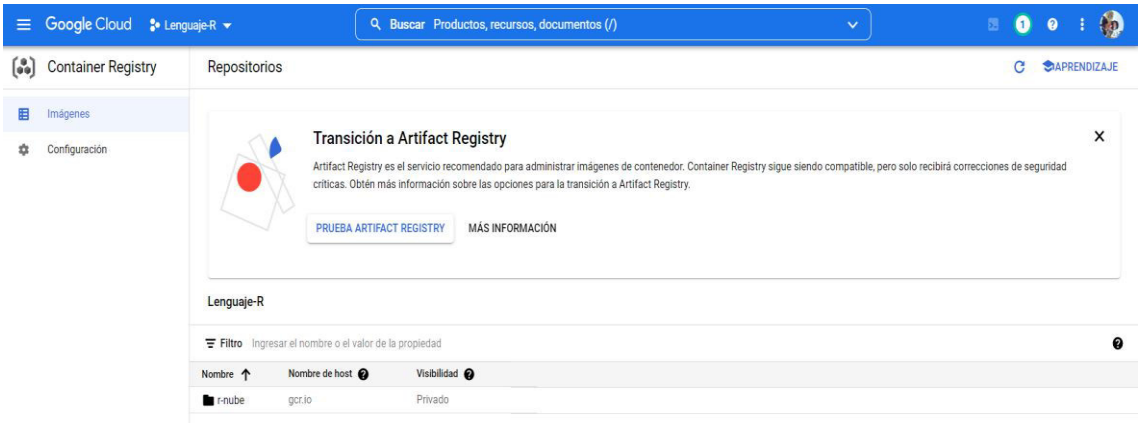

Figura 3.28 Comprobación de imagen rstudio en GCP

3.4 Pruebas de funcionamiento de los resultados alcanzados

Una vez implementado los contenedores y alojados en la nube de GCP, se realizaron pruebas de funcionamiento y la verificación del desempeño de cada uno de los contenedores.

Pruebas de funcionamiento del contenedor de *Python* en la nube

Para comprobar el funcionamiento, una vez subida la imagen se seleccionó la carpeta "*Python*-nube", ver Figura 3.29.

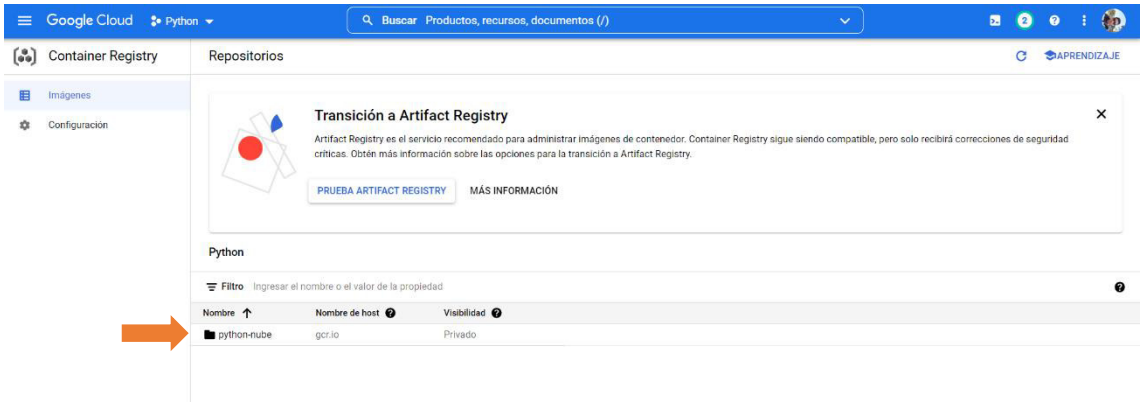

Figura 3.29 Selección de carpeta imagen de *Python*

Después de esto, en la Figura 3.30 se presenta la selección del nombre de la imagen dando clic sobre dicho nombre.

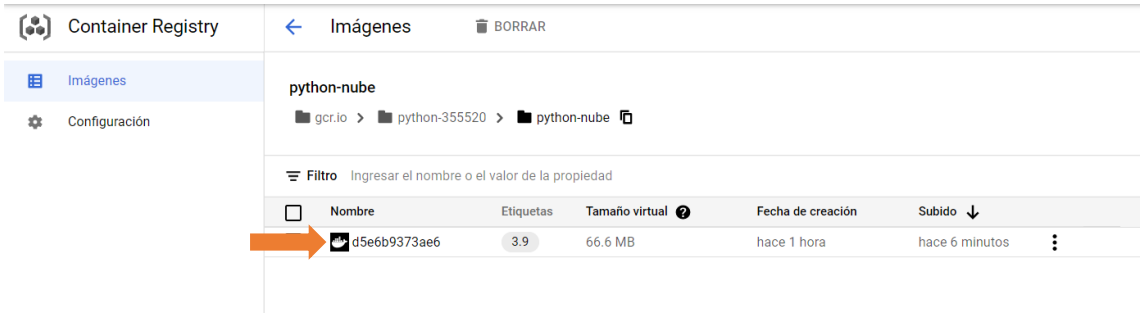

Figura 3.30 Selección imagen de *Python*

Después se extrajo la imagen alojada en GCP seleccionando “Ejecutar en *Cloud Shell*” dentro del menú “EXTRACCIÓN” opción “extrae por etiqueta”, lo cual se expone en la Figura 3.31. Una vez que aparece el *Cloud Shell* se desplegó el comando **pull**, y se presionó la tecla “Enter” para ejecutarlo, observar Figura 3.32.

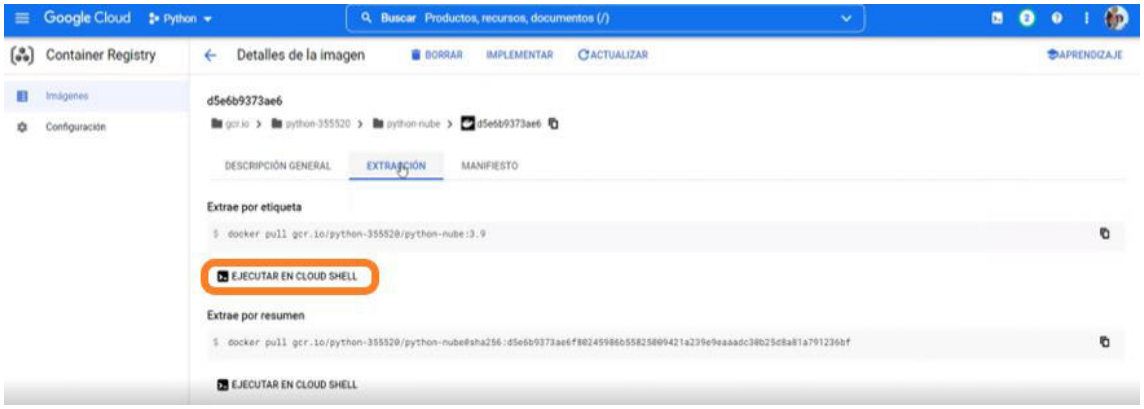

Figura 3.31 Extracción imagen de *Python*

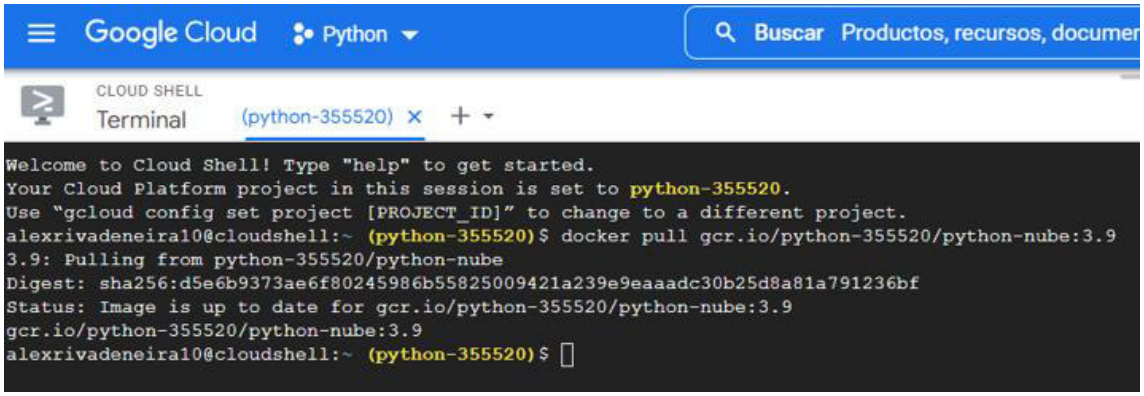

Figura 3.32 Ejecución del comando **pull**

Después se procedió a crear y ejecutar el contenedor de forma iterativa con el comando **docker run -it gcr.io/*Python*-355520/*Python*-nube:3.9**. En la Figura 3.33 se evidenció el correcto funcionamiento del lenguaje interpretado *Python* alojado en la nube GCP.

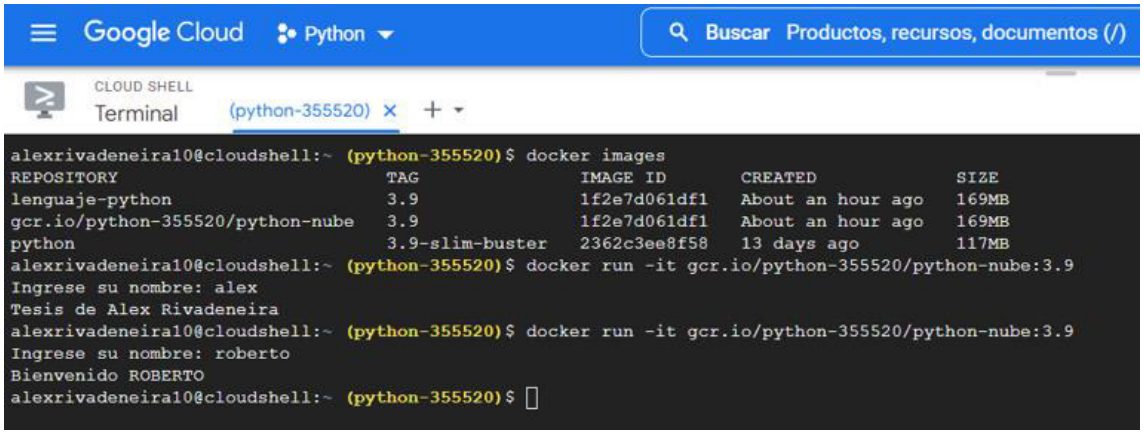

Figura 3.33 Prueba de ejecución del contenedor de *Python*

Además, se comprobó el funcionamiento del editor “Vim” con el comando **docker run -it gcr.io/*Python*-355520/*Python*-nube:3.9 sh**, ver la Figura 3.34. Esta terminación sh sirve para mantener la consola abierta dentro del contenedor.

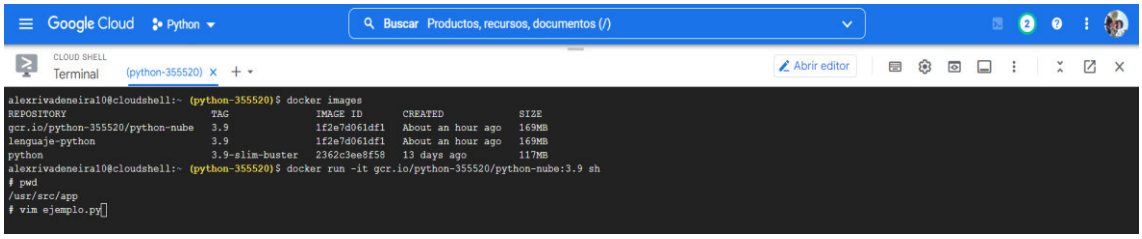

Figura 3.34 Consola del contenedor

Una vez en la consola se ingresó al archivo ejemplo.py mediante el editor “Vim” con el comando **vim ejemplo.py** y se realizó el cambio de texto “Ingrese su nombre” por

"Ingrese el nombre", esto se demuestra en la Figura 3.35. Para verificar el cambio realizado se ejecutó el comando ***Python* ejemplo.py** presentado en la Figura 3.36.

Figura 3.35 Cambio del archivo ejemplo.py

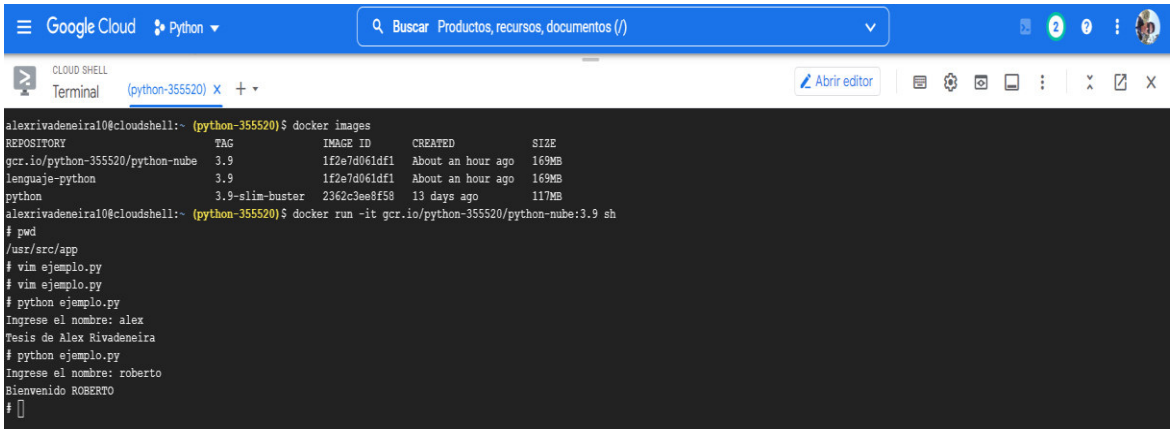

Figura 3.36 Comprobación de cambio realizado en el archivo ejemplo.py

En este contexto se logró evidenciar el funcionamiento del contenedor con el lenguaje de *Python*, al ejecutar con éxito el código ejemplo escrito en dicho lenguaje.

Pruebas de funcionamiento del contenedor de R en la nube

Para comprobar el funcionamiento del contenedor de R, una vez subida la imagen se seleccionó dando clic en el nombre de la carpeta “r-nube”, lo dicho se presenta en la Figura 3.37.

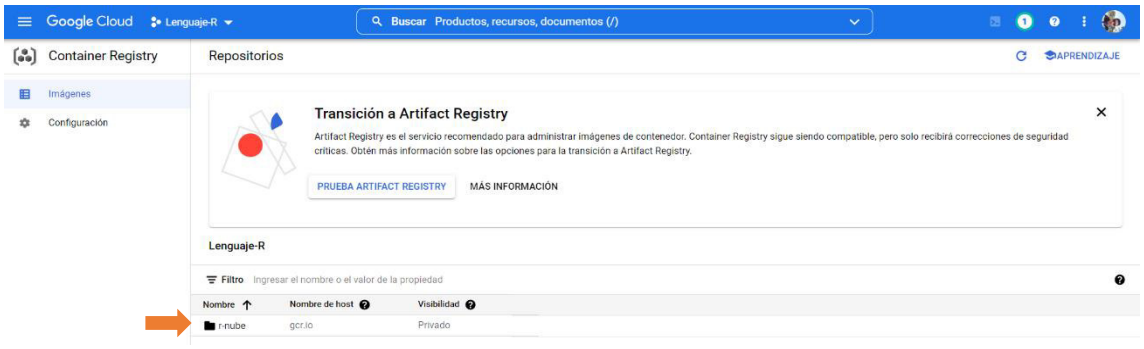

Figura 3.37 Selección carpeta r-nube

La Figura 3.38 permite apreciar que se seleccionó el nombre de la imagen dando clic en el nombre.

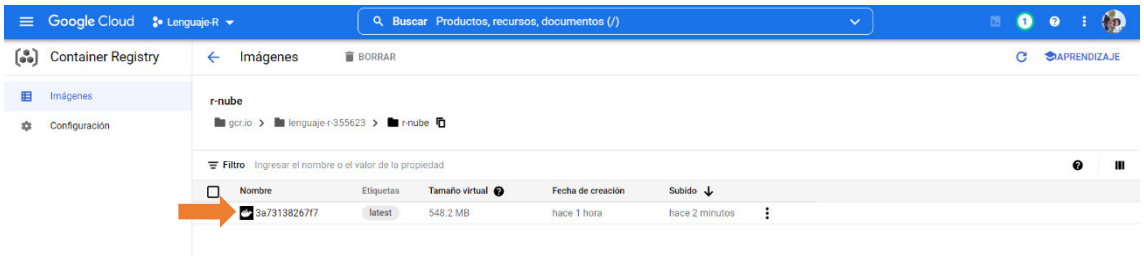

Figura 3.38 Selección imagen de R

En la Figura 3.39 se observa la selección de la opción “Implementación en *Cloud Run*” del menú “IMPLEMENTAR”.

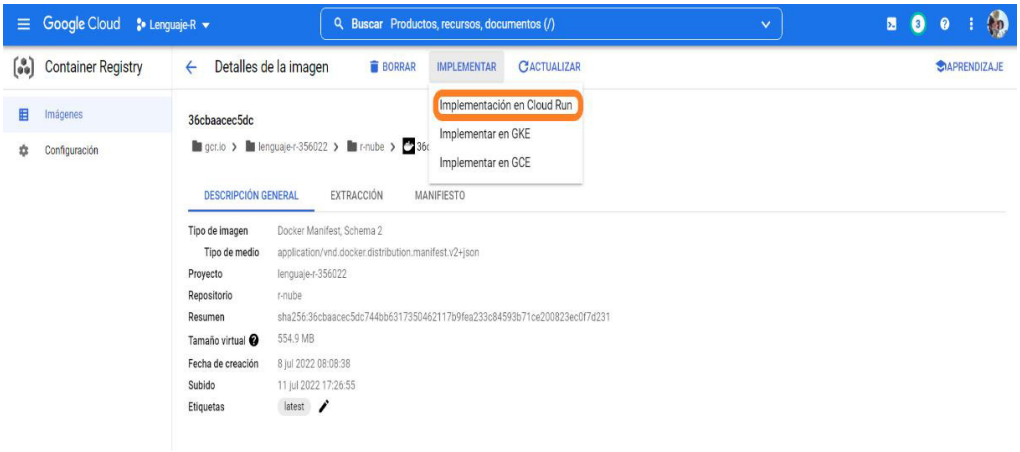

Figura 3.39 Implementación de R en *Cloud Run*

Una vez seleccionada esta opción apareció el formulario de creación del servicio, en el campo “URL de la imagen del contenedor” debe estar seleccionado “gcr.io-lenguaje-r-355623/r-nube:latest”, en el campo “Nombre del servicio” se escribió el nombre del servicio en este caso se colocó “r-nube”, luego de esto se mostró un mensaje que solicitó la habilitación de *Cloud Run* el cual se tiene que habilitar, esto se exhibe en la Figura 3.40.

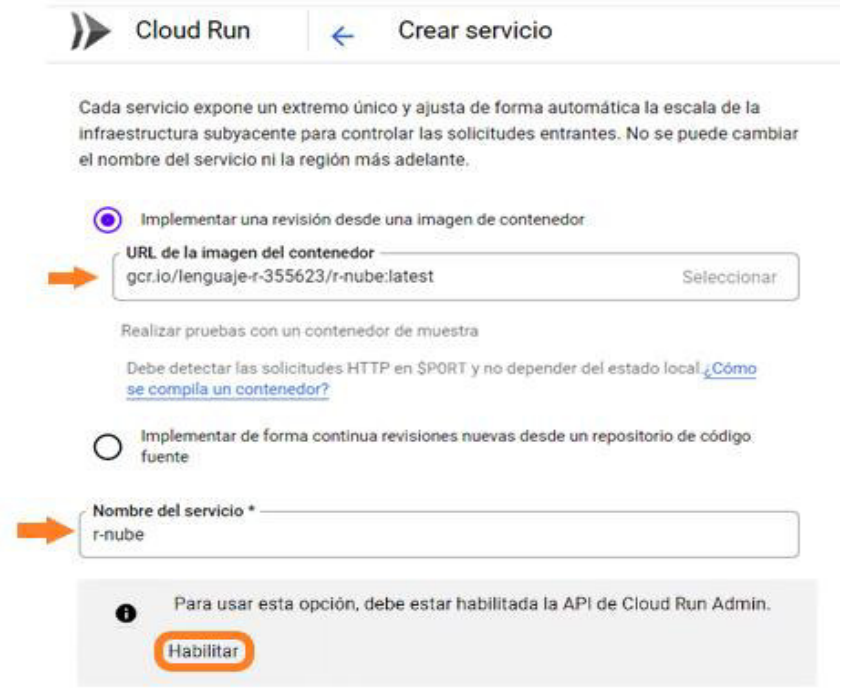

Figura 3.40 Formulario de creación del servicio en *Cloud Run*

Después se abrió una nueva pestaña donde se seleccionó la opción “Comenzar a usar *Cloud Run*”. Una vez seleccionada esta opción se habilitó el servicio de *Cloud Run* para la implementación del contenedor del lenguaje R, señalado en la Figura 3.41.

Figura 3.41 Activación del servicio de *Cloud Run*

Luego de esto se seleccionó la opción imágenes del servicio "*Container Registry*" del menú de navegación "Google *Cloud*", visualizado en la Figura 3.42; esto redirige a la imagen del contenedor R alojada en la nube, en donde se selecciona dando un clic en el nombre de la carpeta "r-nube", expuesto en la Figura 3.37. Luego se seleccionó la imagen dando clic en el nombre, indicado en la Figura 3.38. Más adelante se selecciona la opción "Implementación en *Cloud Run*" del menú "IMPLEMENTAR", ver Figura 3.39.

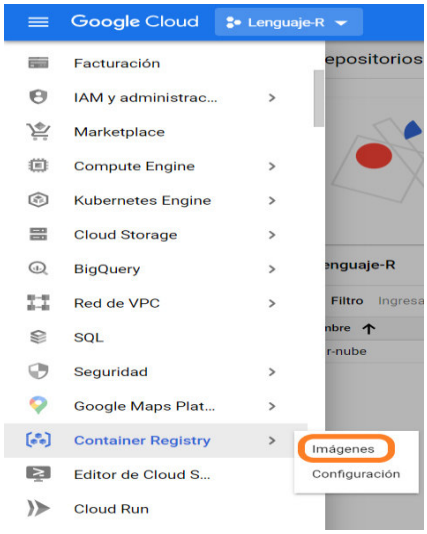

Figura 3.42 Menú de navegación *Google Cloud*

Una vez seleccionada la opción "Implementación en *Cloud Run*", en la Figura 3.43 apareció nuevamente el formulario de creación del servicio con el campo de región habilitado, donde se selecciona la región "us-central1 (Iowa)" para América, lo que aumentará la disponibilidad del servicio por la ubicación.

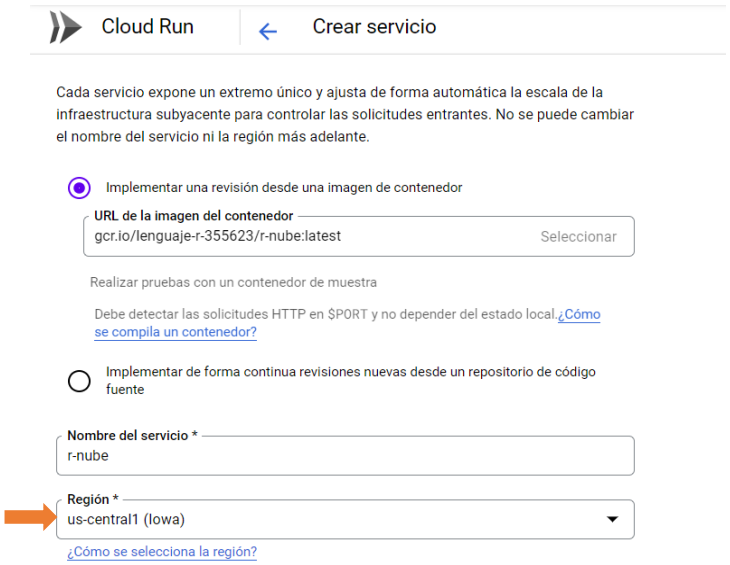

Figura 3.43 Establecimiento de región del servicio en *Cloud Run*

En “Precios y asignación de CPU” se seleccionó la opción “La CPU solo se asigna durante el procesamiento de la solitud”, ya que solo estará disponible el servicio cuando lleguen solicitudes. En “Ajuste de escala Automático” se modificó el “número máximo de instancias” a 10. En “Entrada” se dejó seleccionada la opción “Permitir todo el tráfico”, presentado en la Figura 3.44.

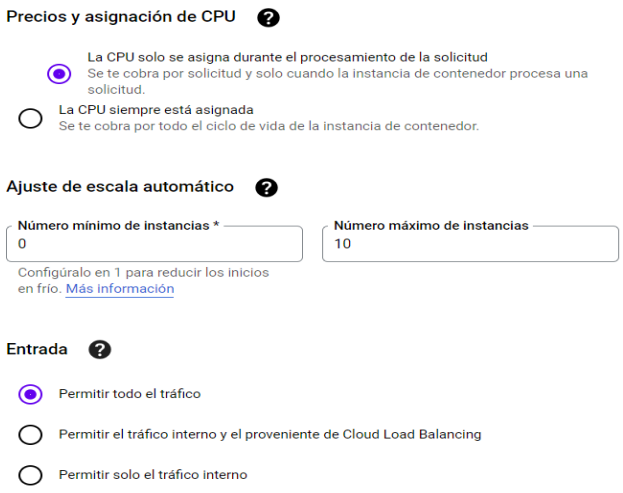

Figura 3.44 Configuración de Precios, Ajuste y Entrada en *Cloud Run*

En “Autenticación” se seleccionó la opción “Permitir invocación sin autenticar”, ya que se trata de una prueba y no es necesario contar con seguridad, mirar Figura 3.45. En la opción “CONTENEDOR” se modificó el puerto del contenedor a 8787, ya que es el puerto que está expuesto en la imagen del contenedor de R, apreciado en la Figura 3.46.

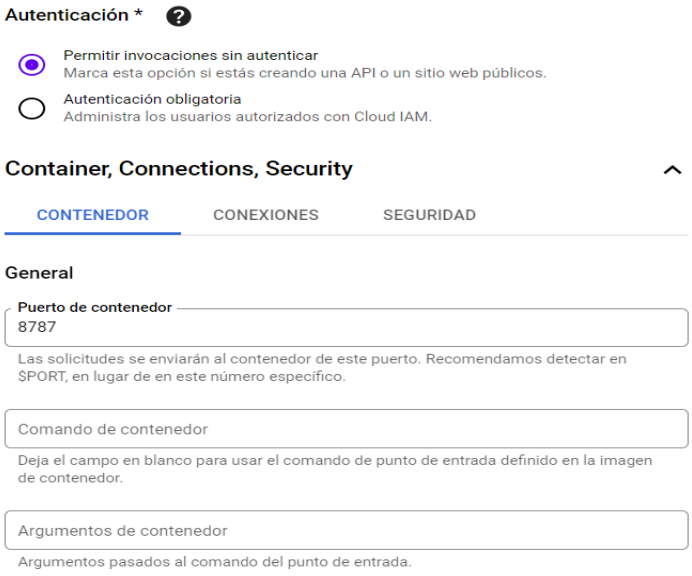

Figura 3.45 Modificación de autenticación y puerto en *Cloud Run*

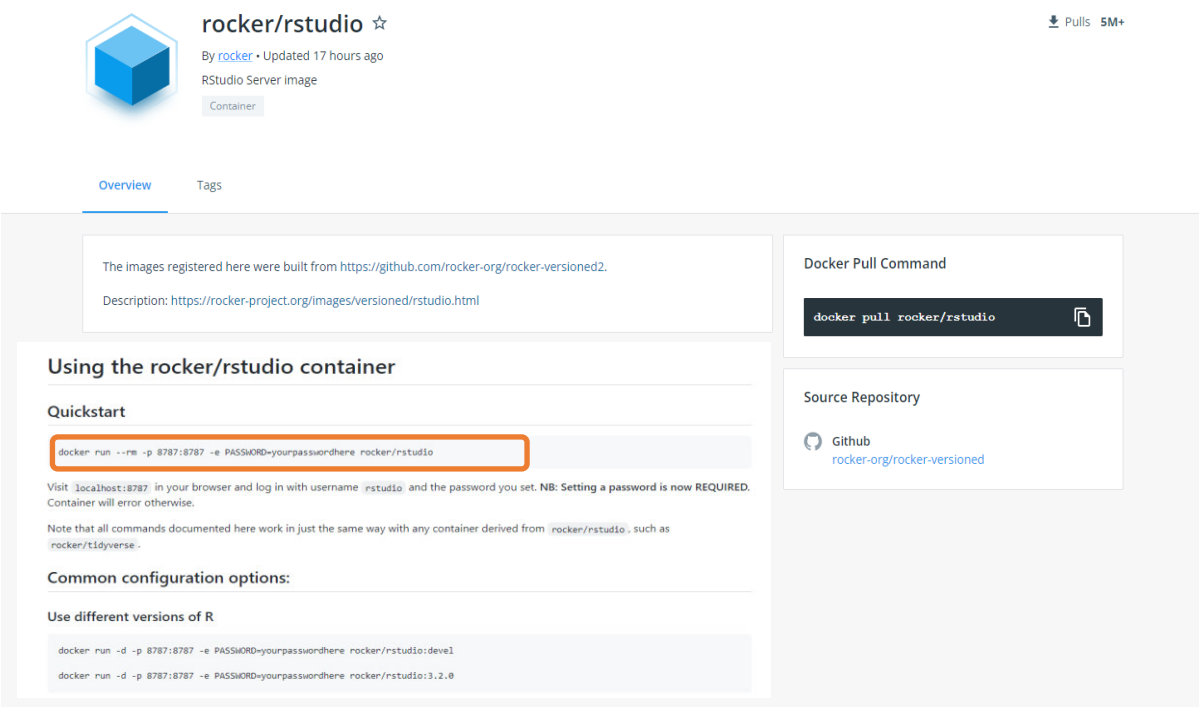

Figura 3.46 Puerto expuesto en la imagen del contenedor de R

La Figura 3.47 presenta la inclusión de una variable de entorno llamada "*PASSWORD*" con el valor "AlexPass" en la pestaña "Variables del entorno", con el fin de personalizar la contraseña para el inicio de sesión en RStudio.

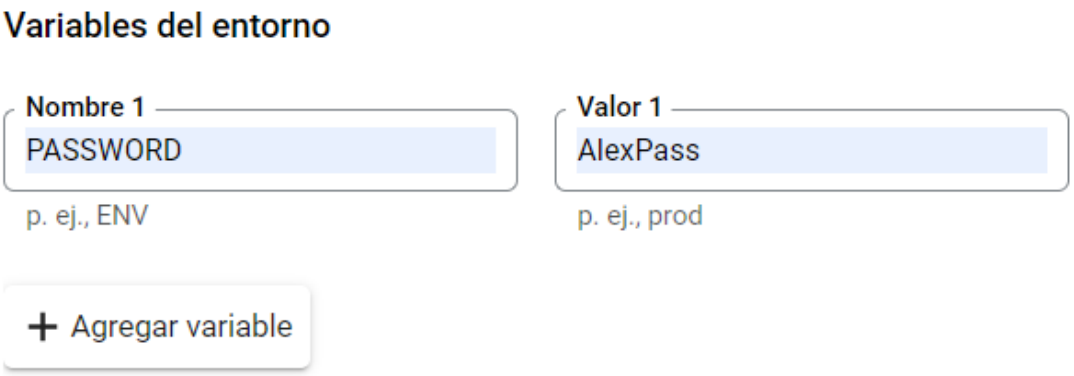

Figura 3.47 Agregación de variable de entorno

En "Capacidad" y "Entorno de ejecución" no fue necesario realizar ningún cambio y finalmente se creó el servicio presionando el botón "crear", lo que se presenta en la Figura 3.48.

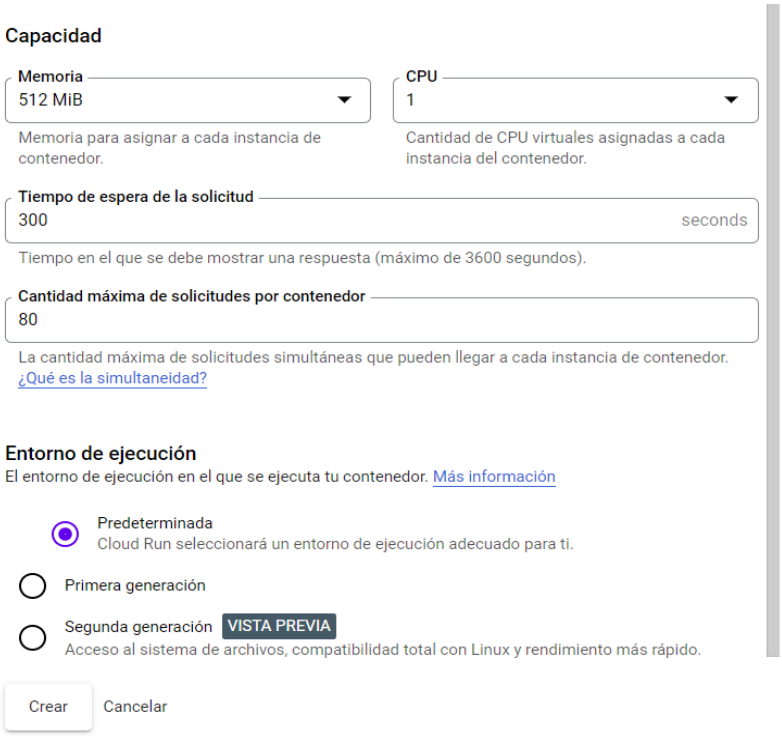

Figura 3.48 Capacidad y Entorno de ejecución en Formulario de *Cloud Run*

En la Figura 3.49 se visualiza que una vez completados los pasos para la creación del servicio, *Cloud Run* verificó el contenedor y creó el servicio generando una URL.

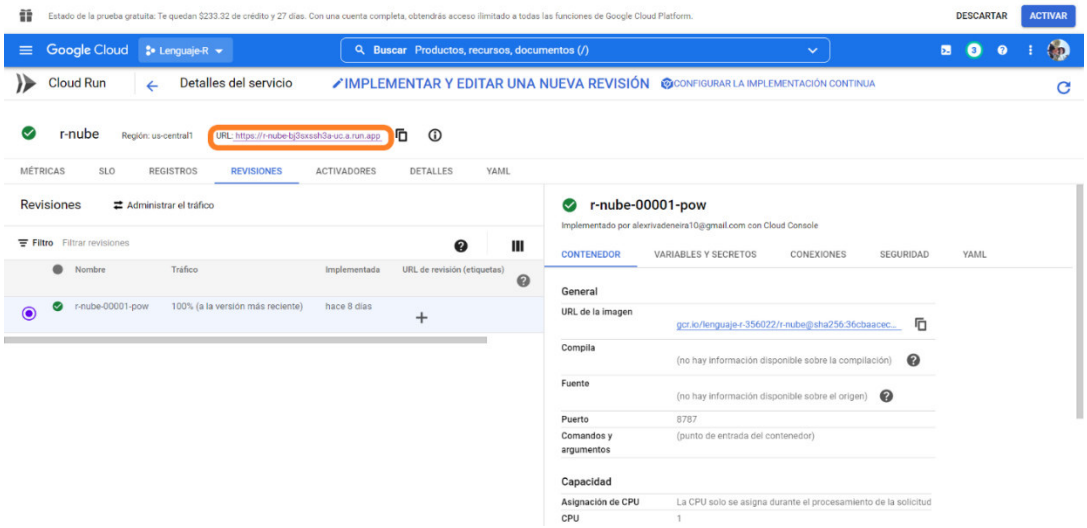

Figura 3.49 Verificación y creación del servicio

Con la URL generada en el paso anterior se accedió al servicio y apareció la página de inicio de sesión de RStudio, donde se ingresó con el nombre de usuario “rstudio” y la clave “AlexPass” que fue personalizada en la creación del servicio, ver Figura 3.50.

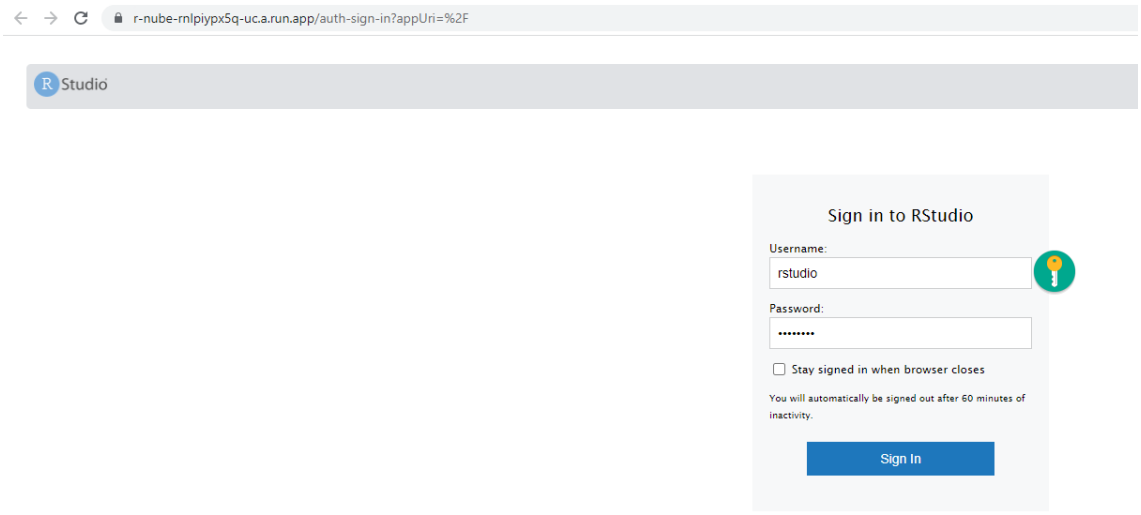

Figura 3.50 Inicio de sesión en RStudio

La Figura 3.51 enseña la verificación del adecuado funcionamiento de RStudio; con lo cual aparecieron tres áreas de trabajo: la primera es el panel de código, la segunda es el panel de entorno, la tercera es el panel de ficheros y ploteo de gráficos.

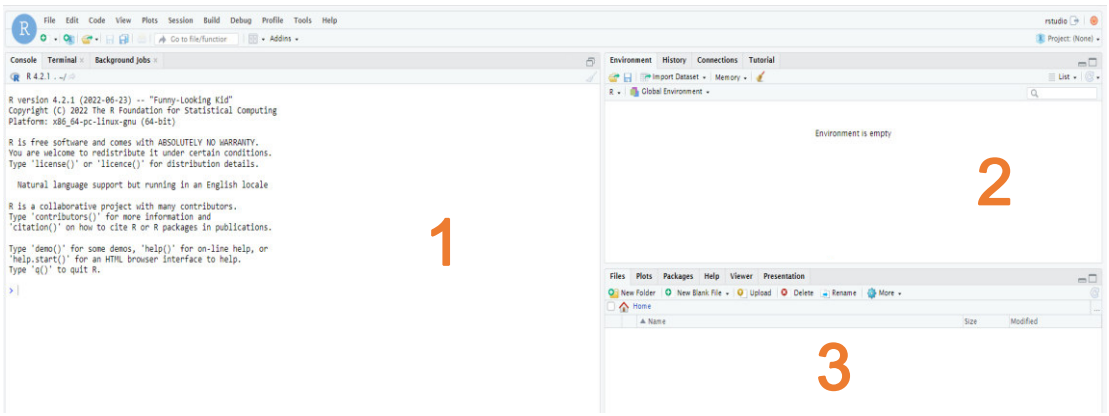

Figura 3.51 Áreas RStudio

Una vez en el panel de ficheros se seleccionó la opción "*New Folder*" dentro del menú "*Files*", esto mostró una ventana en donde se creó una carpeta con el nombre "App" y se presionó en OK, revisar la Figura 3.52.

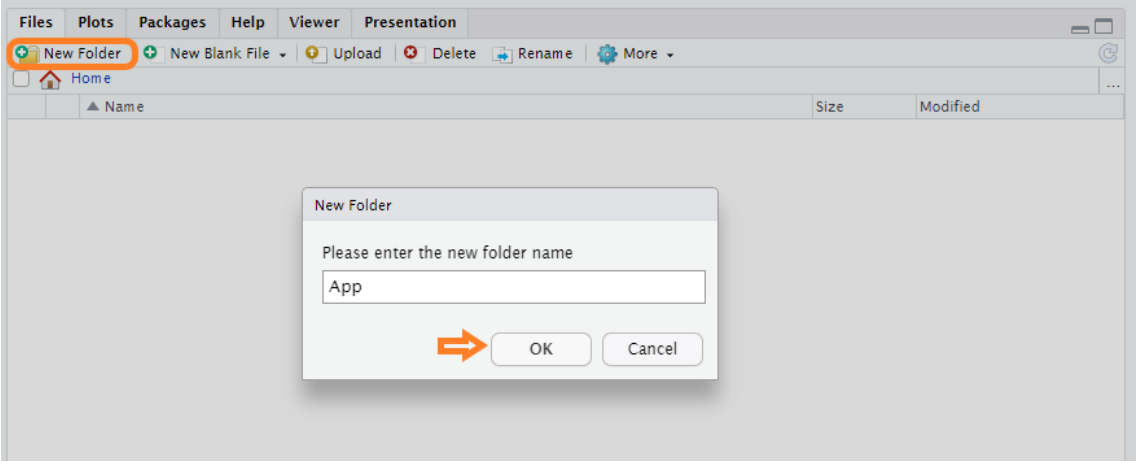

Figura 3.52 Creación carpeta App

En la Figura 3.53 se aprecia que dentro de la carpeta “App” se seleccionó la opción “*Upload*” del menú “*Files*”, lo que desplegó la ventana “*Upload Files*”, en donde se presionó en el botón “Seleccionar archivo”.

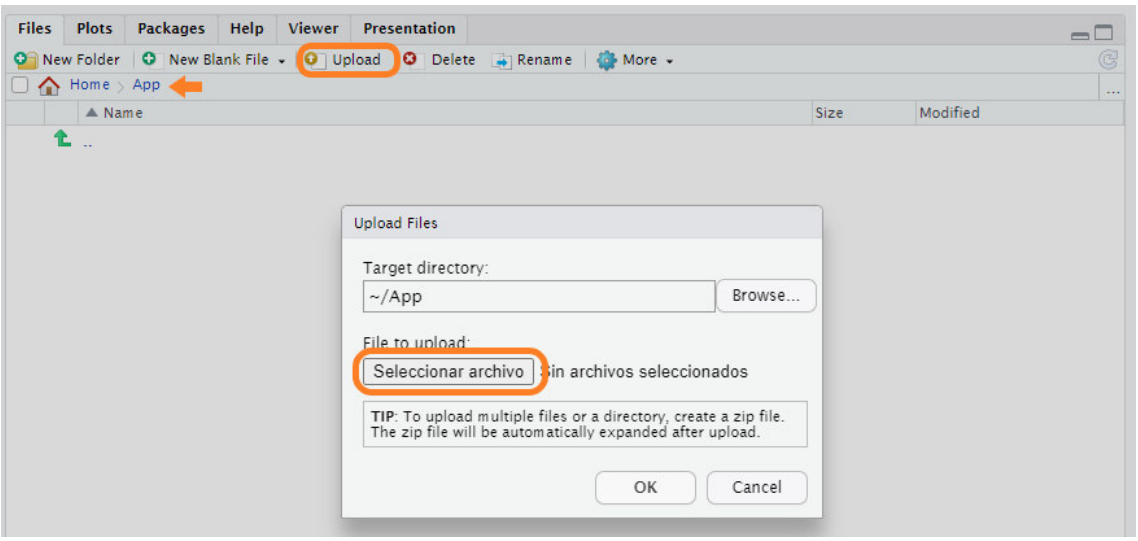

Figura 3.53 Elección del archivo a subir

Luego se desplegó la ventana “Abrir” donde se buscó, seleccionó y abrió el archivo “Registro.csv” que se encuentra ubicado en el escritorio del equipo físico. Este es un archivo de texto separado por comas que contiene una lista de 30 alumnos cada uno con su respectiva nota final, como indica la Figura 3.54.

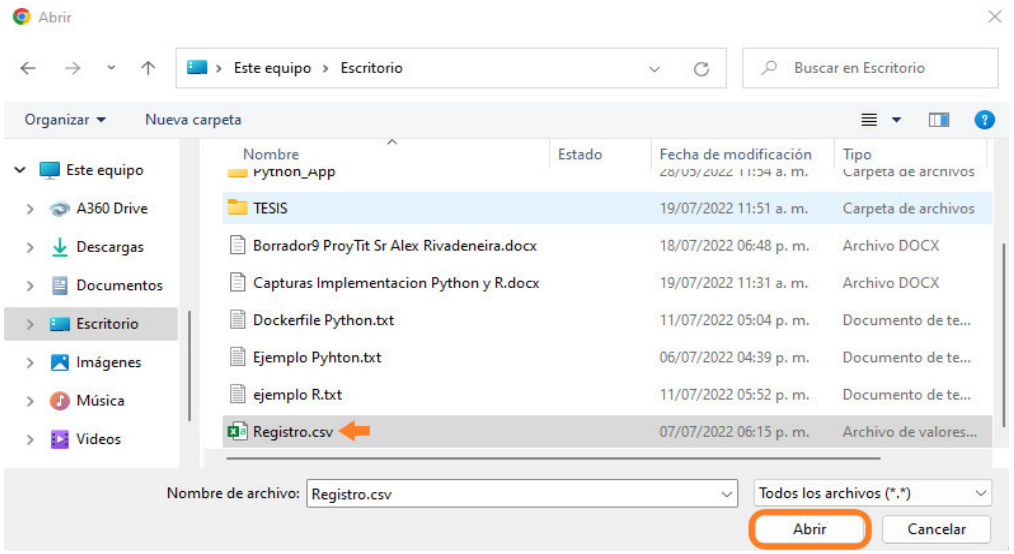

Figura 3.54 Abrir Archivo Registro.csv

Una vez abierto el archivo se regresa a la ventana "*Upload Files*", donde se presiona "Ok" para subir el archivo, observar en la Figura 3.55; después de esto se mostró el archivo subido, ver Figura 3.56.

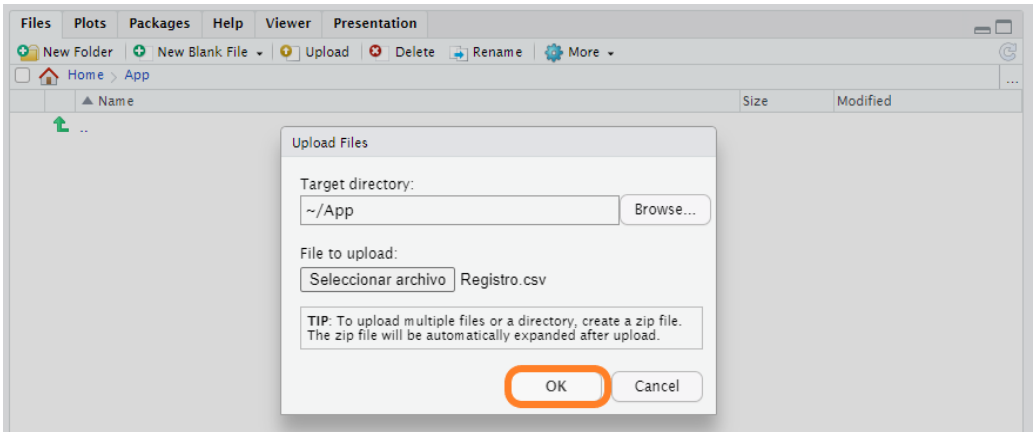

Figura 3.55 Subir el archivo Registro.csv

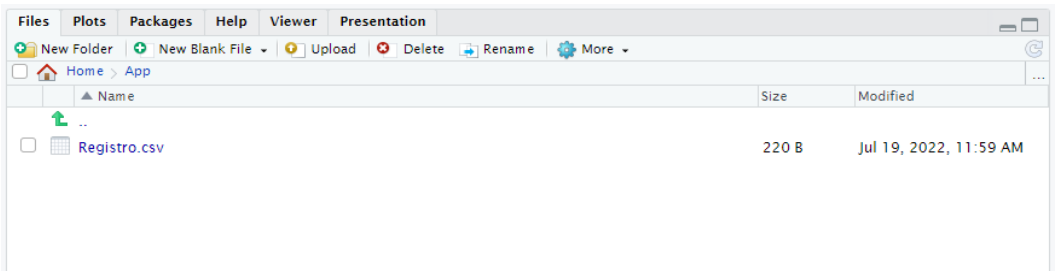

Figura 3.56 Archivo subido

Como se expone en la Figura 3.57, dentro del menú "*File*", submenú "*New File*", se selecciona la opción "R *Script*" para crear un nuevo *Script*.

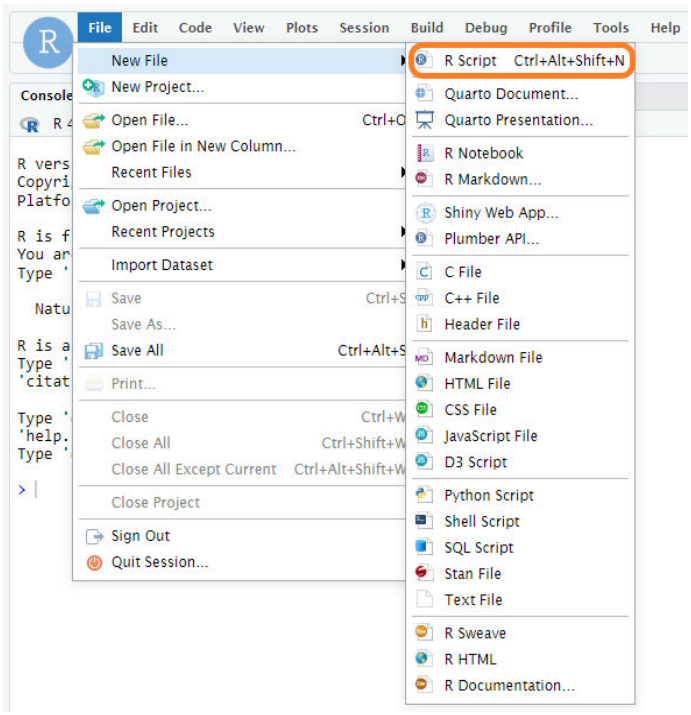

Figura 3.57 Creación de R *Script*

Dentro del archivo *Script* se escribió un código de ejemplo en R.

- En la primera línea se creó la variable registronotas en la que se asignó el resultado de la función read.csv() que lee un archivo .csv y tiene los siguientes parámetros: “App/Registro.csv” es el archivo con la lista de alumnos que está dentro de la carpeta app, header=TRUE indica que el archivo tiene una cabecera, sep=”,” indica que el separador de campos que usa el archivo es la coma.
- En la segunda línea se creó el histograma con la función hist() con los siguientes parámetros: registronotas$NotaFinal es el vector con la cabecera sobre la cual se va a realizar el histograma, col=c() es el vector de colores para las barras del histograma, main=”Estudiantes” es el título principal del histograma, xlabel=”Notas” es el título del eje x, ylabel=”Frecuencia” es el título del eje Y. Todo esto se visualiza en la Figura 3.58.

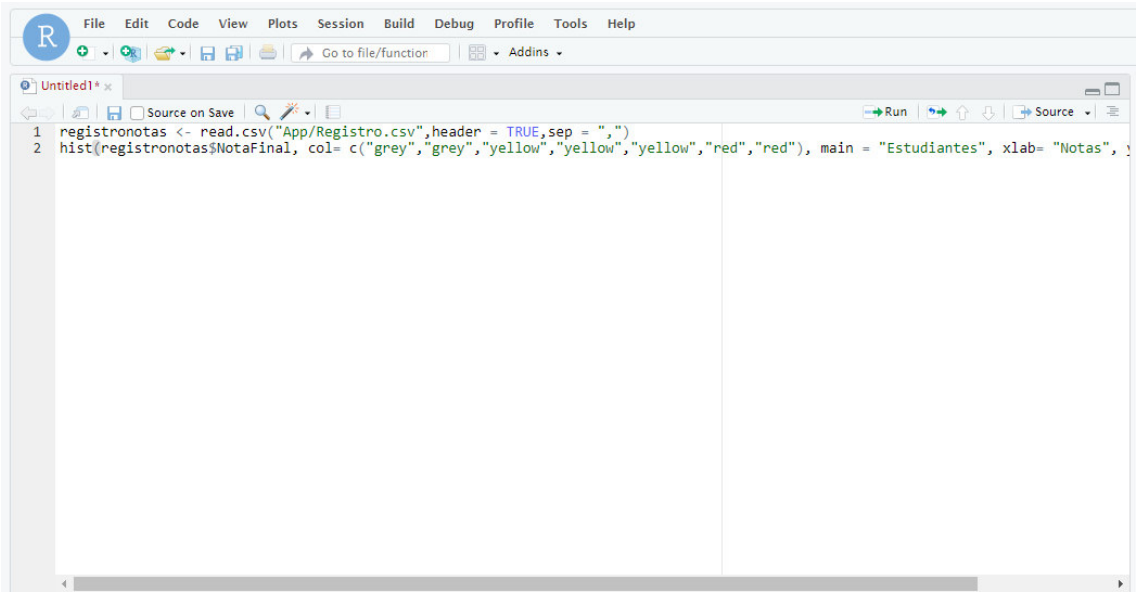

Figura 3.58 Código del ejemplo en R

Una vez escrito el código se presionó las teclas Ctrl + S para grabar y se escribió dentro del cuadro de texto "*File name*" el nombre "Programa Notas", teniendo en cuenta que debe estar dentro de la carpeta "App" y se presionó en el botón "*Save*", revisar la Figura 3.59.

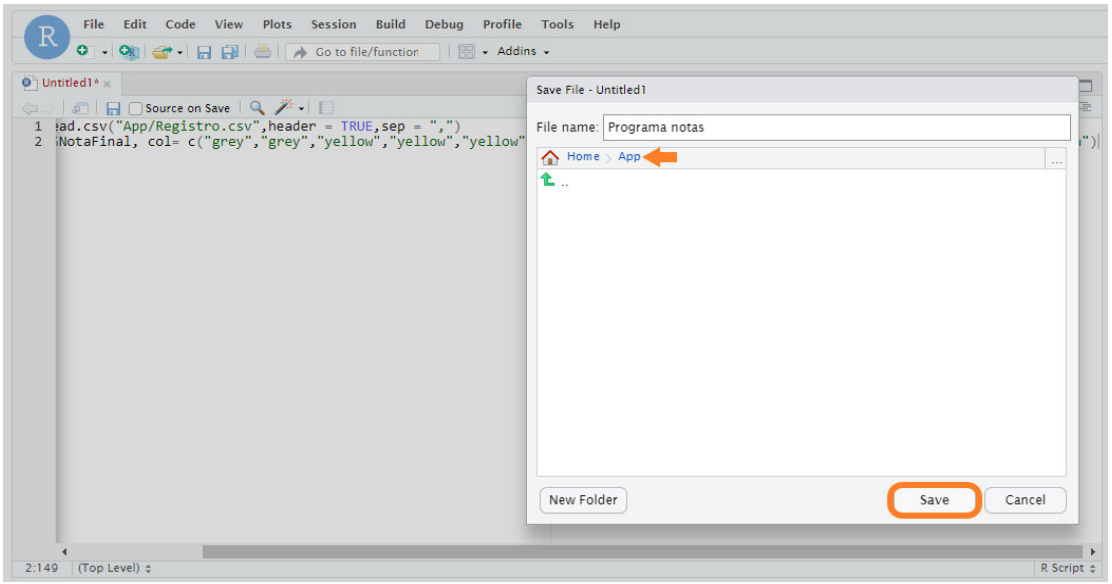

Figura 3.59 Grabar archivo R *Script*

Para verificar se debe observar en el panel de ficheros, que se encuentren los dos archivos creados dentro de la carpeta "App", nótese en la Figura 3.60.

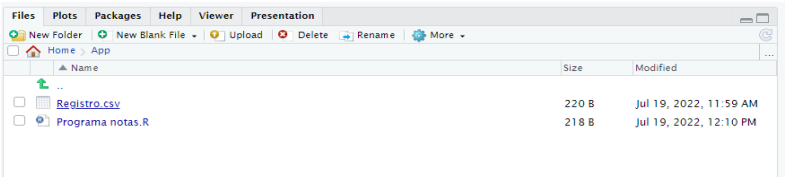

Figura 3.60 Verificación archivos subidos

En la Figura 3.61 se demuestra la ejecución del *Script* línea por línea presionando el botón "*Run*".

Figura 3.61 Ejecución de R *Script*

En la Figura 3.62 se comprueba la correcta ejecución del *Script* mediante el despliegue del histograma en forma gráfica en el panel de ficheros dentro del menú "*Plots*".

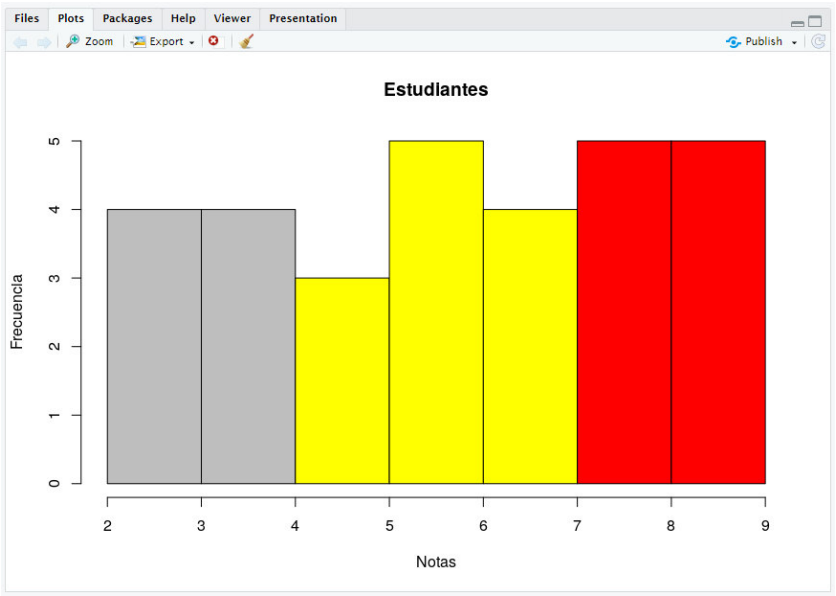

Figura 3.62 Histograma del *Script*

Esto evidencia el correcto funcionamiento del contenedor con el lenguaje interpretado R, ya que se logró trabajar con el mismo creando a satisfacción un ejemplo escrito en R.

4 CONCLUSIONES

- Los contenedores creados en este proyecto se pueden trasladar y distribuir entre múltiples proveedores de nube, actuando de la misma forma en cualquier lugar que se ejecute el contenedor. Esto disminuye los errores, tanto del ser humano como del ambiente de desarrollo y ejecución.
- Los contenedores remplazan a los entornos virtuales, como por ejemplo el entorno "venv" ya que encapsulan la aplicación, librerías y el lenguaje con sus respectivas versiones pudiendo tener en diferentes contenedores varias aplicaciones con diferentes versiones de lenguajes.
- Se concluye que la diferencia más importante entre los contenedores y las VM es que mientras que el hipervisor emula todo el *hardware* del servidor en las VM, los contenedores únicamente abstraen el núcleo del SO. Esto significa que casi siempre los gastos generales se reducen notablemente, con el sistema encapsulado en una representación pequeña y ordenada.
- *Docker* está escrito en el lenguaje Go y usa algunas capacidades disponibles en el *Kernel* de Linux. Además, *Docker* trae varias primicias al mundo de los contenedores. *Docker* hace que los contenedores sean más fáciles y seguros de implementar que LXC y Podman. Además, ayuda a instaurar un consenso en toda la industria y estandariza el formato del contenedor. Por otro lado, se encuentra instalado por defecto en la mayoría de los proveedores de nube.
- "Docker Hub" es el repositorio oficial para compartir y gestionar imágenes de contenedores de *Docker* y se ofrece como un modelo de *software* como servicio. El hospedaje de imágenes públicas es gratis; sin embargo, el hospedaje de imágenes privadas tiene un costo. Además, *Docker* vende una versión de "Docker Hub" *Enterprise* que permite a las organizaciones de gran tamaño tener un repositorio de *Docker* detrás de su *firewall* empresarial.
- De las herramientas DevOps analizadas se concluye que al momento es mejor usar *Docker* para la construcción de los contenedores, ya que su competidor más cercano Podman todavía tiene errores que no han sido corregidos y algunas

de sus características en las que sobresale en realidad no representan una ventaja como por ejemplo la seguridad. Con Podman se puede arrancar contenedores sin ser usuario *root*, pero si no es usuario *root* no se tiene acceso a algunos servicios importantes de la máquina como por ejemplo el de red, perdiéndose comunicación con otros contenedores, esto pasa a ser una gran desventaja.

- El proveedor de nube GCP brinda una gran variedad de servicios para guardar, preservar, ofrecer y analizar datos. Estos servicios en la nube de GCP cuentan con un contorno de nube seguro para los datos, donde se pueden realizar diferentes operaciones y modificaciones sobre los mismos.

- El proveedor de nube GCP tiene mayor apertura para compartir información con otras nubes, lo que permite distribuir información en diferentes nubes y no estar atado a un solo proveedor, debido a que la tendencia en el mercado en la actualidad es tener información en varias nubes más no en una sola.

- Para el desarrollo de este proyecto, la nube de GCP fue la más idónea debido a que no impone una condición de continuar con el servicio después de haber terminado el periodo gratuito; además, tiene un conjunto de herramientas que permiten tener un ambiente de desarrollo completo dentro de la nube para determinados lenguajes como *Python*, Java, etc.

- Los pasos que se siguieron para la implementación de los dos contenedores de *Python* y R respectivamente, se los puede usar para la implementación con otros lenguajes de programación interpretados, debido a que dichos pasos están descritos de forma entendible y detallada.

- Este proyecto será de gran ayuda para los desarrolladores de *software* que usan de *Python*, debido a que les proporciona el entorno de desarrollo listo para comenzar a programar de forma rápida, todo como un contenedor, esto permitirá que entregue la aplicación de forma oportuna y segura a su cliente.

- La implementación del contenedor con el lenguaje interpretado *Python* puede ser extendido con el uso del servicio "*Cloud Build*" que permitirá modificar y añadir librerías necesarias para desarrollar una aplicación específica y reconstruir la imagen del contenedor dentro la nube de GCP.

- El contenedor de *Python* utiliza un archivo "Dockerfile" que permitió agrupar una aplicación ejemplo, el lenguaje y el editor de código llamado "Vim", que se lo

puede usar para un entorno de enseñanza- aprendizaje del lenguaje *Python*; sin requerir la instalación de nada adicional.

- El contendor con el lenguaje interpretado R tiene un ambiente de desarrollo versátil y completo, que se lo puede utilizar para el análisis de datos a nivel empresarial o investigativo. Con la ventaja de ser un contenedor donde no se requiere de la instalación de todo el sistema operativo ahorrando recursos económicos optimizando el rendimiento.
- Mediante el ejemplo escrito en R y aplicado en el entorno de desarrollo llamado "RStudio" se corroboró su correcto funcionamiento, además este entorno permite realizar varias actividades importantes como el análisis de datos, interacción con aplicaciones, reporte de documentos, gráficos, etc. Esto conlleva a que en este contenedor se puedan aplicar dichas funcionalidades en la elaboración de la minería de datos.

5 RECOMENDACIONES

- Previo a la construcción de los contenedores de *Python* y R, se recomienda probar de manera local las imágenes de los lenguajes de programación interpretados a usar, obteniéndolas del repositorio digital de "Docker Hub".
- Se recomienda usar *Docker* con Kubernetes para la implementación de varios contenedores en aplicaciones grandes con microservicios porque es allí donde se ejecutan cientos de aplicaciones en contenedores en producción.
- Es recomendable utilizar el servicio de *Cloud Run* para implementar el contenedor con el lenguaje interpretado R, debido a que con dicho servicio se puede ejecutar el contenedor mediante solicitudes HTTP y HTTPS.
- Se recomienda verificar en qué número de puerto está expuesta la aplicación dentro de la imagen del contenedor, ya que si no se conoce el puerto no se puede mapear al puerto de la máquina física, en donde se va a correr la imagen; y si se ingresa un puerto erróneo no se levantará el servicio que proporciona la imagen.
- Se recomienda usar varios proveedores de nube para desplegar aplicaciones críticas que requieran de un funcionamiento permanente, ya que en el desarrollo de este proyecto se evidenció que el servicio de *Cloud Run* funcionó con intermitencias durante un día completo.

- Se recomienda crear los contenedores con la herramienta *Docker*, ya que es una plataforma mucho más madura en cuanto a la implementación de contenedores con respecto a otras herramientas, esto garantiza que muchos errores de esta se hayan corregido y se tenga una solución rápida y efectiva. Además, esta herramienta se encuentra por defecto instalada en la nube de GCP.
- Una vez elegida y descargada la plantilla de una imagen de contenedor del repositorio digital de "Docker Hub", se recomienda guardar dicha plantilla dentro del usuario creado en este repositorio para poderla reutilizar, ya que las imágenes que se encuentran alojadas en este repositorio pueden ser eliminadas o modificadas por el usuario que subió dichas imágenes.
- Se recomienda al usuario estar en constante actualización sobre la documentación y el funcionamiento de la plataforma de nube a utilizar, esto debido a que los proveedores de nube cada cierto tiempo actualizan sus interfaces de usuario y crean nuevos servicios.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] S. R. J. B. N. Buchanan, Dentro de los contenedores Docker, Berkeley, CA: Apress, 2020.

[2] «Coursera,» [En línea]. Available: https://www.coursera.org/learn/introduccion-a-contenedores-con-docker/lecture/vYbvl/introduccion-a-los-contenedores. [Último acceso: 17 06 2022].

[3] «Introducción a Contenedores y Docker,» [En línea]. Available: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/container-docker-introduction/. [Último acceso: 17 06 2022].

[4] «Contenedores frente a Maquinas Virtuales,» [En línea]. Available: https://www.ibm.com/cloud/blog/containers-vs-vms. [Último acceso: 18 06 2022].

[5] «TechTarget,» [En línea]. Available: https://www.techtarget.com/searchsecurity/feature/What-are-cloud-containers-and-how-do-they-work. [Último acceso: 14 07 2022].

[6] «freeCodeCamp,» 10 01 2020. [En línea]. Available: https://www.freecodecamp.org/news/compiled-versus-interpreted-languages/. [Último acceso: 20 06 2022].

[7] R. G. Duque, *Python* para todos, España.

[8] «Tokio School,» 07 03 2022. [En línea]. Available: https://www.tokioschool.com/noticias/caracteristicas-principales-de-*Python*/. [Último acceso: 20 06 2022].

[9] S. D. Quintero, Aprende *Python*, 2022.

[10] D. Smith, Una introducción a R, R-proyecto, 1992.

[11] J. M. L. d. Guevara, Fundamentos de Programacion en Java, Madrid : EME.

[12] E. G. C. Castillo, Desarrollando soluciones con Java, Lima: MACRO, 2011.

[13] «What is DevOps,» [En línea]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-devops/#devops-overview. [Último acceso: 25 07 2022].

[14] «Docker,» [En línea]. Available: https://docs.docker.com/get-started/overview/. [Último acceso: 23 06 2022].

[15] E. Kahuha, «EARTHLY,» 18 02 2022. [En línea]. Available: https://earthly.dev/blog/lxc-vs-docker/. [Último acceso: 23 06 2022].

[16] «Linux Containers,» [En línea]. Available: https://linuxcontainers.org/. [Último acceso: 23 06 2022].

[17] «Podman,» Sphinx, 2019. [En línea]. Available: https://docs.podman.io/en/latest/. [Último acceso: 26 06 2022].

[18] M. Aleksic, «phoenixNAP,» 03 03 2022. [En línea]. Available: https://phoenixnap.com/kb/podman-vs-docker. [Último acceso: 26 06 2022].

[19] L. S. Mazin Yousif, Cloud Computing, Viena, Austria: ICST, 2012.

[20] C. Preimesberger, «eWeek,» 15 03 2021. [En línea]. Available: https://www.eweek.com/cloud/aws-vs-google-cloud-platform/. [Último acceso: 30 06 2022].

[21] «Google Cloud,» [En línea]. Available: https://cloud.google.com/container-registry/docs/overview?hl=es-419. [Último acceso: 30 06 2022].

[22] S. Wickramasinghe, «bmc,» 2005. [En línea]. Available: https://www.bmc.com/blogs/aws-vs-azure-vs-google-cloud-platforms/. [Último acceso: 01 07 2022].

[23] «AWS,» [En línea]. Available: https://aws.amazon.com/. [Último acceso: 30 06 2022].

[24] «Azure,» [En línea]. Available: https://azure.microsoft.com/. [Último acceso: 30 06 2022].

[25] «IntelliPaat,» [En línea]. Available: https://intellipaat.com/blog/aws-vs-azure-vs-google-cloud/. [Último acceso: 04 07 2022].

[26] «Javatpoint,» [En línea]. Available: https://www.javatpoint.com/aws-vs-azure-vs-google-cloud-platform. [Último acceso: 04 07 2022].

[27] «Veritis,» 2022. [En línea]. Available: https://www.veritis.com/blog/aws-vs-azure-vs-gcp-the-cloud-platform-of-your-choice/#:~:text=Their%20respective%20cloud%20platforms%2C%20AWS,security%20and%20compliance%2C%20among%20others.. [Último acceso: 04 07 2022].

7 ANEXOS

ANEXO I: Certificado de Originalidad

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 26 de Agosto de 2022

De mi consideración:

Yo, GABRIELA KATHERINE CEVALLOS SALAZAR, en calidad de Director del Trabajo de Integración Curricular titulado IMPLEMENTACION DE UN LENGUAJE DE PROGRAMACIÓN INTERPRETADO MEDIANTE CONTENEDORES ALOJADOS EN LA NUBE elaborado por el estudiante ALEX ROBERTO RIVADENEIRA AGUALONGO de la carrera en TECNÓLOGÍA SUPERIOR EN REDES Y TELECOMUNICACIONES, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 9%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el link del informe generado por la herramienta Turnitin.

https://epnecuador-my.sharepoint.com/:f:/g/personal/gabriela_cevalloss_epn_edu_ec/EgCsEA0vdJ5HkR6ix13CTIYBkK2O5d01Y15k86-Mmmyr1w?e=WcI265

Atentamente,

Gabriela Cevallos Salazar

Docente

Escuela de Formación de Tecnólogos

ANEXO II: Enlaces

https://www.youtube.com/watch?v=t5PeMcAF6xY

Anexo II.I Código QR y enlace correspondiente al video del desarrollo y funcionamiento de los contenedores de *Python* y R.

ANEXO III: Códigos Fuente

Anexo III.I"Dockerfile"

```
FROM Python:3.9-slim-buster

WORKDIR /usr/src/app

RUN apt-get update && apt-get install -y vim

COPY app .

CMD [ "Python", "ejemplo.py" ]
```

Anexo III.II Código ejemplo del lenguaje *Python*

```
nombre:str = input("Ingrese su nombre: ")

if nombre.lower() == "alex":

    print("Tesis de Alex Rivadeneira")

else:

    print("Bienvenido "+nombre.upper())
```

Anexo III.III Código ejemplo del lenguaje R

```
registronotas <- read.csv("App/Registro.csv",header = TRUE,sep = ",")

hist(registronotas$NotaFinal, col= c("grey","grey","yellow","yellow","yellow","red","red"),
main = "Estudiantes", xlab= "Notas", ylab= "Frecuencia")
```