



UNIVERSIDAD DE GRANADA
MÁSTER DE CIENCIA DE DATOS E INGENIERÍA DE
COMPUTADORES
CURSO ACADÉMICO 2019-2020
BIG DATA 1
CLOUD COMPUTING

Desarrollo y despliegue de contenedores

*Documentación del proceso seguido para el desarrollo y
despliegue de contenedores en una arquitectura local y en la
nube.*

Nicolás Cubero

12 de Abril de 2020

Índice

Índice de figuras	4
1. Introducción	5
2. Contenedor sgdb	5
2.1. Instalación en la máquina virtual	5
2.1.1. Ejecución de operaciones de prueba	9
2.2. Instalación en Azure	13
2.2.1. Ejecución de operaciones de prueba	22
3. Contenedor cdpypthon	24
3.1. Instalación en la máquina virtual	24
3.1.1. Ejemplo de ejecución	28
3.2. Instalación en Azure	30
3.2.1. Ejemplo de ejecución	36
4. Contenedor cdr	37
4.1. Instalación en la máquina virtual	38
4.1.1. Ejemplo de ejecución	41
4.2. Instalación en Azure	43
4.2.1. Ejemplo de ejecución	48

Índice de figuras

1.	Captura de pantalla de la página sobre la imagen mysql de docker hub.	6
2.	Captura de la shell remota donde se refleja la descarga de la imagen mysql la cual es descargada desde el repositorio Docker Hub.	7
3.	Captura de la shell remota donde se refleja el arranque del contenedor con servidor MySQL.	8
4.	Captura de la shell donde se el acceso a la terminal de MySQL del servidor instalado en el contenedor.	8
5.	Captura de la shell donde se refleja el proceso de detención de la ejecución del demonio asociado al servidor MySQL y el salvado de una imagen del mismo.	9
6.	Captura de la shell donde se refleja el proceso de ejecución del script Python para la creación de la tabla Empleado y la inserción de datos sobre la misma en el servidor del contenedor.	11
7.	Captura de la shell del sgdb donde se muestra el resultado de la consulta con todos los datos insertados por el script Python de inserción de prueba.	12
8.	Captura de la ejecución del script Python de consulta de los datos de la tabla Empleado almacenados en el sistema gestor de base de datos del contenedor sgdb.	13
9.	Captura de la configuración establecida en el panel Datos básicos en el proceso de creación del Registro de contenedor.	15
10.	Captura del panel Revisar y crear, donde se muestra la validación y resumen de la configuración establecida.	16
11.	Captura que refleja éxito en la creación del registro de contenedor.	17
12.	Captura que refleja éxito en la creación del registro de contenedor.	18
13.	Captura que refleja el etiquetado de la imagen del contenedor y su subida al registro de contenedor.	18
14.	Captura que refleja la configuración en el panel Datos básicos para la creación de la instancia sgdb.	19
15.	Captura que refleja la configuración en el panel Redes donde se ha añadido el puerto 3306.	20
16.	Captura que refleja la información mostrada en el panel Revisar y crear.	21
17.	Captura que refleja éxito en la creación de la instancia de sgdb.	22

18.	Captura que refleja la edición de la dirección IP configurada en el script cliente de inserción.	23
19.	Captura que refleja la ejecución exitosa del script python de inserción sobre la base de datos del contenedor sgdb montado en Azure.	23
20.	Captura que refleja la ejecución exitosa del script python de para la consulta de los datos insertados en la tabla Empelado en el contenedor sgdb montado en Azure.	24
21.	Captura de la página de docker hub sobre la imagen del contenedor de Python.	25
22.	Captura de la shell donde se refleja el proceso de descarga de la imagen de Python.	25
23.	Captura de la shell donde se muestra el inicio de una terminal interactiva en un contenedor de Python 3.8.	26
24.	Captura de la shell donde se muestra la ejecución de los comandos de instalación de las librerías de Python mediante el gestor pip.	27
25.	Captura de la shell donde se refleja la ejecución del commit con la imagen cdpthon.	27
26.	Captura de la shell donde se refleja el proceso de copia de la máquina hospedadora a la máquina virtual.	29
27.	Captura de la shell donde se refleja la ejecución del <i>script</i> Python de clustering SVM.	30
28.	Copia del fichero con el Dockerfile de cdpthon a la máquina virtual.	31
29.	Captura de la shell donde se muestra el proceso de etiquetado y subida de la imagen cdpthon al registro de contenedor. . .	32
30.	Captura que refleja la configuración en el panel Datos básicos para la creación de la instancia cdpthon.	33
31.	Captura que refleja la configuración en el panel Redes donde se ha añadido el puerto 22 para permitir la conexión por ssh. .	34
32.	Captura que refleja la información mostrada en el panel Revisar y crear.	35
33.	Captura que refleja la instancia creada exitosamente y funcionando.	36
34.	Captura que refleja el acceso a la instancia de contenedor por ssh mediante el usuario administrador.	36
35.	Captura que refleja le ejecución del script Python que ejecuta un clasificador SVM sobre el dataset iris en la instancia de contenedor.	37
36.	Captura de la página de la imagen r-base en docker hub. . . .	38

37.	Captura de la shell donde se refleja el proceso de descarga de la imagen de r-base y la salida obtenida.	38
38.	Captura de la terminal de R del contenedor ejecutado.	39
39.	Captura de la terminal de bash del contenedor cdr donde se refleja la instalación de los paquetes libcurl4-openssl-dev xml2.	40
40.	Captura donde se refleja el proceso de copia del script de ejemplo a la máquina virtual.	42
41.	Captura donde se refleja el proceso de inicio del contenedor cdr y la ejecución del script de prueba.	42
42.	Captura donde se refleja el etiquetado y subida de la imagen del contenedor cdr al registro de contenedor de Azure.	44
43.	Captura que refleja la configuración en el panel Datos básicos para la creación de la instancia cdr.	45
44.	Captura que refleja la configuración en el panel Redes donde se ha añadido el puerto 22 para permitir la conexión por ssh.	46
45.	Captura que refleja la información mostrada en el panel Revisar y crear.	47
46.	Captura que refleja la instancia creada exitosamente y funcionando.	48
47.	Captura que refleja el acceso a la instancia de contenedor por ssh mediante el usuario administrador.	48
48.	Captura que refleja la ejecución del script R que ejecuta un clustering KMeans sobre el conjunto de datos de iris (excluyendo la etiqueta) y muestra las medidas de bondad de este clustering.	49

1. Introducción

La computación en la nube o **Cloud Computing** ofrece un conjunto muy diverso, flexible y escalable de servicios de cómputo, almacenamiento, procesamiento, desarrollo software, etc, que, ofrecidos bajo demanda y a través de la red, abre un nuevo horizonte de oportunidades aprovechables en multitud de procesos de negocio, entre ellos el *Big Data*.

Cloud Computing ofrece al *Big Data* las arquitecturas de cómputo y almacenamiento requeridas por los procesos de *Big Data* y que, por lo general, son tan exigentes que difícilmente pueden ser proporcionadas por arquitecturas locales.

Uno de los servicios más destacados del *Cloud Computing* son los **contenedores**. Los contenedores permiten empaquetar, transportar y desplegar de forma ligera e independiente, todo el entorno de ejecución de un software, de modo que pueda ser desplegado rápidamente en una arquitectura virtual de altas prestaciones para la ejecución de los procesos de *Big Data*.

En este proyecto, se tratará la generación y despliegue de diferentes contenedores en una máquina virtual con Ubuntu Server y en la plataforma *Azure* de Microsoft.

Más concretamente, se desplegaran tres contenedores: un contenedor con un gestor de bases de datos basado en *MySQL* y dos contenedores basados en *Python* y *R*, dotados de todas las librerías necesarias para la aplicación de algoritmos de ciencia de datos.

2. Contenedor sgdb

Se tratará de generar un contenedor con *MySQL* como sistema de gestión de base de datos y se configurará para constituir en el mismo un servidor de base datos que permita la ejecución de consultas de forma remota.

2.1. Instalación en la máquina virtual

Primeramente, se tratará el despliegue en una máquina virtual con *Ubuntu Server 18* con el gestor de paquetes **Docker**, a la cual se le ha configurado su interfaz de red en **modo adaptador puente** para permitir su conexión a la red local como un equipo independiente.

La dirección IP de esta máquina virtual en la red local es 192.168.1.41.

Accedemos a la *shell* de la máquina virtual y comenzamos la instalación del contenedor **sgdb**, que se generará a partir de la imagen **mysql** del repositorio del sitio web *docker hub* ¹.

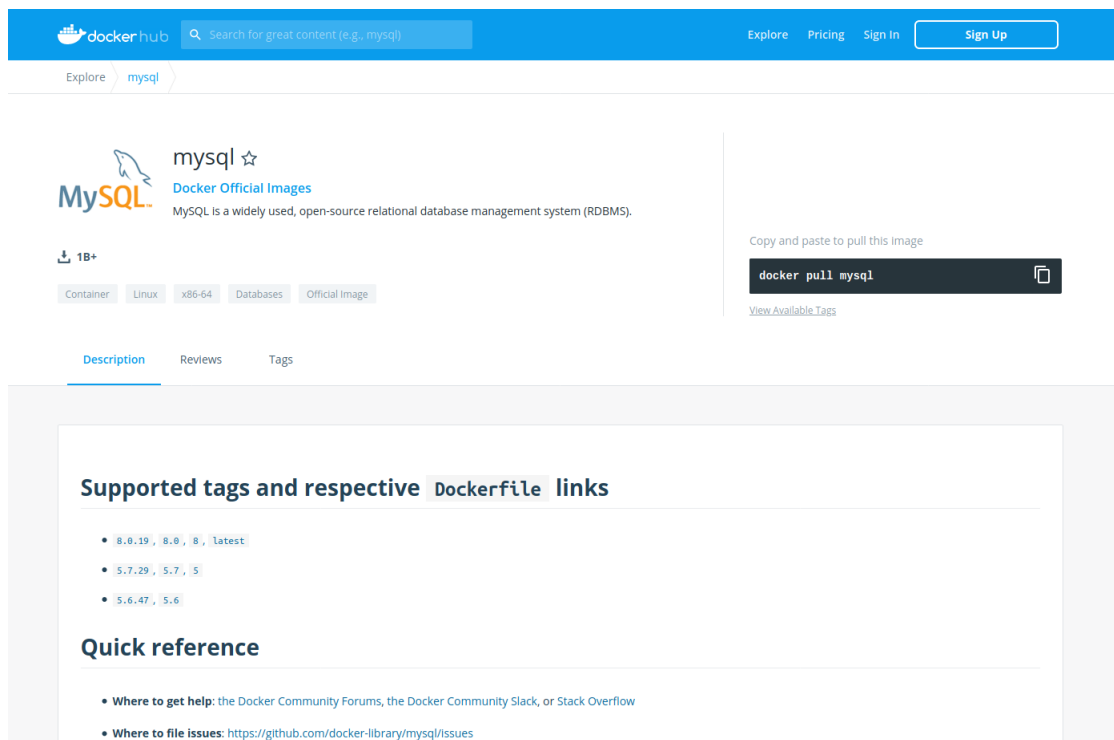


Figura 1: Captura de pantalla de la página sobre la imagen mysql de docker hub.

Se descarga esta imagen mediante el siguiente comando:

```
$ sudo docker pull mysql:latest
```

¹Enlace a la imagen de mysql en el repositorio de docker hub:
https://hub.docker.com/_/mysql

```
nico@nicoserver:~$ docker pull mysql/mysql-server:latest
Got permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock: Post http://%2Fvar%2Frun%2Fdocker.sock/v1.40/images/create?fromImage=mysql%2Fmysql-server&tag=latest: dial unix /var/run/docker.sock: connect: permission denied
nico@nicoserver:~$ sudo docker pull mysql/mysql-server:latest
latest: Pulling from mysql/mysql-server
c7127dfa6d78: Pull complete
530b30ab10d9: Pull complete
59c6388c2493: Pull complete
cca3f8362bb0: Pull complete
Digest: sha256:7cd104d6ff11f7e6a16087f88b1ce538bcb0126c048a60cd28632e7cf3dbe1b7
Status: Downloaded newer image for mysql/mysql-server:latest
docker.io/mysql/mysql-server:latest
nico@nicoserver:~$ _
```

Figura 2: Captura de la shell remota donde se refleja la descarga de la imagen mysql la cual es descargada desde el repositorio Docker Hub.

Una vez descargada la imagen, procedemos a iniciar la ejecución del contenedor en modo demonio, estableciendo el puerto 3306 como puerto de escucha en el servidor hospedado en el contenedor, y lo mapeamos también al puerto 3306 de la máquina hospedadora.

Por su parte, configuramos las variables de entorno para configurar en el contenedor lo siguiente:

- Contraseña del usuario *root*: *tarara_blanca82*.
- Crear la base de datos *Gestion_trabajadores*.
- Crear al usuario *administrador* con la contraseña: *granada_lorca32* y otorgar permisos totales sobre *Gestion_trabajadores*.
- Configurar el modo de autenticación al **modo nativo con contraseña** puesto que el cliente desde el cual se ejecutarán las inserciones y consultas de prueba, no soporta el modo de autenticación con contraseña *SHA2* en caché.

Todo esto se ejecuta por medio del siguiente comando:

```
$ sudo docker run -p 3306:3306 --name sgdb
-e MYSQL_ROOT_PASSWORD=tarara_blanca82
-e MYSQL_USER=administrador
-e MYSQL_PASSWORD=granada_lorca32
-e MYSQL_DATABASE=Gestion_trabajadores
-d mysql:latest --default-authentication-plugin=mysql_native_password
```



```
nico@nicoserver:~$ sudo docker run -p 3306:3306 --name sgdb -e MYSQL_ROOT_PASSWORD=tarana_blanca82 -e MYSQL_USER=administrador -e MYSQL_PASSWORD=granada_lorca82 -e MYSQL_DATABASE=Gestion_trabajadores -d mysql --default-authentication-plugin=mysql_native_password
c7e23f82d5d05c2543de1a21d47df0a8f16c27f42a402fad9ba6fd39ba5ff724
nico@nicoserver:~$ sudo docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
c7e23f82d5d0	mysql	"docker-entrypoint.s..."	6 seconds ago	Up 4 seconds
0.0.0.0:3306->3306/tcp, 33060/tcp		sgdb		

Figura 3: Captura de la shell remota donde se refleja el arranque del contenedor con servidor MySQL.

Accedemos a la consola de la base de datos con el usuario `administrador` para comprobar que el contenedor se ha arrancado con éxito y que es accesible:

```
$ sudo docker exec -it sgdb mysql -u administrador -p
```

```
nico@nicoserver:~$ sudo docker exec -it sgdb mysql -u administrador -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 10
Server version: 8.0.19 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE Gestion_trabajadores;
Database changed
mysql>
```

Figura 4: Captura de la shell donde se el acceso a la terminal de MySQL del servidor instalado en el contenedor.

Para acabar, **almacenamos la imagen del contenedor resultante**, deteniendo en primer lugar su ejecución y, realizando un *commit* del mismo:

```
$ sudo docker stop sgdb
$ sudo docker commit sgdb
```

```
nico@nicoserver:~$ sudo docker stop sgdb;
[sudo] password for nico:
sgdb
nico@nicoserver:~$ sudo docker commit sgdb;
sha256:2923a653d3613444d71984494a1606f5dc53c71885e2de0e5d27ea5b8ea0abee
nico@nicoserver:~$
```

Figura 5: Captura de la shell donde se refleja el proceso de detención de la ejecución del demonio asociado al servidor MySQL y el salvado de una imagen del mismo.

Por último, el fichero *Dockerfile* asociado a la instalación de este servidor es el que se expone a continuación. Si bien mediante este fichero es posible llevar a cabo la construcción de este contenedor, evitando la construcción manual efectuada en esta sección, se ha preferido realizar primeramente la construcción de los contenedores de forma manual y hacer uso del *Dockerfile* ya en el despliegue en la plataforma *Azure*.

```
1 FROM mysql:latest
2 LABEL maintainer Nicolás Cubero Torres (nicocu97@correo.ugr.es)
3
4 # Habilitar el modo de autenticación al nativo con contraseña
5 CMD ["mysqld", "--default-authentication-plugin=mysql_native_password"]
6
7 # Establecer la contraseña de roor
8 ENV MYSQL_ROOT_PASSWORD "tarara_blanca82"
9
10 # Crear base de datos
11 ENV MYSQL_DATABASE "Gestion_trabajadores"
12
13 # Crear usuario datcom y establecer su contraseña
14 ENV MYSQL_USER "administrador"
15 ENV MYSQL_PASSWORD "granada_lorca32"
16
17 # Iniciar el servidor de MySQL
18 EXPOSE 3306
```

Script 1: Archivo Dockerfile que define la instalación del contenedor sgdb

2.1.1. Ejecución de operaciones de prueba

Para comprobar que el contenedor *sgdb* instalado admite conexiones desde clientes externos y permite, asimismo, la ejecución de operaciones de inserción de datos y consulta, desde *hosts* remotos, ejecutaremos desde la máquina hospedadora (con dirección IP 192.168.1.37 en la red local), dos *scripts* en Python que actuarán como clientes del servidor, y llevarán a cabo operaciones de inserción y consulta de prueba simulando una conexión al contenedor instalado en el *host* virtualizado (que cuenta con la dirección IP propia

192.168.1.41 en la red local gracias a su interfaz de red configurada en modo puente).

1. Operación de inserción:

Ejecutamos desde la máquina hospedadora el siguiente *script* Python que conecta con el servidor y ejecuta la creación de la tabla *Empleado*, además de un conjunto de inserciones:

```
1  # -*- coding: utf-8 -*-
2
3  ## Librerías importadas
4  import sys
5  import mysql.connector
6
7
8  ## Conexión con la base de datos
9  try:
10     con = mysql.connector.connect(host='192.168.1.41', user='
        administrador',
11                                   passwd='granada_lorca32',
12                                   database='Gestion_trabajadores',
13                                   auth_plugin='mysql_native_password')
14 except Exception as e:
15     print('Error al tratar de conectar a la base de datos: ', str(e),
16           file=sys.stderr)
17     exit(-1)
18
19 print('Conectado con éxito a {} en el puerto {} con usuario "{}".\nModo
        SQL: {}'.format(con.server_host, con.server_port, con.user, con.
        sql_mode))
20 print()
21
22 cursor_db = con.cursor()
23
24 ## Crear la tabla Alumno si no existe
25 cursor_db.execute(
26     """
27 CREATE TABLE IF NOT EXISTS Empleado (
28     username VARCHAR(20) PRIMARY KEY,
29     dni CHAR(9) NOT NULL,
30     nombre VARCHAR(100) NOT NULL,
31     apellidos VARCHAR(100) NOT NULL,
32     fecha_nacimiento DATE NOT NULL,
33     sexo TINYINT NOT NULL, #0: Hombre, 1: Mujer
34     direccion VARCHAR(100) NOT NULL,
35     ciudad VARCHAR(1000) NOT NULL,
36     pais VARCHAR(1000) NOT NULL,
37     fecha_registro DATETIME NOT NULL
38 ) CHARACTER SET utf8 COLLATE utf8_general_ci;""")
39
40 ## Insertar datos en la tabla Alumno
41 cursor_db.execute("INSERT INTO Empleado VALUES('paloflores', '87210436Y
        ', 'Paloma','Flores',DATE('1996-11-27'),1,'Calle Bolivar n° 23','La
        Colorada', 'México', NOW());")
42 cursor_db.execute("INSERT INTO Empleado VALUES('jipef92', '12568765M','
        Francisco','Jiménez Pérez',DATE('1992-01-08'),0,'Calle La Alcachofa
        n° 2','Córdoba', 'España', NOW());")
```

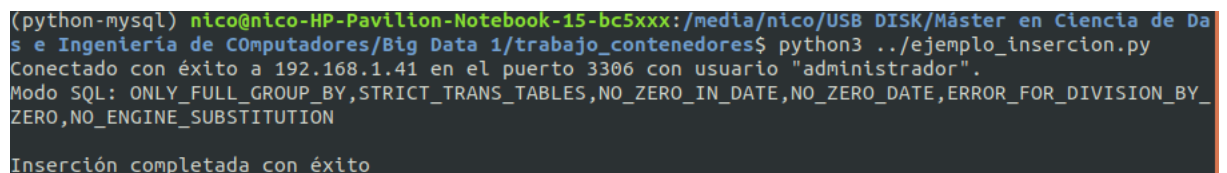
```

43 cursor_db.execute("INSERT INTO Empleado VALUES('pedriquiroa', '35868795
    S','Pedro','Ramírez Quiroa',DATE ('1996-04-07'),0,'Calle La Rambla n
    ° 5','Barcelona', 'España', NOW());")
44 cursor_db.execute("INSERT INTO Empleado VALUES('caspaola', '04940436M',
    'Paola','Catelli Belgrano',DATE ('1997-07-08'),1,'Rue Almeghino n°
    3','Volterra', 'Italia', NOW());");
45 cursor_db.execute("INSERT INTO Empleado VALUES('dasanchez', '89138565A
    ','David','Sánchez Montenegro', DATE ('1997-05-15'), 0,'Calle Ramón
    y Cajal n° 1', 'Málaga', 'España', NOW());")
46 cursor_db.execute("INSERT INTO Empleado VALUES('paquitagallego',
    '57824547E', 'Francisca','Gallego de Dios',DATE ('1990-08-06'),1,'
    Calle Waldo n° 17','Córdoba', 'Argentina', NOW());")
47 cursor_db.execute("INSERT INTO Empleado VALUES('cruzyelena', '11324956A
    ','Elena','Cruz',DATE ('1995-03-11'),1,'Plaza Díaz n° 12','La
    Hanana', 'Cuba', NOW());")
48
49 # Hacer commit de los cambios
50 con.commit()
51
52 # Cerrar la conexión y el cursor
53 cursor_db.close()
54 con.close()
55
56 print('Inserción completada con éxito')

```

Script 2: Script Python que simula la creación de la tabla Empleado y realiza algunas inserciones en la base de datos del contenedor sgdb

La ejecución del *script* en la máquina hospedadora se refleja en la siguiente imagen:



```

(python-mysql) nico@nico-HP-Pavilion-Notebook-15-bc5xxx:/media/nico/USB DISK/Máster en Ciencia de Da
s e Ingeniería de Computadores/Big Data 1/trabajo_contenedores$ python3 ../ejemplo_insercion.py
Conectado con éxito a 192.168.1.41 en el puerto 3306 con usuario "administrador".
Modo SQL: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_
ZERO,NO_ENGINE_SUBSTITUTION
Inserción completada con éxito

```

Figura 6: Captura de la shell donde se refleja el proceso de ejecución del script Python para la creación de la tabla Empleado y la inserción de datos sobre la misma en el servidor del contenedor.

Si accedemos a la terminal del sistema gestor de base de datos en el contenedor con el usuario Empleado y realizamos una consulta sobre todos los datos de la tabla Empleado, podemos observar nos devuelve todos los datos insertados por el *script*:

```
mysql> SELECT * FROM Empleado;
```

username	dni	nombre	apellidos	fecha_nacimiento	sexo	direccion
	ciudad	pais	fecha_registro			
caspaola	04940436M	Paola	Catelli Belgrano	1997-07-08	1	Rue Almegh
ino n# 3	Volterra	Italia	2020-04-06 17:15:53			
cruzyelena	11324956A	Elena	Cruz	1995-03-11	1	Plaza D#az
n# 12	La Habana	Cuba	2020-04-06 17:15:53			
dasanchez	89138565A	David	S#nchez Montenegro	1997-05-15	0	Calle Ram#
n y Cajal n# 1	M#laga	Espa#a	2020-04-06 17:15:53			
jipef92	12568765M	Francisco	Jim#nez P#rez	1992-01-08	0	Calle La A
lcachofa n# 2	C#rdoba	Espa#a	2020-04-06 17:15:53			
paloflores	87210436Y	Paloma	Flores	1996-11-27	1	Calle Boli
var n# 23	La Colorada	M#xico	2020-04-06 17:15:53			
paquitagallego	57824547E	Francisca	Gallego de Dios	1990-08-06	1	Calle Wald
b n# 17	C#rdoba	Argentina	2020-04-06 17:15:53			
pedriquiroa	35868795S	Pedro	Ram#rez Quiroa	1996-04-07	0	Calle La R
ambla n# 5	Barcelona	Espa#a	2020-04-06 17:15:53			

```

7 rows in set (0.00 sec)

mysql> _

```

Figura 7: Captura de la shell del sgdb donde se muestra el resultado de la consulta con todos los datos insertados por el script Python de inserción de prueba.

2. Operación de consulta:

Lanzamos ahora un segundo *script* Python desde la máquina hospedadora para la ejecución de una operación de consulta sobre el contenedor sgdb por la cual, se van a recuperar todos los datos insertados en el mismo con el anterior *script*:

```

1 # -*- coding: utf-8 -*-
2
3 ## Librerías importadas
4 import sys
5 import mysql.connector
6
7
8 ## Conexión con la base de datos
9 try:
10     con = mysql.connector.connect(host='192.168.1.41', user='
11                                     administrador',
12                                     passwd='granada_lorca32',
13                                     database='Gestion_trabajadores')
14 except Exception as e:
15     print('Error al tratar de conectar a la base de datos: ', str(e),
16           file=sys.stderr)
17     exit(-1)
18
19 print('Conectado con éxito a {} en el puerto {} con usuario "{}.\nModo
20       SQL: {}'.format(con.server_host, con.server_port, con.user, con.
21                       sql_mode))
22 print()

```

```

20 cursor_db = con.cursor()
21
22
23 ## Realizar consulta de los datos
24 try:
25     cursor_db.execute('SELECT * FROM Empleado;')
26     result = cursor_db.fetchall()
27
28     print('Usuario, DNI, Nombre, Apellidos, Fecha Nacimiento, Sexo,
29           'Dirección,\
30           ' Ciudad, País, Fecha de registro')
31     for alumno in result:
32         print(alumno)
33 except Exception as e:
34     print('No se pudo ejecutar la consulta: ', str(e))
35
36 ## Cierre de cursores y de conexión
37 cursor_db.close()
38 con.close()

```

Script 3: Script Python que simula la consulta de los datos introducidos en la tabla Empleado en el contenedor sgdb

La ejecución de este *script* sobre la máquina hospedadora nos permite recuperar existosamente todos los datos que fueron previamente insertados:

```

(python-mysql) nico@nico-HP-Pavilion-Notebook-15-bc5xxx:/media/nico/USB DISK/Máster en Ciencia de Datos e Ingeniería de Computadores/Big Data 1/trabajo_contenedores$ python3 ../ejemplo_consulta.py
Conectado con éxito a 192.168.1.41 en el puerto 3306 con usuario "administrador".
Modo SQL: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION

Usuario, DNI, Nombre, Apellidos, Fecha Nacimiento, Sexo, Dirección, Ciudad, País, Fecha de registro
('caspaola', '04940436H', 'Paola', 'Catelli Belgrano', datetime.date(1997, 7, 8), 1, 'Rue Almoghino n° 3', 'Volterra', 'Italia', datetime.datetime(2020, 4, 10, 0, 56, 48))
('cruz', '11324956A', 'Elena', 'Cruz', datetime.date(1995, 3, 11), 1, 'Plaza Diaz n° 12', 'La Habana', 'Cuba', datetime.datetime(2020, 4, 10, 0, 56, 48))
('diaz', '89133565A', 'David', 'Sanchez Montenegro', datetime.date(1997, 5, 15), 0, 'Calle Ramon y Cajal n° 1', 'Alicante', 'España', datetime.datetime(2020, 4, 10, 0, 56, 48))
('jimenez', '12568765H', 'Francisco', 'Jimenez Perez', datetime.date(1992, 11, 8), 0, 'Calle La Alcachofa n° 2', 'Córdoba', 'España', datetime.datetime(2020, 4, 10, 0, 56, 48))
('palomares', '87210436V', 'Paloma', 'Flores', datetime.date(1998, 11, 27), 1, 'Calle Bolivar n° 23', 'La Colorada', 'México', datetime.datetime(2020, 4, 10, 0, 56, 48))
(python-mysql) nico@nico-HP-Pavilion-Notebook-15-bc5xxx:/media/nico/USB DISK/Máster en Ciencia de Datos e Ingeniería de Computadores/Big Data 1/trabajo_contenedores$

```

Figura 8: Captura de la ejecución del script Python de consulta de los datos de la tabla Empleado almacenados en el sistema gestor de base de datos del contenedor sgdb.

2.2. Instalación en Azure

Ahora, instalaremos este contenedor en **Azure** y repetiremos la ejecución de las operaciones de inserción y consulta de prueba.

La instalación se llevará a cabo, mediante el fichero *Dockerfile* que se implementó previamente.

Para llevar a cabo este proceso, se debe de crear un **registro de contenedor** en **Azure** al cual se subirá la imagen generada a partir del *Dockerfile*.

Completada la subida de la imagen al **registro de contenedores**, se creará una **instancia de contenedor** en **Azure** con la imagen mediante la siguiente operación:

1. Primeramente, creamos un **registro de contenedores** en **Azure**, por lo que debemos acceder a la plataforma desde nuestra cuenta de usuario, buscar **Registro de contenedores** e iniciar la creación de un nuevo registro.
2. En el formulario de creación del nuevo **Registro de contenedor**, en **Datos básicos** asociamos este registro al **Grupo de recursos BigData1** (grupo de recursos creado para almacenar todos los recursos creados en los proyectos de esta asignatura).

Asociamos como **Nombre del Registro** nicocu97bigdata1, habilitamos el **uso de un usuario administrador** y, por simplicidad y para ahorrar costes establecemos **SKU** a básico:

Crear Registro de contenedor

[Datos básicos *](#) [Encryption](#) [Etiquetas](#) [Revisar y crear](#)

Azure Container Registry permite compilar, almacenar y administrar artefactos e imágenes de contenedor en un registro privado para todos los tipos de implementación de contenedor. Use registros de contenedor de Azure con sus canalizaciones de desarrollo e implementación de contenedores actuales. Use Azure Container Registry Tasks para compilar imágenes de contenedor en Azure a petición, o bien automatizar compilaciones desencadenadas por actualizaciones del código fuente, actualizaciones de la imagen base de un contenedor o temporizadores. [Más información](#)

Detalles del proyecto


Suscripción *	<div>Azure para estudiantes</div>
Grupo de recursos *	<div>BigData1</div> <div>Crear nuevo</div>

Detalles de instancia

Nombre del Registro *	<div>nicocu97bigdata1</div> <div>.azurecr.io</div>
Ubicación *	<div>(Europe) Oeste de Europa</div>
Usuario administrador * ⓘ	<div><div>Habilitar</div><div>Deshabilitar</div></div>
SKU * ⓘ	<div>Básico</div>

Figura 9: Captura de la configuración establecida en el panel Datos básicos en el proceso de creación del Registro de contenedor.

3. No se realiza ningún ajuste adicional y accedemos a la pestaña **Revisar y crear** donde el sistema validará que la configuración sea válida y nos mostrará un resumen de los parámetros establecidos para este contenedor:



Crear Registro de contenedor

✓ Validación superada

Datos básicos *

Encryption

Etiquetas

Revisar y crear

Detalles del registro

Datos básicos

Nombre del Registro	nicocu97bigdata1
Suscripción	Azure para estudiantes
Grupo de recursos	BigData1
Ubicación	Oeste de Europa
Usuario administrador	true
SKU	Basic

Encryption

Customer-Managed Key	Deshabilitado
Identity	Ninguno
Key Vault	Ninguno
Encryption key	Ninguno
Version	Ninguno

Crear

< Anterior

Siguiente >

Figura 10: Captura del panel Revisar y crear, donde se muestra la validación y resumen de la configuración establecida.

Seleccionamos **Crear** y tras un breve periodo, se completará la creación del **registro de contenedor**:

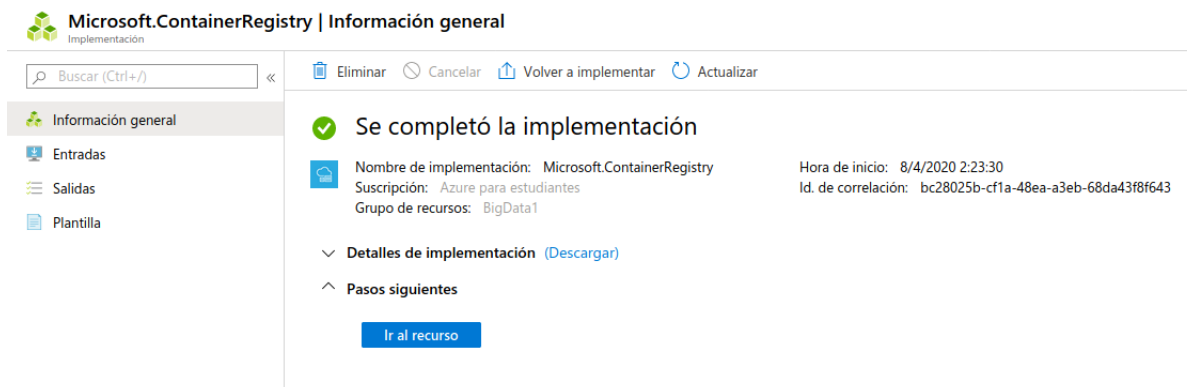


Figura 11: Captura que refleja éxito en la creación del registro de contenedor.

Azure ha asociado al **Registro de contenedor** a un servidor bajo el siguiente dominio: `nicocu97bigdata1.azurecr.io`

4. Para subir la imagen del contenedor, se requiere hacer uso del gestor un gestor de contenedores como **docker**, por lo que llevaremos a cabo la subida desde la máquina virtual que se usó en el epígrafe anterior y que ya lleva instalado este gestor.

Para evitar conflictos con los archivos *Dockerfile* del resto de contenedores que se van a desplegar en esta práctica, el fichero *Dockerfile* escrito para la generación de este contenedor, es copiado a un directorio que se le ha denominado *sgdb*. Este fichero es copiado a la máquina virtual haciendo uso de **scp**.

5. Una vez copiado el directorio con el fichero *Dockerfile*, construimos la imagen (se requiere haber eliminado tanto el contenedor como la imagen creados en el epígrafe anterior):

```
$ sudo docker build -t sgdb sgdb
```

6. Como paso previo a la subida, debemos **loguearnos** en el **registro de contenedor** desde el gestor *docker* de la máquina virtual y proporcionando la dirección del servidor en el que **Azure** ha alojado el **registro de contenedor**:

```
$ sudo docker login nicocu97bigdata1.azurecr.io
```

Nos identificamos con el nombre de usuario y contraseña establecidos (son asignados por Azure de forma automática y se puede consultar accediendo al recurso del **registro de contenedor**, en el botón **Claves de acceso** del panel **Configuración**).

```
nico@nicoserver:~$ sudo docker login nicocu97bigdata1.azurecr.io
Username: nicocu97bigdata1
Password:
WARNING! Your password will be stored unencrypted in /home/nico/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
nico@nicoserver:~$
```

Figura 12: Captura que refleja éxito en la creación del registro de contenedor.

7. Una vez logueados, etiquetamos la imagen para poder identificarla en el **registro de contenedor** y hacemos push hacia el mismo:

```
$ sudo docker tag sgdb nicocu97bigdata1.azurecr.io/sgdb
$ sudo docker push nicocu97bigdata1.azurecr.io/sgdb
```

```
nico@nicoserver:~$ sudo docker tag sgdb nicocu97bigdata1.azurecr.io/sgdb
nico@nicoserver:~$ sudo docker push nicocu97bigdata1.azurecr.io/sgdb
The push refers to repository [nicocu97bigdata1.azurecr.io/sgdb]
2ea3f5d1b2f3: Pushed
f1a617f37b0a: Pushed
dbfa2f8c95c3: Pushed
1ca405fbe05e: Pushing 10.56MB/411MB
d501f79e3989: Pushed
5de3946ea013: Pushed
5fce4d95d4af: Pushing 3.724MB/52.23MB
1ea6ef84dc3a: Pushed
ad160f341db9: Pushed
0c615b40cc37: Pushing 3.72MB/9.342MB
a9f6b7c7101b: Pushed
f2cb0ecef392: Pushing 3.248MB/69.21MB
```

Figura 13: Captura que refleja el etiquetado de la imagen del contenedor y su subida al registro de contenedor.

8. Una vez completada la subida, procedemos a crear una **instancia del contenedor** de la imagen que acabamos de subir:

Accedemos una vez más a la plataforma **Azure** desde nuestra cuenta e iniciamos la creación de una instancia de contenedor.

9. Desde el panel **Datos básicos**, nuevamente asociamos este recurso al grupo de recursos de **BigData1**. Nombramos este contenedor **sgdb** y seleccionamos cargar la imagen **sgdb** desde nuestro **registro de recursos**:

Crear instancia de contenedor

[Datos básicos](#) [Redes](#) [Opciones avanzadas](#) [Etiquetas](#) [Revisar y crear](#)

Azure Container Instances (ACI) le permite ejecutar contenedores en Azure de forma rápida y fácil, sin necesidad de administrar servidores o de tener que aprender a usar nuevas herramientas. ACI ofrece facturación por segundo para minimizar el costo de ejecución de los contenedores en la nube. [Más información acerca de Azure Container Instances](#)

Detalles del proyecto

Seleccione la suscripción para administrar recursos implementados y los costes. Use los grupos de recursos como carpetas para organizar y administrar todos los recursos.

Suscripción * ⓘ

Azure para estudiantes ▼

Grupo de recursos * ⓘ

BigData1 ▼

[Crear nuevo](#)

Detalles del contenedor

Nombre de contenedor * ⓘ

sgdb ✓

Región * ⓘ

(Europe) Oeste de Europa ▼

Origen de imagen * ⓘ

☐ Imágenes de inicio rápido

☒ Azure Container Registry

☐ Docker Hub u otro registro

Registro * ⓘ

nicocu97bigdata1 ▼

Imagen * ⓘ

sgdb ▼

Etiqueta de imagen * ⓘ

latest ▼

Tipo de SO

Linux

Tamaño * ⓘ

1 vcpu, 1.5 GiB de memoria, 0 gpu

[Cambiar el tamaño](#)

Revisar y crear

< Anterior

Siguiente: Redes >

Figura 14: Captura que refleja la configuración en el panel Datos básicos para la creación de la instancia sgdb.

10. Pasamos a la pestaña **Redes** y añadimos el puerto 3306 con protocolo *tcp*:

Crear instancia de contenedor

Datos básicos Redes Opciones avanzadas Etiquetas Revisar y crear

Choose between three networking options for your container instance:

- **'Public'** will create a public IP address for your container instance.
- **'Private'** will allow you to choose a new or existing virtual network for your container instance. This is not yet available for Windows containers.
- **'None'** will not create either a public IP or virtual network. You will still be able to access your container logs using the command line.

Networking type ☒ Public ☐ Private ☐ None

Etiqueta de nombre DNS ⓘ

.westeurope.azurecontainer.io

Puertos ⓘ





Puertos	Protocolo de puertos
80	TCP 
3306 	TCP  
<input type="text"/>	<input type="text"/>

Figura 15: Captura que refleja la configuración en el panel Redes donde se ha añadido el puerto 3306.

11. Finalmente accedemos a la pestaña **Revisar y crear** donde se validará la configuración de la instancia y se resumirá la configuración de la misma:

Crear instancia de contenedor

✓ Validación superada

Datos básicos

Suscripción	Azure para estudiantes
Grupo de recursos	BigData1
Región	Oeste de Europa
Nombre de contenedor	sgdb
Tipo de imagen	Private
Servidor de inicio de sesión del registro de imágenes	nicocu97bigdata1.azurecr.io
Imagen	nicocu97bigdata1.azurecr.io/sgdb:latest
Nombre de usuario del registro de imágenes	nicocu97bigdata1
Tipo de SO	Linux
Memoria (GiB)	1.5
Número de núcleos de CPU	1
Tipo de GPU	None
Recuento de GPU	0

Redes

Networking type	Public
Puertos	80 (TCP), 3306 (TCP)

Opciones avanzadas

Directiva de reinicio	En caso de error
Invalidación de comando	[]

Etiquetas

(ninguno)

Crear


< Anterior


Siguiente >

[Descargar una plantilla para la automatización](#)

Figura 16: Captura que refleja la información mostrada en el panel Revisar y crear.

12. Seleccionamos **Crear** y tras un breve periodo, el proceso concluirá con la creación del contenedor.

 **Se completó la implementación**



Nombre de implementación: Microsoft.ContainerInstances-202004...

Suscripción: Azure para estudiantes

Grupo de recursos: BigData1

Hora de inicio: 8/4/2020 3:28:22

Id. de correlación: 873189b7-e4a9-40ef-8430-2048ab888fff

▼ Detalles de implementación [\(Descargar\)](#)

^ Pasos siguientes

[Ir al recurso](#)

Figura 17: Captura que refleja éxito en la creación de la instancia de sgdb.

2.2.1. Ejecución de operaciones de prueba

Ejecutamos las operaciones de inserción y de consulta de prueba mediante los *script* Python al igual que se realizó con el contenedor instalado en la máquina virtual.

La IP asignada a la instancia de contenedor es 51.124.19.132.

1. Operación de inserción:

Editamos el *script* Python que ejecutaba las inserciones de prueba y especificamos la dirección IP de la instancia de contenedor.

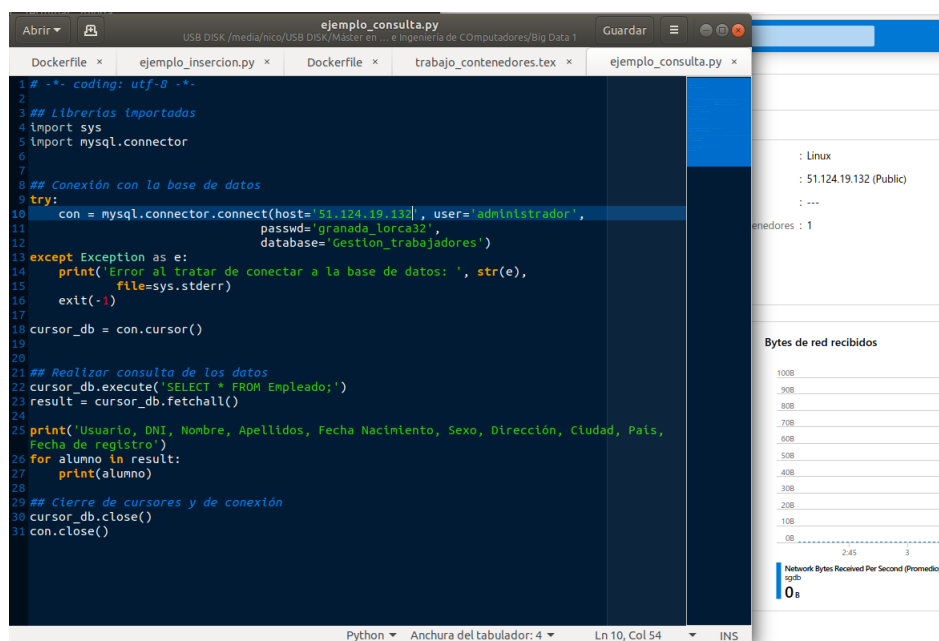


Figura 18: Captura que refleja la edición de la dirección IP configurada en el script cliente de inserción.

La ejecución se realiza exitosamente:

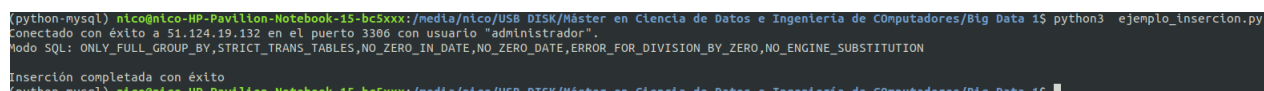


Figura 19: Captura que refleja la ejecución exitosa del script python de inserción sobre la base de datos del contenedor sgdb montado en Azure.

2. Operación de consulta:

Se edita también el *script* de consulta para establecer la dirección IP de la **instancia del contenedor** y se ejecuta, devolviendo los siguientes resultados:


```
(python-mysql) ntco@ntco-HP-Pavilion-Notebook-15-bc5xxx:/media/ntco/USB DISK/Máster en Ciencia de Datos e Ingeniería de Computadores/Big Data $ python3 ejemplo_consulta.py
Conectado con éxito a 51.124.19.132 en el puerto 3306 con usuario "administrador".
Modo SQL: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION

Usuario, DNI, Nombre, Apellidos, Fecha Nacimiento, Sexo, Dirección, Ciudad, País, Fecha de registro
('caspaola', '04940436H', 'Paola', 'Catellí Belgrano', datetime.date(1997, 7, 8), 1, 'Rue Almeghino n° 3', 'Volterra', 'Italia', datetime.datetime(2020, 4, 8, 1, 31, 53))
('cruzylena', '11324956A', 'Elena', 'Cruz', datetime.date(1995, 3, 11), 1, 'Plaza Díaz n° 12', 'La Habana', 'Cuba', datetime.datetime(2020, 4, 8, 1, 31, 53))
('dasanchez', '89138565A', 'David', 'Sanchez Montenegro', datetime.date(1997, 5, 15), 0, 'Calle Ramón y Cajal n° 1', 'Málaga', 'España', datetime.datetime(2020, 4, 8, 1, 31, 53))
('jlopef92', '12568765M', 'Francisco', 'Jiménez Pérez', datetime.date(1992, 1, 8), 0, 'Calle La Alcachofa n° 2', 'Córdoba', 'España', datetime.datetime(2020, 4, 8, 1, 31, 53))
('paloflores', '87210436V', 'Paloma', 'Flores', datetime.date(1996, 11, 27), 1, 'Calle Bolívar n° 23', 'La Colorada', 'México', datetime.datetime(2020, 4, 8, 1, 31, 53))
('paquitagallego', '57824547E', 'Francisca', 'Gallego de Dios', datetime.date(1990, 8, 6), 1, 'Calle Waldo n° 17', 'Córdoba', 'Argentina', datetime.datetime(2020, 4, 8, 1, 31, 53))
('pedriquiró', '35868795S', 'Pedro', 'Ramírez Quiró', datetime.date(1996, 4, 7), 0, 'Calle La Rambla n° 5', 'Barcelona', 'España', datetime.datetime(2020, 4, 8, 1, 31, 53))
```

Figura 20: Captura que refleja la ejecución exitosa del script python de para la consulta de los datos insertados en la tabla *Empelado* en el contenedor *sgdb* montado en Azure.

3. Contenedor *cdpython*

El segundo contenedor que se va a configurar, se trata de un contenedor *Python* 3.8 dotado de las librerías más usuales para la Ciencia de Datos: En concreto, se instalará las librerías *pandas*, *scikit-learn*, *matplotlib*, *scipy* y *numpy*.

Al igual que con el contenedor *sgdb*, primero se tratará la generación de este contenedor en la máquina virtual y, posteriormente en **Azure**:

3.1. Instalación en la máquina virtual

Para generar este contenedor, se parte del contenedor oficial de Python alojado en *docker hub* ².

²Enlace a la página sobre la imagen del contenedor de Python del repositorio de docker hub: https://hub.docker.com/_/python

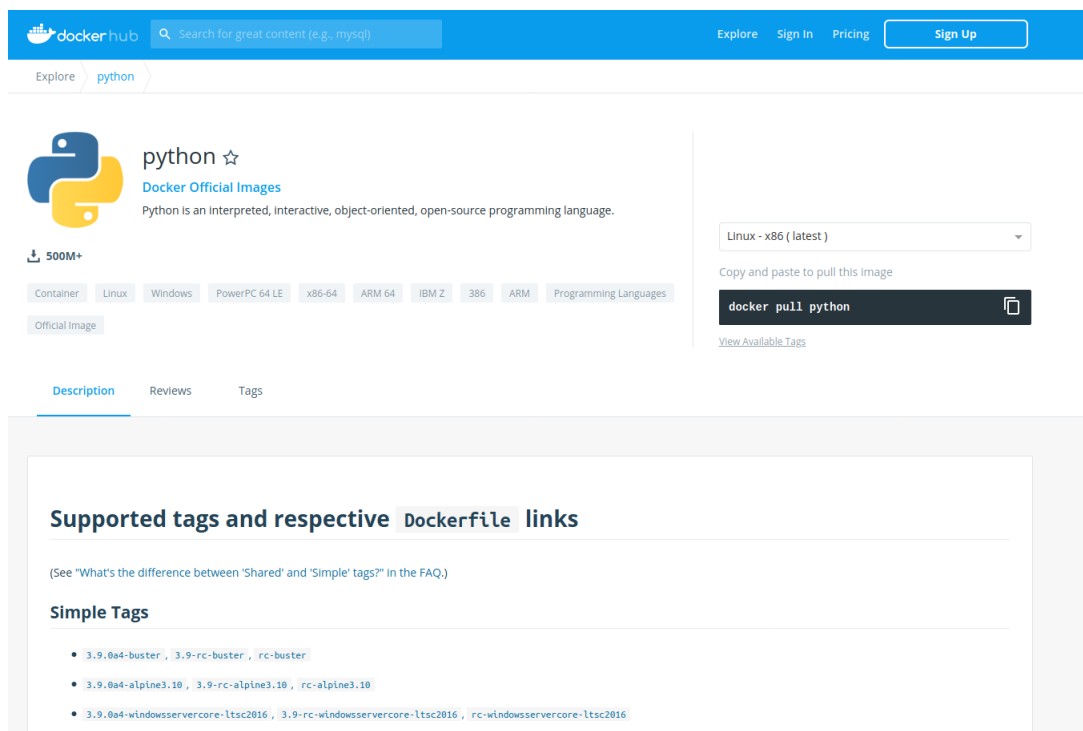


Figura 21: Captura de la página de docker hub sobre la imagen del contenedor de Python.

Se procede a descargar la imagen con el siguiente comando:

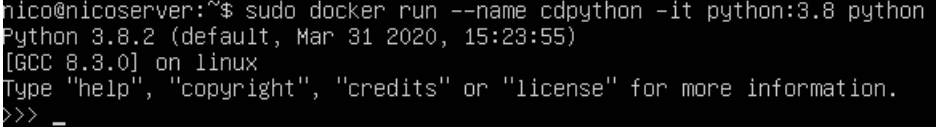
```
$ sudo docker pull python:3.8
```

```
nico@nicoserver:~$ sudo docker pull python:3.8
3.8: Pulling from library/python
50e431f79093: Pull complete
dd8c6d374ea5: Pull complete
c85513200d84: Pull complete
55769680e827: Pull complete
f5e195d50b88: Pull complete
94cdd3612287: Pull complete
8b37b69935d4: Pull complete
b9add85f08c4: Pull complete
aa1f4a29beac: Pull complete
Digest: sha256:45afb066f200549ea61466a3be2dfd4d1d2541ae11ac39c4608d1e4e702cbe5c
Status: Downloaded newer image for python:3.8
docker.io/library/python:3.8
nico@nicoserver:~$
```

Figura 22: Captura de la shell donde se refleja el proceso de descarga de la imagen de Python.

Iniciamos la ejecución de un nuevo contenedor de esta imagen accediendo a la terminal de Python:

```
$ sudo docker run -it python:3.8 --name cdpypython python
```



```
nico@nicoserver:~$ sudo docker run --name cdpypython -it python:3.8 python
Python 3.8.2 (default, Mar 31 2020, 15:23:55)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

Figura 23: Captura de la shell donde se muestra el inicio de una terminal interactiva en un contenedor de Python 3.8.

Accedemos a una consola de *bash* del contenedor con Python y mediante el gestor de paquetes *pip*, instalamos todos los paquetes (numpy, matplotlib, scikit-learn, etc):

```
$ podman run -it python:3.8 --name cdpypython /bin/bash
```

E instalamos todos los paquetes:

```
$ pip3 install numpy
$ pip3 install matplotlib
$ pip3 install pandas
$ pip3 install scipy
$ pip3 install scikit-learn
```

El proceso de instalación se visualiza en la siguiente figura 24:

```

root@eaabeeeeafffa:/# pip3 install pandas
Collecting pandas
  Downloading pandas-1.0.2-cp38-cp38-manylinux1_x86_64.whl (10.0 MB)
    | 10.0 MB 1.1 MB/s
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.8/site-packages (from pandas) (1.18.2)
Collecting pytz>=2017.2
  Downloading pytz-2019.3-py2.py3-none-any.whl (509 kB)
    | 509 kB 1.0 MB/s
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/python3.8/site-packages (from pandas) (2.8.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/site-packages (from python-dateutil>=2.6.1->pandas) (1.14.0)
Installing collected packages: pytz, pandas
Successfully installed pandas-1.0.2 pytz-2019.3
root@eaabeeeeafffa:/# pip3 install scipy
Collecting scipy
  Downloading scipy-1.4.1-cp38-cp38-manylinux1_x86_64.whl (26.0 MB)
    | 26.0 MB 1.1 MB/s
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.8/site-packages (from scipy) (1.18.2)
Installing collected packages: scipy
Successfully installed scipy-1.4.1
root@eaabeeeeafffa:/# pip3 install scikit-learn
Collecting scikit-learn
  Downloading scikit_learn-0.22.2.post1-cp38-cp38-manylinux1_x86_64.whl (7.0 MB)
    | 7.0 MB 1.1 MB/s
Collecting joblib>=0.11
  Downloading joblib-0.14.1-py2.py3-none-any.whl (294 kB)
    | 294 kB 982 kB/s
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.8/site-packages (from scikit-learn) (1.4.1)
Requirement already satisfied: numpy>=1.11.0 in /usr/local/lib/python3.8/site-packages (from scikit-learn) (1.18.2)
Installing collected packages: joblib, scikit-learn
Successfully installed joblib-0.14.1 scikit-learn-0.22.2.post1
root@eaabeeeeafffa:/#

```

Figura 24: Captura de la shell donde se muestra la ejecución de los comandos de instalación de las librerías de Python mediante el gestor pip.

Por último, detenemos la ejecución del contenedor y hacemos un commit de los cambios:

```

$ sudo docker stop cdpypthon
$ sudo docker commit cdpypthon

```

```

nico@nicoserver:~$ sudo docker commit cdpypthon
sha256:6d0578840ad350cced0eafa2db4840001bf6974668661d1b7ba4067309f3583f
nico@nicoserver:~$

```

Figura 25: Captura de la shell donde se refleja la ejecución del commit con la imagen cdpypthon.

El fichero **Dockerfile** que recoge la construcción de este contenedor es el siguiente:

```

1 FROM python:3.8
2 LABEL maintainer Nicolás Cubero Torres (nicocu97@correo.ugr.es)
3
4 RUN pip install numpy # Instalar numpy
5 RUN pip install pandas # Instalar pandas
6 RUN pip install scipy # Instalar scipy
7 RUN pip install scikit-learn # Instalar scikit-learn
8 RUN pip install matplotlib # Instalar matplotlib
9
10 # Copiar el script de ejemplo al contenedor
11 ADD ejemplo_svm_iris.py /

```

Script 4: Fichero Dockerfile para la construcción del contenedor cdpthon

En el anterior Dockerfile, se ha incluido una sentencia para copiar el *script* de ejemplo al contenedor y poder lanzar su ejecución.

3.1.1. Ejemplo de ejecución

A continuación, ponemos a prueba este contenedor con la ejecución del siguiente *script* Python que ejecuta un clasificador *SVM* sobre el conjunto de datos de *iris*:

```

1 # -*- coding: utf-8 -*-
2
3 ## Importar librerías
4 import pandas as pd
5 import numpy as np
6 from sklearn.svm import SVC
7 from sklearn.datasets import load_iris
8 from sklearn.model_selection import train_test_split
9 from sklearn.metrics import (confusion_matrix, f1_score,
10                             precision_score,
11                             recall_score)
12
13 np.random.seed(7)
14
15 ## Cargar datos de iris
16 iris = load_iris()
17 X, y = iris['data'], iris['target']
18 X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2)
19
20 ## Entrenar con un modelo SVM
21 print('Entrenando con modelo SVM')
22 svm = SVC(kernel='rbf', C=2.2, gamma=0.21)
23 svm.fit(X_train, y_train)
24
25 ## Evaluar y predecir el conjunto de test
26 print('Prediciendo un conjunto de test')
27 score = svm.score(X_test, y_test)
28 print('Se ha obtenido un Accuracy de {}'.format(score))
29
30 y_predict = svm.predict(X_test)
31
32 print('Matriz de confusión: Filas->Clase real, Columnas->Clase predicha
33       :')

```

```

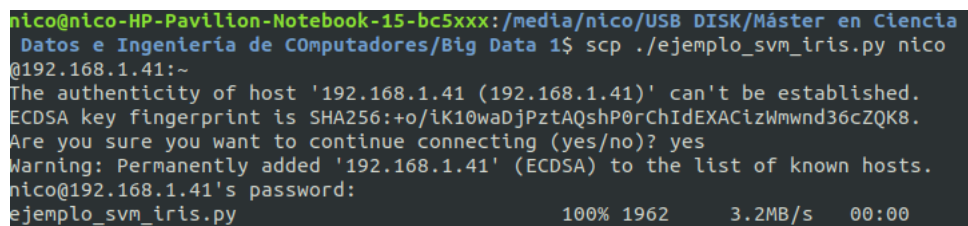
32 print(confusion_matrix(y_test, y_predict))
33 print('Precisión media asociada al clasificador: ', precision_score(
    y_test,
34         y_predict, average='weighted'))
35 print('Recall medio asociada al clasificador: ', recall_score(y_test,
36         y_predict, average='weighted'))
37 print('Puntuación F1 media asociada al clasificador: ', f1_score(y_test
    ,
38         y_predict, average='weighted'))

```

Script 5: Script Python que ejecuta un clasificador SVM sobre el conjunto de datos de iris y evalúa sus métricas de rendimiento

Este *script* parte el conjunto de datos de *iris* en dos subconjuntos de entrenamiento y test, entrena un clasificador *SVM* con el conjunto de entrenamiento y clasifica el conjunto de test evaluando diversas métricas de rendimiento.

Para la ejecución del *script* por el contenedor, se debe de copiar este *script* en el sistema de ficheros del contenedor, por lo que primeramente, se copia este *script* en la máquina virtual haciendo uso del comando *scp*:



```

nico@nico-HP-Pavilion-Notebook-15-bc5xxx:/media/nico/USB DISK/Máster en Ciencia
Datos e Ingeniería de Computadores/Big Data 1$ scp ./ejemplo_svm_iris.py nico
@192.168.1.41:~
The authenticity of host '192.168.1.41 (192.168.1.41)' can't be established.
ECDSA key fingerprint is SHA256:+o/iK10waDjPztAQshP0rChIdEXACizWmwnd36cZQK8.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.41' (ECDSA) to the list of known hosts.
nico@192.168.1.41's password:
ejemplo_svm_iris.py                                100% 1962      3.2MB/s   00:00

```

Figura 26: Captura de la shell donde se refleja el proceso de copia de la máquina hospedadora a la máquina virtual.

Ya en la *shell* del servidor copiamos el *script* al contenedor *cdpython*:

```
$ sudo docker cp ejemplo_svm_iris.py cdpthon:./
```

Y ejecutamos el *script* con el contenedor:

```
$ sudo docker exec -it cdpthon python ejemplo_svm_iris.py
```

Observamos que el *script* se ejecuta satisfactoriamente:

```
nico@nicoserver:~$ sudo docker exec -it cdpypython python ejemplo_svm_iris.py
Entrenando con modelo SVM
Prediciendo un conjunto de test
Se ha obtenido un Accuracy de 0.9666666666666667
Matriz de confusión: Filas->Clase real, Columnas->Clase predicha:
[[ 7  0  0]
 [ 0 11  1]
 [ 0  0 11]]
Precisión media asociada al clasificador: 0.9694444444444444
Recall medio asociada al clasificador: 0.9666666666666667
Puntuación F1 media asociada al clasificador: 0.9666666666666667
nico@nicoserver:~$ _
```

Figura 27: Captura de la shell donde se refleja la ejecución del *script* Python de clustering SVM.

Nota: En el fichero *Dockerfile* se ha añadido una instrucción para copiar este *script* en el sistema de ficheros del contenedor y evitar tener que repetir esta operación en los contenedores generados a partir del *Dockerfile*.

3.2. Instalación en Azure

Repetimos el despliegue del contenedor *cdpypython*, en la plataforma **Azure**, al igual con el contenedor *sgdb*.

No obstante, en la plataforma *Azure* al no disponer de un gestor de paquetes mediante el cual lanzar el intérprete de Python, se decide **instalar un servidor SSH** en el contenedor *cdpypython* desde el cual poder acceder remotamente al contenedor:

Más concretamente, se sealizan los siguientes ajustes en el contenedor *cdpypython*:

- Instalación del servidor ssh **openssh-server**, inicio de su ejecución y escucha en el puerto 22.
- Creación del usuario **administrador** con contraseña **granada_lorca32** mediante el cual se accederá al contenedor.

Para implementar todos estos ajustes, se edita el *Dockerfile* para la generación de este contenedor:

```
1 FROM python:3.8
2 LABEL maintainer Nicolás Cubero Torres (nicocu97@correo.ugr.es)
3
4 # Instalar Open SSH
5 RUN apt-get -y update
6 RUN apt-get install -y openssh-server
7 RUN mkdir /var/run/sshd
8
```

```

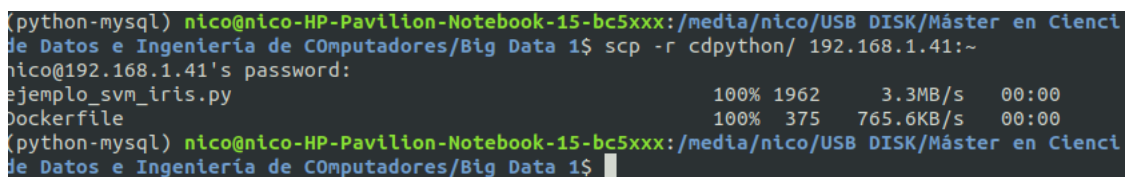
9 # Crear usuario "administrador" y asignar contraseña
10 RUN useradd -s /bin/bash administrador
11 RUN echo "administrador:granada_lorca32" | chpasswd
12
13 RUN pip install numpy # Instalar numpy
14 RUN pip install pandas # Instalar pandas
15 RUN pip install scipy # Instalar scipy
16 RUN pip install scikit-learn # Instalar scikit-learn
17 RUN pip install matplotlib # Instalar matplotlib
18
19 # Copiar el script de ejemplo al contenedor
20 ADD ejemplo_svm_iris.py /
21
22 # Habilitar el demonio ssh y el puerto
23 EXPOSE 22
24 CMD ["/usr/sbin/sshd", "-D"]

```

Script 6: Fichero Dockerfile del contenedor `cdpython` modificado para llevar a cabo la instalación de un servidor ssh con el que poder acceder remotamente al contenedor

Mediante este **Dockerfile** modificado, se generará la imagen desde la máquina virtual y se subirá al **registro de contenedor** creado anteriormente al igual que se hizo con el contenedor `sgdb`, concluyéndose con la creación de una **instancia de contenedor** con la imagen generada:

1. Al igual que con el contenedor `sgdb`, el fichero **Dockerfile** se ha introducido en un directorio que se ha nombrado `cdpython` y que se copia a la máquina virtual mediante `scp`:



```

(pyhton-mysql) nico@nico-HP-Pavilion-Notebook-15-bc5xxx:/media/nico/USB DISK/Máster en Cienci
de Datos e Ingeniería de Computadores/Big Data 1$ scp -r cdpthon/ 192.168.1.41:~
nico@192.168.1.41's password:
ejemplo_svm_iris.py                                100% 1962    3.3MB/s   00:00
Dockerfile                                          100% 375     765.6KB/s 00:00
(pyhton-mysql) nico@nico-HP-Pavilion-Notebook-15-bc5xxx:/media/nico/USB DISK/Máster en Cienci
de Datos e Ingeniería de Computadores/Big Data 1$ █

```

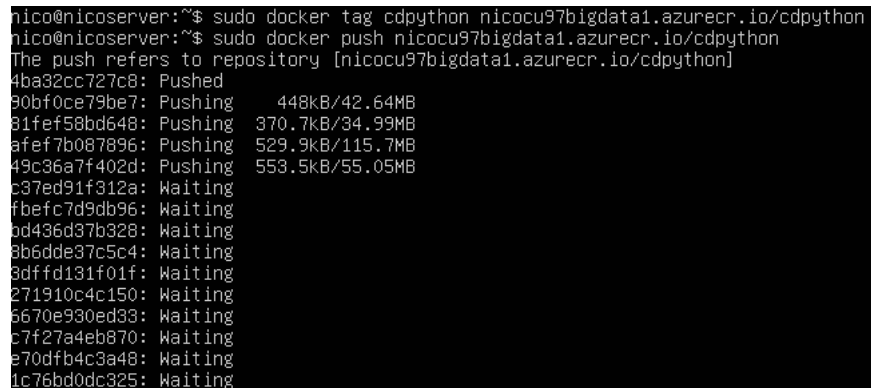
Figura 28: Copia del fichero con el Dockerfile de `cdpython` a la máquina virtual.

2. Una vez en la máquina virtual, no se necesita volver a repetir el **logueo** en **registro de contenedor** y se procede a construir la imagen a partir del **Dockerfile** (para lo cual se necesita haber borrado previamente tanto el contenedor como la imagen existentes creadas en el epígrafe anterior):

```
$ sudo docker build -t cdpthon cdpthon
```


3. Se etiqueta la imagen y se sube al **registro de contenedor** tal y como se realizó para el contenedor **sgdb**:

```
$ sudo docker tag cdpypthon nicocu97bigdata1.azurecr.io/cdpypthon
$ sudo docker push nicocu97bigdata1.azurecr.io/cdpypthon
```



```
nico@nicoserver:~$ sudo docker tag cdpypthon nicocu97bigdata1.azurecr.io/cdpypthon
nico@nicoserver:~$ sudo docker push nicocu97bigdata1.azurecr.io/cdpypthon
The push refers to repository [nicocu97bigdata1.azurecr.io/cdpypthon]
4ba32cc727c8: Pushed
90bf0ce79be7: Pushing    448kB/42.64MB
31fef58bd648: Pushing    370.7kB/34.99MB
afef7b087896: Pushing    529.9kB/115.7MB
49c36a7f402d: Pushing    553.5kB/55.05MB
c37ed91f312a: Waiting
fbefc7d9db96: Waiting
bd436d37b328: Waiting
8b6dde37c5c4: Waiting
3dffd131f01f: Waiting
271910c4c150: Waiting
6670e930ed33: Waiting
c7f27a4eb870: Waiting
e70dfb4c3a48: Waiting
1c76bd0dc325: Waiting
```

Figura 29: Captura de la shell donde se muestra el proceso de etiquetado y subida de la imagen **cdpypthon** al registro de contenedor.

4. Una vez terminada la subida, procedemos a crear una **instancia de contenedor** con la imagen, para ello iniciamos la creación de una nueva instancia de contenedor y en el panel **Datos básicos** asociamos este recurso al grupo de recursos de **BigData1**, nombramos al contenedor como **cdpypthon** y seleccionamos cargar la imagen **cdpypthon** desde nuestro **registro de recursos**:

Inicio > Instancias de contenedor > Crear instancia de contenedor

Crear instancia de contenedor

Datos básicos

Redes

Opciones avanzadas

Etiquetas

Revisar y crear

Azure Container Instances (ACI) le permite ejecutar contenedores en Azure de forma rápida y fácil, sin necesidad de administrar servidores o de tener que aprender a usar nuevas herramientas. ACI ofrece facturación por segundo para minimizar el costo de ejecución de los contenedores en la nube. [Más información acerca de Azure Container Instances](#)

Detalles del proyecto

Seleccione la suscripción para administrar recursos implementados y los costes. Use los grupos de recursos como carpetas para organizar y administrar todos los recursos.

Suscripción * ⓘ

Azure para estudiantes

Grupo de recursos * ⓘ

BigData1

[Crear nuevo](#)

Detalles del contenedor

Nombre de contenedor * ⓘ

cdpython

Región * ⓘ

(Europe) Oeste de Europa

Origen de imagen * ⓘ

☐ Imágenes de inicio rápido

☒ Azure Container Registry

☐ Docker Hub u otro registro

Registro * ⓘ

nicocu97bigdata1

Imagen * ⓘ

cdpython

Etiqueta de imagen * ⓘ

latest

Tipo de SO

Linux

Tamaño * ⓘ

1 vcpu, 1.5 GiB de memoria, 0 gpu

[Cambiar el tamaño](#)

Revisar y crear

< Anterior

Siguiente: Redes >

Figura 30: Captura que refleja la configuración en el panel Datos básicos para la creación de la instancia cdpython.

5. Pasamos a la pestaña **Redes** y añadimos el puerto 22 con protocolo *tcp* para permitir la conexión por **ssh**:

Crear instancia de contenedor

Datos básicos **Redes** Opciones avanzadas Etiquetas Revisar y crear

Choose between three networking options for your container instance:

- **'Public'** will create a public IP address for your container instance.
- **'Private'** will allow you to choose a new or existing virtual network for your container instance. This is not yet available for Windows containers.
- **'None'** will not create either a public IP or virtual network. You will still be able to access your container logs using the command line.

Networking type ☒ Public ☐ Private ☐ None

Etiqueta de nombre DNS ⓘ

.westeurope.azurecontainer.io

Puertos ⓘ





Puertos	Protocolo de puertos
80	TCP 
22 	TCP  
<input type="text"/>	<input type="text"/>

Figura 31: Captura que refleja la configuración en el panel Redes donde se ha añadido el puerto 22 para permitir la conexión por ssh.

6. Finalmente accedemos a la pestaña **Revisar y crear** donde se validará la configuración de la instancia y se resumirá la configuración de la misma:

Crear instancia de contenedor

✓ Validación superada

Datos básicosRedesOpciones avanzadasEtiquetasRevisar y crear

Datos básicos

Suscripción	Azure para estudiantes
Grupo de recursos	BigData1
Región	Oeste de Europa
Nombre de contenedor	cdpython
Tipo de imagen	Private
Servidor de inicio de sesión del registro de nicocu97bigdata1.azurecr.io imágenes	
Imagen	nicocu97bigdata1.azurecr.io/cdpython:latest
Nombre de usuario del registro de imágenes	nicocu97bigdata1
Tipo de SO	Linux
Memoria (GiB)	1.5
Número de núcleos de CPU	1
Tipo de GPU	None
Recuento de GPU	0

Redes

Networking type	Public
Puertos	80 (TCP), 22 (TCP)

Opciones avanzadas

Directiva de reinicio	En caso de error
Invalidación de comando	[]

Etiquetas

(ninguno)

Crear

< Anterior

Siguiente >

[Descargar una plantilla para la automatización](#)

Figura 32: Captura que refleja la información mostrada en el panel Revisar y crear.

7. Seleccionamos **Crear** y esperamos que la creación de la instancia concluya.

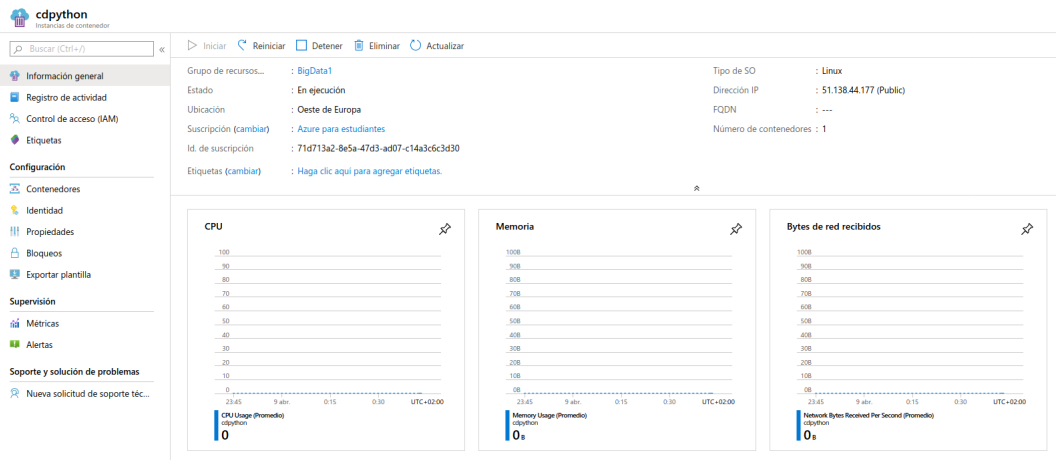


Figura 33: Captura que refleja la instancia creada exitosamente y funcionando.

3.2.1. Ejemplo de ejecución

Probamos la instancia que acabamos de desplegar con la ejecución del *script* Python que se utilizó para probar el contenedor desplegado en la máquina virtual.

Accedemos al contenedor via **ssh** mediante la dirección IP asignada por Azure a la **instancia de contenedor**: 51.138.44.177:

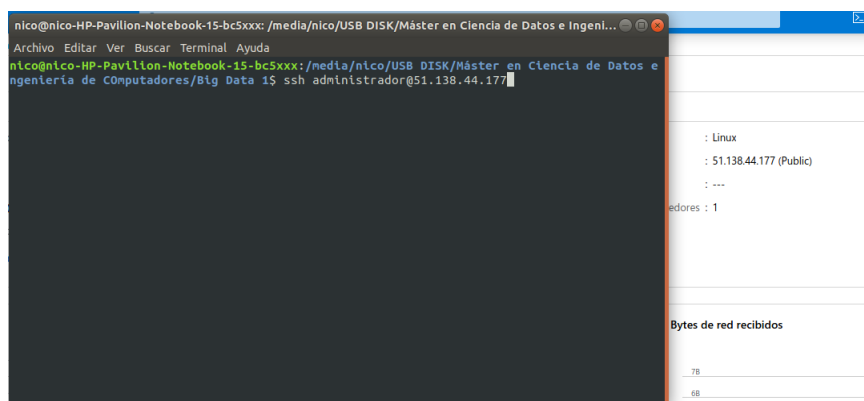


Figura 34: Captura que refleja el acceso a la instancia de contenedor por ssh mediante el usuario administrador.

Una vez conectado al contenedor, se invoca el intérprete de *Python* para ejecutar el *script* de ejemplo:

```

administrador@wk-caas-0d33c82e9682406fa6cbe9382c8c2e8d-994eb54f33e815b010dc40:/$ python3 ejemplo_svm_iris.py
Entrenando con modelo SVM
Prediciendo un conjunto de test
Se ha obtenido un Accuracy de 0.9666666666666667
Matriz de confusión: Filas->Clase real, Columnas->Clase predicha:
[[ 7  0  0]
 [ 0 11  1]
 [ 0  0 11]]
Precisión media asociada al clasificador: 0.9694444444444444
Recall medio asociada al clasificador: 0.9666666666666667
Puntuación F1 media asociada al clasificador: 0.9666666666666667
administrador@wk-caas-0d33c82e9682406fa6cbe9382c8c2e8d-994eb54f33e815b010dc40:/$

```

Figura 35: Captura que refleja le ejecución del script Python que ejecuta un clasificador SVM sobre el dataset iris en la instancia de contenedor.

Se observa que el *script* se ejecuta exitosamente con los resultados esperados.

4. Contenedor cdr

El último contenedor que se va a desplegar, es un contenedor con *R* junto con los siguientes paquetes: *tidyverse* , *caret* , *RSNNS* , *frbs* , *FSinR*, *forecast* y *e1071*.

Primeramente, se llevará a cabo la instalación de este contenedor en la máquina virtual y, posteriormente, se repetirá la instalación de este contenedor en la pataforma **Azure**.

Para la construcción de este contenedor, se partirá de la imagen *r-base* alojada en docker hub ³

³Enlace a la página de docker hub con la imagen r-base: <https://hub.docker.com/r/rocker/r-base>

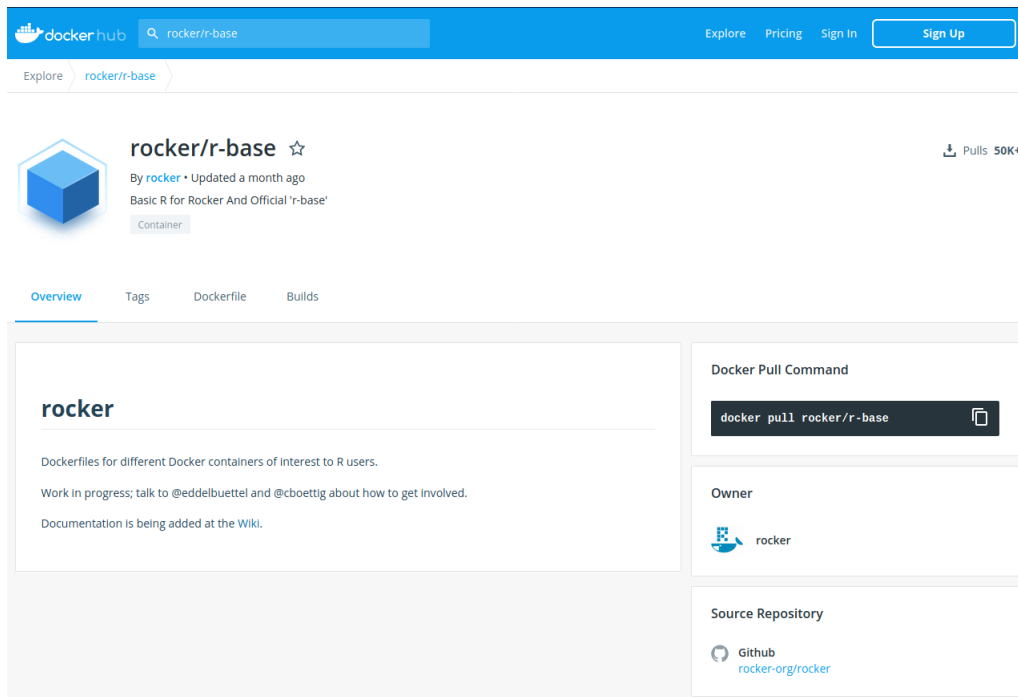


Figura 36: Captura de la página de la imagen r-base en docker hub.

4.1. Instalación en la máquina virtual

Se descarga la imagen en primer lugar:

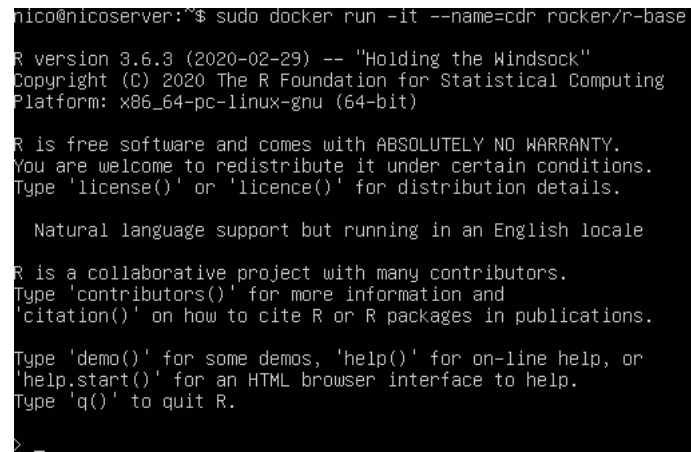
```
$ sudo docker pull rocker/r-base
```

```
nico@nicoserver:~$ sudo docker pull rocker/r-base
Using default tag: latest
latest: Pulling from rocker/r-base
1fcd5305bc72: Already exists
99ff90e54cd7: Pull complete
a3e49982641b: Pull complete
5c32f27a96d8: Pull complete
bec945f98c52: Pull complete
19dab81f8799: Pull complete
Digest: sha256:855c74722904f70e80f22df76defcedcb20899df850a4c8957135e0dc4999861
Status: Downloaded newer image for rocker/r-base:latest
docker.io/rocker/r-base:latest
nico@nicoserver:~$
```

Figura 37: Captura de la shell donde se refleja el proceso de descarga de la imagen de r-base y la salida obtenida.

Una vez instalada la imagen, se procede a iniciar un nuevo contenedor y a acceder a la terminal interactiva de R en el mismo:

```
$ sudo docker run --name cdr -it rocker/r-base R
```



```
nico@nicoserver:~$ sudo docker run -it --name=cdr rocker/r-base R
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> _
```

Figura 38: Captura de la terminal de R del contenedor ejecutado.

Procedemos entonces, a realizar la instalación de todos los paquetes (*tidyverse* , *caret* , *RSNNS* , *frbs*, etc).

No obstante, dado que *tidyverse* y *forecast* requieren la instalación de los paquetes *libcurl4-openssl-dev*, *libxml2-dev*, *xml2* y *libssl-dev* respectivamente, debemos de acceder a la consola de *bash* del contenedor e instalarlos mediante el gestor de paquetes *apt*:

```
$ sudo docker exec -it cdr bash
```

y instalan los paquetes por medio de *apt*:

```
$ apt-get update # Actualizar los paquetes instalados
```

```
$ apt-get install libcurl4-openssl-dev xml2 libxml2-dev libssl-dev
```



```

root@5fd206636756:/# apt-get update
Hit:1 http://deb.debian.org/debian testing InRelease
Hit:2 http://cdn-fastly.deb.debian.org/debian sid InRelease
Reading package lists... Done
root@5fd206636756:/# apt-get install libcurl4-openssl-dev xml2
Reading package lists... Done
Building dependency tree
Reading state information... Done
libcurl4-openssl-dev is already the newest version (7.68.0-1).
The following NEW packages will be installed:
  libxml2 xml2
0 upgraded, 2 newly installed, 0 to remove and 57 not upgraded.
Need to get 723 kB of archives.
After this operation, 2,048 kB of additional disk space will be used.
Get:1 http://deb.debian.org/debian testing/main amd64 libxml2 amd64 2.9.10+dfsg-4 [709 kB]
Get:2 http://deb.debian.org/debian testing/main amd64 xml2 amd64 0.5-3 [14.2 kB]
Fetched 723 kB in 1s (659 kB/s)
debconf: delaying package configuration, since apt-utils is not installed
Selecting previously unselected package libxml2:amd64.
(Reading database ... 17962 files and directories currently installed.)
Preparing to unpack .../libxml2_2.9.10+dfsg-4_amd64.deb ...
Unpacking libxml2:amd64 (2.9.10+dfsg-4) ...
Selecting previously unselected package xml2.
Preparing to unpack .../archives/xml2_0.5-3_amd64.deb ...
Unpacking xml2 (0.5-3) ...
Setting up libxml2:amd64 (2.9.10+dfsg-4) ...
Setting up xml2 (0.5-3) ...
Processing triggers for libc-bin (2.29-10) ...
root@5fd206636756:/#

```

Figura 39: Captura de la terminal de bash del contenedor cdr donde se refleja la instalación de los paquetes libcurl4-openssl-dev xml2 libxml2-dev libssl-dev.

Tras esta operación, se accede a la consola de R y se efectúan las siguientes operaciones:

```

> install.packages('caret')
> install.packages('RSNNS')
> install.packages('frbs')
> install.packages('FSinR')
> install.packages('forecast')
> install.packages('e1071')
> install.packages('tidyverse')

```

Para terminar, salimos del contenedor, detenemos su ejecución y salvamos la imagen:

```

$ sudo docker stop cdr
$ sudo docker commit cdr

```

El archivo Dockerfile que automatiza todo este proceso de instalación es el siguiente:

```

1 FROM rocker/r-base
2 LABEL maintainer Nicolás Cubero Torres (nicocu97@correo.ugr.es)
3
4 # Actualizar e instalar todos los paquetes de sistema
5 RUN apt-get -y update
6 RUN apt-get -y install libcurl4-openssl-dev xml2 libxml2-dev libssl-dev
7
8
9 # Instalar todos los paquetes de R
10 RUN R -e "install.packages(c('caret', 'RSNNS', 'frbs', 'FSinR', '
    forecast', 'e1071', 'tidyverse'))"
11
12 # Añadir el script de ejemplo
13 ADD ejemplo_clustering.R /

```

Script 7: Fichero Dockerfile para la construcción de este contenedor cdr

4.1.1. Ejemplo de ejecución

Al igual que con el anterior contenedor, probamos este contenedor con la ejecución del siguiente *script* R que, ejecuta un *clustering k-means* sobre el conjunto de datos de *iris* excluyendo las etiquetas y muestra: el número de patrones asignado a cada clúster, la distancia SS (*Sum of Squares*) entre cada patrón y el clúster más cercano y la distancia intracúster (distancia SS entre los centroides):

```

1 # Paquetes importados
2 library('dplyr')
3
4 set.seed(7)
5
6 # Usar dataset iris y separar los datos de la etiqueta usando dplyr
7 # (se podría haber usado el operador de slicing, pero se prefiere
    testear tidyverse)
8 attach(iris)
9 iris.data <- iris %>% select(c("Sepal.Length", "Sepal.Width", "Petal.
    Length",
10                               "Petal.Width"))
11 iris.class <- iris %>% select('Species')
12
13 # Normalizar dataset
14 iris.mean <- apply(iris.data, MARGIN = 2, FUN=mean)
15 iris.std <- apply(iris.data, MARGIN = 2, FUN=sd)
16
17 iris.data <- t(apply(iris.data, MARGIN=1,
18                     FUN=function(X,mean,std){(X-mean)/std},
19                     iris.mean, iris.std))
20
21 # Ejecutar clustering
22 clust <- kmeans(iris.data, centers=3)
23
24 # Mostrar los resultados
25 cat('Número de instancias agrupadas en cada cluster:', fill=TRUE)
26 print(table(clust$cluster))
27

```

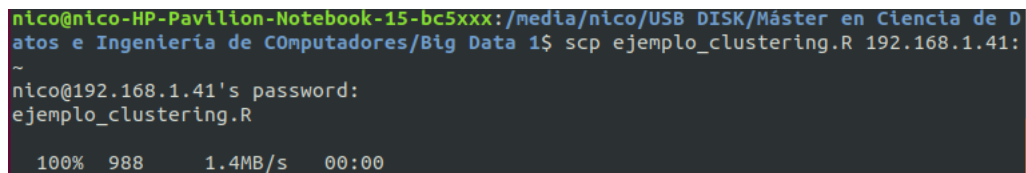
```

28 cat('Distancia SS asociada al clustering: ', clust$totss, fill=T)
29
30 cat('Distancia SS intercluster: ',
31     sum(dist(clust$centers, method = 'manhattan')^2), fill=T)

```

Script 8: Script R que ejecuta un clustering k-means sobre los datos de iris (sin la etiqueta de cada flor) y muestra métricas asociadas al clustering

Copiamos este *script* a la máquina virtual haciendo uso nuevamente de *scp*:



```

nico@nico-HP-Pavilion-Notebook-15-bc5xxx:/media/nico/USB DISK/Máster en Ciencia de D
atos e Ingeniería de Computadores/Big Data 1$ scp ejemplo_clustering.R 192.168.1.41:
~
nico@192.168.1.41's password:
ejemplo_clustering.R
100% 988 1.4MB/s 00:00

```

Figura 40: Captura donde se refleja el proceso de copia del script de ejemplo a la máquina virtual.

Desde la máquina virtual, iniciamos la ejecución del contenedor *cdr*, copiamos el *script* al contenedor y llevamos a cabo la ejecución del *script* con el intérprete de R:

```

$ sudo docker start cdr
$ sudo docker cp ejemplo_clustering.R cdr:./
$ sudo docker exec -it cdr Rscript ejemplo_clustering.R

```



```

nico@nicoserver:~$ sudo docker start cdr
cdr
nico@nicoserver:~$ sudo docker cp ejemplo_clustering.R cdr:./
nico@nicoserver:~$ sudo docker exec -it cdr Rscript ejemplo_clustering.R

Attaching package: 'dplyr'

The following objects are masked from 'package:stats':
  filter, lag

The following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

Número de instancias agrupadas en cada cluster:
 1  2  3
53 47 50
Distancia SS asociada al clustering: 596
Distancia SS intercluster: 102.638

```

Figura 41: Captura donde se refleja el proceso de inicio del contenedor *cdr* y la ejecución del script de prueba.

Observamos que el *script* se ejecuta correctamente en el contenedor.

4.2. Instalación en Azure

Por último, se instalará este contenedor en una **instancia de contenedor de Azure**.

Nuevamente, para permitir la conexión al contenedor una vez desplegado en **Azure**, se configura en el contenedor, un servidor **ssh** al igual que se realizó con el contenedor **cdpython**, por lo que editamos el fichero *Dockerfile* para añadir este servidor **ssh** y al usuario *administrador*:

```
1 FROM rocker/r-base
2 LABEL maintainer Nicolás Cubero Torres (nicocu97@correo.ugr.es)
3
4 # Actualizar e instalar todos los paquetes de sistema
5 RUN apt-get -y update
6 RUN apt-get -y install libcurl4-openssl-dev xml2 libxml2-dev libssl-dev
7
8 # Instalar Open SSH
9 RUN apt-get install -y openssh-server
10 RUN mkdir /var/run/sshd
11
12 # Crear usuario "administrador" y asignar contraseña
13 RUN useradd -s /bin/bash administrador
14 RUN echo "administrador:granada_lorca32" | chpasswd
15
16 # Instalar todos los paquetes de R
17 RUN R -e "install.packages(c('caret', 'RSNNS', 'frbs', 'FSinR', '
    forecast', 'e1071', 'tidyverse'))"
18
19 # Añadir el script de ejemplo
20 ADD ejemplo_clustering.R /
21
22 # Habilitar el demonio ssh y el puerto
23 EXPOSE 22
24 CMD ["/usr/sbin/sshd", "-D"]
```

Script 9: Fichero Dockerfile modificado para la creación de una imagen del contenedor **cdr** con un servidor **ssh**

Al igual que con los anteriores contenedores, este **Dockerfile** es copiado en un directorio al que se denomina también *cdr* y se copian a la máquina virtual, desde la que se construye la imagen que se subirá al **registro de contenedor**:

```
$ sudo docker build -t cdr cdr
```

Seguidamente, se etiqueta la imagen y se inicia su subida al **registro de contenedor**:

```
$ sudo docker tag cdr nicocu97bigdata1.azurecr.io/cdr
$ sudo docker push nicocu97bigdata1.azurecr.io/cdr
```

```
nico@nicoserver:~$ sudo docker tag cdr nicocu97bigdata1.azurecr.io/cdr
[sudo] password for nico:
nico@nicoserver:~$ sudo docker push nicocu97bigdata1.azurecr.io/cdr
The push refers to repository [nicocu97bigdata1.azurecr.io/cdr]
6f315e5d7c06: Pushed
04a200928402: Pushing 6.921MB/409.8MB
dbb9286a6fca: Pushed
695895c13760: Pushed
c6a0987cdb9d: Pushed
23ae78ba84ae: Pushing 664.1kB/30.99MB
ffec1673456e: Pushing 1.135MB/21.76MB
0e5cf58ee85d: Waiting
39e28d826f2c: Pushing 527.5kB/503.2MB
2a22e391034d: Waiting
40c017c013cd: Waiting
f89bc9fec02f: Waiting
4870bcf34e2b: Waiting
7ef5d826a6a7: Waiting
```

Figura 42: Captura donde se refleja el etiquetado y subida de la imagen del contenedor `cdr` al registro de contenedor de Azure.

Procedemos ahora, a crear una **instancia de contenedor** con esta imagen:

Iniciamos la agregación de una nueva instancia y, en el panel **Datos básicos** de nuevo, asignamos esta instancia, grupo de recursos `BigData1`, establecemos `cdr` como nombre de contenedor y seleccionamos la imagen `cdr` desde nuestro **registro de contenedor**:

Crear instancia de contenedor

[Datos básicos](#) [Redes](#) [Opciones avanzadas](#) [Etiquetas](#) [Revisar y crear](#)

Azure Container Instances (ACI) le permite ejecutar contenedores en Azure de forma rápida y fácil, sin necesidad de administrar servidores o de tener que aprender a usar nuevas herramientas. ACI ofrece facturación por segundo para minimizar el costo de ejecución de los contenedores en la nube. [Más información acerca de Azure Container Instances](#)

Detalles del proyecto

Seleccione la suscripción para administrar recursos implementados y los costes. Use los grupos de recursos como carpetas para organizar y administrar todos los recursos.

Suscripción *	<div>①</div> <div>Azure para estudiantes</div> <div>▼</div>
Grupo de recursos *	<div>①</div> <div>BigData1</div> <div>▼</div>

[Crear nuevo](#)

Detalles del contenedor

Nombre de contenedor *	<div>①</div> <div>cdr</div> <div>✓</div>
Región *	<div>①</div> <div>(Europe) Oeste de Europa</div> <div>▼</div>
Origen de imagen *	<div>①</div> <div><div><input type="radio"/> Imágenes de inicio rápido</div><div><input checked="" type="radio"/> Azure Container Registry</div><div><input type="radio"/> Docker Hub u otro registro</div></div>
Registro *	<div>①</div> <div>nicocu97bigdata1</div> <div>▼</div>
Imagen *	<div>①</div> <div>cdr</div> <div>▼</div>
Etiqueta de imagen *	<div>①</div> <div>latest</div> <div>▼</div>
Tipo de SO	Linux
Tamaño *	<div>①</div> <div>1 vcpu, 1.5 GiB de memoria, 0 gpu</div> <div>Cambiar el tamaño</div>

[Revisar y crear](#)

[< Anterior](#)

[Siguiente: Redes >](#)

Figura 43: Captura que refleja la configuración en el panel Datos básicos para la creación de la instancia cdr.

Pasamos a la pestaña **Redes** y al igual que con la instancia de contenedor `cdpython` añadimos el puerto 22 con protocolo `tcp` para permitir la conexión por `ssh`:

Crear instancia de contenedor

[Datos básicos](#) **[Redes](#)** [Opciones avanzadas](#) [Etiquetas](#) [Revisar y crear](#)

Choose between three networking options for your container instance:

- **'Public'** will create a public IP address for your container instance.
- **'Private'** will allow you to choose a new or existing virtual network for your container instance. This is not yet available for Windows containers.
- **'None'** will not create either a public IP or virtual network. You will still be able to access your container logs using the command line.

Networking type ☒ Public ☐ Private ☐ None

Etiqueta de nombre DNS ⓘ

.westeurope.azurecontainer.io

Puertos ⓘ





Puertos	Protocolo de puertos
80	TCP 
22 	TCP  
<input type="text"/>	<input type="text"/>

Figura 44: Captura que refleja la configuración en el panel Redes donde se ha añadido el puerto 22 para permitir la conexión por ssh.

Finalmente en la pestaña **Revisar y crear** se valida la configuración de la instancia y comprobamos que la configuración establecida sea correcta:

Crear instancia de contenedor

✓ Validación superada

Datos básicosRedesOpciones avanzadasEtiquetasRevisar y crear

Datos básicos

Suscripción	Azure para estudiantes
Grupo de recursos	BigData1
Región	Oeste de Europa
Nombre de contenedor	cdr
Tipo de imagen	Private
Servidor de inicio de sesión del registro de imágenes	nicocu97bigdata1.azurecr.io
Imagen	nicocu97bigdata1.azurecr.io/cdr:latest
Nombre de usuario del registro de imágenes	nicocu97bigdata1
Tipo de SO	Linux
Memoria (GiB)	1.5
Número de núcleos de CPU	1
Tipo de GPU	None
Recuento de GPU	0

Redes

Networking type	Public
Puertos	80 (TCP), 22 (TCP)

Opciones avanzadas

Directiva de reinicio	En caso de error
Invalidación de comando	[]

Etiquetas

(ninguna)

Crear

< Anterior

Siguiente >

[Descargar una plantilla para la automatización](#)

Figura 45: Captura que refleja la información mostrada en el panel Revisar y crear.

Seleccionamos **Crear** y esperamos que el proceso termine.

Una vez terminada la creación de esta instancia, accedemos al recurso y podemos comprobar que está activo y la dirección IP que le ha sido asignada (51.137.25.111).

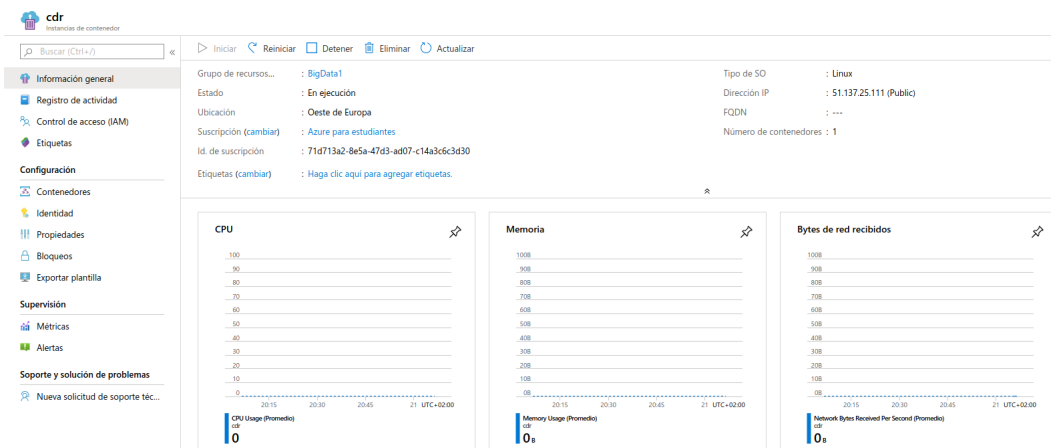


Figura 46: Captura que refleja la instancia creada exitosamente y funcionando.

4.2.1. Ejemplo de ejecución

Probamos esta instancia de contenedor sometiendo a la ejecución del *script* R que fue empleado para probar el contenedor desplegado en la máquina virtual.

Accedemos al contenedor via `ssh`:

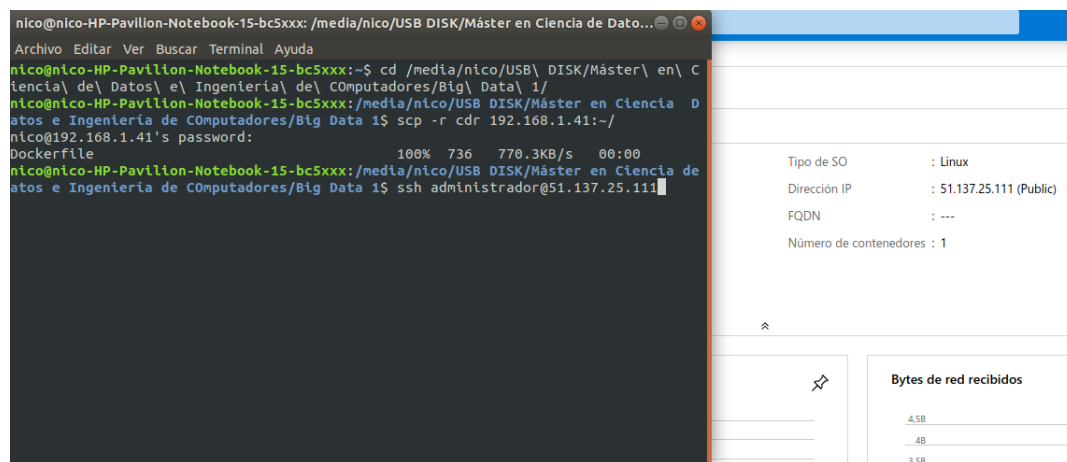


Figura 47: Captura que refleja el acceso a la instancia de contenedor por ssh mediante el usuario administrador.

Una vez conectado al contenedor, se ejecuta el intérprete de R con el *script* de ejemplo y observamos que se ejecuta correctamente:

```
administrador@wk-caas-625c44276b274a4cbb6ce24e20e54353-ce50384a5ee6aeaceb8456:/$ Rscript ejemplo_clustering.R
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':
  filter, lag

The following objects are masked from 'package:base':
  intersect, setdiff, setequal, union

Número de instancias agrupadas en cada cluster:
  1  2  3
53 47 50
Distancia SS asociada al clustering: 596
Distancia SS intercluster: 102.638
administrador@wk-caas-625c44276b274a4cbb6ce24e20e54353-ce50384a5ee6aeaceb8456:/$
```

Figura 48: Captura que refleja la ejecución del script R que ejecuta un clustering KMeans sobre el conjunto de datos de iris (excluyendo la etiqueta) y muestra las medidas de bondad de este clustering.