Podman, Buildah & Skopeo

I.E.S. GONZALO NAZARENO
Proyecto integrado
Víctor Manuel Ruiz Mesa
Fecha:22/04/20

Indice

1. Introducción

1.1. Descripción del proyecto

Vamos a realizar la instalación de una máquina virtual con CentOS y vamos a realizar una configuración completa de contenedores con distintas aplicaciones web en varios lenguajes para familiarizarnos un poco más con estas nuevas tecnologías. Vamos a desplegar:

- Una página web estática muy simple con un servidor web apache.
- Una apliación web php (Drupal) en un contenedor apache con una base de datos en otro contenedor.
- Una aplicación Java (Opencms) en un contenedor con Tomcat y otro contenedor con la base de datos en Postgresql
- Una aplicación Python (Mezzanine) en un contenedor con variables de entorno personalizadas, una base de datos MySQL en otro contenedor con almacenamiento persistente y un proxy inverso (Nginx) en otro contenedor encargado de recibir las peticiones y mapear los puertos.

2. ¿Qué es Podman, Buildah y Skopeo?

Podman, Buildah y skopeo es la alternativa que ofrece Red Hat a Docker containers. El punto fuerte de estas técnologías se basa en la optimización de recursos ya que cada contenedor que levanta Podman corresponde a un único servicio en la máquina anfitrión frente a los 32 que levanta Docker además de que la creación de imágenes se crean con la filosofía Demon-less que no es más que creación de imágenes sin necesidad de demonios.

2.1. *Podman*

Podman es el motor de contenedores que nos permite levantar dichos contenedores de forma similar a Docker pero con algunas diferencias fundamentales:

- Root-less: Nos permite levantar contenedores sin necesidad de ser root
- Daemon-less: Podman no necesita levantar un unico demonio de muchos servicios para funcionar, es más bien algo parecido a la arquitectura de microservicios, levanta los servicios necesarios para cada contenedor de forma que se hace mucho mas "tratable" la gestion de servicios en el anfitrión
- Pods: Podman acuña el termino pod tal y como lo conocemos con Kubernetes de forma que podríamos levantar pods de uno o varios contenedores y aislarlos de otros pods.
- Linea de comandos: Su sintaxis es calcada a la de docker de forma que la curva de aprendizaje es ínfima.

2.2. Buildah

Buildah es el encargado de crear las imagenes OCI. Permite la creación de imágenes from scratch e incluso con Dockerfiles y no necesitamos privilegios de root.

2.3. Skopeo

Skopeo es el encargado de gestionar las imágenes de Buildah y subirlas a diferentes registros incuido el de docker, permite sacar información de las imágens, copiarlas de un registro a otro y de forma local e incluso borrarlas.

3. Como instalar

Vamos a explicar el proceso de instalación para varios sistemas operativos. Podman, Buildah & Skopeo está disponible para casi todos los sistemas Unix, en mi caso usaré un centos RHEL 8 pero vamos a explicar la instalación en varios sistemas Unix para ver su fácil instalación y por si llegásemos a necesitar estas técnologias en otros sistemas.

Podman viene instalado por defecto en la nueva versión de CentOS RHEL 8, si tenemos una versión anterior de este sistema operativo podríamos instalarlo de una forma muy sencilla con

```
sudo yum -y install podman buildah skopeo
```

Para instalarlo en otro sistema operativo como Fedora sería algo parecido a centos pero usando dnf en lugar de yum

```
sudo dnf -y install podman buildah skopeo
```

pero si queremos instalarlo en alguna otra distro como Bebian por ejemplo solo tenemos que retocar nuestros repositorios añadiendo lo siguiente

```
echo 'deb
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/
Debian_10/ /' > /etc/apt/sources.list.d/devel:kubic:libcontainers:stable.list

curl -L
https://download.opensuse.org/repositories/devel:/kubic:/libcontainers:/stable/
Debian_10/Release.key | sudo apt-key add -

sudo apt-get update -qq

sudo apt-get -qq -y install podman buildah skopeo
```

4. Escenarios

4.1. Contenedor con página estática

Vamos a lanzar un contenedor que contenga una página estática en HTML, este es el procedimiento más sencillo a la hora de usar contenedores para ello creamos el siguiente dockerfile

```
FROM debian

MAINTAINER Víctor Manuel Ruiz Mesa "victormruiz92@gmail.com"

RUN apt update && apt install apache2 -y

RUN rm -f /var/www/html/*

ADD ["index.html","/var/www/html"]

RUN chown -R www-data:www-data /var/www/html
WORKDIR /var/www/html

EXPOSE 80

CMD ["/usr/sbin/apache2ctl", "-D", "FOREGROUND"]
```

a continuación vamos a crear la imagen con buildah

```
[victor@localhost ~]$ sudo buildah bud -t static .
```

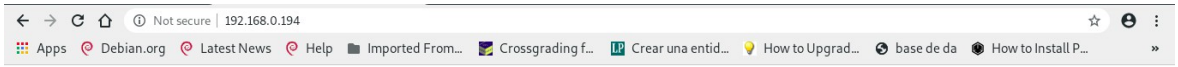
Una vez tengamos la imagen creada vamos a comprobar que se ha creado correctamente, para ello ejecutamos lo siguiente

```
[victor@localhost ~]$ sudo buildah images | grep static
localhost/static                    latest   c8dc73d071c5   About a minute ago
251 MB
```

el siguiente paso es levantar nuestro contenedor, para ello ejecutamos lo siguiente

```
[victor@localhost ~]$  sudo podman run -d --name static --ip 10.88.0.20 -p
80:80 static
a082957d948697c84e74e496df99596be22afbb3e8a112c413108aad93fb77cb
```

solo resta acceder a la IP de nuestra máquina virtual para comprobar que se ha lanzado el contenedor



probando buildah

4.2. Aplicacion PHP con base de datos descentralizada

Ya sabemos como levantar un contenedor con Podman usando Buildah, ahora vamos a lanzar una aplicación php con una base de datos en otro contenedor, en mi caso instalaré Drupal.

Lo primero que vamos a hacer es ver la información de Docker hub de la imagen oficial de Drupal, para ello ejecutamos lo siguiente

```
sudo skopeo inspect docker://docker.io/library/drupal
```

nos aparece mucha información ya que solo en las versiones de php compatibles con la imagen nos ocupa toda la salida del prompt, pero voy a poner un framento del inspect

```
    "Created": "2020-06-10T05:00:06.745516967Z",
    "DockerVersion": "18.09.7",
    "Labels": null,
    "Architecture": "amd64",
    "Os": "linux",
    "Env": [
        "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "PHPIZE_DEPS=autoconf \t\tdpkg-dev \t\tfile \t\tg++ \t\tgcc \t\tlibc-
dev \t\tmake \t\tpkg-config \t\tre2c",
        "PHP_INI_DIR=/usr/local/etc/php",
        "APACHE_CONFDIR=/etc/apache2",
        "APACHE_ENVVARS=/etc/apache2/envvars",
        "PHP_EXTRA_BUILD_DEPS=apache2-dev",
        "PHP_EXTRA_CONFIGURE_ARGS=--with-apxs2 --disable-cgi",
        "PHP_CFLAGS=-fstack-protector-strong -fpic -fpie -O2
```

```
-D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64",
        "PHP_CPPFLAGS=-fstack-protector-strong -fpic -fpie -O2
-D_LARGEFILE_SOURCE -D_FILE_OFFSET_BITS=64",
        "PHP_LDFLAGS=-Wl,-O1 -pie",
        "GPG_KEYS=42670A7FE4D0441C8E4632349E4FDC074A4EF02D
5A52880781F755608BF815FC910DEB46F53EA312",
        "PHP_VERSION=7.4.6",
        "PHP_URL=https://www.php.net/distributions/php-7.4.6.tar.xz",
        "PHP_ASC_URL=https://www.php.net/distributions/php-7.4.6.tar.xz.asc",

"PHP_SHA256=d740322f84f63019622b9f369d64ea5ab676547d2bdcf12be77a5a4cffd06832",
        "PHP_MD5=",
        "DRUPAL_VERSION=8.9.0",
        "DRUPAL_MD5=3f57e9e8a7c2fe9c499712d5d254f55b"
    ]
}
```

como podemos ver tenemos toda la información oficial de la imagen de Drupal de Docker hub pero vamos a hacer una instalación manual para comprender como funciona Buildah, para ello lo primero que vamos a hacer es crear el Dockerfile de nuestro servidor al que le vamos a instalar el cms

```
FROM debian


MAINTAINER Víctor Manuel Ruiz Mesa "victormruiz92@gmail.com"


RUN apt-get update && apt-get -y upgrade && DEBIAN_FRONTEND=noninteractive apt-
get -y install \
    apache2 php php-mysql unzip php-zip php-gd php-curl php-mysqli php-dom php-
simplexml php-xml || :


RUN a2enmod rewrite


RUN rm /var/www/html/index.html


ADD ["drupal-9.0.0.zip","/tmp"]


RUN unzip /tmp/drupal-9.0.0.zip -d /tmp \
    && cp -r /tmp/drupal-9.0.0/* /var/www/html/ \
    && rm -r /tmp/*


WORKDIR /tmp/drupal-9.0.0/
```

```
RUN cp /var/www/html/sites/default/default.settings.php
/var/www/html/sites/default/settings.php \
    && mkdir /var/www/html/sites/default/files \
    && chown -R www-data:www-data /var/www/html

EXPOSE 80

CMD /usr/sbin/apache2ctl -D FOREGROUND
```

ahora montamos la imagen con

```
Sudo buildah bud -t drupal .
```

una vez terminada la imagen vamos a levantar un contenedor con la base de datos de la siguiente forma

```
sudo podman run --name mysqldrupal --ip 10.88.0.30 -d -e
MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=drupal -e MYSQL_USER=usuario -e
MYSQL_PASSWORD=usuario mariadb
```

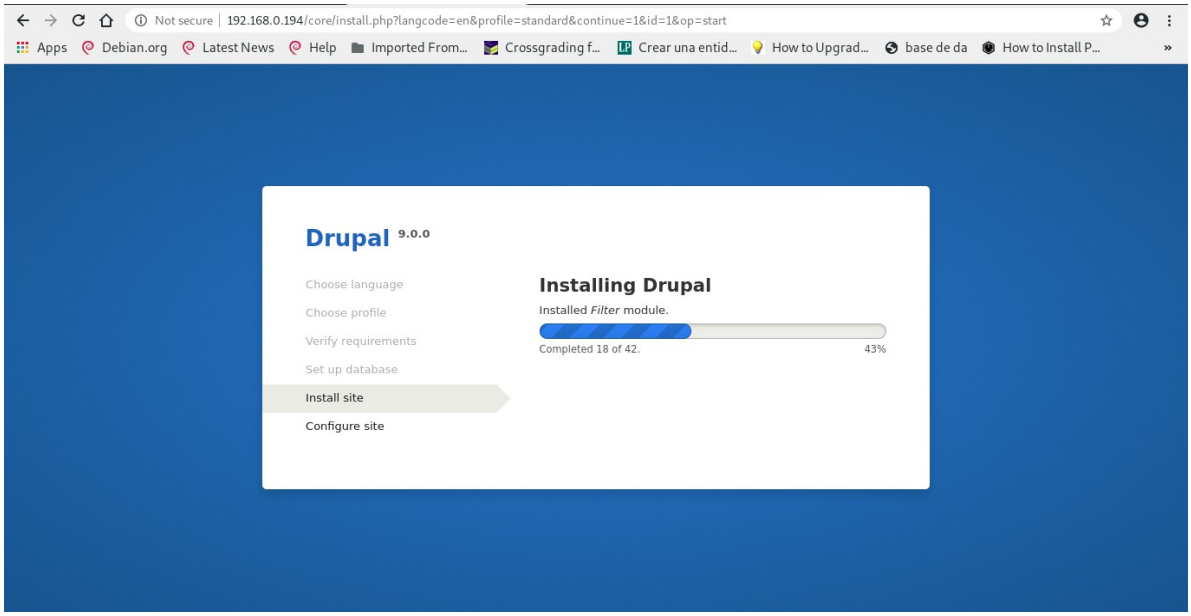
como vemos hacemos uso de la imagen de Docker hub oficial de mariadb con sus variables de entorno correspondientes, ahora levantamos nuestro contenedor con la imagen creada anteriormente

```
sudo podman run --name drupal --ip 10.88.0.31 -p 80:80 -d drupal
```

ya tendríamos nuestros contenedores levantados y en la misma red, para comprobarlo ejecutamos

```
[victor@localhost drupal]$ sudo podman ps -a
CONTAINER ID  IMAGE                                COMMAND               CREATED
STATUS                  PORTS               NAMES
71a01689843f  localhost/drupal:latest           /bin/sh -c /usr/s...  5 minutes
ago  Up 5 minutes ago        0.0.0.0:80->80/tcp  drupal
943df4055f8c  docker.io/library/mariadb:latest  mysqld                2 hours
ago    Up 2 hours ago                             mysqldrupal
```

solo resta comprobar que la instalación de drupal una vez rellenado todos sus parametros de configuración se instala correctamente.



Como vemos que nuestra imagen funciona correctamente vamos a subirla a nuestro registro docker hub para ello usaremos buildah de la siguiente forma

```
[victor@localhost drupal]$ sudo buildah login docker.io
Username: victormrm
Password:
Login Succeeded!
[victor@localhost drupal]$ sudo buildah push localhost/drupal
docker://docker.io/victormrm/primerproyecto:latest
Getting image source signatures
Copying blob 622b3623af2b done
Copying blob 8803ef42039d skipped: already exists
Copying config d778c256ea done
Writing manifest to image destination
Storing signatures
```

4.3. Instalación de aplicación Java con postgreSQL

En el siguiente ejemplo vamos a instalar una aplicación Java con una base de datos postgres, para cambiar tanto de lenguaje como de base de datos.

Lo primero que vamos a hacer es comprobar que en docker hub existe una imagen de tomcat ya que es el servidor de aplicaciones java que vamos a usar

```
sudo skopeo inspect docker://docker.io/library/tomcat
"Created": "2020-06-10T08:18:02.444145908Z",
    "DockerVersion": "18.09.7",
    "Labels": null,
    "Architecture": "amd64",
    "Os": "linux",
    "Env": [
      "PATH=/usr/local/tomcat/bin:/usr/local/openjdk-
11/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
        "LANG=C.UTF-8",
        "JAVA_HOME=/usr/local/openjdk-11",
        "JAVA_VERSION=11.0.7",
        "JAVA_BASE_URL=https://github.com/AdoptOpenJDK/openjdk11-upstream-
binaries/releases/download/jdk-11.0.7%2B10/OpenJDK11U-jdk_",
        "JAVA_URL_VERSION=11.0.7_10",
        "CATALINA_HOME=/usr/local/tomcat",
        "TOMCAT_NATIVE_LIBDIR=/usr/local/tomcat/native-jni-lib",
        "LD_LIBRARY_PATH=/usr/local/tomcat/native-jni-lib",
        "GPG_KEYS=05AB33110949707C93A279E3D3EFE6B686867BA6
07E48665A34DCAFAE522E5E6266191C37C037D42
47309207D818FFD8DCD3F83F1931D684307A10A5 541FBE7D$
        "TOMCAT_MAJOR=9",
        "TOMCAT_VERSION=9.0.36",

"TOMCAT_SHA512=75e16a00e02782961a7753dc9afaf6d209afa5f22d320319778fd0ee5e3b4700
9da522ac735599f1739bff6e809c2da9081dbbd4b8de54a82cf5b8cfbd8030$
    ]
}
```

como podemos ver la imagen existe e incluso podemos ver algunas variables de entorno que vienen configuradas en la imagen.

Ahora vamos a traernosla de forma local con

```
[victor@localhost opencms]$ buildah pull tomcat
Getting image source signatures
Copying blob e9afc4f90ab0 done
Copying blob fb1a405f128d done
Copying blob 5573c4b30949 done
Copying blob af14b6c2f878 done
Copying blob 989e6b19a265 done
Copying blob 612a9f566fdc done
Copying blob cf63ebed1142 done
Copying blob fbb20561cd50 done
Copying blob e99c920870d7 done
Copying blob b7f793f2be47 done
Copying config 2eb5a12030 done
Writing manifest to image destination
Storing signatures
```

```
2eb5a120304e4e7ab6c901e2ca3ed7ea50e57cd4756818f847b1eaa4d34c3881
```

una vez tengamos la imagen vamos a descargarnos el cms que queremos, en mi caso será Opencms

```
wget http://www.opencms.org/en/modules/downloads/begindownload.html?
id=b192f2af-ce4a-11e9-8925-7fde8b0295e1
```

a continuación vamos a descomprimir el fichero

```
unzip opencms-11.0.1.zip
```

y creamos el dockerfile

```
FROM tomcat

MAINTAINER Víctor Manuel Ruiz Mesa "victormruiz92@gmail.com"

RUN apt-get update

ADD ["opencms.war","/tmp"]

RUN cp -r /tmp/* /usr/local/tomcat/webapps/ \
    && rm -r /tmp/*

EXPOSE 8080

CMD ["catalina.sh", "run"]
```

Una vez tengamos el dockerfile terminado, creamos la imagen

```
sudo buildah bud -t opencms .
```

ahora vamos a crear el contenedor con la base de datos postgreSQL. Para ello primero debemos descargar la imagen de forma local

```
podman pull postgresql
```

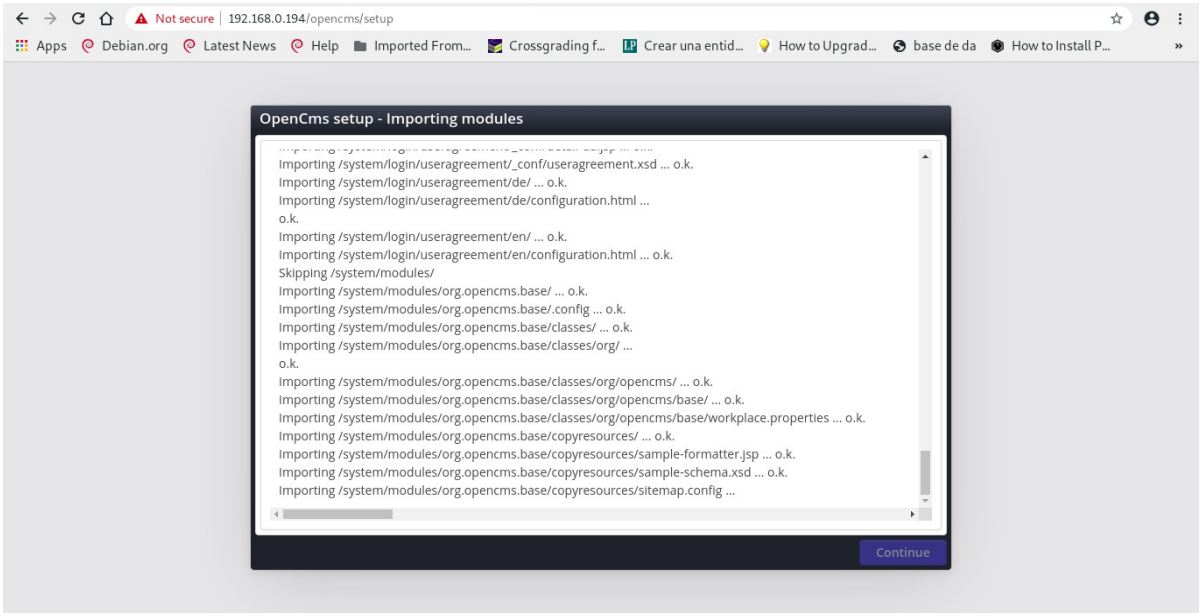
y lanzamos el contenedor con nuestra base de datos

```
sudo podman run --name pgopencms --ip 10.88.0.40 -d -e POSTGRES_PASSWORD=root
postgres
2a1a203c86181ed0f7ca20193d6a526c3363040dfb06a92d130aa1e41372b2e3
```

ahora lanzamos el contenedor con nuestra imagen

```
sudo podman run --name opencms --ip 10.88.0.41 -p 80:8080 -d opencms
8e28a33348b3d5b17398858e4539f82b2d7e4dfcddfb4f47507842360129b60e
```

solo resta comprobar que podemos instalar nuestro opencms después de pasarle los parámetros de configuración



4.4. Instalación de aplicación python personalizada con almacenamiento persistente

Una vez que ya nos hemos familiarizado con estas 3 tecnologías vamos a subir un poco el nivel, hemos visto que las aplicaciones oficiales usan variables de entorno para que los usuarios puedan personalizar su contenedores con estas variables, ahora vamos a realizar el despliegue de una aplicación Python con almacenimiento persistente, personalizando dicha aplicación con variables de entorno.

Necesitamos tener claro algunas cosas

- El Dockerfile debe contener las instrucciones para crear el contenedor pero no las instrucciones que debemos ejecutar cuando ejecutemos el contenedor, para ello vamos a usar un script como entrypoint
- Las aplicaciones Python necesitan entornos virtuales para funcionar correctamente de forma que tendremos que asegurarnos en todo momento que las instrucciones se ejecutan dentro de dicho entorno
- Vamos a poner como proxy inverso un Nginx que debemos configurar para que todo se sirva correctamente
- También vamos a necesitar un contenedor mysql que es al que le pondremos el almacenamiento persistente por lo que debemos asegurarnos de ejecutar este contenedor fuera de la seguridad de SELinux ya que esto impide la creación de contenedores con volúmenes

Teniendo claro todo lo mencionado anteriormente, empecemos.

Lo primero que debemos hacer es crear nuestro Dockerfile, en mi caso ya tengo el proyecto de mezzanine en el directorio "/app/mezzanine" el cual también contiene la base de datos que vamos a importar a nuestro mysql.

```
FROM debian

MAINTAINER Víctor Manuel Ruiz Mesa "victormruiz92@gmail.com"

RUN apt update && apt install python3 python-virtualenv nano -y

ADD ["app/mezzanine","/tmp"]
ADD ["mezzanine.sh","/tmp"]

WORKDIR /tmp

RUN python3 -m virtualenv /opt/venv \
    && /bin/bash -c "source /opt/venv/bin/activate \
    && pip install -r /tmp/requirements.txt \
    && pip install  mysql-connector-python \
    && pip install gunicorn"

run ls /opt/venv/bin

EXPOSE 8000

ENTRYPOINT ["bash", "mezzanine.sh"]
```

Como podemos ver este Dockerfile también le pasa a nuestro contenedor un script el cual crearemos a continuación

```
#!/bin/bash

MYHOST=$(cat /etc/hosts | grep 10 | cut -f 1)

sed -i 's/DATABASEMYSQL/"'${MYSQL_DATABASE}'"/'
/tmp/implantacion/local_settings.py
sed -i 's/USERMYSQL/"'${MYSQL_USER}'"/' /tmp/implantacion/local_settings.py
sed -i 's/PASSWORDMYSQL/"'${MYSQL_PASSWORD}'"/'
/tmp/implantacion/local_settings.py
sed -i 's/HOSTMYSQL/"'${HOST_DATABASE}'"/' /tmp/implantacion/local_settings.py
sed -i 's/CLIENTEAPP/"'${CLIENT_APP}'"/' /tmp/implantacion/local_settings.py
sed -i 's/ESTEHOST/"'${MYHOST}'"/' /tmp/implantacion/local_settings.py


source /opt/venv/bin/activate
python /tmp/manage.py migrate
python /tmp/manage.py loaddata /tmp/bd.json
python /tmp/manage.py collectstatic --noinput
python /tmp/manage.py runserver 0.0.0.0:8000
```

Este script es muy simple. Contempla las variables de entorno que le vamos a pasar al crear el contenedor y el se encarga de ponerlas en la configuración de la aplicación, a su vez también realiza la creación y población de la base de datos y como último paso genera los ficheros estaticos (css, js...) para al final levantar el servicio en el puerto 8000 para todas las IP.

El siguiente paso es configurar la aplicación para que admita las variables de entorno que hemos definido en el script, para ello editamos el fichero "mezzanine/app/mezzanine/implantacion/local_settings.py" de la siguiente forma

```
SECRET_KEY = "lzok4y9j5$6ggxs-66556gjtfg$8nbuz!%#-&4_72sqb4-by0z"
NEVERCACHE_KEY = "6z*3p81bfgbntlc7ny+p8@_6&c3zlgpx79juykq4lklbk%vjk_"

DATABASES = {
    "default": {
        # Ends with "postgresql_psycopg2", "mysql", "sqlite3" or "oracle".
        "ENGINE": "mysql.connector.django",
        # DB name or path to database file if using sqlite3.
        "NAME": DATABASEMYSQL,
        # Not used with sqlite3.
        "USER": USERMYSQL,
        # Not used with sqlite3.
        "PASSWORD": PASSWORDMYSQL,
        # Set to empty string for localhost. Not used with sqlite3.
        "HOST": HOSTMYSQL,
        # Set to empty string for default. Not used with sqlite3.
        "PORT": "3306",
    }
}

# Allowed development hosts
ALLOWED_HOSTS = ["localhost", "127.0.0.1", "::1", HOSTMYSQL , CLIENTEAPP,
ESTEHOST]
```

Ya lo tenemos todo listo para crear nuestra imagen, para ello ejecutamos lo siguiente

```
sudo buildah bud -t python:latest .
```

Una vez tengamos nuestra imagen creada ya podríamos levantar nuestro contenedor pero se crearía en estado de error ya que no tenemos ningún contenedor para almacenar nuestra base de datos, ese es el siguiente paso. Para ello vamos a usar la imagen de mysql que tenemos en Docker hub y recordemos que debemos ejecutarla de forma que no le impida SELinux el almacenamiento de datos en el anfitrión, para ello creamos el directorio de la base de datos

```
mkdir /home/victor/mezzanine/disk
mkdir /home/victor/mezzanine/disk/database
```

Y a continuación creamos el contenedor

```
podman run --security-opt label=disable --name mysqlpython --ip 10.88.0.200 -d
-v /home/victor/mezzanine/disk/database:/var/lib/mysql/:rw -e
MYSQL_DATABASE=mezzanine -e MYSQL_USER=usuario -e MYSQL_PASSWORD=usuario -e
MYSQL_ROOT_PASSWORD=root mysql
```

Como podemos ver lanzamos el contenedor con la opción "--security-opt label=disable" que nos permite almacenar datos en el anfitrión, también tenemos nuestras variables definidas y el directorio donde vamos a almacenar la base de datos de nuestro contenedor

de esta forma ya podríamos crear nuestro contenedor con la aplicación python, para ello ejecutamos lo siguiente

```
sudo podman run --name python --ip 10.88.0.201 -d -e MYSQL_DATABASE=mezzanine
-e MYSQL_USER=usuario -e MYSQL_PASSWORD=usuario -e HOST_DATABASE=10.88.0.200 -e
CLIENT_APP=192.168.0.145 python:latest
```

Como podemos ver especificamos todas las variables de entorno. Solo resta ejecutar nuestro proxy inverso que es el que nos servirá la aplicación, para ello lo primero que vamos a hacer es su Dockerfile

```
FROM nginx

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

RUN rm /etc/nginx/conf.d/default.conf

COPY nginx.conf /etc/nginx/conf.d
```

Vamos a usar su imagen de Docker hub y vemos que le vamos a pasar un fichero de configuración de forma local, este fichero es el que tendrá la configuración de proxy inverso, así que lo creamos y lo editamos de la siguiente forma

```
server {
  listen 80;

  location /static {
    proxy_pass http://10.88.0.201:8000/static;
  }

  location / {
    proxy_pass http://10.88.0.201:8000;
  }
}
```

Una vez lo tengamos todo creamos la imagen

```
Sudo buildah bud -t nginx:v2 .
```

Y una vez creado, lo lanzamos

```
 sudo podman run -d --name nginx --ip 10.88.0.202 -p 80:80 nginxpy:v2
```

A este contenedor le asignamos un mapeo de puerto de forma que al acceder al anitrión por el puerto 80 este lo redirija al puerto 8000 del contenedor.

Ahora vamos a comprobar que todo está correcto antes de la prueba de funcionamiento para enterder bién que ha pasado.
Lo primero es comprobar los logs del contenedor Python, para ello ejecutamos

```
 Sudo podman logs python
```

Este log nos muestra mucha información, más que nada es lo que ha ejecutado nuestro script a la hora de crearse el contenedor (crear la base de datos, cargar los datos, recolectar los ficheros estaticos y levantar el servicio en el puerto 8000).

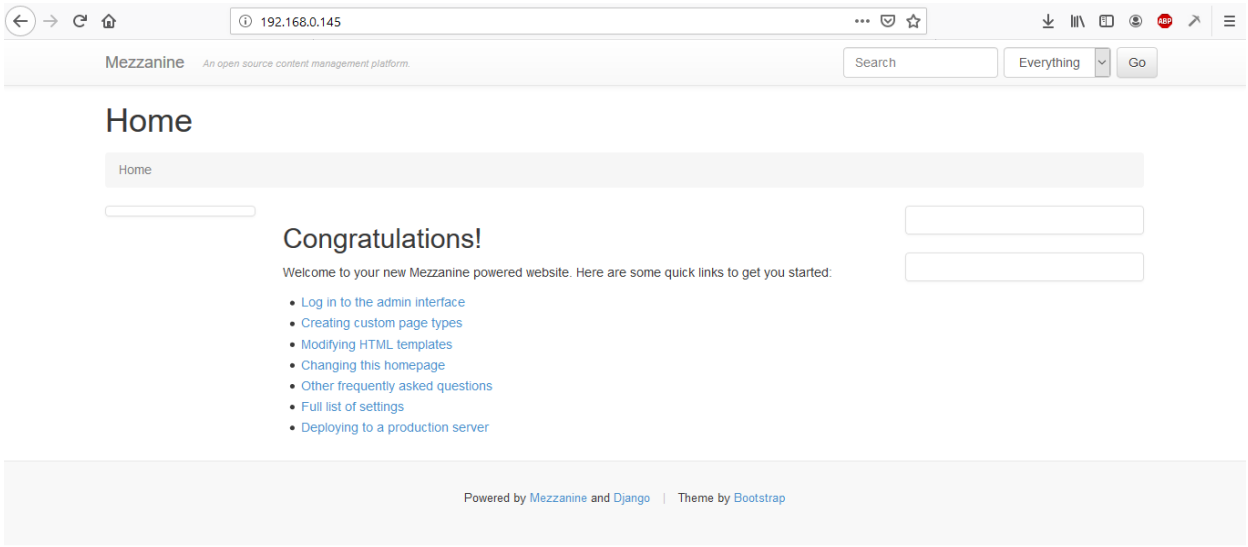
una vez comprobemos que todo ha salido sin errores vamos a subir nuestra imagen a docker hub, para ello lo primero que debemos es logearnos en dicho registro con Buildah

```
[victor@localhost mezzanine]$ sudo buildah login docker.io
Username: victormrm
Password:
Login Succeeded!
```

Y subimos nuestra imagen a nuestro proyecto

```
[victor@localhost mezzanine]$ sudo buildah push python:latest
docker://docker.io/victormrm/python:latest
Getting image source signatures
Copying blob 572f3781e01f done
Copying blob 8803ef42039d skipped: already exists
Copying config 3f5d7ec1da done
Writing manifest to image destination
Storing signatures
```

Ya lo tenemos todo creado, comprobado y subido, solo falta la prueba de funcionamiento



De esta forma ya podríamos borrar los contenedores que, si lo volvemos a lanzar, tendríamos la información de respaldo y no se perderían datos

5. Cruiosidades

Hemos visto que tanto Podman como Buildah se parecen mucho a Docker tanto en comandos como en forma de trabajar (a simple vista) pero no hemos podido ver exactamente como trabaja skopeo porque no nos ha hecho falta para poco mas que comprobar si una imagen existe en un resgistro y poder inspecionar dicha imagen, ahora vamos a copiar una imagen de un registro a otro para ver la utilidad de este pequeño pero potente software, en mi caso voy a copiar una imagen de un registro de Fedora a mi registro de Docker hub

```
[root@localhost mezzanine]# skopeo copy
docker://registry.fedoraproject.org/fedora:latest
docker://docker.io/victormrm/fedora:latest
Getting image source signatures
Copying blob 1657ffead824 done
Copying config eb7134a03c done
Writing manifest to image destination
Storing signatures
```

Durante los ejemplos hemos visto que hemos creado contenedores a través de Dockerfile pero Buildah ofrece la opción de crear contenedores base y poder ejecutar instrucciones directamente en el contenedor para poder luego hacer un commit y generar la imagen, de esta forma podríamos configurar a mano un contenedor y obtener la imagen sin necesidad de dockerfile e incluso nos permite montar su sistema de ficheros de forma local y acceder a su root desde el anfitrión. Vamos a hacer una pequeña prueba instalando en una imagen base Debian un servidor Apache

```
[root@localhost victor]# buildah from debian
debian-working-container
[root@localhost victor]# buildah containers
CONTAINER ID  BUILDER  IMAGE ID     IMAGE NAME                       CONTAINER
NAME
0ca2c53f670e     *     58075fe9ecce                                  debian-
working-container
b928a5ad7398     *     3de0e2c97e5c                                  debian-
working-container-1
aef40e0e8ca8     *     3de0e2c97e5c                                  debian-
working-container-2
9bba12e1c52d     *     3de0e2c97e5c                                  debian-
working-container-3
f47d4ad89915     *     1b686a95ddbf docker.io/library/debian:latest  debian-
working-container-4
```

Ahora vamos a instalar el servidor apache en "debian-working-container"

```
[root@localhost victor]# buildah run debian-working-container apt install
apache2
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils bzip2 ca-certificates file krb5-
locales libapr1
  libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap libbrotli1 libcurl4
```

```
libexpat1
  libgdbm-compat4 libgdbm6 libgpm2 libgssapi-krb5-2 libicu63 libjansson4
libk5crypto3 libkeyutils1
  libkrb5-3 libkrb5support0 libldap-2.4-2 libldap-common liblua5.2-0 libmagic-
mgc libmagic1
  libncurses6 libnghttp2-14 libperl5.28 libprocps7 libpsl5 librtmp1 libsasl2-2
libsasl2-modules
  libsasl2-modules-db libsqlite3-0 libssh2-1 libssl1.1 libxml2 lsb-base mime-
support netbase
  openssl perl perl-modules-5.28 procps psmisc publicsuffix ssl-cert xz-utils
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom www-browser
bzip2-doc gdbm-l10n gpm
  krb5-doc krb5-user sensible-utils libsasl2-modules-gssapi-mit | libsasl2-
modules-gssapi-heimdal
  libsasl2-modules-ldap libsasl2-modules-otp libsasl2-modules-sql perl-doc
  libterm-readline-gnu-perl | libterm-readline-perl-perl make libb-debug-perl
liblocale-codes-perl
  openssl-blacklist
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils bzip2 ca-certificates file
krb5-locales libapr1
  libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap libbrotli1 libcurl4
libexpat1
  libgdbm-compat4 libgdbm6 libgpm2 libgssapi-krb5-2 libicu63 libjansson4
libk5crypto3 libkeyutils1
  libkrb5-3 libkrb5support0 libldap-2.4-2 libldap-common liblua5.2-0 libmagic-
mgc libmagic1
  libncurses6 libnghttp2-14 libperl5.28 libprocps7 libpsl5 librtmp1 libsasl2-2
libsasl2-modules
  libsasl2-modules-db libsqlite3-0 libssh2-1 libssl1.1 libxml2 lsb-base mime-
support netbase
  openssl perl perl-modules-5.28 procps psmisc publicsuffix ssl-cert xz-utils
0 upgraded, 54 newly installed, 0 to remove and 1 not upgraded.
Need to get 25.5 MB of archives.
After this operation, 112 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Configuramos su entrypoint

```
buildah config --entrypoint /usr/sbin/apache2ctl debian-working-container
```

Y por último hacemos el commit de la imagen

```
 buildah commit --format docker debian-working-container ejemploapache:latest
Getting image source signatures
Copying blob 24efcd549ab5 skipped: already exists
Copying blob ffd952c066ea done
Copying config bbee897c14 done
Writing manifest to image destination
Storing signatures
bbee897c143f3feb3bc7b790b054437ade4a9c879a354655d6066a167309d375
```

Vamos a ver si se ha creado la imagen correctamente

```
[root@localhost victor]# buildah images | grep ejemplo
localhost/ejemploapache            latest   bbee897c143f   7 minutes ago   254
```

```
MB
```

Ahora vamos a suponer que necesitamos retocar la configuración del contenedor con apache, para ello vamos a montar de forma local su sistema de ficheros

```
[root@localhost victor]# buildah mount debian-working-container
/
var/lib/containers/storage/overlay/bda5b8ba025650cdbb71023c18e3e13c3a2215143603
740c96b7598a1f5f380c/merged

[root@localhost victor]# ls
/var/lib/containers/storage/overlay/bda5b8ba025650cdbb71023c18e3e13c3a221514360
3740c96b7598a1f5f380c/merged/etc/apache2/
apache2.conf    conf-enabled  magic           mods-enabled  sites-available
conf-available  envvars       mods-available  ports.conf    sites-enabled
```

Bastante útil para la creación de imágenes más sencillitas y de forma mucho más rápida.

6. Conclusión

Podman, Buildah y Skopeo son unas muy buenas alternativas a Docker, su curva de aprendizaje si conoces ya Docker es muy liviana y bajo mi punto de vista tanto Buildah como Podman tiene puntos muy fuertes, no necesitan un demonio que interprete los comandos, cosa que es muy útil si queremos liberar un poco de trabajo a un servidor con mucha carga, ademas permiten que cada usuario de forma independiente puedan ejecutar contenedores, construir imágenes y gestionar sus registros sin necesidad de ser root, esto para mi es algo muy importante. Otro punto fuerte de podman es que cada contenedor se levanta con menos procesos que Docker, esto es debido a que podman levanta un contenedor con los procesos mínimos y en caso de necesitar alguno extra lo levanta cuando sea necesario, de esta forma, un contenedor Apache, por ejemplo, Podman lo levanta con 8 procesos mientras que docker lo levanta con 32. Además Skopeo permite manejar registros también sin demonios con 3 comandos sencillos y sin necesidad de una interfaz web para Docker hub por ejemplo o acceder al servidor donde tengas almacenado el registro, simplemente te logeas en todos los registros que quieras y vas gestionando OCI desde cualquier máquina.

7. Siguientes pasos

- Configurar k8s para que use Podman en lugar de Docker
- Configurar Openshift para use estas 3 nuesvas tecnologías
- desarrollar un sistema de gestión de imagenes entre registros automatizada
- DNS para los contenedores
- Podman para Openstack
- Script de automatización de creación de contenedores

8. Webgrafía

- https://servicesblog.redhat.com/2019/10/09/say-hello-to-buildah-podman-and-skopeo/
- http://docs.podman.io/en/latest/
- https://buildah.io/
- https://github.com/containers/skopeo
- https://www.openshift.com/blog/promoting-container-images-between-registries-with-skopeo
- https://github.com/containers/buildah
- https://developers.redhat.com/blog/2019/02/21/podman-and-buildah-for-docker-users/