Montar Servidor XMPP y BD de postgresql en Contenedor Docker

Acosta Rosales Jair Sebastián

6 de mayo de 2020

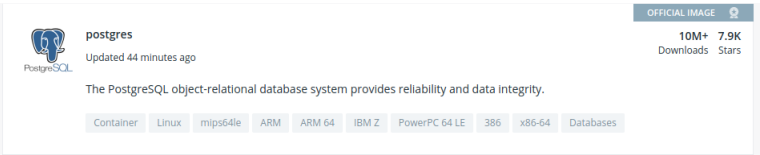
1. Introducción XMPP

XMPP, cuyo significado es Extensible Messaging and Presence Protocol, es un protocolo de mensajeria basado en XML, el cual permite un servicio de mensajerio entre usuarios que permite

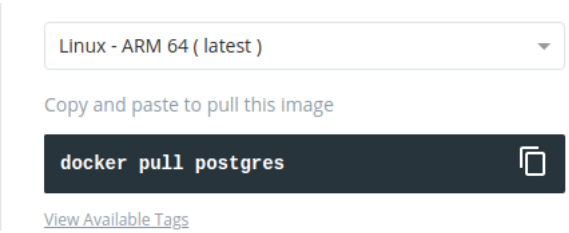
2. Descargando imagen Docker para Postgresql 11.4

Para este apartado no se creará un archivo Dockerfile para postgresql, en lugar de ello se implementará una imagen oficial desde la página `https://hub.docker.com` en donde es posible encontrar imagenes de múltiples servicios, uno de ellos de bases de datos, como lo pueden ser el propio postgrsql, mysql, etc.

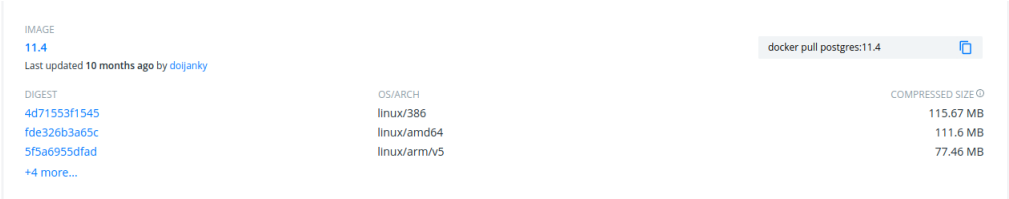
Por lo tanto, al acceder a la página hub.docker.com y buscar postgresql, deberá aparecer lo siguiente:



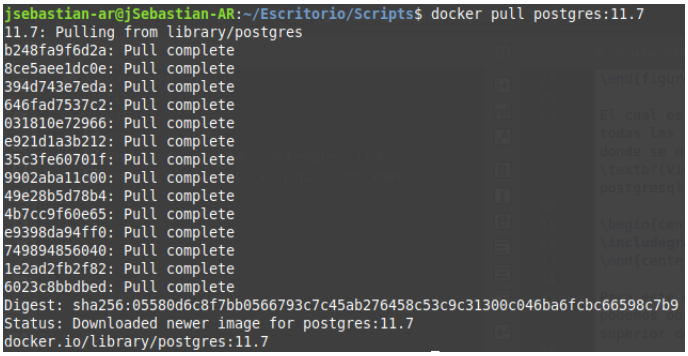
El cual es la imagen oficial para postgresql, haciendo click a este enlace nos dirigira al sitio con todas las imagenes posibles para descargar e implementar en docker, iremos al apartado de la derecha donde se encuentra un comando de docker, en la parte de abajo habrá un enlace con lo siguiente **View Available Tags** lo que significa que ahí podremos ver las versiones disponibles de postgresql, y descargar la que deseemos.



Para este ejemplo se busco y se selecciono la versión 11.4 de postgresql, en la siguiente imagen podemos observar el comando con el cual podemos descargar esta imagen, ubicado en la esquina superior derecha, el cual es: **docker pull postgres:11.4**

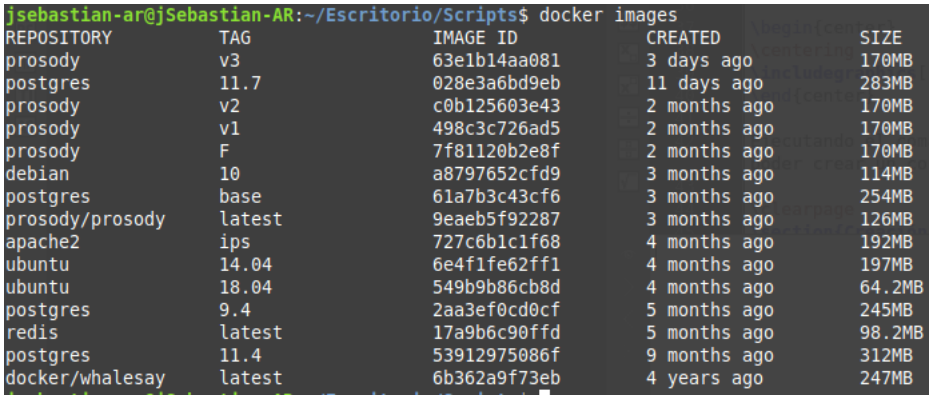


Ejecutando el comando en consola esperaremos a que la imagen sea descargada en nuestra computadora y poder crear un contenedor con este.



En esta imagen se hizo el pull con la versión 11.7 de postgresql

Una vez descargado la nueva imagen procedemos a revisar las imagenes disponibles en nuestra version local de docker, con el comando **docker images**, donde se desplegará la lista de todas las imágenes disponibles.



Así podemos corroborar que tenemos la imagen de postgresql para su futuro uso en un contenedor.

3. Creación de imagen Docker para servidor XMPP

Para la creación de una imagen personalizada del servidor XMPP, se hará uso del servidor Prosody, del cual se obtuvo un Dockerfile de la siguiente dirección: `https://github.com/prosody/prosody-docker`, que requiere de la descarga del archivo .deb para la instalación del servidor prosody, el cual se puede obtener en: `https://debian.pkgs.org/10/debian-main-amd64/prosody_0.11.2-1_amd64.deb.html`, siendo la versión de debian, pues el SO base que será usado para esta imagen será Debian 10, sin embargo, hubo ciertas modificaciones a este archivo, las cuales se listan:

- **Se elimino el entrypoint**
  La imagen que es creada mediante el dockerfile requiere de un entrypoint que debe usarse cuando el comando **docker run** para la creación del contenedor es ejecutado, el entrypoint es ejecutado dentro de las instrucciones del dockerfile y para que el servidor sea iniciado, un parámetro extra debe agregarse al comando "docker run", si el parámetro es agregado y el entrypoint lo reconoce, el servidor se inicia, si el entrypoint no lo detecta, entonces el servidor no se inicia y el contenedor muere a los pocos segundos de haber sido iniciado, pues no tiene un proceso en ejecución para que siga viviendo.

  La gran desventaja de trabajar con ese entrypoint es que cuando el contenedor es creado y se ejecuta correctamente(con el servidor XMPP funcionando), este contenedor no podía ser detenido(docker stop) o reiniciado(docker restart), pues cuando el contenedor quisiese iniciarse nuevamente, el entrypoint se ejecutaria y lo que sucederia es que necesita pasar el parametro para iniciar el servidor, cosa que en un comando como **docker start** o **docker restart**, no es posible pasar parámetros.

- Comandos de edición del archivo de configuracion **prosody.cfg.lua**, dado que ya se poseían esos archivos con la configuración deseada, por lo que era más fácil realizar un COPY de archivos dentro del Dockerfile, que la ejecución de comandos para editar los archivos.
- Se eliminaron el resto de puertos a ser expuestos, manteniendo solo el puerto 5222.

De modo que el archivo de configuración ha quedado de la siguiente forma:

```
FROM debian:10

# Install dependencies
RUN apt-get update \
    && DEBIAN_FRONTEND=noninteractive apt-get install -y --no-install-recommends \
        lsb-base \
        procps \
        adduser \
        libidn11 \
        libicu63 \
        libssl1.1 \
        lua-bitop \
        lua-dbi-mysql \
        lua-dbi-postgresql \
        lua-dbi-sqlite3 \
        lua-event \
        lua-expat \
        lua-filesystem \
        lua-sec \
        lua-socket \
        lua-zlib \
        lua5.1 \
        lua5.2 \
        openssl \
        ca-certificates \
        ssl-cert \
        nano \
        net-tools \
    && rm -rf /var/lib/apt/lists/*

# Install and configure prosody
COPY ./prosody.deb /tmp/prosody.deb
RUN dpkg -i /tmp/prosody.deb

COPY prosody.cfg.lua /etc/prosody/
COPY docker.com.key /var/lib/prosody/
COPY docker.com.crt /var/lib/prosody/

RUN chown prosody: /etc/prosody/prosody.cfg.lua && chown prosody: /var/lib/prosody/docker.com.key && chown prosody: /var/lib/prosody/
docker.com.crt

EXPOSE 5222

CMD ["prosodyctl", "start"]
```

En el podemos observar las distintas partes de instrucciones que lo conforman:

1. El SO base será Debian 10
2. Las dependencias requeridas para el servidor, como los certificados ssl, los modulos de configuración de bases de datos, el editor nano para poder visualizar los archivos y verificar su configuración, la paquetería que permite visualizar las interfaces con el comando **ifconfig**, todos estos paquetes instalados bajo la bandera -y, la cual hace referencia a la palabra zes.[ei]ndica que estamos deacuerdo con instalar los paquetes, asimismo indicamos que

no se instalen paquetes recomendados de debian que pueden aparecer en la pantalla al momento de crear la imagen.

3. Se copia y pega dentro del ambiente el paquete prosody.deb, el cual permitira instalar el servidor XMPP, además de que se ejecuta el comando correspondiente para la instalación de este.

4. Se copian y pegan tres archivos previamente configurados o creados, los cuales son:

   - **prosody.cfg.lua**: Contiene toda la configuración del funcionamiento de nuestro servidor, es copiado en la ruta correspondiente donde se instalaría normalmente los archivos de configuración de prosody, es decir /etc/prosody.
   - **docker.com.crt**: Es el certificado de seguridad ssl creado específicamente para el servidor, el cual permite cifrar los mensajes que los usuarios se estén enviando entre si.
   - **docker.com.key**: El archivo key es generado junto con el archivo .ctr y contiene la llave de cifrado de mensajes.

5. Una vez que los archivos son copiados, el comando **chown** es ejecutado para estos, con la finalidad de que el usuario "prosody"pueda leerlos y ejecutarlos.

6. Se expone el puerto 5222, visible para otros contenedores.

7. Como intrucción final se configura para que el comando ejecutado en consola sea **prosodyctl start** lo cual permite iniciar el servidor y por lo tanto, permite que el servicio se ejcute y mantenga vivo el contenedor, esta intrucción sustituye al entrypoint, que como se explico, traía consigo desventajas, pero con esta instrucción es posible detener o reiniciar la ejecución de un contenedor y que este al ser iniciado pueda ejecutarse sin problemas.

Una vez terminado de configurar el Dockerfile, se procede con la creación de la imagen correspondiente, a través del comando **docker build file_path_Dockerfile -t name:tag**.

Donde:

- **file_path_Dockerfile**: Es la ruta de la carpeta donde se encuentra ubicado el Dockerfile
- **-t**: Esta bandera hace referencia a que se agregara un tag en especial para identificar entre imágenes que posean el mismo nombre.
- **name:tag**: La parte name hace referencia al nombre que se le dará a la imagen, mientras que tag será la etiqueta única que identificará a esa imagen entre otras posibles con el mismo nombre.

Finalmente al crear la imagen:

```
sudo docker build prosody -t prosody:f
```

```
Successfully built d60c9ebf768c
Successfully tagged prosody:f
```

Obtenemos un mensaje exitoso de que la creación de la imagen ha terminado y si revisamos las imagenes con el comando **docker images**.

```
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_2/docker$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
prosody             f                   d60c9ebf768c        13 seconds ago      170MB
prosody             v3                  63e1b14aa081        3 days ago          170MB
postgres            11.7                028e3a6bd9eb        12 days ago         283MB
```

4. Creacion de subred para la conexión entre contenedores

Se creará una subred interna de docker del tipo bridge, con la finalidad de que ambos contenedores se encuentren en la misma red y puedan comunicarse entre si a través de sus direcciones ip.

Lo anterior se puede crear con el siguiente comando:

docker network create –driver bridge –subnet subred_deseada nombre_de_red

Lo anterior se puede entender de la siguiente forma:

1. –**driver bridge**: Esta bandera indica el tipo de red a la que estará conectado el contenedor, existen tres tipos:

   - **NONE**: Este tipo de red indica que será una red aislada, en la que el contenedor conectado no tendrá acceso ni a las interfaces de la computadora donde está ejecutandose ni tampoco a la comunicación con otros contenedores.
   - **HOST**: Este tipo de red, permite al contenedor tener acceso y conectividad a todas las interfaces de red, de la computadora donde se está ejecutando.
   - **BRIDGE**: Este tipo de red permite tanto conectividad con las interfaces de red de la computadora donde se está ejecutando y comunicación con otros contenedores.

     Para este caso, nosotros deseamos conectividad con otro contenedor XMPP,aunque no necesariamente que la BD este expuesta a otros servicios, pues solo el servidor Prosody se comunicará con este contenedor, de este modo se elige el tipo de subred bridge.

2. –**subnet subred_deseada**: Esta bandera nos permitirá indicar de que subred se tratará, así como las direcciones disponibles en ella, de modo que subred deseada será **192.168.10.0/24** con la notación **/24** indicando el número de bits que ocupará para la parte de la red, dejando el último octeto para los hosts disponibles en la red.

3. **nombre_de_red**: Este será el nombre que deseamos darle a la red para identificarla.

Para este caso el comando quedará de la siguiente manera.

docker network create –driver bridge –subnet 192.168.10.0/24 prosodynet

Una vez ejecutado el comando anterior podemos listar las redes de docker y notar que tenemos la nueva red.

```
jsebastian-ar@jSebastian-AR:/run$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
a6e92a6c2098        bridge              bridge              local
eef0b09685ef        host                host                local
f6d85fba7579        none                null                local
9578f9adf7da        prosodynet          bridge              local
61e663c147be        subnet2             bridge              local
3902cdb3d31f        subnet3             bridge              local
```

```
jsebastian-ar@jSebastian-AR:/run$ docker network inspect prosodynet
[
    {
        "Name": "prosodynet",
        "Id": "9578f9adf7da0c7246636aba0b4ce8a3eb4595c0b274c47f9b9215d3bb629ecd",
        "Created": "2020-05-01T20:27:24.520992558-05:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "192.168.10.0/24",
                    "Gateway": "192.168.10.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {},
        "Options": {},
        "Labels": {}
    }
]
```

Observamos más a detalle la información de la subred

5. Creación de los contenedores

Una vez que se tienen las imágenes, procederemos a crear los contenedores para la ejecución de los servicios.

Contenedor de Postgresql Para la creación del contenedor de postgresql, se ejecutara el siguiente comando:

docker run -d –name=nombre_contenedor
docker run -d –name=name_container -p host_port:container_port