

# Montar Servidor XMPP y BD de postgresql en Contenedor Docker

Acosta Rosales Jair Sebastián

5 de mayo de 2020

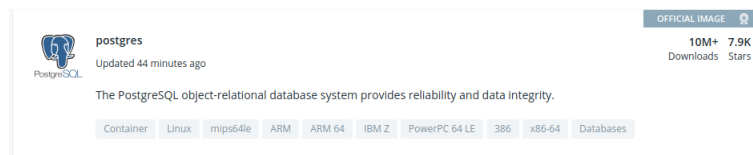
## 1. Introducción XMPP

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

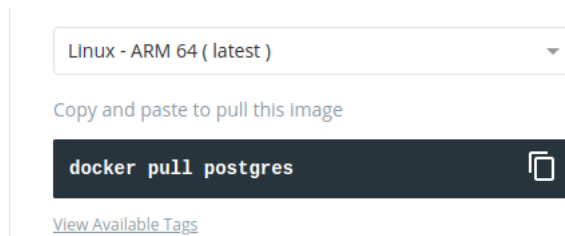
## 2. Descargando imagen Docker para Postgresql 11.4

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 imágenes de múltiples servicios, uno de ellos de bases de datos, como lo pueden ser el propio postgresql, mysql, etc.

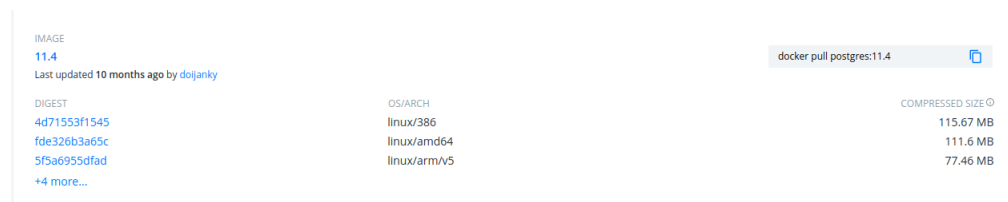
Por lo tanto, al acceder a la página [hub.docker.com](https://hub.docker.com) y buscar postgresql, deberá aparecer lo siguiente:



El cual es la imagen oficial para postgresql, haciendo click a este enlace nos dirigirá al sitio con todas las imágenes 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 esperamos a que la imagen sea descargada en nuestra computadora y poder crear un contenedor con este.

```
jsebastian-ar@jSebastian-AR:~/Escritorio/Scripts$ docker pull postgres:11.7
11.7: Pulling from library/postgres
b248fa9f6d2a: Pull complete
8ce5aeel1dc0e: Pull complete
394d743e7eda: Pull complete
646fad7537c2: Pull complete
031810e72966: Pull complete
e921d1a3b212: Pull complete
35c3fe60701f: Pull complete
9902aba11c00: Pull complete
49e28b5d78b4: Pull complete
4b7cc9f60e65: Pull complete
e9398da94ff0: Pull complete
749894856040: Pull complete
1e2ad2fb2f82: Pull complete
6023c8bbdbed: Pull complete
Digest: sha256:05580d6c8f7bb0566793c7c45ab276458c53c9c31300c046ba6fcbc66598c7b9
Status: Downloaded newer image for postgres:11.7
docker.io/library/postgres:11.7
```

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.

```
jsebastian-ar@jSebastian-AR:~/Escritorio/Scripts$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
prosody	v3	63e1b14aa081	3 days ago	170MB
postgres	11.7	028e3a6bd9eb	11 days ago	283MB
prosody	v2	c0b125603e43	2 months ago	170MB
prosody	v1	498c3c726ad5	2 months ago	170MB
prosody	F	7f81120b2e8f	2 months ago	170MB
debian	10	a8797652cfd9	3 months ago	114MB
postgres	base	61a7b3c43cf6	3 months ago	254MB
prosody/prosody	latest	9eae5f92287	3 months ago	126MB
apache2	ips	727c6b1c1f68	4 months ago	192MB
ubuntu	14.04	6e4f1fe62ff1	4 months ago	197MB
ubuntu	18.04	549b9b86cb8d	4 months ago	64.2MB
postgres	9.4	2aa3ef0cd0cf	5 months ago	245MB
redis	latest	17a9b6c90ffd	5 months ago	98.2MB
postgres	11.4	53912975086f	9 months ago	312MB
docker/whalesay	latest	6b362a9f73eb	4 years ago	247MB

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://deb.debian.org/debian/main-amd64/prosody\\_0.11.2-1\\_amd64.deb](https://deb.debian.org/debian/main-amd64/prosody_0.11.2-1_amd64.deb), 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 entryptpoint**

La imagen que es creada mediante el dockerfile requiere de un entryptpoint que debe usarse cuando el comando **docker run** para la creación del contenedor es ejecutado, el entryptpoint 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 entryptpoint lo reconoce, el servidor se inicia, si el entryptpoint 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 entryptpoint 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 entryptpoint 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 configuración **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 \
        libc6 \
        libssl1.1 \
        lua-bitop \
        lua-dbi-mysql \
        lua-dbi-postgresql \
        lua-dbi-sqlite3 \
        lua-event \
        lua-expat \
        luafilesystem \
        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 módulos 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 *yes*.<sup>e</sup>Indica que estamos de acuerdo 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 `.crt` 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 intrucción 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
prosody	v3	60b135603e43	2 months ago	170MB

```
docker run -d --name=name_container -p host_port:container_port
```