# Montar Servidor XMPP y BD de postgresql en Contenedor Docker

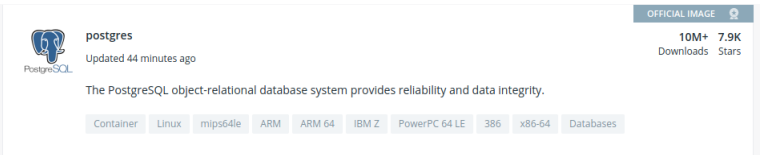
Acosta Rosales Jair Sebastián

14 de mayo de 2020

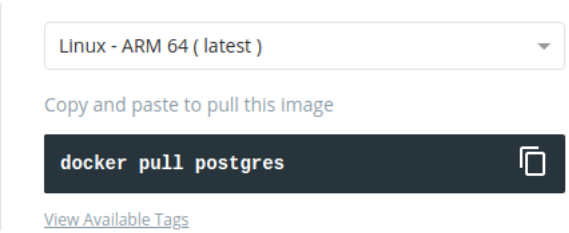
## 1. Descargando imagen Docker para Postgresql 11.4

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.

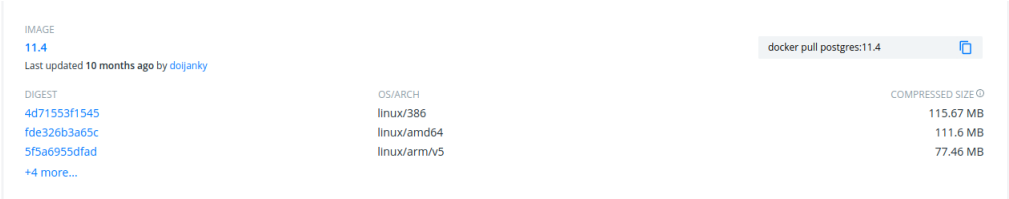
Por lo tanto, al acceder a la página hub.docker.com y buscar postgresql, deberá aparecer lo siguiente:



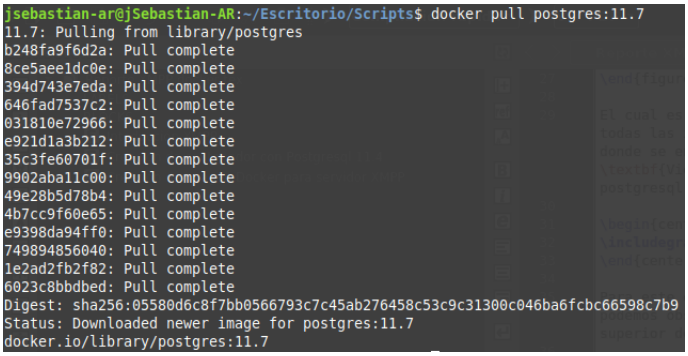

El cual es la imagen oficial para postgresql, haciendo click a este enlace nos dirigira al sitio con todas las imagenes posibles para descargar e implementar en docker, iremos al apartado de la derecha donde se encuentra un comando de docker, en la parte de abajo habrá un enlace con lo siguiente **View 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**

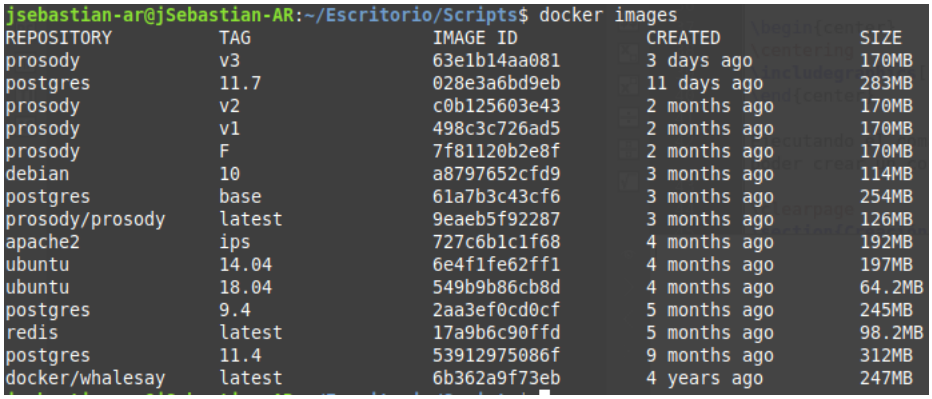



Ejecutando el comando en consola esperaremos a que la imagen sea descargada en nuestra computadora y poder crear un contenedor con este.




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.

## 2. 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

RUN mkdir -p /var/run/prosody && chown prosody:prosody /var/run/prosody

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 crea la carpeta donde se almacenara el archivo .pid del proceso de prosody, este archivo pid contendrá el id único del proceso en el contenedor, **nota: Se identifico que sin está instrucción el servidor no es capaz de mantenerse activo, se especula que es debido a que necesita de esta carpeta para poder almacenar el id del proceso y por lo tanto poder asignarle uno.**

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

8. 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
```

# 3. 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.

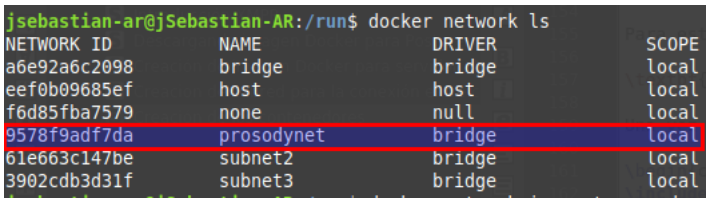
2. –**subnet subred_deseada**: Esta bandera nos permitirá indicar de que subred se tratará, así como las direcciones disponibles en ella, de modo que subred deseada será **192.168.10.0/24** con la notación **/24** indicando el número de bits que ocupará para la parte de la red, dejando el último octeto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

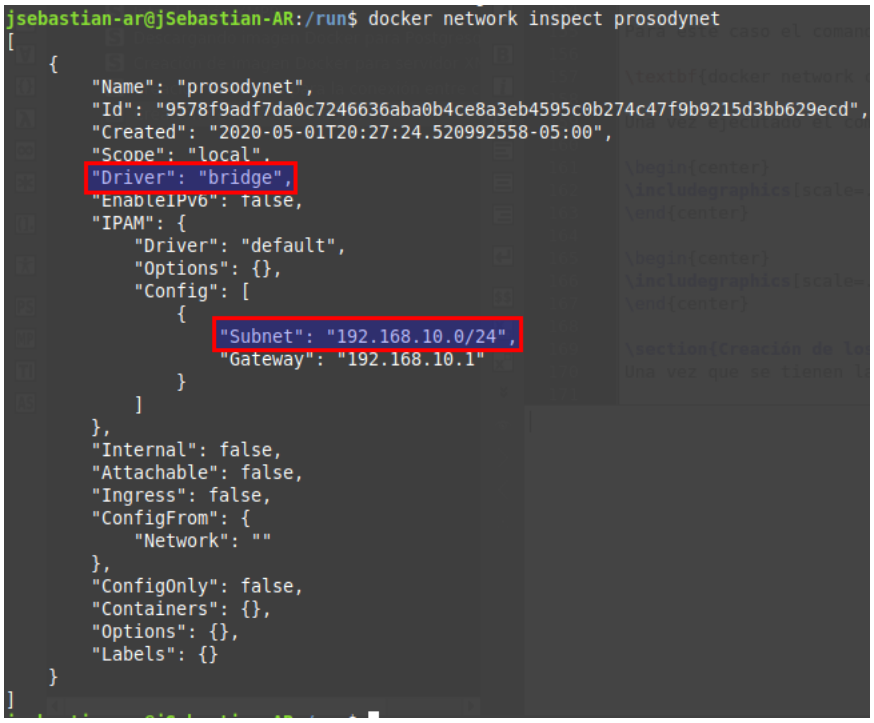
3. **nombre_de_red**: Este será el nombre que deseamos darle a la red para identificarla.

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.






Observamos más a detalle la información de la subred

# 4. Creación y configuración de los contenedores

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

## 4.1. Contenedor de Postgresql

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

**docker run -d –name nombre_contenedor –subnet nombre_subred –ip dir_ip nombre_imagen**

Donde:

1. **-d**: Bandera que ejecuta la acción en *detached mode*, esto significa que el contenedor y servidor se ejecutara pero de forma oculta, no será posible funcionar el contenedor al momento de ejecutar el comando.

2. –**name**: Permite asignar un nombre al contenedor

3. –**subnet**: Permite asignar la subred a la que deseamos que el contenedor pertenezca (aquí se especificara la subred que creamos).

4. –**ip**: Permite especificar la dirección ip que deseamos que tenga el contenedor, esto será útil para realizar la conexión entre ambos contenedores, pues en el archivo de configuración de Prosody *Prosody.cfg.lua* debemos especificar la ip donde se encuentra la base de datos.

```
sql = { driver = "PostgreSQL", database = "prosody", username = "postgres", password = "postgres", host = "192.168.10.3" }
```

Línea de configuración de la BD en el archivo Prosody.cfg.lua

5. **nombre_imagen**: Nombre de la imagen a la que le será creado un contenedor.

Finalmente el comando queda de la siguiente manera:

**docker run -d –name bdpg –subnet prosodynet –ip 192.168.10.3 postgres:11.4**

Una vez ejecutado, podemos verlo en ejecución con ayuda del comando *docker ps*

```
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_2/docker$ docker run -d --name bdpg --network prosodynet --ip 192.168.10.3 postgres:11.4
e712f246101ad0a2d85dad07a57e3b85664e15b8e97667c654a70663d41e9100
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_2/docker$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
e712f246101a        postgres:11.4       "docker-entrypoint.s…"   11 seconds ago      Up 9 seconds        5432/tcp            bdpg
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_2/docker$
```

Con el comando *docker inspect nombre_contenedor* podremos ver las especificaciones de un contenedor en particular, en este caso "bdpg". En la siguiente imagen se observan las especificaciones de la red a la que fue asignado y su ip, como era de esperarse.


```
"Networks": {
    "prosodynet": {
        "IPAMConfig": {
            "IPv4Address": "192.168.10.3"
        },
        "Links": null,
        "Aliases": [
            "e712f246101a"
        ],
        "NetworkID": "9578f9adf7da0c7246636aba0b4ce8a3eb4595c0b274c47f9b9215d3bb629ecd",
        "EndpointID": "d9e41d46c577f77ef3c12c3e8c70fba089d298e4673569ba2e1549201e5221d6",
        "Gateway": "192.168.10.1",
        "IPAddress": "192.168.10.3",
        "IPPrefixLen": 24,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "MacAddress": "02:42:c0:a8:0a:03",
        "DriverOpts": null
```

## 4.2. Configuración de contenedor bdpg

Una vez creado el contenedor, procedemos a crear la base de datos donde serán almacenados los mensajes que el servidor XMPP reciba y envie. Para ello accederemos al contenedor con el comando **docker exec -it bdpg bash**, este comando nos permite acceder a la consola del contenedor para poder observar su funcionamiento.

```
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_2/docker$ docker exec -it bdpg bash
root@e712f246101a:/#
```

Ahora procedemos a ejecutar el comando para crear la base de datos, el cual es: *createdb -U postgres name_ db*, donde name_db es el nombre de la base de datos que especificamos en el archivo de configuración *Línea de configuración de la BD en Prosody*

De modo que el comando a ejecutar será:

```
root@e712f246101a:/# createdb -U postgres prosody
```

Listando las BD del contenedor

```
postgres=# \l
                                  List of databases
   Name    |  Owner   | Encoding |  Collate   |   Ctype    |   Access privileges
-----------+----------+----------+------------+------------+-----------------------
 postgres  | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 prosody   | postgres | UTF8     | en_US.utf8 | en_US.utf8 |
 template0 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
 template1 | postgres | UTF8     | en_US.utf8 | en_US.utf8 | =c/postgres          +
           |          |          |            |            | postgres=CTc/postgres
(4 rows)
```

Observamos que la base de datos recién creada no contiene ninguna tabla, pues el servidor Prosody creara automáticamente estas tablas al hacer los primeros registros de mensajes en ella.

```
prosody=# \dt
Did not find any relations.
prosody=#
```

## 4.3. Contenedor de Prosody

Para el contenedor de prosody se ejecutara el siguiente comando

**docker run -d –name xmpp –subnet prosodynet –ip 192.168.10.2 -p host_port:container_port prosody:f**

En este comando podemos notar una nueva bandera a diferencia del que se ejecuto para Postgresql, la cual es **-p**, esta bandera nos permite conectar un puerto de nuestro host(computadora) a un puerto del contenedor a ser creado, en este caso conectaremos el puerto 5222 del contenedor dado que es el predeterminado para el servidor Prosody, y para el servicio XMPP en general. De

igual forma, el puerto elegido del host será el 5222.

Finalmente el comando queda:
**docker run -d –name xmpp –subnet prosodynet –ip 192.168.10.2 -p 5222:5222 prosody:f**

```
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_2/docker$ docker run -d --name xmpp --network prosodynet --ip 192.168.10.2 -p 5222:5222 prosody:f
630c303f3734d89cbc38e987f940696790d98ec3c44feae57e6dc377fad842f1
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_2/docker$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                    NAMES
630c303f3734        prosody:f           "prosodyctl start"       5 seconds ago       Up 3 seconds        0.0.0.0:5222->5222/tcp   xmpp
e712f246101a        postgres:11.4       "docker-entrypoint.s…"   10 minutes ago      Up 10 minutes       5432/tcp                 bdpg
jsebastian-ar@jSebastian-AR:~/Documentos/git-repos/ASR/Practica_2/docker$
```

## 4.4. Configuración de contenedor xmpp

Ingresando a la consola del contenedor con el comando **docker exec -it xmpp bash**. Crearemos dos usuarios para el servidor Prosody, de modo que estos puedan mandarse mensajes y estos mensajes sean almacenados en la base de datos en el contenedor *bdpg*.

Para ello ejecutaremos comandos específicos del servidor prosody, como lo es: **prosodyctl adduser JID**, donde **JID** es el nombre que le daremos al usuario, pero siempre terminando con el nombre del host virtual especificado en el archivo de configuración de prosody (en este caso el nombre del host virtual es "docker.com") acompañado de un "@".

```
VirtualHost "docker.com"
ssl = {
    certificate = "/var/lib/prosody/docker.com.crt";
    key         = "/var/lib/prosody/docker.com.key";

}
```

Apartado del archivo Prosody.cfg.lua donde se declara el host virtual

```
root@630c303f3734:/# prosodyctl adduser dsebas-lap@docker.com
Enter new password:
Retype new password:
root@630c303f3734:/# prosodyctl adduser dsebas-mi@docker.com
Enter new password:
Retype new password:
root@630c303f3734:/#
```

Dos usuarios son creados con sus respectivas contraseñas

# 5. Configuración de cuentas

Una vez hechas las configuraciones correspondientes en los contenedores, procedemos a realizar las pruebas de intercambio de mensajes entre dos usuarios, concretamente:

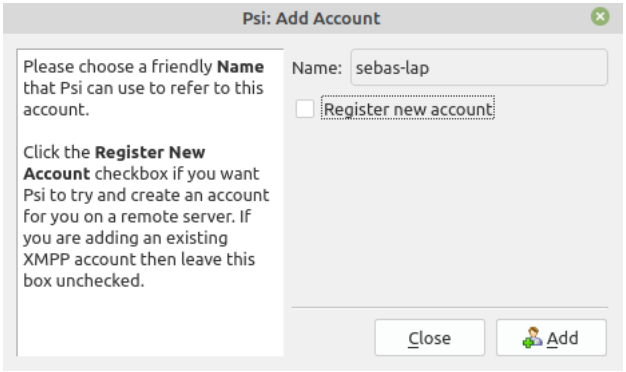
- **dsebas-lap@docker.com**: Usuario de laptop, que enviará mensajes a través de la aplicación *Psi*.

- **dsebas-mi@docker.com**: Usuario de smartphone, conectado por medio de la aplicación *Bruno the Jabber Bear (XMPP)*.

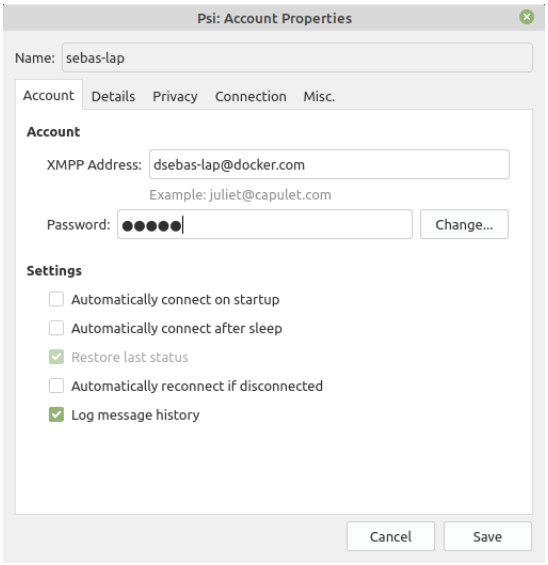
## 5.1. Configurando cuenta en Psi

Para configurar una nueva cuenta en Psi, es necesario seleccionar el ícono de la aplicación y seleccionar la opción .ªdd contact".

Procedemos a darle un nombre a la cuenta dentro del equipo.

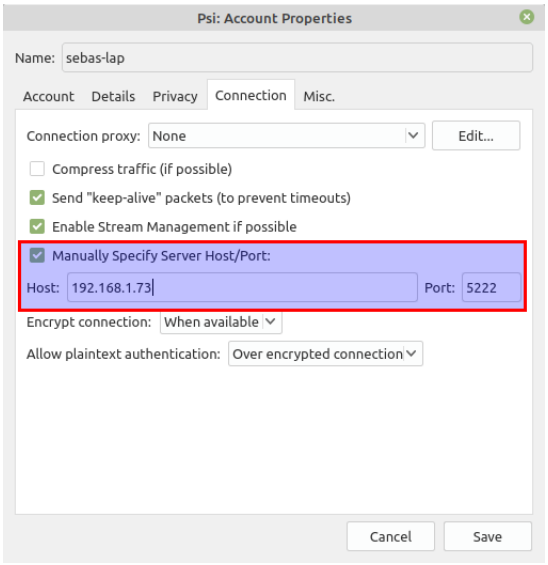



Ahora se agregará el nombre de la cuenta que fue creada en el servidor prosody (*dsebas-lap@docker.com*) exclusivamente para el usuario de laptop con su respectiva contraseña.

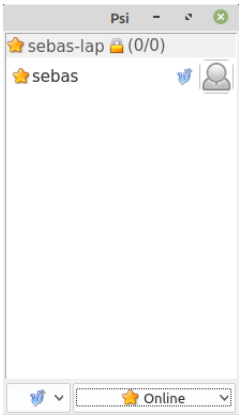



En el apartado connection se da check para especificar la dirección del host (la dirección del equipo donde estarán corriendo los contenedores), para saber

su dirección ip, basta con ejecutar en consola el comando **ifconfig**, además de que se especifica el puerto donde se atenderán las peticiones, mismo puerto que fue conectado al puerto interno del contenedor.



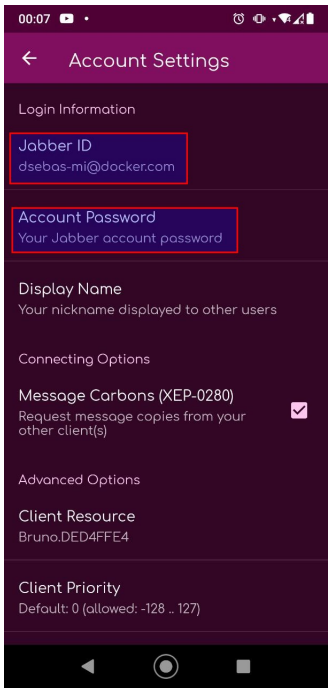

Terminada la configuración procedemos a poner la cuenta en modo online, aparecerá una ventana emergente preguntando si confiamos en los certificados de seguridad del servidor, aceptamos (*Trust this certificate*); después se pedirá que ingresemos la contraseña de nuestra cuenta nuevamente; además pedira llenar un registro con algunos de nuestros datos (bastara con poner nuestro nombre y nickname); y finalmente veremos la cuenta en linea, esto quiere decir que se ha conectado exitosamente al servidor Prosody que se está ejecutando en el contenedor.



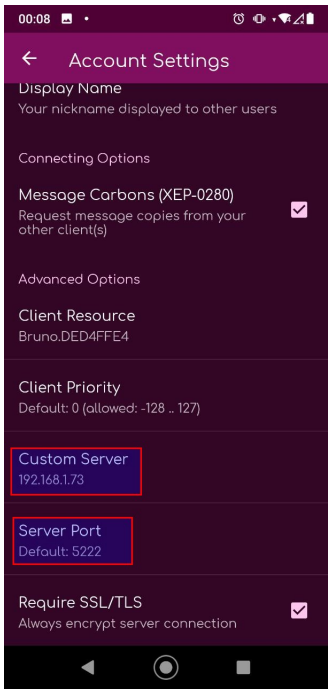
## 5.2. Configuración de cuenta en Bruno the Jabber Bear (XMPP)

Al abrir la aplicación deberemos ir al apartado de configuración y posteriormente en configuración de cuenta:

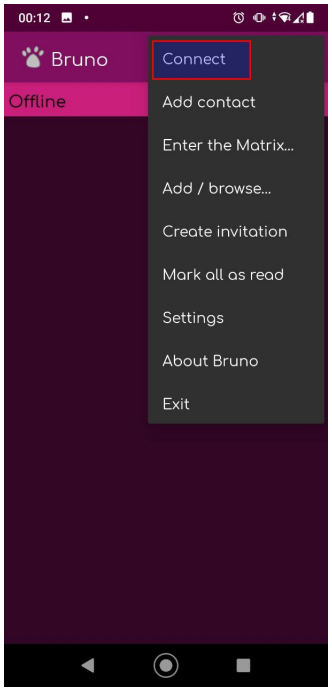
Aquí se agregará la información de la cuenta creada para smartphone desde Prosody (*dsebas-mi@docker.com*) y su respectiva contraseña.




Ahora se agregará la configuración del host

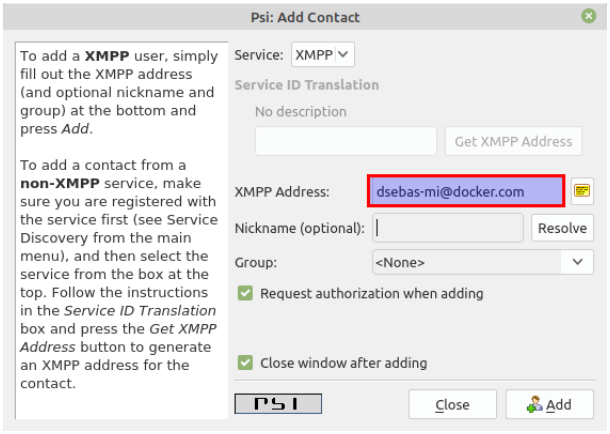



Finalmente se procederá a conectar la cuenta:

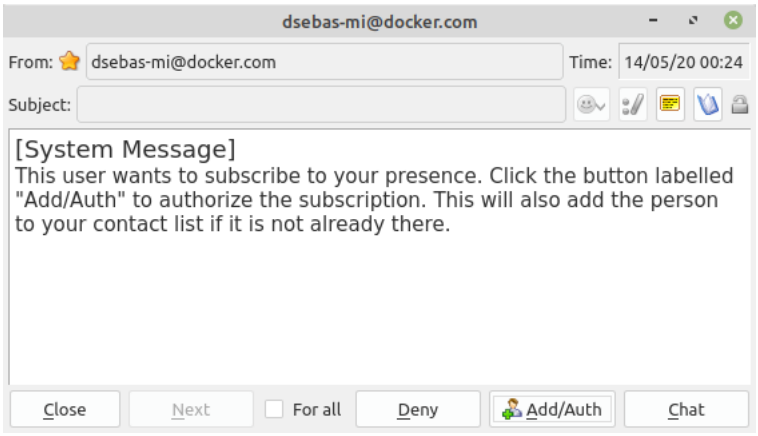


Una vez que ambas cuentas están conectadas, para que ambas cuentas se puedan enviar mensajes, deben agregarse, esto puede ser desde la cuenta en *Psi* o *Bruno*, en este caso se realizo desde Psi.

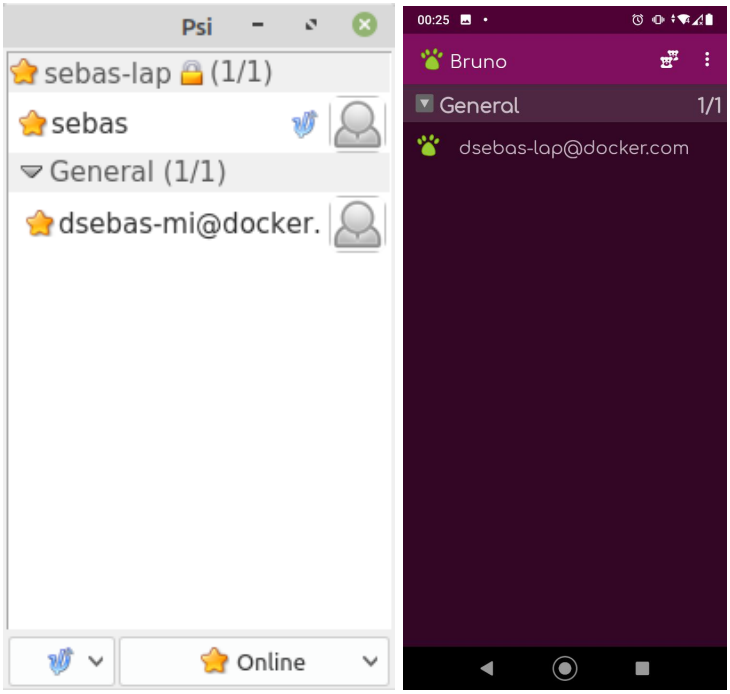
En el apartado de .ªdd contact", debemos escribir el nombre del usuario (registrado en Prosody) que deseamos agregar a nuestros contactos.




Una vez que lo agregamos, nos llegará una solicitud automática para que el usuario agregado, también nos pueda agregar desde su dispositivo.

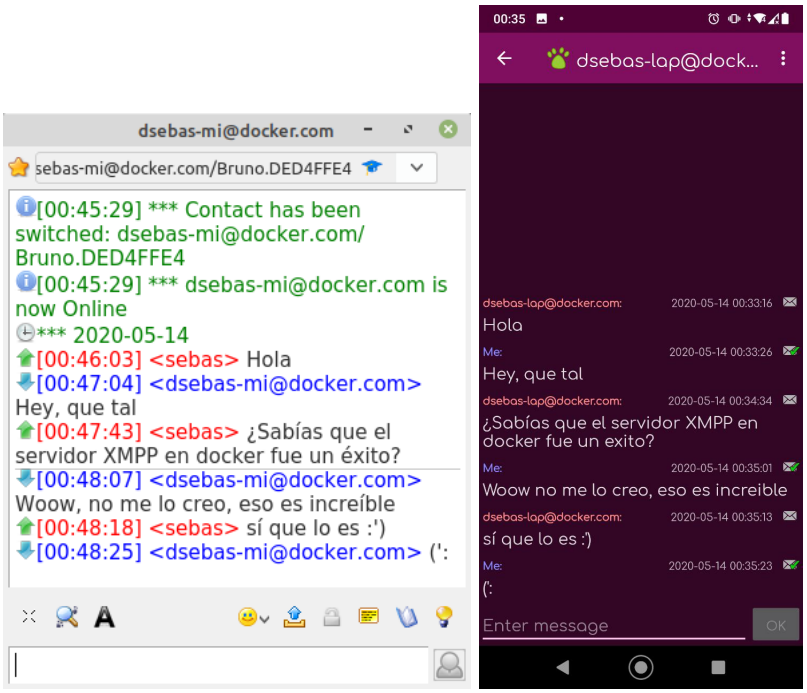



Una vez aceptado es posible que ambos desde sus respectivos dispositivos vean al otro y se puedan comunicar.




# 6. Pruebas

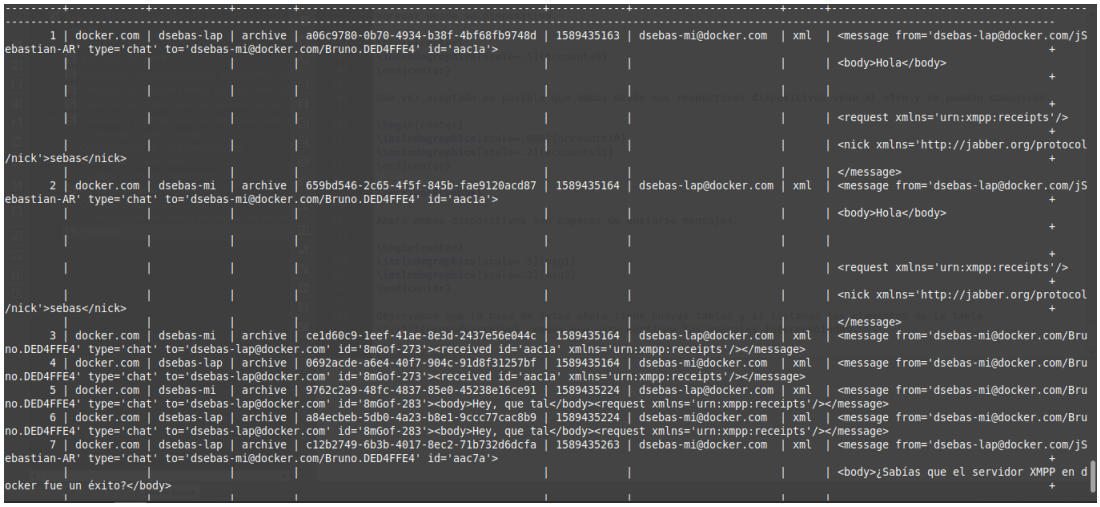
Ahora ambos dispositivos son capaces de enviarse mensajes.




Observamos que la base de datos ahora tiene nuevas tablas y si listamos los elementos de la tabla *prosodyarchive*, vemos que esta contiene los mensajes intercambiados.

```
prosody=# \dt
                List of relations
 Schema |      Name      | Type  |  Owner
--------+----------------+-------+----------
 public | prosody        | table | postgres
 public | prosodyarchive | table | postgres
(2 rows)
```

Mensajes almacenados



```
--------+-----------+-----------+--------+--------------------------------------+------------+----------------------+------+-----------------------------------
-------------------------------------------------------------------------------------------------------------------------------------------------------------
       1 | docker.com | dsebas-lap | archive | a06c9780-0b70-4934-b38f-4bf68fb9748d | 1589435163 | dsebas-mi@docker.com  | xml  | <message from='dsebas-lap@docker.com/jS
ebastian-AR' type='chat' to='dsebas-mi@docker.com/Bruno.DED4FFE4' id='aac1a'>                                                                                +
         |            |            |         |                                      |            |                       |      | <body>Hola</body>
                                                                                                                                                             +
         |            |            |         |                                      |            |                       |      |
                                                                                                                                                             +
         |            |            |         |                                      |            |                       |      | <request xmlns='urn:xmpp:receipts'/>
                                                                                                                                                             +
         |            |            |         |                                      |            |                       |      | <nick xmlns='http://jabber.org/protocol
/nick'>sebas</nick>                                                                                                                                          +
         |            |            |         |                                      |            |                       |      | </message>
       2 | docker.com | dsebas-mi  | archive | 659bd546-2c65-4f5f-845b-fae9120acd87 | 1589435164 | dsebas-lap@docker.com | xml  | <message from='dsebas-lap@docker.com/jS
ebastian-AR' type='chat' to='dsebas-mi@docker.com/Bruno.DED4FFE4' id='aac1a'>                                                                                +
         |            |            |         |                                      |            |                       |      | <body>Hola</body>
                                                                                                                                                             +
         |            |            |         |                                      |            |                       |      |
                                                                                                                                                             +
         |            |            |         |                                      |            |                       |      | <request xmlns='urn:xmpp:receipts'/>
                                                                                                                                                             +
         |            |            |         |                                      |            |                       |      | <nick xmlns='http://jabber.org/protocol
/nick'>sebas</nick>                                                                                                                                          +
         |            |            |         |                                      |            |                       |      | </message>
       3 | docker.com | dsebas-mi  | archive | ce1d60c9-1eef-41ae-8e3d-2437e56e044c | 1589435164 | dsebas-lap@docker.com | xml  | <message from='dsebas-mi@docker.com/Bru
no.DED4FFE4' type='chat' to='dsebas-lap@docker.com' id='8mGof-273'><received id='aac1a' xmlns='urn:xmpp:receipts'/></message>
       4 | docker.com | dsebas-lap | archive | 0692acde-a6e4-40f7-904c-91d8f31257bf | 1589435164 | dsebas-mi@docker.com  | xml  | <message from='dsebas-mi@docker.com/Bru
no.DED4FFE4' type='chat' to='dsebas-lap@docker.com' id='8mGof-273'><received id='aac1a' xmlns='urn:xmpp:receipts'/></message>
       5 | docker.com | dsebas-mi  | archive | 9762c2a9-48fc-4837-85e0-45238e16ce91 | 1589435224 | dsebas-lap@docker.com | xml  | <message from='dsebas-mi@docker.com/Bru
no.DED4FFE4' type='chat' to='dsebas-lap@docker.com' id='8mGof-283'><body>Hey, que tal</body><request xmlns='urn:xmpp:receipts'/></message>
       6 | docker.com | dsebas-lap | archive | a84ecbeb-5db0-4a23-b8e1-9ccc77cac8b9 | 1589435224 | dsebas-mi@docker.com  | xml  | <message from='dsebas-mi@docker.com/Bru
no.DED4FFE4' type='chat' to='dsebas-lap@docker.com' id='8mGof-283'><body>Hey, que tal</body><request xmlns='urn:xmpp:receipts'/></message>
       7 | docker.com | dsebas-lap | archive | c12b2749-6b3b-4017-8ec2-71b732d6dcfa | 1589435263 | dsebas-mi@docker.com  | xml  | <message from='dsebas-lap@docker.com/jS
ebastian-AR' type='chat' to='dsebas-mi@docker.com/Bruno.DED4FFE4' id='aac7a'>                                                                                +
         |            |            |         |                                      |            |                       |      | <body>¿Sabias que el servidor XMPP en d
ocker fue un éxito?</body>                                                                                                                                   +
         |            |            |         |                                      |            |                       |      |
```