

High Availability, Load Balancing y Replication con **PostgreSQL**



Juan Antonio Velasco Gómez
Miguel Sánchez Maldonado

Índice del trabajo

1. Bibliografía
2. Introducción a PostgreSQL
3. La alta disponibilidad en PostgreSQL
 - 3.1. ¿Cómo podemos definir la alta disponibilidad?
 - 3.2. Características
4. Replicación en PostgreSQL
 - 4.1. Introducción a la replicación en PostgreSQL
 - 4.2. Métodos de replicación
5. Balanceo de cargas en PostgreSQL
 - 5.1. Sobrecarga de la base de datos
6. Ejemplo de uso de PostgreSQL
 - 6.1. Instalación de PostgreSQL
 - 6.2. Creación de la base de datos
 - 6.2.1. Primeros comandos
 - 6.2.2. Creación de la base de datos y las tablas
 - 6.3. Configurando servidor maestro
 - 6.4. Configurando servidor esclavo
 - 6.5. Replicando la base de datos inicial
7. Conclusiones

1. Bibliografía

1. Wiki de PostgreSQL
<http://www.postgresql.org/docs/8.3/static/high-availability.html>
2. Página oficial de PostgreSQL
<http://www.postgresql.org/about/>
3. Curso alta disponibilidad
<https://www.youtube.com/watch?v=v3p-VEKmrnY>
4. PostgreSQL Enterprise DB
<http://www.enterprisedb.com/products-services-training/pgdownload>
5. Replicación en PostgreSQL
<http://www.rassoc.com/greg/weblog/2013/02/16/zero-to-postgresql-streaming-replication-in-10-mins/>
6. Replicación en PostgreSQL (II)
<https://www.digitalocean.com/community/tutorials/how-to-set-up-master-slave-replication-on-postgresql-on-an-ubuntu-12-04-vps>

2. Introducción a PostgreSQL



Figura 1: Logotipo de PostgreSQL

https://wiki.postgresql.org/wiki/File:PostgreSQL_logo.3colors.svg

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con código fuente disponible de forma gratuita. Utiliza un modelo cliente-servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema.

Es un proyecto que no es manejado por una empresa y/o persona sino que es una comunidad de desarrolladores los que trabajan en su desarrollo. Dicha comunidad se denomina PGDG (PostgreSQL Global Development Group). Nació en la Universidad de Berkeley en 1977 y que tiene mucha historia detrás de él.

file:///home/juanvelasco/Universidad/SWAPArchivos/PostgreSQL

También es posible que escuchemos el nombre de “Postgres” como abreviatura para PostgreSQL puesto que este fue su nombre original.

Actualmente su versión más estable es la 9.3.5 aunque recientemente (el 5 de Abril del 2015) salió a la luz la versión 9.4.1 como versión más reciente.

Al ser de código abierto, PostgreSQL está disponible para todas las plataformas, desde Linux, UNIX(AIX,BSD,SGI IRIX, Mac OS X, Solaris) y Windows. En general, incluye todos los tipos de datos que integra SQL desde 2008, véase enteros, numérico, booleano, char, varchar, date, etc.

PostgreSQL es capaz de soportar más de una docena de lenguajes, incluidos los famosos Java, Perl, Python, Tcl, Ruby, C, C++ o su propio lenguaje PL/pgSQL (que es similar al de Oracle PL/SQL). Cuenta también con una extensa biblioteca de interfaces, permitiendo compilar e interpretar varios lenguajes a una interfaz con PostgreSQL. Existen interfaces para todos los lenguajes descritos anteriormente.

Pero lo mejor de todo es que el código fuente de PostgreSQL está disponible para su modificación y distribución gratuita. PostgreSQL no es sólo un potente sistema de bases de datos para empresas sino también una plataforma muy útil para casas, webs o productos que lo requieran.

Podríamos definir PostgreSQL con algunos atributos, como por ejemplo, su estabilidad, su potencia, su robustez, su facilidad de administración y sus estándares. Su documentación es también muy importante, puesto que al estar bien documentada, hace que se pueda entender fácilmente.

Algunas de las empresas más conocidas que utilizan este sistema de gestión de bases de datos son Skype, la Sociedad Estadounidense de Química, el Instituto Geográfico Francés (que tiene todos sus mapas montados sobre PostgreSQL) ó IMDb (base de datos de películas en internet).

A continuación y a lo largo del trabajo, iremos desarrollando algunas de las características más importantes de este sistema de gestión de bases de datos.

3. La alta disponibilidad en PostgreSQL

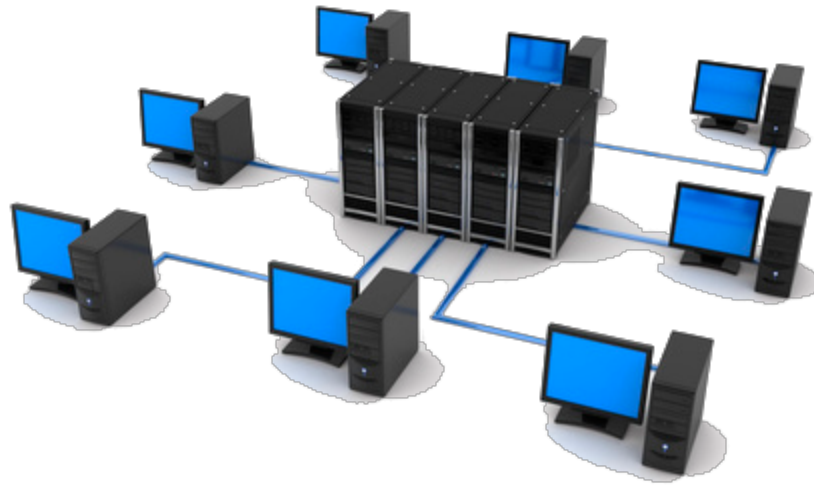


Figura 2: Sistemas de alta disponibilidad

<http://www.damitel.com/wp-content/uploads/2010/12/altadisponibilidad.png>

3.1. ¿Cómo podemos definir la alta disponibilidad? [High availability]

Se define como un protocolo de diseño del sistema y su implementación asociada que asegura un cierto grado absoluto de continuidad operacional durante un período de mediación dado. Cuando hablamos de disponibilidad nos estamos refiriendo a la habilidad de la comunidad de usuarios para acceder al sistema, para someter nuevos trabajos, para actualizar/alterar trabajos ya existentes o bien para recoger los resultados de trabajos previos.

Cuando un usuario no puede acceder al sistema entonces decimos que no está disponible. Ese tiempo de inactividad se conoce como *downtime*.

Al ser un protocolo de diseño, cada persona puede diseñarlo de una manera diferente. No es único. La alta disponibilidad viene definida por el resultado que se obtiene, no por la estrategia que se sigue a implementar. Se puede medir en términos de “tiempo en activo” y “tiempo caído”.

Usualmente, la disponibilidad de un sistema se mide como un porcentaje del tiempo de funcionamiento en un año dado. Algunos valores comunes de disponibilidad, típicamente enunciado como el número de “nueves” para sistemas altamente

disponibles, son:

- 99,9% = 43.8 minutos/mes u 8,76 horas/año ("tres nueves")
- 99,99% = 4.38 minutos/mes o 52.6 minutos/año ("cuatro nueves")
- 99,999% = 0.44 minutos/mes o 5.26 minutos/año ("cinco nueves")

El principal problema que tiene la alta disponibilidad son las caídas de servicios y los tiempos de recuperación ante ellos. Lo ideal sería buscar que nuestra estrategia fallase lo menos posible y además, en caso de hacerlo, tuviese un tiempo de recuperación mínimo. En el área de las bases de datos se busca mantener esa "calidad" que se ofrece o mejorarla.

3.2. Características

Recuperar un sistema caído lleva su tiempo. La estrategia que se sigue en estos casos, es, por excelencia, la redundancia de recursos. Se busca realizar algunas copias (replicaciones) de los datos que tenemos en nuestra bases de datos por si el sistema falla.

Esto nos lleva a introducir el siguiente concepto y punto a tratar en el trabajo: La replicación de datos en PostgreSQL.

4. Replicación en PostgreSQL

4.1. Introducción a la replicación en PostgreSQL

La replicación de los datos es un proceso por el cual copiamos y mantenemos actualizados los datos en varios nodos de las bases de datos. También se pueden replicar los recursos de red, de seguridad, etc.

Uno de los problemas que tienen las réplicas es que no siempre tienen el mismo tamaño que la base de datos original.

4.2. Métodos de replicación

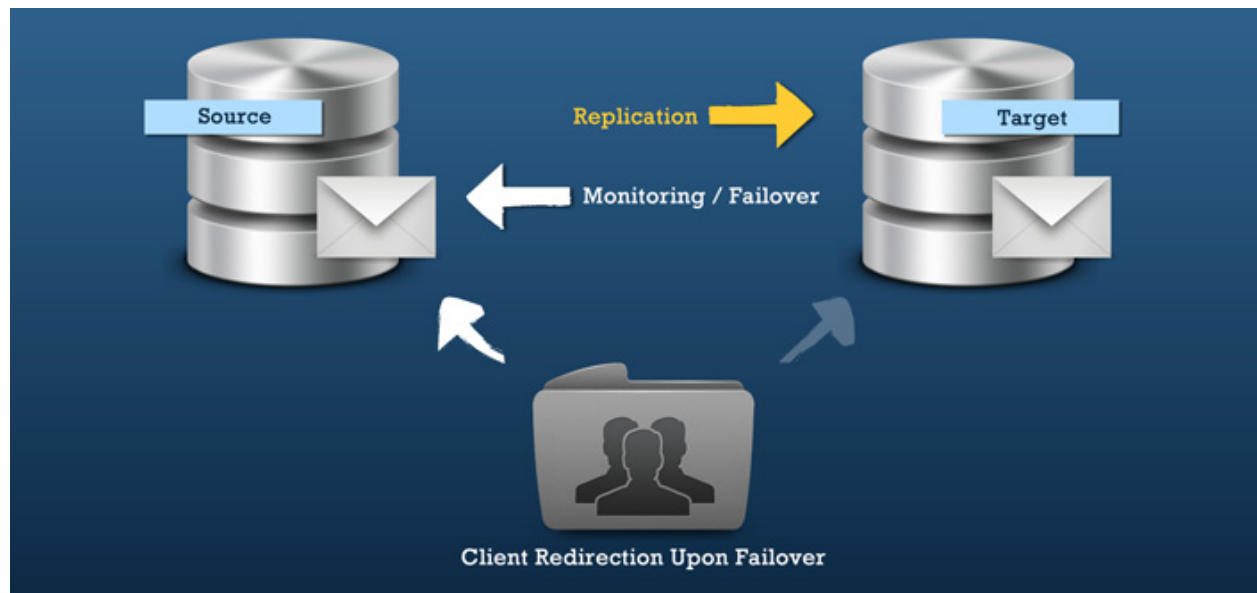


Figura 3: Replicación de una base de datos
<http://www.pandataservices.com/soluciones/alta-disponibilidad/>

Existen muchos métodos de replicación de datos en una base de datos, algunos de los más utilizados o conocidos son los siguientes:

Replicación de transacciones: Se copia cada una de las transacciones ejecutadas en el sistema donde se encuentran conectados los usuarios hacia un conjunto de bases de datos secundarias.

Estas bases de datos secundarias reciben los cambios propagados por la base de datos primaria a través de un canal de comunicación. En esta técnica existe el posible inconveniente de las diferencias generadas por funciones como *now()* y *random()* que pueden darnos problemas y deberíamos tener mucho cuidado a la hora de usarlas en la replicación.

Replicación por bitácoras: Este método está basado en ver los resultados de lo que hace la base de datos, no lo que está haciendo. Es un método que se usa con frecuencia y que está basado en el envío de logs (que son archivos binarios donde se registran todas las sentencias SQL de modificación de los datos o de su estructura). Se suele utilizar en porciones más pequeñas de nuestra base de datos.

Replicación con formatos específicos: Para cada replicación se ofrece una solución diferente. Tiene muchas ventajas pero también riesgos como que son demasiado personalizadas y privadas, por lo que a un caso particular es difícil que se pueda adaptar. Cuando se diseñan bien funcionan a la perfección.

5. Balanceo de cargas en PostgreSQL

5.1. Sobrecarga de la base de datos.

Otro de los problemas más comunes en una base de datos es la sobrecarga de la misma. Muchas veces sobrepasamos el límite de datos e información que nuestra base de datos puede manejar o mantener.

El concepto de **balance de carga** para mejorar la lectura de datos está generalmente asociado al concepto de la replicación de los mismos.

Algunas soluciones para la alta disponibilidad de una base de datos incorporan balanceadores de carga dentro de sus características más esenciales. Otro simplemente delegan esta responsabilidad sobre otros programas altamente dependientes de la plataforma de Sistema Operativo u otros programas de terceros.

Si no tenemos servidores idénticos, podemos balancear, en las capacidades que cada servidor tenga, cuánto nos va a ocupar el trabajo que estemos considerando.

Veamos un ejemplo. Supongamos que tenemos un servidor con 24 procesadores y otro con 8 procesadores. Claramente el servidor de 24 procesadores tiene mucha más capacidad que el servidor que tiene 8 procesadores. Nuestro balanceador de carga distribuirá la carga de nuestro trabajo en función de la capacidad que tengan nuestro servidores.

Se utilizan las **consultas distribuidas** (es otro nombre que recibe el balanceo de cargas). En el caso de PostgreSQL se llama “Multi-Server Parallel Query Execution”.

6. Ejemplo de uso de PostgreSQL

6.1. Instalación de PostgreSQL

Para la instalación de PostgreSQL crearemos dos máquinas virtuales como ya hemos hecho en las prácticas de esta misma asignatura fijándonos bien en el siguiente punto de la instalación.

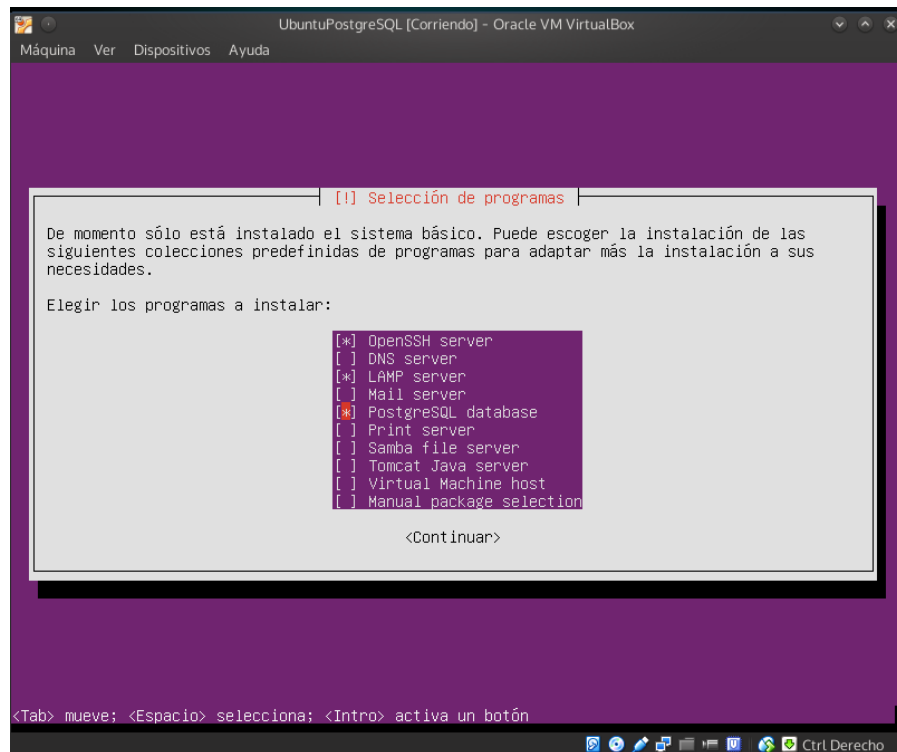


Figura 4: Instalando las máquinas virtuales

puesto que debemos seleccionar correctamente que queremos instalar la opción

PostgreSQL database

De manera que terminamos de instalar las dos máquinas virtuales para después pasar a configurarlas.

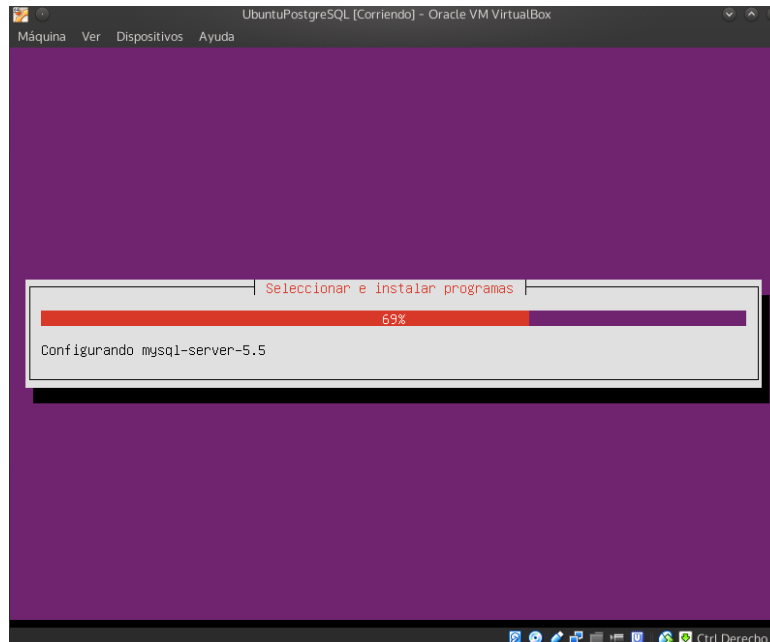


Figura 5: Instalando las máquinas virtuales

La configuración de estas dos máquinas virtuales es exactamente la misma que se realizó en las primeras prácticas, debemos configurar una dirección IP para cada una de ellas (en nuestro caso usaremos 192.168.1.105 para la MV maestro y 192.168.1.106 para la MV esclavo).

Además, debemos configurar la tarjeta de red de la máquina virtual desde las opciones de nuestro simulador, en este caso, VirtualBox.

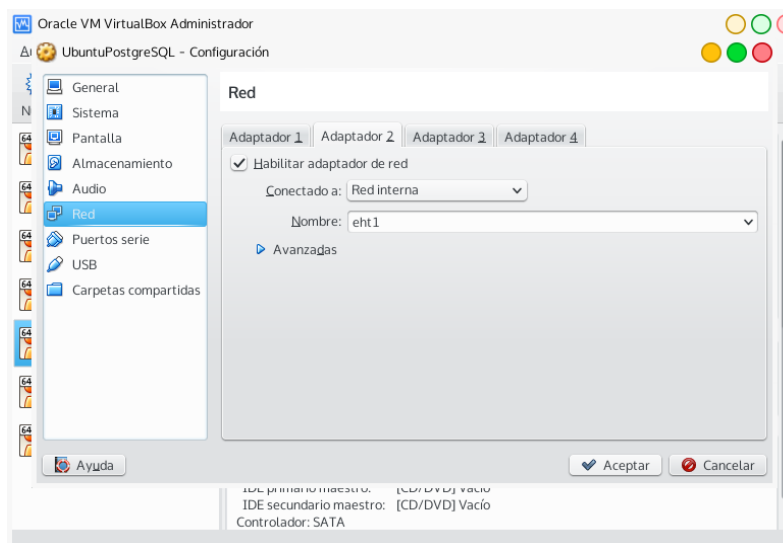
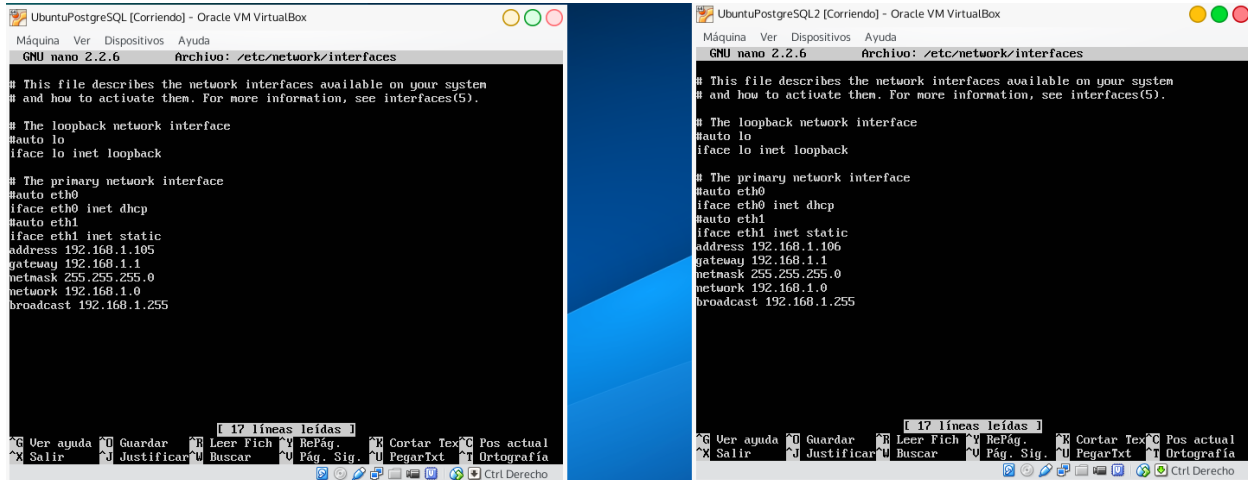


Figura 6: Configurando la red interna

De esta manera tendremos configuradas ambas máquinas virtuales y conectadas por una misma red. Por último, tendremos que configurar el archivo `/etc/network/interfaces` (de cada una de las máquinas virtuales) donde escribimos lo siguiente:



```
GNU nano 2.2.6 Archivo: /etc/network/interfaces

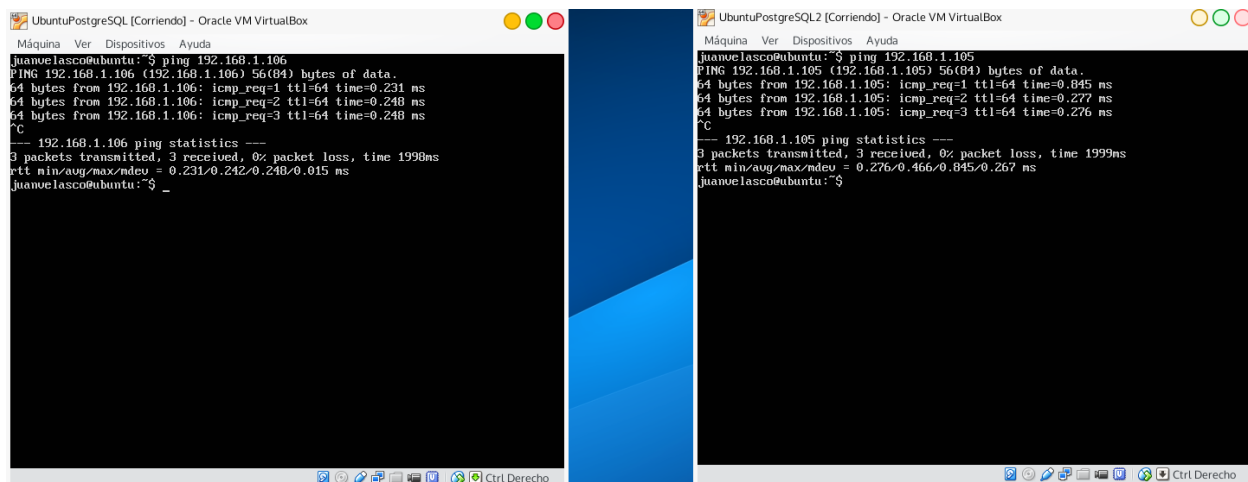
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
#auto lo
iface lo inet loopback

# The primary network interface
#auto eth0
iface eth0 inet dhcp
#auto eth1
iface eth1 inet static
address 192.168.1.105
gateway 192.168.1.1
netmask 255.255.255.0
network 192.168.1.0
broadcast 192.168.1.255
```

Figura 7: Configurando el archivo `/etc/network/interfaces`

Y finalmente vemos que se comunican correctamente entre ellas:



```
juanvelasco@ubuntu:~$ ping 192.168.1.106
PING 192.168.1.106 (192.168.1.106) 56(84) bytes of data:
64 bytes from 192.168.1.106: icmp_req=1 ttl=64 time=0.231 ms
64 bytes from 192.168.1.106: icmp_req=2 ttl=64 time=0.248 ms
64 bytes from 192.168.1.106: icmp_req=3 ttl=64 time=0.248 ms
^C
--- 192.168.1.106 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.231/0.242/0.248/0.015 ms
juanvelasco@ubuntu:~$ _
```

```
juanvelasco@ubuntu:~$ ping 192.168.1.105
PING 192.168.1.105 (192.168.1.105) 56(84) bytes of data:
64 bytes from 192.168.1.105: icmp_req=1 ttl=64 time=0.845 ms
64 bytes from 192.168.1.105: icmp_req=2 ttl=64 time=0.277 ms
64 bytes from 192.168.1.105: icmp_req=3 ttl=64 time=0.276 ms
^C
--- 192.168.1.105 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.276/0.466/0.845/0.267 ms
juanvelasco@ubuntu:~$
```

Figura 8: Conexión correcta entre ambas máquinas virtuales

6.2. Creación de la base de datos

6.2.1. Primeros comandos

Pasaremos a crear las tablas de nuestra base de datos. Para ello recordamos algunos de los comandos más usuales de PostgreSQL.

Entrar en la base de datos	<code>sudo su postgres -c psql</code>
Crear base de datos	<code>createdb contactos</code>
Conectar base de datos	<code>\c contactos</code>
Ver las bases de datos creadas	<code>\l</code>
Ver las tablas de la base de datos	<code>\d</code>
Describe tabla	<code>\d datos</code>

6.2.1. Creación de la base de datos y las tablas

Aquí se muestra la base de datos y un ejemplo de creación de tablas que hicimos nosotros:

```
UbuntuPostgreSQL [Corriendo] - Oracle VM VirtualBox
Máquina Ver Dispositivos Ayuda
Type "help" for help.

postgres=# \c contactos
You are now connected to database "contactos" as user "postgres".
contactos=# \d
      List of relations
Schema | Name  | Type  | Owner
-----+-----+-----+-----
public | datos | table | postgres
(1 row)

contactos=# \d+ datos
      Table "public.datos"
  Column  |      Type      | Modifiers | Storage | Description
-----+-----+-----+-----+-----
 nombre   | character(20)   |           | extended |
 telefono | integer         |           | plain   |
Has OIDs: no

contactos=# SELECT * FROM datos;
      nombre      | telefono
-----+-----
 Pepe             | 958233459
 Cristina         | 658765412
 Roberto          | 958753212
 Alberto          | 958789878
 Sofia            | 912678562
(5 rows)

contactos=#
```

Hemos creado un ejemplo de base de datos con la tabla “datos”. También hemos introducido una serie de ejemplos para que la tabla no estuviera vacía.

6.3. Configurando servidor maestro

Lo primero que haremos será crear un usuario en PostgreSQL. Para ello realizamos lo siguiente:

En nuestra máquina virtual 1 o maestro, **accedemos a PostgreSQL**.

```
sudo su postgres -c psql
```

Después creamos nuestro usuario, cuyo nombre de usuario y contraseña pueden ser lo que queramos. En nuestro caso será rep (*replicación*) y pass nuestra contraseña.

```
CREATE USER juanvelasco REPLICATION LOGIN CONNECTION LIMIT 1 ENCRYPTED  
PASSWORD 'pass';
```

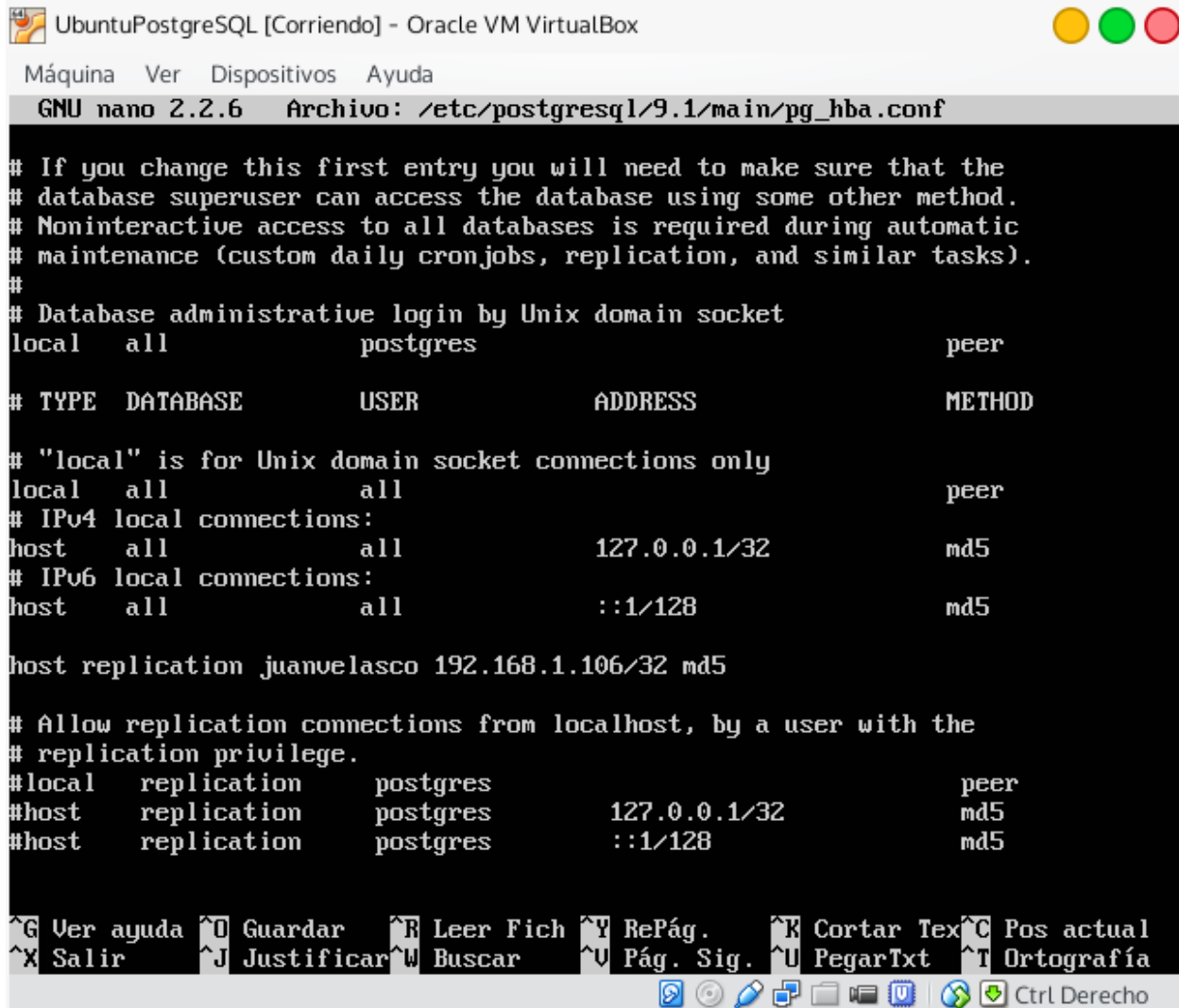
Lo siguiente que haremos será acceder al siguiente archivo de configuración:

```
sudo nano /etc/postgresql/9.1/main/pg_hba.conf
```

Donde escribiremos, en cualquier parte del archivo (salvo en la última línea), la siguiente línea que da acceso al nuevo usuario al servidor.

```
host replication juanvelasco Direccion_IP_Esclavo/32 md5
```

donde recordamos que **Direccion_IP_Esclavo** es la dirección IP que le habíamos asignado al servidor esclavo, en nuestro caso, 192.168.1.106.



```
GNU nano 2.2.6 Archivo: /etc/postgresql/9.1/main/pg_hba.conf

# If you change this first entry you will need to make sure that the
# database superuser can access the database using some other method.
# Noninteractive access to all databases is required during automatic
# maintenance (custom daily cronjobs, replication, and similar tasks).
#
# Database administrative login by Unix domain socket
local all postgres peer

# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all peer
# IPv4 local connections:
host all all 127.0.0.1/32 md5
# IPv6 local connections:
host all all ::1/128 md5

host replication juanvelasco 192.168.1.106/32 md5

# Allow replication connections from localhost, by a user with the
# replication privilege.
#local replication postgres peer
#host replication postgres 127.0.0.1/32 md5
#host replication postgres ::1/128 md5

^G Ver ayuda ^O Guardar ^R Leer Fich ^Y RePág. ^K Cortar Tex ^C Pos actual
^X Salir ^J Justificar ^W Buscar ^U Pág. Sig. ^U PegarTxt ^T Ortografía
Ctrl Derecho
```

Guardamos y cerramos el archivo.

El siguiente paso es configurar otro archivo, ahora escribiremos:

```
sudo nano /etc/postgresql/9.1/main/postgresql.conf
```


En este archivo debemos buscar los siguientes parámetros y descomentarlos (si es que están comentados) o modificar sus valores.

```
listen_addresses = 'localhost,Direccion_IP_Maestro'  
wal_level = 'hot_standby'  
archive_mode = on  
archive_command = 'cd .'  
max_wal_senders = 1  
hot_standby = on
```

Y reiniciamos el servicio de PostgreSQL en la servidor maestro.

```
sudo service postgresql restart
```

Una vez hecho esto tendremos nuestra servidor maestro configurada.

6.4. Configurando servidor esclavo

Empezaremos parando el servidor de postgres en el servidor esclavo.

```
sudo service postgresql stop
```

Lo siguiente que haremos será acceder al siguiente archivo de configuración:

```
sudo nano /etc/postgresql/9.1/main/pg_hba.conf
```

Donde escribiremos, en cualquier parte del archivo (salvo en la última línea), la siguiente línea que da acceso al nuevo usuario al servidor.

```
host replication juanvelasco Direccion_IP_Maestro/32 md5
```

donde recordamos que **Direccion_IP_Maestro** es la dirección IP que le habíamos asignado al servidor maestro, en nuestro caso, 192.168.1.105.

Guardamos y cerramos el archivo.

El siguiente paso es configurar otro archivo, ahora escribiremos:

```
sudo nano /etc/postgresql/9.1/main/postgresql.conf
```

En este archivo debemos buscar los siguientes parámetros y descomentarlos (si es que están comentados) o modificar sus valores.

```
listen_addresses = 'localhost,Direccion_IP_Esclavo'  
wal_level = 'hot_standby'  
archive_mode = on  
archive_command = 'cd .'  
max_wal_senders = 1  
hot_standby = on
```

Guardamos y cerramos el archivo.

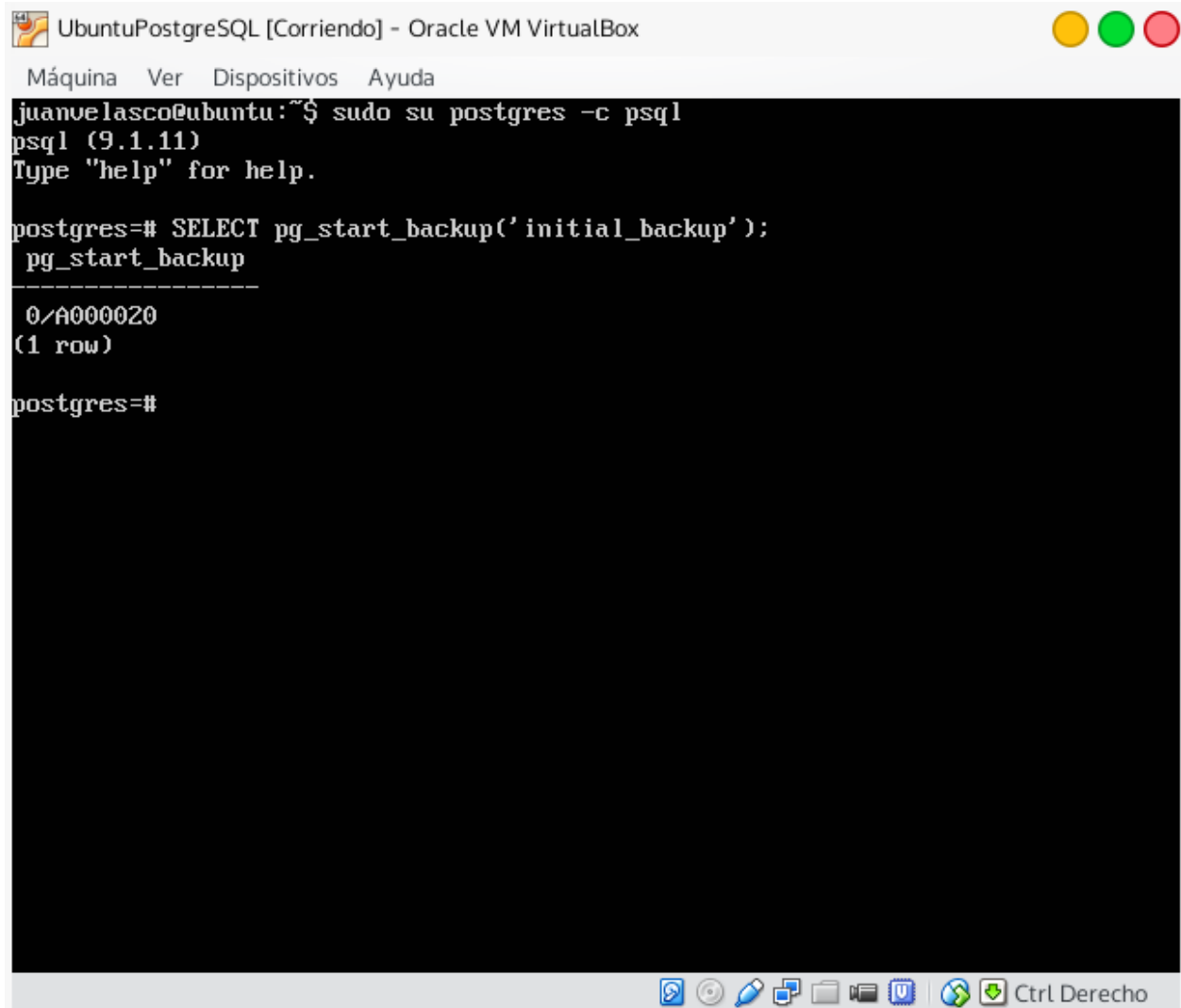
6.5. Replicando la base de datos inicial

Antes de que el esclavo pueda replicar en el maestro, tenemos que darle una base de datos inicial. Desde el servidor maestro, usaremos un comando interno de postgres que hace *backup*. Para ello escribiremos:

```
sudo su postgres -c psql
```

Una vez conectados a PostgreSQL escribimos en la terminal

```
SELECT pg_start_backup('initial_backup');
```

A screenshot of a terminal window titled "UbuntuPostgreSQL [Corriendo] - Oracle VM VirtualBox". The window has a menu bar with "Máquina", "Ver", "Dispositivos", and "Ayuda". The terminal shows a user "juanvelasco@ubuntu" running "sudo su postgres -c psql". The prompt changes to "psql (9.1.11)" and "Type 'help' for help.". The user enters "SELECT pg_start_backup('initial_backup');". The output shows "pg_start_backup" followed by a separator line and "0/A000020". Below that, it says "(1 row)". The prompt returns to "postgres=#". The bottom of the window shows a taskbar with various icons and the text "Ctrl Derecho".

```
juanvelasco@ubuntu:~$ sudo su postgres -c psql
psql (9.1.11)
Type "help" for help.

postgres=# SELECT pg_start_backup('initial_backup');
 pg_start_backup 
-----
 0/A000020
(1 row)

postgres=#
```

Otra opción de realizar esto sin entrar y salir de postgres es conectándose a la cuenta de nuestro usuario y escribir en la terminal

```
psql -c "SELECT pg_start_backup('initial_backup');"
```

Salimos de PostgreSQL con el comando `\q` y escribimos

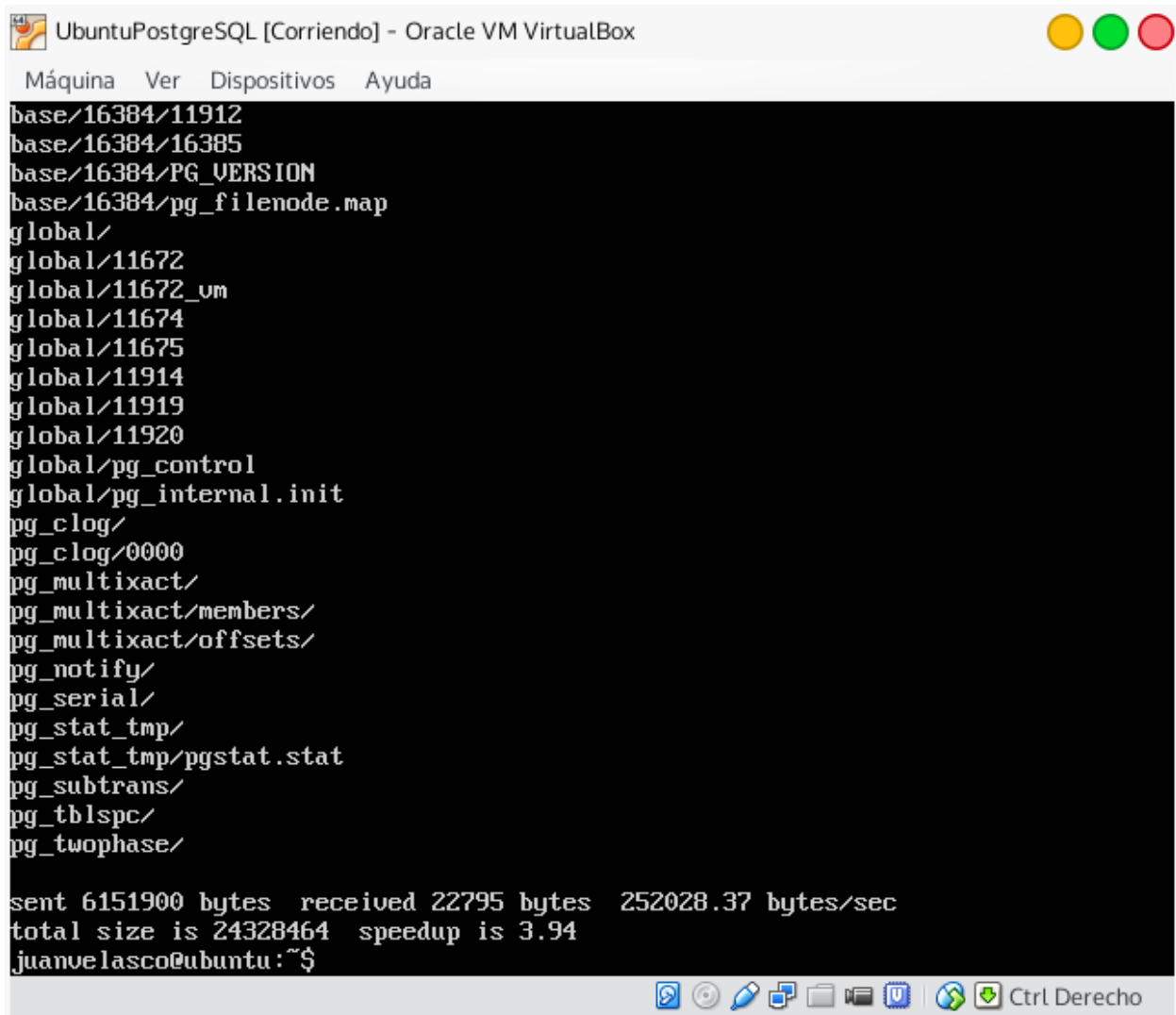
```
rsync -cva --inplace --exclude=*pg_xlog* /var/lib/postgresql/9.1/main/  
Direccion_IP_Esclavo:/var/lib/postgresql/9.1/main/
```

Puede ser que la salida de rsync nos de algunos fallos a la hora de modificar ciertos archivos, pero no pasa nada. De nuevo repetimos el proceso de conectarnos a PostgreSQL y escribimos

```
SELECT pg_stop_backup();
```

De igual forma haríamos

```
psql -c "SELECT pg_stop_backup():"
```



```
base/16384/11912
base/16384/16385
base/16384/PG_VERSION
base/16384/pg_filenode.map
global/
global/11672
global/11672_um
global/11674
global/11675
global/11914
global/11919
global/11920
global/pg_control
global/pg_internal.init
pg_clog/
pg_clog/0000
pg_multixact/
pg_multixact/members/
pg_multixact/offsets/
pg_notify/
pg_serial/
pg_stat_tmp/
pg_stat_tmp/pgstat.stat
pg_subtrans/
pg_tblspc/
pg_twophase/

sent 6151900 bytes received 22795 bytes 252028.37 bytes/sec
total size is 24328464 speedup is 3.94
juanvelasco@ubuntu:~$
```

Resultado de ejecutar rsync

Por último deberemos crear un archivo de recovery o de recuperación. Para ello escribimos:

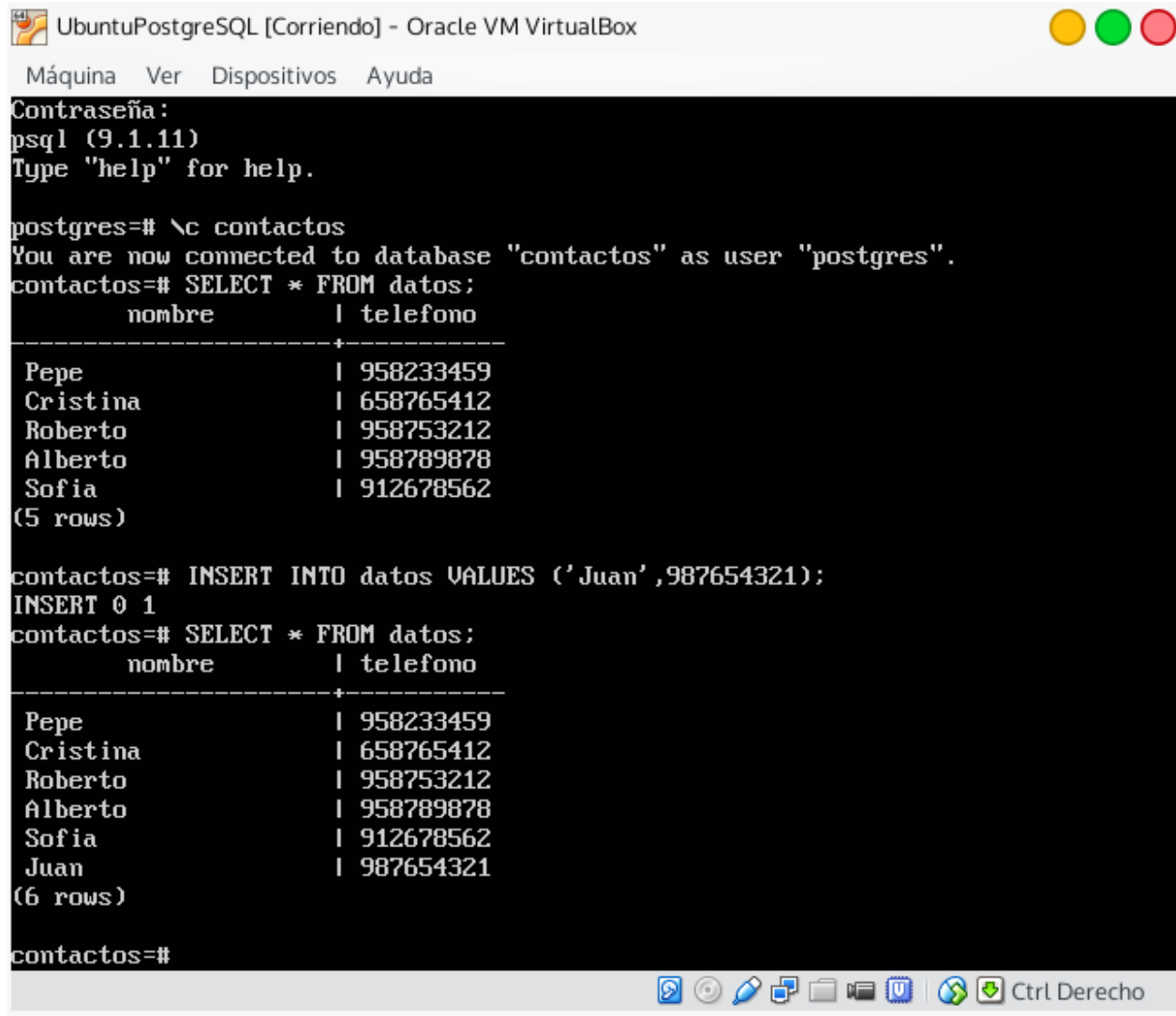
```
sudo nano /var/lib/postgresql/9.1/main/recovery.conf
```

Y escribimos dentro lo siguiente

```
standby_mode='on'  
primary_conninfo='host=192.168.1.106 port=5432 user=juanvelasco  
password=tucontraseña'  
trigger_file='/tmp/postgresql.trigger.5432'
```

El script detiene el servidor esclavo, borra el antiguo directorio cluster del esclavo, ejecuta el comando `pg_basebackup` para conectar al maestro y copiar las bases de datos, creará también un archivo `recovery.conf` que nos será muy útil si más adelante tenemos problemas y, por último, inicia de nuevo el servidor esclavo.

Llegados a este punto habríamos acabado de configurar la relación Maestro-Esclavo. Podemos probar a escribir nuevos datos en la base de datos del maestro y deberían verse reflejados en la base de datos del esclavo. En nuestro caso hemos añadido un dato nuevo a nuestra tabla.



```
UbuntuPostgreSQL [Corriendo] - Oracle VM VirtualBox
Máquina Ver Dispositivos Ayuda
Contraseña:
psql (9.1.11)
Type "help" for help.

postgres=# \c contactos
You are now connected to database "contactos" as user "postgres".
contactos=# SELECT * FROM datos;
      nombre      | telefono
-----+-----
Pepe              | 958233459
Cristina          | 658765412
Roberto           | 958753212
Alberto           | 958789878
Sofia             | 912678562
(5 rows)

contactos=# INSERT INTO datos VALUES ('Juan',987654321);
INSERT 0 1
contactos=# SELECT * FROM datos;
      nombre      | telefono
-----+-----
Pepe              | 958233459
Cristina          | 658765412
Roberto           | 958753212
Alberto           | 958789878
Sofia             | 912678562
Juan              | 987654321
(6 rows)

contactos=#
```

```
INSERT INTO datos VALUES ("Juan",987654321);
```

Podemos conocer el estado de la replicación escribiendo lo siguiente:

Iniciamos postgres en nuestro servidor esclavo

```
sudo su postgres -c psql
```

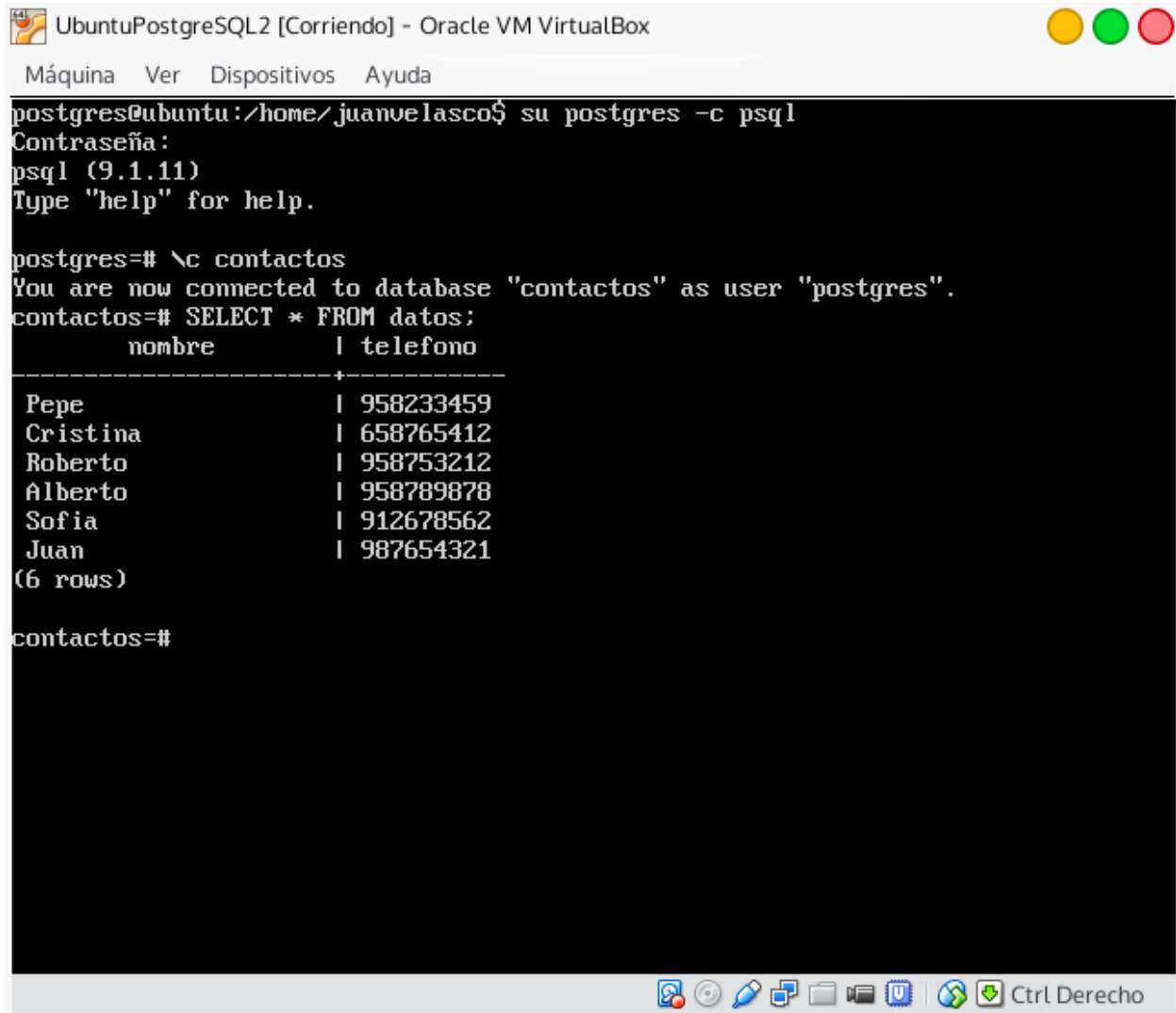
Nos conectamos a la base de datos

```
\c contactos
```

Escribimos la siguiente consulta en nuestra base de datos

```
SELECT * FROM datos;
```

Obteniendo así en nuestro servidor esclavo los cambios del servidor maestro



```
postgres@ubuntu:/home/juanvelasco$ su postgres -c psql
Contraseña:
psql (9.1.11)
Type "help" for help.

postgres=# \c contactos
You are now connected to database "contactos" as user "postgres".
contactos=# SELECT * FROM datos;
      nombre      | telefono
-----+-----
      Pepe       | 958233459
     Cristina    | 658765412
     Roberto     | 958753212
     Alberto     | 958789878
     Sofia       | 912678562
     Juan        | 987654321
(6 rows)

contactos=#
```

7. Conclusiones

Finalmente y después de corregir algunos errores comunes del programa PostgreSQL fuimos capaces de replicar una base de datos entre un servidor maestro y un servidor esclavo.

La información que fuimos encontrando en internet resultó interesante cuando se estudiaban varias versiones de diferentes personas. Resultó ser un trabajo bastante entretenido y, en nuestra opinión, de gran utilidad en un futuro.

Esperamos que este breve tutorial les sirva a ustedes también si necesitan hacer una replicación de bases de datos con el lenguaje PostgreSQL.

Un saludo, los autores del trabajo:

Juan Antonio Velasco Gómez
Miguel Sánchez Maldonado