Capítulo 6

Sistema de gestión de base de datos postgreSQL

Primeros pasos

Funcionamiento

Características

Instalación y configuración

Ejemplos

Capítulo 6 El sistema de gestión de bases de datos postgreSQL

En los últimos años, el software de bases de datos ha experimentado un gran auge debido sobre todo a la necesidad, por parte de las empresas, de ordenar y procesar las grandes cantidades de información que manejan en su interior.

6.1 Introducción a postgreSQL

El sistema de gestión de bases de datos relacionales orientadas a objetos conocido como PostgreSQL, está derivado del paquete Postgres escrito en Berkeley allá por los años 80. Con cerca de una década de desarrollo tras él, PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy en día, ofreciendo control de concurrencia multi-versión y soportando casi toda la sintaxis SQL (incluyendo subconsultas, transacciones, y tipos y funciones definidas por el usuario). Además cuenta también con un amplio conjunto de APIs para la mayoría de lenguajes de programación (incluyendo los más usados, tales como C, C++, Java, PHP, perl, tcl y python).

6.1.1 Un poco de historia

La implementación del DBMS Postgres comenzó en 1986. Los conceptos iniciales para el sistema fueron presentados en The Design of Postgres y la definición del modelo de datos inicial apareció en The Postgres Data Model. El diseño del sistema de reglas fue descrito en ese momento en The Design of the Postgres Rules System. La lógica y arquitectura del gestor de almacenamiento fueron detalladas en The Postgres Storage System.

Postgres ha pasado por varias revisiones importantes desde entonces. Desde la versión 1, que se lanzó para unos pocos usuarios, hasta la 3, se mejoró el ejecutor de consultas y se añadió el soporte para múltiples gestores de almacenamiento.

En su mayor parte, las siguientes versiones hasta el lanzamiento de Postgres95 se centraron en mejorar la portabilidad y la fiabilidad.

El tamaño de la comunidad de usuarios externos casi se duplicó durante 1993. Pronto se hizo obvio que el mantenimiento del código y las tareas de soporte estaban ocupando tiempo que debía dedicarse a la investigación. En

un esfuerzo por reducir esta carga, el proyecto terminó oficialmente con la Versión 4.2.

En 1994, Andrew Yu y Jolly Chen añadieron un intérprete de lenguaje SQL a Postgres. Postgres95 fue publicado a continuación en la Web para que encontrara su propio lugar en el mundo como un descendiente de dominio público y código abierto del código original Postgres de Berkeley.

El código de Postgres95 fue adaptado a ANSI C y su tamaño reducido en un 25%. Muchos cambios internos mejoraron el rendimiento y la facilidad de mantenimiento. Postgres95 v1.0.x se ejecutaba en torno a un 30-50% más rápido en el Wisconsin Benchmark comparado con Postgres v4.2.

En 1996, se hizo evidente que el nombre "Postgres95" no resistiría el paso del tiempo, entonces se eligió un nuevo nombre, PostgreSQL, para reflejar la relación entre el Postgres original y las versiones más recientes con capacidades SQL. Al mismo tiempo, se hizo que los números de versión partieran de la 6.0, volviendo a la secuencia seguida originalmente por el proyecto Postgres.

Durante el desarrollo de Postgres95 se hizo hincapié en identificar y entender los problemas en el código del motor de datos. Con PostgreSQL, el énfasis ha recaído en un aumento de características y capacidades, aunque el trabajo continúa en todas las áreas.

6.2 Funcionamiento

Antes de proceder a la instalación y configuración debemos entender la arquitectura básica de postgres.

En la jerga relacionada con las bases de datos, postgres usa un modelo cliente - servidor. Una sesión de postgres se basa en dos procesos, que mostramos a continuación.

- un proceso servidor, que gestiona los archivos que forman la base de datos, acepta las conexiones que efectúan los clientes con ella y que ejecuta acciones a petición de los clientes.

- la aplicación cliente del usuario (también conocido como frontend) que pretende ejecutar ciertas acciones sobre la base de datos. Estas aplicaciones pueden ser de naturaleza muy diversa, desde herramientas modo texto o aplicaciones gráficas, hasta servidores que acceden para recuperar datos y mostrarlos en una página web. O incluso una aplicación específica para el mantenimiento de la base de datos.

Tal como ocurre en las aplicaciones cliente - servidor, el cliente puede estar en un equipo diferente al que se encuentra el servidor. En ese caso, ambos se comunican mediante una conexión de red TCP/IP. Es necesario tener esto en cuenta, ya que archivos que son accesibles desde un cliente no tienen porqué serlo desde otro. Se debe prestar especial atención a estos temas.

El servidor de postgres puede manejar diferentes conexiones concurrentes de los clientes. Dispone de un proceso padre (conocido como postmaster) que crea distintos procesos hijos que atienden las conexiones entrantes. Un planteamiento típico del modelo cliente - servidor.

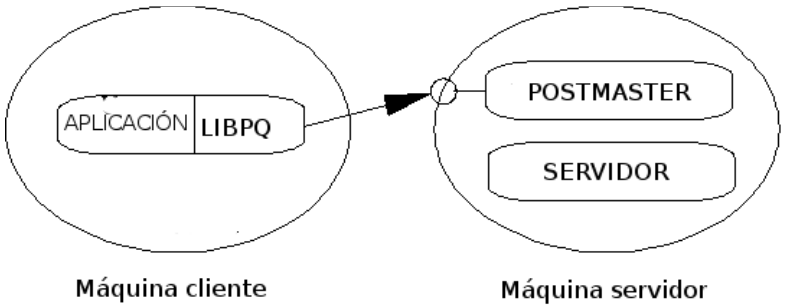

Figura 6.1 Modelo cliente – servidor en postgres

6.2.1 SQL

Postgres es un sistema de gestión de bases de datos relacional (RDBMS). Esto significa que es un sistema para gestionar datos que se encuentran en forma de relaciones. Una relación puede verse como el término matemático para designar a una tabla. La noción de guardar datos en tablas no es algo nuevo hoy en día, y de hecho es muy usual, aunque también existen otras formas, como las bases de datos jerarquicas o las orientadas a objetos.

Cada tabla es un conjunto de filas. Cada fila de una tabla dada tiene el mismo número de columnas, y cada columna pertenece a un tipo de dato específico.

Las tablas se agrupan en bases de datos. Una colección de bases de datos gestionadas por un mismo servidor de postgres constituye un cluster de base de datos.

Debido a la diversidad de lenguajes y de bases de datos existentes, la manera de comunicar unos y otros sería realmente complicada a gestionar de no ser por la existencia de estándares que nos permiten el realizar las operaciones básicas de una forma universal.

SQL no es mas que un lenguaje estándar de comunicación con bases de datos. Hablamos por tanto de un lenguaje normalizado que nos permite

trabajar con cualquier tipo de lenguaje en combinación con cualquier tipo de base de datos.

El hecho de que sea estándar no quiere decir que sea idéntico para cada base de datos. En efecto, determinadas bases de datos implementan funciones específicas que no tienen necesariamente que funcionar en otras.

Aparte de esta universalidad, SQL posee otras dos características muy apreciadas. Por una parte, presenta una potencia y versatilidad notables que contrasta, por otra, con su accesibilidad de aprendizaje.

6.3 Características de postgres

Postgres dispone prácticamente de todas las funciones que podemos encontrar en un sistema de gestión comercial. De hecho, a nivel de empresa, es el sistema de fuentes abiertas más usado.

A nivel medio, en casos en los que no hace falta una base de datos de grandes prestaciones, o se busca principalmente la simplicidad, un sistema destaca sobre el resto: mysql.

Mysql es el servidor de bases de datos más usado del mundo. La simplicidad es su principal ventaja, pero también su mayor inconveniente. Mediante excelentes herramientas, como es phpMyAdmin, podemos crear prácticamente todo lo que se desee, con una gran facilidad.

Los problemas de mysql comienzan cuando pretendemos complicar la situación. Mayores tablas y peticiones más complejas hacen que mysql "se atasque" y funcione de forma mucho más lenta. Sus últimas versiones solucionan en parte estas carencias a costa de un mayor consumo de recursos.

Hemos nombrado mysql ya que vamos a presentar las características principales de postgres comparando ambos sistemas. A continuación mostramos una tabla, donde podemos comprobar como postgres dispone de todas las características que hemos señalado como deseables, en cualquier sistema de gestión de base de datos.

Característica	***MySQL 3.x***	***MySQL 4.0.x***	***MySQL 4.1.x***	***postgreSQL***
Subselecciones	No	Algunas	Si	Si
Vistas	No	No	No	Si
Relaciones clave externa	No	Si	Si	Si
Restricciones clave externa	No	No	No	Si
Triggers	No	No	No	Si
Índices de tipos no triviales	No	No	No	Si
Secuencias	Algunas	Algunas	Algunas	Si
Transacciones	No	Si	Si	Si
Herencia de tablas (OO)	No	No	No	Si
Notificaciones asíncronas	No	No	No	Si
Restricciones	No	No	No	Si
SELECT INTO	No	Si	Si	Si
Stored procedures	No	No	No	Si
Bloqueo a nivel de fila	Si	Si	Si	Si
Bloqueo a nivel de tabla	Si	Si	Si	Si
Control de concurrencia	No	No	No	Si

Comparativa entre MySQL y posgres

Por último vamos a comentar algunos aspectos más generales en el acceso a bases de datos.

- Velocidad. En la mayor parte de situaciones postgres trabaja prácticamente a la misma velocidad que mysql. Esto ocurre respecto a las nuevas versiones de mysql. En las anteriores versiones, mysql era mucho más rápida, disponía de en torno a 2 o 3 veces la velocidad de postgres.

- Estabilidad. En versiones anteriores a las actuales, mysql era más estable que postgres. Actualmente ambas han trabajado mucho este concepto y disfrutan de una muy buena estabilidad.

- Integridad de los datos. Este es uno de los puntos en el que postgres sobresale. Asegura que los datos son correctos mediante la imposición de restricciones. Mysql no dispone de estas restricciones. Además en las versiones 3.x (que son las más usadas) ni siquiera tiene soporte para transacciones ni rollback. Es un hecho comprobado que cuando el disco se encuentra lleno, postgres puede abordar la situación sin afectar a la base de datos. Sin embargo mysql puede corromper tablas (y suele suceder), con la consecuente pérdida de información.

- Características especiales del lado de servidor. Postgres dispone de reglas, triggers, funciones del lado de servidor que pueden ser escritas en lenguajes tales como C, pyton, perl, pgsql o tcl. En temas referentes a desarrollo web, existe un lenguaje que sobresale hoy en día: PHP. El soporte que tiene PHP para mysql es extraordinario; para postgres es simplemente bueno.

- Seguridad. En este apartado prácticamente no existen diferencias. Quizás podamos señalar que la seguridad se encuentra más granulada en mysql. Esto quiere decir que podemos no sólo dar derechos de acceso por usuario o por equipo de red, sino también por nombre de tabla. Por otro lado postgres puede limitar logins basados en diferentes criterios, red de acceso...

- Soporte para bloqueo y concurrencia. Postgres dispone de un mecanismo conocido como MVCC, comparable o incluso superior a las mejores bases de datos comerciales. Permite bloquear a nivel de fila: puede bloquear una fila para escritura en una sesión y en otra puede mostrar las filas inalteradas. Mysql sólo puede realizar bloqueo para una tabla, tanto para lectura como para escritura.

- Objetos muy grandes. En mysql no hay prácticamente distinción entre estos datos y otros, aunque existen algunas limitaciones. Sin embargo en

postgres es algo más difícil y existe un tratamiento específico para estos casos.

- Soporte de lenguajes. Respecto a este tema postgres puede guardar la configuración local por cada cliente, no por cada base de datos.

- Conformidad con el estándar SQL. Postgres entiende un buen subconjunto de SQL 92 y además añaden funciones de orientación de objetos a este subconjunto. Mysql entiende sólo un pequeño conjunto faltándole ciertas funciones importantes, como pueden ser las subselecciones.

6.4 Autenticación del cliente

Cuando una aplicación cliente se conecta al servidor de base de datos, especifica qué nombre de usuario de postgres quiere usar para conectarse. Esto ocurre de la misma forma que un usuario accede a una máquina Linux. Dentro del entorno SQL el nombre de usuario de la base de datos determina los privilegios de acceso a sus objetos.

Mediante la autenticación el servidor de base de datos establece la identidad del cliente y con ello conoce si se le permite conectar con el nombre de usuario utilizado.

Postgres ofrece diferentes métodos de autenticación. El método usado para autenticar una conexión cliente particular puede ser seleccionado partiendo de la dirección de máquina, base de datos y usuario.

Los nombres de usuario de postgres están separados de forma lógica de los usuarios del sistema operativo donde se ejecuta. Si todos los usuarios de un servidor particular tienen cuenta en la máquina donde se aloja el servidor, parece lógico que usen esos mismos nombres. Sin embargo, si tenemos un servidor que acepta conexiones remotas, puede que nos encontremos con muchos usuarios que no dispongan de cuenta en el sistema operativo local. Así pues en este caso no existiría relación entre ambos nombres.

La autenticación en postgres puede adaptarse a cualquiera de estas situaciones. La configuración la veremos en los apartados siguientes.

6.4.1 El fichero pg_hba.conf

Toda la autenticación de cliente se controla mediante el fichero pg_hba.conf, que se encuentra en el directorio data (normalmente se encontrará en /usr/local/pgsql/data/pg_hba.conf).

El formato de este archivo comprende una serie de relaciones o entradas, una por cada línea. Cada relación está compuesta por una serie de campos, separados por caracteres en blanco o tabuladores, algo típico de los ficheros de configuración.

Cada relación especifica un tipo de conexión, un rango de direcciones IP (si es necesario), un nombre de base de datos, un nombre de usuario y el método de autenticación que seguirá todas la conexiones que correspondan con esos parámetros.

La primera relación en la que coincidan el tipo de conexión, dirección de cliente, base de datos deseada y nombre de usuario es la que se utiliza para efectuar la autenticación. Si se toma una relación y la autenticación falla, el resto de relaciones no se consideran. Si ninguna relación coincide con la que la conexión actual, el acceso se deniega.

Una relación puede tener uno de estos siete formatos:

```
local      base de datos    usuario   método de autenticación    [opciones]
host       base de datos    usuario   IP   máscara  método       [opciones]
hostssl    base de datos    usuario   IP   máscara  método       [opciones]
hostnossl  base de datos    usuario   IP   máscara  método       [opciones]
host       base de datos    usuario   IP/longitudmáscara  método [opciones]
hostssl    base de datos    usuario   IP/longitudmáscara  método [opciones]
hostnossl  base de datos    usuario   IP/longitudmáscara  método [opciones]
```

El significado de los campos los mostramos a continuación.

- local. Esta entrada se ajustaría a aquellos intentos de conexiones que hagan uso de los sockets de Unix. Sin una entrada de este tipo, estas conexiones no están permitidas.

- host. Esta entrada se ajustaría a aquellos intentos de conexiones que hagan uso de redes TCP/IP. Debemos tener en cuenta que como medida de seguridad, las conexiones de red están deshabilitadas, a menos que el servidor se ejecute con la opción -i, o con el parámetro de configuración tcpip_socket activado.

- hostssl. Esta entrada se refiere a aquellos intentos de conexión que usen SSL sobre TCP/IP. Las entradas de host coincidirán tanto para intentos que sean SSL como aquellos que no, pero hostssl requiere que sean conexiones SSL. Para hacer uso de esta opción debemos tener el soporte ssl activado.

- hostnossl. Esta entrada es similar a la anterior, pero de forma negativa. Se toma sólo cuando el intento de conexión no está usando SSL.

- base de datos. Este campo indica el nombre de la base de datos a la que se refiere esta entrada. Existen unos valores especiales que puede tomar este campo y que citamos a continuación. Podemos indicar el valor “all“ para referirnos a todas las bases de datos. Para indicar que puede acceder el usuario cuyo nombre es el mismo que la base de datos, usamos el valor “sameuser”.

Para indicar que puede acceder a una base de datos un grupo de usuarios cuyo nombre de grupo es el mismo que el nombre de la base de datos, podemos usar “samegroup”.

- usuario. Especifica a que usuarios va dirigida esta entrada. Podemos indicar todos los usuarios mediante el valor “all”. Debemos remarcar que estos usuarios son de postgres, no del sistema operativo.

- IP y máscara. Estos dos campos contienen la dirección ip y la máscara en notación decimal estándar (separados por puntos). No se pueden usar dominios ni nombres de máquinas. Si se toman juntos especifican la dirección (o direcciones) de la máquina cliente a la que hace referencia la entrada en la que se use.

- longitudmáscara. Este campo, Métodos de autenticación, representa una alternativa a la máscara. Debemos indicar un número entre 0 y 32, que se corresponden a los bits a 1 de la máscara. Esta notación es muy usual, de hecho, ya la hemos visto en el capítulo de Squid.

- método (de autenticación). Este campo especifica el método que se va a usar cuando una conexión use la entrada donde se encuentre. Los valores que puede tomar son: trust, password, kerberos, ident y PAM. En este punto se va a centrar nuestra atención de forma importante. Por ello vamos a verlos detenidamente en el siguiente apartado.

6.4.2 Métodos de autenticación

Los posibles métodos de autenticación son los siguientes.

6.4.2.1 Trust

Este método consiste en permitir la conexión incondicionalmente. Permite que cualquiera pueda conectarse al servidor de bases de datos de postgres y acceda con el usuario que desee, sin la necesidad de ninguna contraseña.

Este método debería ser usado sólo cuando disponemos de un nivel de protección adecuado, a nivel de sistema, en las conexiones con el servidor.

La autenticación mediante trust es apropiada y muy conveniente para conexiones locales en una estación de trabajo de un solo usuario. En casos de más usuarios o conexiones TCP/IP su uso sólo estaría justificado si conociésemos a todas las máquinas y usuarios conectadas al servidor.

Existe también la opción de denegar incondicionalmente. Para ello, en lugar de utilizar trust usaremos reject.

6.4.2.2 Password

Los métodos de autenticación basados en password son md5, crypt y password. Estos métodos operan de manera similar salvo en la forma de enviar el password por la conexión.

El método recomendado es md5 pues de esta forma lo único que circularía por la red será el hash de la contraseña y no ella misma, y que por tanto, previene ataques de sniffers. De la misma forma, crypt sería la siguiente opción, si queremos compatibilidad con clientes de versiones inferiores a 7.2, que no tenían la opción de md5.

El peor método sería password pues se envía la clave en texto plano y por tanto no debería usarse, a menos que tuviéramos ssl por debajo, y por tanto estuviese de esa forma protegido.

Las contraseñas para las base de datos de postgres están separadas de las de los usuarios del sistema operativo. La contraseña para cada usuario de la base de datos se almacena en la tabla pg_shadow. Estas claves se pueden gestionar mediante los comandos SQL, CREATE USER y ALTER USER.

6.4.2.3 Kerberos

Kerberos es un sistema de seguridad apropiado para computación distribuida sobre una red pública.

Postgres soporta tanto Kerberos 4 como Kerberos 5, aunque sólo se recomienda el último. Kerberos 4 se considera inseguro y no se recomienda para uso general.

6.4.2.4 Ident

El método de autenticación ident funciona tomando los nombre de usuario del sistema operativo cliente y determinando así a qué bases de datos tiene permitido acceder. Para ello dispone de un mapa, que no es más que un fichero que contiene los pares usuario – bases de datos permitidas.

Se basa en el protocolo de identificación (identification protocol) descrito en la RCF 1413. Virtualmente cada sistema operativo trabaja con un servidor ident que escucha peticiones TCP por el puerto 113 por defecto. La funcionalidad básica de este servidor es la de responder cuestiones como: ¿qué usuario ha iniciado la conexión en tu puerto X y que se conecta a mi puerto Y? Como postgres conoce ambos puertos cuando se produce una conexión, puede interrogar al servidor ident del equipo que ha hecho la conexión y, teóricamente, determinar el usuario de sistema operativo para cualquier conexión.

El problema de este procedimiento es que depende mucho de la integridad de los clientes. Si un cliente está comprometido, un atacante puede ejecutar un programa por el puerto 113 y devolver cualquier nombre de usuario que escoja. Por tanto, este método requiere que confiemos en todas las máquinas que ejecuten el servidor ident.

6.4.2.5 Pam

Este método opera de forma similar a password. Lo único que difiere es que hace uso de los módulos PAM para conseguir la autenticación.

Esta es la opción por la que vamos a decantarnos para realizar la integración, puesto que disponemos del módulo pam_ldap para autenticarnos contra el servidor OpenLDAP. Por defecto, los módulos PAM usados deben incluirse en el fichero /etc/pam.d/postgresql.

Esta opción puede ser cambiada indicando el nuevo fichero tras la palabra pam en el fichero de configuración pg_hba.conf.

6.5 Instalación y configuración

6.5.1 Obtener el software

El primer paso es obtener el software de la página oficial www.postgresql.org. Actualmente la versión disponible es la 8.0 aunque cuando comenzamos el proyecto la última era la 7.4. Por ello vamos a referirnos siempre a ésta ya que fue la que nos sirvió para ejecutar la fase de pruebas.

Respecto a la versión 8.0 no hay ningún cambio en lo que autenticación se refiere así que todo lo que vamos a hacer es válido para ella.

6.5.2 Requerimientos mínimos

Hardware

Básicamente vamos a tener dos necesidades básicas. En primer lugar debemos tener en cuenta la cantidad de espacio que va a necesitar las bases de datos. Una base de datos vacía consume 25 MB de disco duro. Conforme se introducen datos, el espacio ocupado se incrementa del orden de unas cinco veces el espacio que ocuparían esos mismos datos, si estuviesen en texto plano.

También hace falta espacio para la compilación e instalación de los binarios. En total, unos 65 MB para la compilación y unos 15 MB para albergar a los binarios.

La otra necesidad básica del sistema es la cantidad de memoria RAM que usa. Al igual que ocurre con la mayoría del software existente cuanta más memoria dispongamos, menos debe acceder al disco duro y por tanto, más se aceleran los procesos. En este caso nos permite, entre otras cosas,

mantener un número mayor de conexiones simultáneas con la base de datos.

Uno de los parámetros básicos que intervienen en el rendimiento de postgres son los búferes compartidos. Un búfer compartido es un bloque de memoria que usa para mantener las peticiones que todavía no ha podido pasar por el búfer del kernel ni por la CPU. Este parámetro conserva, por defecto, un valor demasiado pequeño. Debe ser cambiado teniendo en cuenta la cantidad de RAM que va a tener disponible postgres, de tal forma que a mayor tamaño de bufer, mayor rendimiento. Esto ocurre de forma general. Un excesivo valor en este parámetro puede tener un efecto contrario, "asfixiando" de memoria al resto de aplicaciones que se ejecuten en la misma máquina.

La cantidad de memoria RAM necesaria varía según el número de usuarios y el uso que den a las bases de datos. En una situación general no deberíamos tener problemas con 512 MB.

En cuanto al resto de parámetros de hardware no se exige demasiado, aunque como ya hemos comentado, cuanto mejor sea más carga va a soportar el servidor y podrá por tanto dar servicio a más usuarios. Con un procesador cercano al Ghz de velocidad podemos trabajar de forma cómoda. Cualquier procesador actual sobrepasa estas características, así que no debemos tener problemas.

Software

En general cualquier plataforma Linux moderna debería poder ejecutar postgres. A continuación ofrecemos una lista, a modo de ejemplo, de algunas plataformas donde se ha probado satisfactoriamente. Entre ellas están las más importantes, Red Hat, Debian, Solaris...

Sistema Operativo	***Procesador***	***Versión***
AIX	PowerPC, RS6000	8.0.0
BSD/OS	x86	8.0.0
Debian	Alpha, AMD64, ARM, IA64, m68k, MIPS, PA-RISC, PowerPC, S/390, Sparc, x86	8.0.0 (excepto Alpha, IA64 y S390, que sólo aceptan 7.4)
Fedora	AMD64, x86	8.0.0
FreeBSD	Alpha, x86	7.4 y 8.0.0 respectivamente
Gentoo Linux	AMD64, x86	8.0.0
HP-UX	IA64, PA-RISC	8.0.0
IRIX	IA64, PA-RISC, MIPS	8.0.0 (excepto MIPS, 7.4)
Mac OS X	PowerPC	8.0.0
MandrakeLinux	x86	8.0.0
NetBSD	m68k,x86, Sparc	8.0.0 (excepto Sparc, 7.4)
OpenBSD	Sparc, Sparc64, x86	8.0.0
Red Hat Linux	AMD64, IA64, PowerPC, PowerPC 64, S/390, S/390x, x86	8.0.0
Solaris	Sparc, x86	8.0.0
SUSE Linux	AMD64, IA64, PowerPC, PowerPC 64, S/390, S/390x, x86	8.0.0
Windows	x86	8.0.0
Windows con Cygwin	x86	8.0.0

Plataformas posibles para PostgreSQL

A continuación mostramos los paquetes que se requieren para compilar y construir postgres.

- la herramienta GNU make. Otros "make" puede que no funcionen. Esta herramienta se invoca mediante el comando gmake. Son pocos los sistemas que incorporan GNU make como "make" por defecto. Para conocer la versión que dispone nuestro sistema debemos ejecutar

```
gmake --version
```

Se recomienda usar la versión 3.76.1 o superior.

- un compilador de C ISO/ANSI. Cualquier compilador gcc reciente cubre esta necesidad.

- la biblioteca GNU Readline. Se usa por defecto y permite una edición en línea agradable y una cómoda recuperación del historial.

- kerberos, openssl, y/o PAM si vamos a usar algún método de autenticación que comentamos anteriormente. Para nuestro caso disponemos ya de PAM en nuestro sistema, que va a ser el método que utilicemos en la integración de la autenticación.

6.5.3 Pasos a seguir

Como ya hemos dicho anteriormente vamos a construir el software de servidor (y también el de cliente, en este caso) de la versión 7.4.

Los pasos a seguir son los siguientes:

1)Descargar el archivo de la versión que hemos comentado. Se trata del fichero postgresql-7.4.5.tar.bz2.

2)Desempaquetar y descomprimir. Para ello debemos ejecutar lo siguiente.

```
tar –xvjf postgresql–*.tar.bz2
```

Lo cual creará un directorio postgresql-7.4.5

3)Compilar.

Para ello debemos ejecutar el script configure. En este caso sólo tenemos que pasarle un parámetro para que se ajuste a nuestras exigencias. Debemos habilitar el uso de PAM para la autenticación.

```
cd postgresql–*
./configure --with-pam
gmake
gmake install
```

Por defecto se instalará en /usr/local/pgsql.

Ya tenemos instalado el servidor postgres, aunque todavía no puede arrancar, ni siquiera tiene una base de datos a la que acceder. Necesita al menos una para poder examinar el nuevo software instalado.

Los pasos para comenzar a trabajar con postgres los veremos en el próximo apartado: configuración básica.

6.5.4 Configuración básica

Partiendo del paquete postgresql instalado, lo primero que debemos hacer es crear el usuario postgres. Es conveniente, siempre que tengamos un servidor de cara al exterior, ejecutarlo bajo una cuenta de usuario aislada. Esta cuenta de usuario sólo debería poseer los datos que gestiona el servidor y que no debería compartir con otros demonios. Por ello creamos una nueva cuenta y no usamos la del usuario “nobody”, por ejemplo.

Tampoco se aconseja instalar ningún ejecutable mediante esta cuenta ya que se comprometería la seguridad (riesgo de exploits), además de que podríamos perder las funciones definidas de usuario.

Debemos crear por tanto una cuenta para el usuario postgres. En un sistema operativo tipo Linux, debemos ejecutar lo siguiente:

```
adduser postgres
```

La ejecución de este comando exigirá una clave. Le asignamos una que sea válida y así está completado el primer paso.

El siguiente es iniciar el área de almacenamiento de base de datos en el disco duro. Esto se conoce como cluster de base de datos. Como ya hemos comentado anteriormente, un cluster de base de datos no es más que una colección de bases de datos que son accesibles a través de una sola

instancia del servidor de base de datos. Tras la iniciación, el cluster sólo contiene una base de datos, llamada template1 (plantilla1). Como su nombre sugiere, se usará como plantilla para todas las bases de datos que se creen a continuación. De hecho, no debería usarse para trabajar directamente con ella.

Los pasos a seguir para realizar lo dicho son los siguientes:

```
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
su - postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data
```

En términos de sistemas de ficheros, un cluster de bases de datos, no es más que un único directorio, bajo el cual se almacenarán todos los datos. Se suele llamar el directorio de datos o el área de datos y puede elegirse en cualquier localización. De hecho hay que hacerlo, no viene ninguno por defecto.

Es bastante común elegir /usr/local/pgsql/data ya que todas las herramientas del servidor se encuentran en su directorio padre. También podemos usar una partición distinta de donde está montado el sistema operativo o incluso otro disco, con lo que nos permite un mayor aislamiento de los datos.

Por último, y tras cambiar de usuario a postgres, ejecutamos el comando initdb, que se encarga de iniciar el cluster en la localización indicada. Sólo puede ser ejecutado por este usuario, con lo que asegura que ningún otro puede acceder a los datos del directorio.

Antes de poder acceder a la base de datos, debemos arrancar el servidor. El servidor de bases de datos se llama postmaster. Para arrancar debemos

invocarlo indicándole la localización del área de datos, de esta forma:

```
/usr/local/pgsql/bin/postmaster –D /usr/local/pgsql/data > logfile 2> $1 &
```

Además hemos redirigido la salida estándar y la de errores al fichero logfile y lo hemos dejado en segundo plano.

La sintaxis que hemos usado es un poco engorrosa. Por ello, junto con los binarios, se suministra un script, pg_ctl, que nos permitirá de una forma fácil, arrancar el servidor, especificar ficheros de log, e incluso apagarlo.

Estas características hacen que sea ideal para integrarlo en los scripts de inicio y parada de sistema, y de hecho, lo usaremos para crear dichos scripts.

Indicamos a modo de ejemplo una forma alternativa de arrancar el servidor usando el script pg_ctl:

```
pg_ctl start –l logfile
```

Tal como lo hemos escrito exige que la variable de entorno PGDATA indique el área de datos. Si no lo está, podemos definirla, o también indicar el área de datos mediante la opción -D, tal como hacíamos con el comando postmaster.

6.5.5 Configuración avanzada. Integración.

Mediante la configuración básica presentada en el apartado anterior, el sistema estaría preparado para comenzar a crear bases de datos, y en ellas, poder añadir tablas, campos, funciones y recibir peticiones, entre otros. Con los valores fijados por defecto podríamos comenzar a trabajar, sin ningún problema.

En este apartado vamos a configurar la autenticación de postgres, vamos a integrarla, tal como hemos hecho con los anteriores paquetes.

La autenticación en postgres se configura mediante el fichero pg_hba.conf , que según hemos creado el área de datos, se encuentra en el directorio /usr/local/pgsql/data.

Por defecto, postgres utiliza una configuración muy liberal, tal como se señala en el mismo fichero pg_hba.conf. Veámoslo.

```
# CAUTION: The default configuration allows any local user to connect
# using any PostgreSQL user name, including the superuser, over either
# Unix-domain sockets or TCP/IP.  If you are on a multiple-user
# machine, the default configuration is probably too liberal for you.
# Change it to use something other than "trust" authentication.
#
# TYPE  DATABASE    USER      IP-ADDRESS          IP-MASK         METHOD
  local   all       all                                         trust
  host    all       all       127.0.0.1         255.255.255.255   trust
```

En el fichero original aparece otra entrada, que se usaría para direcciones de tipo ipv6. No la vamos a tener en cuenta, así que ni siquiera lo vamos a mostrar.

En cuanto a la autenticación se usa el método "trust" tanto para las conexiones locales que usan sockets de Unix (local) como para las que hacen uso de redes TCP/IP (host).

Según la política integradora que estamos siguiendo, la conexión local también se autenticará contra el servidor OpenLDAP, luego debemos cambiar el método trust por pam en las entradas que acabamos de mostrar.

También debemos añadir entradas para las conexiones que se produzcan desde la red. Para nuestro caso, la red que estamos utilizando de prueba tiene una ip base 192.168.123.100 con una máscara 255.255.255.0.

El fichero pg_hba.conf quedará así:

```
# TYPE  DATABASE    USER      IP-ADDRESS        IP-MASK           METHOD
 local   all         all                                          pam
 host    all         all       127.0.0.1         255.255.255.255   pam
 host    all         all      192.168.123.100    255.255.255.0     pam
```

Es labor del administrador de red (o del administrador de postgres, aunque suelen ser la misma persona) decidir qué usuarios tienen permiso para acceder a una determinada base de datos o para crearla. Por tanto, deben rellenar las entradas necesarias, con los valores adecuados, y también crear los usuarios según se precise.

Por último mostramos el fichero /etc/pam.d/postgresql en el que recae, al fin y al cabo, toda la autenticación.

```
# Fichero /etc/pam.d/postgresql
auth            sufficient      pam_ldap.so
account         sufficient      pam_ldap.so
```

6.5.6 Ejemplo de uso

Partiendo del sistema preparado mediante la configuración básica vamos a ver la manera de comenzar a trabajar con él.

En este momento, con el sistema recién instalado y configurado, no disponemos de ninguna base de datos a la que poder acceder, así que lo primero que vamos a realizar es la creación de una base de datos de prueba.

Se llamará test. Para crearla:

```
/usr/local/pgsql/bin/createdb test
```

Debemos tener en cuenta que la ejecución del comando anterior debe hacerse desde la cuenta del usuario postgres. Recordemos que es la primera vez que accedemos al sistema de gestión y que, por tanto, todavía no hemos creado otros usuarios que puedan acceder. Veremos como hacerlo posteriormente.

Ya creada la base de datos test, el siguiente paso es acceder a ella. Para ello vamos a utilizar el terminal interactivo psql.

Esta no suele ser la forma habitual. Lo normal es acceder mediante una aplicación. Existen APIs para los lenguajes más comunes, siendo PHP uno de los más usados debido, entre otras cosas, a su facilidad de uso.

Nuestra intención no es más que mostrar su uso, por lo que el terminal pgsl será suficiente para tal labor. Para activar dicho terminal debemos ejecutar:

```
psql test
```

Su ejecución nos mostrará una serie de mensajes y un prompt en el que escribir los comandos SQL que deseemos.

```
Welcome to psql 7.4.5, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

test=#
```

El mensaje de bienvenida nos muestra algunos de los códigos de escape que es capaz de interpretar.

Vamos a introducir algunos datos y a realizar ciertas operaciones sobre la base de datos, a modo de ejemplo para mostrar su funcionamiento. Para ello utilizaremos el fichero basics.sql que se proporciona con el código fuente de postgres.

En este fichero podemos encontrar la definición de varias tablas y la inserción de algunas filas en ella, usando para ello sintaxis SQL. De una forma sencilla estamos poblando la base de datos, para después pasar a probarla. Para poblarla debemos ejecutar:

```
test=# \i basics.sql
```

En basic.sql no hay nada más que la definición de dos tablas. En la primera de ellas, ciudades, podemos consultar información de dos ciudades, en este caso Sevilla y Cádiz. En la segunda, tiempo, tenemos algunas estadísticas meteorológicas de ellas.

El fichero también realiza algunas consultas. Podemos ver el resultado de éstas tras la ejecución de la orden anterior. Los mostramos a continuación de forma que queda comprobado, al menos en un primer acercamiento, que el gestor de base de datos funciona correctamente.

```
test=# \i basico.sql
CREATE TABLE
CREATE TABLE
INSERT 17225 1
INSERT 17226 1
INSERT 17227 1
INSERT 17228 1
 ciudad  | temp_baja | temp_alta | prcp |   fecha
---------+-----------+-----------+------+------------
 Sevilla |        12 |        25 | 0.25 | 2005-03-16
 Sevilla |        11 |        23 |    0 | 2005-03-14
 Cadiz   |        12 |        20 |      | 2005-03-16
(3 rows)
```

```
 ciudad  | temp_avg |   fecha
---------+----------+------------
 Sevilla |       18 | 2005-03-16
 Sevilla |       17 | 2005-03-14
 Cadiz   |       16 | 2005-03-16
(3 rows)
```