

**PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR
FACULTAD DE INGENIERÍA
ESCUELA DE SISTEMAS**

**DISERTACIÓN PREVIA PARA LA OBTENCIÓN DEL
TÍTULO DE INGENIERO EN SISTEMAS**

**GUÍA DE LAS MEJORES PRÁCTICAS
ADMINISTRATIVAS, SEGURIDAD Y ALTA
DISPONIBILIDAD, CASO DE ESTUDIO: POSTGRESQL**

JONATHAN ISSAC NARVAEZ COELLO

DIRECTORA: ANITA URGILES

QUITO, 2014

TABLA DE CONTENIDOS

RESUMEN	1
INTRODUCCIÓN.....	2
OBJETIVOS.....	3
General.....	3
Específicos	3
CAPÍTULO I: MARCO TEÓRICO	4
1.1. CONCEPTOS BÁSICOS	4
1.1.1. Base de Datos	4
1.1.2. Sistema de Gestión de Base de Datos.....	4
1.1.3. Arquitectura de los Sistemas de Gestión de Base de Datos	5
1.1.3.1. Nivel interno o físico.....	5
1.1.3.2. Nivel conceptual.....	6
1.1.3.3. El Nivel externo o de visión.....	6
1.1.4. Componentes de los SGDB	7
1.1.4.1. Lenguajes de un SGBD	7
1.1.4.2. Diccionario de Datos.....	7
1.1.4.3. Seguridad e Integridad de los Datos.....	8
1.1.5. Clasificación de los Sistemas de Gestión de Base de Datos.....	9
1.1.5.1. Según el Modelo de Datos	9
1.1.5.2. Según el Número de Usuarios.....	11
1.1.5.3. Según el Número de Sitios	12
1.1.5.4. Según su Propósito.....	12
1.1.5.5. Según su Licencia	13
1.2. POSTGRESQL	14
1.2.1. ¿Qué es PostgreSQL?	14
1.2.2. Características.....	15
1.2.2.1. Características Generales	15
1.2.2.2. Características de Desarrollo y Programación	15
1.2.2.3. Características de SQL	16
1.2.3. Licencia y Versión.....	16
1.3. BUENAS PRÁCTICAS	17
1.4. PLATAFORMAS Y HERRAMIENTAS	18
1.4.1. Windows.....	18

1.4.2.	CentOS	19
1.4.3.	Herramienta GUI para PostgreSQL.....	20
1.4.4.	Herramienta para Alta Disponibilidad.....	21
CAPÍTULO II: BUENAS PRÁCTICAS EN ADMINISTRACIÓN		23
2.1.	ADMINISTRACIÓN EN POSTGRESQL	23
2.2.	MANEJO DE LA ADMINISTRACIÓN.....	24
2.2.1.	Instalación de PostgreSQL sobre CentOS 7	24
2.2.1.1.	Utilizando la versión incluida en la distribución.....	25
2.2.1.2.	Utilizando el repositorio Yum actualizado	27
2.2.1.3.	Utilizando el instalador grafico de Linux.....	30
2.2.2.	Puerto de conexión al Servidor.....	31
2.2.3.	Parámetros de Conexión.....	33
2.2.4.	Afinamiento.....	37
2.2.4.1.	Sistema de Discos	37
2.2.4.2.	Configuración de parámetros de rendimiento.....	39
2.2.4.3.	Recursos en la escritura sobre el disco.....	42
2.2.5.	VACUUM y ANALYZE.....	46
2.2.5.1.	AUTOVACUUM.....	49
2.2.6.	Backups	53
2.2.7.	Restauración de BD.....	55
2.2.8.	Conexión al Servidor.....	57
2.3.	CUADRO DESCRIPTIVO.....	60
CAPÍTULO III: BUENAS PRÁCTICAS DE SEGURIDAD		62
3.1.	SEGURIDAD EN POSTGRESQL.....	62
3.2.	MANEJO DE LA SEGURIDAD	63
3.2.1.	Cambio de contraseña usuario postgres	63
3.2.2.	Usuarios y Privilegios.....	65
3.2.3.	Seguridad en el fichero pg_hba.conf	69
3.2.4.	Diseño de Datos.....	75
3.2.4.1.	Separación física	75
3.2.4.2.	Separación Lógica.....	77
3.3.	CUADRO DESCRIPTIVO.....	79
CAPÍTULO IV: BUENAS PRÁCTICAS EN ALTA DISPONIBILIDAD		81
4.1.	ALTA DISPONIBILIDAD EN POSTGRESQL	81
4.2.	ALTA DISPONIBILIDAD BAJO REPLICACIÓN	84

4.2.1.	Porque usar Replicación	84
4.2.1.1.	Replicación vs Backups	85
4.2.2.	Elección de Plataforma	87
4.2.3.	Arquitectura	88
4.2.4.	Instalación y Configuración.....	92
4.2.4.1.	PostgreSQL	92
4.2.4.2.	Pgpool-II	96
4.2.4.3.	Conexión a través de Pgpool-II.....	103
4.2.5.	Pruebas de Replicación.....	105
4.3.	CUADRO DESCRIPTIVO.....	110
CONCLUSIONES		111
RECOMENDACIONES		113
REFERENCIAS BIBLIOGRÁFICAS		114

RESUMEN

La presente disertación: Guía de las Mejores Prácticas Administrativas, Seguridad y Alta Disponibilidad, en el Caso de Estudio: PostgreSQL; tiene como objetivo, recopilar aquellas prácticas que se consideran como optimas en la gestión de la parte administrativa, en la seguridad y en el uso de alta disponibilidad disponibles en el Sistema de Gestión de Bases de Datos PostgreSQL. A fin de desarrollar una Guía que contenga tales resultados y se convierta en una fuente útil de consulta.

El Capítulo I contiene conceptos básicos de base de datos, introducción a PostgreSQL y un resumen del concepto de buena práctica aplicada a informática. Toda esta información servirá como introducción para el resto de Capítulos.

En el Capítulo II se realizará una recopilación de aquellas buenas prácticas en la parte Administrativa de PostgreSQL, donde se abarcará temas esenciales en aspectos como la configuración, rendimiento y copias de seguridad.

Con el Capítulo III se determinarán las características de seguridad y prácticas recomendables disponibles en PostgreSQL, teniendo énfasis en el acceso y permisos que posee un usuario de la base de datos. La Alta Disponibilidad con el uso de replicación, serán abarcadas en el Capítulo IV, donde se implementará un ejemplo que sirva de refuerzo para los temas definidos.

La disertación finalizará con las conclusiones y recomendaciones recopiladas de los cuatro capítulos desarrollados.

INTRODUCCIÓN

Las Bases de Datos están constituidas en la actualidad, como el pilar fundamental de todo sistema de la información y su uso se encuentra extendido en la mayor parte de empresas, microempresas, organismos gubernamentales e instituciones académicas.

Actualmente su difusión más común se encuentra en los Sistemas de Gestión de Base de Datos los cuales, consisten en un conjunto de programas que ayudan a definir, construir y manipular una base de datos. Su objetivo principal es proporcionar una forma de almacenar y recuperar la información de la base de datos de manera tanto practica como eficiente.

Con la aparición de este tipo de herramientas, se ha promovido el desarrollo de opciones “open source” que proveen utilidades similares sin necesidad de invertir un alto costo de licenciamiento; adicional a esto, en algunos gobiernos se ha adoptado la política de utilización de herramientas de software libre, como es el caso de nuestro país. Un caso claro se encuentra con PostgreSQL máximo exponente de esta tendencia, que se presenta como un Sistema de Gestión de Base de Datos robusto cuyas características y ventajas han hecho que su popularidad y uso crezca en los últimos años.

PostgreSQL cuenta con herramientas para su Administración, opciones y características de seguridad y alternativas para alta disponibilidad de servicio; las que requieren un número de prácticas y actividades que fomente su óptima utilización, para sacar el mayor provecho al uso de este Sistema de Gestión de Base de Datos.

OBJETIVOS

General

Generar una guía con las mejores prácticas administrativas, de seguridad y alta disponibilidad presentes en PostgreSQL.

Específicos

- Determinar un mínimo de 12 buenas prácticas para la parte de administración de PostgreSQL, que abarquen aspectos como: Configuración de parámetros de conexión, Afinamiento, VACUUM, ANALYCE, generación de Backups y Restauración.
- Generar más de 8 buenas prácticas en seguridad de PostgreSQL, enfocadas en temas como: Manipulación de Ficheros, Acceso de Usuarios, Privilegios y Manejo de la estructura lógica y física.
- Recopilar un mínimo de 5 buenas prácticas en el desarrollo de alta disponibilidad de PostgreSQL, definiendo temas necesarios como: Justificaciones, Plataformas, Arquitectura, Parámetros a configurar y Pruebas.
- Implementar un ejemplo de alta disponibilidad con replicación, usando un clúster de dos nodos para la demostración.
- Manejar 2 plataformas para el desarrollo de la guía en los temas de administración y seguridad de PostgreSQL.
- Generar un mínimo de 5 conclusiones y 5 recomendaciones que abarquen las características más importantes encontradas en el desarrollo de la disertación.

CAPÍTULO I: MARCO TEÓRICO

1.1. CONCEPTOS BÁSICOS

1.1.1. Base de Datos

Una BD¹, constituye un conjunto organizado de datos integrados, almacenados en memoria secundaria no volátil y con redundancia controlada. Los datos compartidos por diferentes usuarios y aplicaciones mantienen independencia entre ellos, y constan de una estructura única almacenada junto con los datos. Esto permitirá captar las interrelaciones y restricciones existentes en el mundo real.

Los procedimientos de actualización y recuperación definidos con claridad, facilitarán la seguridad del conjunto de datos (Castaño & Piattini, 1999).

1.1.2. Sistema de Gestión de Base de Datos

Un SGBD², es una colección de datos estructurados y organizados que se encuentran relacionados entre sí, y el conjunto de programas y herramientas que acceden y gestionan esos datos. Su propósito general es manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante para el usuario.

¹ BD – Base de Datos

² SGBD - Sistemas de Gestión de Bases de Datos

Entre los servicios más importantes que un SGDB presenta son:

- La creación y definición de la BD, especificando su estructura, tipos de datos, restricciones y relaciones.
- La manipulación de los datos, con consultas, inserciones y actualizaciones.
- El control del acceso de los usuarios a la BD mediante mecanismos de seguridad
- Mantener la integridad y consistencia de los datos.
- Controlar el acceso e interacción entre usuarios concurrentes a la BD.
- La disponibilidad de mecanismos de respaldo y recuperación en caso de errores en el sistema.

1.1.3. Arquitectura de los Sistemas de Gestión de Base de Datos

Se propone una arquitectura donde el esquema de una BD se define en tres niveles de abstracción los que son:

1.1.3.1. Nivel interno o físico

Describe la estructura física de la BD mediante un esquema. Este esquema contiene los detalles de cómo se almacenan físicamente los datos: los archivos que contienen la información, su organización, los métodos de acceso a los registros, tipos de registros, los campos que la componen y su longitud.

1.1.3.2. Nivel conceptual

Representa la información contenida en la BD al igual que detalla su estructura para un grupo de usuarios mediante un esquema conceptual. El esquema describe las entidades, atributos, relaciones, operaciones y restricciones de los usuarios, ocultando los detalles de las estructuras físicas de almacenamiento.

1.1.3.3. El Nivel externo o de visión

Es el más cercano a los usuarios, en donde se describiendo los esquemas externos o vistas de usuarios. En este nivel se representa la visión del usuario o de un grupo de usuarios de la BD.

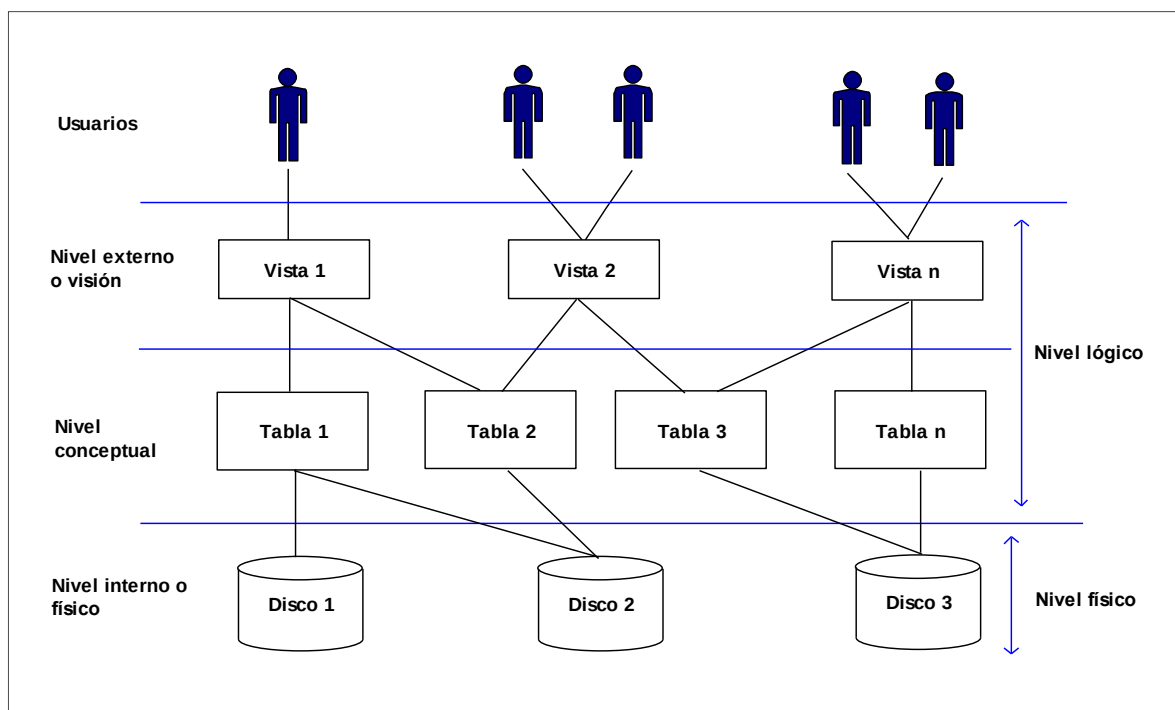


Grafico 1.1: Niveles de abstracción de la arquitectura ANSI (McGraw-Hill, 2006).

1.1.4. Componentes de los SGDB

Entre los componentes que conforman a un SGDB (McGraw-Hill, 2006) tenemos:

1.1.4.1. Lenguajes de un SGBD

Los lenguajes de un SGBD permiten al administrador especificar los datos que componen la BD, su estructura, las relaciones que existen entre ellos, las reglas de integridad, los controles de acceso, las características de tipo físico y las vistas externas de los usuarios. Estos lenguajes se pueden clasificar en:

- **Lenguaje de definición de datos DDL³**, el cual define el esquema conceptual y el esquema interno de la BD, especifica, las vistas de los usuarios y las estructuras de almacenamiento.
- **Lenguaje de manipulación de datos DML⁴**, se utiliza para leer y actualizar los datos de la BD como: consultas, inserciones, eliminaciones y modificaciones.

1.1.4.2. Diccionario de Datos

El diccionario de datos es el depósito de la información de todos los datos que forman parte de la BD, conteniendo sus características y los sitios en el sistema donde se almacenan. El diccionario de datos provee información de:

- La definición de la estructura lógica y física de la BD junto con sus objetos (espacio asignado y utilizado) como: tablas (valores por defecto de sus columnas), vistas, funciones, índices, store procedures, triggers, etc.

³ DDL – Data Definition Language

⁴ DML – Data Manipulation Language

- Administración de los privilegio y roles asignados a cada usuario, junto a la información de restricciones de seguridad aplicadas y herramientas para realizar una auditoría de información en el acceso a los objetos.

El diccionario de datos a parte de estar integrado dentro del SGBD, debe cumplir con las siguientes características:

- Su ubicación debe estar en un medio de almacenamiento con acceso directo, esto ayudara a simplificar y facilitar la recuperación de la información.
- Las descripciones de los modelos lógico, conceptual, interno y externo de la BD, deben estar contenidas.
- Los cambios tanto en la descripción de la BD como en la descripción de programas deben ser reflejados en el diccionario de datos, para la descripción de programas los cambios se registran automáticamente en la librería de descripción de programas. Esto ayuda y mejora la reorganización de las versiones disponibles de la BD.
- Realizar una transferencia eficiente de la información al SGBD con una conexión realizada en tiempo de ejecución entre los modelos interno y externo.

1.1.4.3. Seguridad e Integridad de los Datos

La seguridad e integridad de los datos implica vigilar que las operaciones y modificaciones hechas en la BD no comprometan su definición, existencia y consistencia de los datos; por lo tanto, se tiene muy en cuenta en un SGBD las siguientes características:

- El manejo de mecanismos para implementar restricciones de consistencia y reglas de integridad, que verifiquen los valores de los datos que se almacenan en la BD.
Esto es una medida para prevenir daños accidentales en los datos y violaciones a las especificaciones en seguridad e integridad dispuestas por el administrador.
- Controlar el acceso de los usuarios a la BD, para evitar el acceso de usuarios no autorizados tanto intencionales como accidentales.
- La disponibilidad de herramientas y mecanismos para planificar y realizar copias de seguridad para una eventual restauración y recuperación de la BD hacia un estado consistente.
- Conservar la consistencia de los datos, en el caso que varios usuarios accedan y actualicen de forma concurrente la BD.

1.1.5. Clasificación de los Sistemas de Gestión de Base de Datos

1.1.5.1. Según el Modelo de Datos

1.1.5.1.1. Relacional

Los RDBMS⁵ utilizan como base el modelo relacional, en donde los datos se describen como relaciones representadas por tablas bidimensionales que contiene: filas o tuplas representando ocurrencias e identificadas por una primary key y columnas o atributos que representan las propiedades de la fila.

⁵ RDBMS - Sistema de Gestión de Base de Datos Relacional

Esta forma de organizar la información de la BD, permite recuperar los datos de forma flexible de una o varias tablas, y la posibilidad de combinar registros de diferentes tablas para formar nuevas.

1.1.5.1.2. Orientado a Objetos

Los ODBMS⁶ se basan en el paradigma orientado a objetos, el cual representa a la información por medio de objetos. Se fundamenta en la integración de la BD con un lenguaje de programación orientado a objetos, representando a los objetos de la BD como objetos del lenguaje de programación; esto permite extender de forma transparente los lenguajes con datos persistentes, controlar su concurrencia, su recuperación y la realización de consultas asociativas.

1.1.5.1.3. Objeto-Relacional

Los ORDBMS⁷, están basados en los SGBDR con características de los sistemas orientados a objetos. Las filas o tuplas pueden formar una relación, produciendo una relación de relaciones; este concepto se acerca a la orientación a objetos, donde existe la posibilidad de almacenar objetos complejos en una tabla referenciada a otras relaciones.

⁶ ODBMS - Sistema de Gestión de Base de Datos Orientado a Objetos

⁷ ORDBMS - Sistema de Gestión de Base de Datos Objeto Relacional

1.1.5.1.4. De Red

Se basan en el modelo de datos de red, representado mediante colecciones de registros, junto a las relaciones entre los datos representadas mediante enlaces vistos como punteros. Los registros en la BD están organizados como colecciones de grafos definidos.

1.1.5.1.5. Jerárquico

Basados en el modelo de datos jerárquico, en donde los datos y sus relaciones están representados mediante registros y enlaces. Los registros en la BD están organizados como colecciones de árboles.

1.1.5.2. Según el Número de Usuarios

1.1.5.2.1. Monousuario

Los SGBD monousuarios, pueden atender a solo un usuario a la vez, se utilizan en computadores personales y para pruebas.

1.1.5.2.2. Multiusuarios

Los SGBD multiusuarios, conformar la mayoría de SGBD presentes y su característica es la de atender a uno o más usuarios en el mismo tiempo.

1.1.5.3.Según el Número de Sitios

1.1.5.3.1. Centralizados

En los SGBD centralizados, el software del SGBD y la información de la BD se encuentran almacenados en una sola máquina, independientemente del número de usuarios que acceden a la BD.

1.1.5.3.2. Distribuidos

Los SGBD distribuidos pueden tener al software del SGBD y a la BD distribuidas en varios sitios comunicados por una red; se dividen en homogéneos: Uso del mismo SGDB en varios sitios, y heterogéneos: Varias BD autónomas almacenadas en el SGBD.

1.1.5.4.Según su Propósito

1.1.5.4.1. Propósito General

Estos SGBD no discriminan a ninguna aplicación que requiere acceso a la información de la BD.

1.1.5.4.2. Propósito Específico

Es un SGBD desarrollado para un determinado grupo de aplicaciones, se busca el rendimiento y el fácil acceso a la BD.

1.1.5.5. Según su Licencia

1.1.5.5.1. Comercial

Es aquel SGDB que basa su modelo de negocio en aspectos como: costo de licencia, costo de servicios, soporte, distribución, etc.

1.1.5.5.2. FLOSS

El SGBD FLOSS⁸, es el software licenciado de tal manera que los usuarios pueden estudiar, modificar y mejorar su diseño mediante la disponibilidad de su código fuente y se acoge a las 4 libertades del Software Libre y a las 10 premisas del Software de Código Abierto.

En la actualidad el desarrollo de SGBD bajo estas características está teniendo una gran acogida por los usuarios al igual que por varias empresas y organizaciones, debido a sus prestaciones, adaptabilidad y costo.

⁸ FLOSS – Siglas en Ingles de Software Free/Libre y Open Source/Código Abierto

1.2. POSTGRESQL

1.2.1. ¿Qué es PostgreSQL?

PostgreSQL es un ORDBMS basado en el proyecto POSTGRES versión 4.2, y desarrollado en la Universidad de California en el departamento de ciencias de la computación de Berkeley. PostgreSQL constituye como el descendiente FLOSS del original desarrollado en Berkeley, pudiendo ser ejecutado en la mayoría de sistemas operativo, incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) Y Windows.

PostgreSQL soporta la mayoría del estándar ANSI SQL⁹:2008 y el control de primary keys, foreign keys, joins, vistas, triggers y store procedures. Incluye la mayor parte de tipos de datos presentes en SQL: 2008 (INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, TIMESTAMP, INTERVAL), junto a la posibilidad de almacenar grandes objetos binarios incluyendo, imágenes, sonidos, y videos.

PostgreSQL posee interfaces de programación nativa para C/C++, Java, .Net, Perl, Python, Ruby, ODBC; y utiliza el modelo cliente/servidor usando multiprocesos, de esta forma garantiza la estabilidad del sistemas frente a fallos.

Actualmente a PostgreSQL se lo considera como el SGBD Floss más potente del mercado, sus características y prestaciones administrativas, en disponibilidad de los datos, seguridad e integridad de la información en las últimas versiones, se encuentran a la par de las que poseen opciones comerciales como Oracle y SQL Server.

⁹ SQL – Lenguaje de Consulta Estructurado

1.2.2. Características

1.2.2.1. Características Generales

- Soporta completamente ACID¹⁰.
- Estándar ANSI SQL92, 99, 2003, 2008.
- Integridad referencial bajo Foreign keys.
- Manejo de objetos y sus relaciones.
- Tablespaces y Nested transactions (savepoints).
- PITR¹¹.
- MVCC¹².
- Copias de seguridad en caliente (hot backups).
- Manejo de Unicode y juegos de caracteres internacionales.
- Posee múltiples métodos y opciones de autenticación.
- Soporte nativo de SSL¹³ para la encriptación del acceso.
- Manejo de actualizaciones integradas in-situ (pg_upgrade).
- Disponibilidad para Linux y UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), BeOS y Windows 32/64bit.

1.2.2.2. Características de Desarrollo y Programación

- Manejo de Store Procedures con soporte en los lenguajes PL/pgSQL, PL/Tcl, PL/Python y PL/Perl.

¹⁰ ACID - Atomicidad, Consistencia, Aislamiento y Durabilidad

¹¹ PITR - Point in Time Recovery

¹² MVCC - Multi-Version Concurrency Control

¹³ SSL - Secure Sockets Layer

- Posee interfaces nativas para ODBC, JDBC, C, C++, PHP, Perl, TCL, ECPG, Python y Ruby.
- Posibilidad de definir nuevos tipos de datos, funciones y operadores.
- Provee las APIs¹⁴ para programar en lenguajes como: C/C++, Java, .Net, PHP, Ruby, ODBC, Perl, Python, Tcl, Lisp, Scheme, entre otros.

1.2.2.3. Características de SQL

- Uso de Primary keys y Foreign keys.
- Soporte de Selects anidados y Joins.
- Manejo de constrains Check, Unique y not null.
- Reglas, Vistas, Triggers (comunes, por columna y condicionales), Store Procedures, Cursores y Secuencias.
- Soporte de Herencia entre tablas.
- Índices únicos, compuestos, parciales u funcionales usando los métodos de almacenamiento: B-tree, R-tree, hash o GiST.

1.2.3. Licencia y Versión

Trabaja bajo la licencia libre BSD¹⁵, permitiendo que PostgreSQL pueda ser usado, modificado y distribuido por cualquier usuario sin costo alguno, para cualquier propósito sea privado, comercial o académico. PostgreSQL se encuentra en la versión 9.3.2 tanto para los Sistemas Operativos: Windows, Linux, Solaris, Mac OS X.

¹⁴ API - Application Programming Interface - Interfaz de Programación de Aplicaciones

¹⁵ BSD - Berkeley Software Distribution

1.3. BUENAS PRÁCTICAS

Las buenas prácticas se refieren a todo método guiado por experiencia previa, que cumple los principios, objetivos y procedimientos recomendables para brindar resultados positivos en un respectivo caso práctico. (FEAPS, 2014)

La gestión de calidad hace uso de las buenas prácticas aprovechando las soluciones eficientes y eficaces que proveen, logrando resolver problemas, alcanzar metas u objetivo y generar oportunidades de mejora.

Las buenas prácticas se aplican a una base de datos en todas sus actividades y operaciones, logrando así un mayor desempeño, aseguramiento de la seguridad e integridad de la información y garantizar un óptimo control y administración de la base de datos.

En PostgreSQL se hace uso de las buenas prácticas, en aspectos claves que garantizan el buen uso del SGBD, permitiendo a los usuarios optimizar sus operaciones y mejorar la experiencia de uso DE PostgreSQL.

Las mejores prácticas de PostgreSQL son fruto de la experiencia difundida por los usuarios y empresas especializadas en mejoramiento de la calidad.

1.4. PLATAFORMAS Y HERRAMIENTAS

Para la presente disertación, se utilizarán dos plataformas para la implementación de las mejores prácticas.

1.4.1. Windows

Windows en su versión 8.1 Pro, es la versión de Sistema Operativo actual de Microsoft Windows, que presenta cambios en su menú de inicio, conectividad y aplicación para la interacción con el usuario.

Con Windows 8.1 Pro, se demostrará su factibilidad en el uso PostgreSQL junto a la aplicación de sus buenas prácticas, de manera que se obtendrán las respectivas conclusiones y recomendaciones de los resultados observados.

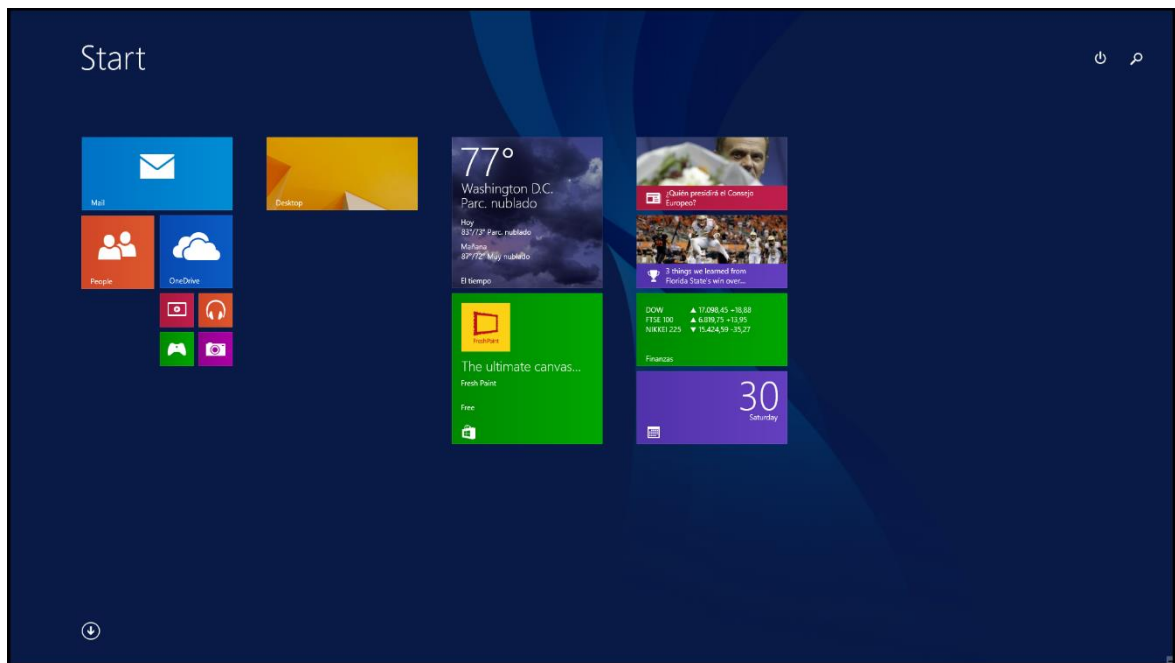


Grafico 1.2: Interfaz de inicio de Windows 8.1 Pro (Narváez, 2014).

1.4.2. CentOS

CentOS¹⁶ en su versión 7.0, es una distribución Linux bajo licencia GPL, basada a nivel binario en Red Hat. CentOS está enfocado en proporcionar una plataforma empresarial libre, apoyada por su comunidad.

Utilizando CentOS 7.0, se implementaran todas las buenas prácticas de PostgreSQL recopiladas en cada capítulo, sobre un ambiente Linux; de igual manera, se usará como plataforma principal para implementar un ejemplo de la función de alta disponibilidad de PostgreSQL.

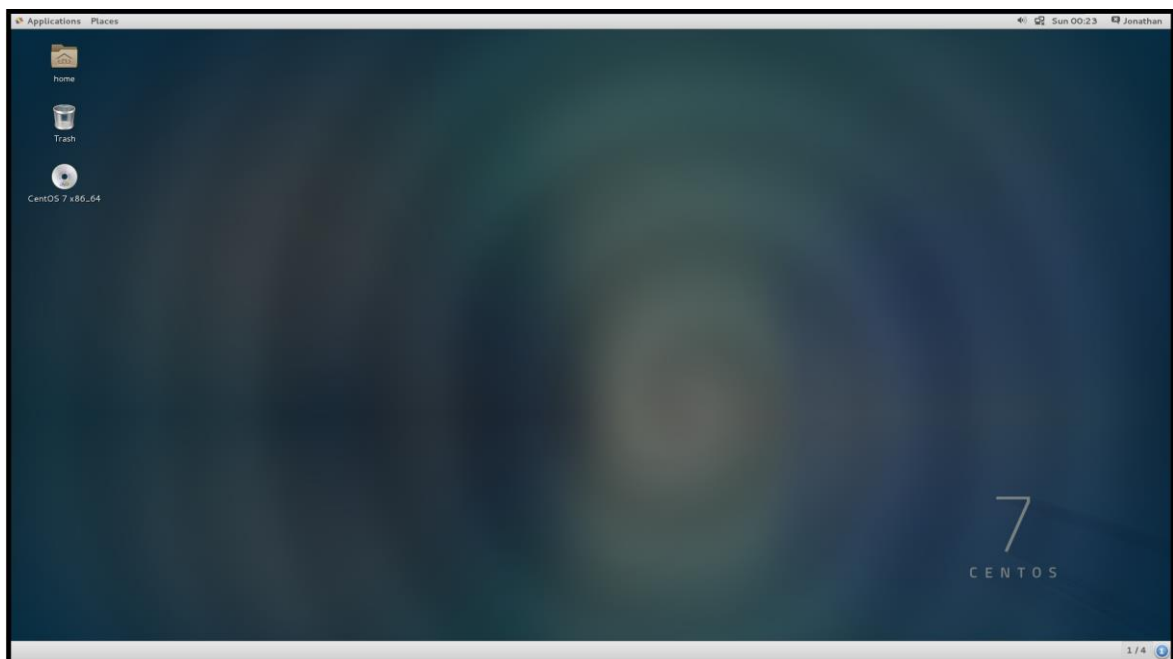


Grafico 1.3: Escritorio GNOME de CentOS 7.0 (Narváez, 2014).

¹⁶ CentOS - Community Enterprise Operating System

1.4.3. Herramienta GUI para PostgreSQL

PostgreSQL cuenta con un gran número de herramientas GUI¹⁷, de las cuales destaca y se encuentra incluida en el paquete de instalación EnterpriseDB PostgreSQL:

PgAdmin3: PgAdmin3 es una herramienta bajo la licencia BSD para la administración de bases de datos PostgreSQL e incluye:

- Interfaz administrativa gráfica.
- Herramienta de consulta SQL y editor de código procedural.
- Agente de planificación SQL/shell/batch y administración de Slony-I.

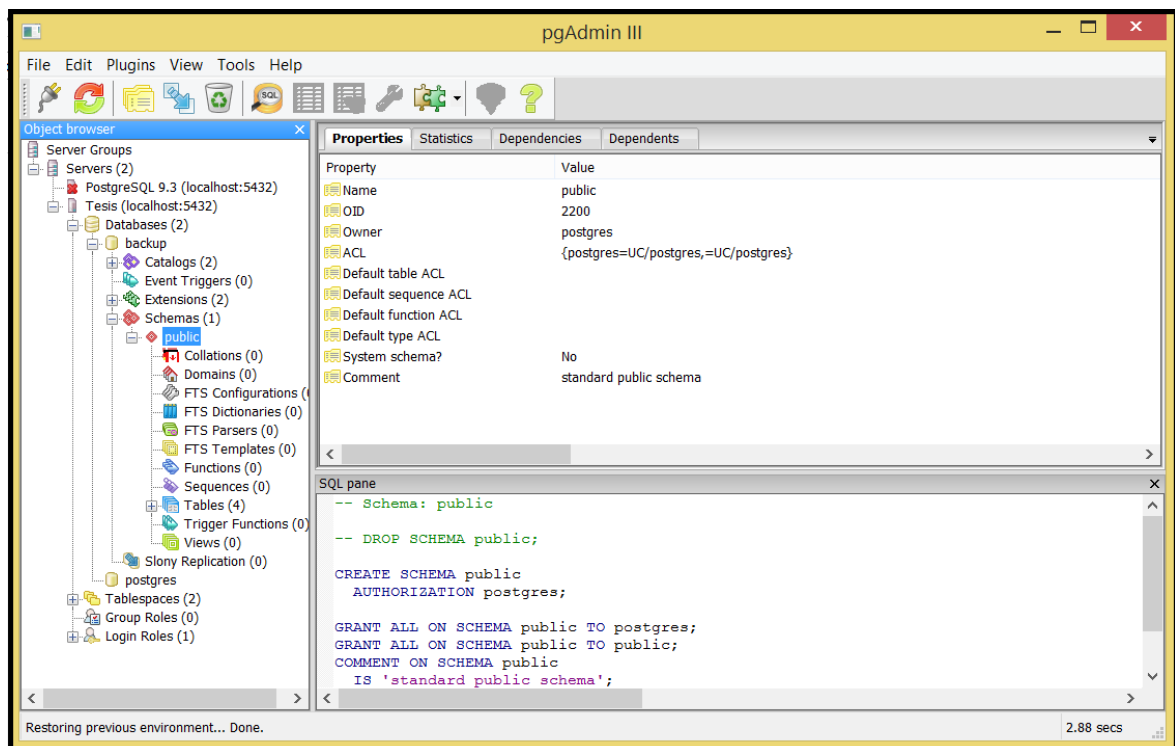


Grafico 1.3: Interfaz en Windows de PgAdmin3 (Narváez, 2014).

¹⁷ GUI – Interfaz Gráfica de Usuario

1.4.4. Herramienta para Alta Disponibilidad

Pgpool-II

Pgpool-II es una herramienta bajo la licencia BSD que trabaja entre los servidor maestro de la base de datos en PostgreSQL y sus respectivos nodos. Entre sus características se encuentran:

- **Pool de conexiones:** pgpool-II almacena las conexiones con los servidores de PostgreSQL, puede reusar estas conexiones en donde sea que de una nueva conexión con las mismas propiedades (nombre de usuario, BD, versión del protocolo). Se reduce la sobrecarga en la conexión y mejora el rendimiento del Sistema.
- **Replicación:** pgpool-II puede administrar varios servidores o nodos de PostgreSQL. Usando esta función podemos crear backups en tiempo real en más de dos discos sólidos.
- **Balanceo de la carga:** En una BD replicada la ejecución de un SELECT en cualquier nodo retornara el mismo resultado, con pgpool-II se reduce la carga resultante de realizar cada ejecución sobre cada nodo distribuyendo tales peticiones a cada nodo según el requerimiento.
- **Limitando el exceso de conexiones:** pspool-II posee un límite en el máximo número de conexiones que acepta evitando el consumo exagerado de recursos y un bajo rendimiento. Las conexiones que exceden el límite son enviadas a una cola de procesos.

Comparación con otras herramientas de Alta Disponibilidad

Programa	Licencia	Madurez	Método de Replicación	Sincronización	Pool de Conexión	Balanceo de la Carga	Posicionamiento de Consultas
PGCluster	BSD	En Espera	Maestro/Maestro	Síncrono	NO	SI	NO
Pgpool-I	BSD	Estable	Maestro/Esclavo	Síncrono	SI	SI	NO
Pgpool-II	BSD	Liberada Recientemente	Maestro/Esclavo	Síncrono	SI	SI	SI
Slony-I	BSD	Estable	Maestro/Esclavo	Asíncrono	NO	NO	NO
Bucardo	BSD	Estable	Maestro/Maestro Maestro/Esclavo	Asíncrono	NO	NO	NO
Londiste	BSD	Estable	Maestro/Esclavo	Asíncrono	NO	NO	NO
Mammoth	BSD	Estable	Maestro/Esclavo	Asíncrono	NO	NO	NO
Rubyrep	MIT	Liberada Recientemente	Maestro/Maestro Maestro/Esclavo	Asíncrono	NO	NO	NO

Tabla 1.1: Comparación entre Opciones de Alta Disponibilidad para PostgreSQL (Hernandez, 2014).

CAPÍTULO II: BUENAS PRÁCTICAS EN ADMINISTRACIÓN

2.1. ADMINISTRACIÓN EN POSTGRESQL

PostgreSQL es el SGBD FLOSS más avanzado y con mejores prestaciones en su administración (Postgresql, 2014); por lo tanto, es necesario definir aquellos métodos y recomendaciones importantes para configurar y hacer uso de sus herramientas y opciones de administración.

Entre los puntos más importantes a tomar en cuenta al administrar nuestro SGBD PostgreSQL tenemos:

- Administrar el Acceso al Servidor, definiendo los parámetros de acceso y conexión.
- Configurar el fichero postgresql.conf, de tal forma que adaptemos nuestro SGB PostgreSQL al tipo de uso que se le va a dar.
- Realizar afinamiento de nuestra base de datos, administrando los recursos para mejorar el rendimiento y la experiencia del usuario al usar PostgreSQL.
- Generar y restaurar backups, utilizando herramientas para generar backups como: pg_dump (backups individuales) y pg_dumpall (backups de toda la instalación); de igual forma realizar restauración utilizando backups generados con herramientas como psql y pg_restore.

2.2. MANEJO DE LA ADMINISTRACIÓN

2.2.1. Instalación de PostgreSQL sobre CentOS 7

CentOS es una distribución perteneciente a la familia de sistemas operativos Red Hat (Red Hat Enterprise Linux, Fedora, Scientific Linux, Oracle Linux, etc.), los cuales vienen con una versión predeterminada de PostgreSQL. En el caso de CentOS, encontramos instalada la versión 9.X de PostgreSQL.

Sin embargo, muchas veces el usuario se ve en la necesidad de usar una versión más actual que la predeterminada en su plataforma; de igual manera, se requiere el uso de herramientas GUI como pgadmin u otras aplicaciones de PostgreSQL.

Pensando en los requerimientos de los usuarios, PostgreSQL provee de un repositorio de paquetes, con todas las versiones soportadas para las distribuciones de plataformas más comunes, al igual que un archivo de instalación grafica para aquellos usuarios con problemas en la instalación vía repositorio.

En esta sección, se detallaran las 3 formas de instalar PostgreSQL en la plataforma CentOS ver 7.0, brindando a los usuarios una guía completa de instalación en distribuciones Red Hat y una fuente de consulta para otras distribuciones Linux.

Se debe tomar en cuenta al momento de instalar, el tipo de servidor que se va a escoger. Para esta guía se instaló la versión GUI de CentOS ver 7, junto a los paquetes de PostgreSQL.

2.2.1.1. Utilizando la versión incluida en la distribución

Utilizando el repositorio incluido en la instalación de la plataforma, hacemos uso del comando Yum¹⁸ para la gestión de paquetes. La versión predeterminada de la plataforma se instalará con este método, si se requiere otros paquetes de aplicaciones de igual manera se usaran los incluidos en el repositorio predeterminado de la distribución.

Implementación

1. Abrimos la terminal de CentOS 7 y entramos como “root”.
2. Usamos el comando YUM para instalar la versión de PostgreSQL contenida en la plataforma, y aceptamos la instalación.

```
yum install postgresql-server
```

3. Para finalizar y por políticas de las distribuciones Red Hat, debemos iniciar manualmente nuestra base de datos.

Comando	Uso
/usr/pgsql-9.3/bin/postgresql93-setup initdb	Inicia la Base de Datos
/bin/systemctl restart postgresql-9.3.service	Reinicia el Servidor de PostgreSQL
/bin/systemctl enable postgresql-9.3.service	Inicio del Servicio al Reiniciar

Tabla 2.2: Lista de Comandos de inicio de PostgreSQL en CentOS 7 (Narváez, 2014).

¹⁸ YUM – Yellowdog Updater Modified

```

[root@localhost jonathan]# /usr/pgsql-9.3/bin/postgresql93-setup initdb
Initializing database ... OK
[root@localhost jonathan]# systemctl start postgresql-9.3.service
[root@localhost jonathan]# systemctl enable postgresql-9.3.service
[root@localhost jonathan]#

```

Grafico 2.1: Comando de Inicio PostgreSQL en CentOS 7 (Narváez, 2014).

```

jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
[jonathan@localhost ~]$ su
Password:
[root@localhost jonathan]# yum install postgresql-server
Loaded plugins: fastestmirror, langpacks
Dependencies Resolved

=====
Package                Arch      Version      Repository    Size
=====
Installing:
 postgresql-server      x86_64    9.2.7-1.el7  base         3.8 M
Installing for dependencies:
 postgresql             x86_64    9.2.7-1.el7  base         2.9 M
 postgresql-libs        x86_64    9.2.7-1.el7  base         229 k

Transaction Summary
=====
Install 1 Package (+2 Dependent packages)

Total download size: 6.9 M
Installed size: 32 M
Is this ok [y/d/N]: █

```

Grafico 2.2: Uso del repositorio predeterminado de CentOS 7 (Narváez, 2014).

Como se puede observar en el Grafico 2.1, la plataforma dispone predeterminadamente, una versión antigua de PostgreSQL.

Se recomienda utilizar las últimas versiones del SGBD PostgreSQL por motivos de seguridad; actualizando el repositorio y evitando el uso de sus versiones predeterminadas de instalación.

2.2.1.2. Utilizando el repositorio Yum actualizado

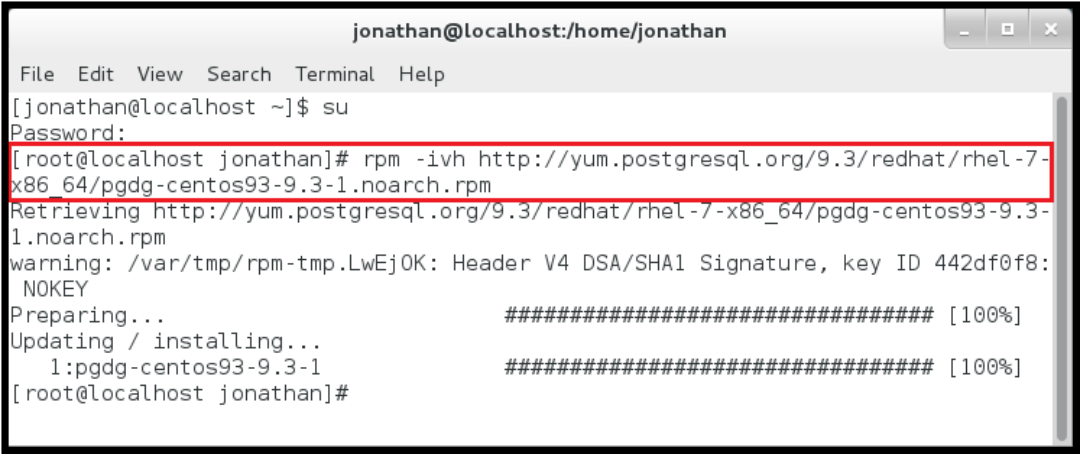
Cuando se requiere instalar la versión actual de PostgreSQL, debemos actualizar el repositorio Yum de la distribución, usando el rpm de la página principal de PostgreSQL.

Implementación

1. Abrimos la terminal de CentOS 7 y entramos como “root”.
2. Descargamos el repositorio de la nueva versión de PostgreSQL (En este caso la versión 9.3), instalando desde el archivo rpm de la página de PostgreSQL. La página para buscar el repositorio deseado es:

<http://yum.postgresql.org/repopackages.php>

```
rpm -ivh http://yum.postgresql.org/9.3/redhat/rhel-7-x86_64/pgdg-centos93-9.3-1.noarch.rpm
```



```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
[jonathan@localhost ~]$ su
Password:
[root@localhost jonathan]# rpm -ivh http://yum.postgresql.org/9.3/redhat/rhel-7-x86_64/pgdg-centos93-9.3-1.noarch.rpm
Retrieving http://yum.postgresql.org/9.3/redhat/rhel-7-x86_64/pgdg-centos93-9.3-1.noarch.rpm
warning: /var/tmp/rpm-tmp.LwEj0K: Header V4 DSA/SHA1 Signature, key ID 442df0f8: NOKEY
Preparing... ##### [100%]
Updating / installing...
  1:pgdg-centos93-9.3-1 ##### [100%]
[root@localhost jonathan]#
```

Grafico 2.3: Descarga y actualización del repositorio de PostgreSQL en CentOS 7 (Narváez, 2014).

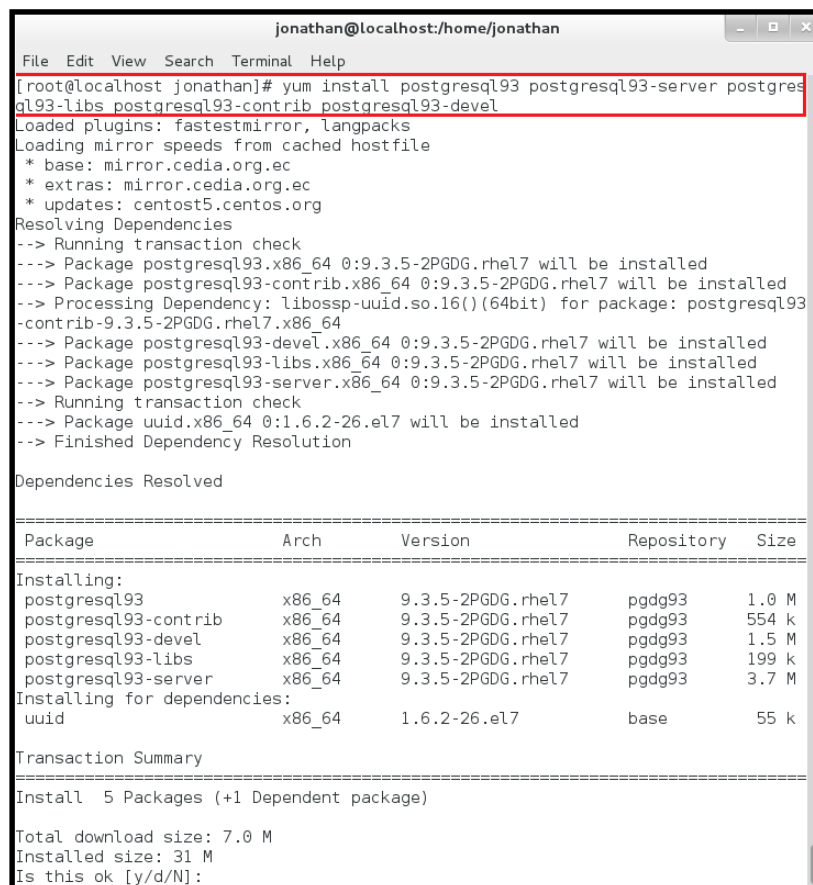
El repositorio contiene varios paquetes destacando:

Paquete	Contenido
postgresql-client	Contiene Librerías
postgresql-server	Servidor de la base de datos
postgresql-contrib	Módulos adicionales
postgresql-devel	Librerías para desarrollo en C
pgadmin3	Herramienta GUI de administración

Tabla 2.2: Paquetes en el repositorio de CentOS (Postgresql, 2014).

3. Instalamos la nueva versión de PostgreSQL y los módulos adicionales y aceptamos la instalación.

```
yum install postgresql93 postgresql93-server postgresql93-libs  
postgresql93-contrib postgresql93-devel
```



```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
[root@localhost jonathan]# yum install postgresql93 postgresql93-server postgresql93-libs postgresql93-contrib postgresql93-devel
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirror.cedia.org.ec
 * extras: mirror.cedia.org.ec
 * updates: centost5.centos.org
Resolving Dependencies
--> Running transaction check
--> Package postgresql93.x86_64 0:9.3.5-2PGDG.rhel7 will be installed
--> Package postgresql93-contrib.x86_64 0:9.3.5-2PGDG.rhel7 will be installed
--> Processing Dependency: libssp-uuid.so.16()(64bit) for package: postgresql93-contrib-9.3.5-2PGDG.rhel7.x86_64
--> Package postgresql93-devel.x86_64 0:9.3.5-2PGDG.rhel7 will be installed
--> Package postgresql93-libs.x86_64 0:9.3.5-2PGDG.rhel7 will be installed
--> Package postgresql93-server.x86_64 0:9.3.5-2PGDG.rhel7 will be installed
--> Running transaction check
--> Package uuid.x86_64 0:1.6.2-26.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====================================================================================================================================
Package                                Arch          Version           Repository        Size
=====================================================================================================================================
Installing:
postgresql93                          x86_64        9.3.5-2PGDG.rhel7 pgdg93            1.0 M
postgresql93-contrib                  x86_64        9.3.5-2PGDG.rhel7 pgdg93            554 k
postgresql93-devel                    x86_64        9.3.5-2PGDG.rhel7 pgdg93            1.5 M
postgresql93-libs                     x86_64        9.3.5-2PGDG.rhel7 pgdg93            199 k
postgresql93-server                   x86_64        9.3.5-2PGDG.rhel7 pgdg93            3.7 M
Installing for dependencies:
uuid                                  x86_64        1.6.2-26.el7      base              55 k
Transaction Summary
-----
Install 5 Packages (+1 Dependent package)

Total download size: 7.0 M
Installed size: 31 M
Is this ok [y/d/N]:
```

Grafico 2.4: Instalación de la versión 9.3 de PostgreSQL en CentOS 7 (Narváez, 2014).

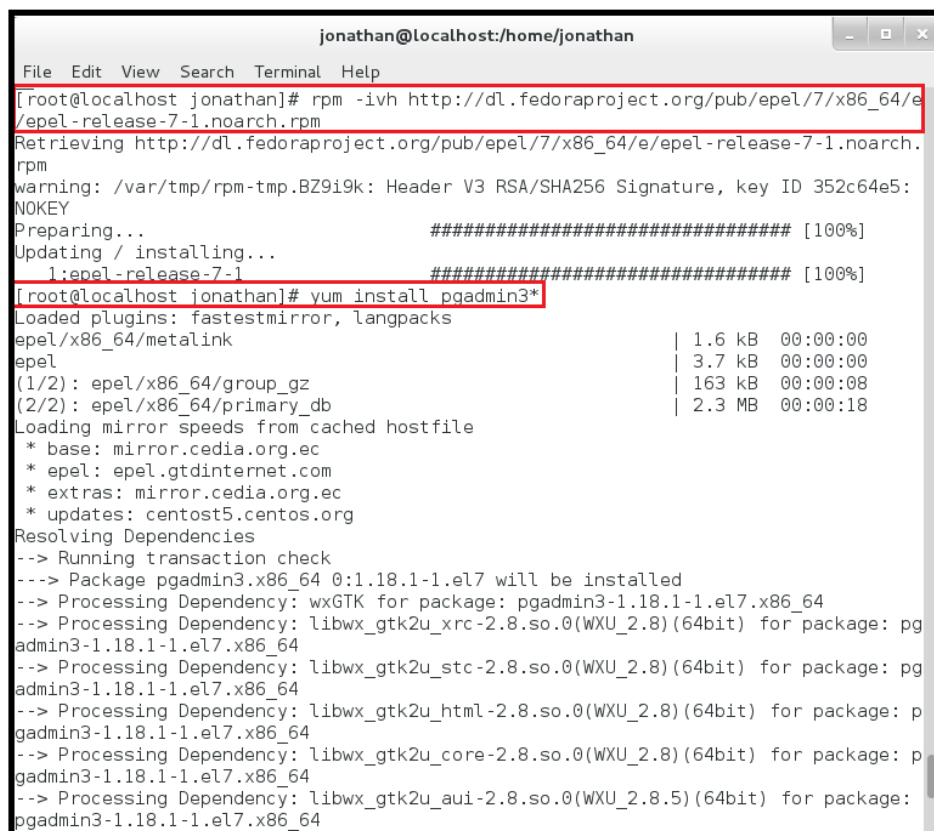
4. Si deseamos instalar pgAdmin3 (solo plataformas Linux con escritorio GNOME), usamos el siguiente comando:

```
yum install pgadmin3*
```

Se pueden presentar problemas en la instalación de pgadmin3 por falta de paquetes para el repositorio Yum, para solucionar este problema usamos el paquete de Fedora EPEL¹⁹ que incluye una serie de paquetes para el repositorio no incluidos al instalar la plataforma. La dirección del repositorio es:

```
http://dl.fedoraproject.org/pub/epel
```

```
rpm -ivh http://dl.fedoraproject.org/pub/epel/7/x86\_64/e/epel-release-7-1.noarch.rpm
```



```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
[root@localhost jonathan]# rpm -ivh http://dl.fedoraproject.org/pub/epel/7/x86\_64/e/epel-release-7-1.noarch.rpm
Retrieving http://dl.fedoraproject.org/pub/epel/7/x86_64/e/epel-release-7-1.noarch.
rpm
warning: /var/tmp/rpm-tmp.BZ9i9k: Header V3 RSA/SHA256 Signature, key ID 352c64e5:
NOKEY
Preparing...                               ##### [100%]
Updating / installing...
 1:epel-release-7-1                         ##### [100%]
[root@localhost jonathan]# yum install pgadmin3*
Loaded plugins: fastestmirror, langpacks
epel/x86_64/metalink                          | 1.6 kB  00:00:00
epel                                          | 3.7 kB  00:00:00
(1/2): epel/x86_64/group_gz                  | 163 kB  00:00:08
(2/2): epel/x86_64/primary_db                | 2.3 MB  00:00:18
Loading mirror speeds from cached hostfile
* base: mirror.cedia.org.ec
* epel: epel.gtdinternet.com
* extras: mirror.cedia.org.ec
* updates: centost5.centos.org
Resolving Dependencies
--> Running transaction check
--> Package pgadmin3.x86_64 0:1.18.1-1.el7 will be installed
--> Processing Dependency: wxGTK for package: pgadmin3-1.18.1-1.el7.x86_64
--> Processing Dependency: libwx_gtk2u_xrc-2.8.so.0(WXU_2.8) (64bit) for package: pg
admin3-1.18.1-1.el7.x86_64
--> Processing Dependency: libwx_gtk2u_stc-2.8.so.0(WXU_2.8) (64bit) for package: pg
admin3-1.18.1-1.el7.x86_64
--> Processing Dependency: libwx_gtk2u_html-2.8.so.0(WXU_2.8) (64bit) for package: p
gadmin3-1.18.1-1.el7.x86_64
--> Processing Dependency: libwx_gtk2u_core-2.8.so.0(WXU_2.8) (64bit) for package: p
gadmin3-1.18.1-1.el7.x86_64
--> Processing Dependency: libwx_gtk2u_aui-2.8.so.0(WXU_2.8.5) (64bit) for package:
pgadmin3-1.18.1-1.el7.x86_64
```

Grafico 2.5: Instalación de pgadmin3 en CentOS 7 (Narváez, 2014).

¹⁹ EPEL – Yellowdog Updater Modified

2.2.1.3. Utilizando el instalador grafico de Linux

Si nuestra plataforma dispone de un entorno de escritorio GNOME, se recomienda el uso del instalador grafico compatible para sistemas Linux; PostgreSQL EnterpriseDB versión Linux, instala el servidor de la base de datos junto a otras aplicaciones graficas como pgadmin3, lo que lo convierte en la mejor opción para plataformas con interfaz gráfica.

Implementación

1. Abrimos la terminal de CentOS ver 7.0 y entramos como “root”.
2. Descargamos el archivo ejecutable de instalación para Linux desde la página de PostgreSQL.
3. Cambiamos el permiso del archivo que descargamos para ser ejecutables y ejecutamos la instalación desde el terminal.

```
chmod +x postgresql-9.3.5-1-linux-x64.run
```

4. Ejecutamos el archivo de instalación desde el terminal.

```
postgresql-9.3.5-1-linux-x64.run
```

5. Debemos tomar en cuenta que la instalación se puede realizar en una diferente carpeta.

Si se dispone de una plataforma Linux con entorno de escritorio GNOME, se recomienda el uso del instalador grafico de PostgreSQL, evitando problemas en la instalación por falta de repositorios.

2.2.2. Puerto de conexión al Servidor.

PostgreSQL usa de forma predeterminada el puerto TCP²⁰ 5432, que sirve para la comunicación con el servidor. Comúnmente se presentan varias instancias de PostgreSQL en un mismo ordenador, gracias a las diferentes versiones manejadas o el uso de aplicaciones que vienen precargadas con el servidor de base de datos.

Al usar varias versiones del servidor de base de datos, se pueden presentar problemas de conexión, producto de conflictos con el acceso al puerto de comunicación (problemas de versionamiento).

Se requiere cambiar el puerto predeterminado cuando existe más de una versión de PostgreSQL, o como norma de seguridad en cualquier ambiente de producción, evitando el mantener el puerto por default.

Implementación

Se cambiará el puerto predeterminado para la versión EnterpriseDB de PostgreSQL.

1. Abrimos el fichero postgresql.conf.

- **Windows:** Abrimos en el explorador la carpeta de nuestra versión de PostgreSQL

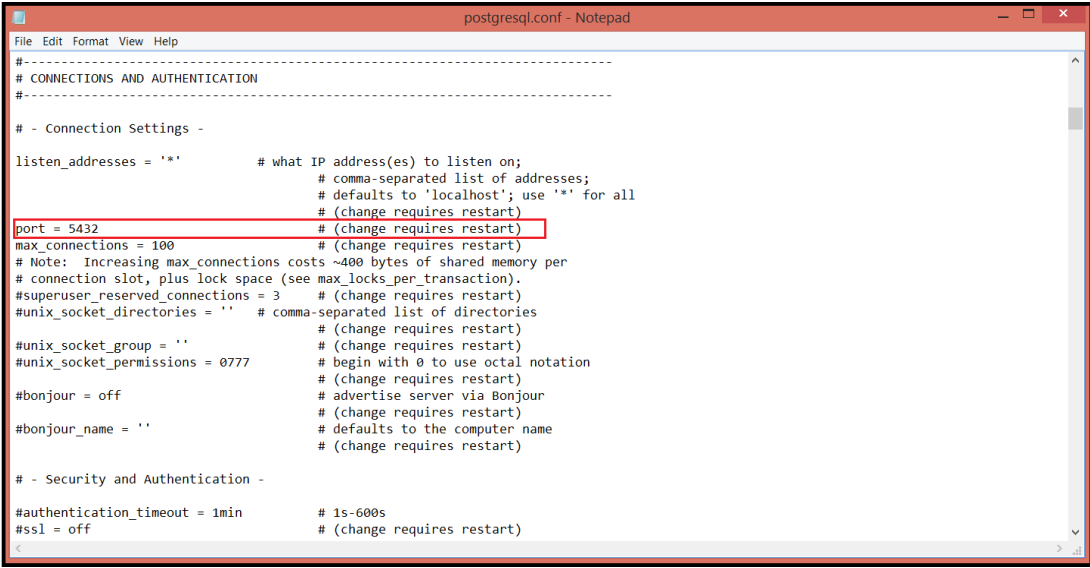
PostgreSQL\<versión>\data\postgresql.conf

- **CentOS:** Abrimos la terminal de CentOS como “root” eh introducimos el comando nano para abrir el fichero en un editor de texto.

²⁰ TCP – Protocolo de Transmisión y Control

```
nano /var/lib/pgsql/9.3/data/postgresql.conf
```

2. Nos dirigimos a la sección de “CONNECTIONS AND AUTHENTICATION”, y cambiamos el valor del puerto <port> a uno determinado por el usuario para luego guardar el fichero. Debemos reiniciar nuestro servidor PostgreSQL.



```
File Edit Format View Help
-----
# CONNECTIONS AND AUTHENTICATION
#-----

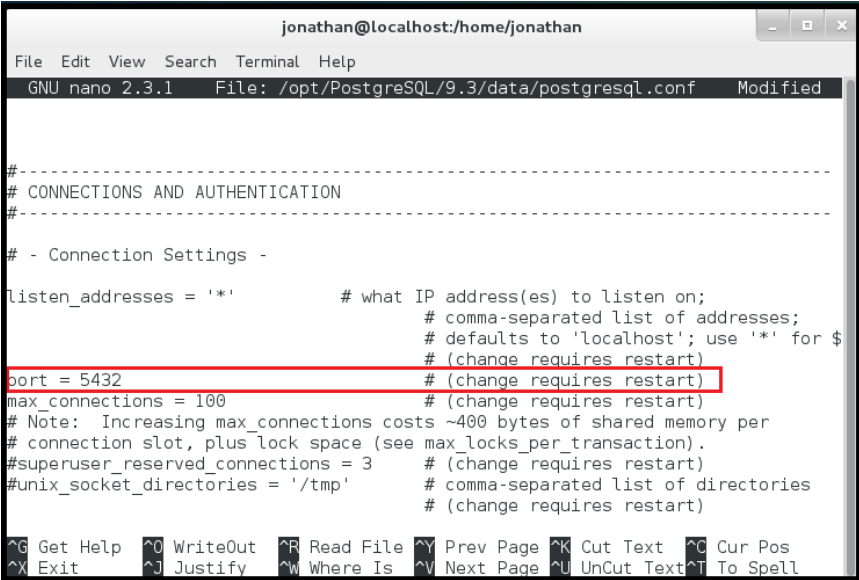
# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
# Note: Increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction).
#superuser_reserved_connections = 3 # (change requires restart)
#unix_socket_directories = ''    # comma-separated list of directories
                                # (change requires restart)
#unix_socket_group = ''         # (change requires restart)
#unix_socket_permissions = 0777 # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off                 # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''             # defaults to the computer name
                                # (change requires restart)

# - Security and Authentication -

#authentication_timeout = 1min  # 1s-600s
#ssl = off                      # (change requires restart)
```

Grafico 2.7: Cambio de puerto PostgreSQL Windows (Narváez, 2014)



```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /opt/PostgreSQL/9.3/data/postgresql.conf Modified

#-----
# CONNECTIONS AND AUTHENTICATION
#-----

# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for $
                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
# Note: Increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction).
#superuser_reserved_connections = 3 # (change requires restart)
#unix_socket_directories = '/tmp' # comma-separated list of directories
                                # (change requires restart)

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^N Next Page ^U UnCut Text ^T To Spell
```

Grafico 2.8: Cambio de puerto PostgreSQL CentOS (Narváez, 2014)

2.2.3. Parámetros de Conexión.

El fichero postgresql.conf posee los parámetros necesarios para configurar la conexión en nuestro SGBD PostgreSQL; de este modo, para una correcta administración se debe tener muy en cuenta las siguientes opciones:

listen_addresses: Especifica las direcciones TCP/IP en las que el servidor acepta conexiones de las aplicaciones del cliente. Entre los valores que dispone tenemos:

- Nombres de hosts y direcciones IP separadas por comas.
- El carácter * corresponde a todas las direcciones IP disponibles.
- 0.0.0.0 acepta todas las direcciones IPv4.
- :: acepta todas las direcciones IPv6.
- Cuando no posee un valor restringe el acceso a todas las direcciones IP.

Por defecto PostgreSQL solo responde a conexiones locales “localhost”, siendo buena práctica, el permitir el acceso a otros sistemas y conexiones a través del estándar TCP/IP usando el parámetro “*”, y administrar quien se conecta a través del fichero pg_hba.conf.

max_connections: Determina el número máximo de conexiones o clientes concurrentes permitidas por PostgreSQL.

- Muy importante e influyente en el rendimiento del servidor.
- Debe ser optimizado si se requiere realizar un afinamiento del rendimiento de las aplicaciones del SGBD.

- PostgreSQL instalado en un buen hardware puede soportar cientos de conexiones (predeterminadamente 100).

Se debe tomar muy en cuenta el parámetro “work_mem” al definir un valor (afinamiento), con valores muy altos se sufrirá un golpe en el rendimiento de la BD; por lo que se recomienda empezar con 100 conexiones e ir aumentando según lo requerido.

authentication_timeout: Tiempo máximo para que el cliente cumpla el proceso de autenticación, o el servidor terminará la conexión

- El tiempo máximo de autenticación por defecto es de 1 minuto.

password_encryption: Parámetro que determina la encriptación de la contraseña de usuario, aun cuando se usan los parámetros “CREATE USER” o “ALTER ROLE” y no se especifica si la contraseña es encriptada o no.

Implementación

1. Abrimos el fichero postgresql.conf.

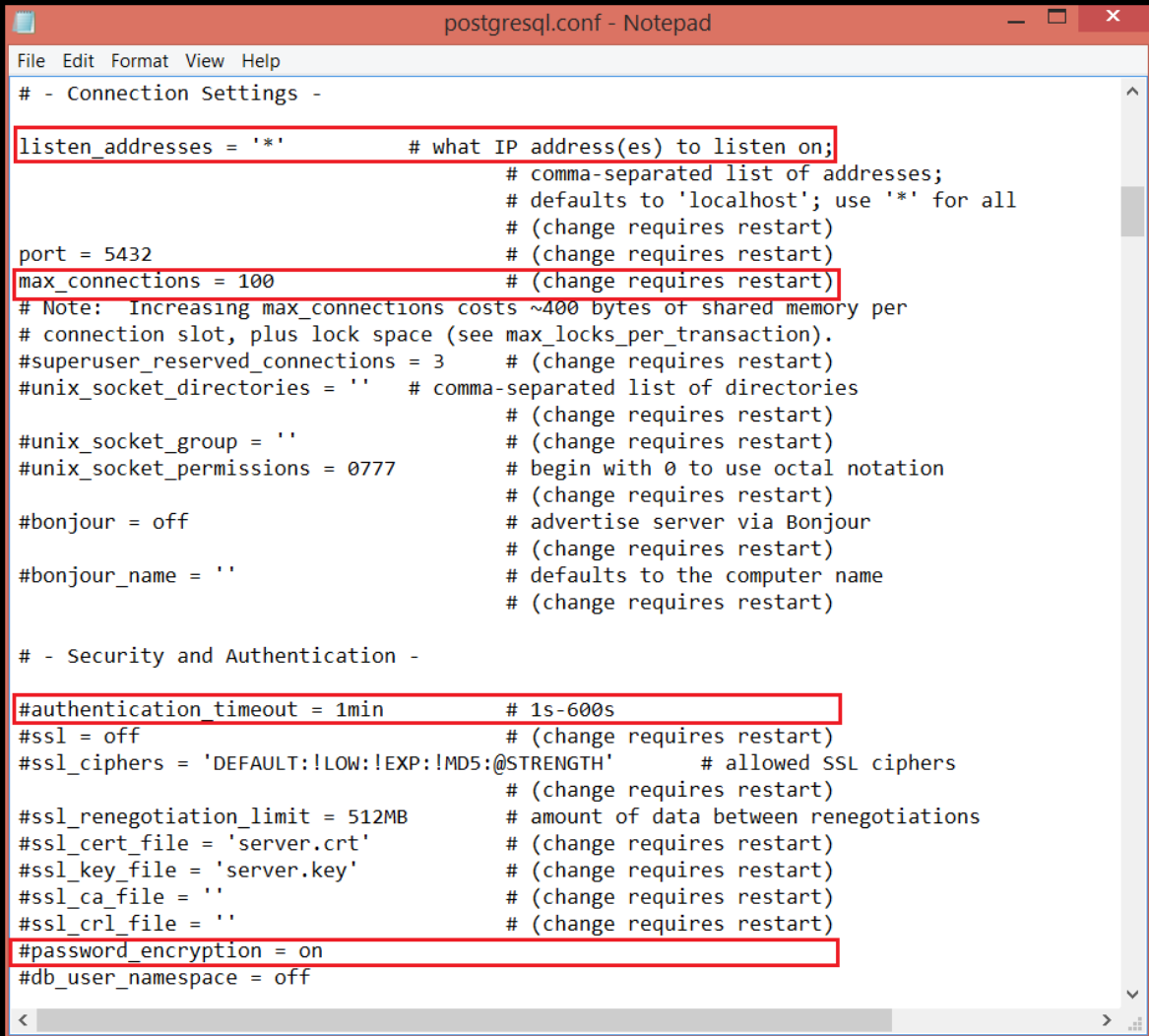
- **Windows:** Abrimos en el explorador la carpeta de nuestra versión de PostgreSQL

PostgreSQL\<versión>\data\postgresql.conf

- **CentOS:** Abrimos la terminal de CentOS como “root” eh introducimos el comando nano para abrir el fichero en un editor de texto.

nano /var/lib/pgsql/9.3/data/postgresql.conf

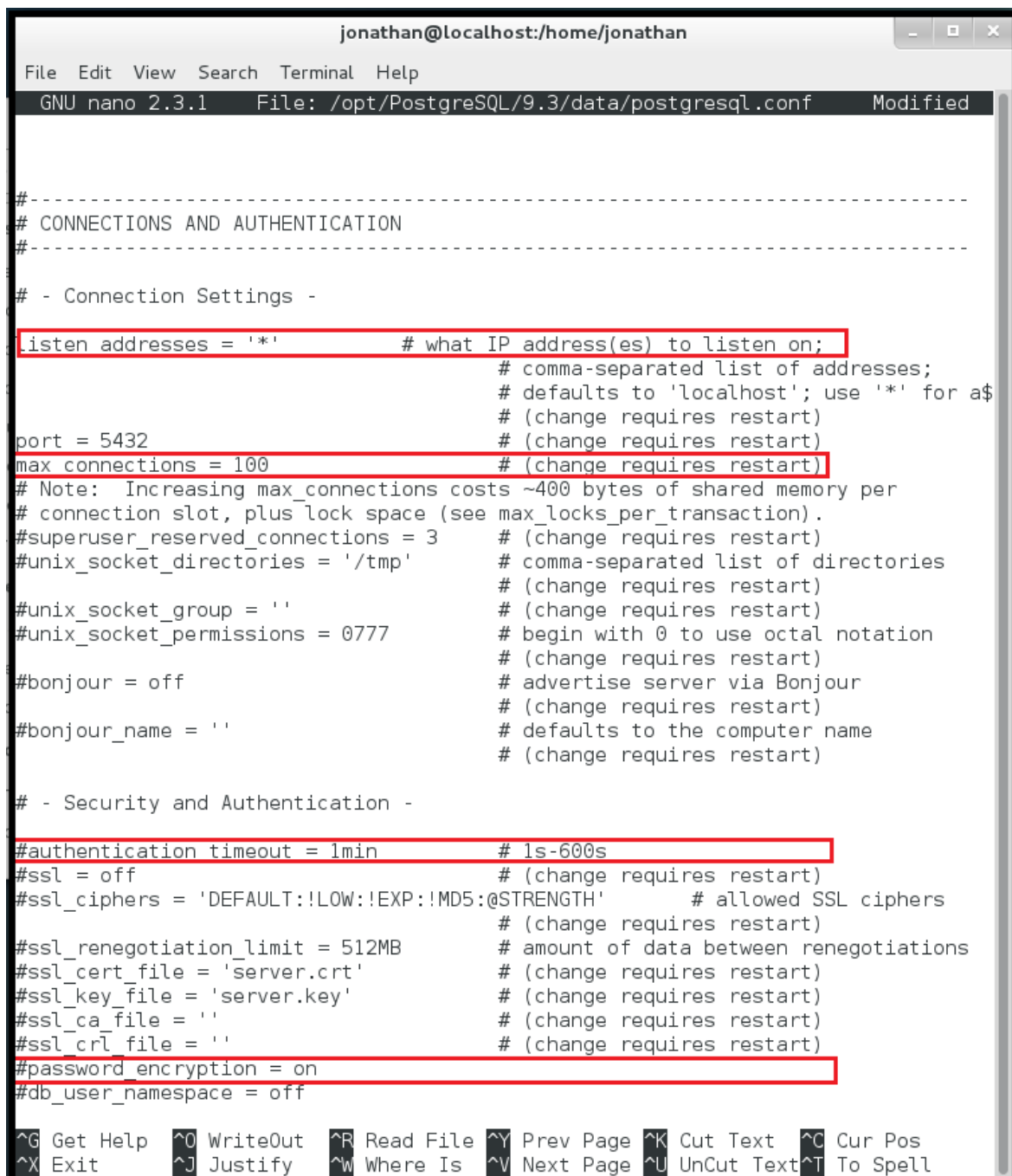
2. Aplicamos las recomendaciones según nuestra configuración y reiniciamos nuestro servidor PostgreSQL.



```
# - Connection Settings -
listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
# Note: Increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction).
#superuser_reserved_connections = 3 # (change requires restart)
#unix_socket_directories = ''    # comma-separated list of directories
                                # (change requires restart)
#unix_socket_group = ''         # (change requires restart)
#unix_socket_permissions = 0777 # begin with 0 to use octal notation
                                # (change requires restart)
#bonjour = off                 # advertise server via Bonjour
                                # (change requires restart)
#bonjour_name = ''             # defaults to the computer name
                                # (change requires restart)

# - Security and Authentication -
#authentication_timeout = 1min  # 1s-600s
#ssl = off                      # (change requires restart)
#ssl_ciphers = 'DEFAULT:!LOW:!EXP:!MD5:@STRENGTH' # allowed SSL ciphers
                                # (change requires restart)
#ssl_renegotiation_limit = 512MB # amount of data between renegotiations
#ssl_cert_file = 'server.crt'   # (change requires restart)
#ssl_key_file = 'server.key'    # (change requires restart)
#ssl_ca_file = ''              # (change requires restart)
#ssl_crl_file = ''             # (change requires restart)
#password_encryption = on
#db_user_namespace = off
```

Grafico 2.9: Configuración conexión y autenticación postgresql.conf en Windows (Narváez, 2014)



```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /opt/PostgreSQL/9.3/data/postgresql.conf Modified

#-----
# CONNECTIONS AND AUTHENTICATION
#-----

# - Connection Settings -

listen_addresses = '*'          # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for all
                                # (change requires restart)
port = 5432                     # (change requires restart)
max_connections = 100           # (change requires restart)
# Note: Increasing max_connections costs ~400 bytes of shared memory per
# connection slot, plus lock space (see max_locks_per_transaction).
#superuser_reserved_connections = 3 # (change requires restart)
#unix_socket_directories = '/tmp'  # comma-separated list of directories
                                   # (change requires restart)
#unix_socket_group = ''           # (change requires restart)
#unix_socket_permissions = 0777  # begin with 0 to use octal notation
                                   # (change requires restart)
#bonjour = off                   # advertise server via Bonjour
                                   # (change requires restart)
#bonjour_name = ''               # defaults to the computer name
                                   # (change requires restart)

# - Security and Authentication -

#authentication_timeout = 1min    # 1s-600s
#ssl = off                       # (change requires restart)
#ssl_ciphers = 'DEFAULT:!LOW:!EXP:!MD5:@STRENGTH' # allowed SSL ciphers
                                   # (change requires restart)
#ssl_renegotiation_limit = 512MB  # amount of data between renegotiations
#ssl_cert_file = 'server.crt'    # (change requires restart)
#ssl_key_file = 'server.key'     # (change requires restart)
#ssl_ca_file = ''                # (change requires restart)
#ssl_crl_file = ''               # (change requires restart)
#password_encryption = on
#db_user_namespace = off

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Grafico 2.10: Configuración conexión y autenticación postgresql.conf en CentOS (Narváez, 2014)

2.2.4. Afinamiento.

Las actividades de negocio y su productividad en muchas organizaciones, se encuentran condicionadas por la disponibilidad y tiempo de respuesta requerido de la información almacenada en una base de datos. Por tal razón, se requiere un afinamiento u optimización de la base de datos y las aplicaciones que se maneja.

El proceso de afinamiento consiste en mejorar el rendimiento de una aplicación a través del aprovechamiento de sus recursos, tanto en memoria, uso de la red de comunicación y acceso al disco duro. Generando optimización en aspectos como: mejor tiempo de respuesta, uso de recursos, reducción de costos y proveer una mejor experiencia al usuario.

Para realizar un correcto afinamiento a nuestro motor PostgreSQL se debe tomar en cuenta varios aspectos.

2.2.4.1. Sistema de Discos

Uno de los puntos a tomar muy en cuenta si se desea mejorar el rendimiento de PostgreSQL, se encuentra en el sistema de discos sólidos donde el SGBD se encuentra operando. Este puede generar un gran cuello de botella entre las peticiones realizadas a la base de datos con la lectura y escritura en el sistema de discos.

Las soluciones a tomar sobre estos problemas no se limitan al aumento de memoria del servidor, encontrando soluciones más eficaces como:

Utilizar la configuración RAID²¹ nivel 10 para mejorar la redundancia y rendimiento del sistema de discos. Esto es considerado como la practica más recomendable.

Implementación

1. Si se posee el presupuesto se debe adquirir un mínimo de 4 discos independientes.
2. Dividir a cada disco en bloques.
3. Configurar WAL (\$PGDATADIR/pg_xlog) en un disco aparte
4. Almacenar en discos separados los datos, índices y archivos temporales. (Casanova, 2014)

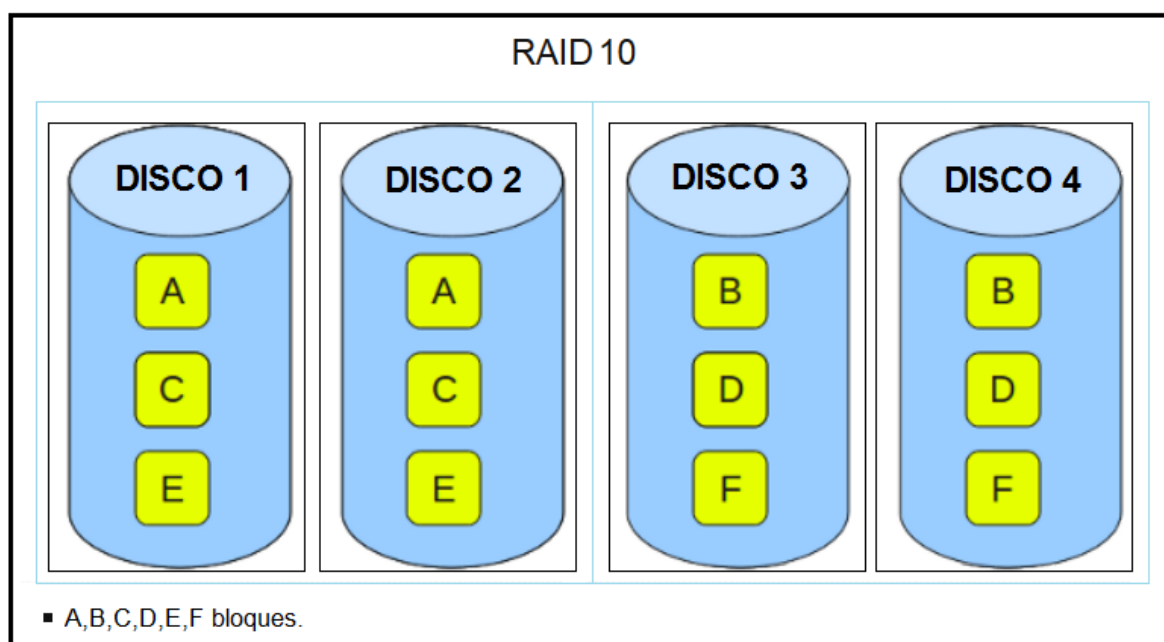


Grafico 2.11: Configuración sistema de discos RAID nivel 10 (Narváez, 2014)

²¹ RAID – Conjunto Redundante de Discos Independientes

2.2.4.2. Configuración de parámetros de rendimiento.

Ciertos parámetros en el archivo postgresql.conf pueden optimizar el rendimiento de la base de datos y permitir afinar nuestro sistema PostgreSQL.

shared_buffers: Determina el monto de memoria dedicada que puede usar PostgreSQL, para mantener los datos en cache.

- Normalmente se trabaja con 8 KB por cada shared buffer.
- El valor predeterminado es de 128 MB = 16.384 shared buffers.

Un alto valor en este parámetro puede llegar a producir un fuerte impacto en el rendimiento del servidor, por lo que se recomienda empezar con un 25% del total de la memoria RAM y no superar el 1/3 del total de memoria.

4 GB RAM = 1.024 MB Como valor inicial = 131.072 shared buffers

work_mem: Determina el monto de memoria que usa PostgreSQL para operaciones de ordenamiento y tablas hash.

- El valor del parámetro es aplicado por cada operación (si una consulta realiza varias operaciones el valor de memoria se multiplicara por ese número de operaciones).

Para asignar el valor de `work_mem`, debemos tomar en cuenta la memoria total de sistema. Si es superior a los 20 GB se debería asignar un máximo de 2 GB, de otra forma un 2% o 10 % del total de la memoria (64 MB como buen valor para empezar).

4 GB RAM = 82 MB Como valor inicial = 2%

max_fsm_pages y max_fsm_relations: Almacenan la información del espacio libre reutilizable debido a la presencia de tuplas muertas en la base de datos.

- Estos parámetros se registran durante VACUUM.
- El valor de `max_fm_relations` debe ser igual al número de tablas e índices existentes en todas las base de datos de la instalación de PostgreSQL.

Implementación

1. Abrimos el fichero `postgresql.conf`.

- **Windows:** Abrimos en el explorador la carpeta de nuestra versión de PostgreSQL

PostgreSQL\<versión>\data\postgresql.conf

- **CentOS:** Abrimos la terminal de CentOS como “root” eh introducimos el comando `nano` para abrir el fichero en un editor de texto.

`nano /var/lib/pgsql/9.3/data/postgresql.conf`

2. Aplicamos las recomendaciones según nuestra configuración y reiniciamos nuestro servidor PostgreSQL.

```
File Edit Format View Help
# RESOURCE USAGE (except WAL)
#-----

# - Memory -

shared buffers = 1024MB          # min 128kB
                                # (change requires restart)
#temp_buffers = 8MB             # min 800kB
#max_prepared_transactions = 0   # zero disables the feature
                                # (change requires restart)
# Note: Increasing max_prepared_transactions costs ~600 bytes of shared memory
# per transaction slot, plus lock space (see max_locks_per_transaction).
# It is not advisable to set max_prepared_transactions nonzero unless you
# actively intend to use prepared transactions.
work mem = 82MB                  # min 64kB
#maintenance_work_mem = 16MB     # min 1MB
#max_stack_depth = 2MB           # min 100kB
```

Grafico 2.12: Configuración parámetros de rendimiento fichero postgresql.conf en Windows (Narváez, 2014)

```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /opt/PostgreSQL/9.3/data/postgresql.conf Modified

# RESOURCE USAGE (except WAL)
#-----

# - Memory -

shared buffers = 1024MB          # min 128kB
                                # (change requires restart)
#temp_buffers = 8MB             # min 800kB
#max_prepared_transactions = 0   # zero disables the feature
                                # (change requires restart)
# Note: Increasing max_prepared_transactions costs ~600 bytes of shared memory
# per transaction slot, plus lock space (see max_locks_per_transaction).
# It is not advisable to set max_prepared_transactions nonzero unless you
# actively intend to use prepared transactions.
work mem = 82MB                  # min 64kB
#maintenance_work_mem = 16MB     # min 1MB
#max_stack_depth = 2MB           # min 100kB

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

Grafico 2.13: Configuración parámetros de rendimiento fichero postgresql.conf en CentOS (Narváez, 2014)

2.2.4.3. Recursos en la escritura sobre el disco.

Para reducir los recursos empleados en la escritura sobre el disco duro, sobre todo al momento de transaccionalidad se puede recurrir a los siguientes parámetros:

fsync y synchronous_commit: Se determina antes de terminar la transacción, si se debe grabar todas las páginas del WAL²² (Todas las modificaciones son escritas en el log antes de ser aplicadas).

- No se recomienda configurar en “off” el parámetro fsync ya que puede causar corrupción de datos ante una eventual caída del servidor, por eso es mejor configurar el parámetro synchronous_commit en “on”, lo que generaría el mismo beneficio sin riesgo a corromper los datos.

commit_delay, commit_siblings: Son parámetros destinados para intentar de ejecutar COMMIT simultáneamente en varias transacciones vigentes.

- De existir operaciones activas cuando la transacción realiza un COMMIT, el servidor espera el tiempo del parámetro commit_delay hasta que las operaciones commit_siblings terminen.

checkpoints_segments: Parámetro que indica a todos los segmentos WAL a ser llenados.

- Un segmento es un archivo WAL de 16 MB.

²² WAL – Write-ahead logging – Registro de escritura anticipada

Es una buena práctica el aumentar el número de `checkpoint_segments` en operaciones de carga pesada como migración (Se recomienda un valor de 64 y en operaciones con gran cantidad de datos un valor de 200), al hacer esto obtendremos temporalmente mayor rapidez al cargar los datos.

Esto ocurre cuando se maneja grandes cantidades de datos en PostgreSQL, lo que causa que los checkpoints ocurran con mayor frecuencia que en operaciones normales.

checkpoints_timeout: El parámetro indica el tiempo en segundos que ha transcurrido desde el último checkpoint.

checkpoints_completion_target: Parámetro que indica la proporción de tiempo que se empleara en la ejecución del checkpoint en el tiempo de espera al siguiente.

- Se recomienda un valor de 0.9. (Postgresql, 2014)

Implementación

1. Abrimos el fichero `postgresql.conf`.

- **Windows:** Abrimos en el explorador la carpeta de nuestra versión de PostgreSQL

```
PostgreSQL\<versión>\data\postgresql.conf
```

- **CentOS:** Abrimos la terminal de CentOS como “root” e introducimos el comando `nano` para abrir el fichero en un editor de texto.

```
nano /opt/PostgreSQL/<versión>/data/postgresql.conf
```

2. Aplicamos las recomendaciones según nuestra configuración y reiniciamos nuestro servidor PostgreSQL.

Grafico 2.5: Configuración parámetros recursos de escritura sobre disco duro fichero postgresql.conf en Windows (Narváez, 2014)

```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /opt/PostgreSQL/9.3/data/postgresql.conf Modified

#-----
# WRITE AHEAD LOG
#-----

# - Settings -

#wal_level = minimal          # minimal, archive, or hot_standby
                              # (archive requires restart)
fsync = on                    # turns forced synchronization on or off
synchronous_commit = on      # synchronization level;
                              # off, local, remote_write, or on
                              # the default is the first option
                              # supported by the operating system:
                              #   open_datasync
                              #   fdatasync (default on Linux)
                              #   fsync
                              #   fsync_writethrough
                              #   open_sync
#wal_sync_method = fsync
#full_page_writes = on       # recover from partial page writes
#wal_buffers = -1             # min 32kB, -1 sets based on shared_buff$
                              # (change requires restart)
#wal_writer_delay = 200ms     # 1-10000 milliseconds

commit_delay = 0              # range 0-100000, in microseconds
commit_siblings = 5           # range 1-1000

# - Checkpoints -

checkpoint_segments = 64      # in logfile segments, min 1, 16MB each
checkpoint_timeout = 5min     # range 30s-1h
checkpoint_completion_target = 0.9 # checkpoint target duration, 0.0 - 1.0
checkpoint_warning = 30s      # 0 disables

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Grafico 2.14: Configuración parámetros recursos de escritura sobre disco duro fichero postgresql.conf en CentOS (Narváez, 2014)

2.2.5. VACUUM y ANALYZE.

VACUUM y ANALYZE son herramientas encargadas de la limpieza y análisis de nuestra base de datos PostgreSQL. Ambas pueden ser implementadas a la vez siendo la opción más recomendable.

VACUUM se encarga de limpiar el espacio en disco ocupado por tuplas “muertas”; normalmente en PostgreSQL aquellas tuplas que son eliminadas o se encuentran obsoletas por acción de una operación UPDATE, permanecen presentes en disco hasta que se realice VACUUM. Por lo que se recomienda realizar VACUUM periódicamente en nuestra base de datos.

Con ANALYZE se recolecta y guardan los datos e información del contenido de las tablas de una base de datos de PostgreSQL en el fichero pg_statistic. El planeador de consultas de PostgreSQL hace uso de esta información para determinar con mayor facilidad el plan de ejecución más eficiente para las consultas.

Cuando usamos VACUUM y ANALYZE, PostgreSQL realiza primero VACUUM y después ANALYZE en cada una de las tablas seleccionadas. Esta es una combinación útil para generar scripts de mantenimiento rutinario.

Se recomienda a aquellas bases de datos con un alto volumen de consultas y actualizaciones, utilizar frecuentemente VACUUM para eliminar tuplas muertas.

Parámetros

- **FULL:** Con este parámetro, VACUUM copia todos los contenidos de una tabla en un nuevo archivo en disco sin el espacio a reutilizar, permitiendo que el espacio no utilizado regresa al sistema operativo.

Este proceso aparte de ser más lento y requerir espacio extra en disco, bloquea exclusivamente a la tabla procesada.
- **FREEZE:** Selecciona una tabla de la base de datos con una marca de tiempo transaccional, que indica a PostgreSQL que aquella tabla no necesita VACUUM.
- **VERBOSE:** Genera un reporte detallado de la actividad de VACUUM por cada tabla.
- **ANALYZE:** Actualiza las estadísticas utilizadas por el planificador para determinar la manera más eficiente de ejecutar una consulta.
- **table_name:** El nombre especifica de la tabla a realizar VACUUM.
- **column_name:** El nombre específico de la columna a aplicar ANALYZE.

No se recomienda utilizar continuamente VACUUM FULL por la alta demanda de recursos y espacio en disco.

Sintaxis

VACUUM	FULL	FREEZE	VERBOSE	TABLA		
VACUUM	FULL	FREEZE	VERBOSE	ANALYZE	TABLA	COLUMNA

Tabla 2.2: Sintaxis VACUUM (Narváez, 2014)

Implementación

Se realizará un ejemplo de VACUUM ANALYZE sobre toda la base de datos, tomando en cuenta las recomendaciones planteadas, y usando la herramienta pgadmin3.

Los pasos son exactamente los mismos en Windows y CentOS.

1. Abrimos pgadmin3 y realizamos conexión con el servidor.
2. Nos direccionamos sobre el esquema de la base de datos.
3. Seleccionamos la pestaña <Tools>, opción <Maintenance>.
4. Configuramos los parámetros según lo requerido (VACUUM ANALYZE VERBOSE).

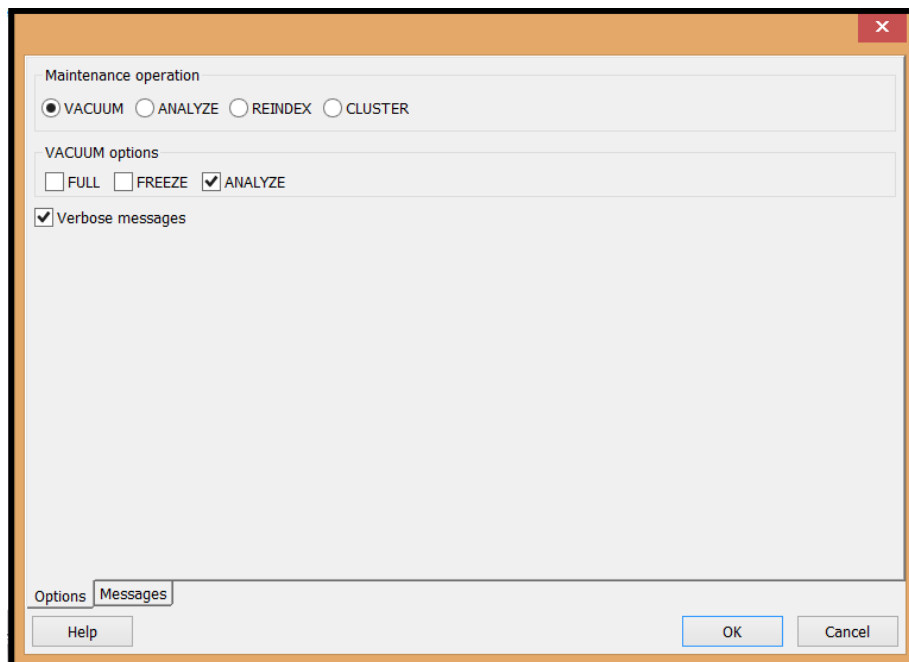


Grafico 2.15: Configuración parámetros de mantenimiento pgadmin3 (Narváez, 2014)

5. Visualizamos el reporte.

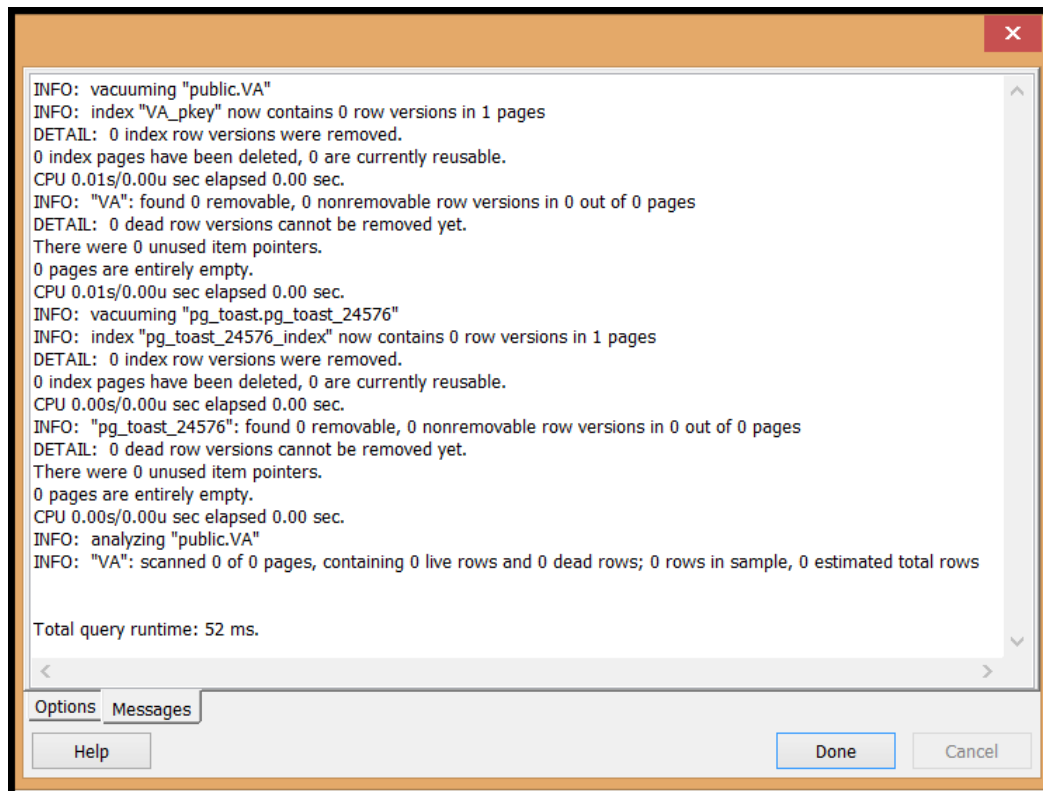


Grafico 2.16: Reporte VACUUM ANALYZE pgadmin3 (Narváez, 2014)

2.2.5.1. AUTOVACUUM.

Desde PostgreSQL 8.1 existe la opción llamada autovacuum daemon, cuyo propósito es la automática ejecución de los comandos VACUUM y ANALYZE. Autovacuum daemon se ejecuta periódicamente revisando todas las tablas que tengan un gran número de tuplas muertas.

Parámetros

Estos son los parámetros a configurar en el fichero `postgresql.conf` cuando se desea la ejecución periódica de `autovacuum daemon`.

- **autovacuum:** Parámetro que indica al servidor que se requiere ejecutar `autovacuum daemon`. Es necesario que las opciones de `stats_start_collector` y `stats_row_level` estén activadas.
- **autovacuum_naptime:** Indica el tiempo de espera entre cada operación `autovacuum` (`VACUUM` y `ANALYZE`). Se recomienda un tiempo de 1 minuto.
- **autovacuum_vacuum_threshold:** Especifica el mínimo número de tuplas muertas que se necesitan sobre una tabla, para activar `VACUUM`. El valor recomendado es de 500.
- **autovacuum_analyze_threshold:** Especifica el mínimo número de tuplas muertas que se necesitan sobre una tabla, para activar `ANALYZE`. El valor recomendado es de 250.
- **autovacuum_vacuum_scale_factor:** Define el tamaño en fracción de la tabla para agregar a `autovacuum_vacuum_threshold` cuando se activa `VACUUM`. Se recomienda un 20% del tamaño de la tabla.
- **autovacuum_analyze_scale_factor:** Define el tamaño en fracción de la tabla para agregar a `autovacuum_analyze_threshold` cuando se activa `ANALYZE`. Se recomienda un 10% del tamaño de la tabla.

La respuesta a los problemas de tiempo y uso de recursos de VACUUM es la aplicación de autovacuum; mientras mayor número de operaciones VACUUM realicemos, las siguientes operaciones tendrán que realizar una menor limpieza y como resultado, un menor consumo de recursos.

Implementación

Se activará la opción de autovacuum daemon en Windows y CentOS, en ambas plataformas se configuraran los mismos parámetros, de acuerdo a las recomendaciones.

1. Abrimos el fichero postgresql.conf.

- **Windows:** Abrimos en el explorador la carpeta de nuestra versión de PostgreSQL

```
PostgreSQL\<versión>\data\postgresql.conf
```

- **CentOS:** Abrimos la terminal de CentOS como “root” eh introducimos el comando nano para abrir el fichero en un editor de texto.

```
nano /var/lib/pgsql/9.3/data/postgresql.conf
```

2. Activamos autovacuum teniendo en cuenta las recomendaciones del tema.
3. Guardamos y reiniciamos nuestro servidor PostgreSQL.

```
File Edit Format View Help
# AUTOVACUUM PARAMETERS
#-----
autovacuum = on           # Enable autovacuum subprocess? 'on'
                           # requires track_counts to also be on.
#log_autovacuum_min_duration = -1  # -1 disables, 0 logs all actions and
                           # their durations, > 0 logs only
                           # actions running at least this number
                           # of milliseconds.
#autovacuum_max_workers = 3        # max number of autovacuum subprocesses
                           # (change requires restart)
autovacuum_naptime = 1min         # time between autovacuum runs
autovacuum_vacuum_threshold = 500 # min number of row updates before
                           # vacuum
autovacuum_analyze_threshold = 250 # min number of row updates before
                           # analyze
autovacuum_vacuum_scale_factor = 0.2 # fraction of table size before vacuum
autovacuum_analyze_scale_factor = 0.1 # fraction of table size before analyze
#autovacuum_freeze_max_age = 200000000 # maximum XID age before forced vacuum
                           # (change requires restart)
#autovacuum_multixact_freeze_max_age = 400000000 # maximum Multixact age
                           # before forced vacuum
                           # (change requires restart)
#autovacuum_vacuum_cost_delay = 20ms # default vacuum cost delay for
                           # autovacuum, in milliseconds;
                           # -1 means use vacuum_cost_delay
#autovacuum_vacuum_cost_limit = -1  # default vacuum cost limit for
                           # autovacuum, -1 means use
                           # vacuum_cost_limit
```

Grafico 2.17: Configuración autovacuum en Windows (Narváez, 2014)

```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /var/lib/pgsql/9.3/data/postgresql.conf Modified
#-----
# AUTOVACUUM PARAMETERS
#-----
autovacuum = on           # Enable autovacuum subprocess? 'on'
                           # requires track_counts to also be on.
#log_autovacuum_min_duration = -1  # -1 disables, 0 logs all actions and
                           # their durations, > 0 logs only
                           # actions running at least this number
                           # of milliseconds.
#autovacuum_max_workers = 3        # max number of autovacuum subprocesses
                           # (change requires restart)
autovacuum_naptime = 1min         # time between autovacuum runs
autovacuum_vacuum_threshold = 500 # min number of row updates before
                           # vacuum
autovacuum_analyze_threshold = 250 # min number of row updates before
                           # analyze
autovacuum_vacuum_scale_factor = 0.2 # fraction of table size before vacuum
autovacuum_analyze_scale_factor = 0.1 # fraction of table size before analyze
#autovacuum_freeze_max_age = 200000000 # maximum XID age before forced vacuum
                           # (change requires restart)
#autovacuum_multixact_freeze_max_age = 400000000 # maximum Multixact age
                           # before forced vacuum
                           # (change requires restart)
#autovacuum_vacuum_cost_delay = 20ms # default vacuum cost delay for
                           # autovacuum, in milliseconds;
                           # -1 means use vacuum_cost_delay
#autovacuum_vacuum_cost_limit = -1  # default vacuum cost limit for
                           # autovacuum, -1 means use
                           # vacuum_cost_limit

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Grafico 2.17: Configuración autovacuum en CentOS (Narváez, 2014)

2.2.6. Backups

PostgreSQL provee de las herramientas para realizar copias de seguridad o backups del contenido de sus bases de datos. Para realizar esta acción, se recurre a herramientas como `pg_dump` para backups individuales y `pg_dumpall` para backups de toda la instalación.

Backups individuales.

Para realizar el backup de una sola base de datos se puede usar la herramienta `pg_dump`, la cual almacena el contenido de todos los objetos de base de datos en un solo archivo.

Backups totales.

Existen dos formas de realizar backups de todas las bases de datos de PostgreSQL.

- La primera opción involucra generar backups secuenciales de todas las bases de datos de nuestro servidor.
- La segunda opción es utilizar `pg_dumpall` que permite realizar backups de todas las bases de datos, pero al mismo tiempo.

No se recomienda el uso de `pg_dumpall`, ya que con este método, la información de las bases de datos es almacenada en un solo fichero, lo que no permite realizar una restauración en paralelo y como resultado, un gran consumo de tiempo en la restauración.

Se recomienda realizar backups de la configuración de los ficheros `postgresql.conf` y `pg_hba.conf`. El método más óptimo y simple es crear una copia completa de cada fichero.

Implementación

Se realizará un backups de una base de datos de PostgreSQL, utilizando la herramienta pg_dump contenida en pgadmin3.

Los pasos son exactamente los mismos en Windows y CentOS.

1. Abrimos pgadmin3 y realizamos conexión con el servidor.
2. Nos direccionamos sobre el esquema de la BD y pulsamos click derecho.
3. Seleccionamos la opción “Backup”.
4. Especificamos las opciones que se requieran (nombre, formato del archivo, propiedades dump, objetos).
5. Realizamos el backup de nuestra BD.

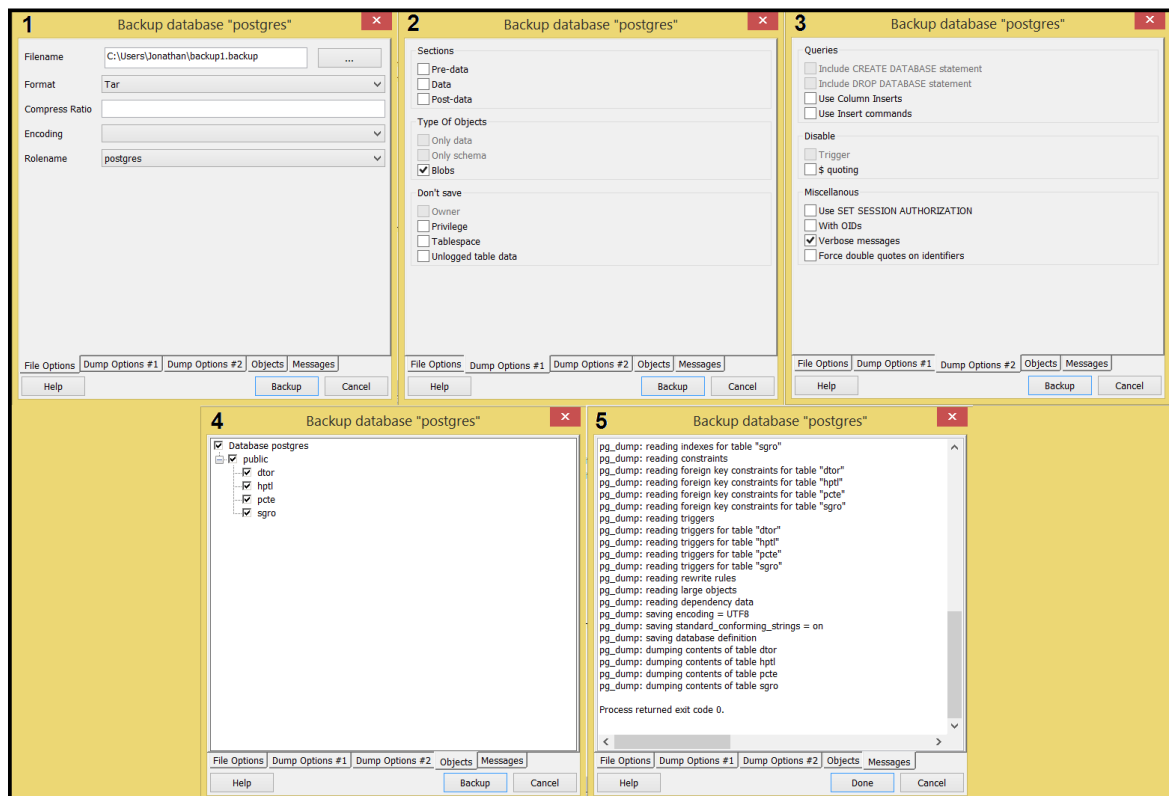


Grafico 2.18: Pasos para realizar un backup en pgadmin3 (Narváez, 2014)

2.2.7. Restauración de BD.

En PostgreSQL utilizamos las herramientas `psql` y `pg_restore` para restaurar una base de datos a través de las copias de seguridad o backups previamente realizados.

Psql.

`Psql` permite restaurar un script sql generado ya sea por `pg_dump`, `pg_dumpall`, o cualquier otra herramienta que genere backups compatibles. Con `psql` es necesario ejecutar el script completo.

Pg_restore.

Con `pg_restore` podemos restaurar de igual manera, aquellos backups generados por `pg_dump`, `pg_dumpall`, o cualquier otra herramienta que genere backups compatibles. No se requiere que sean específicamente scripts sql, también acepta otros tipos de archivos como “tar”.

- Permite realizar restauración paralela, especificando el número de tablas a restaurar.
- Permite especificar que objetos de la base de datos serán restaurados del fichero que contiene el backup de la base de datos.
- Puede restaurar backups de antiguas versiones de la base de datos.

En el proceso de restauración de una base de datos, se recomienda incrementar el valor del parámetro `maintenance_work` y desactivar replicación dentro del fichero de configuración `postgresql.conf`. Esto es muy necesario sobretodo en restauraciones de varias bases de datos y al usar `pg_restore`.

Implementación

Realizaremos la restauración de un backup de PostgreSQL generado con pg_dump, utilizando la herramienta pg_restore contenida en pgadmin3.

Los pasos son exactamente los mismos en Windows y CentOS.

1. Abrimos pgadmin3 y realizamos conexión con el servidor.
2. Si se dispone de una versión antigua de la BD, hacer DROP de la misma.
3. Crear una nueva BD en blanco y seleccionamos la opción “Restore”.
4. Especificamos las opciones que se requieran (nombre del backup, opciones de restauración).
5. Aceptar para realizar la restauración.

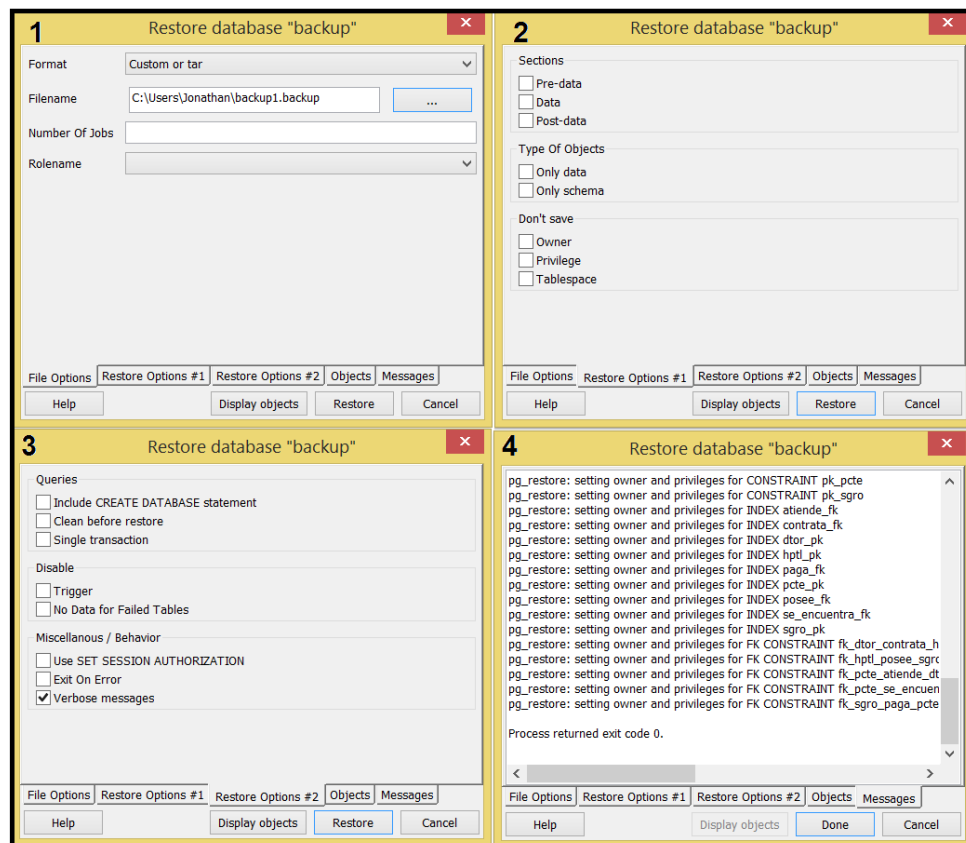


Grafico 2.19: Pasos para realizar un backup en pgadmin3 (Narváez, 2014)

2.2.8. Conexión al Servidor.

Usando la herramienta pgAdmin3 se realizará la conexión al servidor tomando en cuenta los parámetros básicos más importantes:

- **Name:** El nombre será la identificación del servidor en el árbol del pgAdmin3.
- **Host:** La dirección IP²³ de la maquina a contactar o el nombre de un dominio reconocido. Para conexiones remotas se usa la dirección de IP del servidor de la BD, y para conexiones locales el domino "localhost" o la IP "127.0.0.1".
- **Port:** El numero el puerto definido para la versión de PostgreSQL asignada.
- **Maintenance DB:** La base de datos inicial a la que se conectara pgadmin3, la que contiene librerías opcionales como “adminpack” (Herramientas de acceso y visualización de la BD así como edición remota de los archivos “postgresql.conf” y “pg_hba.conf”).

Desde la versión PostgreSQL 8.1 se debe hacer uso de la BD de mantenimiento llamada “postgres”, frente a “template1” (usada en versiones antiguas) para evitar saturación.

- **Username:** El nombre del usuario para el acceso al servidor.
- **Password:** Contraseña del usuario seleccionado, si se marca la opción “Store Password” esta se almacenará en el archivo “postgresqlpgpass.conf”.

²³ IP – Protocolo de Internet

Implementación

Realizaremos conexión con un nuevo servidor en nuestro SGBD PostgreSQL, utilizando las recomendaciones planteadas en el tema.

Los pasos son exactamente los mismos en Windows y CentOS.

1. Iniciamos pgAdmin3 y seleccionamos el icono para nueva conexión con el servidor.

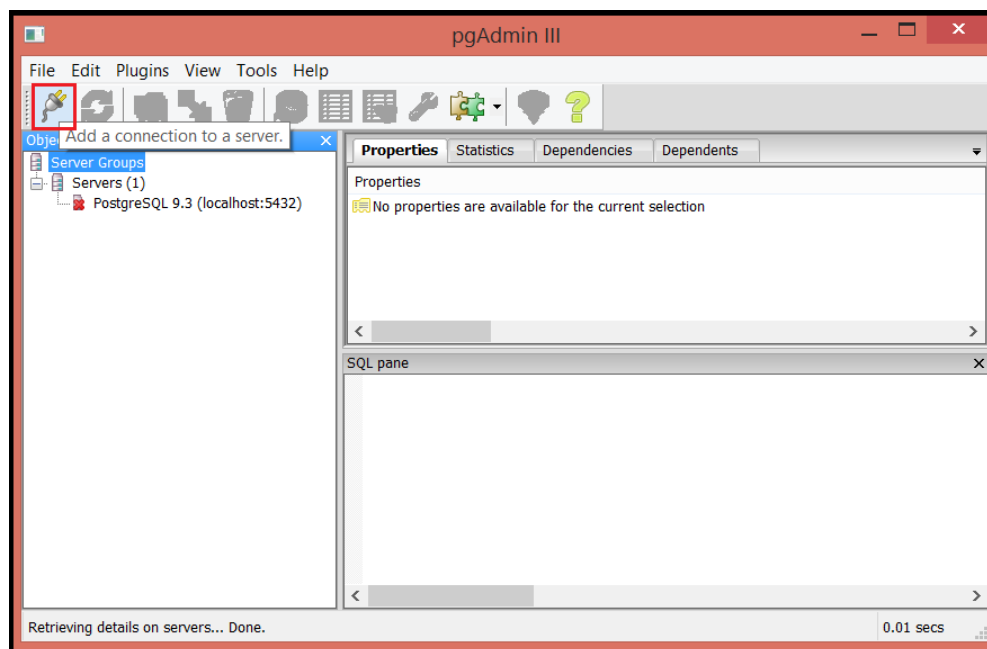


Grafico 2.20: Interfaz pgAdmin3 e Icono de conexión al servidor (Narváez, 2014)

2. Se insertarán los parámetros básicos en la interfaz de configuración de conexión.

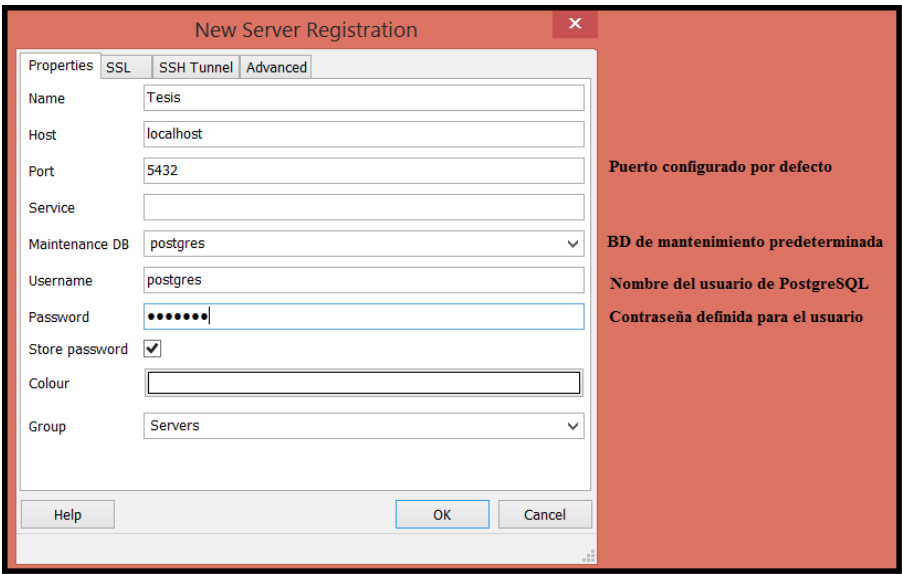


Grafico 2.21: Configuración de la conexión al servidor pgAdmin3 (Narváez, 2014)

3. Terminada la configuración de parámetros básico se pulsa el botón <OK> para realizar la nueva conexión.

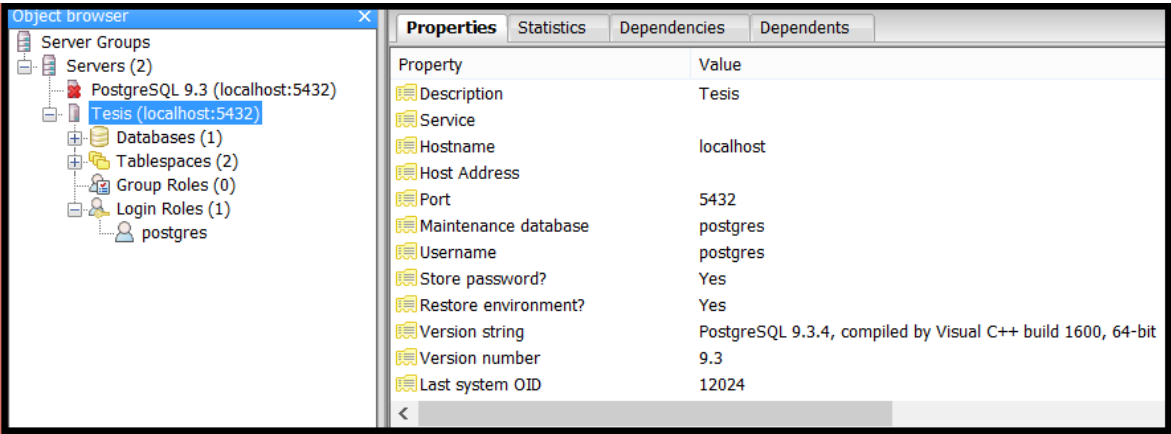


Grafico 2.22: Vista principal de la nueva conexión (Narváez, 2014)

2.3. CUADRO DESCRIPTIVO

El cuadro descriptivo que recopila todas las buenas prácticas administrativas obtenidas del capítulo es el siguiente:

N°	Practica	Descripción	Categoría	Prioridad
1	Utilizar las últimas versiones de PostgreSQL	Se recomienda utilizar las últimas versiones del SGBD PostgreSQL por motivos de seguridad.	Instalación PostgreSQL	ALTA
2	Utilizar el instalador cuando se dispone de una plataforma con escritorio	Si se dispone de una plataforma con entorno de escritorio, se recomienda el uso del instalador grafico de PostgreSQL.	Instalación PostgreSQL	BAJA
3	Cambio del puerto de acceso	Se debe cambiar el puerto de acceso predeterminado cuando existe más de una versión de PostgreSQL y por motivos de seguridad en ambientes en producción.	Conexión de PostgreSQL	ALTA
4	Permitir Acceso de otras conexión vía TCP/IP	Es recomendable permitir el acceso a otros sistemas vía el estándar TCP/IP, con el valor “*” en el parámetro listen_addresses.	Conexión y Autentificación	MEDIA
5	Definir un valor en max_connections acorde al rendimiento.	Es necesario definir un valor para el parámetro max_connections pensando en el parámetro work_mem y el rendimiento del sistema. Comenzar con 100 conexiones e ir aumentando según lo requerido.	Conexión y Autentificación - Afinamiento	ALTA
6	Utilizar configuración RAID nivel 10	Si se posee del capital necesario, se recomienda utilizar la configuración RAID nivel 10, para mejorar la redundancia y rendimiento del sistema de discos.	Afinamiento	MEDIA - BAJA
7	Definir un valor en shared_buffers acorde al rendimiento	Configurar el parámetro shared_buffers de acuerdo al rendimiento del servidor. Se recomienda de un 25% a 1/3 del total de la memoria del sistema.	Afinamiento	ALTA
8	Definir un valor en work_mem acorde al parámetro max_connections	Configurar el parámetro work_mem con un 2% a un 10% del total de la memoria del sistema.	Afinamiento	MEDIA
9	Aumentar el número de checkpoint_segme	Se recomienda aumentar el número de checkpoint_segments en operaciones de carga pesada como	Afinamiento	MEDIA - BAJA

	nts en operaciones de carga pesada	migración, de esta forma se obtendrá temporalmente mayor rapidez al cargar los datos.		
10	Usar VACUUM en BDs con alto volumen de consultas y actualizaciones	Es recomendable usar VACUUM en aquellas bases de datos con un alto volumen de consultas y actualizaciones, para eliminar tuplas muertas.	VACUUM y ANALYZE	ALTA
11	Usar autovacuum para solucionar problemas de VACUUM	El uso de autovacuum es recomendable cuando se tiene problemas de rendimiento al utilizar VACUUM, debido a la necesidad de limpiar constantemente el número de tuplas muertas.	VACUUM y ANALYZE	ALTA – MEDIA
12	Utilizar pg_dump como herramientas para realizar backups sobre pg_dumpall	Se recomienda el uso de pg_dump sobre pg_dumpall para realizar backups totales, ya que el último no permite realizar restauración en paralelo y con ello un gran consumo de tiempo y recursos.	Backups	BAJA
13	Realizar Backups de los ficheros postgresql.conf y ph_hba.conf	Es recomendable realizar backups de la configuración los ficheros postgresql.conf y pg_hba.conf.	Backups	MEDIA
14	Incrementar el valor de maintenance_work y desactivar replicación para restaurar varios backups	Es necesario aumentar valor del parámetro maintenance_work y desactivar replicación dentro del fichero de configuración postgresql.conf, sobre todo al restaurar varias bases de datos.	Restauración	MEDIA
15	Hacer uso de la base de datos de mantenimiento “postgres” frente a “template1”	Se recomienda usar la base de datos de mantenimiento “postgres” frente a “template1” para evitar saturación.	Conexión con el Servidor	BAJA

Tabla 2.3: Cuadro Descriptivo buenas practicas administración PostgreSQL (Narváez, 2014)

CAPÍTULO III: BUENAS PRÁCTICAS DE SEGURIDAD

3.1. SEGURIDAD EN POSTGRESQL

Uno de los aspectos donde sobresale PostgreSQL sobre la mayoría de los SGBD convencionales, es el énfasis en los mecanismos y reglas que posee para incrementar la seguridad e integridad de la información.

Podemos destacar los siguientes puntos que maneja PostgreSQL al tratar la seguridad e integridad de la información almacenada en sus bases de datos:

- Acceso de los usuarios, autenticando y concediendo permisos de accesos a los usuarios de PostgreSQL.
- Privilegios y permisos, manejando la administración del acceso, junto a los permisos concedidos a los diferentes usuarios a través de sus roles.
- Conexión, con el registro que define el tipo de acceso, usuarios admitidos, direcciones de conexión y la información que se encuentra disponible para estos accesos.
- Diseño de datos, manejando la jerarquía lógica y física de PostgreSQL, que permite repartir permisos según la instancia física o el esquema lógico de los datos.

3.2. MANEJO DE LA SEGURIDAD

3.2.1. Cambio de contraseña usuario postgres

El usuario postgres es aquel usuario que viene por defecto en PostgreSQL y se configura al momento de su instalación. Este usuario posee permisos de acceso a todos los servicios del SGBD, y es el usuario con mayor prioridad en su protección; para llevar a cabo esta tarea, uno de las principales recomendaciones es cambiar la contraseña de acceso por una contraseña segura y que tal práctica sea habitual.

Implementación

Se realizará el cambio de la contraseña del usuario postgres usando la herramienta pgAdmin3, se requiere conocer la contraseña a cambiar de dicho usuario.

Los pasos son exactamente los mismos en Windows y CentOS.

1. Abrimos pgAdmin3 y realizamos conexión con el servidor como usuario “postgres”.
2. Una vez realizada la conexión, revisamos el buscador de objetos <Object browser> y nos dirigimos a la opción roles de inicio de sesión <Login Roles>.
3. Seleccionamos el usuario postgres en el árbol de roles de inicio de sesión.

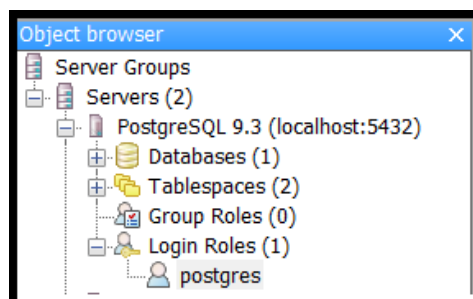


Grafico 3.1: Usuario postgres dentro del buscador de objetos (Narváez, 2014)

4. Seleccionamos las propiedades del usuario y accedemos a la pestaña de definición <Definition>.

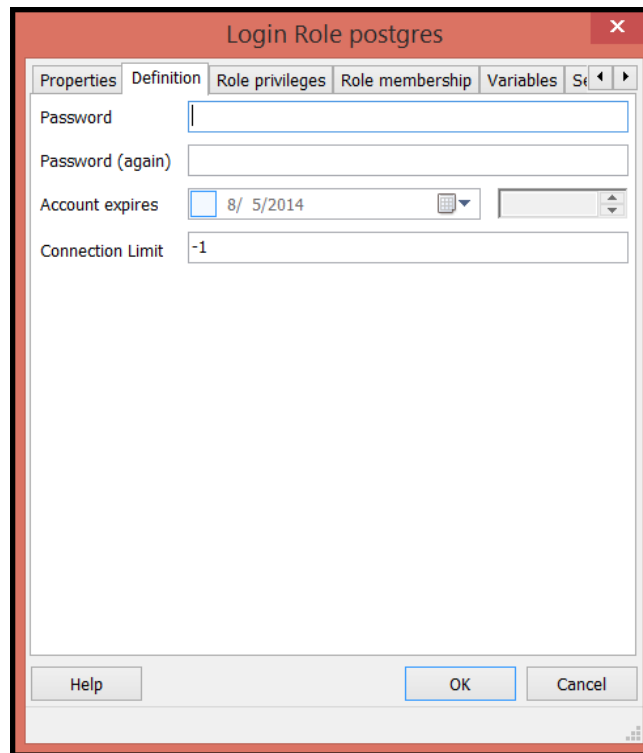


Grafico 3.2: Interfaz cambio de contraseña (Narváez, 2014)

5. Introducimos la nueva contraseña en la parte de <Password> y la reconfirmamos.

Es una buena práctica aun en ambientes de trabajo “seguros” la creación de una contraseña de acceso al servidor que cumpla con características como: un mínimo de 6 caracteres, no use palabras que sean fáciles de adivinar (sustantivos, nombres), maneje mayúsculas, minúsculas, números y sea actualizada en un periodo de 19 días.

3.2.2. Usuarios y Privilegios

La administración se basa en usuarios y privilegios, con la creación de un objeto dentro de PostgreSQL donde el dueño de tal objeto es el propio usuario que lo creo. Existen privilegios de acceso para limitar la lectura, escritura y formulación de reglas a las clases, otorgados con las sentencias grant y revoke.

PostgreSQL utiliza MVCC²⁴ para manejar los accesos a la base de datos, lo que permite a los usuarios realizar consultas simultáneas sin esperar una cola de procesos, agilizando la ejecución sin descuidar la seguridad.

Los permisos que son condiciones a las acciones que realiza el usuario al acceder a la base de datos de otro usuario. El propietario del objeto y el súper usuario pueden establecer permisos al objeto, los permisos se componen de nombre de usuario o grupo y derechos.

Un rol de usuario es a su vez una construcción de la base de datos, que posee una lista de privilegios de seguridad que son aplicados sobre el usuario que se encuentre conectado en esa instancia. Cada rol está disponible a nivel de cluster, permitiendo ser accesible por todas las instancias de PostgreSQL.

Un usuario de PostgreSQL es un rol que posee el permiso CONNECT asociado a él.

²⁴ MVCC - Multi-Version Concurrency Control

Privilegio	Descripción
SELECT	Derecho para leer los datos del objeto.
INSERT	Derecho para insertar datos en el objeto.
UPDATE	Derecho para cambiar los datos en el objeto.
DELETE	Derecho para eliminar los datos del objeto.
TRUNCATE	Derecho para vaciar una tabla o un set de tablas.
CREATE	Derecho para crear nuevos objetos dentro de otro objeto
CONNECT	Derecho para el usuario de conectarse a una específica BD
RULE	Derecho para crear una regla sobre las tablas.
REFERENCES	Derecho para crear una clave externa a una tabla.
TRIGGER	Derecho para crear un desencadenador en la tabla.
TEMPORARY	Derecho para crear una tabla temporal.
EXECUTE	Derecho para ejecutar la función.
USAGE	Derecho para usar la lengua del procedimiento.
ALL	Privilegios adecuados para las tablas.

Tabla 3.1: Derechos otorgados por un permiso (Orlando Corro, 2014).

Se recomienda que a cada usuario de la BD, se le asigne aquella aplicación que este registra, permitiendo la separación de privilegios por sesión iniciada.

Los objetos de la BD como: bases de datos, tablas y esquemas; deben pertenecer a un rol que no acostumbre a modificar los datos de esos objetos. Aquellos usuarios que acostumbran a usar comandos como INSERT, UPDATE, DELETE, SELECT no deben poseer tales objetos, por motivo de integridad de la información.

Es recomendable asignar el tipo de rol de usuario según el acceso a las aplicaciones de la BD, no se debe asignar un role de login a un usuario que usa una base de datos individual.

Implementación

Utilizando pgadmin3, se procederá a crear un rol de inicio de sesión y un rol de grupo, de esta forma podremos visualizar los diferentes permisos y parámetros de configuración de cada uno. Los pasos son exactamente los mismos en Windows y CentOS.

1. Abrimos pgAdmin3 y realizamos conexión con el servidor como usuario “postgres”.
2. Revisamos el esquema de la conexión y nos dirigimos a las dos últimas opciones <Group Roles> y <Login Roles>.
- Para crear un rol de grupo, seleccionamos <Group Roles> seguido de la opción <New Group Role>; e introducimos los parámetros en cada pestaña de la interfaz (nombre del rol de usuario, contraseña, fecha de caducidad, privilegios, pertenencias, etc.)

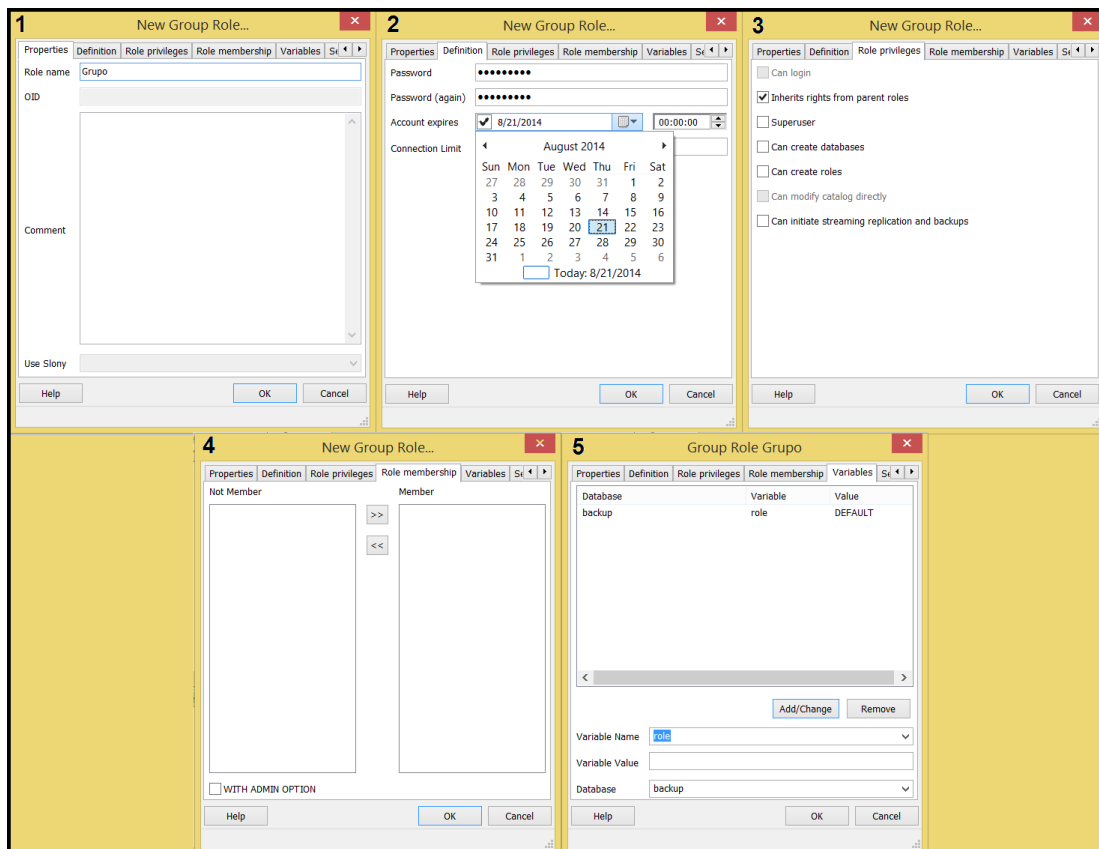


Grafico 3.3: Creación Rol de Grupo pgadmin3 (Narváez, 2014)

- De igual forma para la creación de un rol de inicio de sesión, seleccionamos <Login Roles> seguido de la opción <New Login Role>; introducimos los parámetros necesarios tomando en cuenta el tipo de rol que se va a crear.

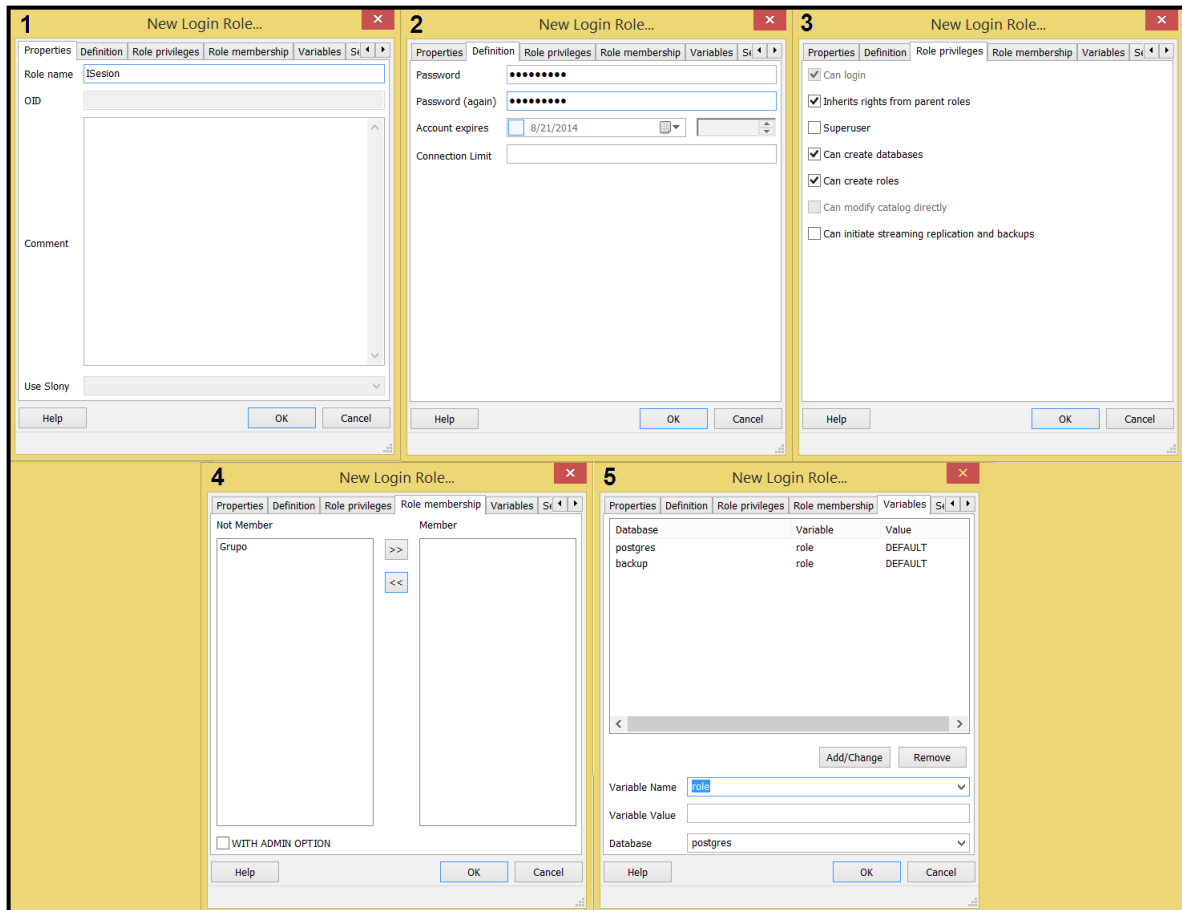


Grafico 3.4: Creación Rol de Inicio de Sesión pgadmin3 (Narváez, 2014)

3.2.3. Seguridad en el fichero pg_hba.conf

La autenticación de cada cliente que accede a una base de datos, se encuentra controlada por el fichero pg_hba.conf almacenado en el directorio de PostgreSQL. Este archivo se utiliza para definir como, donde y desde que sitio un usuario puede acceder a los servicios de PostgreSQL.

Pg_hba.conf controla aspectos como cuales “hosts” tienen permiso de conexión, como se autentifica cada cliente, que nombre de usuario en PostgreSQL pueden utilizar y a cual base de datos pueden acceder.

El formato de registros es el siguiente:

local	BD	Usuario	-	Método
host	BD	Usuario	IP	Método
hostss1	BD	Usuario	IP	Método
hostnoss1	BD	Usuario	IP	Método

Tabla 3.2: Formato de registros pg_hba.conf (Narváez, 2014)

Se requiere añadir líneas en el registro por cada host de la aplicación en uso. Si tenemos problemas de visualización, podemos aplicar reglas de sub-red para organizar el registro de mejor forma.

Tipo de conexión

Las condiciones de la conexión que utiliza los parámetros definidos.

- **local:** Usa conexión de dominio Unix o IPC²⁵ permitiendo que dos procesos distintos referencien y abran la misma conexión con el fin de comunicarse.
- **host:** Son permitidas conexiones utilizando el protocolo TCP/IP.
- **hostssl:** Conexiones que usen el protocolo TCP/IP y utilicen encriptación SSL²⁶.
- **hostnoss:** Conexiones que usen el protocolo TCP/IP, y no acepten encriptación SSL.

Base de Datos: El nombre de la(s) bases de datos que coincide en el registro.

- **all:** Todas las bases de datos son aceptadas.
- **sameuser:** La BD es aceptada si tiene el mismo nombre que el usuario en el registro.
- **samerole:** El usuario debe ser miembro de un rol que tenga el mismo nombre que la BD del registro.
- **replication:** El parámetro indica el registro de la conexión donde se requiere replicación de la(s) bases de datos.

Usuario: Especifica cual nombre de usuario dentro de la base de datos coincide en el registro.

- **all:** Todos los usuarios son aceptados.

²⁵ IPC - Socket de Comunicación Interprocesos

²⁶ SSL - Capa de Conexión Segura

Evitar el uso del parámetro all, permitiendo el acceso al servidor a aquellos usuarios considerados como necesarios; de esta forma, podemos controlar de mejor manera el acceso a nuestro servidor PostgreSQL.

Dirección: Describe la dirección del cliente que coincide en el registro, este campo puede contener un nombre de host o un rango de dirección IP.

- Podemos usar una máscara CIDR²⁷ siendo un integer (entre 0 y 32 para IPv4 o 125 para IPv6) que especifique el número de bits significativos en la máscara.
- Un nombre de host que comienza con un punto <.> coincide con el sufijo del nombre de host real.
- **samehost:** Coincida con cualquiera de las direcciones IP del servidor.
- **Samenet:** Coincida con cualquiera de las direcciones en cualquier subred en la que el servidor se encuentre conectado directamente.

Tipo	BD	Usuario	IP/Mascara		Método
host	all	all	192.168.0.1	255.255.255.255	md5
host	all	all	192.168.0.1/32		md5

Tabla 3.3: IP y mascara en el registro (Narváez, 2014)

Método: Especifica el método de autenticación que se utilizará cuando una conexión coincide con este registro.

²⁷ CIDR - Enrutamiento entre Dominios sin Clases

- **trust:** Conexión incondicional. Permite a cualquier persona que se conecte al servidor de base de datos el poder iniciar sesión como cualquier usuario de PostgreSQL, sin necesidad de una contraseña.
- **reject:** Rechaza conexión incondicional. Es un método útil para "filtrar" determinados hosts de un grupo y bloquear su conexión.
- **md5:** Requiere que el usuario proporcione una contraseña con encriptación MD5 para su autenticación.
- **password:** Requiere que el usuario proporcione una contraseña no encriptada para su autenticación.

Se recomienda desactivar en todos los registros el parámetro de autenticación “trust”.

Para mejorar la seguridad de acceso, es recomendable utilizar el método de autenticación encriptada md5 en todos los registros, sobre todo en los registros tipo host.

Tipo	BD	Usuario	IP	Método
host	all	all	192.168.1.0/32	md5

Tabla 3.4: Modo host con método md5 (Narváez, 2014)

Se recomienda añadir la siguiente línea al final del registro:

Tipo	BD	Usuario	IP	Método
host	all	all	0.0.0.0	reject

Tabla 3.5: Línea de registro pg_hba.conf excepciones (Narváez, 2014)

Pg_hba.conf se procesa de arriba hacia abajo, por lo que de esta forma se rechaza y cubre todo el tráfico no definido que solicite acceso al servidor.

Implementación

Se implementara las recomendaciones planteadas en el registro del fichero pg_hba.conf.

1. Abrimos el fichero postgresql.conf.

- **Windows:** Abrimos en el explorador la carpeta de nuestra versión de PostgreSQL

```
PostgreSQL\<versión>\data\pg_hba.conf
```

- **CentOS:** Abrimos la terminal de CentOS como “root” eh introducimos el comando nano para abrir el fichero en un editor de texto.

```
nano /var/lib/pgsql/9.3/data/pg_hba.conf
```

2. Aplicamos las recomendaciones según nuestra configuración.

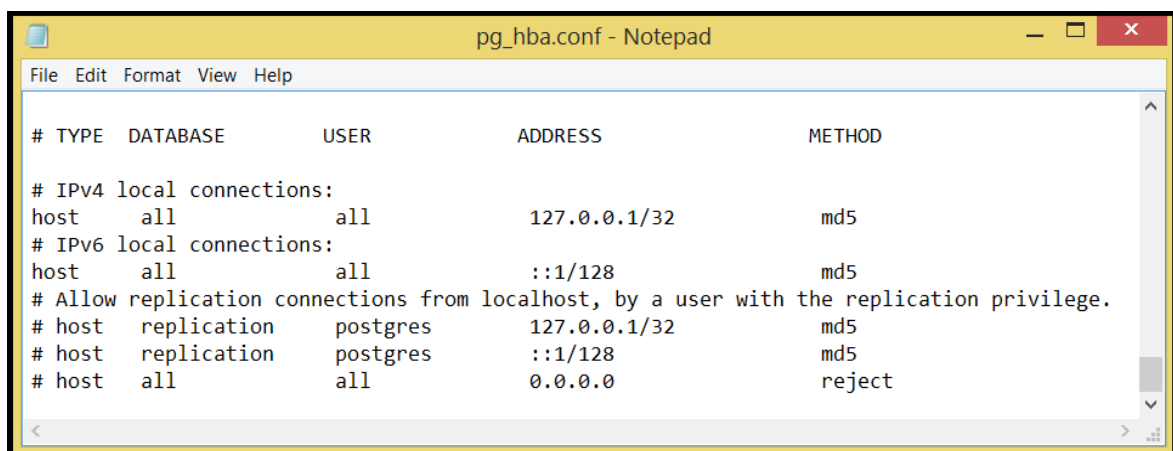


Grafico 3.5: Configuración pg_hba.conf en Windows (Narváez, 2014)

```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /var/lib/pgsql/9.3/data/pg_hba.conf Modified

# "host" records. In that case you will also need to make PostgreSQL
# listen on a non-local interface via the listen_addresses
# configuration parameter, or via the -i or -h command line switches.


# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all md5
# IPv4 local connections:
host all all 127.0.0.1/32 md5
# IPv6 local connections:
host all all ::1/128 md5
# Allow replication connections from localhost, by a user with the
# replication privilege.
#local replication postgres md5
#host replication postgres 127.0.0.1/32 md5
#host replication postgres ::1/128 md5
#host all all 0.0.0.0 reject

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Grafico 3.5: Configuración pg_hba.conf en CentOS (Narváez, 2014)

3.2.4. Diseño de Datos

3.2.4.1. Separación física

La estructura jerárquica de PostgreSQL donde parte la conexión con el servidor. Cada una de las diferentes bases de datos existentes en la jerarquía, corresponde a una instancia “física” dentro de PostgreSQL.

Cuando un usuario inicia sesión en una instancia PostgreSQL (dependiendo de sus privilegios y aplicaciones asignadas), se encontrará hasta que termine esa sesión en una específica base de datos solo teniendo acceso a sus datos.

Si el usuario requiere acceder a los datos y tablas de otra instancia física de base de datos, deberá desconectarse de esa instancia y acceder a otra vía inicio de sesión. Cada conexión debe ser controlada en el fichero `pg_hba.conf`.

Implementación

Utilizando `pgadmin3`, visualizaremos la separación física de las instancias en PostgreSQL.

Los pasos son exactamente los mismos en Windows y CentOS.

1. Abrimos `pgAdmin3`.
2. Realizamos conexión con el servidor deseado, para el ejemplo se usará el usuario “postgres”.
3. Visualizamos 3 instancias físicas de base de datos, la primera llamada “B1”, la segunda “B2” y por último la base de datos template postgres.

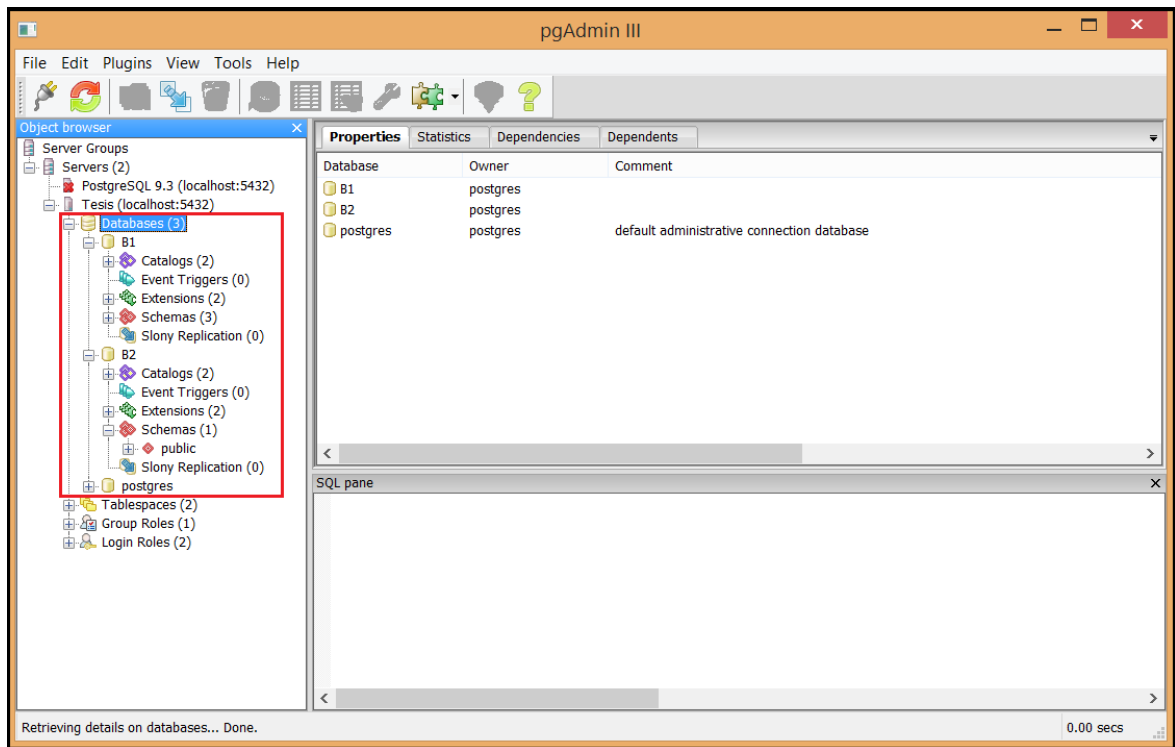


Grafico 3.7: Separación física instancias BD PostgreSQL (Narváez, 2014)

Para el usuario la información de las tablas en estas dos instancias B1 y B2 no puede ser visualizada al mismo tiempo, aun cuando su rol tiene acceso a las dos bases de datos.

Se recomienda mantener la individualidad de aquella información, la cual no se requiere sea disponible para todos los usuarios, separándola en diferentes instancias físicas de base de datos.

Para controlar el acceso a las diferentes instancias físicas de base de datos, se recomienda su administración vía el fichero `pg_hba.conf`.

3.2.4.2. Separación Lógica

PostgreSQL permite una separación lógica de la información a través de sus Esquemas, contenidos dentro de una base de datos física.

Los Esquemas permiten la agrupación de relaciones lógicas como: tablas, secuencias, índices, etc. Su otra función, es proteger del acceso no autorizado al grupo lógico de datos.

Para acceder y manipular la información de varios esquemas dentro de una base de datos, se requiere el permiso de acceso para el rol en uso y la relación establecida entre los Esquemas. Cumplidos tales requerimientos, podremos inclusive unir información entre Esquemas.

Implementación

Utilizando pgadmin3, visualizaremos la separación lógica de los esquemas en una BD PostgreSQL.

Los pasos son exactamente los mismos en Windows y CentOS.

1. Abrimos pgAdmin3.
2. Realizamos conexión con el servidor deseado, para el ejemplo se usará el usuario “postgres”.
3. Visualizamos 3 instancias lógicas o esquemas dentro de la base de datos “B1”: esquema1, esquema2 y esquema por defecto de la base “public”.

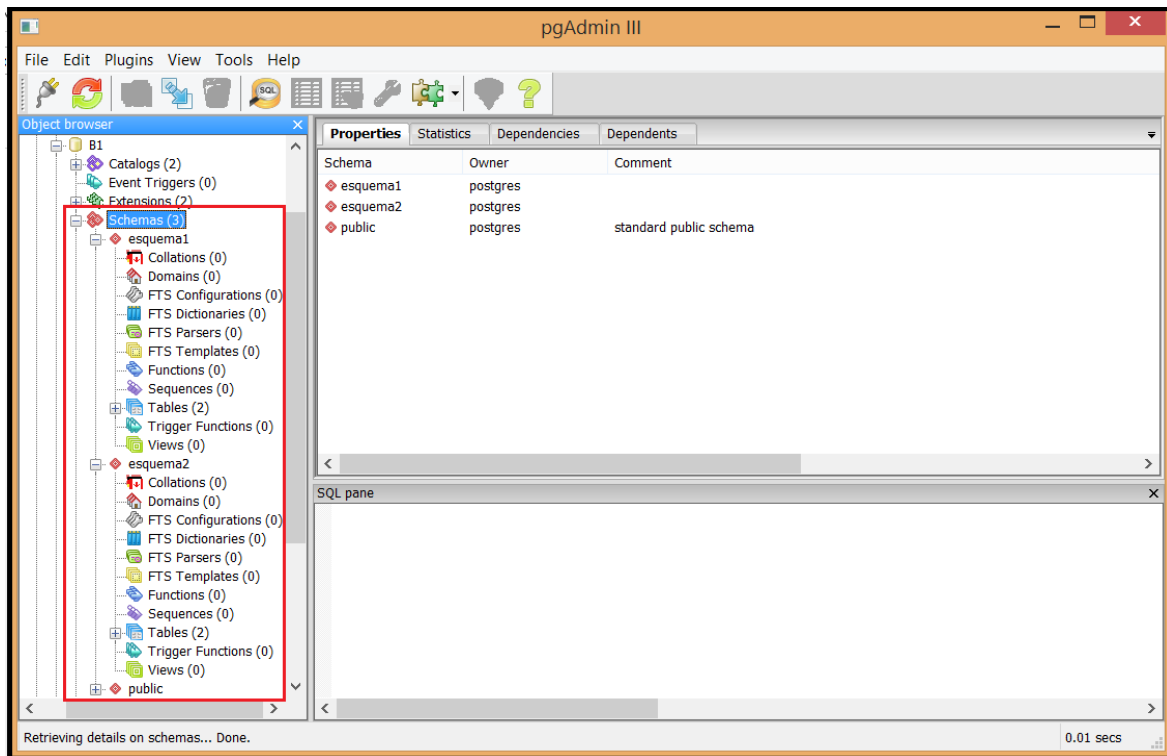


Grafico 3.8: Separación lógica esquemas de una BD PostgreSQL (Narváez, 2014)

Se recomienda separar la información lógicamente en diferentes Esquemas, basado en el tipo de acceso y uso que las aplicaciones dan a aquella información.

3.3. CUADRO DESCRIPTIVO

El cuadro descriptivo que recopila todas las buenas prácticas de seguridad obtenidas del capítulo es el siguiente:

N°	Practica	Descripción	Categoría	Prioridad
1	Cambio de contraseña.	Se debe usar una contraseña segura actualizable cada 19 días, con un mínimo de 6 caracteres, no incluya sustantivos, nombres y maneje mayúsculas, minúsculas y números.	Cambio de Contraseña	ALTA
2	Separar privilegios por sesión iniciada	A cada usuario de la base de datos se le debe asignar privilegios de acuerdo a la aplicación que este registra.	Usuarios y Privilegios	ALTA
3	Mantener la integridad de la información	No asignar permisos sobre objetos como bases de datos, tablas y esquemas a aquellos usuarios que acostumbran a usar comando para modificar la información.	Usuarios y Privilegios	ALTA
4	Asignar el tipo de rol de usuario según el acceso.	Se debe asignar el tipo de rol de usuario según el número y tipo acceso a las aplicaciones de la base de datos.	Usuarios y Privilegios	MEDIA
5	Añadir líneas en el registro por cada aplicación en uso.	Es necesario añadir una línea en el registro del fichero pg_hba.conf por cada host de la aplicación en uso.	Seguridad en el fichero pg_hba.conf	MEDIA
6	No utilizar “trust” para autenticación	No se debe utilizar el parámetro de autenticación “trust” en el registro pg_hba.conf, si se requiere optimizar la seguridad.	Seguridad en el fichero pg_hba.conf	ALTA
7	Utilizar el parámetro de autenticación encriptada md5	Se recomienda utilizar el parámetro de autenticación encriptada md5 en todos los registros del fichero pg_hba.conf.	Seguridad en el fichero pg_hba.conf	MEDIA
8	Añadir la línea host/all/all/0.0.0.0/reject al final del registro	Se debe añadir la línea host/all/all/0.0.0.0/reject al final del registro en el fichero pg_hba.conf para rechazar y cubrir el tráfico no definido.	Seguridad en el fichero pg_hba.conf	MEDIA
9	Mantener la individualidad de la información	Mantener la individualidad de aquella información que no debe estar disponible para todos los usuarios separándola en diferentes bases de datos.	Diseño de Datos	MEDIA

10	Controlar el acceso a las diferentes BD(s) con el fichero pg_hba.conf	Se debe controlar el acceso a las diferentes instancias físicas de bases de datos vía el fichero pg_hba.conf.	Diseño de Datos	ALTA
11	Separa la información lógicamente en Esquemas	Es importante, separar la información lógicamente en diferentes esquemas, según el tipo de acceso y uso de la información.	Diseño de Datos	BAJA

Tabla 3.9: Cuadro Descriptivo buenas prácticas en seguridad PostgreSQL (Narváez, 2014)

CAPÍTULO IV: BUENAS PRÁCTICAS EN ALTA DISPONIBILIDAD

4.1. ALTA DISPONIBILIDAD EN POSTGRESQL

Disponibilidad se refiere a la habilidad de la comunidad de usuarios para acceder al sistema, someter nuevos trabajos, actualizar o alterar trabajos existentes o recoger los resultados de trabajos previos. (Wikipedia, 2014)

Es un protocolo de diseño del sistema y su implementación asociada que asegura un cierto grado absoluto de continuidad operacional durante un período de medición dado, el cual es medido por el tiempo en línea y el tiempo fuera de línea.

- **Clúster**

Colección de dispositivos que trabajan juntos como uno solo para proveer alta disponibilidad. En las bases de datos se hace referencia a una arquitectura en la que se cuenta con varios equipos, que hacen uso de la base de datos para trabajar como un solo sistema. La arquitectura se define de acuerdo a la forma de almacenamiento de los datos en cada nodo.

- **Replicación**

La replicación en PostgreSQL es el proceso que consiste en realizar una copia exacta de una parte o de toda la base de datos y almacenarla en uno o varios sitios alternos llamados nodos. Bajo este concepto operan las configuraciones de nodo maestro – esclavo.

- **Replicación Sincrónica**

Cuando se realiza una transacción la misma se ejecutara en todos los nodos, tal proceso aunque costoso en términos de latencia y número de mensajes enviados evitara la divergencia de datos.

- **Replicación Asincrónica**

En la replicación asincrónica la transacción se ejecutara en los nodos en cualquier momento que se requiera, evitando la sobrecarga de mensajes y la subida de latencia.

- **Balanceo de Cargas**

Es una técnica que se usa para compartir la carga de trabajo que entre varios nodos que poseen una parte o toda la base de datos, se encuentra direccionado hacia sistemas de multiprocesamiento.

- **Consulta en paralelo**

Los datos son divididos en varios servidores, permitiendo ejecutar una consulta al mismo tempo en todos los servidores, reduciendo el tiempo de ejecución total. Esta técnica se recomienda en búsqueda de datos a gran escala.

El presente capítulo se desarrollará en torno a lograr alta disponibilidad usando replicación sincrónica en un clúster de 2 nodos, generando un escenario demostrativo donde se verificará la replicación en el clúster usando sentencias SQL.

Las primeras etapas se enfocan en definir las razones del uso de alta disponibilidad con replicación, sobre otras alternativas para disponer y salvaguardar la información. A continuación se precisarán las razones para la elección de la plataforma a usar en el clúster y la arquitectura a implementar.

Finalizamos el capítulo con la guía para la instalación y configuración de los paquetes de software necesarios para la replicación, junto a una serie de pruebas usando sentencias SQL.

En el transcurso del capítulo se formularán las metodologías y recomendaciones consideradas como buenas prácticas para el desarrollo de la alta disponibilidad en PostgreSQL. Trabajando con la guía “Replicación y alta disponibilidad de PostgreSQL con pgpool-II” (rafaelma, 2014), para la elaboración del ejemplo de replicación sincrónica.

4.2. ALTA DISPONIBILIDAD CON REPLICACIÓN

4.2.1. Porque usar Replicación

La replicación brinda la capacidad de sincronizar la información de una base de datos, a través de varios servidores. Este proceso ofrece varios beneficios que pueden ser de mucha ayuda para las actividades del negocio. (Oracle, 2014)

Los principales beneficios se encuentran en el rendimiento y garantizar la durabilidad de los datos, lo que provee al usuario y al negocio las siguientes características:

- **Mejora la fiabilidad de la aplicación**, gracias a que la información se encuentra almacenada en varios puntos independientes, garantizando su disponibilidad en el evento de una falla de hardware de cualquier servidor que se encuentre en el grupo de replicación.
- **Mejora el rendimiento de acceso a la información**, expandiendo el área geográfica de los nodos y con ello el acceso a la información desde terminales lejanas al servidor principal. Con esto logramos reducción en la latencia y carga de datos.
- **Mayor cobertura del servicio**, utilizando nodos en áreas geográficas lejanas, permitiendo que el servicio pueda ser accedido por usuarios limitados por la distancia al servidor principal; además de esto, permite la inclusión de un mayor número de usuarios al servicio.
- **Permite la sincronización continua de la información**, logrando que los datos de múltiples servidores, sincronicen su información de forma continua e instantánea.

Entre las desventajas de la replicación nos encontramos con un mayor requerimiento de recursos y almacenamiento, generando mayores costos de implementación. La otra desventaja, se encuentra en el requerimiento de mayor cantidad tiempo de procesamiento por parte de cada servidor, gracias a la diferente forma de manipulación de un objeto replicado.

Se recomienda el uso de replicación, cuando requerimos continuidad y rendimiento del servicio, al igual que ampliar su alcance geográfico y número de usuarios.

4.2.1.1. Replicación vs Backups

Una de las preguntas más importantes en un negocio que desea mantener la disponibilidad de su información almacenada en una base de datos, es la selección entre soluciones de backups, replicación o ambas alternativas. Considerando sus ventajas y desventajas, debemos determinar su viabilidad tomando en cuenta los siguientes factores:

- **Recuperación frente a desastres.**

Proceso que resume las actividades del negocio, frente a un desastre que impida su continuidad, son llamados desastres aquellos eventos que ponen en peligro el hardware o el software de la infraestructura de una organización, y por consiguiente la disponibilidad e integridad de su información.

- **Continuidad del negocio.**

Son los procedimientos que una organización implementa, en favor de permitir la continuidad de las funciones esenciales que permiten al negocio trabajar durante y después de un desastre.

Establecidos estos dos conceptos podemos analizar varios casos donde las opciones de replicación y Backups son requeridas.

- La replicación es necesaria cuando la empresa enfrenta desastres naturales o fallas en el arreglo RAID de los discos de su servidor, de esta forma se podrá asegurar la restauración de la información desde una réplica, y permitir la reanudación de las actividades de manera rápida.
- Cuando un usuario borra accidentalmente información del servidor principal, tal acción ocurrirá simultáneamente en el resto de nodos del clúster: esto significa que si un archivo es corrompido o borrado, su recuperación a través de replicación será imposible. La alternativa a este problema se encuentra en la implementación de un sistema de backups históricos en varios lugares de almacenamiento, permitiendo regresar a la versión funcional de la base de datos.

Analizando estos casos se puede concluir, **la necesidad de contar con un escenario ideal donde se logre alta disponibilidad que garantice la continuidad del negocio y la habilidad de restaurar y generar backups históricos.** El uso de estas dos opciones mejorará la protección de la integridad y disponibilidad de la información, aun cuando se requiere de un alto costo de implementación.

4.2.2. Elección de Plataforma

Cuando el administrador debe elegir la plataforma donde se situara la base de datos, se debe tomar en cuenta factores como el costo, compatibilidad, rendimiento y conocimiento de la plataforma; resaltando el uso de servidores bajo Linux, gracias al ahorro de dinero, mejor rendimiento y compatibilidad con ciertos SGBD.

PostgreSQL no distingue el uso de sus principales características en Linux y Windows, aunque existe una clara ventaja del uso de PostgreSQL en sistemas Linux, gracias al apoyo de la comunidad y al desarrollo de aplicaciones solo disponibles en esta plataforma.

Para esta sección de la guía, se trabajará en un ejemplo de Alta Disponibilidad sobre PostgreSQL, requiriendo herramientas que faciliten nuevos requerimientos como pgpool-II. Por lo tanto, la opción ideal trabajar recae en el uso de una plataforma Linux.

La elección de la plataforma donde estará hospedado nuestro SGBD PostgreSQL depende del administrador, pero **se recomienda el uso de Linux en todos los nodos del clúster, por cuestiones de costos, soporte, y el acceso a aplicaciones solo disponibles para esta plataforma.**

4.2.3. Arquitectura

La arquitectura usada para el clúster de servidores, está enfocada en lograr la alta disponibilidad y el rendimiento en el servidor de base de datos (rafaelma, 2014), valiéndonos de los siguientes programas:

- **PostgreSQL:** SGBD donde se lograra la alta disponibilidad.
- **Pgpool-II:** Software que gestiona la alta disponibilidad entre los servidores.

Cada clúster se encuentra alojado en una máquina virtual sobre un procesador Intel Quad Core con 4 GB de RAM asignados. El esquema de la base de datos a replicar es el siguiente:

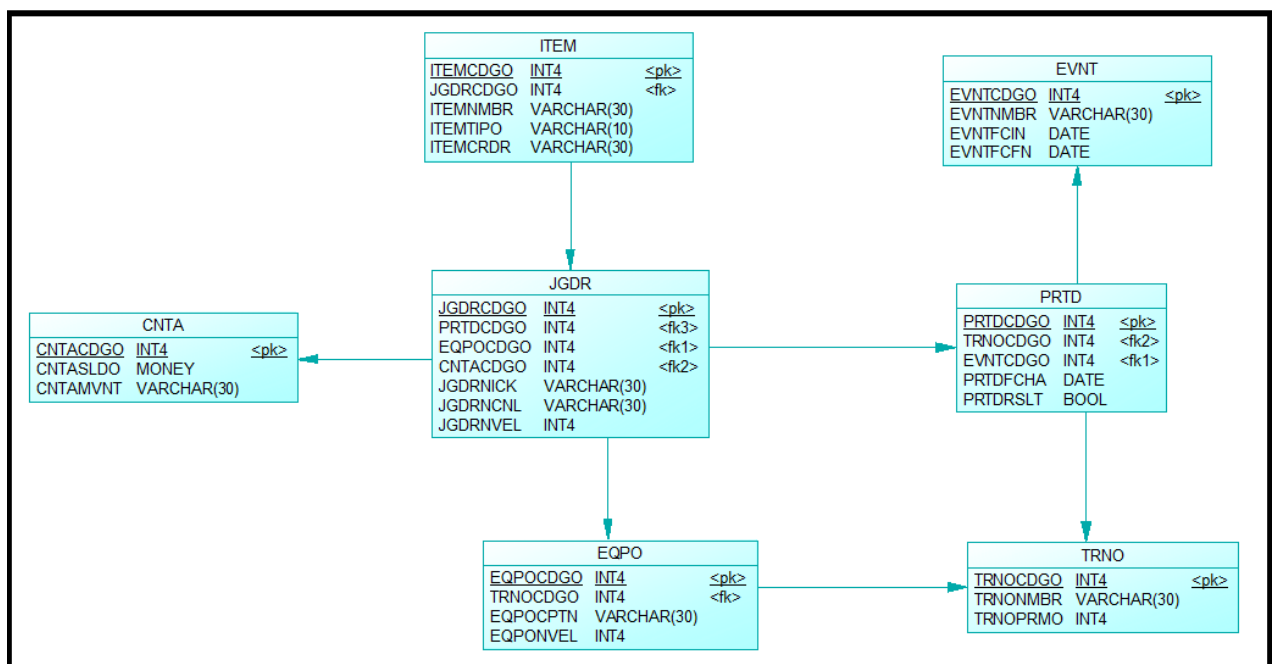


Grafico 4.1: Diagrama físico de la base de datos de prueba (Narváez, 2014)

Para la configuración de cada nodo se usaran los siguientes datos:

- **Nodo 1**
 - Hostname: nodo1.dominio.com
 - Puerto de gestión: 5432
 - Dirección IP de administración: 192.168.0.102
- **Nodo 2**
 - Hostname: nodo2.dominio.com
 - Puerto de gestión: 5432
 - Dirección IP de administración: 192.168.0.103
- **Pgpool-II**
 - Dirección IP de servicio: 192.168.0.102
 - Puerto de gestión: 9898
 - Puerto de servicio: 9999

Por limitaciones del equipo y para mejorar la representatividad de la guía, trabajaremos con la configuración definida, donde se busca realizar la replicación en un clúster de 2 nodos. El proceso es el mismo si se desea trabajar con un mayor número de nodos.

Se recomienda en ambientes en producción el uso de un clúster de varios nodos, permitiendo mayor seguridad en la continuidad del servicio de la base de datos.

El diagrama de la arquitectura usada para este ejemplo es el siguiente:

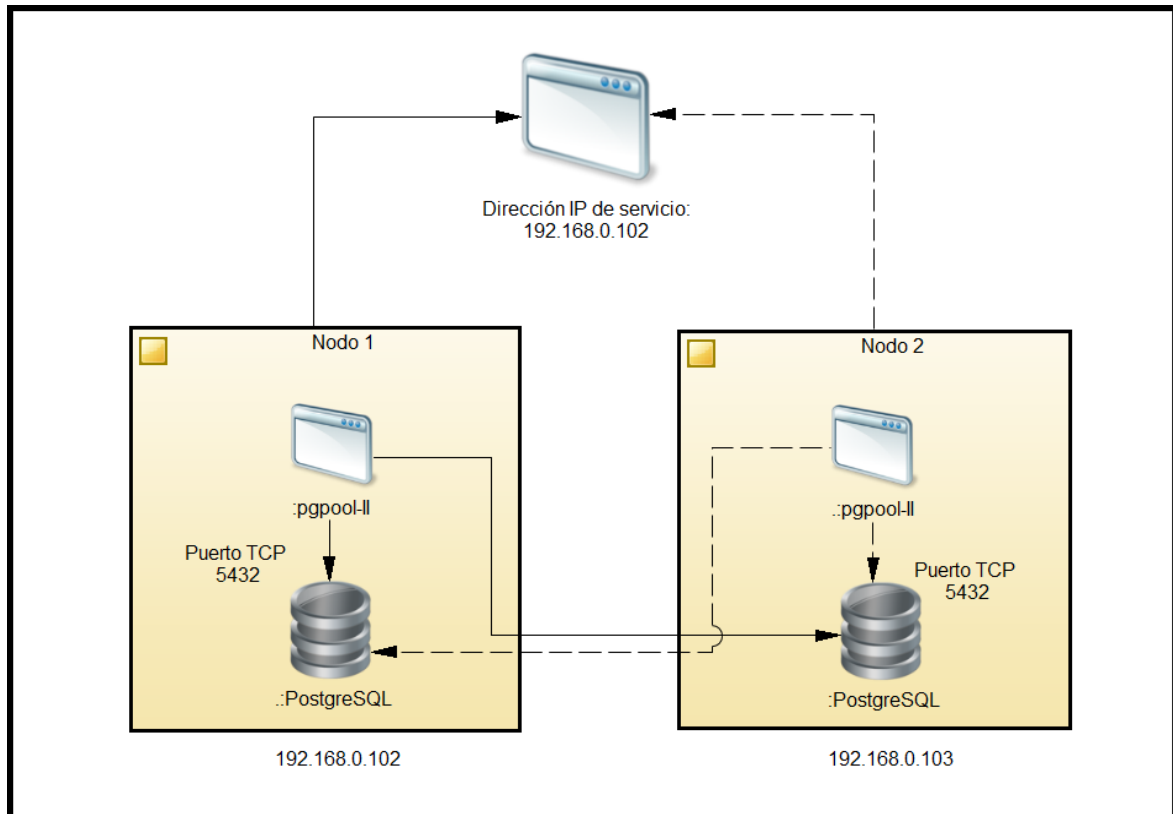


Grafico 4.2: Diagrama arquitectura del clúster (rafaelma, 2014)

Implementación

Configuraremos la conexión en red de los nodos al igual, que cambiaremos la referencia de direcciones IP a nombres de host.

1. Una vez configurada nuestra red y asignadas las direcciones IP definidas en nuestra arquitectura. ingresamos a la terminal de cada nodo.

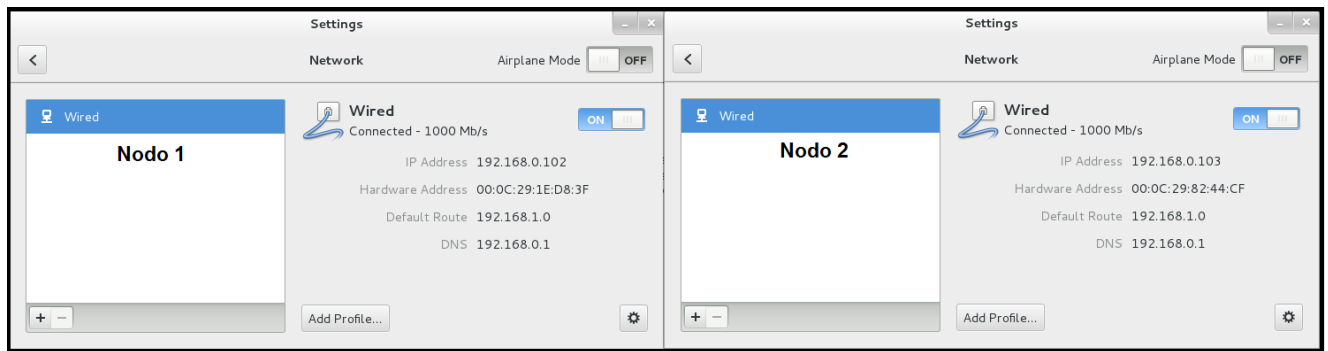


Grafico 4.3: Configuración de red en cada nodo (Narváez, 2014)

2. Cambiamos el nombre de host de cada nodo según la configuración establecida.

```
nano /etc/hostname
```

3. Modificamos el archivo de hosts en cada nodo, con las entradas establecidas en nuestra arquitectura.

```
nano /etc/hosts
```

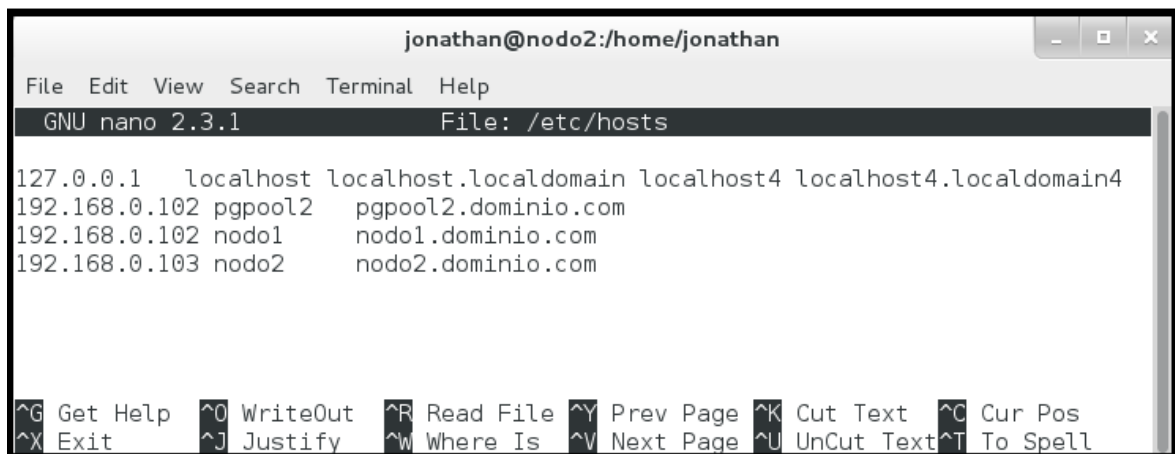


Grafico 4.4: Archivo hosts modificado con entradas (Narváez, 2014)

4. Guardamos nuestra configuración.

4.2.4. Instalación y Configuración

Tomamos como plataforma para cada nodo, al sistema operativo Centos 7 x86_64, utilizando las siguientes versiones de software:

- PostgreSQL 9.3.x
- Pgpool-II 2.2.x

Ciertos aspectos de la instalación y configuración de cada paquete de software, se encuentran contenidos en los capítulos anteriores; de tal forma, que solo se abordará los contenidos relevantes al desarrollo del capítulo.

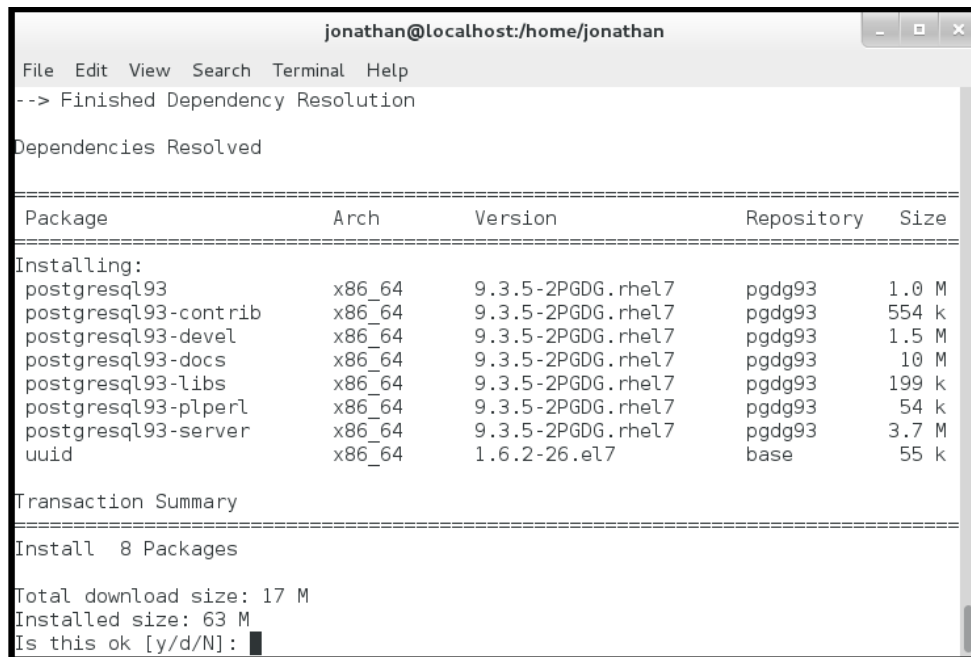
4.2.4.1. PostgreSQL

Instalaremos en cada nodo la versión 9.3.x de PostgreSQL, los pasos para este procedimiento se encuentran contenidos en el Capítulo II sección 2.2.1.

Implementación

1. Instalamos PostgreSQL como se indica en el Capítulo II, añadiendo nuevas librerías, para compatibilidad con pgpool-II:

```
yum install postgresql93 postgresql93-server postgresql93-libs  
postgresql93-contrib postgresql93-devel postgresql93-docs uuid perl-DBD-  
Pg
```



```
jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                               Arch      Version                Repository    Size
=====
Installing:
postgresql93                          x86_64    9.3.5-2PGDG.rhel7     pgdg93       1.0 M
postgresql93-contrib                 x86_64    9.3.5-2PGDG.rhel7     pgdg93       554 k
postgresql93-devel                   x86_64    9.3.5-2PGDG.rhel7     pgdg93       1.5 M
postgresql93-docs                    x86_64    9.3.5-2PGDG.rhel7     pgdg93       10 M
postgresql93-libs                     x86_64    9.3.5-2PGDG.rhel7     pgdg93       199 k
postgresql93-plperl                   x86_64    9.3.5-2PGDG.rhel7     pgdg93        54 k
postgresql93-server                   x86_64    9.3.5-2PGDG.rhel7     pgdg93       3.7 M
uuid                                  x86_64    1.6.2-26.el7          base         55 k
=====

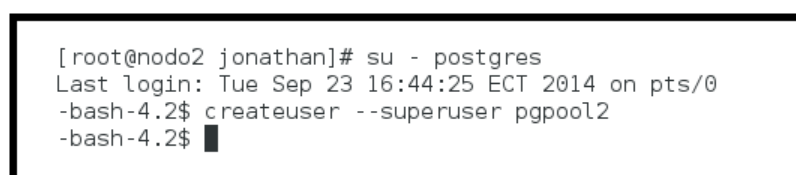
Transaction Summary
=====
Install      8 Packages

Total download size: 17 M
Installed size: 63 M
Is this ok [y/d/N]:
```

Grafico 4.5: Separación física instancias BD PostgreSQL (Narváez, 2014)

- Realizamos estos procedimientos, para tener instalado la versión 9.3.x del SGBD PostgreSQL en cada nodo.
- Creamos un nuevo superusuario pgpool2 en los dos nodos.

```
su - postgres
createuser --superuser pgpool2
```



```
[root@nodo2 jonathan]# su - postgres
Last login: Tue Sep 23 16:44:25 ECT 2014 on pts/0
-bash-4.2$ createuser --superuser pgpool2
-bash-4.2$
```

Grafico 4.6: Creación nuevo usuario en PostgreSQL (Narváez, 2014)

- Modificamos el fichero pg_hba.conf, añadiendo acceso al usuario pgpool2 en la dirección IP donde se ejecutará pgpool-II.

Tipo	BD	Usuario	IP	Método
host	all	pgpool2	192.168.0.102/32	trust

Tabla 4.1: Entrada en el archivo pg_hba.conf usuario pgpool2 (Narváez, 2014)

```

jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /var/lib/pgsql/9.3/data/pg_hba.conf
# TYPE DATABASE USER ADDRESS METHOD
local all all peer
# IPv4 local connections:
host all all 127.0.0.1/32 md5
# IPv6 local connections:
host all pgpool2 192.168.0.102/32 trust
host all all ::1/128 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 Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

jonathan@nodo2:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /var/lib/pgsql/9.3/data/pg_hba.conf Modified
# 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
host all pgpool2 192.168.0.103/32 trust
# IPv6 local connections:
host all all ::1/128 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 Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

```

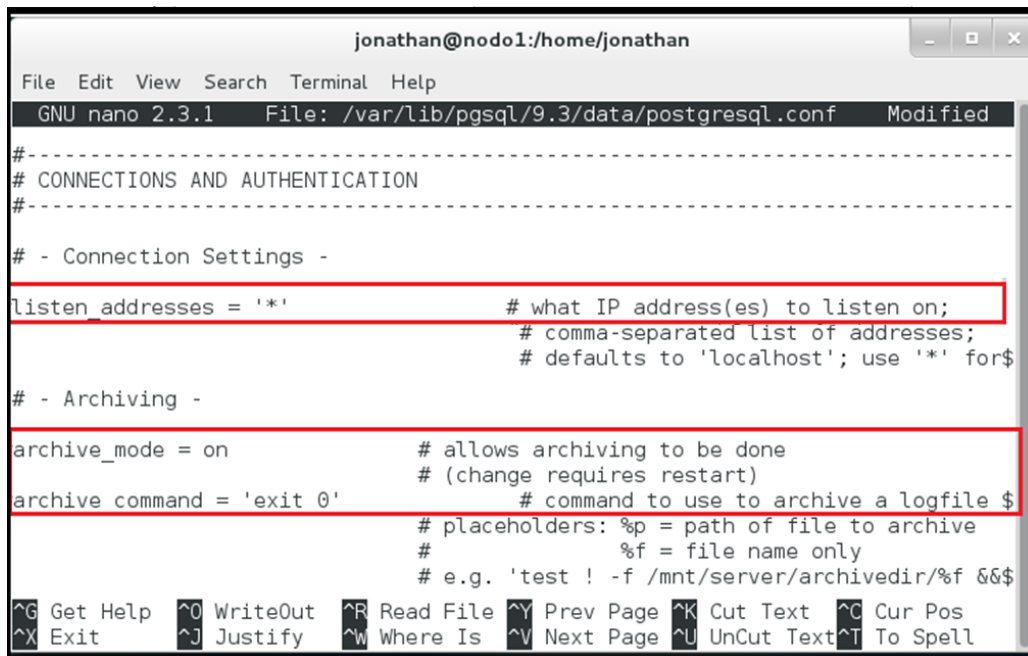
Grafico 4.7: Creación nuevo usuario en PostgreSQL (Narváez, 2014)

En ambientes en producción se recomienda restringir el acceso del usuario a la base de datos destinada a replicación.

5. Modificamos el fichero postgresql.conf tomando en cuenta:

Se activará el archivado WAL²⁸, necesario en el uso de PITR²⁹, cambiando los siguientes parámetros:

- **archive_mode:** Cuando está activado, los segmentos WAL son archivados y almacenados.
- **archive_command:** Define la ruta de archivo de los ficheros WAL, en este caso se maneja en <exit 0>. Con esto lograremos que PostgreSQL archive los ficheros de log en la dirección “/var/lib/postgresql/8.3/main/pg_xlog” rotándolos a partir del octavo almacenado.
- **listen_addresses:** Como se indicó en el Capítulo II, permitimos el acceso al resto de conexiones cambiando el parámetro (*).



```
jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /var/lib/pgsql/9.3/data/postgresql.conf Modified
#-----
# CONNECTIONS AND AUTHENTICATION
#-----
# - Connection Settings -
listen_addresses = '*'           # what IP address(es) to listen on;
                                # comma-separated list of addresses;
                                # defaults to 'localhost'; use '*' for$
# - Archiving -
archive_mode = on                # allows archiving to be done
                                # (change requires restart)
archive_command = 'exit 0'       # command to use to archive a logfile $
                                # placeholders: %p = path of file to archive
                                #               %f = file name only
                                # e.g. 'test ! -f /mnt/server/archivedir/%f &&$
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Grafico 4.8: Configuración postgresql.conf parámetros HA (Narváez, 2014)

²⁸ WAL – Write Ahead-Log

²⁹ BD – Point-In-Time-Recovery

6. Creamos un nuevo directorio llamado `pg_xlog_archive` en el directorio “`/var/lib/postgresql/`”, donde se archivarán los ficheros WAL, dando permiso al usuario postgres.

```
mkdir --mode=700 /var/lib/pgsql/pg_xlog_archive  
chown postgres:postgres /var/lib/pgsql/pg_xlog_archive
```

7. Reiniciamos PostgreSQL.

4.2.4.2. Pgpool-II

Instalaremos en el primer nodo la versión 3.3.x de Pgpool-II, teniendo en cuenta la configuración de la arquitectura ya definida. Del mismo modo instalaremos la versión 3.3.x de pgpoolAdmin.

Implementación

1. Abrimos la terminal de CentOS 7 y entramos como “root”.
2. Descargamos el repositorio de la nueva versión de Pgpool-II (En este caso la versión 3.3), instalando desde el archivo rpm de la página de pgpool. La página para buscar el repositorio deseado es:

```
http://www.pgpool.net/mediawiki/index.php/Downloads
```

```
rpm -ivh http://www.pgpool.net/download.php?f=pgpool-II-pg93-3.3.4-1.pgdg.x86\_64.rpm
```

```

jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
[jonathan@nodol ~]$ su
Password:
[root@nodol jonathan]# rpm -ivh http://www.pgpool.net/download.php?f=pgpool-II-pg93-3.3.4-1.pgdg.x86_64.rpm
Retrieving http://www.pgpool.net/download.php?f=pgpool-II-pg93-3.3.4-1.pgdg.x86_64.rpm
Preparing... ##### [100%]
Updating / installing...
  1:pgpool-II-pg93-3.3.4-1.pgdg ##### [100%]
[root@nodol jonathan]# █

```

Grafico 4.9: Descarga del repositorio de Pgpool-II en CentOS 7 (Narváez, 2014).

3. Instalamos Pgpool-II y aceptamos la instalación.

```
yum install pgpool-II-93
```

```

jonathan@localhost:/home/jonathan
File Edit View Search Terminal Help
[root@nodol jonathan]# yum install pgpool-II-93
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirror.esepoch.edu.ec
 * epel: mirror.esepoch.edu.ec
 * extras: mirror.esepoch.edu.ec
 * updates: mirror.esepoch.edu.ec
Resolving Dependencies
--> Running transaction check
--> Package pgpool-II-93.x86_64 0:3.3.2-1.rhel7 will be installed
--> Processing Dependency: libmemcached.so.11()(64bit) for package: pgpool-II-93-3.3.2-1.rhel7.x86_64
--> Running transaction check
--> Package libmemcached.x86_64 0:1.0.16-3.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package                Arch      Version              Repository           Size
=====
Installing:
pgpool-II-93            x86_64    3.3.2-1.rhel7        pgdg93               417 k
Installing for dependencies:
libmemcached            x86_64    1.0.16-3.el7         base                 236 k
=====
Transaction Summary
=====
Install 1 Package (+1 Dependent package)

Total download size: 654 k
Installed size: 2.1 M
Is this ok [y/d/N]:

```

Grafico 4.10: Instalación de la versión 3.3 de Pgpool-II en CentOS 7 (Narváez, 2014).

4. Si deseamos instalar pgpoolAdmin (solo plataformas Linux con escritorio GNOME), bajamos el rpm de la página anterior e instalamos:

```
rpm -ivh http://www.pgpool.net/download.php?f=pgpoolAdmin-3.3.1-1.pgdg.noarch.rpm
```

Para el uso de pgpoolAdmin debemos tener instalado ciertas librerías de php, entre las más importantes:

```
yum install php-cli php-mbstring php-pdo php php-pgsql php-gd php-soap php-mysql php-pear php-common php-devel
```

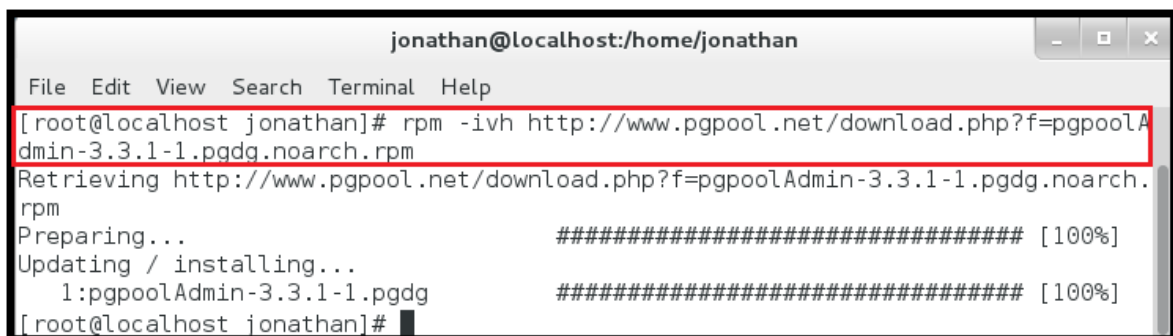


Grafico 4.11: Instalación PgpoolAdmin en CentOS 7 (Narváez, 2014).

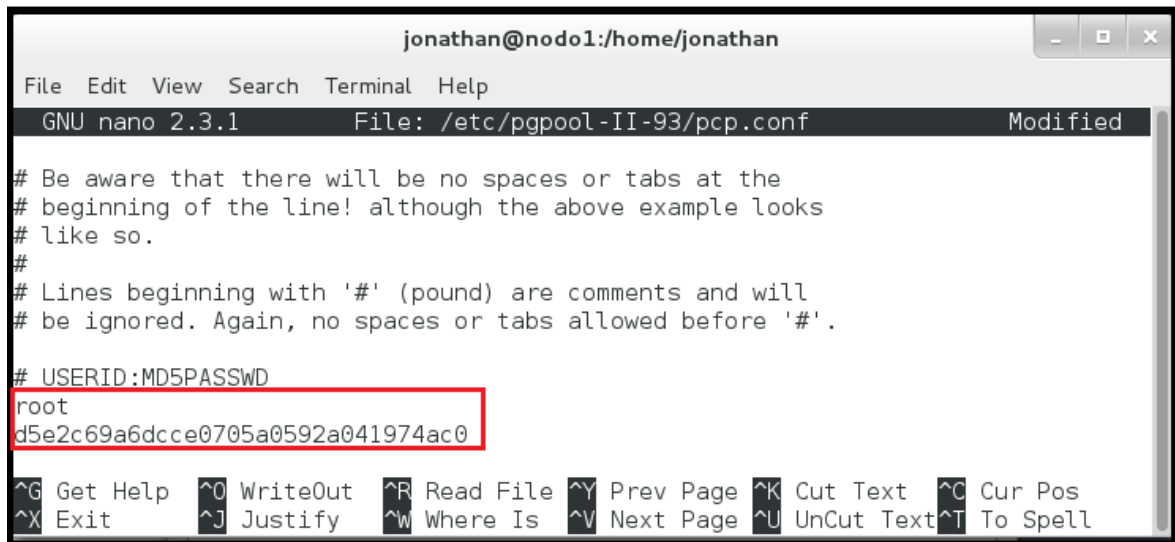
5. Manejamos el fichero pcpcnf que provee los nombres de usuarios y contraseñas, creándolo a partir del fichero pcpcnf.sample.

```
cp --archive /etc/pgpool-II-93/pcpcnf.sample /etc/pgpool-II-93/pcpcnf
```

Editamos el archivo pcpcnf introduciendo un nombre de usuario y contraseña en formato MD5 (usaremos root como nombre de usuario). Podemos utilizar la herramienta pg_md5 para generar nuestra contraseña.

```
/etc/alternatives/pgpool-pg_md5 <PASSWORD>
```

Usamos la contraseña en el archivo pcp.conf.



```
jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /etc/pgpool-II-93/pcp.conf Modified
# Be aware that there will be no spaces or tabs at the
# beginning of the line! although the above example looks
# like so.
#
# Lines beginning with '#' (pound) are comments and will
# be ignored. Again, no spaces or tabs allowed before '#'.
# USERID:MD5PASSWD
root
d5e2c69a6dcce0705a0592a041974ac0
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Grafico 4.12: Configuración fichero pcp.conf (Narváez, 2014).

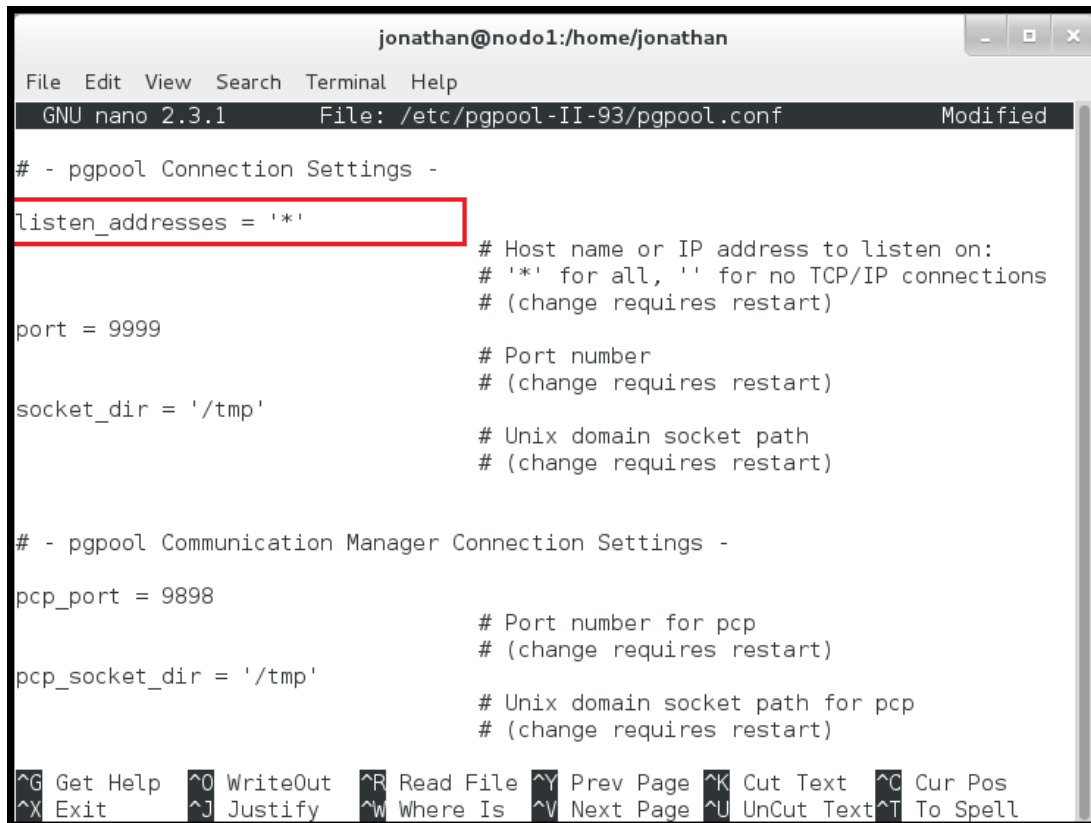
6. Configuramos Pgpool-II con el fichero pgpool.conf creándolo a partir del sample.

```
cp --archive /etc/pgpool-II-93/pgpool.conf.sample /etc/pgpool-II-93/pgpool.conf
```

Editamos el archivo pgpool.conf, tomando en consideración el pool de conexión, la replicación que se requiere alcanzar.

- **Configuración de la Conexión de Pgpool-II**

Modificaremos el parámetro `listen_addresses`, para que acepte conexiones de otros hosts y no sea limitada al local host. Mantenemos los mismos puertos para evitar conflictos con otras aplicaciones.



```
jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /etc/pgpool-II-93/pgpool.conf Modified
# - pgpool Connection Settings -
listen_addresses = '*'
# Host name or IP address to listen on:
# '*' for all, '' for no TCP/IP connections
# (change requires restart)
port = 9999
# Port number
# (change requires restart)
socket_dir = '/tmp'
# Unix domain socket path
# (change requires restart)

# - pgpool Communication Manager Connection Settings -
pcp_port = 9898
# Port number for pcp
# (change requires restart)
pcp_socket_dir = '/tmp'
# Unix domain socket path for pcp
# (change requires restart)

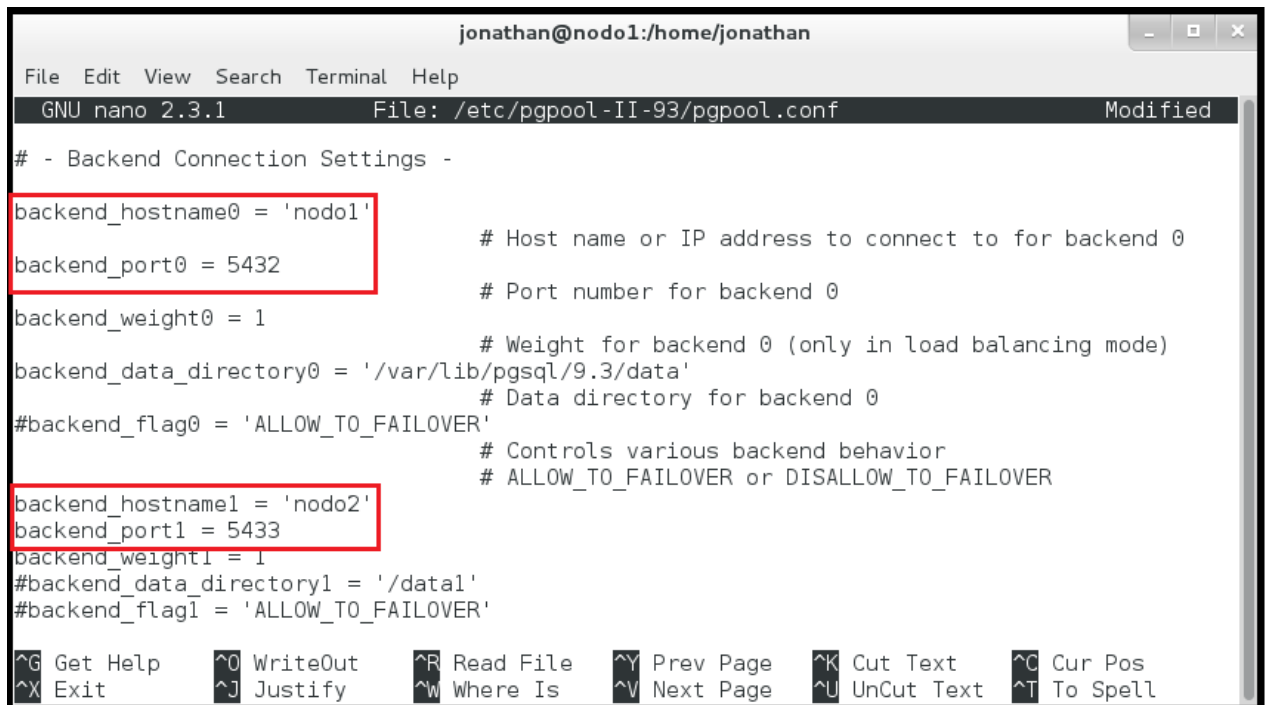
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Grafico 4.13: Configuración conexión fichero pgpool.conf (Narváez, 2014).

- **Configuración de la conexión del servidor**

En esta sección, configuraremos la conexión de nuestro clúster de nodos, se introducirán como parámetros las direcciones y puertos de cada nodo, tomadas de la arquitectura definida del clúster.

backend_host y **backend_post** son el nombre de host y el puerto que usan en PostgreSQL.



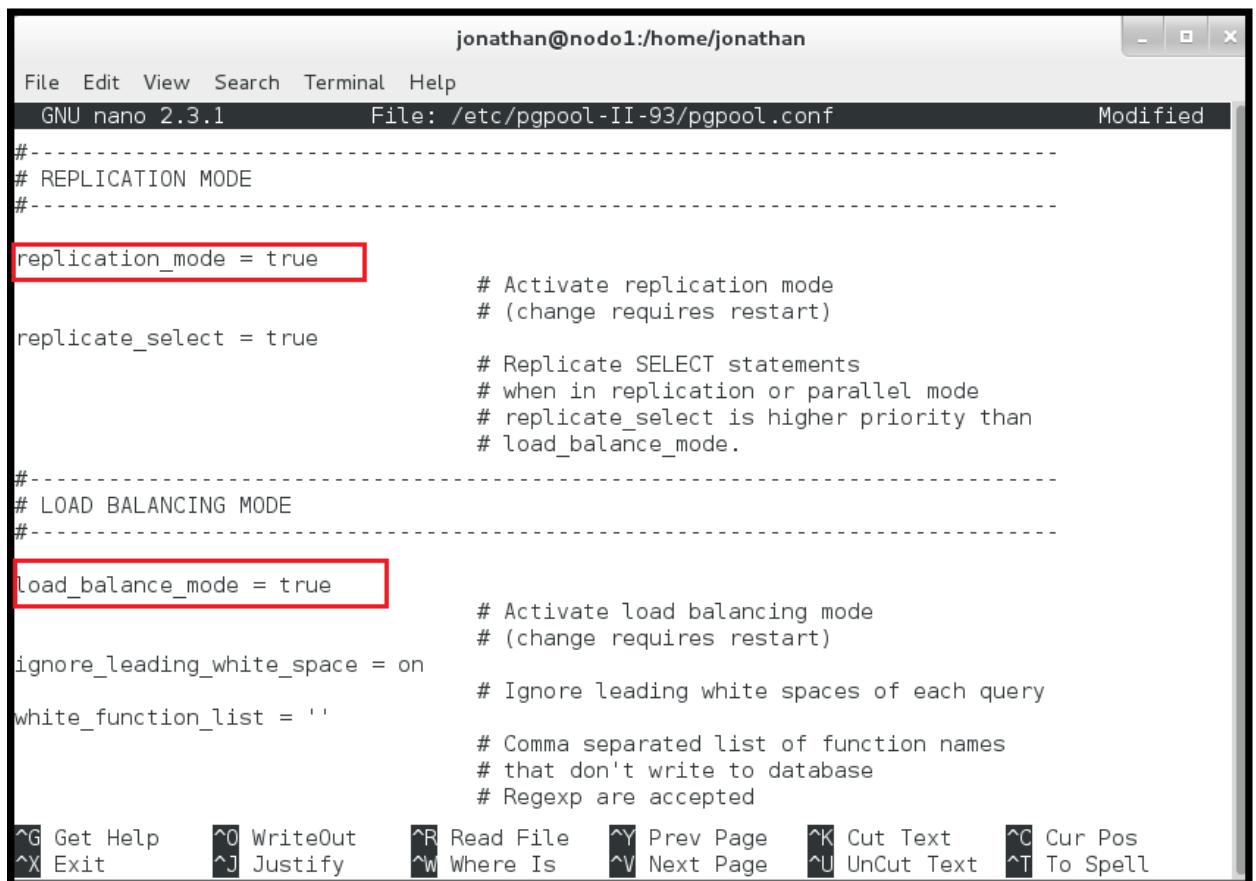
```
jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /etc/pgpool-II-93/pgpool.conf Modified
# - Backend Connection Settings -
backend_hostname0 = 'nodo1'
backend_port0 = 5432
backend_weight0 = 1
backend_data_directory0 = '/var/lib/pgsql/9.3/data'
#backend_flag0 = 'ALLOW_TO_FAILOVER'
backend_hostname1 = 'nodo2'
backend_port1 = 5433
backend_weight1 = 1
#backend_data_directory1 = '/data1'
#backend_flag1 = 'ALLOW_TO_FAILOVER'
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Grafico 4.14: Configuración backend host pgpool.conf (Narváez, 2014).

- **Activación de replicación y balanceo de carga,**

Si queremos utilizar replicación en nuestro clúster, debemos activarla con el parámetro “true”. Esto se realiza en la sección de configuración de conexiones del cluster.

Se recomienda activar la opción de balanceo de cargas, mejorando el rendimiento del clúster en la replicación.



```
jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
GNU nano 2.3.1 File: /etc/pgpool-II-93/pgpool.conf Modified
#-----
# REPLICATION MODE
#-----
replication_mode = true           # Activate replication mode
                                   # (change requires restart)
replicate_select = true          # Replicate SELECT statements
                                   # when in replication or parallel mode
                                   # replicate_select is higher priority than
                                   # load_balance_mode.
#-----
# LOAD BALANCING MODE
#-----
load_balance_mode = true         # Activate load balancing mode
                                   # (change requires restart)
ignore_leading_white_space = on  # Ignore leading white spaces of each query
white_function_list = ''         # Comma separated list of function names
                                   # that don't write to database
                                   # Regexp are accepted
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Grafico 4.15: Activación replicación y balanceo de cargas pgpool.conf (Narváez, 2014).

7. Iniciamos nuestro servidor Pgpool-II.

```
/etc/rc.d/init.d/pgpool-II-93 restart
```

```
[root@nodo1 jonathan]# /etc/rc.d/init.d/pgpool2 start
Starting pgpool2 (via systemctl): [ OK ]
[root@nodo1 jonathan]#
```

Grafico 4.16: Inicio de Pgpool-II (Narváez, 2014).

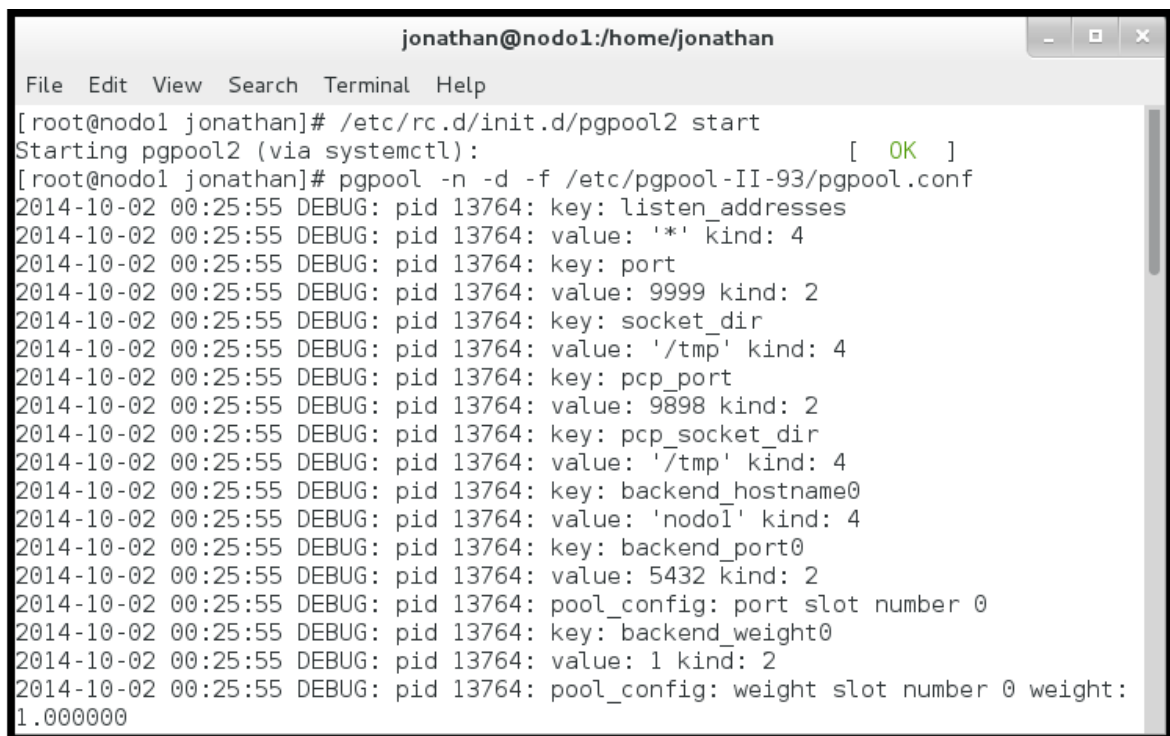
4.2.4.3. Conexión a través de Pgpool-II

Una vez configurado PostgreSQL en cada nodo, y realizada la instalación y configuración de Pgpool-II en el nodo 1, procedemos a realizar la primera conexión a través de Pgpool-II.

Implementación

1. Abrimos la terminal de CentOS 7 y entramos como “root”.
2. Arrancamos Pgpool-II en modo depuración.

```
pgpool -n -d -f /etc/pgpool-II-93/pgpool.conf
```



```
jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
[root@nodo1 jonathan]# /etc/rc.d/init.d/pgpool2 start
Starting pgpool2 (via systemctl): [ OK ]
[root@nodo1 jonathan]# pgpool -n -d -f /etc/pgpool-II-93/pgpool.conf
2014-10-02 00:25:55 DEBUG: pid 13764: key: listen_addresses
2014-10-02 00:25:55 DEBUG: pid 13764: value: '*' kind: 4
2014-10-02 00:25:55 DEBUG: pid 13764: key: port
2014-10-02 00:25:55 DEBUG: pid 13764: value: 9999 kind: 2
2014-10-02 00:25:55 DEBUG: pid 13764: key: socket_dir
2014-10-02 00:25:55 DEBUG: pid 13764: value: '/tmp' kind: 4
2014-10-02 00:25:55 DEBUG: pid 13764: key: pcp_port
2014-10-02 00:25:55 DEBUG: pid 13764: value: 9898 kind: 2
2014-10-02 00:25:55 DEBUG: pid 13764: key: pcp_socket_dir
2014-10-02 00:25:55 DEBUG: pid 13764: value: '/tmp' kind: 4
2014-10-02 00:25:55 DEBUG: pid 13764: key: backend_hostname0
2014-10-02 00:25:55 DEBUG: pid 13764: value: 'nodo1' kind: 4
2014-10-02 00:25:55 DEBUG: pid 13764: key: backend_port0
2014-10-02 00:25:55 DEBUG: pid 13764: value: 5432 kind: 2
2014-10-02 00:25:55 DEBUG: pid 13764: pool_config: port slot number 0
2014-10-02 00:25:55 DEBUG: pid 13764: key: backend_weight0
2014-10-02 00:25:55 DEBUG: pid 13764: value: 1 kind: 2
2014-10-02 00:25:55 DEBUG: pid 13764: pool_config: weight slot number 0 weight:
1.0000000
```

Grafico 4.17: Depuración del archivo Pgpool.conf (Narváez, 2014).

Donde:

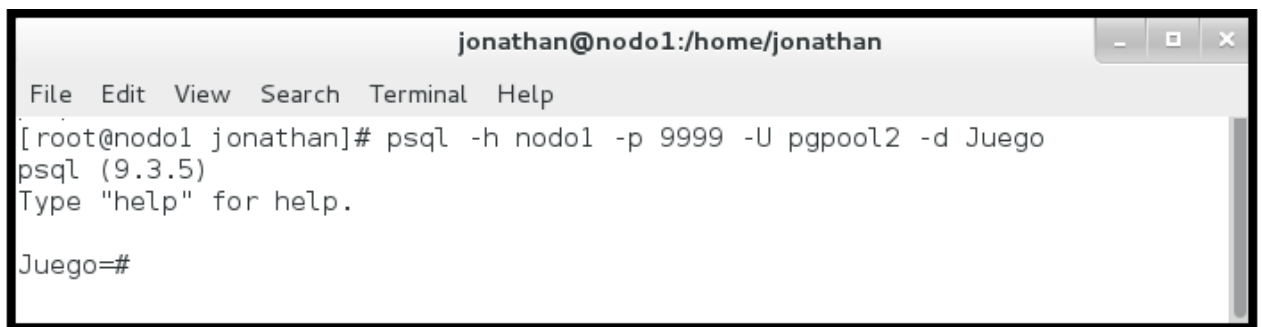
- **n** modo de log.
- **d** modo debug.
- **f** archivo pgpool.conf.

3. Nos conectamos a través de Pgpool-II a la base de datos de PostgreSQL.

```
psql -h nodo1 -p 9999 -U pgpool2 -d Juego
```

Donde:

- **h** nombre de host al que nos conectaremos y en donde está instalado Pgpool-II
- **p** puntero definido al archivo /etc/pgpool-II-93/pgpool.conf.
- **U** usuario creado en los dos nodos del clúster.
- **d** base de datos de PostgreSQL a la que nos vamos a conectar.



```
jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
[root@nodo1 jonathan]# psql -h nodo1 -p 9999 -U pgpool2 -d Juego
psql (9.3.5)
Type "help" for help.

Juego=#
```

Grafico 4.18: Conexión a la base de datos “Juego” a través de Pgpool-II (Narváez, 2014).

La depuración nos servirá para conocer la correcta conexión a través de Pgpool-II hacia PostgreSQL, al igual que nos permitirá visualizar las operaciones realizadas al momento de la replicación.

4.2.5. Pruebas de Replicación

Una vez terminada la instalación y configuración de nuestro clúster, al igual, que realizada la comprobación de la conexión de Pgpool-II y PostgreSQL; nos disponemos a probar la replicación en los nodos del clúster.

Implementación

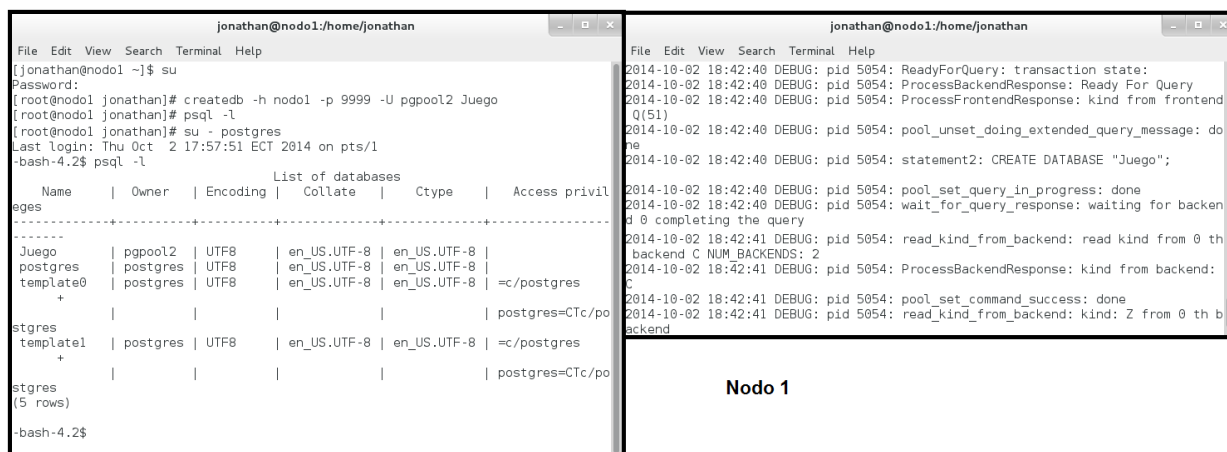
1. Abrimos la terminal de CentOS 7 y entramos como “root”.
2. Creamos una nueva base de datos, seguimos los parámetros establecidos al realizar la conexión Pgpool-II y PostgreSQL.

```
createdb -h nodo1 -p 9999 -U pgpool2 Juego
```

La base de datos debe ser creada a través de Pgpool-II, para poder ser replicada a los otros nodos del clúster. Cualquier cambio debe ser realizado apuntando a la aplicación.

3. Iniciamos sesión como usuario postgres en ambos nodos para comprobar que la creación de base de datos y la replicación tuvieron éxito.

```
su - postgres  
psql -l
```



Nodo 1

Grafico 4.18: Creación BD Juego Nodo 1 con modo depuración de Pgpool-II (Narváez, 2014).

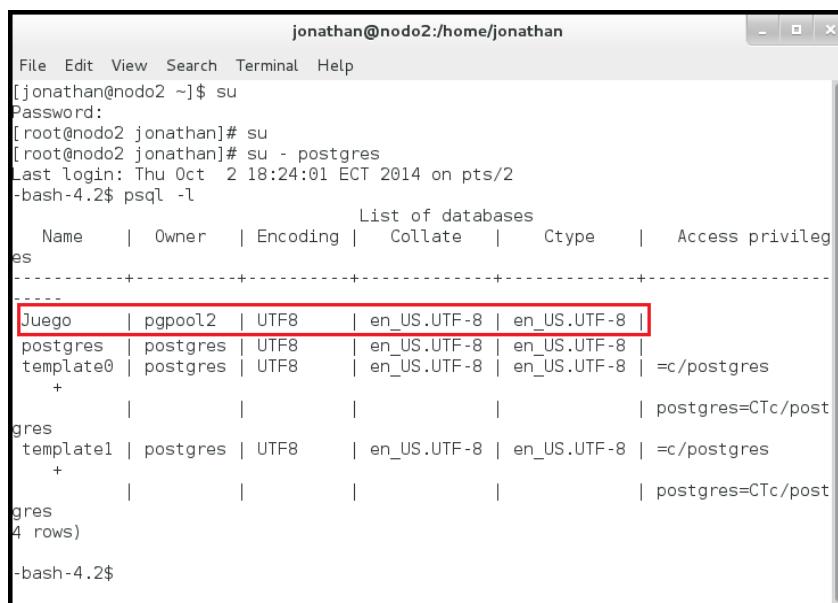


Grafico 4.18: Replicación en el Nodo 2 de la base de datos (Narváez, 2014).

4. Ingresamos a la base de datos recién creada a través del nodo 1, con el propósito de almacenar las tablas del esquema definido en la arquitectura.

```
psql -h nodo1 -p 9999 -U pgpool2 -d Juego
```

5. Comprobamos el contenido de la base de datos, e insertamos una nueva tabla.

```

jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
[jonathan@nodo1 ~]$ su
Password:
[root@nodo1 jonathan]# psql -h nodo1 -p 9999 -U pgpool2 -d Juego
psql (9.3.5)
Type "help" for help.

Juego=# \dt
No relations found.
Juego=# create table CNTA (
Juego(#      CNTACDGO          INT4          not null,
Juego(#      CNTASLDO          MONEY         not null,
Juego(#      CNTAMVNT          VARCHAR(30)   not null,
Juego(#      constraint PK_CNTA primary key (CNTACDGO)
Juego(# );
CREATE TABLE
Juego=#
2014-10-03 01:32:42 DEBUG: pid 9303: ProcessBackendResponse: kind from
2014-10-03 01:32:42 DEBUG: pid 9303: pool_read_message_length: slot: 0
2014-10-03 01:32:42 DEBUG: pid 9303: ReadyForQuery: transaction state:
2014-10-03 01:32:42 DEBUG: pid 9303: pool_unset_query_in_progress: don
2014-10-03 01:32:42 DEBUG: pid 9303: pool_unset_query_in_progress: don
2014-10-03 01:32:42 DEBUG: pid 9303: ProcessBackendResponse: Ready For
2014-10-03 01:32:58 DEBUG: pid 9303: ProcessFrontendResponse: kind fro
Q(51)
2014-10-03 01:32:58 DEBUG: pid 9303: pool_unset_doing_extended_query_m
e
2014-10-03 01:32:58 DEBUG: pid 9303: statement2: create table CNTA (
CNTACDGO          INT4          not null,
CNTASLDO          MONEY         not null,
CNTAMVNT          VARCHAR(30)   not null,
constraint PK_CNTA primary key (CNTACDGO)
);
2014-10-03 01:32:58 DEBUG: pid 9303: pool_set_query_in_progress: done
2014-10-03 01:32:58 DEBUG: pid 9303: wait_for_query_response: waiting

```

Grafico 4.19: Creación tabla en Nodo 1 (Narváez, 2014).

En el nodo 2, verificamos que se haya producido la replicación de la nueva tabla, para después insertar un nuevo registro.

```

jonathan@nodo2:/home/jonathan
File Edit View Search Terminal Help
[jonathan@nodo2 ~]$ su
Password:
[root@nodo2 jonathan]# psql -h nodo1 -p 9999 -U pgpool2 -d Juego
psql (9.3.5)
Type "help" for help.

Juego=# \dt
List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | cnta | table | pgpool2
(1 row)

Juego=# select * from cnta;
 cntacdgo | cntasldo | cntamvnt
-----+-----+-----
(0 rows)

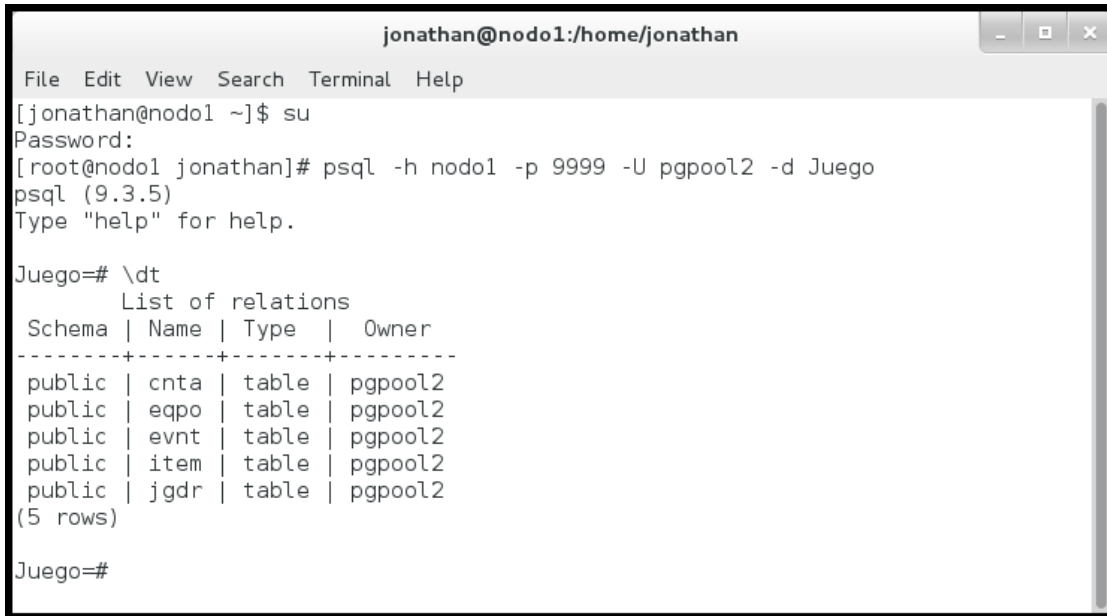
Juego=# insert into cnta values (123,1000,0);
Juego=# select * from cnta;
 cntacdgo | cntasldo | cntamvnt
-----+-----+-----
      123 | $1,000.00 | 0
(1 row)

Juego=#

```

Grafico 4.20: Comprobación tabla en el Nodo 2 (Narváez, 2014).

6. Para finalizar, insertamos todas las tablas con sus relaciones en la base de datos del nodo 1 para ser replicadas al nodo 2. El esquema de la base de datos se encuentra definido en la Arquitectura.

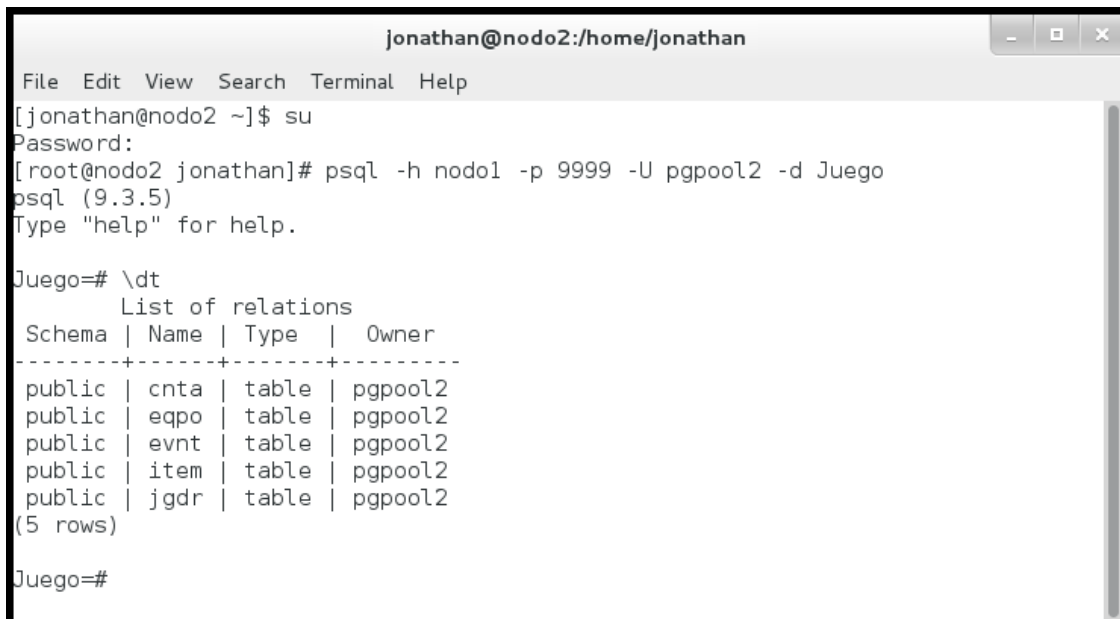


```
jonathan@nodo1:/home/jonathan
File Edit View Search Terminal Help
[jonathan@nodo1 ~]$ su
Password:
[root@nodo1 jonathan]# psql -h nodo1 -p 9999 -U pgpool2 -d Juego
psql (9.3.5)
Type "help" for help.

Juego=# \dt
          List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | cnta | table | pgpool2
public | eqpo | table | pgpool2
public | evnt | table | pgpool2
public | item | table | pgpool2
public | jgdr | table | pgpool2
(5 rows)

Juego=#
```

Grafico 4.21: Base de Datos de prueba en el Nodo 1 (Narváez, 2014).



```
jonathan@nodo2:/home/jonathan
File Edit View Search Terminal Help
[jonathan@nodo2 ~]$ su
Password:
[root@nodo2 jonathan]# psql -h nodo1 -p 9999 -U pgpool2 -d Juego
psql (9.3.5)
Type "help" for help.

Juego=# \dt
          List of relations
Schema | Name | Type | Owner
-----+-----+-----+-----
public | cnta | table | pgpool2
public | eqpo | table | pgpool2
public | evnt | table | pgpool2
public | item | table | pgpool2
public | jgdr | table | pgpool2
(5 rows)

Juego=#
```

Grafico 4.22: Base de Datos de prueba replicada en el Nodo 2 (Narváez, 2014).

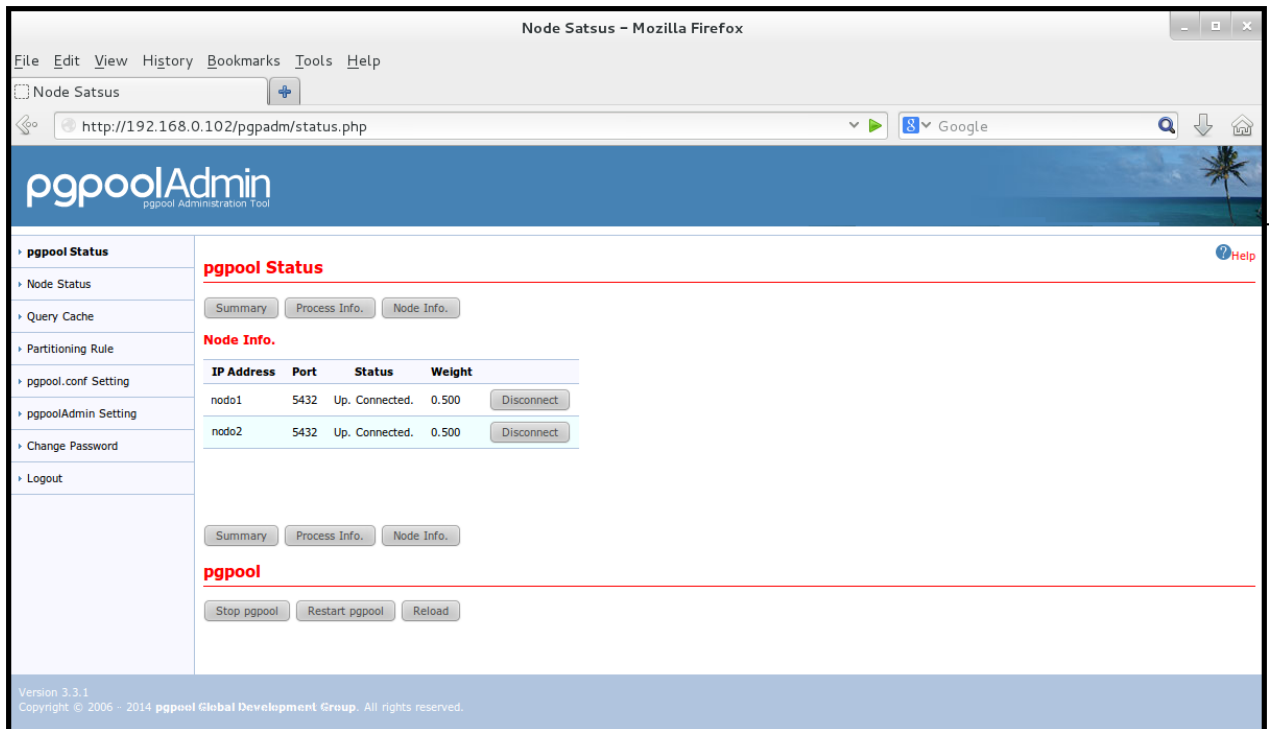


Grafico 4.23: Estado del clúster vista desde pgpoolAdmin (Narváez, 2014).

4.3. CUADRO DESCRIPTIVO

N°	Practica	Descripción	Categoría	Prioridad
1	Usar replicación al requerir continuidad y rendimiento del servicio	Usar replicación, al requerir continuidad y rendimiento del servicio, al igual que ampliar su alcance geográfico y número de usuarios.	Uso de Replicación	ALTA
2	Contar con un escenario que garantice la disponibilidad de la información.	Debemos contar con un escenario ideal donde se garantice la disponibilidad e integridad de la información, usando alta disponibilidad y la habilidad de restaurar y generar backups históricos	Uso de Replicación	MEDIA
3	Implementar alta disponibilidad en PostgreSQL usando una plataforma Linux.	Usar la plataforma Linux en todos los nodos del clúster destinado a alta disponibilidad, por cuestiones de costos, soporte y el acceso a aplicaciones solo disponibles para esta plataforma.	Elección de Plataforma	ALTA
4	Usar en ambientes en producción un clúster de varios nodos asegurando la continuidad del servicio.	Se debe hacer uso en ambientes en producción de un clúster de varios nodos, permitiendo mayor seguridad en la continuidad del servicio de la base de datos.	Arquitectura	MEDIA
5	Restringir el acceso del usuario a la BD destinada a replicación.	En ambientes en producción se debe restringir el acceso del usuario a la base de datos destinada a replicación.	Instalación y Configuración	MEDIA
6	Activar el balanceo de cargas para mejora en el rendimiento.	Activar el balanceo de cargas, mejorando el rendimiento del clúster en la replicación.	Instalación y Configuración	MEDIA
7	La base de datos debe ser creada a través de la aplicación para alta disponibilidad	La base de datos debe ser creada a través de la aplicación para alta disponibilidad, para poder ser replicada a los otros nodos del clúster. Cualquier cambio debe ser realizado apuntando a la aplicación.	Pruebas de Replicación	ALTA

CONCLUSIONES

- Se generaron un mayor número de buenas prácticas de administración en la parte de afinamiento, debido a la prioridad que existe en el correcto manejo de los recursos y memoria del sistema al usar el SGBD.
- La herramienta GUI de PostgreSQL, brinda un mayor control sobre la administración y seguridad del SGBD; esto se demuestra en aspectos como la generación de backups, restauración de la base de datos y manejo del diseño de datos, donde se facilita su uso para el usuario.
- Uno de los pilares fundamentales de la seguridad en PostgreSQL, se encuentra en el correcto uso del fichero `pg_hba.conf`; es en este punto, donde se generó el mayor número de buenas prácticas en seguridad, debido a la importancia que significa controlar la autenticación de cada cliente que accede a una base de datos.
- La replicación no reemplaza el uso de backups como método para mantener la disponibilidad e integridad de la información, siempre es necesario contar con ambientes de contingencias y sistemas de backups históricos, para la solución de problemas y eventualidades que impidan continuar con las actividades de negocio.
- El soporte y las aplicaciones desarrolladas para la versión de la plataforma Linux de PostgreSQL, hacen que esta sea la mejor opción para entornos en producción y cuando se requiere sacar el mayor provecho a las características y recursos del SGBD.
- PostgreSQL demuestra ser el SGBD FLOSS más completo y avanzado en la actualidad, las herramientas y las características que dispone en administración, seguridad y alta disponibilidad son muy intuitivas y comparables a las disponibles en SGBD como Oracle, MySQL y SQL Server.

- La investigación y sobretodo la formulación de las buenas practicas fueron facilitadas, gracias a la consulta de opiniones, comentarios de las comunidad de usuarios, y a la implementación y corroboración de tales experiencias.
- Esta guía de buenas prácticas brinda al lector una visión general de las principales características al administrar, manejar la seguridad y trabajar en alta disponibilidad con PostgreSQL. Se la puede interpretar como el punto de introducción en el manejo y correcto uso del SGBD PostgreSQL.

RECOMENDACIONES

- La fuente de consulta con mayor y mejor contenido es la propia comunidad de usuarios, de esta manera se puede obtener la información necesaria en que carecen de la apropiada documentación.
- Se recomienda tener énfasis y cuidado en el uso de los ficheros `pg_hba.conf` y `postgresql.conf`, ya que son el pilar fundamental de la administración y manejo de la seguridad en PostgreSQL.
- Conocer cómo manejar PostgreSQL en las plataformas Windows y Linux, permite encontrar la mejor opción según la finalidad y características a usar.
- Para darle un buen uso a nuestro SGBD, debemos tener presente las recomendaciones planteadas en esta guía, tanto en administración y seguridad.
- Siempre es necesario, desglosar las ventajas y desventajas de un método a seguir, al igual que compararlo con otras opciones presentes; de esta manera, podemos validar y asegurar que el presente procedimientos es el correcto e indicado.
- Debemos partir de un buen diseño de la arquitectura en un clúster para replicación, evitando problemas futuros y obteniendo una referencia clara que sirva como guía durante el proceso de instalación, configuración y pruebas.
- Se recomienda realizar pruebas usando sentencias SQL en cada nodo del clúster, para validar y comprobar el estado y efectividad de la replicación.
- La guía se encuentra diseñada para que el lector revise paso a paso cada uno de los contenidos, y utilice los cuadros descriptivos como resumen de las buenas practicas detalladas en cada capítulo.

REFERENCIAS BIBLIOGRÁFICAS

Altres cursos. (24 de Abril de 2014). Sistemas de Base de Datos Curso. Obtenido de Adimen:
http://adimen.si.ehu.es/~rigau/teaching/EHU/ABD/Altres%20cursos/mjortin/bd_t01_sistbd_doc.pdf

Casanova, J. (5 de Agostos de 2014). Afinamiento de la base de datos. Obtenido de Wikipostgresql:
https://wiki.postgresql.org/images/c/c5/Afinamiento_de_la_base_de_datos.pdf

Castaño, A. d., & Piattini, M. (1999). Fundamentos y modelos de bases de datos. Albacete: Alfaomega.

dbdude. (10 de Agosto de 2014). PostgreSQL Performance. Obtenido de dbdude:
<http://www.dbdude.com/postgresql/postgresqlperformance.php>

FEAPS. (23 de Abril de 2014). Buenas Practicas. Obtenido de FEAPS:
<http://www.feaps.org/que-hacemos/organizacion/calidad/buenas-practicas.html>

Hernandez, L. (05 de Mayo de 2014). Alta Dsiponibilidad en PostgreSQL. Obtenido de Slideshare: <http://www.slideshare.net/leninmhs/alta-disponibilidadpostgres>

McGraw-Hill. (20 de Abril de 2006). SGBD. Sistemas gestores de bases de datos (primera parte). McGraw-Hill. Obtenido de mailxmail.

Mead, S. (23 de Agosto de 2014). OpenSCG PostgreSQL Operational Procedures. Obtenido de openscg:

<http://www.openscg.com/wp-content/uploads/2013/04/SecurityHardeningPostgreSQL.pdf>

Oracle. (04 de Octubre de 2014). Replication Benefits. Obtenido de Oracle:
http://docs.oracle.com/cd/E17276_01/html/gsg_db_rep/C/repadvantage.html

Orlando Corro, E. T. (1 de Abril de 2014). Base de Datos: PostgreSQL. Obtenido de Sabd15N1: <http://sabd15n1.wikispaces.com/PostgreSQL>

Pgadmin. (04 de Mayo de 2014). Pgadmin. Obtenido de Pgadmin: <http://www.pgadmin.org>

Postgresql. (27 de Marzo de 2014). PostgreSQL 9.3.4 Documentation. Obtenido de Postgresql: <http://www.postgresql.org/docs/9.3/static/index.html>

PostgreSQL.org. (05 de Agosto de 2014). PostgreSQL 9.3.5 Documentation. Obtenido de PostgreSQL.org: <http://www.postgresql.org/docs/9.3/interactive/runtime-config.html>

rafaelma. (18 de Septiembre de 2014). Replicación y alta disponibilidad de PostgreSQL con pgpool-II. Obtenido de postgresql: <http://www.postgresql.org.es/node/313>

Rommgusa. (24 de Abril de 2014). Que es un sgbd. Obtenido de Rommgusa: <https://rommgusa.files.wordpress.com/2011/02/que-es-sgbd-tarea1.pdf>

thegeekstuff. (14 de Agosto de 2014). Raid Levels Tutorial. Obtenido de thegeekstuff:
<http://www.thegeekstuff.com/2010/08/raid-levels-tutorial/>

Thomas, S. M. (2014). PostgreSQL 9 High Availability Cookbook. Birmingham: PACKT Open Source.

Wiki.postgresql. (05 de Mayo de 2014). Pgpool-II. Obtenido de Wiki.postgresql:
<http://wiki.postgresql.org/wiki/Pgpool-II>

Wikipedia. (11 de Septiembre de 2014). Alta disponibilidad. Obtenido de Wikipedia:
http://es.wikipedia.org/wiki/Alta_disponibilidad

Wikipedia. (23 de Abril de 2014). Base de datos relacional. Obtenido de Wikipedia:
http://es.wikipedia.org/wiki/Base_de_datos_relacional

Wikipedia. (30 de Agosto de 2014). CentOS. Obtenido de Wikipedia:
<http://es.wikipedia.org/wiki/CentOS>

Wikipedia. (24 de Abril de 2014). Software Libre y de Codigo Abierto. Obtenido de Wikipedia: http://es.wikipedia.org/wiki/Software_libre_y_de_c%C3%B3digo_abierto

Wikipedia. (30 de Agosto de 2014). Windows 8. Obtenido de Wikipedia:
http://es.wikipedia.org/wiki/Windows_8