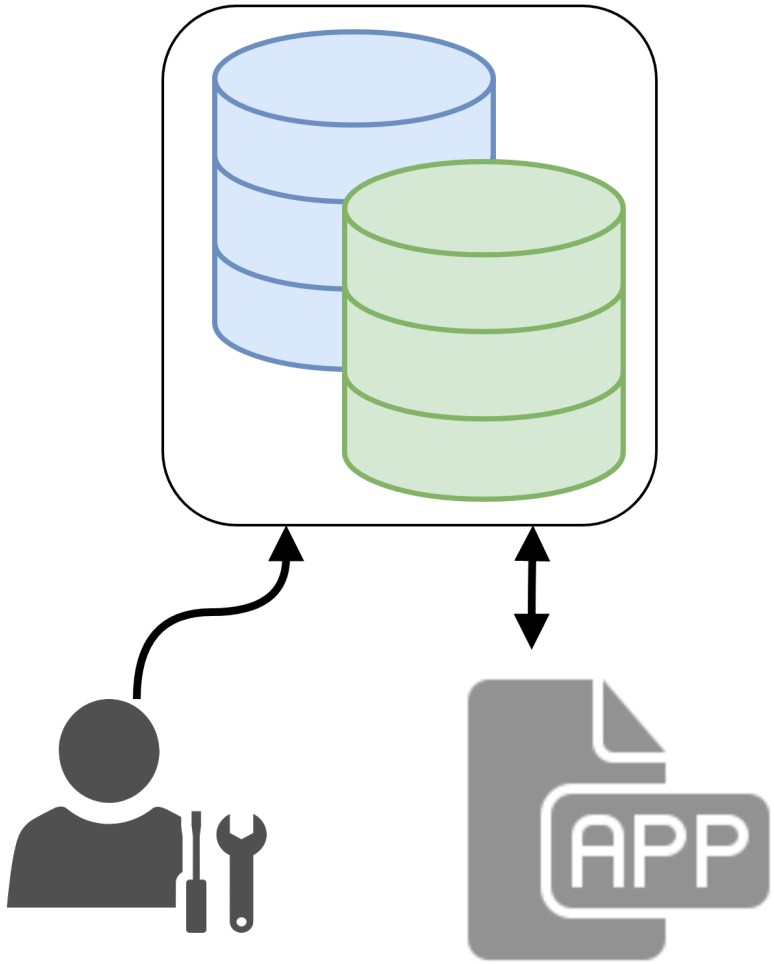
Técnico superior en Administración de Sistemas informáticos en red



Administración de Sistemas Gestores de Bases de Datos

Rubén Gómez Olivencia

2021-2022

- Github: https://github.com/yuki

Puedes encontrar la última versión de este libro en formato **HTML** en el siguiente link, así como otros libros que he creado. Para descargar el código fuente en formato **Markdown** visita el repositorio en GitHub.

Información

Por favor, ponte en contacto conmigo si encuentras algún fallo, falta de ortografía o quieres mejorar de alguna manera este libro. Gracias.

I Administración de Sistemas Gestores de Bases de Datos

II MySQL

VIII Anexos

Administración de Sistemas Gestores de Bases de Datos

1. Introducción

Hoy en día es habitual que los datos que usamos estén almacenados en una Base de Datos. Da igual que ese dato lo estemos utilizando desde un navegador web, en una aplicación de móvil o una videojuego. Cada consulta que realicemos al dato y cada posible modificación o eliminación del mismo realizará una petición (consulta o modificación) a un Sistema Gestor de Base de Datos.

Cada consulta realizada, cada petición de actualización o cada eliminación de datos, tendrá que ser procesada por el Gestor de Base de Datos y analizada para comprobar que lo que se va a realizar, como los permisos de quién pide la acción son adecuados, para posteriormente realizar la acción.

Los datos almacenados en una base de datos son de gran importancia en una empresa, por lo que la continuidad del servicio, así como la seguridad de los accesos recae en los administradores de Bases de Datos (DBA o DataBase Administrator) que deberán asegurar que el funcionamiento sea el esperado, así como la gestión de copias de seguridad de los mismos.

A lo largo de esta asignatura recordaremos los conceptos básicos de las bases de datos, comprenderemos la importancia y las funciones que desempeñan un Sistema Gestor de Base de Datos, aprenderemos a administrar el SGBD, crear y gestionar backups así como montar un sistema en Alta Disponibilidad.

2. Repaso

A modo de repaso rápido de la asignatura Gestión de Bases de Datos, veremos rápidamente unos conceptos que nos deberían ser conocidos.

2.1. ¿Qué es una Base de Datos?

Recordemos que una base de datos es un conjunto de datos que suelen pertenecer a un mismo contexto y que está almacenado para poder ser consultado posteriormente. Aunque actualmente una base de datos se asocia a un sistema informático, una biblioteca también puede considerarse una base de datos, ya que almacena libros que se pueden consultar, y el bibliotecario (el que te da acceso a los libros, si es que perteneces a la biblioteca) podría ser el símil del sistema gestor de la base de datos.

Actualmente las bases de datos se encuentran en todos los lugares, no sólo en servidores específicamente creados para ellos. Algunos ejemplos:

- Cada vez que usamos una aplicación de móvil, la propia aplicación cuenta con una base de datos interna (aparte de los datos que pueda consultar a bases de datos externas)
- Las páginas web que visitamos almacenan datos en pequeñas bases de datos en los propios navegadores que usamos.
- Aplicaciones de escritorio que guardan las preferencias del usuario en bases de datos.

No todas las bases de datos tienen por qué ser gestionadas por sistemas gestores (como los ejemplos

puestos previamente), ya que el acceso a los datos quizá no sea necesario que esté controlado, pero en entornos empresariales es lo habitual.

2.2. Tipos de Bases de Datos

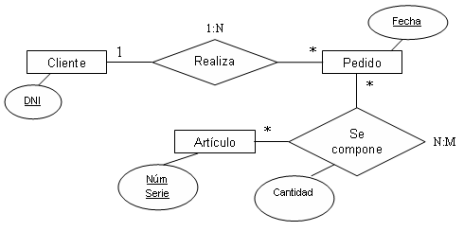
Las bases de datos se pueden clasificar de varias maneras, teniendo en cuenta el contexto que estemos manejando, las necesidades que tengamos, el tipo de datos que estemos utilizando…

Nos vamos a centrar en la clasificación teniendo en cuenta los distintos modelos de administración de los datos, concrétamente en el **modelo relacional**, aunque veremos otros también utilizados.

2.2.1. Bases de datos relacionales

Es el modelo más utilizado actualmente para representar problemas reales y administrarlos de manera dinámica. El paradigma nació en **1970** de la mano de Edgar Frank Codd cuya idea es el uso de "relaciones".

En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (que sí tenían otros modelos previos).



Origen: wikipedia

Para que una base de datos sea considerada relacional debe de seguir el modelo relacional, por lo que antes de introducir datos, para crear la base de datos **habremos realizado los pasos necesarios para pasar del modelo entidad-relación al modelo relacional**. Es por ello que hay que acordarse siempre de realizar la **normalización de la base datos**.

Para este tipo de bases de datos el lenguaje de consultas utilizado es el **SQL** (en inglés *Structured Query Language*; en castellano: lenguaje de consulta estructurada) el cual abordaremos más adelante.

2.2.2. No relacionales

Antes y después de la aparición del modelo relacional han existido distintos modelos de base de datos (jerárquico, de red, multidimensionales…), por lo que hay que entender que el relacional no es el único modelo existente, aunque sí el más utilizado.

2.2.2.1. Bases de datos Documentales

Las bases de datos documentales son aquellas que se encargan de almacenar datos de tipo documento, también conocidos como datos semi-estructurados.

En el dato almacenado puede existir una estructura fija, o que puede ser modificada en el tiempo. Normalmente esta información suele estar almacenada en JSON o XML.

Este tipo de bases de datos entran dentro de las denominadas NoSQL, cuyos datos no requieren estructuras fijas como tablas y cuyo acceso suele realizarse mediante el sistema "clave-valor".

Las bases de datos NoSQL están altamente optimizadas para las operaciones recuperar y agregar, y normalmente no ofrecen mucho más que la funcionalidad de almacenar los registros. No suele ser habitual

el poder realizar consultas de tipo JOIN, por lo que este tipo de operaciones se realizaría desde la aplicación que realiza las consultas de obtención de datos.

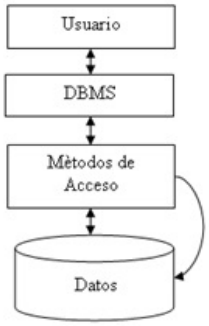
No entraremos en este modelo de base de datos, debido a sus diferencias con el modelo relacional, pero es obligatorio conocer que existen y que son utilizadas en aplicaciones como las redes sociales (por ejemplo).

Algunos ejemplos de gestores NoSQL son: MongoDB, Elasticsearch, …

3. Sistemas Gestores de Bases de Datos

Un Sistema Gestor de Base de Datos es un conjunto de programas que permiten el almacenamiento, modificación y consulta de datos de una base de datos. Teniendo en cuenta los permisos del solicitante se le otorgarán ciertos privilegios lo que hará que se le permitirá acceder (o no) a ciertas funciones que podrá realizar sobre los datos.

Estos sistemas proporcionan distintas tareas para mantener la integridad de los datos, administrar el acceso y la opción de recuperar información en caso de que el sistema se corrompa.



Origen: wikipedia

Tal como se puede ver en la imagen, el ejemplo de uso habitual de un Sistema Gestor de Base de Datos se puede resumir de la siguiente forma:

- El usuario se comunica con el SGBD (Sistema Gestor de Base de Datos, o en inglés DBMS: DataBase Management System).
- El SGBD comprueba que el usuario tiene permisos para acceder a los datos.
- El SGBD conoce cómo están almacenados los datos por lo que hará uso del método de acceso adecuado de cómo se han guardado los datos.
- Se recuperan los datos del dispositivo físico concreto donde estén almacenados los datos, teniendo en cuenta los datos pedidos por el usuario y el método de acceso.
- Se le entregan los datos al usuario.

Todas estas tareas deben de realizarse de la manera más rápida posible, por lo que la optimización de cada una de las partes debe de ser adecuada.

3.1. Componentes de un SGBD

Normalmente un SGBD tiene los siguientes componentes:

- El **motor de la base de datos** acepta peticiones lógicas de los otros subsistemas del SGBD, las convierte en su equivalente físico y accede a la base de datos y diccionario de datos en el dispositivo de almacenamiento.

- El **subsistema de definición de datos** ayuda a crear y mantener el diccionario de datos y define la estructura del fichero que soporta la base de datos.
- El **subsistema de manipulación** de datos ayuda al usuario a añadir, cambiar y borrar información de la base de datos y la consulta para extraer información. El subsistema de manipulación de datos suele ser la interfaz principal del usuario con la base de datos, y normalmente se hace uso del lenguaje SQL.
- El **subsistema de administración** ayuda a gestionar la base de datos ofreciendo funcionalidades como almacenamiento y recuperación, gestión de la seguridad, optimización de preguntas, control de concurrencia y gestión de cambios.

Dependiendo del SGBD que utilicemos, podremos contar con otros apartados que vienen incluídos, o programas externos que podremos utilizar para ampliar alguna funcionalidad del mismo.

3.2. Modelo ACID de transacciones

En bases de datos se denomina **ACID** a las características de los parámetros que permiten clasificar las transacciones de los sistemas de gestión de bases de datos, donde ACID es un acrónimo en inglés de ***A**tomicity, **C**onsistency, **I**solation and **D**urability* (en castellano: Atomicidad, Consistencia, Aislamiento y Durabilidad).

Las definiciones son:

- **Atomicidad**: Una transacción es una unidad lógica de trabajo que contiene una o varias sentencias SQL. El principio básico de una transacción es el todo o nada, una operación atómica tiene éxito o falla como un todo.
  - Un SGBD ha de ser capaz de asegurar la integridad de los datos ante la concurrencia de varios usuarios a la vez.
  - Un SGBD debe de ser capaz de agrupar varias sentencias SQL,de tal manera que puedan ser validadas (**commit**) o desechadas (**rollback**) como una unidad.
  - **Ejemplo**: en el caso de una transacción bancaria o se ejecuta tanto el depósito y la deducción o ninguna acción es realizada
- **Consistencia**: (Integridad). Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de Integridad de la base de datos.
  - El SGBD debe asegurar que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido.
  - El SGBD debe asegurar que los datos son exactos y consistentes, es decir que estén siempre intactos, sean siempre los esperados y que de ninguna manera cambian ni se deformen. De esta manera podemos garantizar que la información que se presenta al usuario será siempre la misma.

- **Isolation / Aislamiento**: Esta propiedad asegura que una operación no puede afectar a otras.
  - Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error.
  - Esta propiedad define cómo y cuándo los cambios producidos por una operación se hacen visibles para las demás operaciones concurrentes.
  - El aislamiento puede alcanzarse en distintos niveles, siendo el parámetro esencial a la hora de seleccionar SGBDs.
- **Durabilidad / Persistencia**: Esta propiedad asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema y que de esta forma los datos sobrevivan.

3.3. Software SGBD

Actualmente existen distintos SGBD que podemos instalar en nuestros servidores. Cada uno de ellos cuentan con sus características propias, por lo que tendremos que conocer las necesidades que tenemos a la hora de elegir entre ellas.

- MySQL: base de datos relacional, desarrollada por Oracle desde que en 2008 ésta se hiciera con Sun Microsystems y de licencia libre (aunque también cuenta con una versión no-libre).
- PostgreSQL: base de datos relacional, desarrollada por PostgreSQL Global Development Group y de licencia libre.
- SQL Server: base de datos relacional, desarrollada por Microsoft.
- Oracle Database: base de datos de tipo objeto-relacional desarrollada por Oracle Corporation.
- DB2: base de datos relacional, desarrollada por IBM.
- MongoDB: base de datos documental, desarrollada por MongoDB y de licencia libre.
- Cassandra: base de datos NoSQL distribuida y basada en el modelo clave-valor, desarrollada por Apache y de licencia libre.
- Elasticsearch: base de datos documental que cuenta con un servidor de búsqueda de texto completo.

Existen otros SGBD (tanto relacionales como no), pero el listado muestra las más conocidas y utilizadas a día de hoy. La elección del SGBD que vayamos a utilizar en nuestro proyecto debería ir acompañado de un análisis profundo de las características de cada uno de ellos, así como de las necesidades que requerimos.

MySQL

1. MySQL como Sistema Gestor de Base de Datos

El SGBD que usaremos durante esta asignatura es MySQL.

1.1. Introducción

MySQL es un sistema de gestión de bases de datos relacional desarrollado actualmente por Oracle Corporation, conocida empresa que también tiene su sistema SGBD Oracle privativo. MySQL cuenta con una **licencia dual**: Licencia Pública General (GPL) y licencia comercial, por lo que en su página web podremos encontrar ambas versiones (la primera de código abierto y gratuita, y la segunda con opción de pago, con servicios extra y soporte).

Actualmente MySQL está considerada como la base de datos relacional de código abierto más popular y se puede instalar en los tres sistemas operativos más conocidos actualmente.

1.1.1. Un poco de historia

MySQL fue inicialmente desarrollado por MySQL AB (empresa fundada por David Axmark, Allan Larsson y Michael Widenius). MySQL AB fue adquirida por Sun Microsystems en 2008, y ésta a su vez fue comprada por Oracle Corporation en 2010.

Es cierto que aunque en sus primeras versiones carecía de características como la integridad referencial y transacciones (debido al motor MyISAM utilizado en la creación de tablas), que son características muy importantes en un SGBD (y que PostgreSQL sí tenía), no impidió que cogiera fama en los denominados entornos LAMP (Linux + Apache + MySQL + PHP).

Poco antes de la compra de Oracle, desde la comunidad libre se realizó un fork (una copia completa) del código fuente de MySQL que dió origen a **MariaDB**. Desde ese momento, ambos SGBD han tenido vidas paralelas, pero el origen es el mismo.

Muchas distribuciones GNU/Linux contaban con MySQL como sistema SGBD para poder ser instalado, pero a medida que el desarrollo de MariaDB fue ganando adeptos muchas distribuciones realizaron el cambio, por lo que en ciertas distribuciones no es posible instalar MySQL desde los repositorios oficiales. De hecho, algunas distribuciones mantienen el paquete MySQL pero siendo un alias para que se instale MariaDB.

Desde la compañía Percona también crearon un fork de MySQL en el que añaden mejoras creadas por ellos y también venden soporte para el mismo.

Como veremos a continuación, el no poder realizar la instalación desde los repositorios oficiales no nos impedirá tener MySQL instalado en nuestro sistema.

1.1.2. Versiones de MySQL

MySQL cuenta con distintas versiones de SGBDs que hay que conocer para saber qué versión se necesita en cada caso concreto. Nos vamos a centrar en las versiones **Community** (las de licenciamiento libre), pero

estas versiones también cuentan con su versión con licencia comercial.

1.1.2.1. MySQL Community Server

Es el SGBD que vamos a utilizar. Es la versión "clásica" de MySQL como SGBD, que permite crear bases de datos, introducir datos, manipularlos, … Esta versión cuenta con la opción de crear un sistema replicado "Primario → Réplica" o "Primario ↔ Primario" como veremos a lo largo del curso.

1.1.2.2. MySQL Cluster

Originalmente MySQL no soportaba crear clusters de servidores, sólo el sistema de replicación que soporta la versión "Community Server", por lo que surgió la necesidad de poder crear un sistema clusterizado de 3 nodos o más que se gestionasen entre ellos, mantuvieran la información replicada… Eso se pudo realizar gracias a la librería Galera, que sirve para sincronizar la replicación de múltiples padres.

Hay que recordar que **MySQL Server es distinto a MySQL Cluster**, aunque desde un punto de vista de usuario que no entiende pueda parecer lo mismo.

1.1.2.3. MySQL Router

MySQL Router provee un enrutado transparente entre la aplicación de un usuario y cualquier servidor de MySQL. Puede sernos útil en sistemas de alta disponibilidad o de escalado para enrutar el tráfico al backend que más nos interese.

1.1.2.4. MySQL Workbench

Es un interfaz gráfico que nos proporciona herramientas para comprobar el estado de MySQL en el sistema remoto que tengamos que administrar. Existen versiones para distintos sistemas operativos y podremos instalarlo para conectarnos a servidores MySQL Remotos.

1.2. Características de MySQL Community Server

MySQL cuenta con una serie de características que hace que sea utilizado como SGBD de manera generalizada actualmente. Entre las características a destacar:

- **Facilidad de uso**: Es un SGBD sencillo de utilizar en comparación con otras alternativas libres o privativas (PostgreSQL u Oracle respectivamente)
- **Soporte de motores de almacenamiento**: Hasta la versión 5.5 se hacía uso del motor MyISAM que no tenía integridad referencial, pero eso se cambió por el uso del motor **InnoDB** que es el utilizado actualmente. Soporta varios motores, entre los que podemos destacar:
  - **InnoDB**: el utilizado por defecto actualmente. Es ACID compliant, seguro en transacciones, posibilidad para realizar commit y rollback.
  - **MyISAM**: debería usarse sólo para tablas de lectura, ya que no soporta transacciones, pero es muy rápido.

    - **Memoria**: se guarda toda la información en RAM, por lo que no sirve para persistencia de datos, pero hace que la información sea más rápida al acceder a ella
    - **CSV**: las tablas realmente son ficheros de texto en formato CSV. No soporta indexado.
- **Diseño multi-thread**: por lo que permite hacer uso de múltiples hilos de CPU en caso de que estén disponibles.
- **Replicación**: Permite crear entornos de replicación Primario → Réplica y Primario ↔ Primario.
- **Multiplataforma**: Funciona en distintas plataformas (distintas versiones de GNU/Linux, MacOS, Windows, FreeBSD, …).
- **Software Libre**: Tiene licencia libre lo que hace que podamos ver cómo funciona, y realizar modificaciones al código. Con ello se ha conseguido:
    - **Mucho soporte de la comunidad**: Existen muchas herramientas realizadas por la comunidad que facilitan el uso y/o la administración de servidores MySQL.

1.3. Instalación de MySQL Community Server

Como ya se ha mencionado antes, en algunas distribuciones MySQL ha sido sustituido por MariaDB (como en el caso de Debian), mientras que en otras se puede realizar la instalación de cualquiera de las dos (el caso de Ubuntu).

Aunque haremos uso de la distribución Ubuntu, también se va a explicar brevemente cómo se haría la instalación en sistemas donde no podemos contar con el paquete en el repositorio oficial.

1.3.1. Sin paquete en el repositorio oficial

En caso de que nuestra distribución no cuente con el paquete en los repositorios oficiales, la instalación no será tan directa, pero eso no significa que sea difícil. La versión Community Server la podremos encontrar en su web de descarga, y desde aquí podremos descargarnos la versión que necesitemos para el sistema operativo que queramos.

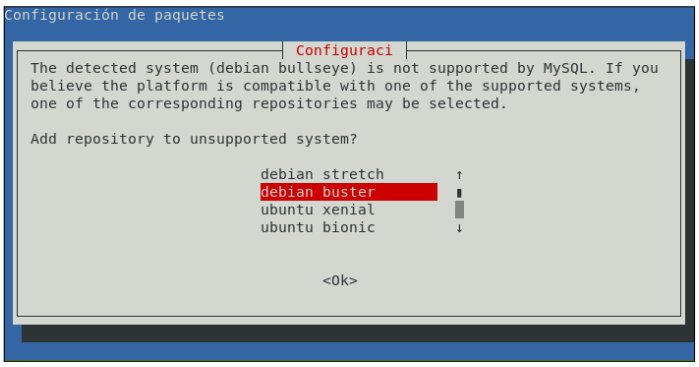
En el caso de que queramos instalarlo en una distribución de GNU/Linux como Debian, Red Hat o Suse podremos hacer uso de los repositorios oficiales de MySQL para realizar la instalación. En el caso de Debian, sería:

- Descargar el paquete para poder configurar el repositorio oficial de MySQL para Debian/Ubuntu.
- Instalar el paquete:

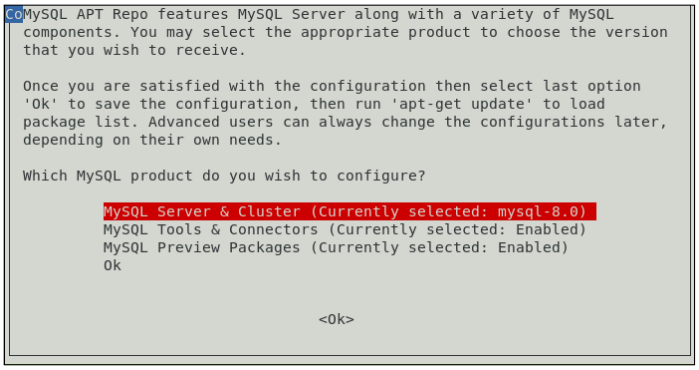
Instalar paquete MySQL descargado de la web oficial

```
ruben@server1:~$ sudo dpkg -i mysql-apt-config_0.8.15-1_all.deb
```

- Elegir en el menú que nos aparecerá:
    - En qué distribución estamos (ya que el mismo paquete sirve para Debian y Ubuntu).



- Qué versión vas a querer instalar (dependiendo de la distribución y la versión en la que nos encontremos nos dejará unas versiones u otras).



- Instalar mysql-server (tal como veremos a continuación en Ubuntu).

1.3.2. En Ubuntu 20.04

La versión LTS de Ubuntu 20.04 cuenta con la versión **8.0 de MySQL** (concretamente la **8.0.21** en el momento en el que es creado este documento).

Debido a que contamos con esta versión, que es la última versión de MySQL podremos realizar la instalación de la siguiente manera:

Instalar paquete MySQL del repositorio de la distribución

```
ruben@server1:~$ sudo apt install mysql-server-8.0
```

2. Administración básica de MySQL

Una vez tenemos instalado nuestro SGBD tenemos que aprender los conceptos básicos para poder conectarnos a él, poder configurarlo y llegar a administrarlo.

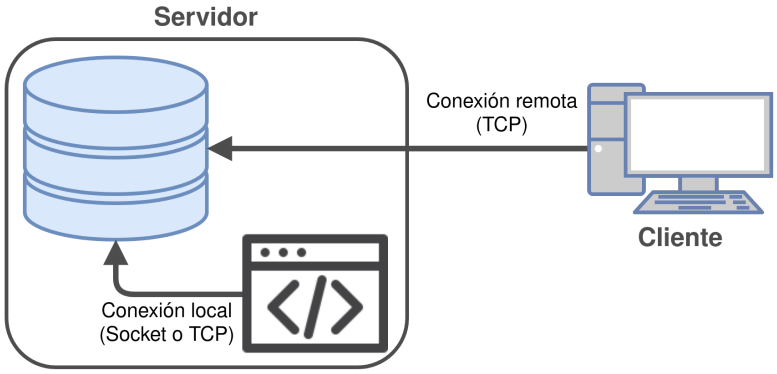
2.1. Antes de empezar

MySQL, al igual que otros SGBD y servidores en general, y más cuando hablamos de Software Libre, cuenta con una documentación online realizada por los creadores del software con la que nos tenemos que familiarizar.

El manual de referencia de MySQL cuenta con mucha información acerca del servicio, de la configuración y administración, pero también de cómo utilizar el lenguaje SQL. Por lo tanto, es obligatorio tener soltura buscando información en él.

2.2. Arquitectura Cliente → Servidor en MySQL

MySQL funciona en "modo servidor" esperando a las conexiones de un cliente, lo que comúnmente se denomina "arquitectura Cliente → Servidor".



El cliente que efectúa la conexión puede encontrarse en la misma máquina donde está situado el servidor (conexión local) o desde una máquina externa (conexión remota).

2.2.1. Conexión local (Socket)

En entornos UNIX existe la posibilidad de realizar la conexión a ciertos servidores que están en la misma máquina desde la que se origina la conexión, mediante lo que se denomina un **Unix domain Socket**.

Los sockets en entornos GNU/Linux se pueden ver como un fichero, y **son un medio de comunicación entre procesos que se ejecutan en la misma máquina**.

La configuración estándar de MySQL arranca creando un fichero Socket en la siguiente ruta por defecto `/var/run/mysqld/mysqld.sock`, por lo que la posibilidad de realizar una conexión local mediante dicho socket es posible.

De hecho, nada más realizar la instalación de MySQL es la única manera de poder realizar la conexión y sólo será posible desde el usuario root:

```
mikeldi@ubuntu:~$ sudo su
root@ubuntu:/home/mikeldi# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 8.0.21-0ubuntu0.20.04.4 (Ubuntu)

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Como se puede ver, en la imagen, los pasos para poder realizar la conexión han sido:

1. Convertirnos en root
2. Ejecutar el comando `>_ mysql`.

Este comando es el **cliente** MySQL que realizará la conexión contra el Servidor MySQL. Debido a que no se le ha pasado ningún parámetro al comando, éste realizará un primer intento de conexión a la ruta del socket estándar (indicada anteriormente).

Una vez realizada la conexión veremos que nos aparece un *prompt* `mysql\>` que indica que la conexión se ha realizado correctamente y estamos en el CLI (*Command Line Interface*, Interfaz de Línea de Comandos) donde podremos ejecutar órdenes de configuración, administración o peticiones a las bases de datos.

2.2.2. Conexión por red (local o remota)

En la configuración inicial de MySQL también aparece la opción de poder realizar una conexión mediante el protocolo de red **TCP**.

>_ Ver puertos de MySQL

```
root@server1:~# ss -punta | grep -i mysql
tcp    LISTEN 0      151      127.0.0.1:3306    0.0.0.0:*      users:(("mysqld",pid=765,fd=34))
tcp    LISTEN 0      70             *:33060         *:*      users:(("mysqld",pid=765,fd=31))
```

Los puertos de escucha son:

- **3306**: Puerto que escucha únicamente en la IP 127.0.0.1
- **33060**: Nuevo protocolo X (desde la versión 5.7.12) que escucha en todas las IPs del sistema.

Para poder realizar conexiones a través de estos puertos los usuarios con los que intentemos conectarnos deberán poder aceptar dicha conexión. Más adelante veremos cómo realizar estas modificaciones.

Para poder realizar esta conexión también haremos uso del comando `>_ mysql`. pero esta vez sí le tendremos que pasar parámetros de conexión. Un ejemplo de cómo conectarnos a un servidor por conexión de red es:

>_ Conexión por red a MySQL

```
ruben@server1:~$ mysql -u usuario -h 192.168.1.100 -p
```

Podemos ver que se han pasado 3 parámetros y a continuación el valor que se le pasa a cada parámetro en el comando `>_ mysql`. Los parámetros son:

- **-u** o **--user**, es el usuario con el que vamos a realizar la conexión. Si no se usa éste parámetro, por defecto se le pasará el usuario con el que estamos logueados en el sistema
- **-h** o **--host**, es el servidor al que queremos conectarnos mediante conexión por red. Si no se pone este parámetro la conexión se intentará a 127.0.0.1
- **-p** o **--password**, para que nos pregunte por la contraseña de conexión del usuario. Si no se pasa este parámetro se entiende que no se va a necesitar contraseña.

2.3. Arranque, parada y estado del Servidor

Una vez instalado MySQL tenemos que conocer cuál es su estado y conocer los procedimientos de arranque y parada del servicio.

2.3.1. Comprobar estado del servidor

En la instalación de MySQL se ha instalado automáticamente un servicio en Systemd que nos permitirá conocer el estado de MySQL, si está arrancado o no.

Para ello deberemos ejecutar el siguiente comando:

Estado del servidor MySQL

```
ruben@server1:~$ sudo systemctl status mysql
```

Y tras realizar la ejecución del comando, en caso de que el servicio esté arrancado veremos:

```
mikeldi@ubuntu:~$ sudo systemctl status mysql
[sudo] password for mikeldi:
● mysql.service - MySQL Community Server
     Loaded: loaded (/lib/systemd/system/mysql.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2020-09-01 18:20:31 UTC; 21h ago
   Main PID: 1730 (mysqld)
     Status: "Server is operational"
      Tasks: 39 (limit: 4620)
     Memory: 333.0M
     CGroup: /system.slice/mysql.service
             └─1730 /usr/sbin/mysqld

sep 01 18:20:30 ubuntu systemd[1]: Starting MySQL Community Server...
sep 01 18:20:31 ubuntu systemd[1]: Started MySQL Community Server.
```

2.3.2. Arranque y parada del servidor

En caso de que queramos parar el servidor, tendremos que hacer uso de Systemd:

Parar el servidor MySQL

```
ruben@server1:~$ sudo systemctl stop mysql
```

Y para arrancarlo de nuevo:

Arrancar el servidor MySQL

```
ruben@server1:~$ sudo systemctl start mysql
```

2.4. Primeros pasos

Como suele ser habitual cuando instalamos un servidor, viene con una configuración genérica que dista mucho de ser la ideal en un entorno productivo, y es por ello que tenemos que conocer cómo realizar modificaciones en la configuración para obtener mejores resultados.

Para poder administrar y configurar de manera correcta MySQL tendremos que conocer, al menos, lo siguiente:

- Especificaciones hardware donde está el servidor:
  - Cantidad de memoria RAM
  - Espacio en disco duro
  - Velocidad de escritura/lectura a disco duro
- Tipo de aplicación que va a utilizar MySQL
- Conexiones esperadas
- Cantidad de usuarios que se van a conectar
- Origen de las conexiones
- …

Con esta información podremos realizar un primer análisis para intentar prever si el servidor donde está el servicio MySQL corriendo es suficiente o no. Si la instalación nos corresponde a nosotros, tendremos que conocer parte de las preguntas planteadas anteriormente para realizar la instalación en un servidor que vaya a poder adecuarse a las exigencias pedidas.

2.4.1. Securizar configuración inicial

Dependiendo del método elegido para instalar MySQL la configuración inicial puede venir con usuarios o bases de datos de prueba que no nos interesan y que se pueden borrar. Para poder securizarlo tenemos el siguiente comando:

Securizar la instalación

```
root@server1:~# mysql_secure_installation
```

Este script nos guiará con una serie de preguntas para securizar el servidor, que es muy utilizado en versiones anteriores, pero que no está de más ver las preguntas que nos realiza:

- "**Would you like to setup VALIDATE PASSWORD component?**": Habilita el componente de validación de contraseñas, que sirve para habilitar test de fuerza de las contraseñas, y en caso de no ser lo suficientemente seguras, no se permite crear el usuario con esa contraseña.

  ¡Atención!

  En versiones anteriores era un PLUGIN

  Si lo habilitamos, no preguntará qué tipo de política queremos activar sobre las contraseñas:

  - **LOW**: contraseñas de longitud mínima 8
  - **MEDIUM**: como las LOW pero con números, caracteres especiales y mayúsculas y minúsculas.
  - **STRONG**: como las MEDIUM pero que no se encuentren en un

- Nos pedirá la **contraseña del administrador "root"** de MySQL. En la instalación de Ubuntu el usuario root se conecta mediante socket, por lo que este paso no cambiará nada.
- "**Remove anonymous users?**": En algunas distribuciones se crean unos usuarios anónimos, que deberían ser borrados.
- "**Disallow root login remotely?**": De nuevo, en Ubuntu el usuario root sólo se puede conectar en local mediante el socket, pero en otras distribuciones es posible que root pueda conectarse por TCP desde otros servidores y/o equipos. Lo recomendable suele ser que para root no se permitan conexiones remotas.
- "**Remove test database and access to it?**": MySQL suele venir con una base de datos de prueba llamada "test". Se puede borrar.
- "**Reload privilege tables now?**": Recargar los permisos de los privilegios, ya que hemos realizado cambios en las contraseñas.

2.5. Ficheros de configuración

Como es habitual en GNU/Linux, los ficheros de configuración de los servicios están situados en el path `/etc` en este caso concreto dentro de la ruta `/etc/mysql` MySQL puede leer distintos ficheros de configuración, por lo que dependiendo de la distribución puede variar cuál tengamos, pero el orden suele ser:

- `/etc/my.cnf`: Es el primer fichero de configuración que se busca (en Ubuntu no lo tendremos)
- `/etc/mysql/my.cnf` : Fichero principal de configuración. En Ubuntu es un enlace simbólico a `/etc/alternatives/my.cnf` que a su vez es un enlace simbólico a `/etc/mysql/mysql.cnf`. Esto es debido a que en versiones antiguas esta última ruta era la oficial.

Si vemos este fichero de configuración veremos que tiene una directiva "**!includedir**", esto significa que se van a incluir todos los ficheros con extensión "**cnf**" de esos directorios.

Estos ficheros de configuración tienen el formato conocido como "INI" y suele ser de este aspecto:

Ejemplo de fichero de configuración por secciones

```
[section]
option1    = value

[section2]
option2     = value
```

Debido a que durante la instalación de MySQL se han instalado otras herramientas, la sección determinará

para qué servicio o aplicación será esa parte de la configuración:

- **mysqld**: servidor MySQL (**mysql d**aemon)
- **mysql**: cliente de consola para realizar la conexión
- **mysqldump**: programa para realizar backups de las bases de datos.

Cuando realicemos modificaciones en la configuración de la configuración, si es para el servidor de MySQL tendremos que realizar un reinicio del servicio para que estas modificaciones sean tenidas en cuenta.

2.5.1. Analizando la configuración inicial

Tal como hemos comentado previamente, los ficheros de configuración en `/etc/mysql` pueden estar en distintos ficheros, por lo que es recomendable echar un ojo a los ficheros que tenemos tras realizar la instalación. Vamos a analizar parte del fichero: `/etc/mysql/mysql.conf.d/mysqld.cnf`:

Ejemplo de fichero de configuración de MySQL

```
#
# The MySQL database server configuration file.
# ...
[mysqld]
#
# * Basic Settings
#
user            = mysql
# pid-file      = /var/run/mysqld/mysqld.pid
# socket        = /var/run/mysqld/mysqld.sock
# port          = 3306
# datadir       = /var/lib/mysql
bind-address      = 127.0.0.1

#
# * Fine Tuning
#
key_buffer_size         = 16M
# max_connections        = 151

#
# * Logging and Replication
#
# Both location gets rotated by the cronjob.
#
```

```
# Log all queries
# Be aware that this log type is a performance killer.
# general_log_file        = /var/log/mysql/query.log
# general_log             = 1
#
# Error log - should be very few entries.
#
log_error = /var/log/mysql/error.log
#
# Here you can see queries with especially long duration
# slow_query_log        = 1
# slow_query_log_file   = /var/log/mysql/mysql-slow.log
# long_query_time = 2
# log-queries-not-using-indexes
```

Como se puede comprobar, en el fichero se pueden comprobar distintos tipos de filas, y el usar un editor que realice resaltado de sintaxis nos puede ayudar a diferenciar las líneas y lo que son. Vamos a realizar una pequeña explicación de varias líneas:

- Líneas que empiezan por "**#**: Al igual que sucede con otros ficheros de configuración, en este caso las líneas que empiezan por almohadilla, o"#", son comentarios. Estas líneas son ignoradas y nos pueden ayudar a comprender la estructura del fichero o podremos poner comentarios para explicar para qué sirve una directiva de configuración.
- **[mysqld]**: Es el comienzo de la sección, y como se ha dicho previamente, esta sección será la que se utilice al arrancar MySQL. Por lo tanto, toda directiva de configuración dentro de esta sección servirá para modificar el comportamiento por defecto del servidor.
- **user**: indica al servidor MySQL con qué usuario arrancará el servidor
- **bind-address**: dirección en la que escuchará el servidor MySQL al arrancar
- **log_error**: fichero donde aparecerá lo acontecido durante el arranque del servicio y en caso de haber errores nos indicará parte de la razón.

Otras variables que aparecen comentadas en la configuración inicial y por tanto el valor que tiene esa opción de configuración no tiene por qué ser la que se esté utilizando actualmente:

- **pid-file**: fichero donde se escribirá el PID (*Process IDentificator*) durante el arranque.
- **socket**: fichero donde está el socket.
- **port**: puerto en el que escucha MySQL.
- **datadir**: lugar donde se almacena todos los datos de las bases de datos.

- **max_connections**: número máximo de conexiones que aceptará MySQL
- **general_log_file**: lugar donde se escribe el log general en caso de estar activo.
- **general_log**: activar, o no, el log general. Como se puede comprobar en un comentario previo, puede suponer un problema en el rendimiento del servidor.
- **slow_query_log**: log de queries lentas.
- **long_query_time**: el tiempo mínimo para que una slow query se loguee.
- **log-queries-not-using-indexes**: loguear las queries que no usan índices.

Como se puede ver en estas variables de configuración, y en las anteriores, el nombre de las mismas suele ser bastante auto-explicativo, por lo que nos podemos hacer una pequeña idea de lo que pueden hacer leyendo el nombre. Eso no quita para que en caso de duda, debemos consultar el listado completo de variables que puede llegar a tener MySQL.

Como se ha comentado previamente, en caso de realizar una modificación en el fichero de configuración, tendremos que realizar un reinicio del servicio como veremos posteriormente.

2.5.2. Validando la configuración

Las variables de configuración que hemos visto hasta ahora cuentan con dos apartados:

- **Nombre de la variable**: puede ser una única palabra (**port**, **socket**) o varias seguidas de guión bajo (**bind_address**, **slow_query_log**).
- **Valor de la variable**: Es el valor que aparece a la derecha del igual (“=”) y vemos que pueden tener distintos valores: mysql, 127.0.0.1, 16M , …

Los nombres de las variables es algo que no nos podemos inventar, y es por ello por lo que deben estar bien escritas en el fichero de configuración. En caso de escribir una variable que no exista, el servidor no arrancará si hacemos un reinicio del mismo. Ejemplo:

Ejemplo de log de error

```
2020-08-01T11:35:50.488392Z 0 [ERROR] [MY-000067] [Server] unknown variable 'mal=32M'
```

Como se puede ver en esta línea de log, hay un error al haber escrito una variable desconocida (en este caso el nombre de la variable es “mal”). Esto hace que el servidor no arranque y por tanto no se pueda acceder a él y haya una pérdida de servicio. Esto no debe suceder en un servidor en producción.

Para evitar este problema, antes de realizar cualquier reinicio del servidor, debemos asegurarnos que el fichero de configuración no contiene variables desconocidas:

Confirmar estado de la configuración

```
root@ubuntu:/etc/mysql/mysql.conf.d# mysqld --validate-config

2020-08-01T12:02:41.763658Z 0 [ERROR] [MY-000067] [Server] unknown variable 'mal=32M'
2020-08-01T12:02:41.763689Z 0 [ERROR] [MY-010119] [Server] Aborting
```

Esta opción nos puede ser útil si realizamos una actualización de MySQL y en la nueva versión ya no existen variables de versiones anteriores (es raro, pero puede haber opciones obsoletas).

¡Cuidado!

Esta validación de configuración no hace una comprobación exhaustiva del valor adjudicado a la variable

Esta validación de configuración no hace una comprobación exhaustiva del valor adjudicado a la variable, por lo que en caso de que la variable sea correcta pero el valor adjudicado sea erróneo, al reiniciar el servidor tendremos que comprobar el log de errores para ver qué sucede. Por ejemplo, de tener lo siguiente en la configuración:

Error puesto en la configuración

```
bind-address            = 127.0.0.1z
```

Al intentar reiniciar, veremos el siguiente error en `/var/log/mysql/error.log`:

Error en el log

```
[ERROR] [MY-010255] [Server] Can't create IP socket: Invalid argument
```

En cambio, si otras variables tienen un valor incorrecto, el servidor arrancará, y el valor que cogerá la variable dependerá de cómo se parsea el fichero de configuración. Por ejemplo:

Error en la configuración

```
key_buffer_size         = 16Mz
max_connections         = -1
```

Para conocer qué valor pueden tener estas variables, lo mejor es ir a la documentación y mirar la variable que queremos modificar o agregar. Por ejemplo, para slow_query_log veremos que en la documentación tiene la siguiente tabla:

Referencia	Valor
Command-Line Format	–slow-query-log=ON

Continúa en la siguiente página

System Variable	slow_query_log
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

Esta tabla nos da mucha información acerca de la variable, ya sea para poder ver qué tipo de valor es (Boolean en este caso), su valor por defecto (OFF) y si podremos o no modificarla dinámicamente (Dynamic = Yes). El valor por defecto será el valor que obtendrá la variable si no aparece en el fichero de configuración con un valor puesto por nosotros.

Otro ejemplo, bind_address:

Referencia	Valor
Command-Line Format	–sbind-addr=addr
System Variable	sbind_address
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	String
Default Value	

2.6. Variables de configuración en MySQL

Tal como hemos visto, el listado completo de variables tiene cientos de posibles variables que podemos ver o modificar para ajustar el servidor a nuestros intereses. Las variables se deben separar en dos partes:

- **System Var**: Variables que sirven para el arranque del servidor o para ajustar el comportamiento del mismo. Se pueden modificar.
- **Status Var**: Variables para comprobar el estado del servidor. Sólo las podremos leer.

Las variables que podremos modificar se pueden pasar por:

- **Fichero de configuración**: la versión más cómoda, tal como hemos visto previamente. Hará que nuestro servidor siempre arranque con la misma configuración ya que los ficheros estarán escritos en el servidor.
- **Línea de comandos**: al arrancar mysqld, se le pueden pasar como parámetros estas variables con los valores que queramos. Se nos puede olvidar pasar algún parámetro.

Hay variables del sistema que si nos fijamos en la tabla no se puede pasar por fichero o por línea de comandos. Esto es debido a que son variables internas de cómo ha sido compilado MySQL o que tienen en cuenta las librerías del sistemas. Por ejemplo: **version**, **have_ssl**.

Si nos fijamos en la columna "**Var Scope**" vemos que puede tener los valores:

- **Global** : son variables que afectan a las nuevas conexiones que se realizan.
- **Session** : son variables que afectan a la sesión actual. Si intentamos leer una variable de sesión que no existe, nos mostrará el valor global.
- **Both** : son variables que existen en ambos estados, global y en sesión, y pueden diferir.
- **Varies** : Hay unas pocas variables que tienen esta opción (y también en la columna Dynamic) y nos indica que varía dependiendo de la versión de MySQL que estemos usando.

2.6.1. Comprobando las variables de estado (STATUS)

El listado de variables de estado nos muestra el nombre de las variables por las que podemos preguntar, el tipo de valor que pueden tener y si son globales o de sesión.

Para comprobar el estado de las variables globales del servidor:

Ver variables de estado

```
mysql> SHOW GLOBAL STATUS;
```

Si nos interesa buscar por el nombre de alguna variable de la sesión y obtener su estado:

Ver variables de sesión

```
mysql> SHOW SESSION STATUS LIKE '%slow%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Slow_launch_threads        | 0     |
| Slow_queries               | 0     |
+----------------------------+-------+
2 rows in set (0.01 sec)
```

2.6.2. Comprobando las variables del sistema (System)

Hemos visto que en los ficheros de configuración no están todas las variables que hemos visto en la documentación, y no tendría mucho sentido tener que estar yendo a ella para mirar qué variables tenemos y sus posibles valores por defecto. Para ver las variables globales del sistema, **es obligatorio poner GLOBAL**:

Ver variables globales

```
mysql> SHOW GLOBAL VARIABLES;
```

Si queremos buscar por una variable en concreto en la sesión, poner SESSION es opcional (en lugar de GLOBAL).

De no poner en qué ámbito queremos mirar las variables, se presupone que queremos ver el estado de la variable en la sesión actual.

Si queremos buscar una variable que contenga algo en el nombre podremos hacer uso de:

Ver variables globales

```
mysql> SHOW VARIABLES like '%slow%';
+---------------------------+---------------------------------+
| Variable_name             | Value                           |
+---------------------------+---------------------------------+
| log_slow_admin_statements | OFF                             |
| log_slow_extra            | OFF                             |
| log_slow_slave_statements | OFF                             |
| slow_launch_time          | 2                               |
| slow_query_log            | OFF                             |
| slow_query_log_file       | /var/lib/mysql/ubuntu-slow.log  |
+---------------------------+---------------------------------+
6 rows in set (0.01 sec)
```

De esta manera buscamos las variables que nos puedan interesar y vemos el valor que tienen actualmente en la sesión, para así poder ver si nos interesa cambiarlo.

También podemos hacer uso desde la línea de comandos con el comando `mysqladmin`, pero no tenemos tanta libertad como desde el CLI de MySQL :

Ver variables desde la consola

```
root@server1:~# mysqladmin  variables
```

2.6.3. Modificar configuración "en caliente"

Debido a que un SGBD es un servicio crítico, no siempre podremos realizar una parada de servicio debido a un cambio de configuración. Esto afectaría a todas las conexiones que tiene establecido el servidor, y cualquier intento de consulta sería rechazado durante la parada y el arranque del mismo.

No sólo eso, ya que cuando un SGBD está en funcionamiento, guarda información en memoria RAM (a modo de caché para las peticiones más realizadas, traspasa parte de las tablas a memoria, ...) y en caso de realizar un reinicio, esa memoria se liberaría y por tanto al arrancar se debería volver a pedir los datos a disco duro, haciendo que los primeros instantes de servicio sea lento.

Por ello, **se pueden realizar modificaciones de la configuración sin parar el servicio**, lo que se suele denominar "hacer modificaciones en caliente".

Bien es cierto que no todas las variables se pueden modificar de esta manera, ya que algunas son necesarias durante el arranque del servicio. Por ejemplo:

- **port**: no podrá ser modificada "en caliente", ya que el puerto debe de ser algo que se necesita durante el arranque.
- **bind_address**: lo mismo que la anterior.
- **slow_query_log**: si se puede modificar en caliente.

Para conocer si una directiva puede ser modificada en tiempo de ejecución podremos ir al listado completo de variables y comprobar la columna "Dynamic", si aparece "Yes" es que se podrá modificar. Anteriormente hemos visto cómo la variable slow_query_log tiene la opción "Dynamic" a "Yes", por lo que se puede modificar el valor dinámicamente, y para cambiarlo haremos:

Cambiar variable

```
mysql> SET slow_query_log = On;
ERROR 1229 (HY000): Variable 'slow_query_log' is a GLOBAL variable
and should be set with SET GLOBAL


mysql> SET GLOBAL slow_query_log = On;
Query OK, 0 rows affected (0.01 sec)

-- alternativa
mysql> SET @@GLOBAL.slow_query_log = On;
Query OK, 0 rows affected (0.01 sec)
```

Como se puede comprobar, se ha intentado modificar la variable sin poner "GLOBAL", y debido a que "slow_query_log" es una variable global, nos ha dado un error. Por eso **es muy importante tener en cuenta**

qué tipo de variable y para qué ámbito queremos realizar la modificación. También se puede ver que la modificación se puede realizar de dos maneras.

¡Atención!

Hay que tener en cuenta que la variable modificada en caliente no se guarda en la configuración y por tanto se perderá en el siguiente reinicio.

2.6.4. Persistencia de las modificaciones

MySQL permite que estas variables que podamos hacer que estas variables persistan tras el arranque haciendo:

Cambiar variable de manera persistente

```
mysql> SET PERSIST slow_query_log  = ON;
```

Esto lo que hace es crear un fichero en `/var/lib/mysql/mysqld-auto.cnf` con formato JSON con las variables que se ha dado la orden de persistir.

¡Cuidado!

No se aconseja utilizar este método. Mejor modificar la configuración.

Aunque resulte cómodo, se aconseja realizar la modificación de la variable en el fichero de configuración correspondiente para tener todo de manera más centralizada. Para deshacer esta persistencia:

Eliminar las variables persistentes

```
mysql> RESET PERSIST;
Query OK, 0 rows affected (0.00 sec)
```

Y eliminará todas las entradas del fichero antes descrito: `/var/lib/mysql/mysqld-auto.cnf`.

III

SQL

1. SQL básico

1.1. Introducción

SQL (por sus siglas en inglés ***S**tructured **Q**uery **L**anguage*) surgió como una evolución de un lenguaje previo llamado SEQUEL. SQL nació en IBM en 1977. Unos años más tarde fue considerado un estándar ANSI y al año siguiente (1987) un estándar ISO.

SQL tiene las siguientes características:

- **Lenguaje de definición de datos (DDL)**: nos permite crear y/o manipular la estructura de la base de datos y las relaciones que en ellas deben existir.
- **Lenguaje de manipulación de datos (DML)**: este apartado se puede subdividir en las siguientes tareas:
  - Consultar determinada información.
  - Insertar nuevos datos.
  - Modificar los datos existentes.
  - Eliminar datos que ya no sean necesarios.
- **Control de transacciones**: para definir cuándo se inicia, cuándo finaliza o si es necesario deshacer una transacción.
- **Control de datos**: permite asignar y revocar permisos a los usuarios, tanto de acceso a la información como de operaciones que pueden realizar sobre los mismos.

El lenguaje SQL puede ser utilizado:

- De forma **interactiva**: en una consola o desde un CLI instalado en la propia máquina donde se encuentra el servidor u otra máquina remota.
- **Inmerso** dentro de un lenguaje de programación anfitrión, ejecutando consultas desde Java o PHP, por ejemplo.

Es un **lenguaje declarativo** ya que permite especificar la operación que se debe llevar a cabo (en forma de sentencias), pero no la forma en la que se llevará a cabo. Esa será la tarea del SGBD, ejecutar la tarea de la manera más adecuada para obtener los datos.

Los SGBD implementan (de manera parcial o total) el estándar SQL y sobre él añaden funcionalidades propias e incluso lenguajes de programación propios derivados de SQL (PL/SQL en Oracle, PL/pgSQL en PostgreSQL. Por lo tanto, **SQL no es un lenguaje de programación, pero hay SGBDs que han creado dicha funcionalidad**.

Debido a que SQL es un estándar, las funciones principales deberían funcionar en cualquier SGBD que haga uso del estándar, aunque es cierto que algunos SGBD amplían SQL con funciones o procedimientos propios

que otros SGBD no tienen porqué tener. Esas funciones o procedimientos extra a veces se incorporan al estándar un tiempo después.

Dentro del lenguaje SQL podemos separar las consultas o acciones teniendo en cuenta qué realizamos con ellos.

1.2. Lenguaje de definición de datos: DDL

El DDL (*Data Definition Language*) de SQL proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación. Existen tres operaciones básicas: **CREATE**, **ALTER** y **DROP**.

1.2.1. Gestión de bases de datos

La función de un SGBD es la de proporcionar la capacidad de tener distintas bases de datos en el sistema, es por ello que tenemos que aprender a gestionar las bases de datos que tenemos, o en un momento dado crear bases de datos.

Crear una base de datos y mostrar las existentes

```
mysql> CREATE DATABASE prueba;

mysql> SHOW DATABASES;
+----------------------------+
| Database                   |
+----------------------------+
| information_schema         |
| mysql                      |
| performance_schema         |
| prueba                     |
| sys                        |
+----------------------------+
5 rows in set (0.00 sec)
```

Para poder entrar en una base de datos y poder visualizar, modificar o usar las tablas y datos que hay en ella:

"Entrar" en una base de datos

```
mysql> USE prueba;
Database changed
```

Borrar una base de datos existente.

Borrar una base de datos

```
mysql> DROP DATABASE prueba;
```

1.2.2. Gestión de tablas

La gestión de tablas se realiza cuando estamos dentro de una base de datos, por lo que deberemos asegurarnos que hemos entrado dentro de una de ellas antes de realizar ninguna gestión sobre ellas.

Crear distintas tablas

```
CREATE TABLE alumnos (
    dni       varchar(9) not null primary key,
    nombre    varchar(20),
    apellido varchar (20),
    telefono integer
);

CREATE TABLE asignaturas (
    codigo integer not null primary key,
    nombre varchar(50)
);

CREATE TABLE matriculacion_alumno (
    codigo_asignatura integer,
    dni_alumno varchar(9),
    nota float,
    fecha_matriculacion date,

    PRIMARY KEY(codigo_asignatura, dni_alumno),
    FOREIGN KEY (codigo_asignatura) REFERENCES asignaturas(codigo),
    FOREIGN KEY (dni_alumno) REFERENCES alumnos(dni) ON DELETE CASCADE
);
```

Si queremos modificar el modelo de la tabla creada previamente

Modificar una tabla, añadiendo una columna

```
mysql> ALTER TABLE alumnos ADD nacimiento date;
```

Eliminar una tabla

```
mysql> DROP TABLE alumnos;
```

TRUNCATE es una función especial que vacía de contenido una tabla. Realmente lo que hace es borrar la tabla y volverla a crear, de ahí que sea considerada una función dentro del DDL.

Truncar una tabla

```
mysql> TRUNCATE TABLE alumnos;
```

Para poder gestionar una tabla debemos saber cómo es y cómo está creada, ya que dependiendo de ello podremos realizar modificaciones sobre la misma.

Para ver las columnas que tiene una tabla podemos hacerlo de dos maneras:

Ver columnas de una tabla

```
mysql> SHOW COLUMNS FROM alumnos;
mysql> DESCRIBE alumnos;
```

Podemos obtener más datos, como los privilegios que tenemos con el usuario que nos hemos conectado sobre cada columna haciendo:

Ver permisos sobre columnas

```
mysql> SHOW FULL COLUMNS FROM alumnos;
```

Y si queremos ver cómo se ha creado la tabla:

Ver cómo se ha creado una tabla

```
mysql> SHOW CREATE TABLE alumnos;
```

1.3. Lenguaje de manipulación de datos: DML

El DML (Data Manipulation Language) nos permite consultar, manipular, insertar o eliminar los datos.

1.3.1. Realizar consultas

A la hora de realizar consultas, podemos crear distintas y de distinto tipo.

Consultar todos los registros de una tabla

```
mysql> SELECT * FROM alumnos;
```

Consultar ciertos campos de una tabla

```
mysql> SELECT dni, nombre, nacimiento FROM alumnos;
```

También podemos realizar consultas condicionales:

Realizar consulta condicional con una fecha

```
mysql> SELECT dni, nombre, nacimiento
            FROM alumnos
            WHERE nacimiento > '1982-01-01';
```

Realizar consulta condicional y ordenada

```
mysql> SELECT dni, nombre, nacimiento
            FROM alumnos
            WHERE nacimiento > '1970-01-01'
            ORDER BY dni ASC;
```

La complejidad de las consultas dependen de nuestro modelo de datos así como de los datos que queremos obtener:

Consulta relacionando varias tablas

```
mysql> SELECT a.dni, a.nombre, a.apellido, asig.nombre, ma.nota
            FROM alumnos as a, asignaturas as asig, matriculacion_alumno as ma
            WHERE a.dni = ma.dni_alumno
                AND asig.codigo = ma.codigo_asignatura
                AND ma.nota < 5
            ORDER BY a.dni ;
```

1.3.2. Inserción y modificación de datos

No sólo podemos realizar consultas de obtención de datos, sino que también podremos insertarlos, modificarlos o borrarlos.

Insertar datos en una tabla

```
mysql> INSERT INTO alumnos VALUES ('12345678Z', 'Alumno', 'Uno', '555123456','1980-01-01');
```

En caso de que sólo queramos insertar parte de los datos, podremos realizar:

Insertar sólo datos obligatorios en una tabla

```
mysql> INSERT INTO alumnos(dni,nombre) VALUES ('87654321A', 'Alumno2');
```

A la hora de actualizar datos, deberemos elegir qué campos queremos elegir

Actualizar ciertos campos en una tabla

```
mysql> UPDATE alumnos SET nacimiento = '1984-01-01' WHERE dni = '87654321A';
```

Y a la hora de eliminar datos, podremos hacerlo de manera condicional.

Eliminar ciertos registros de una tabla

```
mysql> DELETE FROM alumnos where dni = '87654321A';
```

1.4. Control de transacciones

Como ya se ha visto previamente, los SGBDs deben de seguir el modelo ACID de transacciones para asegurar que los datos mantienen la integridad de los datos y el aislamiento a la hora de realizar transacciones sobre los mismos datos.

Por ello, SQL permite realizar inicios de transacciones ***BEGIN***) para posteriormente aplicar todos los cambios que se han realizado (***COMMIT***) o cancelarlos (***ROLLBACK***).

¡Atención!

MySQL utiliza el método **autocommit** por defecto

Empezar una transacción

Nos sirve a la hora de realizar grandes modificaciones en la base de datos. Se puede realizar de las dos maneras siguientes:

Hacer una transacción (se puede hacer de 2 maneras)

```
mysql> BEGIN;
-- alternativa:
mysql> START TRANSACTION;
```

A partir de aquí cualquier acción que se realice no será aplicada (aunque sí lo podremos ver en la sesión actual) hasta que no se termine la transacción.

Aplicar y terminar la transacción

Para aplicar los cambios realizados a lo largo de la transacción, y para que los cambios sean persistentes, se deberá ejecutar:

Aplicar y terminar la transacción

```
mysql> COMMIT;
```

Tras esto, los cambios serán realizados y el resto de conexiones verán las modificaciones realizadas.

Cancelar transacción

En caso de que queramos cancelar las ejecuciones realizadas desde el inicio de la transacción, deberemos ejecutar:

Cancelar la transacción y todo lo realizado en ella

```
mysql> ROLLBACK;
```

Modificar comportamiento autocommit

El comportamiento por defecto de MySQL es que cualquier ejecución es una transacción, por lo que realizará el bloqueo de datos necesario o realizará la modificación de los datos. Si queremos cambiar este comportamiento para la sesión actual deberemos hacer:

Modifiar el comportamiento del autocommit

```
mysql> SET autocommit = OFF;
```

IV

Gestión de usuarios

1. Gestión de usuarios

MySQL permite la creación de cuentas que habilitan la conexión de clientes de usuario al servidor y el acceso a los datos que gestiona el servidor (ya sea conexiones mediante CLI o aplicación).

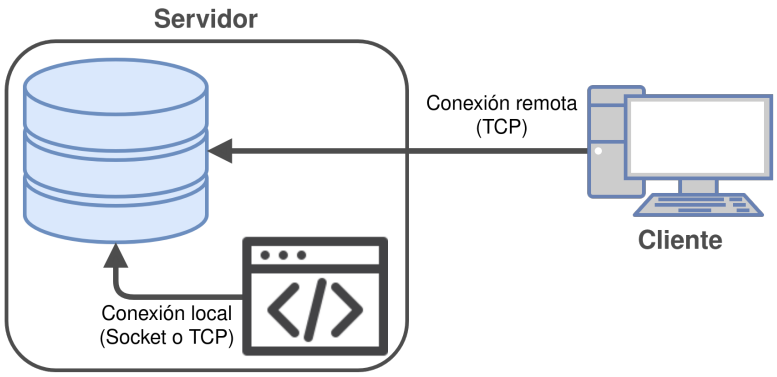
La función principal del sistema de privilegios de MySQL es la de permitir la conexión que se conecta desde un equipo remoto y asociarla a un usuario con permisos sobre una base de datos para que pueda ejecutar funciones tal que SELECT, INSERT, UPDATE y DELETE. Un usuario también puede recibir permisos para realizar funciones privilegiadas.

Para controlar qué usuarios pueden conectarse, cada cuenta tendrá asignada un credencial de autenticación, como puede ser la contraseña. Las funciones sobre usuarios más habituales son: CREATE USER, GRANT y REVOKE.

El sistema de privilegios de MySQL asegura que **los usuarios sólo pueden realizar las funciones que le son permitidas**. Cuando un usuario se conecta al servidor MySQL **su identidad es determinada por**:

- El equipo remoto desde el que se realiza la conexión: no es lo mismo que la conexión se realice desde la IP local, desde una IP 192.168.1.100 o desde 172.26.20.2
- El nombre de usuario que se haya especificado.

Cuando se realice la conexión, el sistema otorga privilegios de acuerdo con la identidad del usuario y desde donde se recibe la conexión, permitiendo, o no, ciertas acciones sobre los datos.



1.1. Cuentas de usuarios y contraseñas

MySQL guarda las cuentas de usuario y sus contraseñas en la tabla **user** de la base de datos del sistema **mysql**. Como ya se ha indicado, una cuenta se define con el nombre de usuario y el host desde el que se establece la conexión. MySQL soporta distintos tipos de autenticación, no sólo contraseñas. La autenticación está gestionado por el plugin de autenticación de cuentas. Algunos tipos de autenticación que soporta MySQL 8:

- Contraseña
- Autenticación PAM (para Linux y Mac)
- LDAP

- Autenticación mediante socket (conexiones locales)

1.1.1. Creación de usuarios

Como ya se ha explicado, es importante entender que para que la conexión sea correcta no sólo importa el usuario de conexión, sino desde donde se realiza la misma. Es por ello que a la hora de crear un usuario tendremos que conocer este dato ya que es importante. Aparte, tendremos que poner una contraseña, si es necesario, que sea lo suficientemente segura como para que pase el paso realizado en la securización inicial. A la hora de crear usuarios hay muchas opciones.

Crear usuario local (SOCKET)

El usuario solo se va a conectar desde la propia máquina, conexión mediante socket:

Crear usuario que sólo se conecte por socket

```
mysql> CREATE USER 'mikeldi'@'localhost' IDENTIFIED WITH auth_socket;
```

Crear usuario local (conexión TCP)

La conexión va a ser local, pero se va a hacer uso del protocolo TCP, y en este caso es necesario introducir contraseña.

Crear usuario que sólo se conecte por TCP

```
mysql> CREATE USER 'usuario'@'localhost' IDENTIFIED BY 'password';
```

Crear usuario para conexión remota (TCP)

Si queremos que la conexión se pueda realizar desde una IP concreta:

Crear usuario que sólo pueda conectarse desde una IP

```
mysql> CREATE USER 'usuario2'@'192.168.122.1' IDENTIFIED BY 'password';
```

Si queremos que un usuario se pueda conectar desde cualquier IP:

Crear usuario que pueda conectarse desde cualquier IP

```
mysql> CREATE USER 'usuario3'@'%' IDENTIFIED BY 'password';
```

Crear usuario y que se le expire la contraseña

El usuario se crea con la contraseña del comando, pero cuando se loguee no podrá realizar nada hasta que cambie la contraseña (ver más adelante):

Crear usuario y que la contraseña expire

```
mysql> CREATE USER 'usuario4'@'%' IDENTIFIED BY 'password' PASSWORD EXPIRE;
```

Si queremos que la contraseña le expire en 10 días:

Crear usuario y que la contraseña expire

```
mysql> CREATE USER 'usuario5'@'%' IDENTIFIED BY 'password' PASSWORD EXPIRE INTERVAL 10 DAY;
```

1.1.2. Conexión de usuarios

Dependiendo de cómo se realice la conexión (por socket o TCP) el método de conexión será de una manera u otra.

Conexión por socket

Para poder conectarnos por socket, tenemos que tener acceso a él, siendo lo habitual conexión local. El usuario utilizado será el usuario del sistema:

Conexión por socket desde el usuario root

```
root@server1:~# mysql
```

Conexión remota

Suponiendo que el servidor MySQL está en la IP 192.168.122.145, la conexión deberá realizarse de la siguiente manera (hay que tener en cuenta el origen de la conexión también).

Conexión al servidor 192.168.122.145

```
ruben@server1:~$ mysql -u usuario2 -h 192.168.122.145 -p
```

Conocer el usuario de conexión:

Para asegurarnos con qué usuario nos hemos conectado:

Ver el usuario con el que nos hemos conectado

```
mysql> select user();
+-----------------------+
| user()                |
+-----------------------+
| usuario2@192.168.122.1 |
+-----------------------+
1 row in set (0.00 sec)
```

Cambiar la contraseña del propio usuario:

Si nos conectamos con un usuario que tiene la opción “PASSWORD EXPIRE” activada, para cambiar la contraseña deberemos loguearnos con ese usuario y realizar:

Cambiar la contraseña

```
mysql> set password='password';
```

1.1.3. Ver los usuarios que existen

Los usuarios se guardan dentro de la tabla user en la base de datos mysql. Para poder ver los usuarios creados:

Mirar tabla de usuarios

```
mysql> use mysql;
mysql> select * from user;
```

Para poder ver de otra manera los datos de los usuarios:

Mirar la tabla en modo vertical

```
mysql> select * from user\G
*************************** 1. row ***************************
                  Host: %
                  User: usuario2
           Select_priv: N
           Insert_priv: N
           Update_priv: N
           Delete_priv: N
           Create_priv: N
...
           max_updates: 0
       max_connections: 0
  max_user_connections: 0
                plugin: caching_sha2_password
authentication_string: $A$005$lq-g:Ps27zVBGU8SnP7eg7WK3qnQ6R
     password_expired: N
password_last_changed: 2020-09-05 18:22:11
...
```

1.1.4. Limitando los recursos de las cuentas

Las cuentas de usuarios pueden tener recursos limitados. Estos límites se pueden poner al crear un usuario:

Limitar el número de queries por hora.

```
mysql> CREATE USER 'lim1'@'localhost' IDENTIFIED BY 'frank' WITH MAX_QUERIES_PER_HOUR 20;
```

O se pueden añadir los límites sobre cuentas ya creadas:

Limitar el número de conexiones a 2 para el usuario

```
mysql> ALTER USER 'usuario'@'localhost' WITH MAX_USER_CONNECTIONS 2;
```

Limitar el número de conexiones a 20 por hora

```
mysql> ALTER USER 'usuario'@'localhost' WITH MAX_CONNECTIONS_PER_HOUR 20;
```

Limitar el número de queries a 500 por hora

```
mysql> ALTER USER 'usuario'@'localhost' WITH MAX_QUERIES_PER_HOUR 500;
```

Limitar el número de updates a 500 por hora

```
mysql> ALTER USER 'usuario'@'localhost' WITH MAX_UPDATES_PER_HOUR 500;
```

1.1.5. Borrado de usuarios

Al borrar usuarios la sentencia “DROP USER” borra una o más cuentas y sus privilegios.

Al hacer un “DROP USER”, si el usuario tiene establecida una conexión no se borrará el usuario podrá seguir haciendo peticiones. Cuando la conexión se cierre será cuando se borre el usuario, y su siguiente conexión no se podrá establecer.

Al borrar un usuario no se pierden los objetos que el usuario haya creado.

Borrar usuario

```
mysql> DROP USER 'usuario'@'localhost';
```

1.1.6. Bloqueo de usuarios

Como borrar un usuario no se puede deshacer, podría ser recomendable bloquear antes la cuenta hasta asegurar al 100 % que el usuario que vamos a borrar no se va a volver a necesitar:

Bloquear un usuario

```
mysql> ALTER USER 'usuario'@'localhost' ACCOUNT LOCK;
```

1.1.7. Renombrar un usuario

Podemos renombrar un usuario, cambiando el usuario y el origen de la conexión. Si la cuenta ya existe obtendremos un error.

Renombrar un usuario

```
mysql> RENAME USER 'usuario'@'localhost' TO 'ruben'@'192.168.122.1';
```

1.1.8. Cambiar contraseña al usuario

Es interesante realizar modificaciones en las contraseñas de usuarios cada cierto tiempo.

Cambiar contraseña de un usuario

```
mysql> SET PASSWORD FOR 'ruben'@'192.168.122.1' = 'password2';
```

1.2. Privilegios de usuarios

Hasta ahora lo que hemos permitido es que un usuario pueda realizar una conexión, pero un usuario conectado no puede realizar ninguna acción sobre tablas ya que no tiene permisos sobre nada. La sentencia **GRANT** da permisos a las cuentas de usuarios y establece qué operaciones puede realizar.

En la documentación oficial podemos ver todo los privilegios que pueden tener los usuarios.

En la siguiente tabla se muestran sólo unos pocos de los privilegios, y siempre conviene ir a la documentación oficial teniendo en cuenta la versión de MySQL que estamos utilizando, para confirmar los privilegios que existen:

Privilege	Grant Table Column	Context
ALL PRIVILEGES	Synonym for “all privileges”	Server administration
ALTER	Alter_priv	Tables
CREATE	Create_priv	Databases, tables, or indexes
CREATE USER	Create_user_priv	Server administration
CREATE VIEW	Create_view_priv	Views
DELETE	Delete_priv	Tables
DROP	Drop_priv	Databases, tables, or views
DROP ROLE	Drop_role_priv	Server administration
GRANT OPTION	Grant_priv	Databases, tables, or stored routines
INDEX	Index_priv	Tables
INSERT	Insert_priv	Tables or columns

Continúa en la siguiente página

SELECT	Select_priv	Tables or columns
…	…	…

Tal como se puede ver, existen multitud de privilegios que se les puede otorgar a los usuarios. Vamos a ver cómo añadir y modificar los permisos de los usuarios.

1.2.1. Añadir permisos a usuarios

Existen muchos tipos de privilegios que podemos otorgar a los usuarios, y veremos una pequeña muestra de ellos.

Dar todos los permisos a todo MySQL

Se le otorgan todos los permisos sobre todo

```
mysql> GRANT ALL ON *.* TO 'nuevo_admin'@'192.168.1.10';
```

Dar todos los permisos sobre tablas

Se le otorgan todos los permisos sobre todas las tablas de la base de datos db1

```
mysql> GRANT ALL ON db1.* TO 'usuario'@'localhost';
```

Dar permisos de SELECT

Es decir, el usuario podrá realizar consultas SELECT sobre la tabla alumnos de la base de datos db1.

Sólo realizar consultas SELECT sobre la tabla alumnos de la base de datos db1.

```
mysql> GRANT SELECT ON db1.alumnos TO 'usuario2'@'%';
```

Dar permisos de INSERT

El usuario podrá realizar inserciones en la tabla asignaturas.

```
mysql> GRANT INSERT ON db1.asignaturas TO 'usuario2'@'%';
```

Delegar permisos (GRANT)

Se puede dar permisos a un usuario para que pueda delegar permisos a otro usuario, pero sólo podrá dar los permisos que él mismo tenga.

Para que el usuario pueda delegar permisos

```
mysql> GRANT ALL ON prueba.* TO 'usuario3'@'%' WITH GRANT OPTION;
```

1.2.2. Activar nuevos permisos

Los nuevos permisos no son aplicados hasta que no se ejecuta la siguiente sentencia:

Para aplicar las modificaciones de permisos

```
mysql> FLUSH PRIVILEGES;
```

1.2.3. Eliminar permisos a usuarios

Se pueden eliminar, o revocar, permisos a usuarios:

Revocar permisos de INSERT al usuario:

```
mysql> REVOKE INSERT ON prueba.asignaturas FROM 'usuario2'@'%';
```

1.2.4. Visualizar permisos usuarios

Ver los permisos de nuestro usuario

Ver los permisos del usuario actual

```
mysql> SHOW GRANTS;
```

Ver los permisos de usuario2

```
mysql> SHOW GRANTS FOR 'usuario2'@'%';
```

V

Copias de seguridad

1. Gestión de backups en MySQL

Con la instalación del servidor de MySQL se realiza la instalación de ciertas aplicaciones de administración del servidor, entre las que se encuentran el programa para la realización de backups: **mysqldump**.

Existen distintas maneras de realizar un backup en MySQL, entre las que se pueden destacar:

- **Backup completo**: Backup de todas las bases de datos que se encuentran instaladas en el servidor, incluyendo las del sistema.
- **Backup de base de datos**: Se realizaría el backup de la base de datos en concreto elegida.
- **Backup de tabla**: En este caso realizaríamos el backup de las tablas de una base de datos que nos interese.

Una vez elegido el tipo de backup que vamos a realizar deberemos elegir la herramienta que usaremos para llevar a cabo la estrategia planteada:

- **mysqldump**: Script incluido en las herramientas instaladas junto con MySQL. Por defecto realiza backups en formato de texto plano que podemos guardar en un fichero. Nos permite realizar backup completos, de ciertas bases de datos o de tablas concretas.
- **Percona XtraBackup**: Herramienta creada por la empresa Percona que nos permite realizar copias de seguridad sin realizar bloqueos en la base de datos.
- **Otras herramientas**: Existen multitud de herramientas para realizar backups para MySQL. Antes de hacer uso de ellas, deberemos asegurarnos que cumplen con nuestras necesidades.

Hay que tener en cuenta que los backups de los datos de las bases de datos no realiza un backup de los ficheros de configuración que se sitúan en `/etc/mysql` por lo que el backup de esta configuración debe realizarse por algún otro método, ya que la configuración del servidor también es muy importante.

1.1. Herramienta propia: mysqldump

Vamos a hacer uso de la herramienta propia que se instala junto a MySQL, que es mysqldump. Este programa es muy sencillo de utilizar ya que los parámetros que reciben son similares al propio CLI.

Por otro lado, el fichero generado con este programa es un fichero de texto que puede ser visualizado con cualquier editor en caso de necesidad, y fácilmente importado en cualquier otro servidor.

Ayuda del comando mysqldump

```
root@ubuntu:~# mysqldump
Usage: mysqldump [OPTIONS] database [tables]
OR     mysqldump [OPTIONS] --databases [OPTIONS] DB1 [DB2 DB3...]
OR     mysqldump [OPTIONS] --all-databases [OPTIONS]
For more options, use mysqldump --help
```

Tal como podemos ver, si ejecutamos el script, nos da una ligera idea de cómo podemos realizar un backup con con el mismo. En caso de que el backup sea de una máquina remota para traerlo a local, podremos hacer uso de varios parámetros, por ejemplo `mysqldump -u USER -h IP\_REMOTA -p`, como cuando usamos el CLI.

1.1.1. Backup completo

Un backup completo de MySQL hará que se guarde toda la información de las bases de datos internas del sistema, en las que se incluyen los usuarios y sus privilegios.

Este sistema es el método más sencillo si queremos realizar después una restauración completa en un nuevo servidor.

Hacer backup completo

```
root@ubuntu:~# mysqldump --all-databases > backup-completo.sql
```

De esta manera, le hemos indicado al script mysqldump que realice un backup de todas las bases de datos y el contenido obtenido lo vuelque al fichero backup-completo.sql. Este fichero generado es un fichero de texto que contendrá las bases de datos del sistema, usuarios, privilegios…

1.1.2. Backup de base de datos

Podemos realizar el backup únicamente de una, o varias, bases de datos que nos pueda interesar. Para poder realizar el backup de la base de datos que nos interese se hace uso del parámetro "**–databases**" seguido del nombre de las bases de datos, o en su forma abreviada "**-B**". Los dos siguientes comandos son iguales:

Hacer backup de sólo las bases de datos DB1 y DB2

```
root@ubuntu:~# mysqldump --databases DB1 DB2 > backup-db1-db2.sql
root@ubuntu:~# mysqldump -B DB1 DB2 > backup2-db1-db2.sql
```

Al crear el backup de una, o varias, base de datos de la anterior manera, en el fichero de backup nos aparecerá algo tal que:

Información interna del backup

```
CREATE DATABASE /*!32312 IF NOT EXISTS*/ `DB1`;
USE `DB1`;
```

De esta manera, lo que nos está indicando el backup es que a la hora de restaurar, en caso de que no exista DB1 se creará previamente. Posteriormente realizará el borrado de las tablas y las volverá a crear para realizar el insert de datos.

También podremos obviar el argumento, por lo que en caso de introducir un único parámetro, lo tomará com la base de datos de la que realizar copia de seguridad.

Hacer backup de una única base de datos

```
root@ubuntu:~# mysqldump DB1 > backup-DB1.sql
```

Al hacer el backup de esta manera, en el fichero no aparecerá la creación de la base de datos, por lo que a la hora de realizar la importación tendremos que hacerlo sobre una base de datos ya creada.

1.1.3. Backup de una tabla

Puede darse casos que nos interese realizar backups de una única tabla, o poder hacer backups de una tabla de manera más seguida, y para ello podremos ejecutar:

De esta manera hacemos backup de la tabla “asignaturas” de la base de datos DB1.

1.2. Importar un backup

Tal como hemos visto existen distintas maneras de realizar un backup, por lo que dependiendo de cómo es el backup que queremos importar, se hará de una manera u otra.

Hay que entender que la importación de un backup puede suponer la pérdida de datos de las bases de datos del servidor donde importamos el backup, por lo que hay que asegurarse muy bien qué queremos restaurar y dónde lo vamos a restaurar para no haya pérdida de datos donde no queremos.

¡Atención!

Importar datos de un backup puede suponer la pérdida de datos en el servidor donde se realiza la importación.

También es lógico pensar que dependiendo de la herramienta utilizada para realizar el backup la importación del mismo no tiene por qué ser igual, ya que distintas herramientas pueden dar lugar a distintos tipos de ficheros.

¡Atención!

Antes de restaurar un backup confirma cómo se hizo y cómo haces la importación.

1.2.1. Importar backups de mysqldump

Como se ha comentado previamente, mysqldump se instala junto con el servidor, y debido a que es una herramienta sencilla de usar, su importación también es sencilla. Para realizar la importación de este tipo de backups haremos uso del propio CLI `mysql`.

1.2.1.1. Importar un backup completo

Al importar un backup completo haremos la sustitución de todas las tablas internas de MySQL por los datos que haya en el backup. Por lo tanto, nos tendremos que asegurar muy bien de que el servidor donde realizaremos la importación de datos es un servidor sin datos importantes.

Para importar un backup completo haremos:

Importar un backup completo

```
root@ubuntu:~# mysql < backup-completo.sql
```

Con esto estaremos indicando que el contenido del fichero de backup se importe sobre la conexión que hemos realizado, en este caso en local.

1.2.1.2. Importar un backup de una base de datos

Tal como hemos visto antes, a la hora de crear el backup de una, o varias, base de datos se puede realizar añadiendo el parámetro "–databases", "-B" o sin especificar nada, sólo la base de datos. Dependiendo de cómo se haya creado el backup, la importación variará:

- Si el backup se ha realizado con "**–databases**" o "**-B**" en el propio backup aparece:

  "Create database" en el propio fichero de backup

  ```
  CREATE DATABASE /*!32312 IF NOT EXISTS*/ `DB1`;
  ```

  y para realizar la importación:

  Importar un backup completo

  ```
  root@ubuntu:~# mysql < backup-DB1.sql
  ```

- Si no hemos especificado más que el nombre de la base de datos, en el backup no aparece la creación o uso de la base de datos, por lo que tendremos que indicarlo en la importación, y **la base de datos "restaurado" debe existir previamente**.

  Importar un backup completo en una base de datos llamada "restaurado"

  ```
  root@ubuntu:~# mysql restaurado < backup-DB1.sql
  ```

1.2.1.3. Importar un backup de una tabla

Por último, si queremos importar un backup donde sólo existen tablas, a la hora de importar deberemos indicar en qué base datos queremos realizar la importación, ya que en el backup no aparece dicha información. Por ejemplo:

Importar un backup de una tabla en la base de datos "restauracion"

```
root@ubuntu:~# mysql restauracion < backup-db1-asignaturas.sql
```

De esta manera estamos indicando al CLI que el contenido del fichero de backup se importe sobre la base de datos *restauración*. Lógicamente esta base de datos debe estar creada previamente.

VI

Monitorización

1. Monitorización de SGBDs

En el caso que nos ocupa, el de los Sistemas Gestores de Bases de Datos, aparte de la monitorización básica comentada en el anexo, necesitaremos monitorizar el estado del SGBD propiamente dicho. Para ello, de nuevo, se crearía una plantilla específica para cada SGBD que podamos tener en nuestra infraestructura. No será lo mismo monitorizar un servidor basado en MySQL o un Oracle, aunque muchos checks a comprobar deban ser lo mismo, pero la manera en la que se realizará la comprobación en el servidor será distinta.

Entre las comprobaciones que podemos realizar en un SGBD nos podemos encontrar con:

- Servicio SGBD arrancado
- Cantidad de RAM utilizada por el SGBD
- Número de conexiones a las bases de datos
- Número de hilos en ejecución del SGBD
- Número de queries en ejecución
- Número de tablas en memoria
- Número de tablas bloqueadas
- …

1.1. Monitorizar MySQL

Todo lo expuesto en el anexo es referente a los sistemas de monitorización en general, y es similar en todos ellos. Tal como se ha comentado previamente, cuando un servidor cuenta con un servicio éste debe ser monitorizado, y dependiendo del servicio a monitorizar se realizará una serie de comprobaciones u otras.

A la hora de monitorizar un sistema gestor de bases de datos deberíamos tener en cuenta al menos las siguientes comprobaciones, y en el caso de MySQL:

- Número de conexiones actuales (usuarios/conexiones tiene el servicio)
- Tiempo del servicio activo (uptime)
- Número de hilos en ejecución
- Tamaño de memoria utilizado
- Tamaño de memoria caché utilizado
- Número de tablas en memoria
- Número de slow queries
- Número de conexiones abortadas
- Número de queries totales

En servidores en alta disponibilidad, deberíamos comprobar, en caso de un clúster:

- Número de nodos en el clúster
- Estado general del clúster
- Latencia entre los nodos del clúster
- Si el nodo está conectado al clúster
    - Puede pasar que un nodo "se salga" del clúster (o lo saquemos) para realizar mantenimiento

Si el servidor está dentro de una infraestructura de **Primario → Réplica**:

- Estado de la replicación
- Retraso de la replicación
- Tamaño del binlog

Las comprobaciones expuestas son un mero ejemplo, y existen muchas más que podemos realizar a nuestros sistemas. Para poder realizar este tipo de monitorización podemos hacer uso de scripts propios (ya que muchas de las comprobaciones son consultas a variables de estado de MySQL), scripts creados por otras personas o scripts de monitorización hechos de manera exclusiva para MySQL.

1.1.1. Scripts propios

Para realizar gran parte de los checks expuestos previamente se puede hacer uso de scripts propios, ya que pueden realizar consultas de las variables de estado para comprobar y determinar si el estado es correcto.

Un ejemplo de script creado *ad hoc* para nuestro sistema podría ser:

Script propio para monitorizar

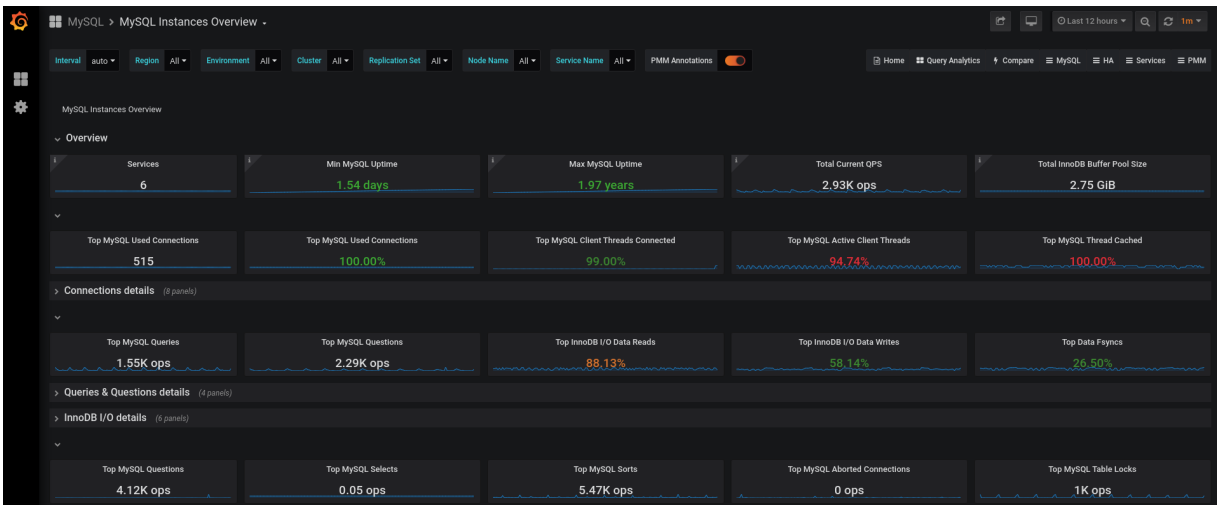
```
#!/bin/bash
mysql -e "SHOW STATUS LIKE 'Slow_queries';" -N | awk '{print $2}'
```

1.1.2. Percona monitoring tools

La alternativa a hacer uso de scripts propios es usar programas o scripts realizados por terceros, que estén ampliamente probados y que tienen detrás un equipo que quizá sea más amplio que el nuestro, y por tanto esté mejor probado.

La empresa Percona tiene un par de soluciones sobre monitorización de SGBDs (no sólo para MySQL, también para PostgreSQL y MongoDB), que son:

- **Percona Monitoring and Management**: Un sistema completo de monitorización, interfaz web incluída, que podremos instalar en nuestros propios servidores. Podemos ver aquí una demo.



- **Percona Monitoring Plugins**: En este caso son un conjunto de plugins (o scripts) que podremos utilizar en nuestro sistema de monitorización propio (Nagios, Centreon o Cacti). En la documentación explican cómo hacer uso de ellos.

VII

Alta Disponibilidad

1. Alta Disponibilidad

La alta disponibilidad en servidores se puede definir como el diseño de infraestructura (y su implantación) que asegura la continuidad del servicio y que no tiene un único punto de fallo.

Es lógico entender que un servicio debe de ser contínuo en el tiempo, ya que debe de dar servicio de manera continuada para que los usuarios puedan acceder a él. Pero para que esta premisa sea efectiva, y para asegurarnos que así sea, **la infraestructura debe de estar redundada y carecer de puntos de fallo únicos en su diseño**.

Esto quiere decir, que de cada servicio y para cada posible punto de fallo deberá haber al menos dos de ellos, para que en caso de que uno deje de funcionar el servicio siga funcionando (dos tomas eléctricas separadas, dos servidores que otorguen el servicio, dos conexiones a internet, ...).

Es habitual que un sistema en Alta Disponibilidad deba de estar pensado desde el diseño. Algunos tipos de servicios pueden empezar como un único servidor y posteriormente realizar un escalado horizontal, formando la alta disponibilidad, mientras que para otros **el diseño en alta disponibilidad debe de estar pensado desde el comienzo** (habitualmente en algunos tipos de clusters).

1.1. Importancia de un sistema en Alta Disponibilidad

Como se ha citado previamente, la alta disponibilidad nos va a asegurar al menos tres grandes ventajas:

- Una continuidad en el servicio
- Un diseño libre de puntos de fallos únicos, gracias a la redundancia.
- Mejorar el rendimiento global.

La redundancia permitirá que en caso de fallo de algún equipamiento/servicio, al estar redundando, no afecte al servicio. Gracias al sistema de monitorización seremos capaces de ver el problema y solventarlo lo antes posible. De estar el diseño correcto, el servicio mantendrá su actividad, mientras que por el contrario, si ha habido algún fallo en el diseño de infraestructura (o el problema es más grave de lo esperado) el servicio se verá afectado.

1.2. Tipos de Alta Disponibilidad

Existen muchos tipos de alta disponibilidad dependiendo de en qué capa de infraestructura estemos hablando. Por poner unos ejemplos:

- **Redundancia eléctrica**: Los servidores normalmente cuentan con doble fuente de alimentación, por lo que cada fuente de alimentación debe de estar conectada a una toma eléctrica completamente separada de la otra.
- **Redundancia de conectividad física**: El acceso a internet debe de ser redundado.
- **Redundancia de conectividad LAN**: El acceso a la LAN/DMZ/red de servicio debe de estar redundado (stacks de switches, LACPs configurados en switches y servidores, …).

- **Redundancia de servidores**: Debe de existir una redundancia de servidores para asegurar que el servicio funcione en más de un servidor físico.
- **Redundancia de servicio**: El servicio que se ofrece debe de estar redundado entre los distintos servidores.

La alta disponibilidad también se puede diferenciar como:

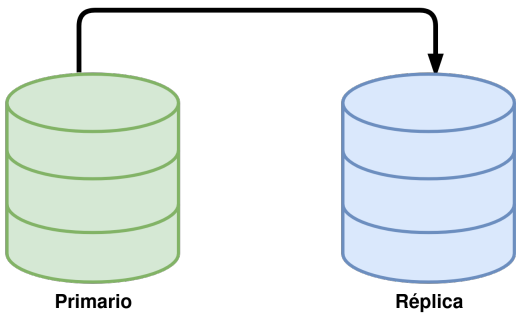
- **Alta disponibilidad real**: En caso de que haya algún problema el servicio continúa como si no hubiese pasado nada, gracias a la redundancia completa de servicios/dispositivos.
- **Alta disponibilidad pasiva**: En caso de error, los servidores activos serían los que reciben el impacto del problema y hay que escalar los servidores pasivos a modo activo para que comiencen a funcionar otorgando el servicio. Como se puede presuponer, esta modificación puede ser realizada de manera automática o de manera manual (lo que llevaría algo de tiempo, y por tanto el servicio se vería afectado).

2. Alta Disponibilidad en un SGBD

En lo que se refiere a los SGBDs, y en concreto a MySQL, podemos diferenciar entre los siguientes tipos de alta disponibilidad que veremos a continuación.

2.1. Primario - Replica

También denominado "Primario → Réplica" (antiguamente denominado "Maestro → Esclavo"). Es un tipo de alta disponibilidad pasiva en el que contamos con dos servidores.



El principal/primario/master es el que recibe todas las modificaciones que se realizan en la base de datos, y por tanto debe de estar en funcionamiento para poder recibir las modificaciones de datos.

El secundario/réplica/esclavo, como su nombre indica, recibe los datos replicados del principal. Es posible recibir en este servidor peticiones de "SELECTS" para que así quite carga al principal, pero hay que asegurarse de no recibir ni INSERTS ni UPDATES.

En caso de que exista algún problema con el servidor primario, el servidor de réplica deberá pasar a ser el primario, por lo que habrá que hacer que también reciba los inserts. En caso de que el servidor haya sido configurado en modo "sólo lectura", habrá que quitarlo.

Podemos destacar de este tipo de replicación las siguientes características:

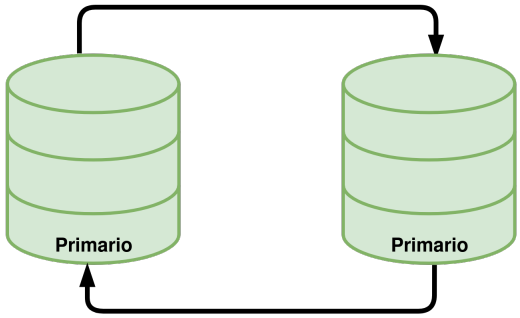
- Sistema sencillo de replicación.

- La sincronización es asíncrona por defecto. El servidor secundario no tiene por qué estar todo el rato conectado. Cuando se conecte, pedirá y recibirá todas las modificaciones desde la última realizada.
- Dependiendo de la configuración realizada en el servidor secundario, se replicará todo, sólo alguna base de datos o sólo alguna tabla.
- Para no cargar al servidor primario, los backups se pueden realizar sobre el servidor de réplica.
- Se pueden realizar réplicas a larga distancia, estando el servidor de réplica en otro datacenter.

Hay que tener en cuenta que si el servidor réplica se detiene, **las modificaciones del primario se guardan en unos ficheros llamados *binlogs***. Estos ficheros se suelen configurar para almacenar una cantidad de datos precisa, por lo que si el tiempo de caída del servidor réplica supera el tiempo que los binlogs almacenan información, al retomar la actividad no obtendrá todos los datos del servidor primario.

2.2. Primario - Primario

Una replicación "Primario ↔ Primario" en MySQL no es más que una replicación "Primario → Réplica" de doble sentido.



Esto quiere decir que ambos servidores pueden recibir todo tipo de consultas y modificaciones. El problema es que podría darse el caso de que dos INSERT simultáneos en ambos servidores obtengan identificadores auto-incrementales idénticos, lo que daría error a la hora de realizar los INSERT.

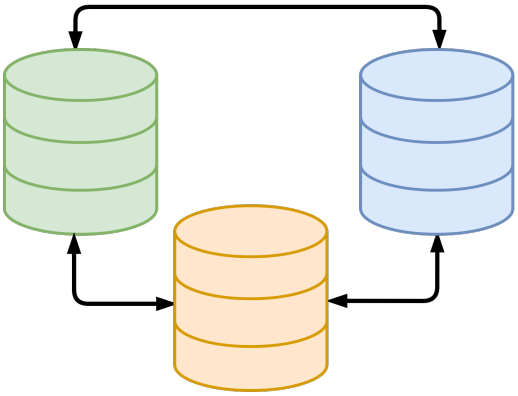
Por lo tanto, para evitar eso, lo que se hace es modificar las variables auto-increment para que en uno de los servidores se den números pares y en el otro impares. Estas variables son:

- **auto_increment_increment**: es el incremento que se realiza entre valor y valor de "auto_increment". En este caso, debe de ser "2" en ambos servidores porque es el número de servidores que tenemos.
- **auto_increment_offset**: determina el comienzo del sistema "auto_increment" en nuestro servidor. En este caso, la variable obtendrá valores distintos en los servidores:
  - "1": en un servidor, por lo tanto, en este siempre habrá valores impares.
  - "2": en el otro servidor, por lo que los valores serán pares.

Para que ambos servidores reciban peticiones, la aplicación que las pida podrá preguntar a ambos servidores o podremos tener un servidor proxy por delante que realice un balanceo de peticiones a ambos servidores.

2.3. MySQL Cluster

Como ya se ha comentado previamente, MySQL Cluster es un SGBD distinto a MySQL Community, y el tipo de Alta Disponibilidad que podemos obtener con él es distinto a los citados previamente. En este caso, el número de servidores mínimos con los que deberemos contar será de 3, y a partir de ahí deberá ser siempre número impar.



No entraremos en detalle en este sistema, pero a grandes rasgos podemos pensar de él como un sistema "Primario ↔ Primario" multi servidor. **En este caso la sincronización es síncrona**, es decir, **antes de dar por terminada una sentencia se asegura que está replicada en todos los servidores**.

3. Sistema "Primario → Réplica" con MySQL

Como se ha visto, MySQL permite distintos tipos de replicación y creación de infraestructura en Alta Disponibilidad. En nuestro caso, vamos a optar por la más sencilla de todas ellas, que es la replicación "Primario → Réplica".

Para ello, contaremos con dos servidores Ubuntu 20.04 recién instalados con MySQL Community también recién instalados, y a partir de aquí comenzaremos a realizar la configuración oportuna que necesitaremos para conseguir nuestro objetivo.

3.1. A tener en cuenta

Antes de comenzar con la configuración tiene que quedar clara una serie de conceptos:

- Los datos deben de ser insertados/actualizados en el principal.
- Una vez realizadas las modificaciones son escritas en los **binary-log** del principal.
- El servidor réplica, para evitar problemas, debería estar en modo "sólo lectura".
- El servidor réplica se conecta al primario en busca de las modificaciones (en los binary-log) y ejecutará las órdenes en el propio servidor réplica.
    - El servidor réplica mantiene la posición del binary-log en la que se encuentra.
    - MySQL acepta replicación de una serie a una versión superior, pero nunca haciendo saltos de dos versiones.
- Cada servidor contará con **server-id** distinto.

3.2. Creación del sistema principal/origen

Con todo lo citado previamente, los pasos para crear el sistema primario son los siguientes:

1. Configurar servidor con IP estática
2. Modificar configuración de MySQL
3. Crear usuario de replicación
4. Hacemos backup inicial de los datos.

Se obvia el primer paso ya que es configuración del sistema operativo y aparece en el anexo.

3.2.1. Configuración del servidor principal

En el fichero de configuración del servidor principal tendremos que modificarlo para que aparezca la siguiente configuración

Configuración MySQL del primario

```
[mysqld]
...
server-id = 1
log_bin = /var/log/mysql/binlog
binlog_expire_logs_seconds      = 2592000
max_binlog_size = 100M
```

Una vez modificado el fichero de configuración, el servidor debe ser reiniciado. Para comprobar si el servidor está en modo "primario" podemos consultarlo de la siguiente manera:

Comprobar estado del primario

```
mysql> SHOW MASTER STATUS \G
*************************** 1. row ***************************
File: mysql-bin.000001
Position: 1397
Binlog_Do_DB:
Binlog_Ignore_DB:
Executed_Gtid_Set:
1 row in set (0.00 sec)
```

Podemos observar el fichero "mysql-bin" y la posición del mismo del servidor principal.

3.2.2. Crear usuario de replicación

Como se ha comentado previamente, el servidor réplica se conectará al principal, por lo que tiene que existir un usuario con el que poder conectarse y comprobar el estado de los binlogs.

Crear usuario para la replicación

```
CREATE USER 'repl'@'192.168.1.21' IDENTIFIED WITH mysql_native_password BY 'password';
GRANT REPLICATION SLAVE ON *.* TO 'repl'@'192.168.1.21';
FLUSH PRIVILEGES;
```

Hay que tener en cuenta que la IP tendrá que ser la IP del servidor réplica y será con este usuario con el que realizará la conexión. Es obligatorio poner “mysql_native_password” ya que el nuevo sistema de autenticación “caching_sha2_password” necesita de certificados para realizar la conexión.

3.2.3. Backup inicial del servidor primario/origen

Una vez configurado el servidor en modo primario, realizaremos el backup inicial que posteriormente será cargado en el servidor réplica. Debido a la configuración que hemos realizado, el servidor réplica deberá conocer a partir de qué posición del binlog obtener las modificaciones después de haberse realizado el backup. Por tanto, el backup debe realizarse de la siguiente manera:

Hacer backup inicial para la replicación

```
root@server1:~# mysqldump --all-databases --source-data > dbdump.sql
```

Y como se puede comprobar, el parámetro nuevo es “--source-data”, que hará que dentro del fichero de backup aparezca (es posible que en el futuro aparezca como “PRIMARY” o “PRIMARY_LOG_”):

Hacer backup inicial para la replicación

```
...
--
-- Position to start replication or point-in-time recovery from
--
CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin.000001', MASTER_LOG_POS=1397;
...
```

Esta información es el punto exacto en el que se ha quedado el binlog (tanto en qué fichero como en qué posición) cuando se ha realizado el backup. Esta información la necesitaremos para posteriormente en el servidor réplica indicarle que es a partir de este punto el que deberá pedir la información a replicar.

3.3. Creación del sistema secundario/réplica

En el caso del servidor secundario los pasos a realizar son:

1. Configurar IP estática.
2. Modificar la configuración de MySQL.
3. Importar backup.
4. Configuración de conexión al servidor primario.
5. Comenzar la replicación.

6. Comprobar estado de la replicación.

3.3.1. Modificar configuración en el servidor RÉPLICA

En este caso, la modificación que debemos de realizar es cambiar el **server-id**, que tendrá que ser distinto al del primario, y podremos poner de manera opcional el nombre del servidor réplica en la variable **report_host**.

Configuración MySQL del servidor réplica

```
[mysqld]
...
server-id = 2
# la opción report_host es opcional
report_host = server2
```

3.3.2. Importar el backup

Tal como se realiza de cualquier backup, teniendo el backup en el propio servidor réplica:

Restaurar backup inicial

```
root@server1:~# mysql < dbdump.sql
```

3.3.3. Configuración de conexión al primario

Desde el CLI de MySQL debemos realizar la siguiente sentencia, teniendo en cuenta:

- La IP de nuestro servidor primario.
- Los datos del usuario de replicación creado previamente.
- Los datos de binlog que hemos podido obtener del fichero de replicación.

Configuración de conexión al servidor primario

```
mysql> CHANGE REPLICATION SOURCE TO
           SOURCE_HOST='192.168.1.129',
           SOURCE_USER='repl',
           SOURCE_PASSWORD='password',
           SOURCE_LOG_FILE='mysql-bin.000001',
           SOURCE_LOG_POS=1397;
```

3.4. Comenzar la replicación

Ahora el servidor réplica tiene que comenzar la replicación a partir de la posición y del fichero del bin-log que se ha configurado. Es de suponer que el servidor primario ha recibido modificaciones, y por tanto, deben ser replicadas.

Comenzar la replicación

```
mysql> START REPLICA;
```

3.5. Comprobar estado de la replicación

La replicación es un proceso que debe de ser monitorizado y nos debemos asegurar que funciona de manera correcta. Para conocer cuál es el estado de la replicación podremos obtenerlo ejecutando:

Comprobar estado de la replicación

```
mysql> SHOW REPLICA STATUS \G
*************************** 1. row ***************************
         Replica_IO_State: Waiting for source to send event
              Source_Host: 192.168.200.10
              Source_User: repl
              Source_Port: 3306
            Connect_Retry: 60
          Source_Log_File: mysql-bin.000001
      Read_Source_Log_Pos: 3256
           Relay_Log_File: ubuntu-relay-bin.000003
            Relay_Log_Pos: 324
    Relay_Source_Log_File: mysql-bin.000001
       Replica_IO_Running: Yes
      Replica_SQL_Running: Yes
...
    Seconds_Behind_Source: 0
...
Replica_SQL_Running_State: Replica has read all relay log;
waiting for more updates
```

Como alternativa, también podremos ver desde el servidor primario cuántos servidores réplica tiene, pero la información proporcionada es escasa, pero nos puede servir para obtener información de un servidor no instalado por nosotros.

Estado de servidores réplica desde el primario

```
mysql> SHOW REPLICAS;
+-----------+---------+------+-----------+--------------------------+
| Server_Id | Host    | Port | Source_Id | Replica_UUID             |
+-----------+---------+------+-----------+--------------------------+
|         2 | server2 | 3306 |         1 | 9fb33895-4c78-11ec-b23... |
```

```
+-----------+--------+------+-----------+--------------------------+
1 row in set (0,00 sec)
```

4. Ficheros binary logs

Durante la replicación se han usado los binary logs, y para activarlos se ha tenido que configurar en el servidor primario/origen las siguientes variables de configuración:

- log_bin = /var/log/mysql/binlog
- binlog_expire_logs_seconds = 2592000
- max_binlog_size = 100M

El significado de las variables es el siguiente:

- **log_bin**: path donde se van a guardar los binlogs. Siendo el ejemplo de arriba, obtendremos unos ficheros tal que:
  - /var/log/mysql/mysql-bin.000001
  - /var/log/mysql/mysql-bin.000002
  - /var/log/mysql/mysql-bin.000003
  - …
- **binlog_expire_logs_seconds**: el número de segundos que pasarán antes de que los binlogs se borren de manera automática. Significa que tendremos tantos ficheros binlog como para poder guardar los cambios acontecidos en la base de datos durante la cantidad de segundos puesta.

  Eso significa, que si nuestra réplica está menos de 30 días sin conectarse al primario (por un fallo de conectividad), podríamos llegar a recuperar la replicación sin necesidad de rehacer nada, sólo consiguiendo que la conectividad vuelva a funcionar. Por otro lado, la cantidad de datos almacenados en esa cantidad de días en los ficheros binlog puede hacer que nuestro servidor ocupe mucho espacio, por lo que se debería pensar en guardar esos binlogs en otra partición o modificar la variable.
- **max_binlog_size**: tamaño máximo que alcanzará cada fichero binlog. En caso de llegar al tamaño especificado, se creará un nuevo fichero binlog, modificando el índice y el nombre del nuevo fichero. El tamaño máximo que puede tener cada binlog es 1GB.

4.1. Borrado de binlogs

En caso de necesidad (por falta de espacio en el servidor), podemos realizar el borrado de ficheros binlogs siempre y cuando nos hayamos asegurado que la réplica ya los ha recibido (es decir, la replicación esté correcta, o el binlog por el que vaya la réplica sea mayor al que queremos borrar).

¡Atención!

Antes de borrar binlogs, debemos asegurar que los servidores réplica están sincronizados

Podemos realizar el borrado de binlogs indicando el fichero:

Borrado de binlogs indicando el fichero

```
mysql> PURGE BINARY LOGS TO 'mysql-bin.000010';
```

Como alternativa, también podemos especificar que borre todos los logs anteriores a la fecha y hora indicada, borrando así los ficheros necesarios para ello. De nuevo, nos tendremos que asegurar de que la replicación esté correcta antes de hacerlo.

Borrado de binlogs indicando la fecha

```
mysql> PURGE BINARY LOGS BEFORE '2020-11-11 00:00:00';
```

VIII

Anexos

1. Instalar Ubuntu 22.04 LTS

En este anexo realizaremos la instalación de la distribución Ubuntu 22.04 LTS en su versión para servidores. En este anexo no se va a explicar cómo realizar la creación de una máquina virtual donde se aloja el sistema operativo, ya que existen distintos tipos de virtualizadores.

No se realizará una guía "paso a paso", sino que se centrará en las partes más importantes de la instalación y en las que más dudas puedan surgir.

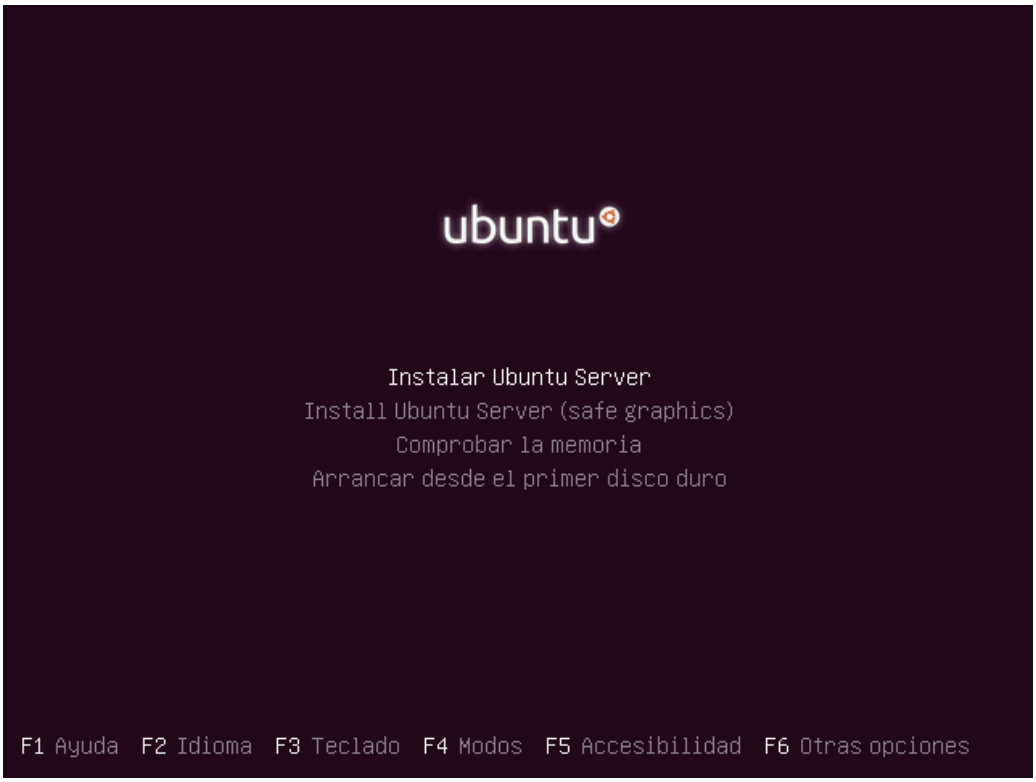
1.1. Descargar Ubuntu 22.04

La ISO la obtendremos de la web oficial y seleccionaremos la versión 22.04 LTS de Ubuntu Server. Esta ISO contendrá el sistema base de Ubuntu y nos guiará para realizar la instalación del sistema operativo.

Una vez descargada la ISO tendremos que cargarla en el sistema de virtualización elegido y arrancar la máquina virtual.

1.2. Instalar Ubuntu 22.04

Tras arrancar la máquina virtual nos aparecerá un menú para seleccionar el idioma durante la instalación y le daremos a "Instalar Ubuntu Server".



A partir de aquí comenzará el instalador y los pasos que nos aparecerán serán los siguientes (algunos de estos pasos puede que no estén 100 % traducidos al castellano):

1. Elegir el idioma del sistema
2. Actualización del instalador:

- Si la máquina virtual se puede conectar a internet, comprobará si existe una actualización del propio instalador de Ubuntu.
- Podemos darle a “Continuar sin actualizar”

3. Configuración del idioma del teclado
4. Configuración de la red
5. Configuración del proxy de red
6. Configuración del “mirror” o servidor espejo desde donde descargarse los paquetes de software para las actualizaciones posteriores.
7. Selección del disco duro donde realizar la instalación
8. Elegir el particionado de disco.
9. Configuración del perfil. Introduciremos el nombre de usuario, el nombre del servidor y la contraseña del usuario que vamos a crear.
10. Configuración de SSH Server. Aceptaremos que se instale el servidor SSH durante la instalación. En caso de no seleccionar esta opción, posteriormente podremos realizar la instalación.
11. “Featured Server Snaps”. En esta pantalla nos permite instalar software muy popular en servidores.

Una vez le demos a continuar, comenzará la instalación en el disco duro. Debido a que durante la instalación tenemos conexión a internet, el propio instalador se descarga las últimas versiones de los paquetes de software desde los repositorios oficiales.

Al terminar la instalación, tendremos que reiniciar la máquina virtual.

1.3. Post-instalación

Tras realizar el reinicio de la máquina virtual nos encontraremos con que el sistema arranca en el sistema recién instalado, y que tendremos que loguearnos introduciendo el usuario y la contraseña utilizadas en la instalación.

1.3.1. Actualización del sistema

Por si acaso, realizaremos la actualización del índice del repositorio, actualizaremos el sistema y en caso necesario realizaremos un nuevo reinicio:

Actualizar Ubuntu

```
root@ubuntu:~# apt update
...
root@ubuntu:~# apt upgrade
...
```

Con estos comandos nos aseguramos que el sistema está actualizado a los últimos paquetes que están en el repositorio.

1.3.2. Poner IP estática

Debido a la configuración de red de nuestro servidor, la IP está puesta en modo dinámica, esto quiere decir que nuestro equipo ha cogido la IP por configuración de DHCP de nuestra red. Debido a que un servidor debe de tener IP estática, tenemos que realizar la modificación adecuada para ponerle la IP estática que mejor nos convenga. Para ello editaremos el fichero de configuración situado en la siguiente ruta: `/etc/netplan/00-installer-config.yaml`

Lo modificaremos para que sea parecido a (siempre teniendo en cuenta la IP y gateway de nuestra red):

Configurando IP estática en Ubuntu

```
network:
  ethernets:
    enp0s3:
      dhcp4: false
      addresses: [192.168.1.199/24]
      routes:
      - to: default
        via: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8,1.1.1.1]
  version: 2
```

El fichero de configuración que hemos modificado es de tipo YAML, que es un formato de texto que suele ser utilizado en programación o en ficheros de configuración. Este tipo de ficheros tiene en cuenta los espacios para el uso de la identación, y no suele permitir el uso de tabuladores.

Para aplicar los cambios realizados en el fichero de configuración deberemos ejecutar el siguiente comando que aplicará los cambios:

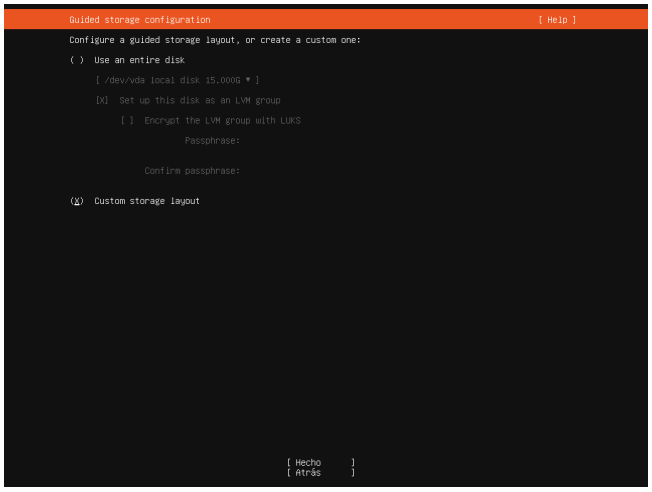
Aplicar configuración de IP

```
root@ubuntu:~# netplan apply
```

2. Configurar RAID 1 durante la instalación de Ubuntu

Tal como hemos podido ver anteriormente, durante la instalación de Ubuntu 20.04, en el paso 7 podemos realizar la instalación en el disco duro que tengamos instalado en el servidor físico o en la máquina virtual.

En este paso podemos realizar distintas configuraciones:



- Usar el disco entero.
    - Nos permitirá crear un sistema con LVM (por defecto activado) y con posibilidad de cifrar la partición creada.
- Crear un diseño de almacenamiento personalizado.

En la segunda opción podremos:

- Crear particiones a nuestro gusto.
- Elegir el sistema de ficheros de las particiones.
- Crear sistema RAID por software.

2.1. Pasos previos

Dado que vamos a crear un sistema RAID 1 durante la instalación de Ubuntu, necesitaremos al menos **dos discos duros** en nuestro servidor antes de comenzar con la instalación.

En nuestro sistema virtualizado hemos añadido dos discos duros de igual tamaño (15GB), en los cuales crearemos particiones para posteriormente sobre ellas realizar el RAID 1.

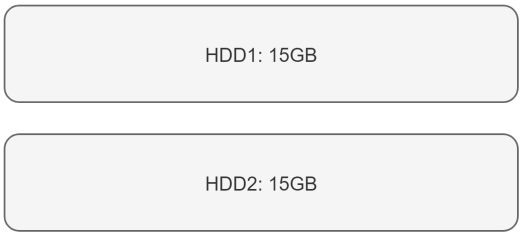
2.2. Entendiendo las particiones a realizar

En este apartado vamos a explicar la teoría que está detrás del sistema de particionado que vamos a necesitar crear y que posteriormente realizaremos en el sistema de instalación de Ubuntu.

2.2.1. Situación inicial: discos duros sin particionar

Como ya se ha comentado, en nuestro servidor vamos a contar con dos discos duros de igual tamaño. Esto suele ser lo habitual, pero lo importante es que las particiones que vayamos a crear sean del tamaño exacto, aunque un disco duro sea de mayor tamaño (aunque lógicamente, ese espacio quedará desaprovechado).

En la siguiente imagen vemos que tenemos dos discos duros de 15GB de tamaño cada uno:



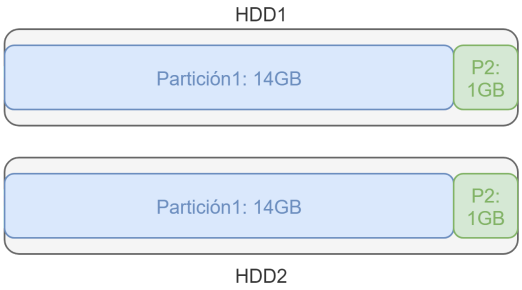
2.2.2. Particionado inicial

A continuación vamos a tener que pensar cómo van a ser las particiones que vamos a necesitar en nuestro servidor. En nuestro caso vamos a crear dos:

- **14GB**: Sistema operativo.
- **1GB**: (o hasta completar) SWAP.

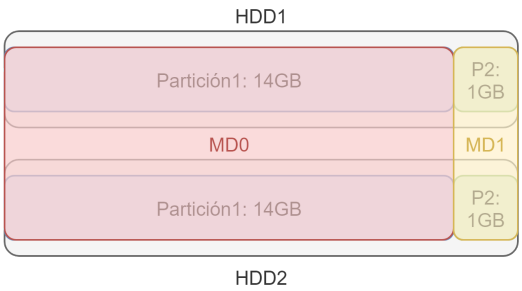
Como se puede entender, al tener una única partición, todo el sistema raiz "/" va a ir en ella, mientras que la otra partición será la usada para el área de intercambio.

Es importante entender que en este paso sólo **vamos a crear las particiones pero sin darles formato**. Por lo tanto, nuestros discos duros ahora tendrían este aspecto:



2.2.3. Crear particiones RAID

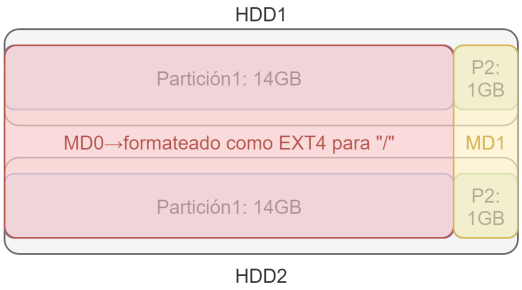
El siguiente paso es crear las particiones "virtuales" RAID. Vamos a crear una primera partición RAID que va a incluir las particiones de 14GB de ambos discos duros, y la segunda partición virtual incluirá las particiones de 1GB.



De esta manera, tendremos unas particiones MD0 y MD1 que son particiones virtuales.

2.2.4. Formatear particiones RAID con el formato adecuado

El último paso de la instalación es hacer uso de las particiones RAID creadas y formatearlas con el sistema de ficheros acorde a las necesidades que tengamos, y elegir el punto de montaje adecuado.



En nuestro caso va a ser:

- **MD0**: sistema de ficheros ext4 y lo vamos a utilizar como sistema de ficheros raíz "/".
- **MD1**: formateado como SWAP y actuará como área de intercambio.

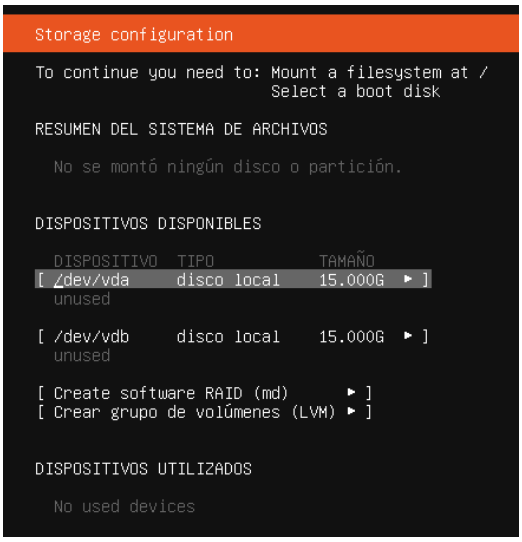
Tras este paso, la instalación del sistema operativo puede continuar de la manera habitual.

2.3. Realizando el particionado en el instalador de Ubuntu

Tras haber entendido las particiones que vamos a realizar, ahora es el momento de proceder en el instalador de Ubuntu. Vamos a seguir los mismos pasos que hemos explicado en el apartado anterior, de esta manera aplicaremos lo aprendido a nivel teórico.

2.3.1. Situación inicial: discos duros sin particionar

Tal como hemos comentado previamente, en el paso 7 de la instalación, elegiremos la opción de crear un "diseño de almacenamiento personalizado". Tras entrar en esta opción, el instalador tendrá el siguiente aspecto:



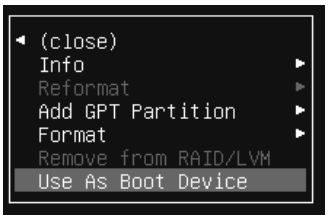
Tal como se puede ver, tenemos dos discos duros en el sistema: **vda** y **vdb**. El nombre de los discos viene de **V**irtual **D**isk, dado que la instalación la estamos realizando en una máquina virtual.

2.3.2. Particionado inicial

En este paso vamos a crear en cada uno de ellos la partición de 14GB y el resto del espacio la usaremos para la segunda partición.

2.3.2.1. Marcar discos como dispositivos de arranque

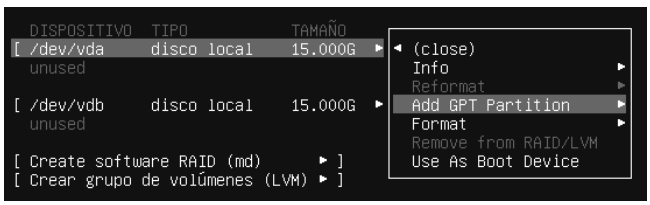
Por cómo funciona el sistema de arranque de Linux, antes de realizar las particiones vamos a marcar que ambos discos duros sean dispositivos de arranque (“Boot Device”). Para ello pulsaremos “Intro” en cada uno de los discos y elegiremos la opción correspondiente (imágenes de cada disco):



De esta manera, el instalador de Ubuntu creará una pequeña partición al inicio del disco donde al terminar se realizará la instalación del sistema de arranque GRUB en ambos discos duros.

2.3.2.2. Crear particiones

Ahora es el momento de crear las particiones, y los pasos serán seleccionar el disco duro, pulsar “Intro”, se nos desplegará un pequeño menú y vamos a elegir la opción *“Add GPT Partition”* y rellenaremos el tamaño de la partición que nos interese en el momento y el formato lo dejaremos en *“Leave unformatted”* (dejar sin formatear).



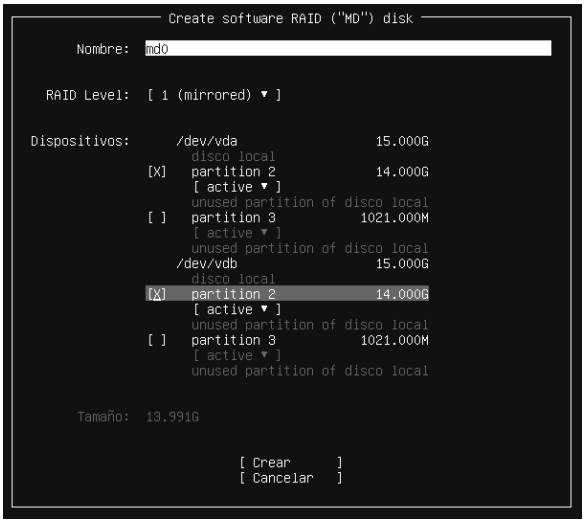
Estos pasos lo realizaremos en cada disco duro con las particiones que vamos a necesitar, quedando al finalizar el sistema así:

Tal como se puede ver, cada disco duro tiene dos particiones con el tamaño deseado que no están siendo utilizadas, y en la parte de abajo aparecen las particiones denominadas “BIOS grub spacer”.

2.3.3. Crear particiones RAID

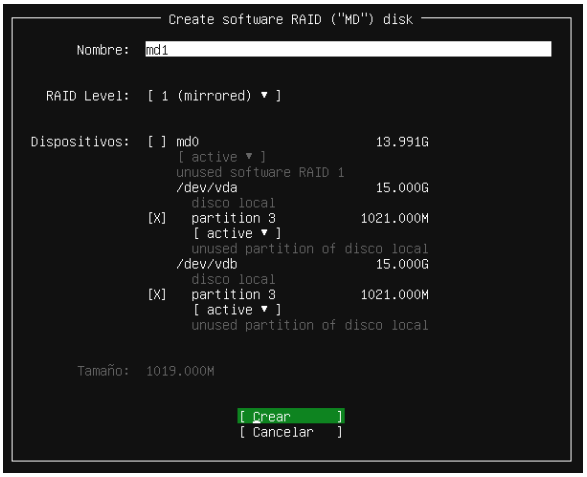
El siguiente paso es crear las particiones RAID en las que haremos que el sistema cree un RAID 1 haciendo uso de las particiones de los discos duros físicos. Seleccionaremos la opción “*Create software RAID (md)*” y nos aparecerá una ventana en la que podremos elegir:

- **Nombre**: de la partición que vamos a crear. Es habitual que estas particiones empiecen por “**md**”, ya que viene de “*multiple device*”.
- **Nivel RAID**: Podremos elegir entre las versiones 0, 1, 5, 6 y 10 de RAID. Por defecto está seleccionada la opción RAID 1.
- **Dispositivos**: sobre el que aplicaremos el RAID.

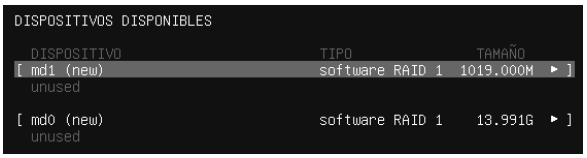


Crearemos primero **md0** seleccionando las particiones de 14GB tal como aparece en la siguiente imagen:

Y a continuación crearemos **md1** con las particiones restantes. Tal como se puede ver a continuación, las particiones de 14GB ya no aparecen, porque están siendo usadas en el otro RAID.



Tras la creación de los dispositivos “md”, nos aparecerán como dispositivos disponibles para usarlas en el siguiente paso:



2.3.4. Formatear particiones RAID con el formato adecuado

Aunque este paso lo vamos a realizar sobre las particiones RAID creadas previamente, es un paso que es habitual realizar cuando queremos que nuestra instalación tenga un sistema de particiones propio.

Tenemos que pensar que las particiones RAID ahora son como particiones normales, a las que les vamos a querer dar un formato y utilizarlas para realizar la instalación.

Vamos a seleccionar la primera, **md0**, posicionándonos encima de ella y dándole a “Intro” y posteriormente dándole a “*Add GPT partition*”:

Dejaremos el tamaño en blanco, indicando que usaremos todo el espacio libre, lo vamos a formatear con el sistema de ficheros **ext4** y se va a montar como el sistema de ficheros “/”.

En **md1** también le daremos a crear nueva partición, y haremos lo mismo pero usando el formato **swap**. Quedando una vez terminado el sistema de particiones de la siguiente manera:

Y tras esto podremos continuar la instalación de manera normal.

3. Administración remota

En informática no siempre tenemos los equipos que administramos en nuestra oficina. Pueden estar en otro edificio, la oficina de un cliente, en internet … por lo tanto no siempre es posible acceder de manera física a ellos, y por tanto entra en juego la **administración remota**.

Podemos definir la administración remota como el sistema que nos permite realizar ciertas acciones "lanzadas" desde nuestro equipo local pero que serán ejecutadas en un equipo remoto.

Se pueden diferenciar varios tipos de sistemas dentro de la administración remota, pero nos vamos a centrar en los siguientes:

- **Cliente remoto**: Lanzamos una acción a ejecutar desde un equipo remoto a través de algún tipo de interfaz o comando (que viajará a través de un protocolo securizado) y esperaremos a la respuesta.
- **Acceso remoto**: En este caso lo que hacemos es conectarnos al equipo a través de un protocolo que nos va a permitir administrarlo como si estuviésemos delante de él.

Todos estos sistemas pueden ser complementarios, y puede que podamos administrar un mismo servicio de todas estas maneras, por lo que queda a nuestra disposición elegir el mejor método en cada momento.

Por otro lado, dependiendo de qué tipo de administración vayamos a llevar a cabo, o el protocolo que utilice, tendremos que tener acceso al servidor de alguna manera (ya sea conexión directa o mediante VPN).

Información

Dependiendo de la administración remota que realicemos, necesitaremos conexión directa o mediante VPN al equipo que nos queremos conectar.

Por último, también debemos de conocer el tipo de protocolo que vamos a utilizar al realizar la conexión remota y por dónde va a pasar esa comunicación. Siempre hay que premiar la seguridad de la comunicación, y más cuando esta puede pasar por redes no controladas. Por lo tanto, deberemos asegurar que el protocolo utilizado es seguro, o en caso contrario, securizarlo de alguna manera.

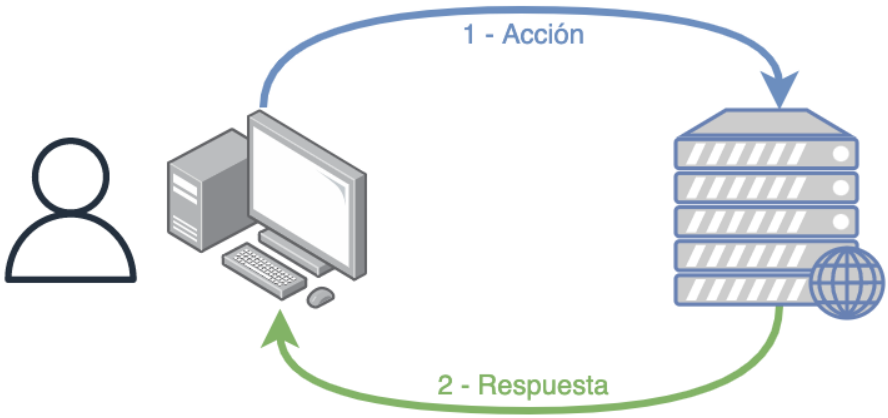
¡Cuidado!

Siempre debemos confirmar que la comunicación que se realiza para la administración remota viaja cifrada.

Más adelante veremos cómo securizar una comunicación no segura realizando un túnel mediante el protocolo SSH en entornos GNU/Linux.

3.1. Cliente remoto

Este sistema de administración permite enviar acciones al equipo remoto a través de un protocolo establecido, y dependiendo de la acción ejecutada se esperará una respuesta o no.



Hoy en día es muy habitual este tipo de sistemas a través de disintos **CLI** (*client line interface*) o **GUI** (*graphic user interface*) que nos permiten administrar servicios remotos. Por ejemplo:

- **https://www.mysql.com/MySQL**: El sistema gestor de base de datos MySQL cuenta con un cliente para realizar la conexión, ya sea desde el propio equipo o desde un equipo remoto.

  Este cliente se puede ejecutar desde línea de comandos, aunque también viene integrado en distintos interfaces gráficos como MySQL Benchmark, Dbeaver, ...

- **AWS CLI**: Es el interfaz de línea de comandos para poder administrar de manera remota los servicios contratados en la nube AWS de Amazon.

- **Gcloud CLI**: Similar al caso anterior pero esta vez para Google Cloud.

- **Remote Server Administration Tools for Windows 10**: En este caso se trata de un interfaz gráfico (**GUI**) que nos permite administrar un Windows Server desde un equipo Windows 10.

Antes de poder realizar la conexión remota con el cliente **debemos configurar un sistema de autenticación** para que el servicio remoto acepte las peticiones enviadas. En algunos casos será usando unos sistemas de certificados y en otros introduciendo un usuario y contraseña que establecerá una sesión temporal.

En el caso de AWScli y GCloud no nos estamos conectando directamente a nuestros servidores alojados en esas nubes, si no que lanzamos la orden a un "proxy" que verificará nuestros credenciales, verá los permisos que tenemos y después realizará la acción solicitada.

3.2. Acceso remoto

Este sistema permite acceder al sistema y podremos administrarlo como si nos encontrásemos delante. Dependiendo del sistema la conexión nos permitirá interactuar de alguna de las siguientes maneras:

- **CLI**: Mediante una conexión de línea de comandos. Es el caso más habitual en servidores GNU/Linux y la conexión se hace a través del protocolo seguro **SSH**.

- **GUI**: Podremos obtener un interfaz gráfico con el que veremos lo que está ocurriendo en pantalla en ese momento. En este caso, dependiendo del sistema, existirán distintas opciones, pero vamos a nombrar dos de ellas:

- **RDP**: Es el protocolo de escritorio remoto de Microsoft que transmite la información gráfica que el usuario debería ver por la pantalla, la transforma en el formato propio del protocolo, y la envía al cliente conectado. El problema es que este sistema desconecta al usuario que está logueado para poder hacer uso del escritorio remoto.
- **VNC**: En inglés *Virtual Network Computing*, es un servicio con estructura **cliente-servidor** que permite visualizar el escritorio del servidor desde un programa cliente. En este caso, no existe desconexión del usuario que está logueado y por tanto podrá ver lo que le están realizando de manera remota.

  Es muy habitual que los equipos de usuarios ya tengan la instalación del servidor hecha, para que de esta manera, en caso de incidencia, poder realizar la conexión remota sin que el usuario tenga que realizar ninguna acción.

A continuación se va a detallar algunos de los métodos mencionados.

3.3. SSH

SSH es un protocolo de comunicación segura mediante cifrado cuya función principal es el acceso remoto a un servidor. La arquitectura que utiliza es la de cliente-servidor.

Aunque el uso más habitual de SSH es el acceso remoto, también se puede utilizar para:

- Securizar protocolos no seguros mediante la realización de túneles.
- Acceder a un equipo saltando a través de otro.

Estas funcionalidades las veremos más adelante.

3.3.1. Servidor SSH

En el servidor al que nos queramos conectar deberá estar instalado el demonio/servicio SSH, conocido como **sshd**. Es habitual que ya esté instalado en sistemas GNU/Linux, pero de no ser así deberemos usar el sistema de paquetes de nuestra distribución para hacer la instalación. El nombre suele ser **openssh-server**.

Este servicio por defecto se pondrá a la escucha en el puerto 22/TCP:

SSHd escuchando en puerto 22

```
ruben@vega:~$ sudo ss -pntaln
State   Recv-Q   Send-Q   Local Address:Port   Peer Address:Port   Process
LISTEN  0        128      0.0.0.0:22           0.0.0.0:*           users:(("sshd",pid=1122,fd=3))
LISTEN  0        128      [::]:22              [::]:*              users:(("sshd",pid=1122,fd=4))
```

La configuración del servicio se realiza a través de un fichero de configuración que está situado en la ruta `/etc/ssh/sshd\_config`. Las distribuciones de GNU/Linux ya traen una configuración predeterminada que suele constar de las siguientes directivas (aunque hay muchas más):

- **Port**: Normalmente viene comentada, ya que el puerto por defecto es el 22. En caso de querer cambiar el puerto, podremos modificar esta línea, asegurando que no esté comentada.

- **ListenAddress**: Por defecto SSH se pondrá a la escucha en todos los interfaces que tengamos configurados. Si sólo nos interesa escuchar en alguna de las IPs que tengamos configuradas, deberemos modificar esta configuración.
- **PermitRootLogin**: Para evitar problemas de seguridad, esta directiva suele estar configurada a "**No**", para evitar que se puedan usar los credenciales de root para hacer el login.
- **PubkeyAuthentication**: Esta directiva permite realizar la conexión a través de unas claves públicas/privadas que podemos crear. Se explicará más adelante.

Hoy día también se puede instalar en Windows 10 y posteriores a través de un comando, siendo administrador de PowerShell:

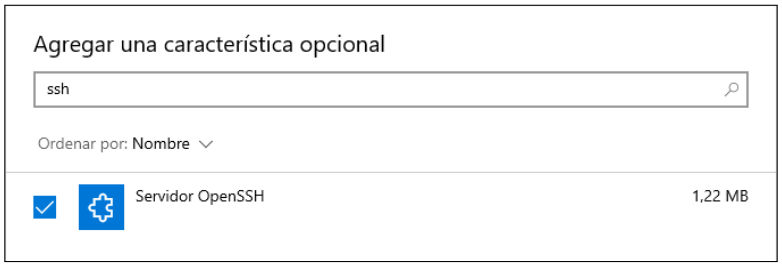
Instalando OpenSSH Server en Windows 10

```
PS C:\Windows\System32> Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0

PS C:\Windows\System32> Start-Service sshd

PS C:\Windows\System32> Set-Service -Name sshd -StartupType 'Automatic'
```

O desde el interfaz gráfico a través de las "**Características opcionales**", buscando por ssh:



3.3.2. Cliente SSH

El cliente SSH es aquel programa que a través del protocolo SSH se puede conectar a un servidor SSH. Existen distintos tipos de clientes que podemos utilizar:

- **CLI**: El cliente de consola es el más habitual. Está instalado de forma habitual en todas las distribuciones de GNU/Linux (normalmente el paquete se llama **openssh-client**). También lo encontramos instalado por defecto en MacOS.

Hoy día también está instalado en Windows 10, y de no estar, se puede instalar a través de las "**Características Opcionales**".

- **GUI**: Existen distintos interfaces gráficos que nos puede interesar utilizar:
  - **Putty**: Un cliente muy habitual en entornos Windows.
  - **Kitty**: Una versión mejorada del anterior.
  - **Termius**: Cuenta con versión de escritorio y móvil.

Para realizar la conexión al servidor SSH debemos conocer:

- **Dirección del servidor**: Ya sea mediante IP o nombre FQDN (*fully qualified domain name*) que se resuelva.
- **Puerto**: Ya hemos comentado que por defecto el puerto es 22.
- **Usuario**: Para realizar el sistema de autenticación, necesitamos un usuario que exista en el sistema.
- **Contraseña**: Los credenciales de acceso del usuario.

Para realizar la conexión desde un cliente de consola ejecutaremos:

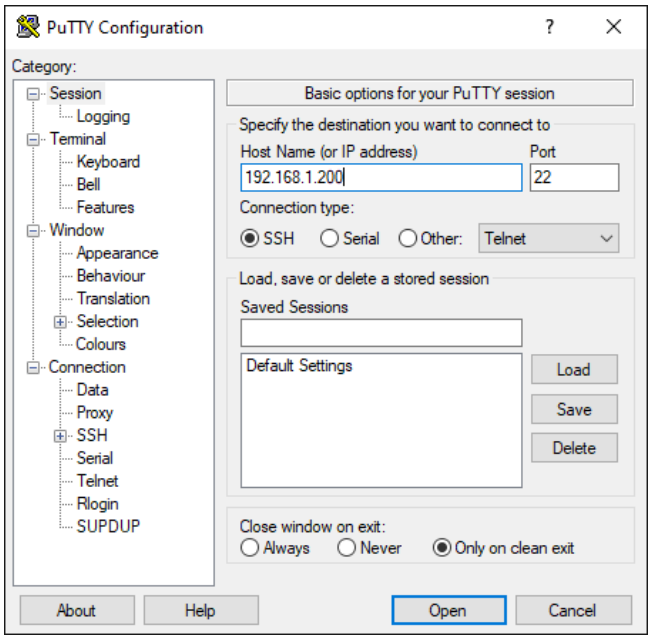
Conexión SSH

```
ruben@vega:~$  ssh usuario@192.168.1.200 -p 22
```

En el comando anterior podemos identificar:

- **ssh**: el cliente de consola
- **usuario**: el nombre del usuario con el que nos queremos conectar al servidor remoto.
- **@**: la arroba en inglés significa “at”, que indica “usuario en el servidor X”.
- **192.168.1.200**: La IP del servidor al que nos queremos conectar.
- **-p 22**: Estos dos parámetros van juntos, “-p” indica que vamos a indicar el puerto de conexión y “22” que nos queremos conectar a ese puerto. Debido a que 22 es el puerto por defecto, podríamos no poner estas opciones si sabemos que el servidor escucha en el puerto 22.

Si realizamos la conexión a través de un cliente de interfaz, como es putty, el aspecto será el siguiente, donde sólo podremos introducir la IP del servidor. Cuando se comience con la conexión nos pedirá los credenciales de acceso.



Si es la primera vez que nos conectamos a un servidor mediante SSH nos saldrá un mensaje como el siguiente:

Conexión SSH

```
ruben@vega:~$  ssh usuario@192.168.1.200 -p 22

The authenticity of host '192.168.1.200 (192.168.1.200)' can't be established.
ECDSA key fingerprint is SHA256:uK9MOl0gLDhTtCrlcafc1zVObVA/vnOMn6TWFsQb23o.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Este "key fingerprint" es un identificador que está relacionado con el fichero de "**clave pública**" del servidor. Es como el DNI del servidor. La primera vez se nos guarda ese *fingerprint*, y en caso de que en una próxima conexión varíe, nos avisará. No suele ser habitual que este identificador cambie.

3.3.3. Conexión mediante certificados de clave pública/clave privada

Existe una alternativa a la hora de realizar una conexión SSH para que no nos pida la contraseña del usuario, y es **hacer uso de los certificados de clave pública y clave privada**. Este concepto de "clave pública y clave privada" viene de la **criptografía asimétrica**.

Este sistema de criptografía asimétrica hace uso de dos claves que están asociadas entre sí:

- **Clave privada**: Es la base del sistema criptográfico, y como su nombre indica, se debe de mantener en privado. **Nunca se debe de compartir**, ya que entonces se podrían hacer pasar por nosotros.
- **Clave pública**: Asociada a la clave privada, la clave pública puede ser compartida y enviada a otros ordenadores para poder realizar la conexión.

Para generar el par de claves se realiza con el siguiente comando:

Crear par de claves pública/privada

```
ruben@vega:~$  ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/ruben/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):
Enter same passphrase again:

Your identification has been saved in /home/ruben/.ssh/id_rsa
Your public key has been saved in /home/ruben/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:SPqPOYBmPb8PCFhcZgqcWZPZzaL5RNfMeKmHqebvC7E ruben@vega
The key's randomart image is:
+---[RSA 3072]----+
|o +oB o = .      |
| * B.+ = *       |
```

```
|  + + + =        |
| o o + = .       |
|. .o+.o S        |
|  +.+*o          |
| o  +Eo          |
|     .+=         |
|      *B+        |
+----[SHA256]-----+
```

El comando muestra los siguientes pasos:

1. Creación de la pareja de claves pública/privada haciendo uso del sistema criptográfico **RSA**.
2. Lugar donde se va a guardar la clave privada. Por defecto en `\~/.ssh/id\_rsa`.
3. Contraseña para securizar la clave privada. De esta manera, para poder usarla habrá que introducir dicha contraseña. Dado que nosotros queremos evitar introducir contraseñas, lo dejaremos en blanco.
4. Lugar donde se va a guardar la clave pública. Por defecto en `\~/.ssh/id\_rsa.pub`

Una vez tenemos nuestro par de claves, podemos copiar la clave pública al usuario del servidor que nos interese mediante el siguiente comando:

Crear par de claves pública/privada

```
ruben@vega:~$  ssh-copy-id user@servidor_remoto
```

Para ello es imprescindible conocer previamente la contraseña del usuario en el servidor. El comando `ssh-copy-id` realizará una conexión SSH y copiará el contenido de la **clave pública**, `\~/.ssh/id\_rsa.pub`, dentro del fichero `\~/.ssh/authorized\_keys` del usuario en el servidor remoto. Este paso se puede realizar a mano (con un editor de texto).

¡Cuidado!

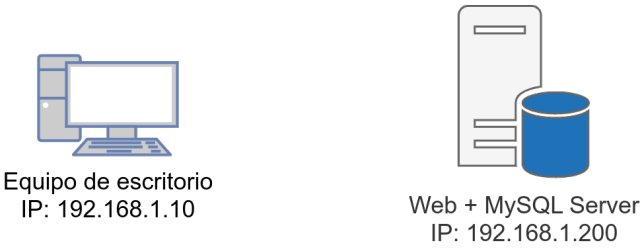
Windows no tiene el comando "ssh-copy-id", por lo que deberemos hacer el paso a mano, tal como se ha explicado.

Al realizar la siguiente conexión, ya no necesitaremos introducir la contraseña del usuario, ya que el sistema remoto podrá autenticarnos a través del sistema clave pública/privada.

3.3.4. Crear túneles SSH

Una de las funcionalidades extra de SSH es la posibilidad de crear "túneles" para securizar protocolos no seguros, o poder acceder a servicios que sólo escuchan en *localhost*.

Pongamos como ejemplo el siguiente escenario:



Tenemos un servidor web con Apache y MySQL al que queremos acceder. Por seguridad MySQL **sólo está escuchando en localhost (127.0.0.1)**, por lo que el acceso al servicio MySQL no es posible. Para administrarlo nos tenemos que conectar al servidor, y realizarlo de manera local.

En este punto es donde entra en juego la creación de un túnel seguro al servidor, para poder acceder al servicio remoto. **Para ello es imprescindible poder realizar una conexión SSH** (ya sea mediante usuario o clave pública/privada).

Para crear un túnel, desde el equipo de escritorio, lanzaremos el siguiente comando:

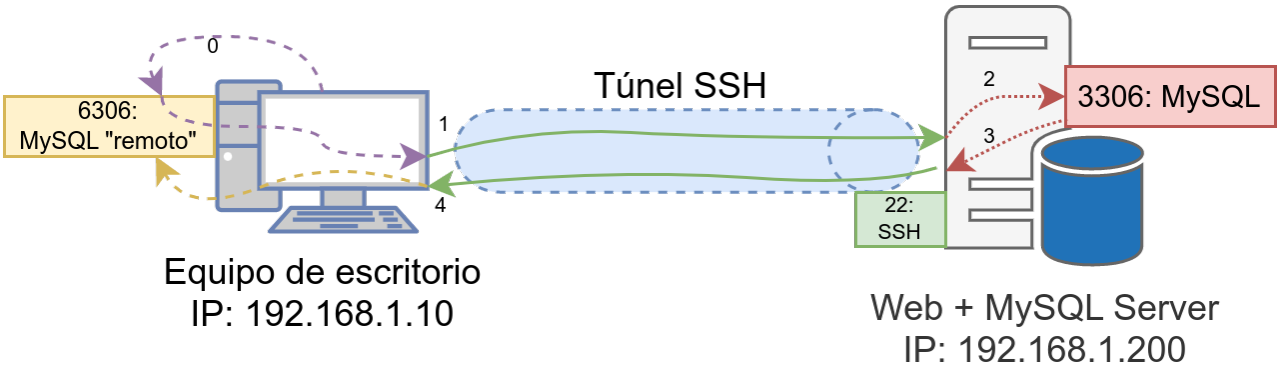
Crear par de claves pública/privada

```
ruben@vega:~$  ssh usuario@192.168.1.200 -L 6306:127.0.0.1:3306 -N
```

Al ejecutar este comando, habremos creado un túnel que enlaza el puerto remoto 3306 (que sólo se escucha en el "127.0.0.1" del servidor), con el puerto local 6306 del equipo de escritorio. A continuación la explicación del comando y sus parámetros:

- "**ssh usuario@192.168.1.200**": es como una conexión SSH normal. Lo que estamos indicando es que queremos conectarnos con "usuario" al servidor remoto 192.168.1.200 a través de SSH.
- **-L 6306:127.0.0.1:3306**: Especifica que el puerto local especificado se va redirigir al puerto e IP remota. Para entender esto hay que separar dos partes de los parámetros:
  - **6306**: Especifica la IP y el puerto local. En este caso, antes del puerto no hemos especificado IP, por lo que se creará un puerto 6306 que sólo se pone a la escucha en ***localhost*** en el equipo de escritorio
  - **127.0.0.1:3306**: Esta es la dirección y puerto remoto al que nos queremos conectar. En este caso, es el puerto 3306 que está escuchando en la IP 127.0.0.1 del servidor.
- **-N**: Sirve para que no ejecute ningún comando en el servidor remoto, y por tanto no nos abrirá conexión de terminal.

A nivel visual, y para entender de mejor manera lo realizado, sirva la siguiente imagen y los pasos que se pueden dar en un escenario real:



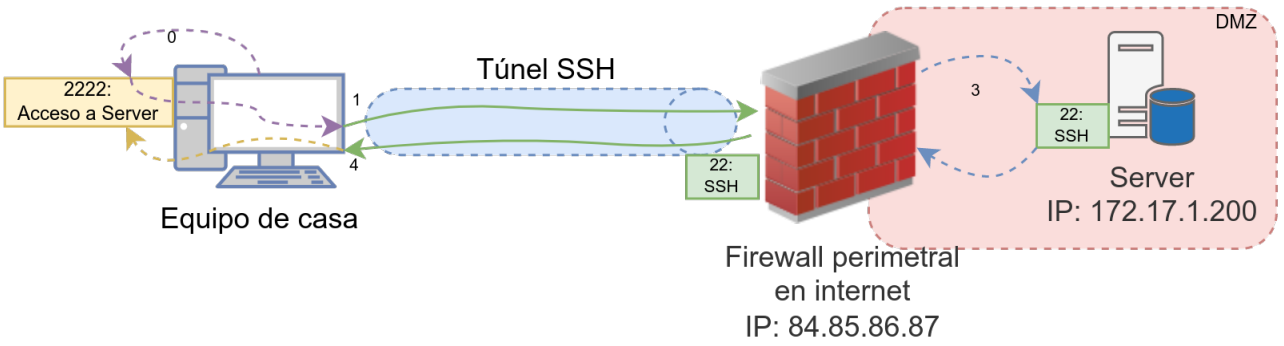
Con una aplicación en el equipo de escritorio queremos conectarnos al servidor MySQL que sólo escucha en el servidor remoto. Ejecutamos el túnel visto anteriormente, y los pasos que podremos realizar son los siguientes:

0. Mediante una aplicación nos podemos conectar al puerto local 6306, que ha sido creado mediante el comando anterior. Este puerto local está redirigido al puerto del servidor remoto. Por lo tanto la conexión se securiza a través del túnel
1. Como el túnel está establecido, la conexión viaja de manera segura a través de él.
2. Al llegar al servidor remoto, sabe que la conexión debe ir al puerto 3306 de la IP 127.0.0.1, que es lo establecido en el comando.
3. La conexión vuelve al túnel.
4. Viaja por el túnel hasta llegar a la comunicación que se había establecido previamente.

De esta manera, hemos podido realizar una conexión a un servicio remoto a través de SSH y completamente seguro.

3.3.4.1. Acceder a un equipo saltando a través de otro.

Este es un caso especial de túnel, similar a lo explicado previamente. En lugar de querer realizar una conexión a un servicio del equipo al que nos conectamos, en este caso lo usaremos de salto para acceder a otro servidor.



En este caso estando en casa nos queremos conectar a un equipo de la oficina. No tenemos VPN, y no hay redirección de puertos directa, pero podemos acceder al firewall perimetral. Por lo tanto, lo podemos utilizar para saltar al servidor que nos interesa.

En este caso, el comando a ejecutar sería:

Crear par de claves pública/privada

```
ruben@vega:~$  ssh usuario_firewall@84.85.86.87 -L 2222:172.17.1.200:22 -N
```

De esta manera, ahora desde nuestro equipo podremos realizar una conexión SSH al puerto 2222 que realmente será una redirección que viaja a través del túnel hasta el firewall, y que a su vez redirige al puerto 22 del servidor 172.17.1.200.

¡Atención!

El servicio remoto al que nos queremos conectar debe escuchar en la IP del equipo al que nos queremos conectar. El equipo que usamos para saltar debe poder conectarse a él.

4. Gestión de copias de seguridad (backups)

4.1. Introducción

Una copia de seguridad, o ***backup***, **es una copia de los datos originales** que se realiza con el fin de disponer de un medio **para recuperarlos en caso de pérdida**. Las copias de seguridad son útiles en caso de que los datos originales se hayan perdido por un fallo humano, fallo de sistema (rotura de disco duro), modificado por un virus, evento natural (incendio del edificio), un ataque (borrado deliberado de los datos), etc…

En el proceso de copia de seguridad también **se tiene que tener en cuenta el método de restauración de los datos**, que es el momento en el que necesitaríamos realizar la restauración de los datos del *backup* para dejarlos en la ubicación original.

A la hora de pensar un sistema de copias de seguridad, tendremos que tener en cuenta los siguientes puntos:

- Importancia de los datos
- Espacio ocupado por la copia de seguridad
- Ubicación de la copia de seguridad
- Plan de recuperación ante desastre
- Punto máximo que podemos recuperar
- Tiempo estimado de recuperación de los datos

Por otro lado, también tendremos que tener en cuenta el método que vamos a utilizar para realizar la copia de seguridad, y cómo va a ser el proceso. Este proceso dependerá del servicio/sistema al que vayamos a realizar el backup.

Y por último, la estrategia a utilizar:

- Estrategia multi-backup completos
- Incrementales y/o diferenciales
- Estrategia personalizada
- La que el programa que vayamos a utilizar nos permita
- …

Antes de realizar la copia de seguridad de los datos tendremos que tener muy en cuenta distintos apartados, que deberán ser analizados en detalle y puede que modificados en el futuro.

4.2. A tener en cuenta

Durante la **planificación** de la creación de las copias de seguridad hay ciertos aspectos que tenemos que tener en cuenta para que la gestión de copias de seguridad se realice de manera satisfactoria.

Esta planificación debe realizarse conociendo los datos que maneja la empresa, la importancia de los mismos, el volumen que ocupan, la cantidad de cambios que reciben, ...

Información

Una buena planificación para la gestión de copias de seguridad de una empresa puede involucrar a varias personas de distintos ámbitos internos.

4.2.1. Conocer los datos y la importancia de los mismos

Para saber sobre qué datos tenemos que realizar las copias de seguridad, debemos conocer los datos que manejamos en la empresa, la importancia que tienen y su ubicación. **Ejemplo**: no es lo mismo la importancia de los datos gestionados en la base de datos de clientes, o el directorio "Descargas" el PC de un usuario.

También tendremos que conocer el tamaño actual de esos datos, y saber cuánto volumen total alcanzan. Cuanto mayor tamaño tengan, también mayor será el espacio ocupado por las copias de seguridad.

Otro aspecto a tener en cuenta es saber las modificaciones que sufren esos datos, y estimar el volumen de esos cambios. **Ejemplo**: no es lo mismo que una base de datos reciba 10 clientes nuevos al día o que reciba 15.000 ventas a la hora en nuestra tienda online.

Información

Debemos conocer los datos que existen en nuestra empresa, la ubicación, la importancia y el volumen que ocupan para así poder realizar un buen plan de gestión de copias de seguridad.

4.2.2. Espacio ocupado por la copia de seguridad

Estimar el espacio que nos va a ocupar una copia de seguridad puede parecer sencillo, pero no siempre es así, y es por eso que conocer los datos, tal como hemos dicho antes, es de vital importancia.

Tendremos que tener en cuenta cuánto ocupan nuestros datos originales, cuál va a ser la estrategia de backup a utilizar, estimar el incremento de espacio que puede haber en los datos originales…

Por lo tanto, siempre deberíamos estimar por alto el espacio ocupado, y siempre pudiendo realizar la expansión de ese espacio "en caliente", para que las copias de seguridad funcionen cada día.

¡Cuidado!

No realizar copias de seguridad debido a falta de espacio debería ser considerado un fallo de primer orden y que debe ser subsanado lo antes posible.

4.2.3. Punto mínimo en el tiempo que queremos recuperar

Es posible tener copias de seguridad que estén completamente actualizadas en cuanto se realizan cambios en los datos originales, pero la puesta a punto de estos sistemas puede no estar a nuestro alcance (por no

conocer cómo se realizan o por un coste elevado).

En ciertos casos, algunas empresas pueden aceptar la pérdida de ciertos datos, ya que el gestionar que las copias se realicen “en tiempo real” es más caro que el repetir el proceso de crear esos datos de nuevo.

Ejemplo: Una empresa puede no necesitar hacer backup de la base de datos de cuándo fichan los empleados en tiempo real, porque si se pierden los datos de un día es asumible (el backup se hace cada noche). En cambio, la base de datos de marketing tiene que realizarse en tiempo real.

4.2.4. Punto máximo en el tiempo que queremos recuperar

Otro aspecto importante a tener en cuenta es **pensar cuánto tiempo para atrás queremos poder llegar para poder realizar la restauración de los datos**. No será lo mismo querer realizar una restauración de los datos de hace una hora, un día, de hace una semana o de hace un año, ya que esta decisión influirá en toda estrategia de copia de seguridad.

Es por eso, que a los datos hay que darle la importancia que se merece, y es bastante probable que distintos datos tengan distintos puntos de restauración máximo.

Ejemplo: No es lo mismo los datos de facturación de una empresa, que la ley nos marca que debemos guardar dichos datos (4 años), los datos de una página web que apenas cambia (igual nos sirve una copia a la semana) o la carpeta personal de un usuario en su equipo (dado que en su equipo no debería guardar nada importante de la empresa, se podría asumir una pérdida de datos).

4.2.5. Ubicación de la copia de seguridad

Es muy importante también tener en cuenta cuál va a ser la ubicación de la copia de seguridad y las consecuencias que eso puede conllevar. Vamos a poner varios ejemplos:

- Si decidimos hacer la copia de seguridad en el mismo equipo informático, un problema de electricidad podría suponer la pérdida de los datos originales y los backups.
- Si decidimos realizar la copia de seguridad en la misma sala de ordenadores donde se sitúan los datos originales, si esa sala se ve involucrada por un factor que destruya todo (un incendio, un robo, …), perderíamos todos los datos.
- Si decidimos realizar la copia en la misma oficina, edificio, puede pasar lo mismo que el punto anterior
- Si decidimos realizar la copia en la “nube”, el tiempo para realizar la copia de seguridad y el tiempo de restauración de los datos puede incrementarse
- Si decidimos realizar la copia en otra oficina, dependemos de la conexión, los datos deberían ir cifrados, tiempo de creación de la copia de seguridad y restauración…

Como se puede ver, cada ejemplo puede tener unos pros y unos contras.

¡Cuidado!

La copia de seguridad nunca debería estar en el mismo equipo informático, y mucho menos en el mismo disco duro físico, que los datos originales.

4.2.6. Tiempo estimado de la copia

Teniendo en cuenta lo comentado previamente, el tiempo estimado de la copia de seguridad es muy importante, y es posible que varíe en el tiempo. No es lo mismo realizar la copia de seguridad de unos pocos megabytes o de varios gigabytes.

Es por eso por lo que tendremos que tener en cuenta el tiempo que tarda en realizarse nuestras copias de seguridad para así asegurar que se realizan de manera correcta y no se solapan en el tiempo.

4.2.7. Plan de recuperación ante desastre

Tan importante es tener una copia de seguridad como un plan ante un posible desastre. De nada sirve tener una copia de seguridad si restaurarla nos va a llevar semanas por no saber el estado de las mismas, o si su restauración es compleja.

Por lo tanto, **es muy importante disponer de un plan de recuperación ante desastres que esté actualizado**, que contemple si ha habido modificaciones en la gestión de copias de seguridad, que tenga en cuenta los pasos a realizar…

Este plan de recuperación **debería ser puesto en práctica cada cierto tiempo para asegurar que sigue estando vigente**, y si no, realizar las modificaciones oportunas.

¡Atención!

Es muy importante disponer de un plan de recuperación ante desastres que esté actualizado y probar que siga vigente cada cierto tiempo.

El plan de recuperación ante desastre debería contener:

- Los distintos métodos y estrategias de copias de seguridad que existen en la empresa
- Ubicaciones donde se realizar las copias de seguridad
- Importancia de los datos de los que se realizar la copia de seguridad
- Cómo realizar la restauración de cada sistemas de los que se ha realizado la copia de seguridad
- Orden para realizar las restauraciones y dada la importancia de los datos, la prioridad de las restauraciones
- ...

Información

El plan de recuperación es algo que toda compañía debe tener a la hora de crear sus sistema de copias de seguridad, ya que deben de ir de la mano.

4.2.8. Tiempo estimado de recuperación de los datos

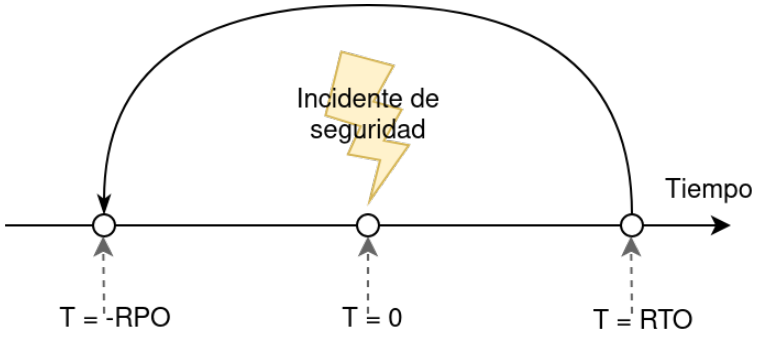
El último punto, pero no por ello menos importante, es el tiempo estimado de recuperación de los datos.

¡Cuidado!

De poco sirve tener copias de seguridad y un plan de recuperación, si luego es posible que estemos dos semanas para recuperar los datos.

Es por eso que tenemos que conocer el tiempo estimado de recuperación de los datos para poder gestionar los tiempos con los empleados, los clientes…

Este tiempo estimado, al igual que el tamaño de las copias, puede variar en el tiempo, por lo que **debemos asegurar cada cierto tiempo el realizar simulacros para asegurar qué tiempos se manejan**.



Teniendo en cuenta el gráfico anterior:

RTO: es el objetivo de tiempo de recuperación (*Recovery Time Objective*), y es el tiempo que se tarda en restablecer el servicio, o al menos los mínimos acordados. Cuanto menor sea el tiempo de recuperación, mejor.

RPO: es el objetivo de punto de recuperación (*Recovery Point Objective*), y es el último instante de tiempo previo al incidente al que los sistemas son capaces de regresar. Habitualmente suele ser marcado por la frecuencia con que se realicen las copias de seguridad. De nuevo, cuanto más cercano a T=0 mejor.

4.3. Métodos de Backup

Existen distintas estrategias a la hora de crear un backup, por lo que tendremos que analizar los datos que queremos salvaguardar y decidir la mejor estrategia.

4.3.1. Copiar los ficheros

Sería la metodología más sencilla. Simplemente haremos una copia de los datos importantes en otra ubicación/soporte. Este sistema hace que cada X tiempo, se guarden los ficheros y obtendremos una copia

completa de los datos.

El problema es que **no tendremos la posibilidad de recuperar un fichero en un estado anterior a la última copia**. Es decir, si se realiza la copia de seguridad a las 6:00 de la mañana, no podremos obtener el estado del fichero previo a ese. Si a las 11:00 el usuario corrompe el fichero, podríamos recuperar al estado de las 6:00, pero no al estado de ayer.

4.3.2. Sistema incremental a nivel de bloque

Se basa en copiar sólo los bloques físicos que han sido modificados. Tendremos que tener guardado el fichero original por un lado, y a partir de ahí sólo guardaremos las modificaciones de manera incremental.

Para recuperar el fichero, primero tendremos que elegir restaurar el fichero original completo, y posteriormente aplicar los bloques incrementales hasta llegar al punto de restauración que nos interesa.

4.3.3. Sistema incremental o diferencial binaria

Similar al anterior, pero esta vez en lugar de hacer uso de bloques (1K, 4K, 8K…), la parte incremental o diferencial sería a nivel binario.

4.3.4. Versionado de ficheros

El versionado de ficheros se puede realizar de distintas maneras:

- **Utilizando el sistema de ficheros**: Existen sistemas de ficheros que cuando uno es modificado, automáticamente crea una versión nueva del mismo.
- **Versionado manual**: cuando el usuario quiere crear una nueva versión del fichero, debe ejecutar a mano una acción para guardar la nueva versión del mismo.

4.4. Estrategias de backup

Existen muchas estrategias a la hora de realizar una copia de seguridad, quizá tantas como entornos existen, ya que se pueden utilizar estrategias generales o crear una para cada caso concreto. Vamos a explicar las más habituales.

4.4.1. Multi-backup completos

Este método sería el más sencillo, pero también el que más espacio nos ocuparía, y posiblemente el que más tiempo puede tardar, dependiendo de los datos a copiar.

En este método, podríamos hacer una copia completa de los datos que queremos guardar en tantas ubicaciones como queramos. Supongamos que queremos poder llegar a restaurar hasta 3 días de un fichero corrupto, eso supondría que tendríamos que tener 3 ubicaciones distintas donde cada día haríamos un backup completo de los datos. Por ejemplo:

- Ubicación **1**: el **lunes** haríamos el primer backup completo
- Ubicación **2**: el **martes** haríamos el segundo backup completo

- Ubicación **3**: el **miércoles** haríamos el tercer backup completo
- Ubicación **1**: el **jueves** haríamos el backup completo (machando los datos previos que había del lunes)
- …

Como se puede ver, es una estrategia cíclica, sencilla, y que en ciertos casos puede ser más que suficiente. Pero como se ha comentado previamente, cada día se realizará un backup completo, por lo que habrá que tener en cuenta el tiempo que tarda, el espacio que ocupa, ...

4.4.2. Incrementales y/o diferenciales

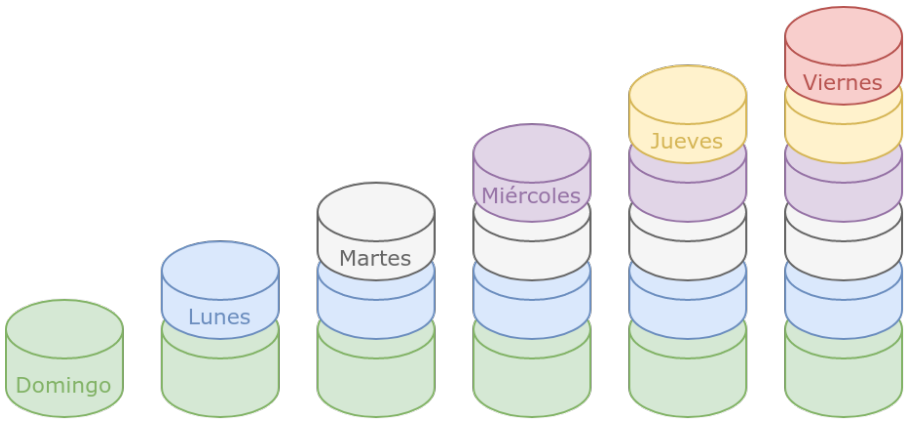
Normalmente es el método más habitual y todos los programas suelen tener estos métodos a la hora de realizar copias de seguridad.

El hacer uso de copias incrementales y/o diferenciales nos permitirá poder restaurar los datos hasta un punto concreto. Por eso será muy importante tener en cuenta cuánto tiempo queremos poder llegar a restaurar, porque no es lo mismo a la hora de estimar espacio ocupado, si queremos restaurar los datos de una semana, de hace un mes o de hace un año.

Por lo tanto, a la hora de utilizar esta metodología, como ya se ha comentado, que es la más habitual, la estimación de espacio debe de ser holgada, y pensar a futuro, en cuánto puede llegar a aumentar el espacio ocupado.

4.4.2.1. Incrementales

Supongamos que realizamos una copia completa de los datos el domingo, esta será la base para las siguientes copias incrementales de los datos:

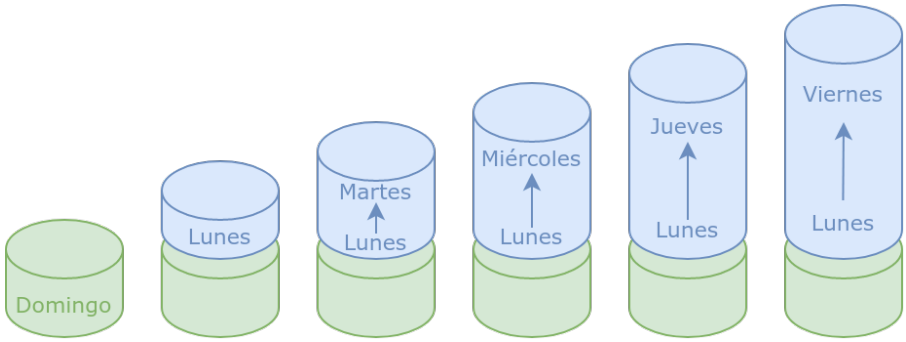


El lunes se hará una copia guardando sólo los datos que han cambiado respecto a la copia del domingo. El martes, se hará una copia de los datos que han sido modificados desde la copia del lunes… Y así sucesivamente.

Este método es rápido, fácil de realizar y sabemos claramente cuándo se han modificado los datos, por lo que si tenemos cada incremental separado en distintas carpetas, la restauración sería también sencilla.

4.4.2.2. Diferencial

Similar al incremental, pero cada día se guarda todos los datos que han sido modificados desde el último backup completo:



Esto hace que cada backup diferencial pueda ocupar más que un incremental, pero cada día obtendremos todos los cambios desde el último backup completo.

4.4.2.3. Incremental y diferencial

También suele ser habitual realizar la mezcla de ambas estrategias:

- El primer domingo de cada mes realizar la copia completa.
- Hacer uso del método incremental de lunes a sábado
- Los domingos realizar un diferencial de toda la semana.

4.4.3. Estrategia personalizada

Como se ha comentado previamente, pueden existir tantas estrategias como entornos existan, por lo que dependiendo de nuestros datos, nuestro entorno, nuestro modelo de negocio, tendremos que crearnos una estrategia propia.

Este método puede ser el más complejo, ya que quizá tengamos que realizarlo a mano, pero con ello nos aseguraremos de darle la importancia que se merece a cada parte de los datos que tengamos que realizar un backup.

4.4.4. La que el programa nos marque

Si hacemos uso de un programa de gestión de backups, al final estaremos limitados a lo que el programa nos permita. Como se ha comentado, la estrategia incremental/diferencial suele ser la más ampliamente utilizada, por lo que casi todos los programas deberían tener esta opción.

Dependiendo del programa, nos permitirá hacer mezclas de estrategias, realizar copias a sistemas remotos (la nube, por FTP, SSH…), varios cada día… Por lo que quedaría en nuestra mano analizar distintos programas y ver cuál es el que mejor se adecúa a nuestras necesidades.

4.5. Regla 3-2-1

Con todo lo dicho anteriormente, una de las estrategias más habituales, de manera generalizada y simplificada, es la conocida como la "**Regla 3-2-1**".

Esta regla se resume en:

- **3 copias completas** de los datos.
- En **2 tipos de almacenamiento distintos**
- **1 de las copias debe estar fuera**

Imaginemos una base de datos de una tienda online que está situada en un servidor en un RACK junto con otros servidores nuestros en un proveedor contratado. En esta base de datos tenemos **los datos originales**.

Haremos uso de otro de los servidores del RACK en el que realizaremos una copia de seguridad completa. Esta será la **segunda copia completa**, que está en otro servidor y que podremos acceder de manera rápida a ella.

La **tercera copia** de los datos la tendremos en nuestra oficina, ya que realizaremos una copia remota de los datos. Esta tercera copia está situada fuera del proveedor contratado, por lo que si en ese CPD hubiese un incendio, tendríamos una copia en local.

4.6. Realización de simulacros de recuperación de los datos

De nada sirve lo comentado hasta ahora, si no se realizan simulacros de desastre y de recuperación de los datos para asegurar que todo funciona de manera correcta. Desgraciadamente, **también puede haber fallos en los sistemas de backups**, por lo que aunque creamos que se están realizando correctamente, quizá el día que vayamos a necesitarlos nos damos cuenta que no es así, por lo que no podríamos realizar la recuperación de datos.

¡Cuidado!

Deberíamos crear simulacros de desastre para asegurar que todos los eslabones de la cadena de nuestro sistema de copias de seguridad es funciona de manera correcta.

Estos simulacros nos deberían ayudar a asegurar que:

- Las copias de seguridad funcionan de manera correcta
- Podemos realizar restauraciones de los datos a fecha que nos interesa
- El plan de recuperación ante desastre está actualizado y es funcional
- El tiempo estimado de recuperación es el adecuado una vez restaurados los datos, nuestro sistema es funcional y podemos seguir con la actividad comercial

¡Atención!

Si alguno de los puntos anteriores falla y no es correcto, deberemos repasar todo el plan estratégico de copias de seguridad para solventar los fallos, y así poder volver a realizar un nuevo simulacro para confirmar que esta vez es correcto.

5. Sistemas de Monitorización

El sistema de monitorización se encarga de recopilar información acerca del estado de los servidores de una infraestructura. Entre las métricas y datos que debe recopilar se encuentran:

- **Estado del servidor**: cantidad de RAM utilizada, estado de los discos duros, carga del servidor, sistema de ficheros, …
- Estado de los **servicios que tiene el servidor**: servidor web, número de procesos que tiene, bases de datos, conexiones existentes, cola de correos electrónicos para enviar si es un servidor de correo, … Dependiendo del servicio habrá que realizar unas comprobaciones u otras.
- **Infraestructura en la que se encuentran**: estado de la red, conexión a otros servidores, …
- **Estado del clúster**: en caso de que el servidor pertenezca a un clúster, hay que comprobar que el clúster se encuentra en perfecto estado.
- **Dependencias externas**: un servidor puede depender a su vez de otros, o de servicios externos, que deben de estar funcionando de manera correcta.

Normalmente en la monitorización actúa un **agente** (o servicio) instalado en el equipo monitorizado que obtiene la información requerida que se envía a un servidor central que recopila la información de toda la infraestructura monitorizada.

Esta información suele ser almacenada durante un periodo de tiempo determinado (un año, por ejemplo) para poder ser usada y comparar la situación de los servidores a lo largo del tiempo. Gracias a esta comparación temporal **se puede llegar a predecir el estado del servidor** a unos días/semanas vista y evitar problemas antes de que sucedan (no tener espacio en discos duros, mal funcionamiento de servicios por falta de RAM, …).

Información

La monitorización de servicios y equipos dentro de una infraestructura debe considerarse parte del proyecto, ya que es una parte muy importante de cara al mantenimiento del mismo.

5.1. Monitorización de servidores

Es habitual que los sistemas de monitorización funcionen en base a plantillas, que posteriormente se pueden asociar a los servidores monitorizados. Estas plantillas contendrán los servicios que deben ser monitorizados en cada tipo de servidor, ya que no es lo mismo monitorizar un servidor web o un servidor con un SGBD.

Para monitorizar un servidor lo habitual suele ser realizar las siguientes operaciones:

- **Comprobar conectividad con el servidor**: suele ser habitual que los servidores estén fuera de nuestra red (en un proveedor de Internet, en un cliente, en otra oficina…), por lo que es necesario

que exista conectividad de alguna manera para poder realizar la monitorización. En caso de no estar en nuestra red, el uso de una VPN es lo más utilizado.

- **Instalar un agente en el propio servidor**: Será el encargado de recopilar la información necesaria para mandarla al servidor central. Dependiendo del sistema de monitorización utilizado, necesitaremos un tipo de agente u otro. Algunos se encargan de realizar todas las comprobaciones y otros llaman a otros programas para realizar las comprobaciones y después mandar el resultado al servidor central.
- **Dar de alta el servidor en el sistema centralizado**: Tal como se ha dicho previamente, lo habitual es contar con un sistema centralizado en el que se tendrán todos los servidores y el estado de las comprobaciones realizadas. De ser así, habrá que darlo de alta, y para ello se necesitará:
  - **Nombre del servidor**: Un nombre que a simple vista identifique el servidor. Suele ser habitual poner el nombre del cliente también, y/o el tipo de servicio que preste.
  - **IP del servidor**: Para poder realizar la conexión al servidor.
  - **Plantillas asociadas**: En caso de utilizar un sistema que utilice plantillas, al dar de alta el servidor se le aplicarán las plantillas necesarias para que realicen todos los checks oportunos. Por ejemplo: plantilla de Servidor Linux + plantilla de Servidor web + Plantilla de MySQL.
  - **Puerto de conexión**: Los agentes de monitorización suelen contar con un puerto que queda a la escucha. Si hemos cambiado el puerto, habrá que indicarlo a la hora de dar de alta.
  - **Otras opciones**: Dependiendo del sistema de monitorización se podrán añadir muchas más opciones, como por ejemplo:
    - **Servidores de los que se depende**: Imaginemos que el servidor monitorizado depende a su vez de un router que también está monitorizado. Si el router cae, no llegaríamos al servidor, por lo que realmente es una caída por dependencia, aunque el servidor puede estar funcionando de manera correcta. Esto nos puede permitir crear “árboles de dependencias” de servidores.
    - **Periodos de monitorización**: Lo habitual es que un servidor esté monitorizado 24x7, pero quizá nos interese realizar cambios y que sólo se monitorice en unas horas determinadas (quizá el resto del tiempo está apagado).
    - …

5.2. Funcionamiento de la monitorización

Para conocer cómo funciona un sistema de monitorización lo mejor es que tomemos como ejemplo un tipo de servicio que queremos monitorizar. Como ejemplo se puede tomar las comprobaciones que queremos realizar a un SGBD (Sistema Gestor de Bases de Datos).

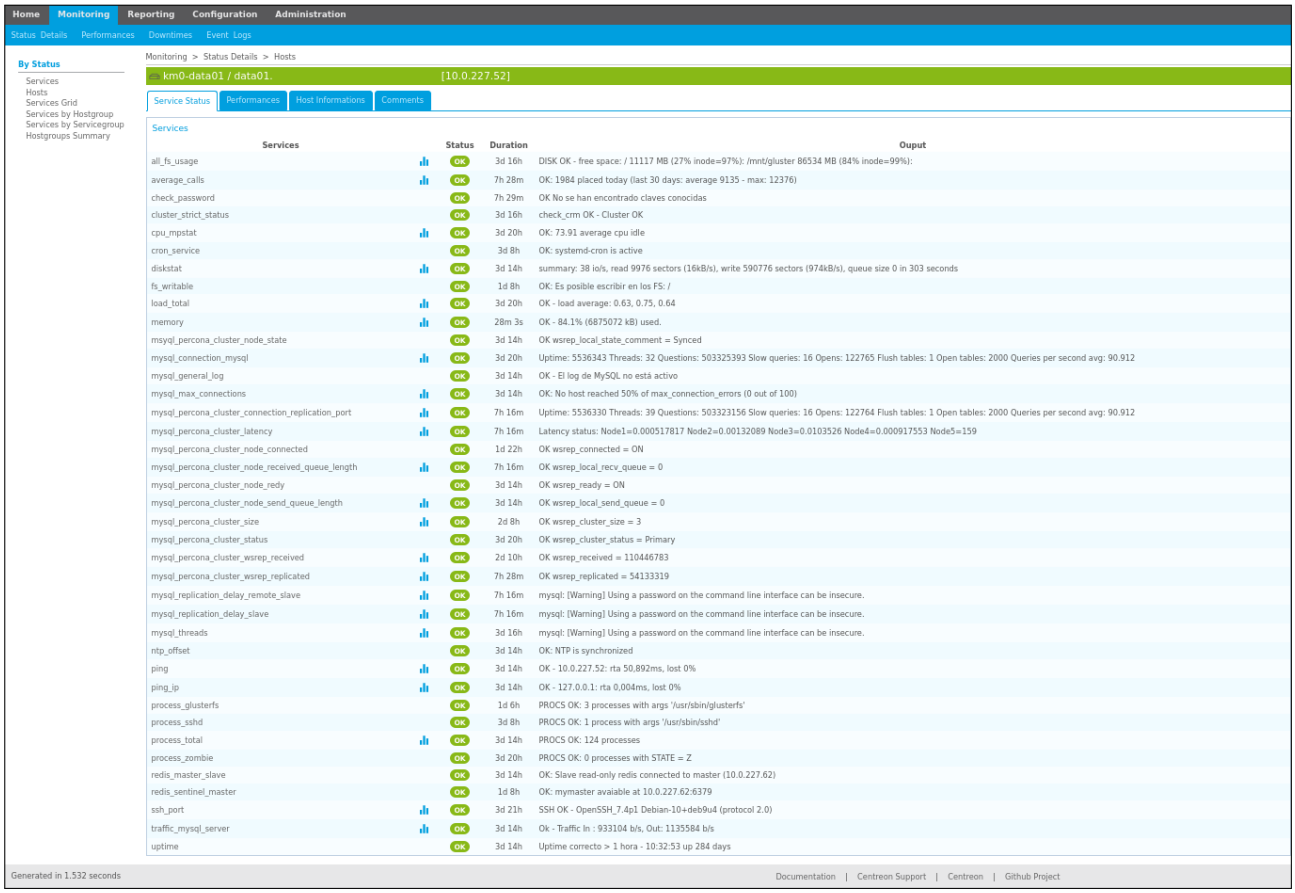
No será lo mismo realizar la monitorización de un servidor MySQL o de un Oracle, pero las comprobaciones

que queremos realizar en ellos deberían ser similares. Vamos a querer realizar la monitorización de las mismas comprobaciones: estado de las tablas en memoria, número de hilos en ejecución, número de *slow_queries*, …; pero los scripts ejecutados serán distintos.

Información

La monitorización dependerá del propio servicio que vayamos a monitorizar.

A continuación se puede ver el estado de un servidor monitorizado a través del sistema de monitorización Centreon:



Servidor monitorizado en Centreon

En la imagen anterior se puede comprobar un número de ***checks***, o comprobaciones, que se están realizando sobre un servidor concreto. Cada fila es una comprobación y contienen:

- **Nombre del check/servicio**: Un nombre para identificar qué es lo que se está comprobando con el check.
- **Icono para mostrar gráficas**: Algunos checks recibirán información que puede ser graficada para así poder observar patrones en el comportamiento del servidor. Por ejemplo: cantidad de RAM ocupada, número de procesos en el sistema, número de conexiones a un servidor, …
- **Estado del check**: Normalmente, tras realizar la comprobación, el check termina con uno de los

siguientes resultados:

- **OK**: El resultado obtenido es el correcto.
- **Warning**: El resultado obtenido está entre los márgenes de peligro. Es posible que de seguir así pase al estado siguiente:
- **Crítico**: El servicio devuelve un estado que es considerado crítico, lo que puede hacer que llegue a mal funcionamiento del mismo, o incluso que el servidor comience a dejar de funcionar (imaginemos que el servidor está con el 90 % de la RAM ocupada o de disco duro ocupado).
- **Indeterminado**: Por alguna razón, el *check* no se ha realizado, o el valor devuelto es indeterminado o no se puede saber en qué otro estado situarlo.

- **Duración del estado**: Para conocer cuánto tiempo lleva en el estado la comprobación obtenida. Lo ideal es que nunca haya estados que no sean OK y por lo tanto la duración de los mismos sea lo más alta posible.
- **Valor devuelto por la monitorización**: El valor real devuelto por la comprobación realizada. En base a este resultado se puede realizar las gráficas mencionadas previamente.

Información

El estado del servicio dependerá del valor devuelto por la monitorización.

Este resultado se cotejará con los valores que hayamos puesto para que sea considerado OK, Warning o Critical. Es decir, **en algunos casos el estado del servidor depende de los valores devueltos y de la baremación que le hayamos otorgado**.

Pongamos como ejemplo la monitorización de un SGBD:

- **El servicio del SGBD está funcionando**: Ahí no hay baremación posible. Si el servicio no está arrancado, es lógico pensar que el estado es crítico y que por tanto hay que ver qué ha ocurrido.
- **Número de conexiones en el SGBD**: El resultado devuelto será un número entero (que podremos graficar para obtener patrones). En este caso, podemos decidir los rangos para que el resultado sea OK, Warning o Critical. Es decir, si el resultado obtenido está por debajo del umbral de Warning, el sistema considerará que el estado es OK. Si está en dicho rango, será Warning y si está en el rango de Critical, así lo indicará.

Esta baremación y **estos rangos** se suelen aplicar también en las plantillas de los servicios. Hay que entender que también **pueden ser modificados y personalizados para un servidor concreto**. No es lo mismo que un SGBD tenga 500 conexiones simultáneas si tiene 8Gb de RAM o si tiene 128Gb (en el primer servidor se puede considerar que es un estado crítico mientras que en el segundo es lo esperado).

Cuando un *check* termina siendo un Warning o un Critical **es habitual que haya un sistema de alarmas configurado**. Dependiendo del sistema utilizado, notificará a los administradores mediante e-mail,

mensajería instantánea, SMS, … para que realicen un análisis lo antes posible y solucionen el estado del servicio.

Información

Los sistemas de monitorización suelen contar con un sistema de alarmas para que nos avise de los servicios caídos.

5.2.1. Monitorización básica

Tal como se ha comentado, en los servidores se suele realizar una monitorización del estado del mismo que suele ser común para todos, por lo que lo habitual suele ser tener una plantilla genérica para todos los servidores con la que se monitorizará:

- Cantidad de RAM utilizada
- Cantidad de memoria virtual utilizada
- Carga de la CPU
- Espacio libre en las unidades de disco duro
- Estado del sistema RAID del servidor (en caso de tenerlo)
- Cantidad de usuarios conectados a la máquina
- Estado de puertos de conexión (SSH, por ejemplo)
- Latencia hasta llegar al servidor
- …

Es cierto que no será lo mismo monitorizar un sistema GNU/Linux o un sistema Windows (ya que puede variar alguno de las comprobaciones a realizar), pero el estado general que queremos conocer es el mismo. Por lo tanto, lo habitual es tener dos plantillas, una específica para servidores Windows y otra para GNU/Linux.

5.2.2. Monitorización de Servicios

Aparte de la monitorización básica comentada previamente, necesitaremos monitorizar el estado de los servicios que pueda tener el servidor propiamente dicho. Para ello, de nuevo, se crearía una plantilla específica para cada tipo de Servicio que podamos tener en nuestro servidor.

No será lo mismo monitorizar un servidor que tenga un servidor web, un servidor de base de datos, un proxy… O puede que el servidor cuente con todos esos servicios.

Es por eso que a la hora de realizar la monitorización de un servidor **es muy importante conocer qué funciones desempeña cada servidor en la infraestructura a la que pertenece** y analizar los servicios que tiene arrancados para posteriormente ser monitorizados.

Información

Es muy importante conocer qué funciones desempeña cada servidor en la infraestructura a la que pertenece.

5.3. Tipos de monitorización

Existen varias maneras de realizar la monitorización de un servidor, y dependerá del gestor de monitorización que usemos (en caso de usar uno).

Es habitual que cuando nos referimos a sistemas de monitorización lo dividamos en dos grandes familias:

- Monitorización Activa
- Monitorización Pasiva

Estas dos maneras de monitorización suelen ser excluyentes, aunque algunos sistemas de monitorización permiten ambas, por lo que nos puede interesar usar una u otra dependiendo de la situación.

5.3.1. Monitorización pasiva

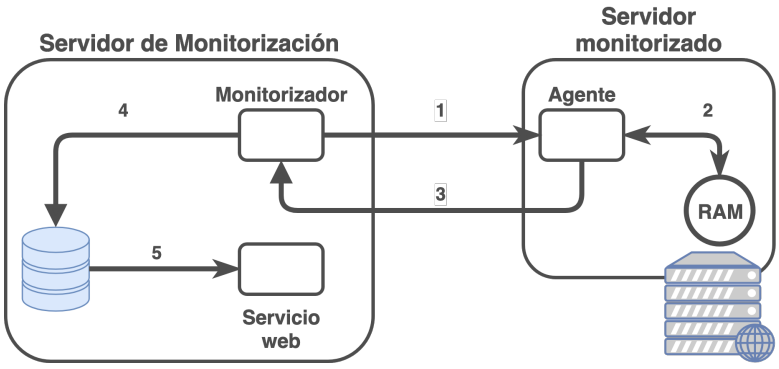
En la monitorización pasiva el servidor (u objeto monitorizado) es el encargado de mandar la información de manera periódica al servidor central. El agente instalado se ejecutará como una tarea programada cada cierto tiempo (habitualmente unos pocos minutos) e informará de la situación cambiante, de haberla, al servidor central.

Esta manera de monitorización es utilizada también cuando no hay un servidor central. En este caso, si la comprobación ha sido incorrecta, podría mandar un mail al administrador del servidor.

5.3.2. Monitorización activa

Suele ser la manera habitual de proceder de los sistemas que cuentan con un servidor centralizado de monitorización. El servidor de monitorización se encarga de preguntar al servidor, a través de la conexión con el agente, por la comprobación de alguno de los checks, y el agente devuelve la información.

A continuación se puede observar las etapas que existen en un sistema de monitorización activa utilizando un servidor de monitorización central:



Proceso de monitorización activa

Las etapas serían:

0. El sistema de monitorización tiene un scheduler (o planificador) que decide cuándo tiene que realizar cada comprobación (normalmente, cada pocos minutos).
1. El servicio encargado de monitorizar **establece conexión con el agente remoto** y le pide que compruebe un estado. En este ejemplo se ha optado por la RAM.
2. El agente en el servidor que se quiere monitorizar recibe la notificación y realiza una **comprobación local** (normalmente llamando a scripts locales) para obtener la cantidad de RAM ocupada, total y libre que tiene
3. El agente envía al sistema de monitorización el resultado obtenido en la ejecución de los scripts del paso anterior.
    1. El monitorizador al recibir el resultado, lo coteja con los rangos de baremación que tiene y decide si el check está en estado OK, Warning o Critical.
    2. Lo habitual es que si el resultado del servicio no es OK, se ejecute en el servidor de monitorización algún tipo de alarma (ya sea enviar un mail, sistema de mensajería, …) para notificar a los administradores.
4. El sistema de monitorización guarda en una base de datos los resultados obtenidos para así poder realizar posteriores análisis o comprobaciones temporales de los mismos.
5. Esos datos se suelen visualizar en una interfaz web, tal como hemos visto previamente.

Estos pasos son ejecutados de manera continuada en el servidor de monitorización para cada comprobación que se realiza en cada uno de todos los servidores que se monitorizan. Por lo tanto, se entiende que el propio servidor de monitorización también tiene que ser monitorizado ya que es de vital importancia que su estado sea óptimo.

5.3.3. Monitorización centralizada

Como ya se ha comentado, es el sistema habitual de monitorización. Las ventajas que podemos obtener al hacer uso de este sistema son muchas, pero se pueden destacar las siguientes:

- **Monitorización centralizada**: Aunque parezca obvio, el tener un único sistema en el que concentrar toda la información es muy útil y eficaz.
    - La alternativa sería tener una monitorización distinta en cada servidor.
- **Interfaz web**: Hoy en día suele ser habitual que los sistemas de monitorización tengan un servicio web en el que visualizar todos los datos obtenidos.
- **Sistema de plantillas**: De nuevo, es lo habitual, lo que hace que la gestión de monitorización de servidores sea más cómoda.
- **Gestión de usuarios**: Podremos tener usuarios que puedan ver unos servidores u otros, por lo que podemos tener equipos especializados en distintos grupos de monitorización y que sólo se enfoquen en ellos.

- Esto también es útil para dar acceso a los clientes a la monitorización de sus propios servidores.

5.3.4. Monitorización reactiva

La monitorización reactiva se puede definir como el sistema de monitorización que no sólo se encarga de comprobar y recibir el estado de los servidores, si no que también reacciona a los mismos para tratar de solucionar los problemas encontrados. Tras esta definición está la idea de que **existen ciertos fallos recurrentes que no siempre necesitan la intervención humana para solucionarse**, y que por tanto, se puede tratar de ejecutar antes de que sea considerado un problema real.

Como **ejemplo sencillo** se puede poner **el espacio libre en disco duro**. Imaginemos que se comprueba que apenas hay espacio en el disco duro de un servidor. En este caso, el sistema de monitorización recibirá que el servidor **está al 99.95 %** de espacio ocupado, y por tanto, en lugar de notificar a un humano indicando el estado crítico, **el sistema reacciona de manera automática tratando de liberar espacio**. Se habrá configurado previamente que en la reacción de este error trate de borrar ficheros temporales, vaciar papelera, limpiar ficheros de caché de ciertas rutas … Una vez hecho esto, se volverá a comprobar el estado del servidor. Si el espacio ocupado en disco duro ha bajado y está en modo OK no habrá que hacer nada más, y se habrá evitado que un administrador tenga que realizar dicha tarea. Si por el contrario el estado sigue siendo incorrecto, el sistema notificará el error para que se realice un análisis y se solucione el problema.

Como **ejemplo extremo** (que no suele ser habitual configurarlo así), imaginemos que **la RAM consumida por un SGBD es muy alta** y esté poniendo en peligro el estado del servidor, se podría configurar para que **el sistema reaccione reiniciando el SGBD para que libere la RAM** y vuelva a prestar servicio.

5.4. Gestores de monitorización

Hoy día existen muchos sistemas de monitorización, y dependiendo de nuestras necesidades deberemos optar por uno u otro. A continuación se expondrán varios ejemplos de gestores de monitorización basados en Software Libre, aunque la gran mayoría de ellos cuentan con un sistema dual. Es decir, se puede descargar y montarlo en tu propio servidor o puedes contratar a la empresa para que ellos tengan el servicio central:

- **Nagios**: Se puede considerar uno de los sistemas de monitorización más conocidos y del que se han basado otros. Generó mucha comunidad de administradores creando muchos scripts/plugins para hacerlos funcionar con él. Estos mismos scripts suelen ser utilizables en otros sistemas de monitorización.
- **Centreon**: Originalmente se creó como interfaz web para Nagios, pero poco a poco fue sustituyendo partes de Nagios hasta terminar siendo un sistema de monitorización completo. Existe la posibilidad de realizar la instalación por paquetes, descargar el sistema operativo en una ISO que te instala todo o incluso una máquina virtual con todo ya instalado y con configuración básica. (Demo).
- **PandoraFMS**: Sistema de monitorización creado por el español Sancho Lerena Urrea. Al igual que los anteriores, tiene sistema dual y la instalación se puede realizar por varios métodos.
- **Cacti**: Sistema más sencillo que los anteriores y habitualmente utilizado sólo en servidores sueltos,

es decir, no de manera centralizada.

- **Munin**: Igual que el anterior, ideal para monitorizar unos pocos servidores, ya que no se puede considerar un sistema centralizado como los primeros. Ver anexo de instalación de Munin.

Existen otros sistemas de monitorización basados “en la nube”, cuya funcionalidad es similar a lo expuesto previamente. Para hacer uso de estos sistemas nos descargamos un agente, lo instalamos y se encargará de mandar la información a los servidores de la plataforma contratada. Lógicamente, dependiendo del gasto realizado obtendremos más o menos servicios. Entre este tipo de servicios se pueden destacar:

- New Relic
- DataDog

6. Instalar sistema de monitorización Munin

A continuación se detalla cómo realizar la instalación y configuración básica de Munin como sistema de monitorización.

6.1. Introducción

Munin es un sistema de monitorización que se instala en cada servidor que queramos monitorizar. Es un sistema de monitorización sencillo que automáticamente monitoriza el propio servidor. Munin puede utilizarse para:

- Monitorizar el propio servidor
- Crear gráficas con los resultados de monitorización
- Usar plugins para monitorizar distintos servicios
- Realizar avisos sencillos por mail por situaciones anómalas

Munin necesita de un servidor web, como puede ser Apache, para poder visualizar las gráficas que crea con los datos obtenidos de la monitorización. Estas gráficas no son más que datos que se obtienen de la monitorización, que son almacenados en pequeñas bases de datos y que mediante un proceso CRON se generan las imágenes con dichos datos.

Aunque Munin puede ser suficiente para la monitorización de un servidor, no se suele utilizar en casos en los que se tenga más de 10-15 servidores, ya que hay que ir servidor a servidor para realizar la configuración, por lo que dificulta la administración.

Munin posee un sistema básico de monitorización centralizada, pero dista mucho de lo que se puede considerar un sistema de monitorización centralizada real. Tal como se ha dicho, para pocos servidores puede ser suficiente, pero no para infraestructuras grandes.

La ventaja que se obtiene al utilizar Munin es que es una instalación muy sencilla, tal como veremos a continuación, y que por tanto, para infraestructuras pequeñas puede ser útil.

6.2. Instalación

Para que Munin funcione se va a necesitar instalar el propio Munin y Apache como servidor web para mostrar la web en la que se visualizan los datos obtenidos de la monitorización.

Para realizar la instalación, ejecutaremos:

Instalar Munin y Apache2

```
root@vega:~# apt install munin apache2
```

Tras la ejecución de este comando, ya se habrá instalado los servicios necesarios para que Munin funcione y el servidor web Apache.

6.2.1. Configuración de Apache

La configuración para que Munin se pueda ver a través del servidor web Apache se encuentra en el fichero de configuración `/etc/munin/apache24.conf`. Normalmente esta configuración ya está aplicada en el Apache tras realizar la instalación, y se puede ver que existe un enlace simbólico, que si seguimos desde `/etc/apache2/conf-enabled/munin.conf` nos llevará al fichero mencionado previamente.

Esta configuración permite ver la web de Munin pero sólo si estamos conectados desde el propio servidor, a modo de seguridad sólo permite conexiones desde "localhost". Esto no suele ser lo habitual en servidores, por lo que vamos a modificar la configuración para poder acceder a esta web desde cualquier IP.

¡Cuidado!

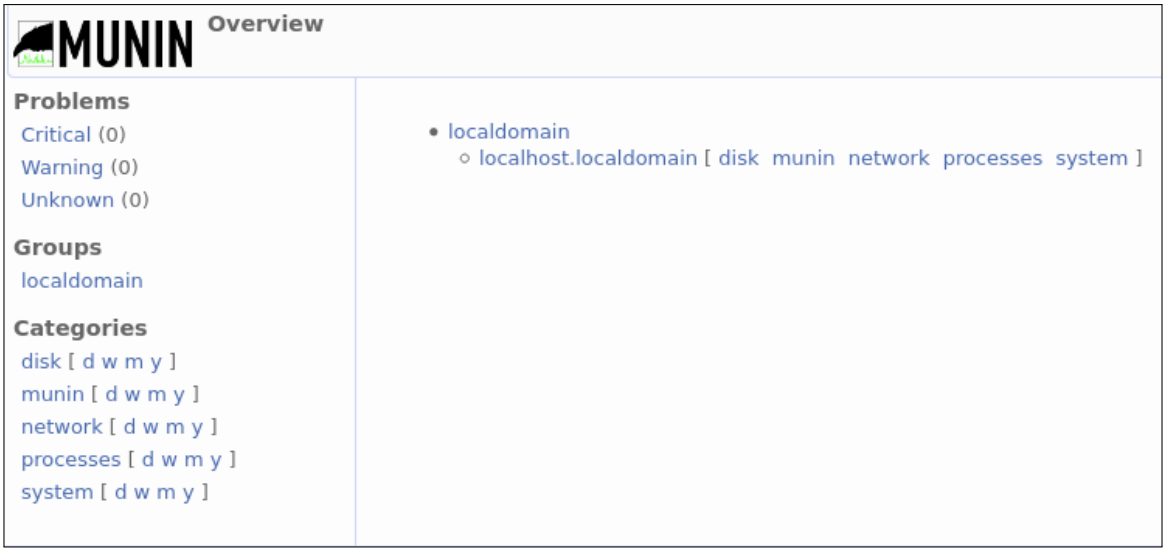
Permitir el acceso desde cualquier IP es un fallo de seguridad. Habría que modificar la configuración y poner autenticación con "auth-basic".

Para ello, modificaremos el fichero arriba mencionado y modificaremos las líneas donde aparece "Require" y pondremos "**Require all granted**". Tras realizar la modificación de la configuración de Apache, deberemos reiniciar el servicio:

Reiniciamos Apache para que coja la configuración

```
root@vega:~# systemctl restart apache2
```

Y tras esto, ya deberíamos poder ir al navegador web y poniendo **http://IPSERVIDOR/munin** (donde IPSERVIDOR es la IP del servidor que acabamos de configurar) y ver el interfaz de Munin:



Interfaz de Munin

6.3. Configuración

Tras la instalación, Munin por defecto tiene una configuración sencilla que hace una monitorización básica del servidor. La configuración se puede encontrar en `/etc/munin/` y podemos diferenciar 3

ficheros/directorios:

- **munin.conf**: fichero de configuración principal. En él se puede configurar todo lo relacionado con Munin, avisos, procesos...
- **munin-node.conf**: es el fichero de configuración que se utiliza si se utiliza el sistema básico de monitorización centralizada. En este fichero especificaremos desde qué servidores permitiremos que puedan monitorizar el propio servidor.
- **plugins**: directorio con los plugins activados que se van a utilizar durante la monitorización. En este directorio están los enlaces simbólicos de los plugins que van a ser utilizados durante la monitorización del servidor. Los plugins están en `/usr/share/munin/plugins/`. Normalmente suelen ser scripts en los lenguajes Perl, Python o Shell.

6.4. CRON

La monitorización del servidor se realiza a través de un proceso CRON que instala Munin y que por defecto se hace cada 5 minutos. El fichero CRON está situado en `/etc/cron.d/munin`, y cuenta con 4 procesos automáticos que se ejecutan en el servidor:

- Proceso de monitorización
- 3 procesos de limpieza de caché y ficheros temporales que son creados durante la monitorización

7. Virtualbox y adaptadores de red

7.1. Introducción

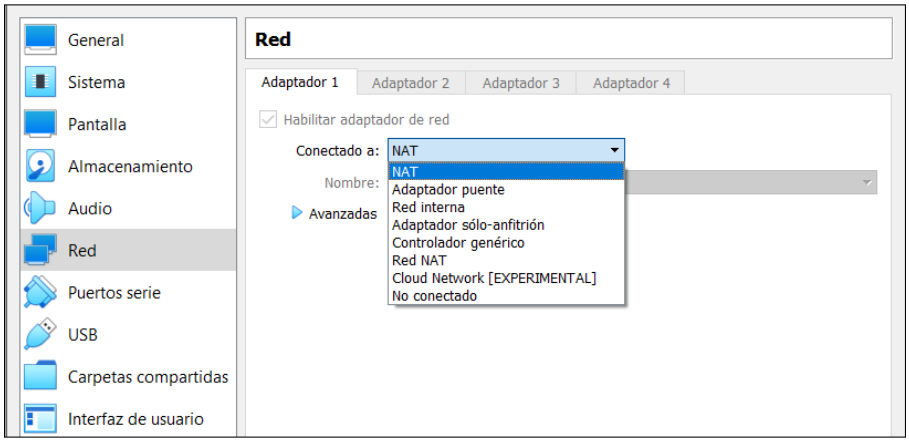
Virtualbox es una herramienta de virtualización para crear máquinas virtuales de manera sencilla. Es multiplataforma por lo que se puede utilizar en Windows, MacOS y Linux y aparte, es Software Libre.

Este documento no va a entrar en detalle en cómo se crean las máquinas virtuales, sino que va a explicar los distintos modos y adaptadores de red que puede tener una máquina virtual en este sistema de virtualización.

7.2. Adaptadores de red

Virtualbox permite que cada máquina virtual cuente con hasta cuatro adaptadores de red, lo que comúnmente se llaman interfaces o NIC (network interface controller).

Al crear las máquinas virtuales sólo tienen un único adaptador activo y suele estar configurado en modo NAT, pero tal como se ve a continuación, en el desplegable se puede ver que existen otras opciones:



En la documentación oficial aparece la explicación de los distintos modos, y es buena práctica leer y entender la documentación del software que utilizamos. También hay que entender que cada tipo de adaptador contará con una serie de ventajas y una serie de limitaciones que aparecen reflejadas en la documentación. Estos modos son comunes a otros sistemas de virtualización (VmWare, Proxmox, …), pero el nombre o el modo de uso puede variar así como las posibles limitaciones que puedan existir.

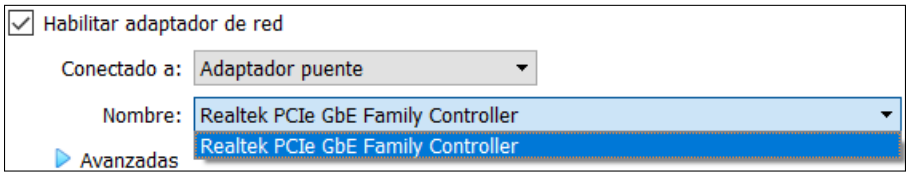
A continuación se va a dar una pequeña introducción a cada tipo de adaptador.

7.2.1. Adaptador puente

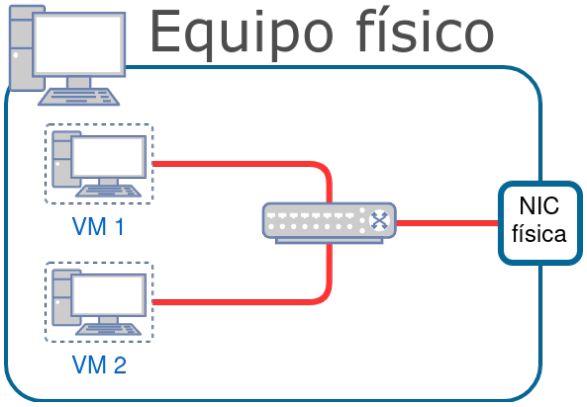
Es el tipo de adaptador que se usará si queremos que las máquinas virtuales aparezcan en la red física como si fueran un equipo más. Para poder entenderlo de mejor manera, podríamos pensar que este tipo de adaptador lo que hace es crear un “switch virtual” entre las máquinas virtuales y el interfaz físico, por lo que es como si fueran un equipo más en la red física.

Si el equipo físico anfitrión cuenta con más de un NIC (por ejemplo, en un portátil el NIC por cable y el NIC

wifi) tendremos que elegir en la máquina virtual sobre qué NIC queremos hacer el puente. En la siguiente imagen en el desplegable sólo se puede seleccionar un interfaz porque el equipo sólo cuenta con un NIC físico.



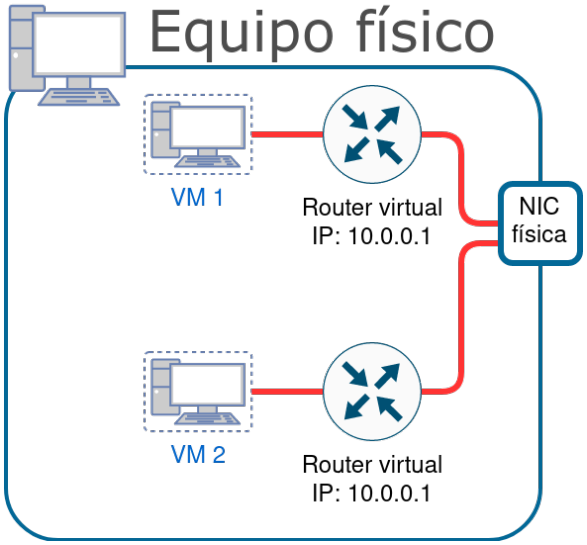
Es el método utilizado cuando virtualizamos servidores, ya que podrán dar sus servicios a toda la red.



Red como Adaptador Puente

7.2.2. NAT

Cada máquina virtual contará con su propio “router virtual” que hará NAT, y por eso todas las máquinas virtuales que usen este modo suelen tener la misma IP, pero no pertenecen a la misma red. Por defecto no se puede realizar conexión desde la red física al equipo virtualizado.



Red en modo NAT

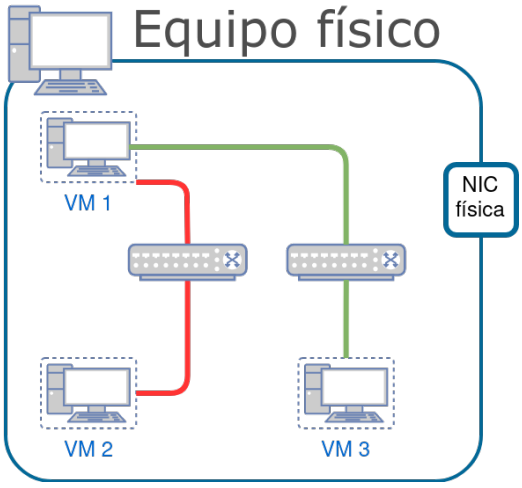
7.2.3. Red interna

Este tipo de adaptador lo que hace es “crear” un “switch virtual” que unirá las distintas máquinas virtuales que estén conectadas al nombre de esa red interna.

En el siguiente ejemplo la VM1 tiene 2 NICs, cada una con una red interna distinta. La VM2 tiene un NIC conectado a una de las redes internas creadas previamente y VM3 está conectada a la otra red interna.

Virtualbox no se encarga de dar IPs en estas redes, por lo que deberemos configurar cada interfaz de la máquina virtual con el direccionamiento que nos interese.

Este método se utiliza si queremos comunicar máquinas virtuales entre sí y que estén aisladas, ya que no podrán conectarse con el exterior, ni siquiera con el propio equipo físico.

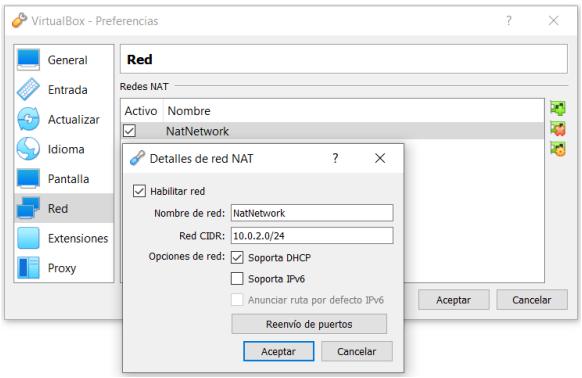


Red Interna

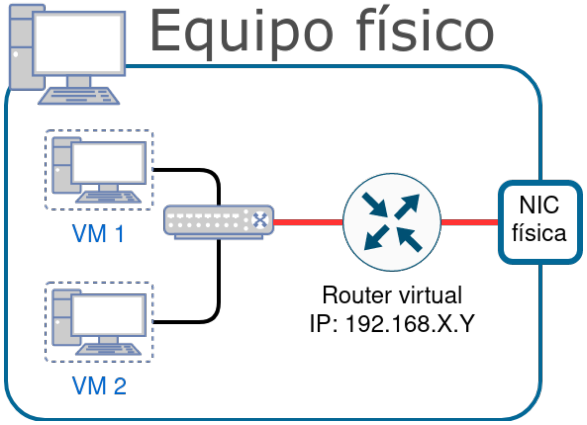
7.2.4. Red NAT

Podría definirse como una mezcla de NAT y red interna. Las máquinas virtuales podrán pertenecer a una única red, se podrán comunicar entre ellas, estarán detrás de un NAT de la red física y se podrán comunicar con el exterior.

Para poder usar ese modo hay que crear la “red NAT” en Virtualbox yendo a “*Archivo → Preferencias → Red*” y ahí se creará las redes NAT que queramos con el direccionamiento interno que nos interese.



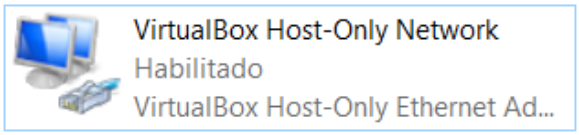
A la hora de crear la máquina virtual y elegir la opción “Red NAT” se podrá elegir entre las redes creadas en el paso anterior.



Red NAT

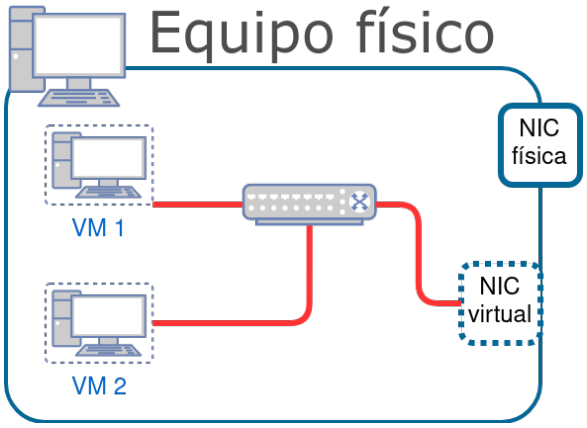
7.2.5. Adaptador sólo-anfitrión

Este tipo de adaptador es similar al de "red interna" pero con la posibilidad de comunicarse con el equipo físico anfitrión. En el equipo físico se crea un interfaz virtual y a través de él se podrá comunicar con las máquinas virtuales.



El direccionamiento que existe entre las VMs y el host se define en Virtualbox, dentro de "*Archivo* → ***Administrador de red de anfitrión***". Las máquinas virtuales podrán coger IP de ese direccionamiento por DHCP.

Mismo uso que "red interna" pero añadiendo la opción de comunicarnos con el host anfitrión.



Red en modo "Sólo Anfitrión"

7.3. Resumen de los adaptadores

A continuación se expone una tabla que resume los distintos tipos de adaptadores que existen y la conectividad posible entre las máquinas virtuales que usan esos adaptadores y el host anfitrión (fuente).

Modo \ Conectividad	VM → Host	VM ← Host	VM1 ← → VM2	VM → LAN real	VM ← LAN real
Adaptador puente	✔	✔	✔	✔	✔
NAT	✔	Redirección de puertos	✘	✔	Redirección de puertos
Red interna	✘	✘	✔	✘	✘
Sólo-anfitrión	✔	✔	✔	✘	✘

En la documentación se explica cómo realizar la redirección de puertos.