

PERFIL del cargo a cubrir:

- Capacidad para instalar y administrar las tecnologías descritas: Administración de servidores Linux; Plataformas de virtualización (Proxmox VE, Linux, Container, QUEMU); Plataformas de contenerización(Docker); Programación básica (linux shell y python); Sistemas de gestión de base de datos mysql, mariadb,postgres (instalación, configuración y backup).
- Capacidad para programar scripts para administración/backup y tareas referidas a la administración deservidores.

Material de estudio

Esta es una guía con material extraído de las fuentes oficiales de documentación de los temas incluidos en la evaluación. Para profundizar cualquiera de ellos se han colocado los enlaces a la documentación completa (marcado en color rojo como “**Documentación Oficial**”)

Parte 1: Administración de servidores Linux

Version en español de The Debian Book 11

Documentación Oficial : <https://www.debian.org/doc/manuals/debian-handbook/index.es.html>

1.1. EL MANUAL DEL ADMINISTRADOR DE DEBIAN

1.1.1. ¿Qué es Debian?

Debian es una distribución GNU/Linux. Más adelante veremos con más detalle qué es una distribución en la Sección 1.5, “El papel de las distribuciones”, pero por ahora nos limitaremos a decir que es un sistema operativo completo, incluyendo el software y los sistemas para su instalación y gestión, todo ello basado en el núcleo Linux y software libre (en especial del proyecto GNU).

Cuando creó Debian en 1993, bajo la dirección de la (FSF), Ian Murdock tenía unos objetivos claros que expresó en el Manifiesto Debian. El sistema operativo libre que buscaba tendría que tener dos características principales. En primer lugar, la calidad: Debian se desarrollaría con el mayor cuidado, para ser dignos del núcleo Linux. También sería una distribución no comercial, lo suficientemente creíble como para competir con las principales distribuciones comerciales. Esta doble ambición, a su modo de ver, sólo podía lograrse mediante la apertura del proceso de desarrollo de Debian al igual que la de Linux y del proyecto GNU. Por lo tanto, la revisión entre pares mejoraría continuamente el producto.

Un sistema operativo multiplataforma

Debian, manteniéndose fiel a sus principios iniciales, ha tenido tanto éxito que, hoy en día, ha alcanzado un tamaño enorme. Actualmente existen 9 arquitecturas de hardware compatibles oficialmente y varias variaciones de cada arquitectura conocidas como “sabores”, y también otros núcleos como FreeBSD (aunque las adaptaciones basadas en FreeBSD no forman parte de las arquitecturas soportadas de forma oficial). Por otra parte, con más de 31.000 paquetes fuente, el software disponible puede satisfacer casi cualquier necesidad tanto en casa como en la empresa.

Solo el tamaño de la distribución puede ser un inconveniente: no es muy razonable distribuir 18 DVD-ROMs (con solo los paquetes calificados como “Free Software”) para instalar la versión completa en un equipo estándar... es por eso que Debian se considera cada vez más una «metadistribución», desde la que se pueden extraer distribuciones más específicas orientadas a un público concreto: Debian Science para uso científico, Debian Edu para uso educativo y pedagógico en entornos académicos, Debian Med para aplicaciones médicas, Debian Jr. para niños, etc. Puede encontrar una lista más completa de los subproyectos en Sección 1.3.3.1, “Existing Debian Sub-Projects and Blends”, dedicada a ese fin.

Estas vistas parciales de Debian se organizan en un marco bien definido, lo que garantiza compatibilidad sin problemas entre las diferentes subdistribuciones. Todas ellas siguen la planificación general para el lanzamiento de nuevas versiones. Y dado que están construidas sobre la misma base, pueden extenderse, completarse y personalizarse fácilmente con las aplicaciones disponibles en los repositorios de Debian.

Todas las herramientas de Debian operan con esto en mente: `debian-cd` permite desde hace tiempo crear conjuntos de CD-ROMs que sólo contengan un conjunto de paquetes preseleccionados, `debian-installer` es también un instalador modular que se adapta fácilmente a las necesidades especiales, `APT` permite instalar paquetes de varios orígenes garantizando al mismo tiempo la consistencia global del sistema.

1.1.5. El papel de las distribuciones

Una distribución GNU/Linux tiene dos objetivos principales: instalar un sistema operativo libre en un equipo (sea con o sin uno o más sistemas preexistentes) y proveer un rango de programas que cubran todas las necesidades del usuario.

1.1.5.1. El instalador: `debian-installer`

`debian-installer`, diseñado de forma extremadamente modular para ser tan genérico como sea posible, apunta al primer objetivo. Cubre un gran rango de situaciones de instalación y, en general, facilita enormemente la creación de un instalador derivado para adecuarse a un caso particular.

Esa modularidad, que también lo hace muy complejo, puede desalentar a los desarrolladores que descubren esta herramienta; pero la experiencia del usuario es similar cuando lo utiliza tanto en modo gráfico como en modo texto. Se ha dedicado mucho esfuerzo reduciendo la cantidad de preguntas realizadas al momento de instalar, particularmente gracias a la inclusión del software de detección automática de hardware.

Es interesante remarcar que las distribuciones derivadas de Debian son muy diferentes en este aspecto y sólo proveen un instalador más limitado (generalmente sólo para las arquitecturas `i386` y `amd64`) pero más amigable al usuario no iniciado. Por el otro lado, generalmente evitan desviarse demasiado en el contenido de los paquetes para poder beneficiarse lo mayor posible del amplio rango de software ofrecido sin causar problemas de compatibilidad.

1.1.5.2. La biblioteca del software

Quantitatively, Debian is undeniably the leader in this respect, with over 31,000 source packages. Qualitatively, Debian's policy and long testing period prior to releasing a new stable version justify its reputation for stability and consistency. As far as availability, everything is available on-line through many mirrors worldwide, with updates pushed out every six hours.

La mayoría de los nuevos programas libres ingresan rápidamente a la versión de desarrollo que les permite ser instalados. Si esto necesita de demasiadas actualizaciones debido a sus dependencias, el programa puede ser recompilado para la versión estable de Debian (revise el Capítulo 15, Creación de un paquete Debian para más información sobre este tema).

1.1.6. Ciclo de vida de una versión

El proyecto tendrá de tres a seis versiones diferentes de cada programa simultáneamente, llamadas «Experimental» (experimental), «Unstable» (inestable), «Testing» (pruebas), «Stable» (estable), Oldstable (antigua estable) e incluso Oldoldstable (antigua «Oldstable»). Cada una de las corresponde a una fase diferente en el desarrollo. Para entender mejor, veamos la travesía

de un programa desde su empaquetado inicial hasta su inclusión en una versión estable de Debian.

1.1.6.1. El estado experimental: Experimental

Primero revisemos el caso particular de la distribución Experimental: este es un grupo de paquetes Debian que corresponde a software que está actualmente en desarrollo y no necesariamente completado, explicando su nombre. No todo pasa por este paso, algunos desarrolladores agregan paquetes aquí para recibir comentarios y sugerencias de usuarios más experimentados (y valientes).

De lo contrario, esta distribución generalmente alberga modificaciones importantes a paquetes base, cuya integración a Unstable con errores serios tendría repercusiones críticas. Es, por lo tanto, una distribución completamente aislada, sus paquetes nunca migran a otra versión (excepto intervención directa y expresa de su responsable o los ftpmaster). Además, no es autocontenida: sólo un subconjunto de los paquetes existentes están presentes en Experimental y generalmente no incluye el sistema base. Por lo tanto, esta distribución es más útil combinada con otra distribución autocontenida, como Unstable.

1.1.6.2. El estado inestable: Unstable

Volvamos al caso típico de un paquete. Su responsable crea un paquete inicial que compila para la versión Unstable y la ubica en el servidor ftp-master.debian.org. Este primer evento involucra una inspección y validación de parte de los ftpmaster. El software luego está disponible en la distribución Unstable, la «cresta de la ola» elegida por los usuarios que prefieren paquetes más actualizados en lugar de menos errores. Ellos descubren el programa y lo prueban.

Si encuentran errores, los reportan al encargado del paquete. Quien prepara versiones corregidas regularmente que vuelve a subir al servidor.

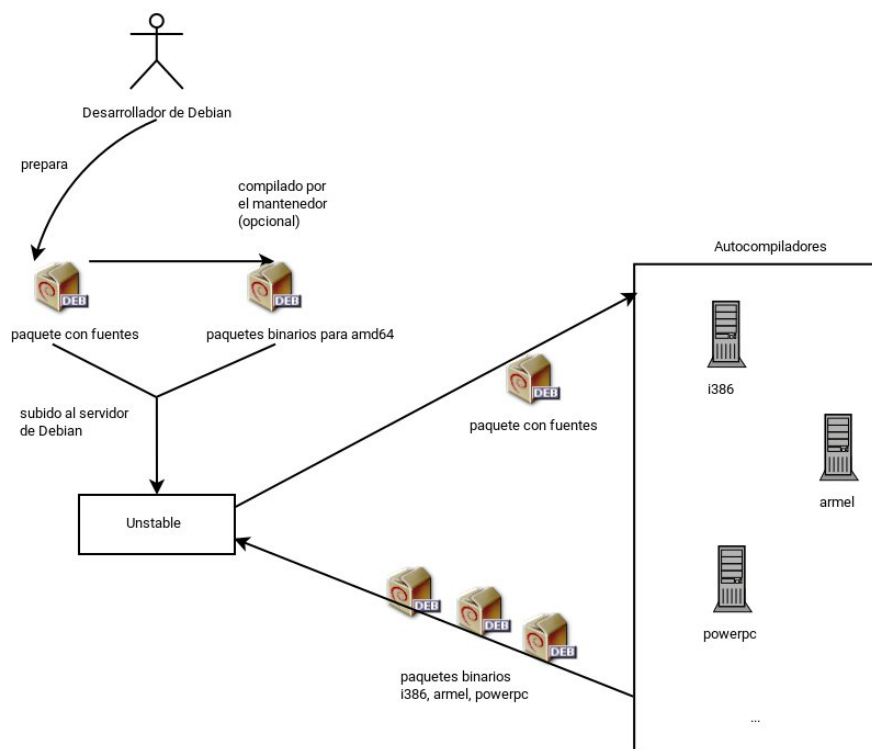


Figura 1.2. Compilación de un paquete por los autobuilders

1.1.6.3. Migración a Testing

Luego, el paquete habrá madurado; compilado en todas las arquitecturas, y no tendrá modificaciones recientes. Será entonces candidato para ser incluido en la distribución de pruebas: Testing — un grupo de paquetes de Unstable elegidos según un criterio cuantificable. Todos los días, un programa selecciona los paquetes a incluir en Testing según elementos que garanticen cierto nivel de calidad:

1. compilación satisfactoria en todas las arquitecturas oficiales;
2. falta de fallos críticos o, al menos, menor cantidad que la versión incluida ya en Testing;
3. al menos 5 días en Unstable, que es normalmente suficiente tiempo para encontrar y reportar problemas serios (pasar con éxito la batería de pruebas del propio paquete, si lo tiene, reduce ese tiempo);
4. dependencias que puedan ser satisfechas en Testing o que, por lo menos, puedan moverse allí junto al paquete en cuestión;
5. los controles de calidad automáticos del paquete (autopkgtest) —si está definido— no muestran ninguna regresión.

Este sistema no es infalible; se encuentran regularmente errores críticos en los paquetes incluidos en Testing. Aún así, generalmente es efectivo y Testing tiene muchos menos problemas que Unstable, convirtiéndola para muchos en un buen compromiso entre estabilidad y novedad.

1.1.6.4. La promoción desde Testing a Stable

Supongamos ahora que nuestro paquete se incluye en Testing. Mientras tenga margen de mejora, el responsable del mismo debe continuar mejorando y volver a iniciar el proceso desde Unstable (aunque generalmente su posterior inclusión en Testing será más rápida: a menos que haya cambiado significativamente todas sus dependencias ya se encuentran disponibles). El desarrollador completa su trabajo cuando alcanza la perfección. El siguiente paso es la inclusión en la distribución Stable que, en realidad, es una simple copia de Testing en un momento elegido por el administrador de versión. Lo ideal sería que esta decisión se tome cuando esté listo el instalador y cuando no exista ningún programa en Testing que tenga errores críticos conocidos.

Ya que este momento nunca llega realmente, en la práctica Debian llega a un compromiso: eliminar paquetes en los que su encargado no corrigió los errores a tiempo o acordar publicar una versión con algunos errores en los miles de programas. El gestor de versiones habrá anunciado previamente un período de estabilización («freeze»), durante el cual cada actualización a Testing debe ser aprobado. El objetivo aquí es evitar cualquier versión nueva (y nuevos errores) y sólo aprobar correcciones de errores.

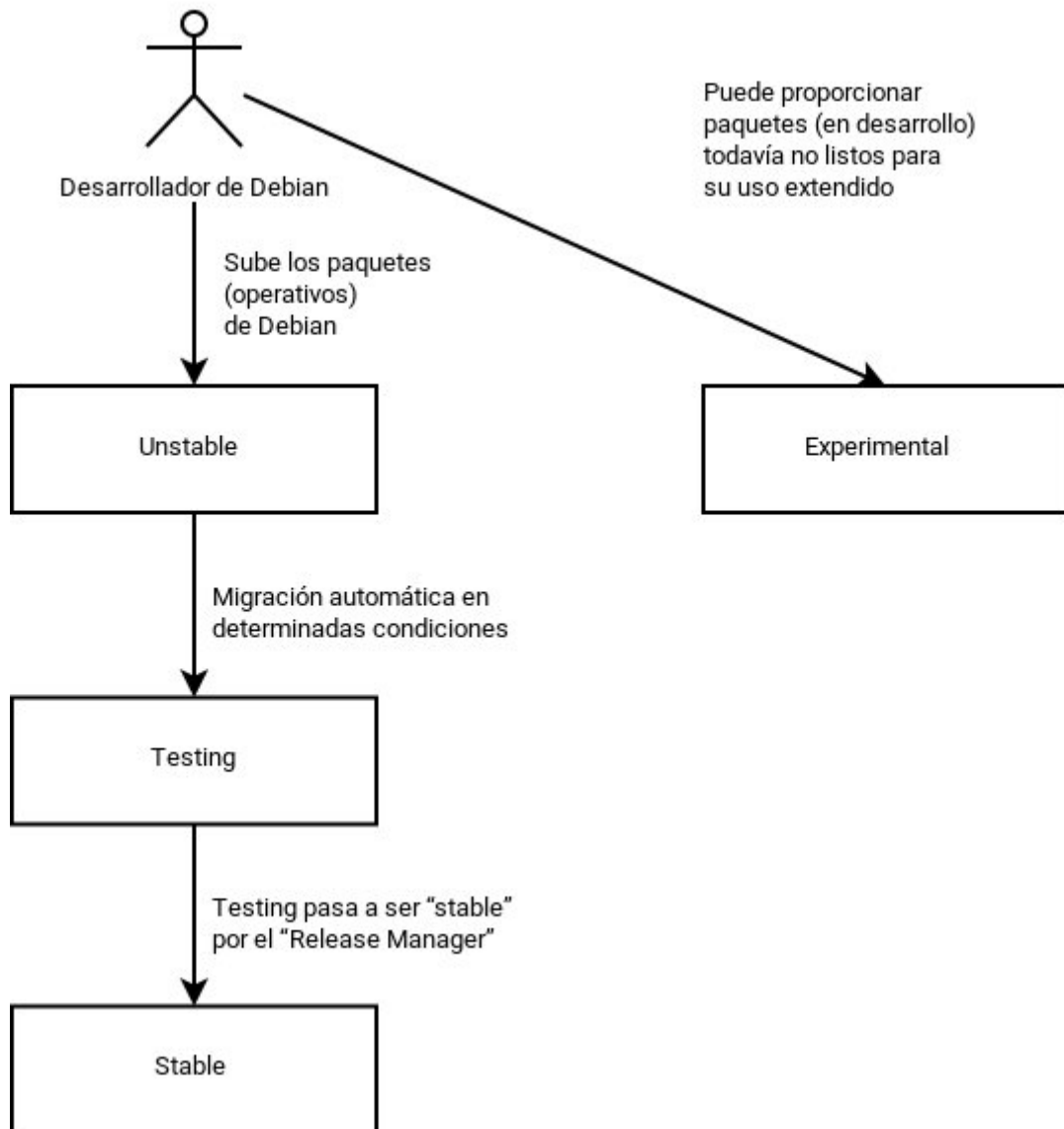


Figura 1.3. El camino de un paquete a través de las varias versiones de Debian

Al final del viaje, nuestro paquete hipotético ahora está incluido en la distribución estable. Este viaje, con sus dificultades, explica las demoras significativas que separan las versiones estables de Debian. Esto contribuye, en general, a su reputación de calidad. Lo que es más, la mayoría de los usuarios son satisfechos utilizando una de las tres distribuciones disponibles simultáneamente. Los administradores de sistemas no necesitan la última y mejor versión de GNOME preocupados por la estabilidad de sus servidores por sobre todas las cosas; ellos pueden elegir Debian Stable y estarán satisfechos. Los usuarios finales, más interesados en las últimas versiones de GNOME o KDE Plasma que en una estabilidad sólida, encontrarán en Debian Testing un buen compromiso entre la falta de problemas serios y software relativamente actualizado. Finalmente, desarrolladores y usuarios más experimentados pueden liderar el camino probando todos los últimos desarrollos en Debian Unstable recién salidos del horno, arriesgándose a sufrir dolores de cabeza y errores inherentes en cualquier nueva versión de un programa. ¡A cada quien su propio Debian!

1.1.6.5. El estado de Oldstable y Oldoldstable

Cada versión estable (Stable) tiene una esperanza de vida de unos 5 años y, dado que se tiende a liberar una nueva versión cada 2 años, pueden haber hasta 3 versiones soportadas en un mismo momento. Cuando se publica una nueva versión, la distribución predecesora pasa a Oldstable y la que lo era antes pasa a ser Oldoldstable.

Este soporte a largo plazo (LTS) de las versiones de Debian es una iniciativa reciente: colaboradores individuales y empresas han unido fuerzas para crear el equipo Debian LTS. Las versiones antiguas que ya no son soportadas por el equipo de seguridad de Debian pasan a ser responsabilidad de este nuevo equipo.

The Debian security team handles security support in the current Stable release and also in the Oldstable release (but only for as long as is needed to ensure one year of overlap with the current stable release). This amounts roughly to three years of support for each release. The Debian LTS team handles the last (two) years of security support so that each release benefits from at least 5 years of support and so that users can upgrade from version N to N+2, for example, from Debian 9 Stretch to Debian 11 Bullseye.

1.5. Sistema de paquetes: herramientas y principios fundamentales

Como un administrador de un sistema Debian generalmente manejará paquetes .deb ya que contienen unidades funcionales consistentes (aplicaciones, documentación, etc.) facilitando su instalación y mantenimiento. Por lo tanto, es buena idea saber qué son y cómo utilizarlos.

Este capítulo describe la estructura y los contenidos de paquetes «binarios» y «fuente». Los primeros son archivos utilizables directamente con dpkg, mientras que los últimos contienen el código fuente así como las instrucciones para crear los paquetes binarios.

1.5.1. Estructura de un paquete binario

El formato del paquete Debian fue diseñado para que su contenido pueda ser extraído en cualquier sistema Unix que tenga los programas clásicos ar, tar y xz, o a veces gzip o bzip2. Esta propiedad aparentemente trivial es importante para portabilidad y recuperación en caso de desastres.

Imagine por ejemplo que eliminó por error el programa dpkg y que, por lo tanto, ya no puede instalar paquetes Debian. Siendo dpkg un paquete en sí mismo pareciera como que su sistema estuviese condenado... afortunadamente conoce el formato de un paquete y puede descargar el archivo .deb para el paquete dpkg e instalarlo manualmente (revise el recuadro HERRAMIENTAS dpkg, APT y ar). Si por cualquier motivo o problema uno o más de los programas ar, tar o gzip/xz/bzip2 desaparecieron sólo necesitará copiar el programa faltante de otro sistema (ya que cada uno de ellos funciona de forma completamente autónoma una simple copia bastará). Si su sistema sufre algún evento de peor fortuna e incluso esto no funciona (¿quizás a su sistema le falten bibliotecas a más bajo nivel?), debería intentar la versión estática del programa busybox (incluido en el paquete busybox-static), el cual es inclusive más autocontenido y proporciona órdenes como busybox ar, busybox tar y busybox xz.

En caso de error más vale que también tengas una copia de seguridad de tu sistema (véase Sección 9.10, “Respaldo”).

HERRAMIENTAS dpkg, APT y ar

dpkg es el programa que maneja los archivos .deb (paquetes binarios), en particular los extrae, analiza y descomprime.

APT (las siglas de Advanced Packaging Tool) es un grupo de programas que permite la ejecución de modificaciones de más alto nivel al sistema: instalar o eliminar un paquete (mientras mantiene dependencias satisfechas), actualizar la lista de paquetes (apt update) y actualizar el sistema (apt upgrade), listar los paquetes disponibles, etc.

As for the ar program, provided by the binutils package, it allows handling files of the same name:

ar t archive displays the list of files contained in such an archive,

ar x archive extracts the files from the archive into the current working directory,

ar d archive file deletes a file from the archive, etc.

Its man page ar(1) documents all its other features. ar is a very rudimentary tool that a Unix administrator would only use on rare occasions, but admins routinely use tar, a more evolved archive and file management program. This is why it is easy to restore dpkg in the event of an erroneous deletion. You would only have to download the Debian package and extract the content from the data.tar.xz archive in the system's root (/):

```
# ar x dpkg_1.20.9_amd64.deb
```

```
# tar -C / -p -xJf data.tar.xz
```

VOLVER A LOS CIMIENTOS Notación de páginas de manual

Los principiantes pueden encontrar confusas las referencias como «ar(1)» en la literatura. Generalmente esta es una forma conveniente de referirse a la página de manual titulada ar en la sección 1.

Algunas veces se utiliza esta notación para eliminar ambigüedades, por ejemplo para distinguir entre el programa printf, que también puede indicarse como printf(1), y la función printf del lenguaje de programación C, que también puede indicarse como printf(3).

El Capítulo 7, Resolución de problemas y búsqueda de información relevante discute las páginas de manual con más detalles (revise la Sección 7.1.1, “Páginas de manual”).

Estos son los contenidos de un archivo .deb:

```
$ ar t dpkg_1.20.9_amd64.deb
```

```
debian-binary
```

```
control.tar.gz
```

```
data.tar.xz
```

```
$ ar x dpkg_1.20.9_amd64.deb
```

```
$ ls
```

```
control.tar.gz data.tar.xz debian-binary dpkg_1.20.9_amd64.deb
```

```
$ tar tJf data.tar.xz | head -n 16
```

```
./
```

```
./etc/
```

```
./etc/alternatives/
```

```
./etc/alternatives/README
```

```
./etc/cron.daily/
```

```
./etc/cron.daily/dpkg
```

```
./etc/dpkg/
```

```
./etc/dpkg/dpkg.cfg
```

```
./etc/dpkg/dpkg.cfg.d/
```



```
.etc/logrotate.d/  
.etc/logrotate.d/alternatives  
.etc/logrotate.d/dpkg  
./sbin/  
./sbin/start-stop-daemon  
./usr/  
./usr/bin/  
$ tar tJf control.tar.xz  
./  
./conffiles  
./control  
./md5sums  
./postrm  
$ cat debian-binary  
2.0
```

Como puede ver, el compendio **ar** de un paquete Debian contiene tres archivos:

debian-binary

This is a text file which simply indicates the version of the **.deb** file package format version. In Debian Bullseye it is still version 2.0.

control.tar.xz

Este compendio contiene toda la metainformación disponible, como el nombre y la versión del paquete, así como algunos «scripts» para ejecutar antes, durante o después de la desinstalación o instalación de este. Alguna de esta metainformación le permite a las herramientas de gestión de paquetes determinar si es posible instalar o desinstalarlo, por ejemplo, según la lista de paquetes que ya se encuentran en el equipo, y si archivos incluidos han sido modificados localmente.

data.tar.xz, data.tar.bz2, data.tar.gz

Este compendio contiene todos los archivos a extraerse del paquete; aquí es donde están almacenados los archivos ejecutables, las bibliotecas, la documentación etc. Los paquetes pueden utilizar formatos de compresión diferentes; en ese caso, el archivo tendrá otro nombre para **xz**, **bzip2** o **gzip**.

1.5.2.2.1. Instalación y actualización

During the initial installation and for each upgrade of a package, **dpkg** calls the so called maintainer scripts such as the **prepm** or **preinst** scripts. These scripts can perform additional actions during the different stages of a package's life-cycle. Script names preceded by **new-** are the scripts from the new version of a package being installed or upgraded to. Script names preceded by **old-** are the scripts from the old version of a package that is being upgraded from.

During each invocation **dpkg** will pass certain arguments to each script such as **upgradenew-version**. The invoked script can then either handle the arguments and perform a particular action, or ignore the arguments and return with an exit code of **0**, if nothing needs to be done during that step. In practice many packages will not need to perform an action during every step in the life

cycle. Thus a typical configuration script will check for a particular argument and ignore all other ones, implicitly returning with exit code **0**.

Here is what happens during an installation (or an update). The *old-version*, *new-version* and *last-version-configured* arguments are placeholders for the actual (old and new) version numbers of the package:

1. For an update, **dpkg** calls the **old-prerm** script and passes **upgradenew-version** as arguments.
2. Still for an update, **dpkg** then executes the **new-preinst** script with the arguments **upgradeold-version**; for the initial installation, it executes the **new-preinst** script and passes **install** as argument. It may add the old version in the last parameter, if the package has already been installed and removed since (but not purged, and thus configuration files have been retained).
3. Se descomprimen los archivos del nuevo paquete. Si un archivo ya existe, es reemplazado pero se guarda una copia de respaldo de forma temporal.
4. For an update, **dpkg** executes the **old-postrm** script and passes **upgradenew-version** as arguments.
5. **dpkg** actualiza toda su información interna (lista de archivos, scripts de configuración, etc.) y elimina los respaldos de los archivos reemplazados. Este es el punto sin retorno: **dpkg** ya no tiene acceso a todos los elementos necesarios para volver al estado anterior.
6. **dpkg** actualizará los archivos de configuración, pidiéndole al usuario que decida si no es capaz de administrar esta tarea automáticamente. Los detalles de este proceso son discutidos en la [Sección 5.2.3, “Checksums, List of Configuration Files, et al.”](#).
7. Finally, **dpkg** configures the package by executing the **new-postinst** script with the arguments **configurelast-version-configured**.

1.5.2.2.2. Eliminación de un paquete

The steps to remove a package are analogous to the installation steps. The main difference is that the removal scripts of the package are called:

1. **dpkg** calls the **prerm** script and passes the **remove** argument.
2. **dpkg** removes all of the package's files, with the exception of the configuration files and maintainer scripts.
3. **dpkg** executes the **postrm** script and passes **remove** as argument. Afterwards, all of the maintainer scripts, except the **postrm** script, are removed. If the user has not used the “purge” option, the process stops here.
4. For a complete purge of the package (command issued with **dpkg --purge** or **dpkg -P**), the configuration files are also deleted, as well as a certain number of copies (***.dpkg-tmp**, ***.dpkg-old**, ***.dpkg-new**) and temporary files; **dpkg** then executes the **postrm** script and passes **purge** as argument.

Los cuatro scripts que aparecen detallados anteriormente se complementan con un script **config** provisto por los paquetes que utilizan **debconf** para adquirir información de configuración del usuario. Durante la instalación este script define en detalle las preguntas realizadas por **debconf**. Se graban las respuestas en la base de datos de **debconf** para futuras referencias. Generalmente **apt** ejecuta el script antes de instalar los paquetes uno por uno para agrupar las preguntas y realizarlas todas al usuario al comienzo del proceso. Los scripts de pre y postinstalación pueden utilizar esta información para operar según los deseos del usuario.

1.5.4. Manipulación de paquetes con dpkg

dpkg es el programa base para manejar paquetes Debian en el sistema. Si tiene paquetes **.deb**, **dpkg** es lo que permite instalar o analizar sus contenidos. Pero este programa sólo tiene una visión parcial del universo Debian: sabe lo que está instalado en el sistema y lo que sea que se le provee en la línea de órdenes, pero no sabe nada más de otros paquetes disponibles. Como tal, fallará si no se satisface una dependencia. Por el contrario, herramientas como **apt** y **aptitude** crearán una lista de dependencias para instalar todo tan automáticamente como sea posible.

1.5.4.1. Instalación de paquetes

dpkg es, sobre todo, la herramienta para instalar un paquete Debian ya disponible (porque no descarga nada). Para hacer esto utilizamos su opción **-i** o **--install**.

A veces **dpkg** fallará intentando instalar un paquete y devolverá un error; si el usuario le ordena ignorarlo sólo generará una advertencia; es por esta razón que tenemos las diferentes opciones **-force***. La orden **dpkg --force-help**, o su documentación, proveerá una lista completa de estas opciones.

1.5.4.2. Eliminación de un paquete

Ejecutar **dpkg** con la opción **-r** o **--remove** seguida del nombre de un paquete eliminará dicho paquete. Esta eliminación, sin embargo, no es completa: se mantendrán todos los archivos de configuración, scripts del encargado, archivos de registros (registros de sistema) y otros datos de usuarios que gestiona el paquete. De esta forma, puede desactivar el programa fácilmente al desinstalarlo pero es posible reinstalarlo rápidamente con la misma configuración. Para eliminar completamente todo lo asociado con un paquete, utilice la opción **-P** o **--purge** seguida del nombre del paquete.

1.5.4.4. Archivo de registro de dpkg

dpkg mantiene un registro de todas sus acciones en **/var/log/dpkg.log**. Este registro es extremadamente detallado ya que incluye cada una de las etapas por las que pasa un paquete gestionado por **dpkg**. Además de ofrecer una forma de rastrear el funcionamiento de **dpkg**, sobre todo ayuda a mantener un historial del desarrollo del sistema: uno puede encontrar el momento exacto en el que se instaló o actualizó un paquete, y esta información puede ser extremadamente útil cuando se intenta entender un cambio de comportamiento reciente. Además, como se registran todas las versiones, es sencillo verificar y referenciar información con el archivo **changelog.Debian.gz** del paquete en cuestión o inclusive con reportes de error online.

1.6. Mantenimiento y actualizaciones: las herramientas APT

Lo que hace a Debian tan popular entre administradores es lo sencillo que resulta instalar software y lo fácil que se puede actualizar el sistema completo. Esta ventaja única es en gran parte debido al programa APT, que los administradores de Falcot Corp estudiaron con entusiasmo.

Se necesita proveerle a APT una «lista de paquetes fuente» (repositorios): el archivo **/etc/apt/sources.list** contendrá una lista de diferentes repositorios que publican paquetes Debian. APT importará la lista de paquetes publicada por cada una de estos repositorios. Realiza esta operación descargando los archivos **Packages.xz** o una variante como **Packages.gz** o **.bz2** (que usa un distinto método de compresión) en caso de una fuente de archivos binarios y analizando sus contenidos. En caso de una fuente de paquetes fuente, APT descarga archivos **Sources.xz** o una variante usando un método de compresión diferente. Cuando ya posee una copia antigua de estos archivos, APT puede actualizarla solo descargando las diferencias (revise el recuadro SUGERENCIA Actualizaciones incrementales).

1.6.1. Contenido del archivo `sources.list`

1.6.1.1. Sintaxis

Cada línea activa en el archivo `/etc/apt/sources.list` representa un paquete fuente (repositorio) y está compuesta de al menos tres partes separadas por espacios. Para una descripción completa del formato de archivo y la estructura de entradas aceptada consulte `sources.list(5)`.

El primer campo indica el tipo de origen:

deb

paquete fuente (repositorio) de paquetes binarios

deb-src

paquete fuente (repositorio) de paquetes fuente

The second field gives the base URL of the source. Combined with the filenames listed in the **Packages.xz** files, it must give a full and valid URL. This can consist in a Debian mirror or in any other package archive set up by a third party. The URL can start with **file://** to indicate a local source installed in the system's file hierarchy, with **http://** or **https://** to indicate a source accessible from a web server, or with **ftp://** or **ftps://** for a source available on an FTP server. The URL can also start with **cdrom:** for CD-ROM/DVD/Blu-ray disc based installations, although this is less frequent, since network-based installation methods are eventually more common. More methods like **ssh://** or **tor+http(s)://** are supported and are either described in `sources.list(5)` or their respective `apt-transport-method` package documentation.

The syntax of the last field depends on the structure of the repository. In the simplest case, you can simply indicate a subdirectory (with a required trailing slash) of the desired source. This is often a simple `./` which refers to the absence of a subdirectory. The packages are then directly at the specified URL. But in the most common case, the repositories will be structured like a Debian mirror, with multiple distributions, each having multiple components. In those cases, name the chosen distribution by its “codename” — see the list in sidebar [COMUNIDAD Bruce Perens, un líder polémico](#) — or by the corresponding “suite” (**oldoldstable**, **oldstable**, **stable**, **testing**, **unstable**) and then the components to enable. A typical Debian mirror provides the components **main**, **contrib**, and **non-free**.

Los elementos **cdrom** describen los CD/DVD-ROMs que posee. A diferencia de otros elementos, un CD-ROM no siempre está disponible ya que debe encontrarse en el dispositivo y sólo un disco puede leerse en un momento dado. Por estas razones, se gestionan estos elementos de una forma ligeramente diferente y necesitan ser agregados con el programa **apt-cdrom**, usualmente ejecutado con el parámetro **add**. Este programa solicitará que introduzca el disco en el dispositivo y navegará su contenido en busca de archivos **Packages**. Utilizará dichos archivos para actualizar su base de datos de paquetes disponibles (generalmente realizada cuando ejecuta **apt update**). Desde ese momento en adelante, APT puede solicitarle introducir el disco si necesita uno de sus paquetes.

1.6.1.2.1. Actualizaciones de seguridad

Debian se toma la seguridad en serio. Las vulnerabilidades de software conocidas en Debian se supervisan en el [Security Bug Tracker](#) (rastreador de errores de seguridad) y normalmente se solucionan en un tiempo razonable. Las actualizaciones de seguridad no se encuentran en la red de réplicas de Debian usual sino en [security.debian.org](#), un conjunto pequeño de equipos administrados por los [Administradores de sistemas de Debian](#). Este compendio contiene las actualizaciones de seguridad preparadas por el equipo de seguridad de Debian, «Debian Security Team», y/o por los encargados de los paquetes para la distribución Stable y Oldstable.

The server can also host security updates for Testing but this doesn't happen very often since those updates tend to reach that suite via the regular flow of updates coming from Unstable. Para incidencias serias, el equipo de seguridad publica un **Aviso de Seguridad de Debian** (en inglés **Debian Security Advisory**, DSA) y lo anuncia junto a la actualización de seguridad en la lista de correo debian-security-announce@lists.debian.org (archivo).

1.6.1.2.2. Actualizaciones de Stable

Las actualizaciones de stable no implican riesgos de seguridad pero son consideradas suficientemente importantes como para ser enviadas a los usuarios antes de la publicación de la siguiente versión menor de stable.

Este repositorio generalmente incluirá correcciones de errores críticos y serios que no pudieron ser actualizados antes de la publicación o que fueron introducidos en actualizaciones posteriores. Dependiendo de la urgencia, también puede contener actualizaciones de paquetes que evolucionaron con el tiempo, como las reglas de detección de spam de spamassassin, la base de datos de virus de clamav, las reglas de horarios de verano de todos los husos horarios (tzdata), la versión ESR de Firefox (firefox-esr) o conjuntos de llaves criptográficas como debian-archive-keyring.

En la práctica, este repositorio es un subgrupo del repositorio **proposed-updates**, cuidadosamente seleccionado por los Gestores de Versiones Estables. Todas las actualizaciones se anuncian en la lista de correo debian-stable-announce@lists.debian.org (archivo) y se incluirán de todas formas en la próxima publicación punto algo de Stable.

1.6.1.2.3. Actualizaciones propuestas

Una vez publicada, la distribución Stable se actualiza sólo una vez cada 2 meses. El repositorio **proposed-updates** es donde se preparan las futuras actualizaciones (bajo la supervisión de los Gestores de la versión estable, «Stable Release Managers»).

Las actualizaciones de seguridad y de estable documentadas en las secciones anteriores siempre son parte de este repositorio, pero también habrá otras ya que los encargados de los paquetes también tienen la oportunidad de corregir errores importantes que no justifican que se publique una nueva versión inmediatamente.

Anyone can use this repository to test those updates before their official publication. The extract below uses the **bullseye-proposed-updates** alias which is both more explicit and more consistent since **buster-proposed-updates** also exists (for the Oldstable updates):

```
deb https://deb.debian.org/debian bullseye-proposed-updates main contrib non-free
```

1.6.1.2.4. Retroadaptaciones para Stable

El repositorio **stable-backports** contiene «retroadaptaciones de paquetes». Es término hace referencia a paquetes de software reciente que fue recompilado para una distribución antigua, generalmente para Stable.

Cuando la distribución entra en años, muchos proyectos de software habrán publicado nuevas versiones que no están integradas en la versión actual Stable, que solo es modificada para corregir los problemas más críticos, como los problemas de seguridad. Debido a que las versiones Testing y Unstable son más riesgosas, los encargados de paquetes a veces ofrecen voluntariamente recompilaciones de aplicaciones de software recientes para Stable, lo que para usuarios y administradores de sistemas tiene la ventaja de limitar la potencial inestabilidad a un número pequeño de paquetes seleccionados. La página web <https://backports.debian.org> proporciona más información.

Solo se crean las retroadaptaciones de **stable-backports** de los paquetes disponibles en Testing. Esto asegura que todas las retroadaptaciones instaladas se actualizarán a la versión estable correspondiente cuando se encuentre disponible la siguiente versión estable de Debian.

Aun cuando este repositorio provea versiones de paquetes más nuevas, **APT** no las instalará a menos que le indique explícitamente que lo haga (o si ya lo hizo con una versión anterior de dicha retroadaptación):

```
$ sudo apt-get install package/bullseye-backports$ sudo apt-get install -t bullseye-backports package
```

1.6.2. Los programas **aptitude**, **apt-get** y **apt**

APT es un proyecto gigante y su plan original incluía una interfaz gráfica. Está basado en una biblioteca que contiene la aplicación central y **apt-get** fue la primera interfaz — basada en la línea de órdenes — desarrollada dentro del proyecto. **apt** es un segundo frontend de línea de comandos proporcionado por APT el cual soluciona algunos errores de diseño de la orden **apt-get**.

Both tools are built on top of the same library and are thus very close, but the default behavior of **apt** has been improved for interactive use and to actually do what most users expect. The APT developers reserve the right to change the public interface of this tool to further improve it. Conversely, the public interface of **apt-get** is well defined and will not change in any backwards incompatible way. It is thus the tool that you want to use when you need to script package installation requests.

Varias otras interfaces gráficas aparecieron luego como proyectos externos: **synaptic**, **aptitude** (que incluye tanto una interfaz en modo texto como una gráfica — aún cuando no esté completa), **wajig**, etc. La interfaz más recomendada, **apt** es la que utilizaremos en los ejemplos de esta sección. Note, sin embargo, que la sintaxis de línea de órdenes de **apt-get** y de **aptitude** son muy similares. Detallaremos cuando existan grandes diferencias entre estas tres órdenes.

1.6.2.1. Inicialización

For any work with APT, the list of available packages needs to be updated; this can be done simply through **apt update**. Depending on the speed of your connection and configuration, the operation can take a while, since it involves downloading a certain number of (usually compressed) files (**Packages**, **Sources**, **Translation-language-code**), which have gradually become bigger and bigger as Debian has developed (at least 10-16 MB of data for the **main** section). Of course, installing from a CD-ROM/DVD set does not require any downloading — in this case, the operation is very fast.

1.6.2.2. Instalación y eliminación

Con APT puede agregar o eliminar paquetes del sistema, con **apt install paquete** y **apt remove paquete** respectivamente. En ambos casos APT automáticamente instalará las dependencias necesarias o eliminará los paquetes que dependen del paquete que está siendo eliminado. La orden **ap purge paquete** realiza una desinstalación completa —eliminando también los archivos de configuración—.

Si el archivo **sources.list** menciona varias distribuciones, es posible indicar la versión del paquete a instalar. Se puede proporcionar un número de versión específico con **apt install paquete=versión**, pero generalmente es preferible indicar la distribución de origen (Stable, Testing o Unstable) utilizando **apt install paquete/distribución**. Con esta orden es posible volver a una versión antigua de un paquete (si sabe que funciona bien, por ejemplo), siempre que aún esté disponible en alguno de los orígenes a los que se refiere el archivo **sources.list**. De lo contrario, el archivo **snapshot.debian.org** puede llegar al rescate (revise el recuadro **GOING FURTHER** Old package versions: **snapshot.debian.org** and **archive.debian.org**).

Si el paquete que se ha de instalar se te ha proporcionado con la forma de un simple archivo **.deb** sin ningún repositorio de paquete asociado, es posible usar APT para instalarlo junto a sus dependencias (siempre que las dependencias estén disponibles en los repositorios configurados) con una simple orden: **apt install ./ruta-al-paquete.deb**. El **./** que precede es importante para dejar claro que nos referimos a un nombre de archivo y no a un nombre de paquete disponible en uno de los repositorios.

1.8. Configuración básica: red, cuentas, impresión...

1.8.4. Bases de datos de usuarios y grupos

Generalmente se almacena la lista de usuarios en el archivo **/etc/passwd** y el archivo **/etc/shadow** almacena las contraseñas con hash. Ambos son archivos de texto en un formato relativamente simple que pueden leerse y modificarse con un editor de texto. Se muestra cada usuario en una línea con varios campos separados por dos puntos («:»).

1.8.4.1. Lista de usuarios: **/etc/passwd**

Esta es una lista de los campos en el archivo **/etc/passwd**:

- nombre de usuario, por ejemplo **rhertzog**;
- **password**: this is a password encrypted by a one-way function (**crypt**), relying on **DES**, **MD5**, **SHA-256** or **SHA-512**. The special value “**x**” indicates that the encrypted password is stored in **/etc/shadow**;
- **uid**: número único que identifica a cada usuario;
- **gid**: número único del grupo principal del usuario (de forma predeterminada, Debian crea un grupo específico para cada usuario);
- **GECOS**: campo de datos que generalmente contiene el nombre completo del usuario;
- directorio de inicio de sesión, asignado al usuario para almacenar sus archivos personales (al que generalmente apunta la variable de entorno **\$HOME**);
- programa a ejecutar al iniciar sesión. Generalmente es un intérprete de órdenes (consola) que le da libertad al usuario. Si especifica **/bin/false** (que no hace nada y vuelve el control inmediatamente), el usuario no podrá iniciar sesión.

As mentioned before, one can edit this file directly. But there are more elegant ways to apply changes, which are described in [Sección 8.4.3, “Modificación de una cuenta o contraseña existente”](#).

1.8.4.2. El archivo de contraseñas ocultas y cifradas: **/etc/shadow**

El archivo **/etc/shadow** contiene los siguientes campos:

- nombre de usuario;
- contraseña cifrada;
- varios campos que administran el vencimiento de la contraseña.

One can expire passwords using this file or set the time until the account is disabled after the password has expired.

1.8.4.3. Modificación de una cuenta o contraseña existente

The following commands allow modification of the information stored in specific fields of the user databases: **passwd** permits a regular user to change their password, which in turn, updates the **/etc/shadow** file (**chpasswd** allows administrators to update passwords for a list of users in batch mode); **chfn** (CHange Full Name), reserved for the super-user (root), modifies

the **GECOS** field. **chsh** (CHange SHell) allows the user to change their login shell; however, available choices will be limited to those listed in **/etc/shells**; the administrator, on the other hand, is not bound by this restriction and can set the shell to any program of their choosing.

Finalmente **chage** (cambiar edad: «CHange AGE») permite al administrador cambiar la configuración de expiración de la contraseña (la opción **-l usuario** mostrará la configuración actual). También puede forzar la expiración de una contraseña utilizando la orden **passwd -e usuario**, que obligará al usuario a cambiar su contraseña la próxima vez que inicie sesión.

Besides these tools the **usermod** command allows to modify all the details mentioned above.

1.8.4.4. Desactivación de una cuenta

You may find yourself needing to “disable an account” (lock out a user), as a disciplinary measure, for the purposes of an investigation, or simply in the event of a prolonged or definitive absence of a user. A disabled account means the user cannot login or gain access to the machine. The account remains intact on the machine and no files or data are deleted; it is simply inaccessible. This is accomplished by using the command **passwd -l user** (lock). Re-enabling the account is done in similar fashion, with the **-u** option (unlock). This, however, only prevents password-based logins by the user. The user might still be able to access the system using an SSH key (if configured). To prevent even this possibility you have to expire the account as well using either **chage -E 1 user** or **usermod -e 1 user** (giving a value of **-1** in either of these commands will reset the expiration date to **never**). To (temporarily) disable all user accounts just create the file **/etc/nologin**.

You can disable a user account not only by locking it as described above, but also by changing its default login shell (**chsh -s shelluser**). With the latter changed to **/usr/sbin/nologin**, a user gets a polite message informing that a login is not possible, while **/bin/false** just exits while returning **false**. There is no switch to restore the previous shell. You have to get and keep that information before you change the setting. These shells are often used for system users which do not require any login availability.

1.8.4.5. Lista de grupos: /etc/group

Se enumeran los grupos en el archivo **/etc/group**, una simple base de datos de texto en un formato similar al del archivo **/etc/passwd** con los siguientes campos:

- nombre del grupo;
- contraseña (opcional): sólo es utilizada para unirse a un grupo cuando no es un miembro normal (con **newgrp** o **sg**, revise el recuadro [VOLVER A LOS CIMIENTOS Trabajar con varios grupos](#));
- **gid**: número único de identificación del grupo;
- lista de miembros: lista separados por comas de nombres de usuario que son miembros del grupo.

Los programas **addgroup** y **delgroup** agregan o eliminan un grupo respectivamente. **groupmod** modifica la información de un grupo (su identificador o **gid**). La orden **gpasswd grupo** cambia la contraseña del grupo mientras que **gpasswd -r grupo** elimina dicha contraseña.

1.8.9. Otras configuraciones: sincronización de tiempo, registros, acceso compartido...

Es recomendable que cualquiera que quiera dominar todos los aspectos de configuración de un sistema GNU/Linux conozca los muchos elementos incluidos en esta sección. Se los trata, sin embargo, brevemente y generalmente lo dirigirán a la documentación.

1.8.9.1. Zona horaria

The timezone, configured during initial installation, is a configuration item for the tzdata package. To modify it, use the **dpkg-reconfigure tzdata** command, which allows you to choose the timezone to be used in an interactive manner. Its configuration is stored in the **/etc/timezone** file. Additionally, **/etc/localtime** becomes a symbolic link to the corresponding file in the **/usr/share/zoneinfo**; the file that contains the rules governing the dates where daylight saving time (DST) is active, for countries that use it.

Cuando necesite cambiar la zona horaria temporalmente utilice la variable de entorno **TZ** que tiene más prioridad que la configurada en el sistema:

1.8.9.2. Sincronización de tiempo

La sincronización de tiempo, que puede parecer superfluo en un equipo, es muy importante en una red. Debido a que los usuarios no tienen permisos para modificar la fecha y hora es importante que esta información sea precisa para evitar confusión. Lo que es más, tener sincronizados todos los equipos de una red permite cruzar referencias de información en registros de diferentes máquinas. Por lo tanto, en caso de un ataque, es más sencillo reconstruir la secuencia cronológica de acciones en todos los equipos involucrados en el mismo. Los datos recolectados en varios equipos por motivos estadísticos no tendrán demasiado sentido si no están sincronizados.

1.8.9.2.2. Para servidores

Los servidores rara vez son reiniciados y es muy importante que la hora de estos sistemas sea correcta. Para mantener la hora correcta debe instalar un servidor NTP local, un servicio ofrecido en el paquete **ntp**. En su configuración predeterminada el servidor se sincronizará con **pool.ntp.org** y proveerá la hora como respuesta a pedidos que provengan de la red local. Puede configurarlo editando el archivo **/etc/ntp.conf**, siendo la alteración más importante el servidor NTP al que se refiere. Si la red tiene muchos servidores podría ser interesante tener un servidor de tiempo local que sincroniza con los servidores públicos y es utilizado como fuente de tiempo por los demás servidores de la red.

1.8.9.3. Rotación de archivos de registro

Los archivos de registro pueden crecer, rápido, y es necesario archivarlos. El esquema más común es un archivado rotativo: el archivo de registro es almacenado regularmente y sólo se mantienen los últimos *X* archivos. **logrotate**, el programa responsable de estas rotaciones, responde a las directivas presentes en el archivo **/etc/logrotate** y todos los archivos en el directorio **/etc/logrotate.d/**. El administrador puede modificar estos archivos si desean adaptar la política de rotación de registros definida por Debian. La página de manual **logrotate(1)** describe todas las opciones disponibles en estos archivos de configuración. Podría desear aumentar la cantidad de archivos mantenidos en la rotación o mover los archivos de registros a un directorio específico dedicado a su archivado en lugar de eliminarlos. También puede enviarlo por email para archivarlos en otro lado.

El programa **logrotate** es ejecutado diariamente por la aplicación **cron** (descrita en la [Sección 9.7, "Programación de tareas con cron y atd"](#)).

1.8.9.4. Compartición de permisos de administración

Frecuentemente, muchos administradores trabajan en la misma red. Compartir contraseñas de root no es muy elegante y abre la puerta al abuso debido al anonimato generado. La solución a este problema es el programa **sudo** que permite a ciertos usuarios ejecutar ciertas órdenes con permisos especiales. En el caso de uso más común, **sudo** permite a un usuario confiable ejecutar cualquier orden como root. Para hacerlo, el usuario simplemente ejecuta **sudo programa** y provee su contraseña personal como autenticación.

When installed, the sudo package gives full root rights to members of the **sudo** Unix group. To delegate other rights, the administrator can use the **visudo** command, which allows them to modify the **/etc/sudoers** configuration file (here again, this will invoke the **vi** editor, or any other editor indicated in the **EDITOR** environment variable). Alternatively they might put rules in small files in **/etc/sudoers.d/** as long as this directory is included by **/etc/sudoers** via **@includedir /etc/sudoers.d**, which is the default for Debian. Adding a line with **usernameALL=(ALL) ALL** allows the user in question to execute any command as root.

More sophisticated configurations allow authorization of only specific commands to specific users. All the details of the various possibilities are given in the sudoers(5) manual page.

1.9. Servicios Unix

Este capítulo cubre un número básico de servicios que son comunes a varios sistemas Unix. Todos los administradores deberían estar familiarizados con ellos.

1.9.1.2. El sistema de inicio System V

The System V init system (which we'll call **init** for brevity) executes several processes, following instructions from the **/etc/inittab** file. The first program that is executed (which corresponds to the **sysinit** step) is **/etc/init.d/rcS**, a script that executes all of the programs in the **/etc/rcS.d/** directory.

Entre estos encontrará sucesivamente programas a cargo de:

- configurar el teclado de la consola;
- cargar controladores: el núcleo carga por sí mismo la mayoría de los módulos a medida que el hardware es detectado; los controladores extras se cargan automáticamente cuando los módulos correspondientes son listados en **/etc/modules**;
- verificar la integridad de los sistemas de archivos;
- montar particiones locales;
- configurar la red;
- montar sistemas de archivos de red (NFS).

Después de esta etapa, **init** toma el control e inicia los programas activados en el nivel de ejecución («runlevel») predeterminado (generalmente el nivel 2). Ejecuta **/etc/init.d/rc 2**, un script que inicia todos los servicios enumerados en **/etc/rc2.d/** y aquellos cuyos nombres comiencen con la letra «S». Los números de dos cifras que le sigue fueron utilizados históricamente para definir el orden en el que se iniciarán los servicios, pero actualmente el sistema de inicio predeterminado utiliza **insserv**, que programa todo automáticamente basándose en las dependencias de los scripts. Cada script de inicio, por lo tanto, declara las condiciones a cumplir para iniciar o detener el servicio (por ejemplo, si debe iniciar antes o después de otro servicio); **init** luego los ejecuta en un orden que satisfaga estas condiciones. El enumerado estático de los scripts ya no se tiene en cuenta (pero sus nombres siempre deben comenzar con «S» seguidos de dos números y el nombre real del script utilizado para dependencias). Generalmente, se inician primero los servicios de base (como los registros con **rsyslogd** o la asociación de puertos con **portmap**) seguidos de los servicios estándar y la interfaz gráfica (**gdm**).

Este sistema de inicio basado en dependencias hace posible reenumerar automáticamente los scripts, lo que sería tedioso de hacer manualmente y limita el riesgo de error humano ya que se realiza la programación según los parámetros indicados. Otro beneficio es que se pueden iniciar los servicios en paralelo cuando son independientes entre ellos, lo cual puede acelerar el proceso de inicio.

init distingue varios niveles de ejecución («runlevel») y puede cambiar de uno a otro ejecutando **telinitnuevo-nivel**. Inmediatamente, **init** ejecuta nuevamente **/etc/init.d/rc** con el

nuevo nivel de ejecución. Luego, este script ejecutará los servicios faltantes y detendrá aquellos que ya no se desean. Para hacerlo, se refiere al contenido del archivo `/etc/rcX.d` (donde `X` representa el nuevo nivel de ejecución). Los scripts cuyos nombres comienzan con «S» (por «start», iniciar) son los servicios a iniciar; aquellos cuyos nombres comienzan con «K» (por «kill», matar) son los servicios a detener. El script no inicia ningún servicio que ya haya estado activo en el nivel de ejecución anterior.

De forma predeterminada, el inicio System V en Debian utiliza cuatro niveles de ejecución diferentes:

- Nivel 0: sólo se lo utiliza temporalmente mientras se apaga el equipo. Como tal, sólo contiene scripts «K».
- Nivel 1: también conocido como modo de usuario único, corresponde al sistema en modo degradado; sólo incluye servicios básicos y está destinado a operaciones de mantenimiento donde no se desea la interacción con usuarios normales.
- Nivel 2: es el nivel para operaciones normales, lo que incluye servicios de red, una interfaz gráfica, sesiones de usuario, etc.
- Nivel 6: similar a nivel 0, excepto a que es utilizada durante la fase de cierre que precede a un reinicio.

Existen otros niveles, especialmente del 3 al 5. De forma predeterminada están configurados para operar de la misma forma que el nivel 2, pero el administrador puede modificarlos (agregando o eliminando scripts en los directorios `/etc/rcX.d` correspondientes) para adaptarlos a necesidades particulares.

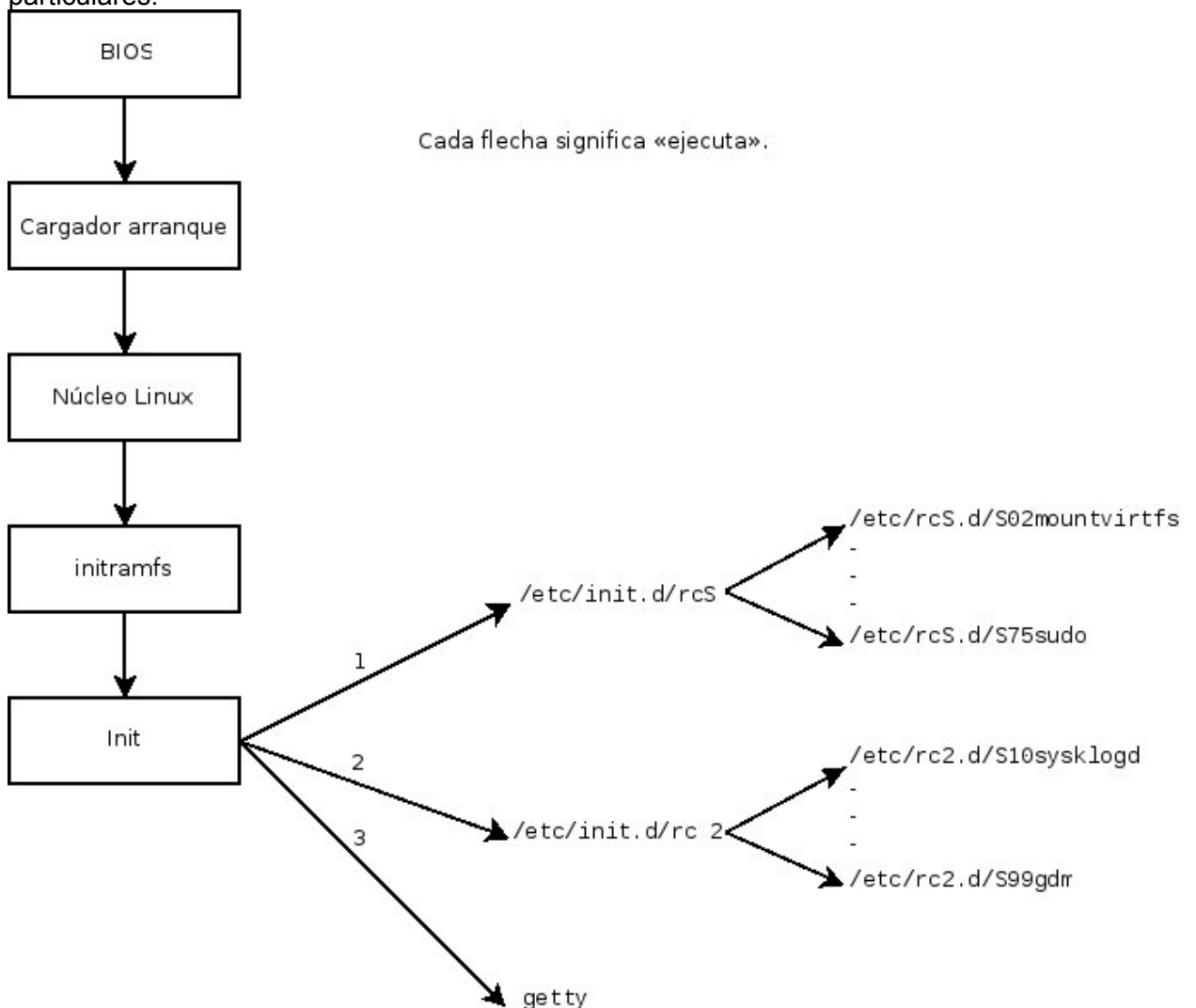


Figura 9.2. Secuencia de inicio de un equipo ejecutando Linux con inicio System V

All the scripts contained in the various `/etc/rcX.d` directories are really only symbolic links — created upon package installation by the `update-rc.d` program — pointing to the actual scripts which are stored in `/etc/init.d/`. The administrator can fine tune the services available in each runlevel by re-running `update-rc.d` with adjusted parameters. The `update-rc.d(1)` manual page describes the syntax in detail. Please note that removing all symbolic links (with the `remove` parameter) is not a good method to disable a service. Instead you should simply configure it to not start in the desired runlevel (while preserving the corresponding calls to stop it in the event that the service runs in the previous runlevel). Since `update-rc.d` has a somewhat convoluted interface, you may prefer using `rcconf` (from the `rcconf` package) which provides a more user-friendly interface.

Finalmente, `init` inicia los programas de control para varias consolas virtuales (`getty`). Muestra un prompt esperando por un nombre de usuario y luego ejecuta `loginusuario` para iniciar una sesión.

1.9.2. Inicio de sesión remoto

Es esencial para el administrador poder conectarse a un equipo de forma remota. Los servidores, aislados en su propia habitación, rara vez están equipados con monitores y teclados permanentes — pero están conectados a la red.

9.2.1. Inicio seguro de sesión remota: SSH

El protocolo SSH (interprete de órdenes seguro: «Secure SHell») fue diseñado pensando en la seguridad y la confiabilidad. Las conexiones que utilizan SSH son seguras: la otra parte es autenticada y se cifran todos los datos intercambiados.

SSH también ofrece dos servicios de transferencia de archivos. `scp` es una herramienta para la terminal que puede utilizar como `cp` excepto que cualquier ruta a otro equipo utilizará un prefijo con el nombre de la máquina seguido de dos puntos («:»).

```
$ scp archivo equipo:/tmp/
```

`sftp` es un programa interactivo similar a `ftp`. En una sola sesión `sftp` puede transferir varios archivos y es posible manipular archivos remotos con él (eliminar, renombrar, cambiar permisos, etc.).

Debian utiliza OpenSSH, una versión libre de SSH mantenida por el proyecto **OpenBSD** (un sistema operativo libre basado en el núcleo BSD enfocado en seguridad) que es una bifurcación («fork») del software SSH original desarrollado por la empresa SSH Communications Security Corp de Finlandia. Esta empresa inicialmente desarrolló SSH como software libre pero eventualmente decidió continuar su desarrollo bajo una licencia privativa. El proyecto OpenBSD luego creó OpenSSH para mantener una versión libre de SSH.

OpenSSH is split into two packages: the client part is in the `openssh-client` package, and the server is in the `openssh-server` package. The `ssh` meta-package depends on both parts and facilitates installation of both (`apt install ssh`), while the `task-ssh-server`, often chosen during the initial installation, depends on the server package only.

1.9.3. Administración de permisos

Linux es definitivamente un sistema multiusuario por lo que necesita proveer un sistema de permisos para controlar el conjunto de operaciones autorizadas sobre archivos y directorios, lo que incluye todos los recursos del sistema y los dispositivos (en un sistema Unix cualquier

dispositivo es representado por un archivo o un directorio). Este principio es común a todos los sistemas Unix pero siempre es útil recordarlo, especialmente porque existen algunos usos avanzados interesantes y relativamente desconocidos.

1.9.3.1. Owners and Permissions

Cada archivo o directorio tiene permisos específicos para tres categorías de usuarios:

- su dueño (representado con **u** por «usuario»);
- su grupo dueño (representado con **g** por «grupo»), que incluye a todos los miembros del grupo;
- y los demás (representado con **o** por «otros»).

Three basic types of rights can be combined:

- lectura (representado por **r** por «read»: leer);
- escritura (o modificación, representado con **w** por «write»: escribir);
- ejecución (representado con **x** por «eXecute»: ejecutar).

En el caso de un archivo, estos permisos se entienden fácilmente: la lectura permite acceder al contenido (inclusive copiarlo), la escritura permite cambiarlo y la ejecución permite ejecutarlo (lo cual sólo funcionará si es un programa).

Los directorios se manejan diferente. El permiso de lectura provee acceso para consultar su lista de elementos (archivos y directorios), el permiso de escritura permite crear o borrar archivos y el permiso de ejecución permite atravesarlo (especialmente para llegar a él con **cd**). Poder atravesar un directorio sin leerlo permite acceder a los elementos que contenga siempre que se conozca su nombre, pero no le permitirá encontrarlos si no sabe que existen o conoce sus nombres exactos.

Tres programas controlan los permisos asociados a un archivo:

- **chown *usuario archivo*** cambia el dueño de un archivo;
- **chgrp *group archivo*** modifica el grupo dueño;
- **chmod *permisos archivo*** cambia los permisos del archivo.

Hay dos formas de representar permisos. Entre ellas, la representación simbólica es probablemente la más sencilla de entender y recordar. Involucra las letras mencionadas anteriormente. Puede definir permisos para cada categoría de usuarios (**u/g/o**) definiéndolos explícitamente (con **=**, agregar permisos (**+**) o eliminar (**-**) permisos. Por lo tanto, la fórmula **u=rwx,g+rw,o-r** provee al dueño permisos de lectura, escritura y ejecución, agrega permisos de lectura y escritura al grupo dueño y elimina el permiso de lectura para los otros usuarios. Los permisos que no son modificados cuando se agreguen o eliminen permisos en estas fórmulas se mantienen intactos. La letra **a** (por «all», todos) incluye las tres categorías de usuarios, por lo que **a=rx** otorga los mismos permisos (lectura y ejecución, pero no escritura) a las tres categorías de usuario.

La representación numérica (octal) asocia cada permiso con un valor: 4 para lectura, 2 para escritura y 1 para ejecución. Asociamos cada combinación de permisos con la suma de dichos valores. Se asigna cada valor a las diferentes categorías de usuarios uniéndolos en el orden usual (dueño, grupo, otros).

For instance, the **chmod 754 file** command will set the following rights: read, write and execute for the owner (since $7 = 4 + 2 + 1$); read and execute for the group (since $5 = 4 + 1$); read-only for others. The **0** (zero) means no rights; thus **chmod 600 file** allows for read/write rights for the owner, and no rights for anyone else. The most frequent right combinations are **755** for executable files and directories, and **644** for data files.

Para representar permisos especiales, puede agregar un cuarto dígito antes que los demás según el mismo principio, donde los bits **setuid**, **setgid** y «**sticky**» son, respectivamente, 4, 2 y 1. **chmod 4754** asociará el bit **setuid** con los permisos descritos anteriormente.

El uso de notación octal sólo permite definir todos los permisos en un archivo de forma simultánea; no puede utilizarse para agregar un nuevo permiso a un conjunto anterior, como p.ej. agregar el permiso de lectura al grupo dueño, ya que deben tenerse en cuenta los permisos existentes y hay que calcular el nuevo valor numérico correspondiente.

1.9.5. syslog Eventos de sistema

1.9.5.1. Principio y mecanismo

El demonio **rsyslogd** es responsable de recolectar los mensajes de servicio que provienen de aplicaciones y el núcleo para luego distribuirlos en archivos de registros (usualmente almacenados en el directorio **/var/log/**). Obedece a su archivo de configuración: **/etc/rsyslog.conf**.

Cada mensaje de registro es asociado con un subsistema de aplicaciones (llamados «facility» en la documentación):

- **auth** y **authpriv**: para autenticación;
- **cron**: proviene servicios de programación de tareas, **cron** y **atd**;
- **daemon**: afecta un demonio sin clasificación especial (DNS, NTP, etc.);
- **ftp**: el servidor FTP;
- **kern**: mensaje que proviene del núcleo;
- **lpr**: proviene del subsistema de impresión;
- **mail**: proviene del subsistema de correo electrónico;
- **news**: mensaje del subsistema Usenet (especialmente de un servidor NNTP — protocolo de transferencia de noticias en red, «Network News Transfer Protocol» — que administra grupos de noticias);
- **syslog**: mensajes del servidor **syslogd** en sí;
- **user**: mensajes de usuario (genéricos);
- **uucp**: mensajes del servidor UUCP (programa de copia Unix a Unix, «Unix to Unix Copy Program», un protocolo antiguo utilizado notablemente para distribuir correo electrónico);
- **local0** a **local7**: reservados para uso local.

Cada mensaje tiene asociado también un nivel de prioridad. Aquí está la lista en orden decreciente:

- **emerg**: «¡Ayuda!» Hay una emergencia y el sistema probablemente está inutilizado.
- **alerta**: apúrese, cualquier demora puede ser peligrosa, debe reaccionar inmediatamente;
- **crit**: las condiciones son críticas;
- **err**: error;
- **warn**: advertencia (error potencial);
- **notice**: las condiciones son normales pero el mensaje es importante;
- **info**: mensaje informativo;
- **debug**: mensaje de depuración.

1.14. Seguridad

Un sistema de información puede tener un nivel variable de importancia dependiendo del entorno. En algunos casos es vital para la supervivencia de una empresa. Por lo tanto, debe ser protegido de los diversos tipos de riesgos. El proceso de evaluación de estos riesgos y la definición e implementación de la protección se conocen en su conjunto como «proceso de seguridad».

1.14.1. Definición de una política de seguridad

La palabra «seguridad» en sí misma cubre un amplio rango de conceptos, herramientas y procedimientos, ninguno de los cuales es universal. Seleccionar entre ellos requiere una idea precisa de sus metas. Asegurar un sistema comienza con responder unas pocas preguntas. Al precipitarse a implementar un conjunto arbitrario de herramientas corre el riesgo de enfocarse en los aspectos de seguridad equivocados.

Lo primero a determinar, por lo tanto, es el objetivo. Un buen método para ayudar con esta determinación comienza con las siguientes preguntas:

- ¿Qué estamos tratando de proteger? La política de seguridad será diferente dependiendo de si queremos proteger los equipos o los datos. En este último caso, también es necesario saber qué datos.
- ¿Contra qué estamos tratando de protegernos? ¿Fuga de datos confidenciales? ¿Pérdida accidental de datos? ¿Pérdida de ingresos por interrupción del servicio?
- También ¿contra quién estamos tratando de protegernos? Las medidas de seguridad serán diferentes para protegerse contra el error de un usuario regular del sistema de lo que serían contra un grupo de atacantes determinado.

Habitualmente, se utiliza el término «riesgo» para referirse al conjunto de estos tres factores: qué proteger, qué necesitamos prevenir antes que suceda y quién intentará hacer que suceda. Modelar el riesgo requiere respuestas a estas tres preguntas. A partir de este modelo de riesgo, podemos construir una normativa de seguridad e implementarla con acciones concretas.

Vale la pena el tomar en cuenta restricciones adicionales, dado que pueden limitar el alcance de las políticas disponibles. ¿Hasta dónde estamos dispuestos a llegar para asegurar un sistema? Esta pregunta tiene un gran impacto en la política a implementar. La respuesta es a menudo definida en términos de costos monetarios, pero debe considerar otros elementos, tal como la cantidad de inconvenientes impuestos a los usuarios del sistema o una degradación en el rendimiento.

Una vez que modelamos el riesgo, podemos comenzar a pensar en diseñar una política de seguridad real.

En la mayoría de los casos, el sistema de información puede ser segmentado en subconjuntos coherentes y en su mayoría independientes. Cada subsistema tendrá sus propios requisitos y limitaciones, por lo que se deberá llevar a cabo la evaluación de riesgos y el diseño de la política de seguridad por separado para cada uno. Un buen principio a tener en cuenta es que un perímetro corto y bien definido es más fácil de defender que una frontera larga y sinuosa. Se debe diseñar en consecuencia también la organización de la red: se deben concentrar los servicios sensibles en un pequeño número de máquinas y estas máquinas sólo deben ser accesibles a través de un número mínimo de puntos de control, asegurar estos puntos de control será más fácil que asegurar todas las máquinas sensibles contra la totalidad del mundo exterior. Es en este punto que se hace evidente la utilidad del filtrado de red (incluyendo los firewalls). Puede implementar este filtrado con hardware dedicado, pero posiblemente una solución más simple y flexible sea utilizar un firewall en software como el que se integra en el núcleo Linux.

1.14.2. Firewall o el filtrado de paquetes

Un firewall es una puerta de enlace de la red con filtro y sólo es eficaz en aquellos paquetes que deben pasar a través de ella. Por lo tanto, sólo puede ser eficaz cuando la única ruta para estos paquetes es a través del firewall.

El núcleo Linux incorpora el cortafuegos netfilter, que se puede controlar desde el espacio de usuario con los programas **iptables**, **ip6tables**, **arptables** y **ebtables** commands.

However, Netfilter iptables commands are being replaced by nftables, which avoids many of its problems. Its design involves less code duplication, and it can be managed with just the **nft** command. Since Debian Buster, the nftables framework is used by default. The commands mentioned before are provided by versions, which use the nftables kernel API, by default. If one requires the "classic" commands, the relevant binaries can be adjusted using **update-alternatives**.

1.14.2.1. Comportamiento de nftables

As the kernel is processing a network packet, it pauses and allows us to inspect the packet and decide what to do with that packet. For example, we might want to drop or discard certain incoming packets, modify other packets in various ways, block certain outgoing packets to control against malware or redirect some packets at the earliest possible stage to bridge network interfaces or to spread the load of incoming packets between systems.

Es esencial una buena comprensión de las capas 3, 4 y 5 del modelo OSI (Open Systems Interconnection) para sacarle el máximo partido a netfilter.

Este cortafuegos está configurado con tablas (tables), que contienen reglas (rules) que forman parte de cadenas (chains). A diferencia de iptables, nftables no tiene ninguna tabla predeterminada. El usuario decide cuáles y cuántas tablas crear. Cada tabla debe tener solo una de las siguientes cinco familias asignadas: **ip**, **ip6**, **inet**, **arp** y **bridge**. Se usa **ip** si no se especifica la familia.

There are two types of chains: base chains and regular chains. A base chain is an entry point for packets from the networking stack. Base chains are registered into the Netfilter hooks, e.g. they see packets flowing through the TCP/IP stack. On the other hand, a regular chain is not attached to any hook so it does not see any traffic. But it may be used as a jump target for better organization of the rules.

Las reglas están formadas por declaraciones («statements»), lo que incluye algunas expresiones que deben coincidir y luego una declaración de veredicto como **accept**, **drop**, **queue**, **continue**, **return**, **jump chain** y **goto chain**.

1.14.2.3. Sintaxis de nft

Las órdenes de **nft** permiten manipular tablas, cadenas y reglas. La opción **table** admite varias operaciones: **add**, **create**, **delete**, **list** y **flush**. **nft add table ip6 mangle** añade una nueva tabla de la familia **ip6**.

Para insertar una nueva cadena base en la tabla **filter**, puede ejecutar la siguiente orden (tenga en cuenta que se debe escapar el punto y coma con una barra invertida cuando se usa Bash):

```
# nft add chain filter input { type filter hook input priority 0 \; }
```

Normalmente las reglas se añaden con la siguiente sintaxis: **nft add rule [familia] tabla cadena handle handle statement**.

insert es similar a la orden **add**, pero la regla proporcionada se antepone al principio de la cadena o antes de la regla con el handle proporcionado en vez de al principio o al final de la regla. Por ejemplo, la siguiente orden inserta una regla ante de la regla con el handle número 8:

```
# nft insert rule filter output position 8 ip daddr 127.0.0.8 drop
```


Las órdenes **nft** ejecutadas no realizan cambios permanentes en la configuración, así que se pierden si no se guardan. Las reglas del cortafuegos se encuentran en **/etc/nftables.conf**. Una forma simple de guardar la configuración actual del cortafuegos es ejecutando **nft list ruleset > /etc/nftables.conf** como superusuario.

nft permite muchas más operaciones, consulte su página de manual `nft(8)` para más información.

Parte 2: Plataforma de Virtualizacion

The Proxmox Administrator Guide

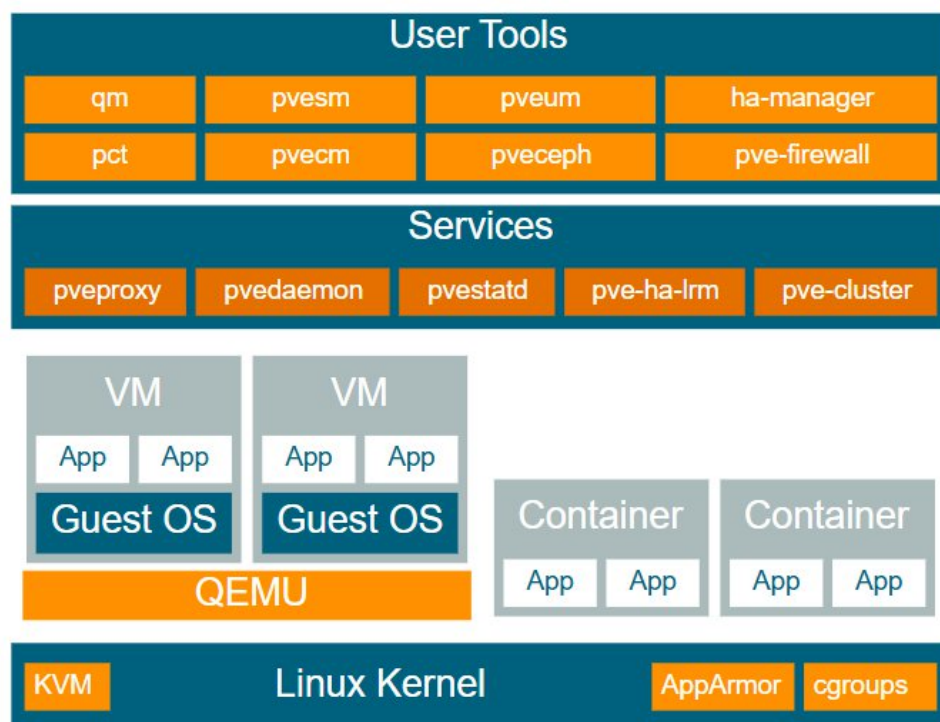
Documentacion Oficial : <https://pve.proxmox.com/pve-docs/pve-admin-guide.html>

2. PROXMOX VE ADMINISTRATION GUIDE

2.1 Introduction

Proxmox VE is a platform to run virtual machines and containers. It is based on Debian Linux, and completely open source. For maximum flexibility, we implemented two virtualization technologies - Kernel-based Virtual Machine (KVM) and container-based virtualization (LXC).

One main design goal was to make administration as easy as possible. You can use Proxmox VE on a single node, or assemble a cluster of many nodes. All management tasks can be done using our web-based management interface, and even a novice user can setup and install Proxmox VE within minutes.



2.1.1. Central Management

While many people start with a single node, Proxmox VE can scale out to a large set of clustered nodes. The cluster stack is fully integrated and ships with the default installation.

Unique Multi-Master Design

The integrated web-based management interface gives you a clean overview of all your KVM guests and Linux containers and even of your whole cluster. You can

easily manage your VMs and containers, storage or cluster from the GUI. There is no need to install a separate, complex, and pricey management server.

Proxmox Cluster File System (pmxcfs)

Proxmox VE uses the unique Proxmox Cluster file system (pmxcfs), a database-driven file system for storing configuration files. This enables you to store the configuration of thousands of virtual machines. By using corosync, these files are replicated in real time on all cluster nodes. The file system stores all data inside a persistent database on disk, nonetheless, a copy of the data resides in RAM which provides a maximum storage size of 30MB - more than enough for thousands of VMs.

Proxmox VE is the only virtualization platform using this unique cluster file system.

Web-based Management Interface

Proxmox VE is simple to use. Management tasks can be done via the included web based management interface - there is no need to install a separate management tool or any additional management node with huge databases. The multi-master tool allows you to manage your whole cluster from any node of your cluster. The central web-based management - based on the JavaScript Framework (ExtJS) - empowers you to control all functionalities from the GUI and overview history and syslogs of each single node. This includes running backup or restore jobs, live-migration or HA triggered activities.

Command Line

For advanced users who are used to the comfort of the Unix shell or Windows Powershell, Proxmox VE provides a command-line interface to manage all the components of your virtual environment. This command-line interface has intelligent tab completion and full documentation in the form of UNIX man pages.

REST API

Proxmox VE uses a RESTful API. We choose JSON as primary data format, and the whole API is formally defined using JSON Schema. This enables fast and easy integration for third party management tools like custom hosting environments.

Role-based Administration

You can define granular access for all objects (like VMs, storages, nodes, etc.) by using the role based user- and permission management. This allows you to define privileges and helps you to control access to objects. This concept is also known as access control lists: Each permission specifies a subject (a user or group) and a role (set of privileges) on a specific path.

Authentication Realms

Proxmox VE supports multiple authentication sources like Microsoft Active Directory, LDAP, Linux PAM standard authentication or the built-in Proxmox VE authentication server.

2.1.2. Flexible Storage

The Proxmox VE storage model is very flexible. Virtual machine images can either be stored on one or several local storages or on shared storage like NFS and on SAN. There are no limits, you may configure as many storage definitions as you like. You can use all storage technologies available for Debian Linux.

One major benefit of storing VMs on shared storage is the ability to live-migrate running machines without any downtime, as all nodes in the cluster have direct access to VM disk images.

We currently support the following Network storage types:

- LVM Group (network backing with iSCSI targets)
- iSCSI target
- NFS Share
- CIFS Share
- Ceph RBD
- Directly use iSCSI LUNs
- GlusterFS

Local storage types supported are:

- LVM Group (local backing devices like block devices, FC devices, DRBD, etc.)
- Directory (storage on existing filesystem)
- ZFS

2.1.3. Integrated Backup and Restore

The integrated backup tool (vzdump) creates consistent snapshots of running Containers and KVM guests. It basically creates an archive of the VM or CT data which includes the VM/CT configuration files.

KVM live backup works for all storage types including VM images on NFS, CIFS, iSCSI LUN, Ceph RBD. The new backup format is optimized for storing VM backups fast and effective (sparse files, out of order data, minimized I/O).

2.1.4. High Availability Cluster

A multi-node Proxmox VE HA Cluster enables the definition of highly available virtual servers. The Proxmox VE HA Cluster is based on proven Linux HA technologies, providing stable and reliable HA services.

2.1.5. Flexible Networking

Proxmox VE uses a bridged networking model. All VMs can share one bridge as if virtual network cables from each guest were all plugged into the same switch. For connecting VMs to the outside world, bridges are attached to physical network cards and assigned a TCP/IP configuration.

For further flexibility, VLANs (IEEE 802.1q) and network bonding/aggregation are possible. In this way it is possible to build complex, flexible virtual networks for the Proxmox VE hosts, leveraging the full power of the Linux network stack.

2.1.6. Integrated Firewall

The integrated firewall allows you to filter network packets on any VM or Container interface. Common sets of firewall rules can be grouped into “security groups”.

2.1.7. Hyper-converged Infrastructure

Proxmox VE is a virtualization platform that tightly integrates compute, storage and networking resources, manages highly available clusters, backup/restore as well as disaster recovery. All components are software-defined and compatible with one another.

Therefore it is possible to administrate them like a single system via the centralized web management interface. These capabilities make Proxmox VE an ideal choice to deploy and manage an open source hyper-converged infrastructure.

2.1.7.1. Benefits of a Hyper-Converged Infrastructure (HCI) with Proxmox VE

A hyper-converged infrastructure (HCI) is especially useful for deployments in which a high infrastructure demand meets a low administration budget, for distributed setups such as remote and branch office environments or for virtual private and public clouds.

HCI provides the following advantages:

- **Scalability:** seamless expansion of compute, network and storage devices (i.e. scale up servers and storage quickly and independently from each other).
- **Low cost:** Proxmox VE is open source and integrates all components you need such as compute, storage, networking, backup, and management center. It can replace an expensive compute/storage infrastructure.
- **Data protection and efficiency:** services such as backup and disaster recovery are integrated.
- **Simplicity:** easy configuration and centralized administration.
- **Open Source:** No vendor lock-in.

2.1.7.2. Hyper-Converged Infrastructure: Storage

Proxmox VE has tightly integrated support for deploying a hyper-converged storage infrastructure. You can, for example, deploy and manage the following two storage technologies by using the web interface only:

- **Ceph:** a both self-healing and self-managing shared, reliable and highly scalable storage system. Checkout how to manage Ceph services on Proxmox VE nodes

- **ZFS**: a combined file system and logical volume manager with extensive protection against data corruption, various RAID modes, fast and cheap snapshots - among other features. Find out how to leverage the power of ZFS on Proxmox VE nodes.

Besides above, Proxmox VE has support to integrate a wide range of additional storage technologies. You can find out about them in the Storage Manager chapter.

2.1.8. Why Open Source

Proxmox VE uses a Linux kernel and is based on the Debian GNU/Linux Distribution. The source code of Proxmox VE is released under the GNU Affero General Public License, version 3. This means that you are free to inspect the source code at any time or contribute to the project yourself.

At Proxmox we are committed to use open source software whenever possible. Using open source software guarantees full access to all functionalities - as well as high security and reliability. We think that everybody should have the right to access the source code of a software to run it, build on it, or submit changes back to the project. Everybody is encouraged to contribute while Proxmox ensures the product always meets professional quality criteria.

Open source software also helps to keep your costs low and makes your core infrastructure independent from a single vendor.

2.1.9. Your benefits with Proxmox VE

- Open source software
- No vendor lock-in
- Linux kernel
- Fast installation and easy-to-use
- Web-based management interface
- REST API
- Huge active community
- Low administration costs and simple deployment

2.20 Frequently Asked Questions

New FAQs are appended to the bottom of this section.

1. *What distribution is Proxmox VE based on?*

Proxmox VE is based on Debian GNU/Linux

2. *What license does the Proxmox VE project use?*

Proxmox VE code is licensed under the GNU Affero General Public License, version 3.

3. *Will Proxmox VE run on a 32bit processor?*

Proxmox VE works only on 64-bit CPUs (AMD or Intel). There is no plan for 32-bit for the platform.

VMs and Containers can be both 32-bit and 64-bit.

4. *Does my CPU support virtualization?*

To check if your CPU is virtualization compatible, check for the vmx or svm tag in this command output:

```
egrep '(vmx|svm)' /proc/cpuinfo
```

5. *Supported Intel CPUs*

64-bit processors with Intel Virtualization Technology (Intel VT-x) support. (List of processors with Intel VT and 64-bit)

6. *Supported AMD CPUs*

64-bit processors with AMD Virtualization Technology (AMD-V) support.

7. *What is a container/virtual environment (VE)/virtual private server (VPS)?*

In the context of containers, these terms all refer to the concept of operating-system-level virtualization. Operating-system-level virtualization is a method of virtualization, in which the kernel of an operating system allows for multiple isolated instances, that all share the kernel. When referring to LXC, we call such instances containers. Because containers use the host's kernel rather than emulating a full operating system, they require less overhead, but are limited to Linux guests.

8. *What is a QEMU/KVM guest (or VM)?*

A QEMU/KVM guest (or VM) is a guest system running virtualized under Proxmox VE using QEMU and the Linux KVM kernel module.

9. *What is QEMU?*

QEMU is a generic and open source machine emulator and virtualizer. QEMU uses the Linux KVM kernel module to achieve near native performance by executing the guest code directly on the host CPU. It is not limited to Linux guests but allows arbitrary operating systems to run.

10. *How long will my Proxmox VE version be supported?*

Proxmox VE versions are supported at least as long as the corresponding Debian Version is oldstable. Proxmox VE uses a rolling release model and using the latest stable version is always recommended.

Proxmox VE Version	Debian Version	First Release	Debian EOL	Proxmox EOL
Proxmox VE 8	Debian 12 (Bookworm)	2023-06	tba	tba
Proxmox VE 7	Debian 11 (Bullseye)	2021-07	2024-07	2024-07
Proxmox VE 6	Debian 10 (Buster)	2019-07	2022-09	2022-09
Proxmox VE 5	Debian 9 (Stretch)	2017-07	2020-07	2020-07
Proxmox VE 4	Debian 8 (Jessie)	2015-10	2018-06	2018-06

Proxmox VE 3	Debian 7 (Wheezy)	2013-05	2016-04	2017-02
Proxmox VE 2	Debian 6 (Squeeze)	2012-04	2014-05	2014-05
Proxmox VE 1	Debian 5 (Lenny)	2008-10	2012-03	2013-01

11. *How can I upgrade Proxmox VE to the next point release?*

Minor version upgrades, for example upgrading from Proxmox VE in version 7.1 to 7.2 or 7.3, can be done just like any normal update. But you should still check the release notes for any relevant notable, or breaking change.

For the update itself use either the Web UI *Node* → *Updates* panel or through the CLI with:

```
apt update
apt full-upgrade
```

Always ensure you correctly setup the package repositories and only continue with the actual upgrade if apt update did not hit any error.

12. *How can I upgrade Proxmox VE to the next major release?*

Major version upgrades, for example going from Proxmox VE 4.4 to 5.0, are also supported. They must be carefully planned and tested and should **never** be started without having a current backup ready.

Although the specific upgrade steps depend on your respective setup, we provide general instructions and advice of how a upgrade should be performed:

- a. Upgrade from Proxmox VE 7 to 8
- b. Upgrade from Proxmox VE 6 to 7
- c. Upgrade from Proxmox VE 5 to 6
- d. Upgrade from Proxmox VE 4 to 5
- e. Upgrade from Proxmox VE 3 to 4

13. *LXC vs LXD vs Proxmox Containers vs Docker*

LXC is a userspace interface for the Linux kernel containment features. Through a powerful API and simple tools, it lets Linux users easily create and manage system containers. LXC, as well as the former OpenVZ, aims at **system virtualization**. Thus, it allows you to run a complete OS inside a container, where you log in using ssh, add users, run apache, etc...

LXD is built on top of LXC to provide a new, better user experience. Under the hood, LXD uses LXC through liblxc and its Go binding to create and manage the containers. It's basically an alternative to LXC's tools and distribution template system with the added features that come from being controllable over the network.

Proxmox Containers are how we refer to containers that are created and managed using the Proxmox Container Toolkit (pct). They also target **system virtualization** and use LXC as the basis of the container offering. The Proxmox Container Toolkit (pct) is tightly coupled with Proxmox VE. This means that it is aware of cluster setups, and it can use the same network and storage resources as QEMU virtual machines (VMs). You can even use the Proxmox VE firewall, create and restore backups, or manage containers using the HA framework. Everything can be controlled over the network using the Proxmox VE API.

Docker aims at running a **single** application in an isolated, self-contained environment. These are generally referred to as “Application Containers”, rather than “System Containers”. You manage a Docker instance from the host, using the Docker Engine command-line interface. It is not recommended to run docker directly on your Proxmox VE host.

█ If you want to run application containers, for example, *Docker* images, it is best to run them inside a Proxmox QEMU VM.

Parte 3: Plataformas de contenerización

Docker desktop

Documentación Oficial : <https://docs.docker.com/get-started/>

3.1 Docker Engine

Docker Engine is an open source containerization technology for building and containerizing your applications. Docker Engine acts as a client-server application with:

A server with a long-running daemon process dockerd.

APIs which specify interfaces that programs can use to talk to and instruct the Docker daemon.

A command line interface (CLI) client docker.

The CLI uses Docker APIs to control or interact with the Docker daemon through scripting or direct CLI commands. Many other Docker applications use the underlying API and CLI. The daemon creates and manages Docker objects, such as images, containers, networks, and volumes.

3.2 What is a container?

Explanation

Imagine you're developing a killer web app that has three main components - a React frontend, a Python API, and a PostgreSQL database. If you wanted to work on this project, you'd have to install Node, Python, and PostgreSQL.

How do you make sure you have the same versions as the other developers on your team? Or your CI/CD system? Or what's used in production?

How do you ensure the version of Python (or Node or the database) your app needs isn't affected by what's already on your machine? How do you manage potential conflicts?

Enter containers!

What is a container? Simply put, containers are isolated processes for each of your app's components. Each component - the frontend React app, the Python API engine, and the database - runs in its own isolated environment, completely isolated from everything else on your machine.

Here's what makes them awesome. Containers are:

Self-contained. Each container has everything it needs to function with no reliance on any pre-installed dependencies on the host machine.

Isolated. Since containers are run in isolation, they have minimal influence on the host and other containers, increasing the security of your applications.

Independent. Each container is independently managed. Deleting one container won't affect any others.

Portable. Containers can run anywhere! The container that runs on your development machine will work the same way in a data center or anywhere in the cloud!

Containers versus virtual machines (VMs)

Without getting too deep, a VM is an entire operating system with its own kernel, hardware drivers, programs, and applications. Spinning up a VM only to isolate a single application is a lot of overhead.

A container is simply an isolated process with all of the files it needs to run. If you run multiple containers, they all share the same kernel, allowing you to run more applications on less infrastructure.

Using VMs and containers together

Quite often, you will see containers and VMs used together. As an example, in a cloud environment, the provisioned machines are typically VMs. However, instead of provisioning one machine to run one application, a VM with a container runtime can run multiple containerized applications, increasing resource utilization and reducing costs.

Try it out

In this hands-on, you will see how to run a Docker container using the Docker Desktop GUI.

Follow the instructions to run a container using the CLI:

Open your CLI terminal and start a container by using the docker run command:

```
docker run -d -p 8080:80 docker/welcome-to-docker
```

The output from this command is the full container ID.

Congratulations! You just fired up your first container!

View your running containers

You can verify if the container is up and running by using the docker ps command:

```
docker ps
```

You will see output like the following:

<i>CONTAINER ID</i>	<i>IMAGE</i>	<i>COMMAND</i>	<i>CREATED</i>	<i>STATUS</i>
<i>PORTS</i>	<i>NAMES</i>			
<i>a1f7a4bb3a27</i>	<i>docker/welcome-to-docker</i>	<i>"/docker-entrypt..."</i>	<i>11 seconds ago</i>	<i>Up 11 seconds</i>
<i>0.0.0.0:8080->80/tcp</i>	<i>gracious_keldysh</i>			

This container runs a web server that displays a simple website. When working with more complex projects, you'll run different parts in different containers. For example, a different container for the frontend, backend, and database.

Tip

The docker ps command will show you only running containers. To view stopped containers, add the -a flag to list all containers: docker ps -a

Stop your container

The `docker/welcome-to-docker` container continues to run until you stop it. You can stop a container using the `docker stop` command.

Run `docker ps` to get the ID of the container

Provide the container ID or name to the `docker stop` command:

```
docker stop <the-container-id>
```

3.3 What is an image?

Explanation

Seeing a container is an isolated process, where does it get its files and configuration? How do you share those environments?

That's where container images come in. A container image is a standardized package that includes all of the files, binaries, libraries, and configurations to run a container.

For a PostgreSQL image, that image will package the database binaries, config files, and other dependencies. For a Python web app, it'll include the Python runtime, your app code, and all of its dependencies.

There are two important principles of images:

Images are immutable. Once an image is created, it can't be modified. You can only make a new image or add changes on top of it.

Container images are composed of layers. Each layer represents a set of file system changes that add, remove, or modify files.

These two principles let you to extend or add to existing images. For example, if you are building a Python app, you can start from the Python image and add additional layers to install your app's dependencies and add your code. This lets you focus on your app, rather than Python itself.

Finding images

Docker Hub is the default global marketplace for storing and distributing images. It has over 100,000 images created by developers that you can run locally. You can search for Docker Hub images and run them directly from Docker Desktop.

Docker Hub provides a variety of Docker-supported and endorsed images known as Docker Trusted Content. These provide fully managed services or great starters for your own images. These include:

Docker Official Images - a curated set of Docker repositories, serve as the starting point for the majority of users, and are some of the most secure on Docker Hub

Docker Verified Publishers - high-quality images from commercial publishers verified by Docker

Docker-Sponsored Open Source - images published and maintained by open-source projects sponsored by Docker through Docker's open source program

For example, Redis and Memcached are a few popular ready-to-go Docker Official Images. You can download these images and have these services up and running in a matter of seconds.

There are also base images, like the Node.js Docker image, that you can use as a starting point and add your own files and configurations.

Try it out

Follow the instructions to search and pull a Docker image using CLI to view its layers.

Search for and download an image

Open a terminal and search for images using the docker search command:

```
docker search docker/welcome-to-docker
```

You will see output like the following:

<i>NAME</i>	<i>DESCRIPTION</i>	<i>STARS</i>	<i>OFFICIAL</i>
<i>docker/welcome-to-docker</i>	<i>Docker image for new users getting started w...</i>	<i>20</i>	

This output shows you information about relevant images available on Docker Hub.

Pull the image using the docker pull command.

```
docker pull docker/welcome-to-docker
```

You will see output like the following:

```
Using default tag: latest
```

```
latest: Pulling from docker/welcome-to-docker
```

```
579b34f0a95b: Download complete
```

```
d11a451e6399: Download complete
```

```
1c2214f9937c: Download complete
```

```
b42a2f288f4d: Download complete
```

```
54b19e12c655: Download complete
```

```
1fb28e078240: Download complete
```

```
94be7e780731: Download complete
```

```
89578ce72c35: Download complete
```

```
Digest: sha256:eedaff45e3c78538087bdd9dc7afafac7e110061bbdd836af4104b10f10ab693
```

```
Status: Downloaded newer image for docker/welcome-to-docker:latest
```

```
docker.io/docker/welcome-to-docker:latest
```

Each of line represents a different downloaded layer of the image. Remember that each layer is a set of filesystem changes and provides functionality of the image.

Learn about the image

List your downloaded images using the docker image ls command:

```
docker image ls
```

You will see output like the following:

```
REPOSITORY          TAG      IMAGE ID      CREATED      SIZE
docker/welcome-to-docker latest  eedaff45e3c7 4 months ago 29.7MB
```

The command shows a list of Docker images currently available on your system. The docker/welcome-to-docker has a total size of approximately 29.7MB.

Image size

The image size represented here reflects the uncompressed size of the image, not the download size of the layers.

List the image's layers using the docker image history command:

```
docker image history docker/welcome-to-docker
```

You will see output like the following:

```
IMAGE      CREATED      CREATED BY          SIZE      COMMENT
648f93a1ba7d 4 months ago COPY /app/build /usr/share/nginx/html # buil... 1.6MB
buildkit.dockerfile.v0
<missing>    5 months ago /bin/sh -c #(nop) CMD ["nginx" "-g" "daemon... 0B
<missing>    5 months ago /bin/sh -c #(nop) STOPSIGNAL SIGQUIT          0B
<missing>    5 months ago /bin/sh -c #(nop) EXPOSE 80                    0B
<missing>    5 months ago /bin/sh -c #(nop) ENTRYPOINT ["/docker-entr... 0B
<missing>    5 months ago /bin/sh -c #(nop) COPY file:9e3b2b63db9f8fc7... 4.62kB
<missing>    5 months ago /bin/sh -c #(nop) COPY file:57846632acc8975... 3.02kB
<missing>    5 months ago /bin/sh -c #(nop) COPY file:3b1b9915b7dd898a... 298B
<missing>    5 months ago /bin/sh -c #(nop) COPY file:caec368f5a54f70a... 2.12kB
<missing>    5 months ago /bin/sh -c #(nop) COPY file:01e75c6dd0ce317d... 1.62kB
<missing>    5 months ago /bin/sh -c set -x  && addgroup -g 101 -S ... 9.7MB
<missing>    5 months ago /bin/sh -c #(nop) ENV PKG_RELEASE=1          0B
<missing>    5 months ago /bin/sh -c #(nop) ENV NGINX_VERSION=1.25.3  0B
<missing>    5 months ago /bin/sh -c #(nop) LABEL maintainer=NGINX Do... 0B
<missing>    5 months ago /bin/sh -c #(nop) CMD ["/bin/sh"]          0B
<missing>    5 months ago /bin/sh -c #(nop) ADD file:ff3112828967e8004... 7.66MB
```

This output shows you all of the layers, their sizes, and the command used to create the layer.

Viewing the full command

If you add the `--no-trunc` flag to the command, you will see the full command. Note that, since the output is in a table-like format, longer commands will cause the output to be very difficult to navigate.

In this walkthrough, you searched and pulled a Docker image. In addition to pulling a Docker image, you also learned about the layers of a Docker Image.

3.4 What is Docker Compose?

Explanation

If you've been following the guides so far, you've been working with single container applications. But, now you're wanting to do something more complicated - run databases, message queues, caches, or a variety of other services. Do you install everything in a single container? Run multiple containers? If you run multiple, how do you connect them all together?

One best practice for containers is that each container should do one thing and do it well. While there are exceptions to this rule, avoid the tendency to have one container do multiple things.

You can use multiple `docker run` commands to start multiple containers. But, you'll soon realize you'll need to manage networks, all of the flags needed to connect containers to those networks, and more. And when you're done, cleanup is a little more complicated.

With Docker Compose, you can define all of your containers and their configurations in a single YAML file. If you include this file in your code repository, anyone that clones your repository can get up and running with a single command.

It's important to understand that Compose is a declarative tool - you simply define it and go. You don't always need to recreate everything from scratch. If you make a change, run `docker compose up` again and Compose will reconcile the changes in your file and apply them intelligently.

Dockerfile versus Compose file

A Dockerfile provides instructions to build a container image while a Compose file defines your running containers. Quite often, a Compose file references a Dockerfile to build an image to use for a particular service.

Try it out

In this hands-on, you will learn how to use a Docker Compose to run a multi-container application. You'll use a simple to-do list app built with Node.js and MySQL as a database server.

Start the application

Follow the instructions to run the to-do list app on your system.

Download and install Docker Desktop.

Open a terminal and clone this sample application.

```
git clone https://github.com/dockersamples/todo-list-app
```

Navigate into the todo-list-app directory:

```
cd todo-list-app
```

Inside this directory, you'll find a file named `compose.yaml`. This YAML file is where all the magic happens! It defines all the services that make up your application, along with their configurations. Each service specifies its image, ports, volumes, networks, and any other settings necessary for its functionality. Take some time to explore the YAML file and familiarize yourself with its structure.

Use the `docker compose up` command to start the application:

```
docker compose up -d --build
```

When you run this command, you should see an output like this:

```
[+] Running 4/4
```

```
✓ app 3 layers [⋮⋮⋮] 0B/0B Pulled 7.1s
✓ e6f4e57cc59e Download complete 0.9s
✓ df998480d81d Download complete 1.0s
✓ 31e174fedd23 Download complete 2.5s
```

```
[+] Running 2/4
```

```
⋮ Network todo-list-app_default Created 0.3s
⋮ Volume "todo-list-app_todo-mysql-data" Created 0.3s
✓ Container todo-list-app-app-1 Started 0.3s
✓ Container todo-list-app-mysql-1 Started 0.3s
```

A lot happened here! A couple of things to call out:

Two container images were downloaded from Docker Hub - node and MySQL

A network was created for your application

A volume was created to persist the database files between container restarts

Two containers were started with all of their necessary config

If this feels overwhelming, don't worry! You'll get there!

With everything now up and running, you can open `http://localhost:3000` in your browser to see the site. Feel free to add items to the list, check them off, and remove them.

Tear it down

Since this application was started using Docker Compose, it's easy to tear it all down when you're done.

In the CLI, use the docker compose down command to remove everything:

```
docker compose down
```

You'll see output similar to the following:

```
[+] Running 2/2
```

```
✓ Container todo-list-app-mysql-1 Removed    2.9s
✓ Container todo-list-app-app-1  Removed    0.1s
✓ Network todo-list-app_default  Removed    0.1s
```

Volume persistence

By default, volumes aren't automatically removed when you tear down a Compose stack. The idea is that you might want the data back if you start the stack again.

If you do want to remove the volumes, add the `--volumes` flag when running the docker compose down command:

```
docker compose down --volumes
```

3.5 Publishing and exposing ports

Explanation

If you've been following the guides so far, you understand that containers provide isolated processes for each component of your application. Each component - a React frontend, a Python API, and a Postgres database - runs in its own sandbox environment, completely isolated from everything else on your host machine. This isolation is great for security and managing dependencies, but it also means you can't access them directly. For example, you can't access the web app in your browser.

That's where port publishing comes in.

Publishing ports

Publishing a port provides the ability to break through a little bit of networking isolation by setting up a forwarding rule. As an example, you can indicate that requests on your host's port 8080 should be forwarded to the container's port 80. Publishing ports happens during container creation using the `-p` (or `--publish`) flag with docker run. The syntax is:

```
docker run -d -p HOST_PORT:CONTAINER_PORT nginx
```

HOST_PORT: The port number on your host machine where you want to receive traffic

CONTAINER_PORT: The port number within the container that's listening for connections

For example, to publish the container's port 80 to host port 8080:

```
docker run -d -p 8080:80 nginx
```

Now, any traffic sent to port 8080 on your host machine will be forwarded to port 80 within the container.

Important

When a port is published, it's published to all network interfaces by default. This means any traffic that reaches your machine can access the published application. Be mindful of publishing databases or any sensitive information. Learn more about published ports here.

Publishing to ephemeral ports

At times, you may want to simply publish the port but don't care which host port is used. In these cases, you can let Docker pick the port for you. To do so, simply omit the `HOST_PORT` configuration.

For example, the following command will publish the container's port 80 onto an ephemeral port on the host:

```
docker run -p 80 nginx
```

Once the container is running, using `docker ps` will show you the port that was chosen:

```
docker ps
```

<i>CONTAINER ID</i>	<i>IMAGE</i>	<i>COMMAND</i>	<i>CREATED</i>	<i>STATUS</i>	<i>PORTS</i>
<i>a527355c9c53</i>	<i>nginx</i>	<i>"/docker-entrypoint..."</i>	<i>4 seconds ago</i>	<i>Up 3 seconds</i>	<i>0.0.0.0:54772->80/tcp</i>

In this example, the app is exposed on the host at port 54772.

Publishing all ports

When creating a container image, the `EXPOSE` instruction is used to indicate the packaged application will use the specified port. These ports aren't published by default.

With the `-P` or `--publish-all` flag, you can automatically publish all exposed ports to ephemeral ports. This is quite useful when you're trying to avoid port conflicts in development or testing environments.

For example, the following command will publish all of the exposed ports configured by the image:

```
docker run -P nginx
```

Try it out

In this hands-on guide, you'll learn how to publish container ports using both the CLI and Docker Compose for deploying a web application.

Use the Docker CLI

In this step, you will run a container and publish its port using the Docker CLI.

Download and install Docker Desktop.

In a terminal, run the following command to start a new container:

```
docker run -d -p 8080:80 docker/welcome-to-docker
```

The first 8080 refers to the host port. This is the port on your local machine that will be used to access the application running inside the container. The second 80 refers to the container port. This is the port that the application inside the container listens on for incoming connections. Hence, the command binds to port 8080 of the host to port 80 on the container system.

3.6 Overriding container defaults

Explanation

When a Docker container starts, it executes an application or command. The container gets this executable (script or file) from its image's configuration. Containers come with default settings that usually work well, but you can change them if needed. These adjustments help the container's program run exactly how you want it to.

For example, if you have an existing database container that listens on the standard port and you want to run a new instance of the same database container, then you might want to change the port settings the new container listens on so that it doesn't conflict with the existing container. Sometimes you might want to increase the memory available to the container if the program needs more resources to handle a heavy workload or set the environment variables to provide specific configuration details the program needs to function properly.

The docker run command offers a powerful way to override these defaults and tailor the container's behavior to your liking. The command offers several flags that let you to customize container behavior on the fly.

Here's a few ways you can achieve this.

Overriding the network ports

Sometimes you might want to use separate database instances for development and testing purposes. Running these database instances on the same port might conflict. You can use the `-p` option in docker run to map container ports to host ports, allowing you to run the multiple instances of the container without any conflict.

```
docker run -d -p HOST_PORT:CONTAINER_PORT postgres
```

Setting environment variables

This option sets an environment variable foo inside the container with the value bar.

```
docker run -e foo=bar postgres env
```

You will see output like the following:

```
HOSTNAME=2042f2e6ebe4
```

```
foo=bar
```

Tip

The `.env` file acts as a convenient way to set environment variables for your Docker containers without cluttering your command line with numerous `-e` flags. To use a `.env` file, you can pass `--env-file` option with the docker run command.

```
docker run --env-file .env postgres env
```

Restricting the container to consume the resources

You can use the `--memory` and `--cpus` flags with the `docker run` command to restrict how much CPU and memory a container can use. For example, you can set a memory limit for the Python API container, preventing it from consuming excessive resources on your host. Here's the command:

```
docker run -e POSTGRES_PASSWORD=secret --memory="512m" --cpus="0.5" postgres
```

This command limits container memory usage to 512 MB and defines the CPU quota of 0.5 for half a core.

Monitor the real-time resource usage

You can use the `docker stats` command to monitor the real-time resource usage of running containers. This helps you understand whether the allocated resources are sufficient or need adjustment.

By effectively using these `docker run` flags, you can tailor your containerized application's behavior to fit your specific requirements.

Try it out

In this hands-on guide, you'll see how to use the `docker run` command to override the container defaults.

Download and install Docker Desktop.

Run multiple instance of the Postgres database

Start a container using the Postgres image with the following command:

```
docker run -d -e POSTGRES_PASSWORD=secret -p 5432:5432 postgres
```

This will start the Postgres database in the background, listening on the standard container port 5432 and mapped to port 5432 on the host machine.

Start a second Postgres container mapped to a different port.

```
docker run -d -e POSTGRES_PASSWORD=secret -p 5433:5432 postgres
```

This will start another Postgres container in the background, listening on the standard postgres port 5432 in the container, but mapped to port 5433 on the host machine. You override the host port just to ensure that this new container doesn't conflict with the existing running container.

Run Postgres container in a controlled network

By default, containers automatically connect to a special network called a bridge network when you run them. This bridge network acts like a virtual bridge, allowing containers on the same host to communicate with each other while keeping them isolated from the outside world and other hosts. It's a convenient starting point for most container interactions. However, for specific scenarios, you might want more control over the network configuration.

Here's where the custom network comes in. You create a custom network by passing `--network` flag with the `docker run` command. All containers without a `--network` flag are attached to the default bridge network.

Follow the steps to see how to connect a Postgres container to a custom network.

Create a new custom network by using the following command:

```
docker network create mynetwork
```

Verify the network by running the following command:

```
docker network ls
```

This command lists all networks, including the newly created "mynetwork".

Connect Postgres to the custom network by using the following command:

```
docker run -d -e POSTGRES_PASSWORD=secret -p 5434:5432 --network mynetwork postgres
```

This will start Postgres container in the background, mapped to the host port 5434 and attached to the mynetwork network. You passed the `--network` parameter to override the container default by connecting the container to custom Docker network for better isolation and communication with other containers. You can use `docker network inspect` command to see if the container is tied to this new bridge network.

Key difference between default bridge and custom networks

DNS resolution: By default, containers connected to the default bridge network can communicate with each other, but only by IP address. (unless you use `--link` option which is considered legacy). It is not recommended for production use due to the various technical shortcomings. On a custom network, containers can resolve each other by name or alias.

Isolation: All containers without a `--network` specified are attached to the default bridge network, hence can be a risk, as unrelated containers are then able to communicate. Using a custom network provides a scoped network in which only containers attached to that network are able to communicate, hence providing better isolation.

Manage the resources

By default, containers are not limited in their resource usage. However, on shared systems, it's crucial to manage resources effectively. It's important not to let a running container consume too much of the host machine's memory.

This is where the `docker run` command shines again. It offers flags like `--memory` and `--cpus` to restrict how much CPU and memory a container can use.

```
docker run -d -e POSTGRES_PASSWORD=secret --memory="512m" --cpus=".5" postgres
```

The `--cpus` flag specifies the CPU quota for the container. Here, it's set to half a CPU core (0.5) whereas the `--memory` flag specifies the memory limit for the container. In this case, it's set to 512 MB.

Override the default CMD and ENTRYPOINT in Docker Compose

Sometimes, you might need to override the default commands (CMD) or entry points (ENTRYPOINT) defined in a Docker image, especially when using Docker Compose.

Create a `compose.yml` file with the following content:

services:

postgres:

image: postgres

entrypoint: ["docker-entrypoint.sh", "postgres"]

command: ["-h", "localhost", "-p", "5432"]

environment:

POSTGRES_PASSWORD: secret

The Compose file defines a service named postgres that uses the official Postgres image, sets an entrypoint script, and starts the container with password authentication.

Bring up the service by running the following command:

```
docker compose up -d
```

This command starts the Postgres service defined in the Docker Compose file.

3.7 Persisting container data

Explanation

When a container starts, it uses the files and configuration provided by the image. Each container is able to create, modify, and delete files and does so without affecting any other containers. When the container is deleted, these file changes are also deleted.

While this ephemeral nature of containers is great, it poses a challenge when you want to persist the data. For example, if you restart a database container, you might not want to start with an empty database. So, how do you persist files?

Container volumes

Volumes are a storage mechanism that provide the ability to persist data beyond the lifecycle of an individual container. Think of it like providing a shortcut or symlink from inside the container to outside the container.

As an example, imagine you create a volume named log-data.

```
docker volume create log-data
```

When starting a container with the following command, the volume will be mounted (or attached) into the container at /logs:

```
docker run -d -p 80:80 -v log-data:/logs docker/welcome-to-docker
```

If the volume log-data doesn't exist, Docker will automatically create it for you.

When the container runs, all files it writes into the /logs folder will be saved in this volume, outside of the container. If you delete the container and start a new container using the same volume, the files will still be there.

Sharing files using volumes

You can attach the same volume to multiple containers to share files between containers. This might be helpful in scenarios such as log aggregation, data pipelines, or other event-driven applications.

Managing volumes

Volumes have their own lifecycle beyond that of containers and can grow quite large depending on the type of data and applications you're using. The following commands will be helpful to manage volumes:

```
docker volume ls - list all volumes
```

```
docker volume rm <volume-name-or-id> - remove a volume (only works when the volume is not attached to any containers)
```

```
docker volume prune - remove all unused (unattached) volumes
```

Try it out

In this guide, you'll practice creating and using volumes to persist data created by a Postgres container. When the database runs, it stores files into the `/var/lib/postgresql/data` directory. By attaching the volume here, you will be able to restart the container multiple times while keeping the data.

Use volumes

Download and install Docker Desktop.

Start a container using the Postgres image with the following command:

```
docker run --name=db -e POSTGRES_PASSWORD=secret -d -v postgres_data:/var/lib/postgresql/data postgres
```

This will start the database in the background, configure it with a password, and attach a volume to the directory PostgreSQL will persist the database files.

Connect to the database by using the following command:

```
docker exec -ti db psql -U postgres
```

In the PostgreSQL command line, run the following to create a database table and insert two records:

```
CREATE TABLE tasks (
```

```
  id SERIAL PRIMARY KEY,
```

```
  description VARCHAR(100)
```

```
);
```

```
INSERT INTO tasks (description) VALUES ('Finish work'), ('Have fun');
```

Verify the data is in the database by running the following in the PostgreSQL command line:

```
SELECT * FROM tasks;
```

You should get output that looks like the following:

```
id | description
```

```
----+-----
```

```
1 | Finish work
```

```
2 | Have fun
```

```
(2 rows)
```

Exit out of the PostgreSQL shell by running the following command:

```
\q
```

Stop and remove the database container. Remember that, even though the container has been deleted, the data is persisted in the `postgres_data` volume.

```
docker stop db
```

```
docker rm db
```

Start a new container by running the following command, attaching the same volume with the persisted data:

```
docker run --name=new-db -d -v postgres_data:/var/lib/postgresql/data postgres
```

You might have noticed that the `POSTGRES_PASSWORD` environment variable has been omitted. That's because that variable is only used when bootstrapping a new database.

Verify the database still has the records by running the following command:

```
docker exec -ti new-db psql -U postgres -c "SELECT * FROM tasks"
```

View volume contents

The Docker Desktop Dashboard provides the ability to view the contents of any volume, as well as the ability to export, import, and clone volumes.

Open the Docker Desktop Dashboard and navigate to the Volumes view. In this view, you should see the `postgres_data` volume.

Select the `postgres_data` volume's name.

The Data tab shows the contents of the volume and provides the ability to navigate the files. Double-clicking on a file will let you see the contents and make changes.

Right-click on any file to save it or delete it.

Remove volumes

Before removing a volume, it must not be attached to any containers. If you haven't removed the previous container, do so with the following command (the `-f` will stop the container first and then remove it):


```
docker rm -f new-db
```

There are a few methods to remove volumes, including the following:

Select the Delete Volume option on a volume in the Docker Desktop Dashboard.

Use the docker volume rm command:

```
docker volume rm postgres_data
```

Use the docker volume prune command to remove all unused volumes:

```
docker volume prune
```

3.8 Sharing local files with containers

Explanation

Each container has everything it needs to function with no reliance on any pre-installed dependencies on the host machine. Since containers run in isolation, they have minimal influence on the host and other containers. This isolation has a major benefit: containers minimize conflicts with the host system and other containers. However, this isolation also means containers can't directly access data on the host machine by default.

Consider a scenario where you have a web application container that requires access to configuration settings stored in a file on your host system. This file may contain sensitive data such as database credentials or API keys. Storing such sensitive information directly within the container image poses security risks, especially during image sharing. To address this challenge, Docker offers storage options that bridge the gap between container isolation and your host machine's data.

Docker offers two primary storage options for persisting data and sharing files between the host machine and containers: volumes and bind mounts.

Volume versus bind mounts

If you want to ensure that data generated or modified inside the container persists even after the container stops running, you would opt for a volume. See [Persisting container data](#) to learn more about volumes and their use cases.

If you have specific files or directories on your host system that you want to directly share with your container, like configuration files or development code, then you would use a bind mount. It's like opening a direct portal between your host and container for sharing. Bind mounts are ideal for development environments where real-time file access and sharing between the host and container are crucial.

Sharing files between a host and container

Both `-v` (or `--volume`) and `--mount` flags used with the `docker run` command let you share files or directories between your local machine (host) and a Docker container. However, there are some key differences in their behavior and usage.

The `-v` flag is simpler and more convenient for basic volume or bind mount operations. If the host location doesn't exist when using `-v` or `--volume`, a directory will be automatically created.

Imagine you're a developer working on a project. You have a source directory on your development machine where your code resides. When you compile or build your code, the generated artifacts (compiled code, executables, images, etc.) are saved in a separate subdirectory within your source directory. In the following examples, this subdirectory is `/HOST/PATH`. Now you want these build artifacts to be accessible within a Docker container running your application. Additionally, you want the container to automatically access the latest build artifacts whenever you rebuild your code.

Here's a way to use `docker run` to start a container using a bind mount and map it to the container file location.

```
docker run -v /HOST/PATH:/CONTAINER/PATH -it nginx
```

The `--mount` flag offers more advanced features and granular control, making it suitable for complex mount scenarios or production deployments. If you use `--mount` to bind-mount a file or directory that doesn't yet exist on the Docker host, the `docker run` command doesn't automatically create it for you but generates an error.

```
docker run --mount type=bind,source=/HOST/PATH,target=/CONTAINER/PATH,readonly nginx
```

Note

Docker recommends using the `--mount` syntax instead of `-v`. It provides better control over the mounting process and avoids potential issues with missing directories.

File permissions for Docker access to host files

When using bind mounts, it's crucial to ensure that Docker has the necessary permissions to access the host directory. To grant read/write access, you can use the `:ro` flag (read-only) or `:rw` (read-write) with the `-v` or `--mount` flag during container creation. For example, the following command grants read-write access permission.

```
docker run -v HOST-DIRECTORY:/CONTAINER-DIRECTORY:rw nginx
```

Read-only bind mounts let the container access the mounted files on the host for reading, but it can't change or delete the files. With read-write bind mounts, containers can modify or delete mounted files, and these changes or deletions will also be reflected on the host system. Read-only bind mounts ensures that files on the host can't be accidentally modified or deleted by a container.

Synchronized File Share

As your codebase grows larger, traditional methods of file sharing like bind mounts may become inefficient or slow, especially in development environments where frequent access to files is necessary. Synchronized file shares improve bind mount performance by leveraging synchronized filesystem caches. This optimization ensures that file access between the host and virtual machine (VM) is fast and efficient.

Try it out

In this hands-on guide, you'll practice how to create and use a bind mount to share files between a host and a container.

Run a container

Download and install Docker Desktop.

Start a container using the httpd image with the following command:

```
docker run -d -p 8080:80 --name my_site httpd:2.4
```

This will start the httpd service in the background, and publish the webpage to port 8080 on the host.

Open the browser and access <http://localhost:8080> or use the curl command to verify if it's working fine or not.

```
curl localhost:8080
```

3.9 Multi-container applications

Explanation

Starting up a single-container application is easy. For example, a Python script that performs a specific data processing task runs within a container with all its dependencies. Similarly, a Node.js application serving a static website with a small API endpoint can be effectively containerized with all its necessary libraries and dependencies. However, as applications grow in size, managing them as individual containers becomes more difficult.

Imagine the data processing Python script needs to connect to a database. Suddenly, you're now managing not just the script but also a database server within the same container. If the script requires user logins, you'll need an authentication mechanism, further bloating the container size.

One best practice for containers is that each container should do one thing and do it well. While there are exceptions to this rule, avoid the tendency to have one container do multiple things.

Now you might ask, "Do I need to run these containers separately? If I run them separately, how shall I connect them all together?"

While docker run is a convenient tool for launching containers, it becomes difficult to manage a growing application stack with it. Here's why:

Imagine running several docker run commands (frontend, backend, and database) with different configurations for development, testing, and production environments. It's error-prone and time-consuming.

Applications often rely on each other. Manually starting containers in a specific order and managing network connections become difficult as the stack expands.

Each application needs its docker run command, making it difficult to scale individual services. Scaling the entire application means potentially wasting resources on components that don't need a boost.

Persisting data for each application requires separate volume mounts or configurations within each docker run command. This creates a scattered data management approach.

Setting environment variables for each application through separate docker run commands is tedious and error-prone.

That's where Docker Compose comes to the rescue.

Docker Compose defines your entire multi-container application in a single YAML file called `compose.yml`. This file specifies configurations for all your containers, their dependencies, environment variables, and even volumes and networks. With Docker Compose:

You don't need to run multiple `docker run` commands. All you need to do is define your entire multi-container application in a single YAML file. This centralizes configuration and simplifies management.

You can run containers in a specific order and manage network connections easily.

You can simply scale individual services up or down within the multi-container setup. This allows for efficient allocation based on real-time needs.

You can implement persistent volumes with ease.

It's easy to set environment variables once in your Docker Compose file.

By leveraging Docker Compose for running multi-container setups, you can build complex applications with modularity, scalability, and consistency at their core.

Try it out

In this hands-on guide, you'll first see how to build and run a counter web application based on Node.js, an Nginx reverse proxy, and a Redis database using the `docker run` commands. You'll also see how you can simplify the entire deployment process using Docker Compose.

Set up

Get the sample application. If you have Git, you can clone the repository for the sample application. Otherwise, you can download the sample application. Choose one of the following options.

Clone with git [Download](#)

Use the following command in a terminal to clone the sample application repository.

```
git clone https://github.com/dockersamples/nginx-node-redis
```

Navigate into the `nginx-node-redis` directory:

```
cd nginx-node-redis
```

Inside this directory, you'll find two sub-directories - `nginx` and `web`.

Download and install Docker Desktop.

Build the images

Navigate into the `nginx` directory to build the image by running the following command:

```
docker build -t nginx .
```

Navigate into the `web` directory and run the following command to build the first web image:

```
docker build -t web .
```

Run the containers

Before you can run a multi-container application, you need to create a network for them all to communicate through. You can do so using the docker network create command:

```
docker network create sample-app
```

Start the Redis container by running the following command, which will attach it to the previously created network and create a network alias (useful for DNS lookups):

```
docker run -d --name redis --network sample-app --network-alias redis redis
```

Start the first web container by running the following command:

```
docker run -d --name web1 -h web1 --network sample-app --network-alias web1 web
```

Start the second web container by running the following:

```
docker run -d --name web2 -h web2 --network sample-app --network-alias web2 web
```

Start the Nginx container by running the following command:

```
docker run -d --name nginx --network sample-app -p 80:80 nginx
```

Note

Nginx is typically used as a reverse proxy for web applications, routing traffic to backend servers. In this case, it routes to the Node.js backend containers (web1 or web2).

Verify the containers are up by running the following command:

```
docker ps
```

You will see output like the following:

<i>CONTAINER ID</i>	<i>IMAGE</i>	<i>COMMAND</i>	<i>CREATED</i>	<i>STATUS</i>	<i>PORTS</i>
<i>2cf7c484c144</i>	<i>nginx</i>	<i>"/docker-entripoint...."</i>	<i>9 seconds ago</i>	<i>Up 8 seconds</i>	<i>0.0.0.0:80->80/tcp nginx</i>
<i>7a070c9ffeaa</i>	<i>web</i>	<i>"docker-entripoint.s..."</i>	<i>19 seconds ago</i>	<i>Up 18 seconds</i>	<i>web2</i>
<i>6dc6d4e60aaf</i>	<i>web</i>	<i>"docker-entripoint.s..."</i>	<i>34 seconds ago</i>	<i>Up 33 seconds</i>	<i>web1</i>
<i>008e0ecf4f36</i>	<i>redis</i>	<i>"docker-entripoint.s..."</i>	<i>About a minute ago</i>	<i>Up About a minute</i>	<i>6379/tcp</i>

Simplify the deployment using Docker Compose

Docker Compose provides a structured and streamlined approach for managing multi-container deployments. As stated earlier, with Docker Compose, you don't need to run multiple docker run commands. All you need to do is define your entire multi-container application in a single YAML file called compose.yml. Let's see how it works.

Navigate to the root of the project directory. Inside this directory, you'll find a file named compose.yml. This YAML file is where all the magic happens. It defines all the services that make

up your application, along with their configurations. Each service specifies its image, ports, volumes, networks, and any other settings necessary for its functionality.

Use the docker compose up command to start the application:

```
docker compose up -d --build
```

When you run this command, you should see output similar to the following:

Running 5/5

```
✓ Network nginx-nodejs-redis_default Created 0.0s  
✓ Container nginx-nodejs-redis-web1-1 Started 0.1s  
✓ Container nginx-nodejs-redis-redis-1 Started 0.1s  
✓ Container nginx-nodejs-redis-web2-1 Started 0.1s  
✓ Container nginx-nodejs-redis-nginx-1 Started
```

If you look at the Docker Desktop Dashboard, you can see the containers and dive deeper into their configuration.

A screenshot of the Docker Desktop Dashboard showing the containers of the application stack deployed using Docker Compose

Alternatively, you can use the Docker Desktop Dashboard to remove the containers by selecting the application stack and selecting the Delete button.

A screenshot of Docker Desktop Dashboard that shows how to remove the containers that you deployed using Docker Compose

In this guide, you learned how easy it is to use Docker Compose to start and stop a multi-container application compared to docker run which is error-prone and difficult to manage.

Parte 4: Programacion basica

Tutorial de Python

Documentacion Oficial : <https://docs.python.org/es/3.13/tutorial/index.html>

4.1 The Python Tutorial

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python web site, <https://www.python.org/>, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.


The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth.

4.3 An Informal Introduction to Python

In the following examples, input and output are distinguished by the presence or absence of prompts (`>>>` and `...`): to repeat the example, you must type everything after the prompt, when the prompt appears; lines that do not begin with a prompt are output from the interpreter. Note that a secondary prompt on a line by itself in an example means you must type a blank line; this is used to end a multi-line command.

You can toggle the display of prompts and output by clicking on  in the upper-right corner of an example box. If you hide the prompts and output for an example, then you can easily copy and paste the input lines into your interpreter.

Many of the examples in this manual, even those entered at the interactive prompt, include comments. Comments in Python start with the hash character, `#`, and extend to the end of the physical line. A comment may appear at the start of a line or following whitespace or code, but not within a string literal. A hash character within a string literal is just a hash character. Since comments are to clarify code and are not interpreted by Python, they may be omitted when typing in examples.

Some examples:

```
# this is the first comment
spam = 1 # and this is the second comment
        # ... and now a third!
text = "# This is not a comment because it's inside quotes."
```

4.3.1. Using Python as a Calculator

Let's try some simple Python commands. Start the interpreter and wait for the primary prompt, `>>>`. (It shouldn't take long.)

4.3.1.1. Numbers

The interpreter acts as a simple calculator: you can type an expression at it and it will write the value. Expression syntax is straightforward: the operators `+`, `-`, and `*` can be used to perform arithmetic; parentheses `()` can be used for grouping. For example:

```
>>>2 + 2
4
>>>50 - 5*6
20
>>>(50 - 5*6) / 4
5.0
>>>8 / 5 # division always returns a floating-point number
1.6
```

The integer numbers (e.g. `2`, `4`, `20`) have type `int`, the ones with a fractional part (e.g. `5.0`, `1.6`) have type `float`. We will see more about numeric types later in the tutorial.

Division `()` always returns a float. To do floor division and get an integer result you can use the `/` operator; to calculate the remainder you can use `:`:

```
>>>17 / 3 # classic division returns a float
5.666666666666667
>>>
>>>17 // 3 # floor division discards the fractional part
5
>>>17 % 3 # the % operator returns the remainder of the division
2
>>>5 * 3 + 2 # floored quotient * divisor + remainder
17
```

With Python, it is possible to use the `**` operator to calculate powers [1]:

```
>>>5 ** 2 # 5 squared
25
>>>2 ** 7 # 2 to the power of 7
128
```

The equal sign `()` is used to assign a value to a variable. Afterwards, no result is displayed before the next interactive prompt:

```
>>>width = 20
```



```
>>>height = 5 * 9
>>>width * height
900
```

If a variable is not “defined” (assigned a value), trying to use it will give you an error:

```
>>>n # try to access an undefined variable
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'n' is not defined
```

There is full support for floating point; operators with mixed type operands convert the integer operand to floating point:

```
>>>4 * 3.75 - 1
14.0
```

In interactive mode, the last printed expression is assigned to the variable `_`. This means that when you are using Python as a desk calculator, it is somewhat easier to continue calculations, for example:

```
>>>tax = 12.5 / 100
>>>price = 100.50
>>>price * tax
12.5625
>>>price + _
113.0625
>>>round(_, 2)
113.06
```

This variable should be treated as read-only by the user. Don't explicitly assign a value to it — you would create an independent local variable with the same name masking the built-in variable with its magic behavior.

In addition to int and float, Python supports other types of numbers, such as Decimal and Fraction. Python also has built-in support for complex numbers, and uses the `or` suffix to indicate the imaginary part (e.g. `1j`).

4.3.1.2. Text

Python can manipulate text (represented by type `str`, so-called “strings”) as well as numbers. This includes characters “!”, words “ t”, names “ ”, sentences “ ”, etc. “ ”. They can be enclosed in single quotes () or double quotes () with the same result [2].

```
>>>'spam eggs' # single quotes
'spam eggs'
>>>"Paris rabbit got your back :)! Yay!" # double quotes
'Paris rabbit got your back :)! Yay!'
>>>'1975' # digits and numerals enclosed in quotes are also strings
'1975'
```

To quote a quote, we need to “escape” it, by preceding it with `\"`. Alternatively, we can use the other type of quotation marks:

```
>>>'doesn't' # use \' to escape the single quote...
"doesn't"
>>>"doesn't" # ...or use double quotes instead
"doesn't"
>>>'"Yes," they said.'
'"Yes," they said.'
>>>\"Yes,\" they said."
'"Yes,\" they said.'
>>>\"Isn't,\" they said.'
'"Isn't,\" they said.'
```

In the Python shell, the string definition and output string can look different. The `print()` function produces a more readable output, by omitting the enclosing quotes and by printing escaped and special characters:

```
>>>s = 'First line.\nSecond line.' # \n means newline
>>>s # without print(), special characters are included in the string
'First line.\nSecond line.'
>>>print(s) # with print(), special characters are interpreted, so \n produces new line
First line.
Second line.
```

If you don't want characters prefaced by `\"` to be interpreted as special characters, you can use *raw strings* by adding an `r` before the first quote:

```
>>>print('C:\some\name') # here \n means newline!
C:\some
ame
>>>print(r'C:\some\name') # note the r before the quote
C:\some\name
```

There is one subtle aspect to raw strings: a raw string may not end in an odd number of `\"` characters; see the FAQ entry for more information and workarounds.

String literals can span multiple lines. One way is using triple-quotes: `\"\"\"` or `'''`. End of lines are automatically included in the string, but it's possible to prevent this by adding a `\"` at the end of the line. In the following example, the initial newline is not included:

```
>>>print("""\
...Usage: thingy [OPTIONS]
...  -h            Display this usage message
...  -H hostname  Hostname to connect to
... """)
Usage: thingy [OPTIONS]
  -h            Display this usage message
  -H hostname  Hostname to connect to

>>>
```

Strings can be concatenated (glued together) with the `+` operator, and repeated with `*`:

```
>>># 3 times 'un', followed by 'ium'
>>>3 * 'un' + 'ium'
'unununium'
```

Two or more *string literals* (i.e. the ones enclosed between quotes) next to each other are automatically concatenated.

```
>>>'Py' 'thon'
'Python'
```

This feature is particularly useful when you want to break long strings:

```
>>>text = ('Put several strings within parentheses '
...       'to have them joined together.')
>>>text
'Put several strings within parentheses to have them joined together.'
```

This only works with two literals though, not with variables or expressions:

```
>>>prefix = 'Py'
>>>prefix 'thon' # can't concatenate a variable and a string literal
File "<stdin>", line 1
  prefix 'thon'
    ^^^^^^^
SyntaxError: invalid syntax
>>>('un' * 3) 'ium'
File "<stdin>", line 1
  ('un' * 3) 'ium'
    ^^^^^^^
SyntaxError: invalid syntax
```

If you want to concatenate variables or a variable and a literal, use `+`:

```
>>>prefix + 'thon'
'Python'
```

Strings can be *indexed* (subscripted), with the first character having index 0. There is no separate character type; a character is simply a string of size one:

```
>>>word = 'Python'
>>>word[0] # character in position 0
'P'
>>>word[5] # character in position 5
'n'
```

Indices may also be negative numbers, to start counting from the right:

```
>>>word[-1] # last character
'n'
>>>word[-2] # second-last character
'o'
```

```
>>>word[-6]
'P'
```

Note that since -0 is the same as 0, negative indices start from -1.

In addition to indexing, *slicing* is also supported. While indexing is used to obtain individual characters, *slicing* allows you to obtain a substring:

```
>>>word[0:2] # characters from position 0 (included) to 2 (excluded)
'Py'
>>>word[2:5] # characters from position 2 (included) to 5 (excluded)
'tho'
```

Slice indices have useful defaults; an omitted first index defaults to zero, an omitted second index defaults to the size of the string being sliced.

```
>>>word[:2] # character from the beginning to position 2 (excluded)
'Py'
>>>word[4:] # characters from position 4 (included) to the end
'on'
>>>word[-2:] # characters from the second-last (included) to the end
'on'
```

Note how the start is always included, and the end always excluded. This makes sure that `word[:2] + word[2:]` is always equal to `word`:

```
>>>word[:2] + word[2:]
'Python'
>>>word[:4] + word[4:]
'Python'
```

One way to remember how slices work is to think of the indices as pointing *between* characters, with the left edge of the first character numbered 0. Then the right edge of the last character of a string of n characters has index n , for example:

```
+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+
0 1 2 3 4 5 6
-6 -5 -4 -3 -2 -1
```

The first row of numbers gives the position of the indices 0...6 in the string; the second row gives the corresponding negative indices. The slice from i to j consists of all characters between the edges labeled i and j , respectively.

For non-negative indices, the length of a slice is the difference of the indices, if both are within bounds. For example, the length of `word[2:4]` is 2.

Attempting to use an index that is too large will result in an error:

```
>>>word[42] # the word only has 6 characters
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
IndexError: string index out of range
```

However, out of range slice indexes are handled gracefully when used for slicing:

```
>>>word[4:42]
'on'
>>>word[42:]
''
```

Python strings cannot be changed — they are immutable. Therefore, assigning to an indexed position in the string results in an error:

```
>>>word[0] = 'J'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>>word[2:] = 'py'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

If you need a different string, you should create a new one:

```
>>>'J' + word[1:]
'Jython'
>>>word[:2] + 'py'
'Pypy'
```

The built-in function `len()` returns the length of a string:

```
>>>
>>>s = 'supercalifragilisticexpialidocious'
>>>len(s)
34
```

4.3.1.3. Lists

Python knows a number of *compound* data types, used to group together other values. The most versatile is the *list*, which can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

```
>>>squares = [1, 4, 9, 16, 25]
>>>squares
[1, 4, 9, 16, 25]
```

Like strings (and all other built-in sequence types), lists can be indexed and sliced:

```
>>>squares[0] # indexing returns the item
1
>>>squares[-1]
25
```

```
>>>squares[-3:] # slicing returns a new list
[9, 16, 25]
```

Lists also support operations like concatenation:

```
>>>squares + [36, 49, 64, 81, 100]
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Unlike strings, which are immutable, lists are a mutable type, i.e. it is possible to change their content:

```
>>>cubes = [1, 8, 27, 65, 125] # something's wrong here
>>>4 ** 3 # the cube of 4 is 64, not 65!
64
>>>cubes[3] = 64 # replace the wrong value
>>>cubes
[1, 8, 27, 64, 125]
```

You can also add new items at the end of the list, by using the `list.append()` *method* (we will see more about methods later):

```
>>>cubes.append(216) # add the cube of 6
>>>cubes.append(7 ** 3) # and the cube of 7
>>>cubes
[1, 8, 27, 64, 125, 216, 343]
```

Simple assignment in Python never copies data. When you assign a list to a variable, the variable refers to the *existing list*. Any changes you make to the list through one variable will be seen through all other variables that refer to it.:

```
>>>rgb = ["Red", "Green", "Blue"]
>>>rgba = rgb
>>>id(rgb) == id(rgba) # they reference the same object
True
>>>rgba.append("Alpha")
>>>rgb
["Red", "Green", "Blue", "Alpha"]
```

All slice operations return a new list containing the requested elements. This means that the following slice returns a shallow copy of the list:

```
>>>correct_rgba = rgba[:]
>>>correct_rgba[-1] = "Alpha"
>>>correct_rgba
["Red", "Green", "Blue", "Alpha"]
>>>rgba
["Red", "Green", "Blue", "Alpha"]
```

Assignment to slices is also possible, and this can even change the size of the list or clear it entirely:

```
>>>letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
```

```

>>>letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>># replace some values
>>>letters[2:5] = ['C', 'D', 'E']
>>>letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
>>># now remove them
>>>letters[2:5] = []
>>>letters
['a', 'b', 'f', 'g']
>>># clear the list by replacing all the elements with an empty list
>>>letters[:] = []
>>>letters
[]

```

The built-in function `len()` also applies to lists:

```

>>>letters = ['a', 'b', 'c', 'd']
>>>len(letters)
4

```

It is possible to nest lists (create lists containing other lists), for example:

```

>>>a = ['a', 'b', 'c']
>>>n = [1, 2, 3]
>>>x = [a, n]
>>>x
[['a', 'b', 'c'], [1, 2, 3]]
>>>x[0]
['a', 'b', 'c']
>>>x[0][1]
'b'

```

4.3.2. First Steps Towards Programming

Of course, we can use Python for more complicated tasks than adding two and two together. For instance, we can write an initial sub-sequence of the Fibonacci series as follows:

```

>>># Fibonacci series:
>>># the sum of two elements defines the next
>>>a, b = 0, 1
>>> while a < 10:
...   print(a)
...   a, b = b, a+b
...
0
1
1
2
3
5
8

```

This example introduces several new features.

The first line contains a *multiple assignment*: the variables `a` and `b` simultaneously get the new values 0 and 1. On the last line this is used again, demonstrating that the expressions on the right-hand side are all evaluated first before any of the assignments take place. The right-hand side expressions are evaluated from the left to the right.

The while loop executes as long as the condition (here: `a < 1000`) remains true. In Python, like in C, any non-zero integer value is true; zero is false. The condition may also be a string or list value, in fact any sequence; anything with a non-zero length is true, empty sequences are false. The test used in the example is a simple comparison. The standard comparison operators are written the same as in C: `<` (less than), `>` (greater than), `==` (equal to), `<=` (less than or equal to), `>=` (greater than or equal to) and `!=` (not equal to).

The *body* of the loop is *indented*: indentation is Python's way of grouping statements. At the interactive prompt, you have to type a tab or space(s) for each indented line. In practice you will prepare more complicated input for Python with a text editor; all decent text editors have an auto-indent facility. When a compound statement is entered interactively, it must be followed by a blank line to indicate completion (since the parser cannot guess when you have typed the last line). Note that each line within a basic block must be indented by the same amount.

The `print()` function writes the value of the argument(s) it is given. It differs from just writing the expression you want to write (as we did earlier in the calculator examples) in the way it handles multiple arguments, floating-point quantities, and strings. Strings are printed without quotes, and a space is inserted between items, so you can format things nicely, like this:

```
>>>i = 256*256
>>>print('The value of i is', i)
The value of i is 65536
```

The keyword argument `end` can be used to avoid the newline after the output, or end the output with a different string:

```
>>>a, b = 0, 1
>>> while a < 1000:
...   print(a, end=',')
...   a, b = b, a+b
...
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,
```


Parte 5: Bases de datos

Documentacion MariaDB Server

Documentacion oficial : <https://mariadb.com/kb/es/documentacion-mariadb-server/>

5.1 MariaDB Basics

Contents

- [5.1.1 Connecting to MariaDB](#)
- [5.1.2 Creating a Structure](#)
- [5.1.3 Minor Items](#)
- [5.1.4 Entering Data](#)
- [5.1.5 Retrieving Data](#)
- [5.1.6 Changing & Deleting Data](#)
- [5.1.7 Conclusion](#)

5.1.1 Connecting to MariaDB

MariaDB is a database system, a database server. To interface with the MariaDB server, you can use a client program, or you can write a program or script with one of the popular programming languages (e.g., PHP) using an API (Application Programming Interface) to interface with the MariaDB server. For the purposes of this article, we will focus on using the default client that comes with MariaDB called `mariadb`. With this client, you can either enter queries from the command-line, or you can switch to a terminal, that is to say, monitor mode. To start, we'll use the latter.

From the Linux command-line, you would enter the following to log in as the root user and to enter monitor mode:

```
mariadb -u root -p -h localhost
```

The `-u` option is for specifying the user name. You would replace `root` here if you want to use a different user name. This is the MariaDB user name, not the Linux user name. The password for the MariaDB user `root` will probably be different from the Linux user `root`. Incidentally, it's not a good security practice to use the `root` user unless you have a specific administrative task to perform for which only `root` has the needed privileges.

The `-p` option above instructs the `mariadb` client to prompt you for the password. If the password for the `root` user hasn't been set yet, then the password is blank and you would just hit [Enter] when prompted. The `-h` option is for specifying the host name or the IP address of the server. This would be necessary if the client is running on a different machine than the server. If you've secure-shelled into the server machine, you probably won't need to use the host option. In fact, if you're logged into Linux as `root`, you won't need the user option—the `-p` is all you'll need. Once you've entered the line above along with the password when prompted, you will be logged into MariaDB through the client. To exit, type `quit` or `exit` and press [Enter].

5.1.2 Creating a Structure

In order to be able to add and to manipulate data, you first have to create a database structure. Creating a database is simple. You would enter something like the following from within the [mariadb client](#):

```
CREATE DATABASE bookstore;
USE bookstore;
```

This very minimal, first SQL statement will create a sub-directory called bookstore on the Linux filesystem in the directory which holds your MariaDB data files. It won't create any data, obviously. It'll just set up a place to add tables, which will in turn hold data. The second SQL statement above will set this new database as the default database. It will remain your default until you change it to a different one or until you log out of MariaDB.

The next step is to begin creating tables. This is only a little more complicated. To create a simple table that will hold basic data on books, we could enter something like the following:

```
CREATE TABLE books (
isbn CHAR(20) PRIMARY KEY,
title VARCHAR(50),
author_id INT,
publisher_id INT,
year_pub CHAR(4),
description TEXT );
```

This SQL statement creates the table books with six fields, or rather columns. The first column (isbn) is an identification number for each row—this name relates to the unique identifier used in the book publishing business. It has a fixed-width character type of 20 characters. It will be the primary key column on which data will be indexed. The column data type for the book title is a variable width character column of fifty characters at most. The third and fourth columns will be used for identification numbers for the author and the publisher. They are integer data types. The fifth column is used for the publication year of each book. The last column is for entering a description of each book. It's a [TEXT](#) data type, which means that it's a variable width column and it can hold up to 65535 bytes of data for each row. There are several other data types that may be used for columns, but this gives you a good sampling.

To see how the table we created looks, enter the following SQL statement:

```
DESCRIBE books;
```

Field	Type	Null	Key	Default	Extra
isbn	char(20)	NO	PRI	NULL	
title	varchar(50)	YES		NULL	
author_id	int(11)	YES		NULL	
publisher_id	int(11)	YES		NULL	
year_pub	char(4)	YES		NULL	
description	text	YES		NULL	

To change the settings of a table, you can use the [ALTER TABLE](#) statement. I'll cover that statement in another article. To delete a table completely (including its data), you can use the [DROP TABLE](#) statement, followed by the table name. Be careful with this statement since it's not reversible.

The next table we'll create for our examples is the authors table to hold author information. This table will save us from having to enter the author's name and other related data for each book written by each author. It also helps to ensure consistency of data: there's less chance of inadvertent spelling deviations.

```
CREATE TABLE authors
(author_id INT AUTO_INCREMENT PRIMARY KEY,
name_last VARCHAR(50),
name_first VARCHAR(50),
country VARCHAR(50) );
```

We'll join this table to the books table as needed. For instance, we would use it when we want a list of books along with their corresponding authors' names. For a real bookstore's database, both of these tables would probably have more columns. There would also be several more tables. For the examples that follow, these two tables as they are will be enough.

5.1.3 Minor Items

Before moving on to the next step of adding data to the tables, let me point out a few minor items that I've omitted mentioning. SQL statements end with a semi-colon (or a \G). You can spread an SQL statement over multiple lines. However, it won't be passed to the server by the client until you terminate it with a semi-colon and hit [Enter]. To cancel an SQL statement once you've started typing it, enter \c and press [Enter].

As a basic convention, reserved words are printed in all capital letters. This isn't necessary, though. MariaDB is case-insensitive with regards to reserved words. Database and table names, however, are case-sensitive on Linux. This is because they reference the related directories and files on the filesystem. Column names aren't case sensitive since they're not affected by the filesystem, per se. As another convention, we use lower-case letters for structural names (e.g., table names). It's a matter of preference for deciding on names.

5.1.4 Entering Data

The primary method for entering data into a table is to use the [INSERT](#) statement. As an example, let's enter some information about an author into the authors table. We'll do that like so:

```
INSERT INTO authors
(name_last, name_first, country)
VALUES('Kafka', 'Franz', 'Czech Republic');
```

This will add the name and country of the author Franz Kafka to the authors table. We don't need to give a value for the author_id since that column was created with the [AUTO_INCREMENT](#) option. MariaDB will automatically assign an identification number. You can manually assign one, especially

if you want to start the count at a higher number than 1 (e.g., 1000). Since we are not providing data for all of the columns in the table, we have to list the columns for which we are giving data and in the order that the data is given in the set following the VALUES keyword. This means that we could give the data in a different order.

For an actual database, we would probably enter data for many authors. We'll assume that we've done that and move on to entering data for some books. Below is an entry for one of Kafka's books:

```
INSERT INTO books
(title, author_id, isbn, year_pub)
VALUES('The Castle', '1', '0805211063', '1998');
```

This adds a record for Kafka's book, *The Castle*. Notice that we mixed up the order of the columns, but it still works because both sets agree. We indicate that the author is Kafka by giving a value of 1 for the author_id. This is the value that was assigned by MariaDB when we entered the row for Kafka earlier. Let's enter a few more books for Kafka, but by a different method:

```
INSERT INTO books
(title, author_id, isbn, year_pub)
VALUES('The Trial', '1', '0805210407', '1995'),
('The Metamorphosis', '1', '0553213695', '1995'),
('America', '1', '0805210644', '1995');
```

In this example, we've added three books in one statement. This allows us to give the list of column names once. We also give the keyword VALUES only once, followed by a separate set of values for each book, each contained in parentheses and separated by commas. This cuts down on typing and speeds up the process. Either method is fine and both have their advantages. To be able to continue with our examples, let's assume that data on thousands of books has been entered. With that behind us, let's look at how to retrieve data from tables.

5.1.5 Retrieving Data

The primary method of retrieving data from tables is to use a SELECT statement. There are many options available with the SELECT statement, but you can start simply. As an example, let's retrieve a list of book titles from the books table:

```
SELECT title
FROM books;
```

This will display all of the rows of books in the table. If the table has thousands of rows, MariaDB will display thousands. To limit the number of rows retrieved, we could add a LIMIT clause to the SELECT statement like so:

```
SELECT title
FROM books
LIMIT 5;
```

This will limit the number of rows displayed to five. To be able to list the author's name for each book along with the title, you will have to join the books table with the authors table. To do this, we can use the **JOIN** clause like so:

```
SELECT title, name_last
FROM books
JOIN authors USING (author_id);
```

Notice that the primary table from which we're drawing data is given in the **FROM** clause. The table to which we're joining is given in the **JOIN** clause along with the commonly named column (i.e., `author_id`) that we're using for the join.

To retrieve the titles of only books written by Kafka based on his name (not the `author_id`), we would use the **WHERE** clause with the **SELECT** statement. This would be entered like the following:

```
SELECT title AS 'Kafka Books'
FROM books
JOIN authors USING (author_id)
WHERE name_last = 'Kafka';
+-----+
| Kafka Books |
+-----+
| The Castle  |
| The Trial   |
| The Metamorphosis |
| America    |
+-----+
```

This statement will list the titles of Kafka books stored in the database. Notice that I've added the **AS** parameter next to the column name `title` to change the column heading in the results set to `Kafka Books`. This is known as an alias. Looking at the results here, we can see that the title for one of Kafka's books is incorrect. His book *Amerika* is spelled above with a "c" in the table instead of a "k". This leads to the next section on changing data.

5.1.6 Changing & Deleting Data

In order to change existing data, a common method is to use the **UPDATE** statement. When changing data, though, we need to be sure that we change the correct rows. In our example, there could be another book with the title *America* written by a different author. Since the key column `isbn` has only unique numbers and we know the ISBN number for the book that we want to change, we can use it to specify the row.

```
UPDATE books
SET title = 'Amerika' WHERE isbn = '0805210644';
```

This will change the value of the title column for the row specified. We could change the value of other columns for the same row by giving the `column = value` for each, separated by commas.

If we want to delete a row of data, we can use the **DELETE** statement. For instance, suppose that our fictitious bookstore has decided no longer to carry books by John Grisham. By first running a **SELECT** statement, we determine the identification number for the author to be 2034. Using this author identification number, we could enter the following:

```
DELETE FROM books
WHERE author_id = '2034';
```

This statement will delete all rows from the table `books` for the `author_id` given. To do a clean job of it, we'll have to do the same for the `authors` table. We would just replace the table name in the statement above; everything else would be the same.

5.1.7 Conclusion

This is a very basic primer for using MariaDB. Hopefully, it gives you the idea of how to get started with MariaDB. Each of the SQL statements mentioned here have several more options and clauses each. We will cover these statements and others in greater detail in other articles. For now, though, you can learn more about them from experimenting and by further reading of the Knowledge Base online documentation.

5.2 Configuring MariaDB for Remote Client Access

Contents

- [5.2.1 Finding the Defaults File](#)
- [5.2.2 Editing the Defaults File](#)
- [5.2.3 Granting User Connections From Remote Hosts](#)
- [5.2.4 Port 3306 is Configured in Firewall](#)
- [5.2.5 Caveats](#)

Some MariaDB packages bind MariaDB to 127.0.0.1 (the loopback IP address) by default as a security measure using the `bind-address` configuration directive. Old MySQL packages sometimes disabled TCP/IP networking altogether using the `skip-networking` directive. Before going in to how to configure these, let's explain what each of them actually does:

- `skip-networking` is fairly simple. It just tells MariaDB to run without any of the TCP/IP networking options.
- `bind-address` requires a little bit of background information. A given server usually has at least two networking interfaces (although this is not required) and can easily have more. The two most common are a *Loopback* network device and a physical *Network Interface Card* (NIC) which allows you to communicate with the network. MariaDB is bound to the loopback interface by default because it makes it impossible to connect to the TCP port on the server from a remote host (the `bind-address` must refer to a local IP address, or you will receive a fatal error and MariaDB will not start). This of course is not desirable if you want to use the TCP port from a remote host, so you must remove this `bind-address` directive or replace it either 0.0.0.0 to listen on all interfaces, or the address of a specific public interface.

MariaDB starting with 10.11

Multiple comma-separated addresses can now be given to `bind_address` to allow the server to listen on more than one specific interface while not listening on others.

If [bind-address](#) is bound to 127.0.0.1 (localhost), one can't connect to the MariaDB server from other hosts or from the same host over TCP/IP on a different interface than the loopback (127.0.0.1). This for example will not work (connecting with a hostname that points to a local IP of the host):

```
(/my/maria-10.11) ./client/mariadb --host=myhost --protocol=tcp --port=3306 testERROR 2002 (HY000): Can't connect to MySQL server on 'myhost' (115)(/my/maria-10.11) telnet myhost 3306Trying 192.168.0.11...telnet: connect to address 192.168.0.11: Connection refused
```

Using 'localhost' works when binding with `bind_address`:

```
(my/maria-10.11) ./client/mariadb --host=localhost --protocol=tcp --port=3306 testReading table information for completion of table and column namesYou can turn off this feature to get a quicker startup with -A Welcome to the MariaDB monitor. Commands end with ; or \g...
```

5.2.1 Finding the Defaults File

To enable MariaDB to listen to remote connections, you need to edit your defaults file. See [Configuring MariaDB with my.cnf](#) for more detail.

Common locations for defaults files:

* /etc/my.cnf	(*nix/BSD)	* \$MYSQL_HOME/my.cnf	(*nix/BSD)	*Most
Notably /etc/mysql/my.cnf	* SYSCONFDIR/my.cnf		(*nix/BSD)	*
DATADIR\my.ini		(Windows)		

You can see which defaults files are read and in which order by executing:

```
shell> mariadb --help --verbosemariadb Ver 10.11.5-MariaDB for linux-systemd on x86_64 (MariaDB Server)Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others. Starts the MariaDB database server. Usage: ./mariadb [OPTIONS] Default options are read from the following files in the given order:/etc/my.cnf /etc/mysql/my.cnf ~/.my.cnf
```

The last line shows which defaults files are read.

5.2.2 Editing the Defaults File

Once you have located the defaults file, use a text editor to open the file and try to find lines like this under the `[mysqld]` section:

```
[mysqld] ... skip-networking ... bind-address = <some ip-address> ...
```

(The lines may not be in this order, and the order doesn't matter.)

If you are able to locate these lines, make sure they are both commented out (prefaced with hash (#) characters), so that they look like this:

```
[mysqld] ... #skip-networking ... #bind-address = <some ip-address> ...
```

(Again, the order of these lines don't matter)

Alternatively, just add the following lines **at the end** of your `.my.cnf` (notice that the file name starts with a dot) file in your home directory or alternative **last** in your `/etc/my.cnf` file.


```
[mysqld]skip-networking=0skip-bind-address
```

This works as one can have any number of [mysqld] sections.

Save the file and restart the mariadb daemon or service (see [Starting and Stopping MariaDB](#)).

You can check the options mariadb is using by executing:

```
shell> ./sql/mariadb --print-defaults./sql/mariadb would have been started with the following arguments:--bind-address=127.0.0.1 --innodb_file_per_table=ON --server-id=1 --skip-bind-address ...
```

It doesn't matter if you have the original --bind-address left as the later --skip-bind-address will overwrite it.

5.2.3 Granting User Connections From Remote Hosts

Now that your MariaDB server installation is setup to accept connections from remote hosts, we have to add a user that is allowed to connect from something other than 'localhost' (Users in MariaDB are defined as 'user'@'host', so 'chadmaynard'@'localhost' and 'chadmaynard'@'1.1.1.1' (or 'chadmaynard'@'server.domain.local') are different users that can have completely different permissions and/or passwords.

To create a new user:

- log into the [mariadb command line client](#) (or your favorite graphical client if you wish)

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.Your MariaDB connection id is 36Server version: 5.5.28-MariaDB-mariadb1~lucid mariadb.org binary distribution Copyright (c) 2000, 2012, Oracle, Monty Program Ab and others. Type 'help;' or '\h' for help. Type '\c' to clear the current input statement. MariaDB [(none)]>
```

- if you are interested in viewing any existing remote users, issue the following SQL statement on the [mysql.user](#) table:

```
SELECT User, Host FROM mysql.user WHERE Host <> 'localhost';+-----+-----+ | User | Host | +-----+-----+ | daniel | %    | | root | 127.0.0.1 | | root | ::1    | | root | gandalf | +-----+-----+4 rows in set (0.00 sec)
```

(If you have a fresh install, it is normal for no rows to be returned)

Now you have some decisions to make. At the heart of every grant statement you have these things:

- list of allowed privileges
- what database/tables these privileges apply to
- username
- host this user can connect from
- and optionally a password

It is common for people to want to create a "root" user that can connect from anywhere, so as an example, we'll do just that, but to improve on it we'll create a root user that can connect from anywhere on my local area network (LAN), which has addresses in the subnet 192.168.100.0/24.

This is an improvement because opening a MariaDB server up to the Internet and granting access to all hosts is bad practice.

```
GRANT ALL PRIVILEGES ON *.* TO 'root'@'192.168.100.%' IDENTIFIED BY 'my-new-password' WITH GRANT OPTION;
```

(% is a wildcard)

For more information about how to use GRANT, please see the [GRANT](#) page.

At this point we have accomplished our goal and we have a user 'root' that can connect from anywhere on the 192.168.100.0/24 LAN.

5.2.4 Port 3306 is Configured in Firewall

One more point to consider whether the firewall is configured to allow incoming request from remote clients:

On RHEL and CentOS 7, it may be necessary to configure the firewall to allow TCP access to MariaDB from remote hosts. To do so, execute both of these commands:

```
firewall-cmd --add-port=3306/tcp firewall-cmd --permanent --add-port=3306/tcp
```

5.2.5 Caveats

- If your system is running a software firewall (or behind a hardware firewall or NAT) you must allow connections destined to TCP port that MariaDB runs on (by default and almost always 3306).
- To undo this change and not allow remote access anymore, simply remove the `skip-bind-address` line or uncomment the `bind-address` line in your defaults file. The end result should be that you should have in the output from `./sql/mariadb --print-defaults` the option `--bind-address=127.0.0.1` and no `--skip-bind-address`.