USERS

E-BOOK

Técnico en

REDES &SEGURIDAD

5

ASTERISK

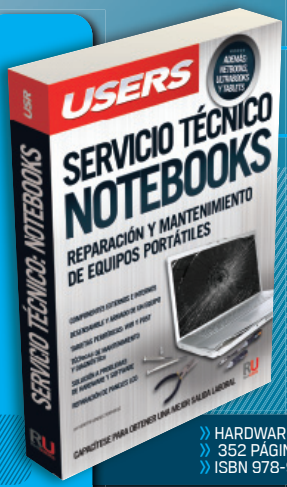
SERVIDOR DE TELEFONÍA IP CON TODAS LAS FUNCIONALIDADES DE UNA CENTRAL TELEFÓNICA

- INSTALACIÓN Y CONFIGURACIÓN
- CARACTERÍSTICAS AVANZADAS
- IMPLEMENTACIÓN DE UNA CENTRAL IP
- GESTIÓN DEL SISTEMA

CONÉCTESE
CON LOS MEJORES LIBROS DE COMPUTACIÓN

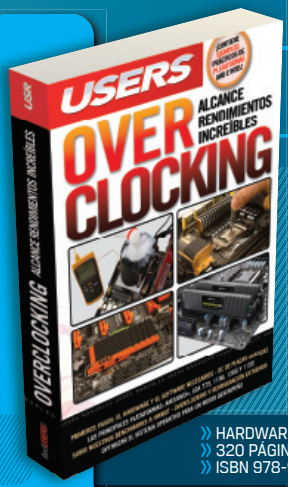
LLEGAMOS A TODO EL MUNDO
VÍA OCA * Y DHL **
usershop.redusers.com
usershop@redusers.com
+54 (011) 4110-8700

* SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // ** VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA



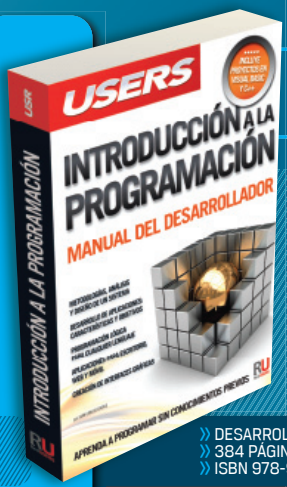
CAPACÍTESE PARA OBTENER UNA MEJOR SALIDA LABORAL

» HARDWARE / MOBILE
» 352 PÁGINAS
» ISBN 978-987-1857-68-5



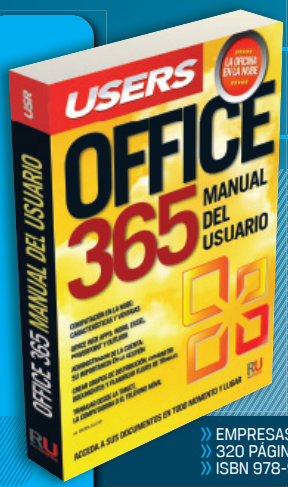
ALCANCE RENDIMIENTOS INCREÍBLES EN SU PC

» HARDWARE
» 320 PÁGINAS
» ISBN 978-987-1857-30-2



APRENDA A PROGRAMAR SIN CONOCIMIENTOS PREVIOS

» DESARROLLO
» 384 PÁGINAS
» ISBN 978-987-1857-69-2



ACCEDA A SUS DOCUMENTOS EN TODO MOMENTO Y LUGAR.

» EMPRESAS / INTERNET
» 320 PÁGINAS
» ISBN 978-987-1857-65-4

USERS

Técnico en REDES & SEGURIDAD

5

ASTERISK

TÍTULO: Asterisk
COLECCIÓN: Pocket Users

Primera edición realizada en abril de MMXIII.

EISBN 978-987-1949-03-8

Arias, Raúl
Asterisk / Raúl Arias y Ignacio Matías Arias García. - a ed. - Buenos Aires : Fox Andina, 2013.
E-Book. - (Pocket users; 30)
ISBN 978-987-1949-03-8
1. Informática. I. Arias García, Ignacio Matías II. Título
CDD 005.3

VISITE NUESTRA WEB

EN NUESTRO SITIO PUEDE OBTENER, DE FORMA GRATUITA, UN CAPÍTULO DE CADA UNO DE LOS LIBROS EN VERSIÓN PDF Y PREVIEW DIGITAL. ADEMÁS, PODRÁ ACCEDER AL SUMARIO COMPLETO, LIBRO DE UN VISTAZO, IMÁGENES AMPLIADAS DE TAPA Y CONTRATAPA Y MATERIAL ADICIONAL.

RedUSERS
COMUNIDAD DE TECNOLOGIA

redusers.com

Nuestros libros incluyen guías visuales, explicaciones paso a paso, recuadros complementarios, ejercicios, glosarios, atajos de teclado y todos los elementos necesarios para asegurar un aprendizaje exitoso y estar conectado con el mundo de la tecnología.

LLEGAMOS A TODO EL MUNDO VÍA OCA* Y DHL**

* SÓLO VÁLIDO EN LA REPÚBLICA ARGENTINA // ** VÁLIDO EN TODO EL MUNDO EXCEPTO ARGENTINA

usershop.redusers.com usershop@redusers.com + 54 (011) 4110-8700

Los **autores**

Raúl Daniel Arias

Es ingeniero en Electrónica por la UTN FRBA. Tiene un posgrado en Telecomunicaciones por la UBA y un MBA en Gestión de Empresas de Telecomunicaciones por el Politécnico de Madrid. Se desempeña como especialista en telecomunicaciones, y como docente universitario y de la escuela media en la Ciudad de Buenos Aires. Ha sido docente en el ITBA, la UTN FRBA y el Colegio Militar de la Nación.

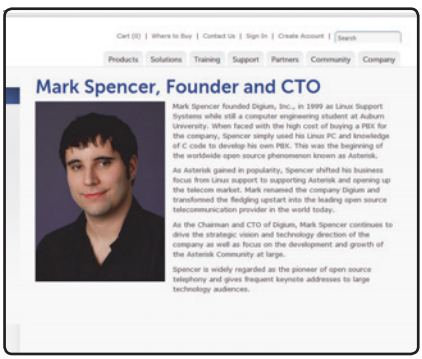
Ignacio Matías Arias García

Es técnico en Electrónica y estudiante de Ingeniería en la Universidad de Buenos Aires. Se interesa en el tema de las telecomunicaciones, Linux y Asterisk. Su dedicación y ganas de aprender lo llevaron a participar en esta obra generando material, probando y sugiriendo modificaciones.

Prólogo **al contenido**

Una de las características del ser humano es que necesita comunicarse, casi tanto como alimentarse. Desde tiempos inmemoriales, se ha dedicado a la invención de sistemas para llevar a cabo este mandato de la especie. El desarrollo de estos sistemas, por lo general, ha seguido un camino propietario, de altos costos y extrema complejidad, hasta la llegada de Asterisk. Este sistema resulta un punto de inflexión en la historia de esta tecnología. Si bien su complejidad es importante, no lo es tanto frente a las implementaciones convencionales. Además, el hecho de ser un software libre lo vuelve muy atractivo, en especial, para las empresas pequeñas y medianas. En este libro exploraremos algunos conceptos básicos de este producto, con la intención de que sea una guía de acceso al mundo de Asterisk. El lector deberá contar con algunos conocimientos de telecomunicaciones y alguna práctica con el sistema operativo Linux para aprovecharlo al máximo. Esperamos que esta obra sea una puerta de entrada al mundo de Asterisk.

Contenido **del libro**



Capítulo 1

Introducción e instalación de Asterisk

Conoceremos qué es Asterisk y los requisitos de hardware para su instalación en Ubuntu.

RedUSERS.com

Introducción

Asterisk es una aplicación que permite la implementación de centrales telefónicas. Su función básica es convertir un hardware en una plataforma de comunicaciones de voz realmente poderosa. Se trata de un software muy flexible, que puede instalarse en casi cualquier sistema **Linux** y en algunos otros sistemas operativos, como **FreeBSD**.

Su potencia dependerá de las características del hardware de cómputo empleado. Por eso, podremos armar una central telefónica adecuada a cualquier necesidad aumentando la inversión en el hardware, según sea requerido por el crecimiento, y manteniendo siempre la misma plataforma básica.

Este software es de libre acceso, y la plataforma que con él podemos armar tendrá todas las características de los productos comerciales, que poseen un costo a veces prohibitivo para algunas empresas y son definitivamente muy altos para una aplicación hogareña.

Para que quede claro: con Asterisk podremos armar e instalar una central telefónica para nuestra casa, pyme, colegio, empresa, etc. Cada una de estas implementaciones se diferenciará en cuanto al hardware, pero todas reutilizarán la experiencia y el trabajo realizado en cualquiera de ellas.

El proyecto Asterisk se inició en 1999, cuando Mark Spencer, su creador, publicó el

Figura 1. **Podemos conocer más sobre Mark Spencer en el sitio** www.digium.com/en/company/bios.php?uid=1.

código inicial bajo licencia de código abierto GPL. Desde entonces, ha sido perfeccionado y testeado por una comunidad siempre en aumento, que también le ha sumado nuevas características. En este momento, Asterisk es mantenido gracias a los esfuerzos combinados de esta comunidad y la empresa **Digium**, fundada por Mark Spencer para dar soluciones alternativas de telefonía.

Dimensionar **la plataforma**

Asterisk es una aplicación que trabaja en tiempo real, o más bien, con datos cuya naturaleza es de tiempo real; con esto nos referimos a la voz o la conversación entre dos o más personas. De esta manera, los requerimientos de recursos son importantes y hacen que la competencia con otras aplicaciones no sea deseable. Si en un sistema en el que está corriendo Asterisk, tenemos que ejecutar otras aplicaciones,

relacionadas o no, estas tendrán que hacerlo con un nivel de prioridad más bajo, puesto que las necesidades de Asterisk son rigurosas en lo que a cómputo se refiere. No obstante, esto no significa que precisemos una supercomputadora para ejecutar la aplicación, con múltiples cores; sino que Asterisk debe tener el procesador disponible cuando lo requiera, y esto es algo que sucede con frecuencia. Si podemos dedicar una máquina para ejecutarlo, nos ahorraremos muchos contratiempos.

El parámetro fundamental para dimensionar un sistema Asterisk es el número de llamadas o canales simultáneos que necesitamos o esperamos tener. La cantidad de terminales/usuarios del sistema impactará en el dimensionamiento de otro factor de hardware (las placas empleadas para la conexión de troncales y dispositivos analógicos), en caso de que nuestro sistema se vincule a la red telefónica convencional.

REQUISITOS DEL SISTEMA

Para un sistema pequeño, de no más de 6 canales, una máquina con un procesador de 400 MHz y 256 MB de memoria será suficiente. Si nuestro sistema es para una empresa y el requerimiento es de más de 30 canales, precisaremos una instalación con múltiples servidores Asterisk que interactúen entre ellos, y los procesadores deberán tener varios cores, con más de 1 GB de memoria por máquina.

Las instalaciones grandes suelen desarrollarse con múltiples servidores interconectados entre sí y comunicados vía un protocolo denominado **DUNDi**, que trabaja en una arquitectura detallada en la especificación **ARA** (**Asterisk Realtime Architecture**). El diseño y la implementación de este tipo de soluciones no están dentro del alcance de este libro, y es materia de personal experimentado tanto con Asterisk como con la telefonía.

La flexibilidad de Asterisk permite obtener soluciones muy efectivas y eficientes para cualquier tipo de empresa, aun las que tienen alta tasa de crecimiento y no pueden abordar los costos de una gran central telefónica hoy, pero que la necesitarán pronto. Las soluciones basadas en Asterisk son altamente escalables y ajustadas al presupuesto del usuario.

En este punto vamos a explicitar el alcance de la presente obra. Si bien Asterisk permite armar complejas estructuras, incluso interconectadas

DIGIUM

Asterisk es utilizado en todo el mundo por pymes, grandes empresas, call centers, proveedores de comunicaciones y gobiernos. Es un software libre y de fuente abierta (open source), patrocinado por Digium (www.digium.com), la empresa de Mark Spencer.

con la red de telefonía básica, nos centraremos en un sistema destinado a la conmutación entre clientes conectados a una red de datos, que sea local de área amplia. En una ampliación posterior, trataremos la interconexión con redes de telefonía básica.

Selección del hardware

El desempeño confiable de un sistema Asterisk depende de la cuidadosa selección de los componentes de hardware, en especial, de la plataforma de cómputo. Elementos clave en esta selección son la **CPU**, la **placa madre** y la **fuente de alimentación**.

Como mencionamos, la cuestión de la potencia está relacionada con la cantidad de llamadas simultáneas que el sistema debe ser capaz de sostener. Desafortunadamente, no contamos con una tabla que estipule los valores o rangos a utilizar tipificados por nivel de desempeño requerido, puesto que este, como en muy pocas situaciones, depende de la aplicación y el uso que se le dará al sistema.

Sin embargo, podemos inferir algunas reglas de selección conociendo la manera en que Asterisk utiliza el sistema. De estas observaciones, se ha determinado una fuerte correlación entre la potencia de cómputo requerida y la utilización de características especiales, como:

- La conferencia y el número de participantes en ella.
- El uso de lógica externa a la programación interna de Asterisk.
- La interconexión con la red telefónica convencional.
- El número de canales simultáneos a tratar, debido a la carga de cómputo de la implementación de los códecs en el procesamiento digital de la señal telefónica.

Se tiene una percepción cualitativa de los efectos de estos elementos sobre el desempeño global del sistema, y las conclusiones obtenidas nos conducen a que debemos prestar especial atención a la selección de los distintos códecs, el tipo, el desempeño y la implementación de la unidad de punto flotante del procesador bajo estudio, la latencia del servicio de interrupciones y las optimizaciones del kernel que se empleará.

PROYECTOS ASTERISK

Existen numerosos proyectos en torno a Asterisk que facilitan su instalación. Si bien dan la posibilidad de armar una PBX rápidamente, no permiten un aprendizaje profundo y detallado de los procesos de instalación y de las particularidades del producto Asterisk.

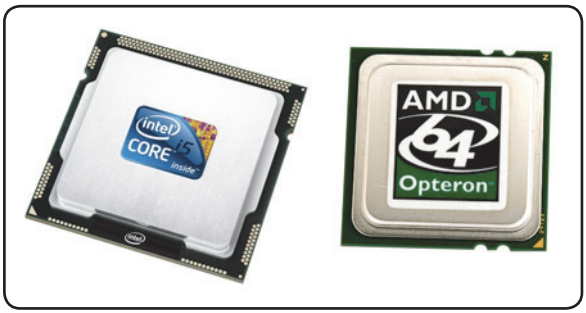


Figura 2. **Ambas marcas de procesadores sirven para estos propósitos.**

LA CPU

A los efectos prácticos, los procesadores mononúcleo actuales tienen **FPUs** que cumplen con los requisitos de Asterisk y permitirán implementaciones de hasta una docena de terminales con capacidades telefónicas. Es posible encontrar ejemplos de montajes de Asterisk en una gran variedad de dispositivos, donde siempre la cuestión pasa por la carga de llamadas simultáneas. Como ejemplo, se sostiene que con procesadores Intel de hasta 700 MHz de reloj puede implementarse un sistema pero con bajas cargas; esto es, a lo sumo, dos llamadas concurrentes. En este caso, más núcleos brindan beneficios, especialmente por las múltiples FPUs. Pero existe evidencia de que sistemas sobre múltiples servidores trabajan mejor que sobre múltiples núcleos, aunque la implementación de estos es más compleja.

LA PLACA MADRE

La selección de una **placa madre** pasa por los detalles, dado que actualmente la mayoría de ellas cumple con los requisitos básicos para montar un sistema Asterisk. Pero los detalles resultan de vital importancia, en especial, cuando deseemos interconectar el sistema con la red telefónica o con PBX convencionales. En este sentido, el elemento clave es el tipo de bus que tiene la placa. Como el PCI es el más difundido, tendremos que poner atención a los chipsets que lo implementan. Los de Intel y NVIDIA son muy recomendados. No obstante, no está de más analizar información sobre el producto que se tenga o que se desee adquirir, buscando cuestiones acerca de problemas relacionados con la latencia en el tratamiento de las interrupciones (IRQ). Asimismo, será deseable que el BIOS de la placa permita un manejo discrecional de la asignación de IRQ.

Si la placa tiene incorporado el acceso a red, podemos utilizarlo, pero siempre será mejor tener una placa separada y utilizarla, para minimizar el riesgo de que un problema que afecte al puerto de red termine por inutilizar toda la placa madre.

LA ALIMENTACIÓN DE ENERGÍA

En cualquier sistema de comunicaciones, la calidad de la alimentación eléctrica de los equipos es de fundamental importancia, y Asterisk

Figura 3. Lo recomendable es utilizar una fuente de alimentación de última generación.

no es la excepción. Las fuentes de alimentación de alta gama que se comercializan para el armado de sistemas multimedia (en general, estaciones de diseño o de juegos de alto nivel) serán las indicadas para formar parte de nuestro sistema. Esto no significa que una fuente convencional no sirva, sino que el hecho de tener una alimentación de energía limpia y de buena calidad redundará en una mejor experiencia para los usuarios del sistema y nos ahorrará mucho esfuerzo en la resolución de problemas inducidos por fallas en la alimentación.

En los sistemas empresariales de gran tamaño, es común disponer de fuentes de alimentación redundantes para asegurar la disponibilidad ante un problema en alguna de ellas. Para enfrentar el corte del suministro de energía, es conveniente disponer de una UPS. Además de su autonomía, es importante que provea el

APLICACIONES DE ASTERISK

Tanta gente trabajando en el proyecto ha contribuido a que Asterisk sea muy robusto en muchas aplicaciones de diferentes contextos. Estas van desde centrales PBX estándar, hasta la implementación de complejas soluciones para contact centers.

servicio de acondicionamiento del suministro. Estos dispositivos son más caros, pero ofrecen protección a través de un transformador de aislación que nos separa en forma segura de los ruidos de la línea de la compañía.

Como en cualquier instalación de aparatos eléctricos, disponer de un plano de tierra seguro y parejo nos devolverá la inversión y nos evitará muchos problemas. Es recomendable que la instalación eléctrica esté implementada por personal certificado. Al menos, deberíamos asegurarnos de no estar en presencia de lazos de retorno por tierra, que el circuito de alimentación sea independiente de otros (un tendido desde el panel de acceso eléctrico hasta el conector de alimentación de pared) y que solo nuestro servidor esté conectado a él.

Asterisk por dentro

Asterisk maneja los elementos que se conectan a él de la misma manera, ya sea que se trate de líneas terminales (conocidas como líneas de abonado, es el aparato telefónico del cliente) o troncales de interconexión, con otros sistemas Asterisk o PBX o la red pública.

Desde el punto de vista operativo, esto es muy útil. Este manejo desinteresado se realiza a través del recurso lógico denominado **canal**, que, conceptualmente, es independiente de las diferencias que existen entre estos tipos de conexión. Todo el tráfico que ingresa o egresa del sistema Asterisk pasa a través de un canal de ciertas características o **tipo de canal**. Habrá diferentes tipos de canales, pero Asterisk los manejará de forma similar.

La arquitectura del software se compone de **módulos** que se cargan por demanda (según las características que requiera la solución) y se combinan con el módulo núcleo de Asterisk. La carga o no de estos módulos depende del contenido de un archivo de configuración de Asterisk, **`/etc/asterisk/modules.conf`**, cuyo manejo será tratado en el Capítulo 2.

Algunos de estos módulos son los que implementan los códecs, las aplicaciones de Asterisk, el Dialplan, el registro de eventos, el registro de llamadas, etcétera. Cabe señalar que es posible iniciar Asterisk sin cargar ninguno de estos módulos, y luego, a través de comandos de carga y descarga, manejarlos en tiempo de ejecución. Esta

COMUNIDAD ASTERISK

Si deseamos explorar productos y aplicaciones de software, hardware y soluciones de comunicaciones completas para los negocios, implementados con Asterisk por los miembros de la comunidad, podemos visitar el sitio www.asteriskexchange.com.

es una herramienta muy útil cuando se implementa un sistema, se ajusta uno que está en fase de preproducción o, simplemente, se intenta diagnosticar problemas en un sistema productivo.

ESTRUCTURA DE DIRECTORIOS

A la hora de trabajar con un sistema Asterisk, ya sea en su implementación o durante la solución de algún problema, conocer la estructura de directorios que se arma en la instalación y el lugar donde se alojan los principales archivos de configuración resulta de fundamental importancia. La descripción que haremos a continuación no pretende ser exhaustiva, sino proporcionar una guía a la hora de buscar archivos relacionados.

- Los **archivos de configuración**, que emplearemos y explicaremos en los capítulos siguientes, se encuentran dentro del directorio **/etc/asterisk**, que será ampliamente utilizado al trabajar con el sistema Asterisk.
- En el directorio **/var/lib/asterisk** se alojan, entre otros, los archivos relacionados con la funcionalidad de **música en espera** y los sonidos de **señalización telefónica**. Aquí también encontraremos la base de información de Asterisk y una serie de subdirectorios cuyos nombres refieren a las funciones relacionadas. Los archivos que se encuentran en estos subdirectorios suelen llamarse **archivos de recursos**.
- En el directorio **/var/spool/asterisk**, entre otros archivos y subdirectorios, están los **mensajes de voz**, dentro de **/var/spool/asterisk/voicemail**; y los archivos que permiten **generar una llamada** están dentro de **/var/spool/asterisk/outgoing**. También encontraremos archivos temporarios en **/var/spool/asterisk/temp**.
- El directorio **/var/log/asterisk** se emplea para guardar toda clase de **eventos** del sistema y los **registros de llamadas**. El contenido de este directorio se utiliza frecuentemente tanto en el diagnóstico y la solución de problemas, como en la auditoría y administración del sistema.

Todos los módulos de software que hemos mencionado anteriormente se encuentran ubicados dentro del directorio **/usr/lib/asterisk/modules** y se cargan por defecto al inicializar el sistema, a menos que se los deshabilite utilizando el archivo de configuración **modules.conf**.

QUEUE GAMES

Cuando armamos un call center, en vez de utilizar música repetitiva para amenizar la espera de los clientes, una buena idea es usar **Queue Games**, un sistema de preguntas tipo trivia. Si el usuario es capaz de acertar, lo atenderán antes.

Las versiones de Asterisk

Sin entrar en detalles sobre los sucesivos cambios que ha sufrido la denominación de versión en el mundo Asterisk y de los motivos que han llevado a esto, diremos que el identificador de versión se compone de dos elementos: **tronco** y **ramas**. Por ejemplo, la versión 1.8 corresponde al tronco 1 rama 8. Dentro de cada rama tendremos las correcciones a los problemas funcionales encontrados y a cuestiones relacionadas con la seguridad. Por su parte, un cambio de tronco significa que se han introducido modificaciones en la arquitectura y en las características.

Nuestro consejo es que, en un ambiente de producción, vale más la estabilidad que disponer de nuevas características, tal vez inestables y raramente utilizadas. En la actualidad, se está migrando a un identificador de dos dígitos con un dígito de subversión.

También existen dos clases de versiones: **standard** y **LTS** (**Long Term Support**). La segunda es la que utilizaremos en este libro, dado que resulta la más estable. Si bien no posee las funcionalidades más sofisticadas, y menos probadas, la encontraremos en la mayoría de las implementaciones.

Asterisk libera versiones nuevas cada año, alternando standard con LTS, y parches cada cuatro meses. Las versiones que se encuentran en mantenimiento (aquellas con una vida superior al año) solo reciben parches de seguridad y bajo demanda. Las que superan la marca de **EOL** (**End Of Life**) reciben asistencia durante un año más después de alcanzada la marca.

Antes de la instalación

Asterisk puede ejecutarse en una gran cantidad de plataformas **Linux**. Por lo general, los usuarios utilizan la que manejan o conocen mejor, pero lo cierto es que el sistema funcionará, la mayoría de las veces, en cualquiera de ellas. Sin embargo, debemos tener ciertos cuidados en la selección del **kernel** que vamos a usar, porque de la misma manera en que una mala selección del hardware ocasionará inconvenientes en las prestaciones

¿MITO O REALIDAD?

Muchos piensan que estar cerca de una torre de telefonía celular aumentaría el riesgo de padecer cáncer. Por esto, las empresas de telecomunicaciones idearon el uso de torres camufladas de árboles, postes, depósitos de agua o torres de iglesias.

Figura 4. En el sitio web de Ubuntu encontraremos gran variedad de información sobre la instalación.

del sistema, el kernel sobre el que vamos a montar Asterisk también puede hacerlo. Lo ideal es disponer de un kernel lo más limpio posible, sin módulos de software, estéticos o de servicios que no se requieran. Además, debemos aclarar que, por su naturaleza, Asterisk no se lleva bien con otras aplicaciones que se ejecuten en el mismo hardware y bajo la administración del mismo sistema operativo.

Linux es el único sistema operativo soportado oficialmente, y se aconseja el uso de la versión de kernel **2.6.25** o superior (aunque Asterisk corre sobre kernels 2.4). Por lo general, se aconsejan algunas distribuciones de libre acceso que permiten construir un sistema con solo lo necesario para proveer el contexto donde ejecutar Asterisk. Las distribuciones **CentOS** (**www.centos.org**) y **Ubuntu Server** (**www.ubuntu.com**) son las preferidas a la hora de seleccionar la plataforma. Las instrucciones para instalar el software de plataforma pueden obtenerse de los sitios oficiales, no las veremos en este libro.

En el desarrollo de nuestro trabajo emplearemos el sistema operativo Ubuntu, porque es uno de los más accesibles y conocidos. Entonces, los comandos y las instrucciones de instalación de Asterisk los daremos en el contexto de este sistema.

CURIOSIDADES CON ASTERISK

¿Problemas de comunicaciones o energía? En un poblado de Uganda, se consiguió la comunicación gracias a una centralita de VoIP, lo que permitió ahorrar caminatas de más de 30 kilómetros. Eso sí, para hacerla funcionar, se necesita que alguien le dé al pedal.

El objetivo de esta obra es armar un sistema Asterisk básico desde cero, para obtener un producto apto para el aprendizaje y entrenamiento con esta tecnología. Dado que algunos lectores podrían encontrar difícil implementar esta propuesta, abordaremos el tema de los proyectos de sistemas abiertos sobre Asterisk, que intentan proporcionar una implementación más sencilla y rápida, a través de paquetes que incluyen todo lo necesario para armar un sistema Asterisk funcional. Aunque debemos saber que esto siempre conlleva el costo de la reducida flexibilidad y la desactualización de estos productos.

PAQUETES REQUERIDOS

Para realizar nuestra instalación, solo necesitamos el paquete Asterisk, y sugerimos tener los archivos de sonidos, como el **asterisk-sounds**, que puede encontrarse también como Core Sound y Extra Sound.

Con respecto a los paquetes de la distribución Linux elegida, tendremos que contar con los siguientes:

- **GCC (3.X)**
- **ncurses-dev**
- **libtermcap-dev**
- **GCC C++ 3.x**
- **libtool** (opcional pero recomendada)
- **GNU make** (versión 3.80 o mayor)
- **libcurl4-openssl-dev**

El código fuente de Asterisk puede descargarse del sitio oficial: www.asterisk.org.

Instalación de Asterisk

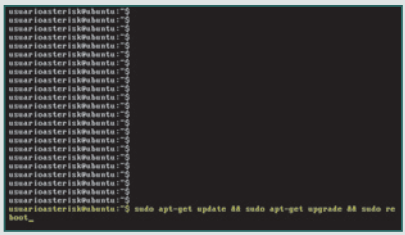
Cualquiera sea el sistema operativo que hayamos elegido, siempre debemos generar un usuario para realizar la instalación y ejecutar el sistema Asterisk que vamos a crear. Este detalle es necesario porque, para ejecutar la aplicación, tenemos que hacerlo bajo un usuario específico, que será su dueño. No es aconsejable que este sea `root`, ya que para ejecutar los comandos de instalación es más seguro hacerlo desde un usuario distinto. En nuestro caso, hemos creado el usuario denominado **usuarioasterisk**. A continuación, veremos cómo realizar la instalación de la aplicación en diferentes procedimientos paso a paso.

DIGITAL SIGNAL PROCESSOR

El término DSP (Digital Signal Processor) define un dispositivo integrado capaz de interpretar y modificar las señales de varias maneras. Este circuito realiza la codificación/decodificación del audio, lo que en general requiere un gran poder de cómputo.

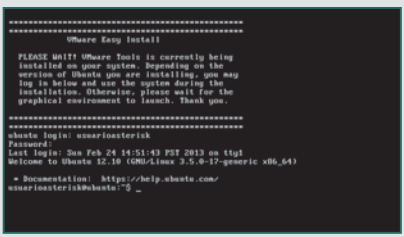
PASO A PASO /1
Preparar la plataforma

1



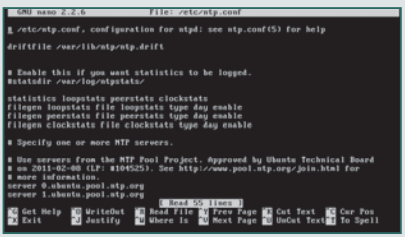
Con el S.O. instalado, realice su actualización y reinicie el sistema. Ingrese el siguiente comando en el prompt: **`# sudo apt-get update && sudo apt-get upgrade && sudo reboot.`**

2



Una vez realizado el reboot, debe ingresar otra vez en el sistema con el usuario y la contraseña correspondientes.

3

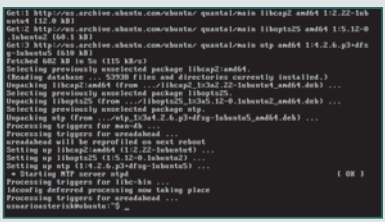


Instale el demonio de sincronización de tiempo (NTP) y actualice el tiempo del sistema: **`# sudo apt-get install ntp`**.

Resulta fundamental mantener la hora precisa en un sistema Asterisk productivo. Como en otros casos de aplicaciones que brindan servicio, es indispensable contar con un registro temporal cierto, tanto para el sincronismo entre los componentes del sistema, como para el registro de las llamadas, su duración, los cargos por tiempo, las notificaciones de la casilla de mensajes de voz, etcétera. Debido al tratamiento que hace Ubuntu de este tema, por defecto, tendremos que reconfigurarlo. Como es usual en Linux, esto significa editar y alterar uno o más archivos de configuración. El editor que trae incorporado Ubuntu se llama **nano**, y es el que emplearemos para hacerlo. A continuación, veremos cómo lograrlo.

PASO A PASO /2
Configurar el demonio NTP

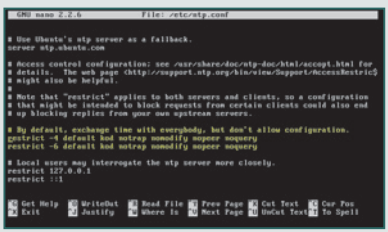
1



En el prompt del S.O., ingrese el siguiente comando: **`# sudo nano /etc/ntp.conf`**.

El archivo **`ntp.conf`** controlará el comportamiento del demonio NTP.

2

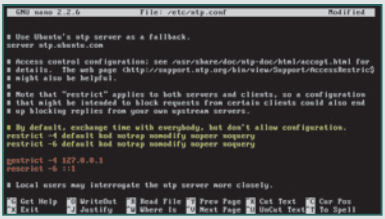


El contenido del archivo **`ntp.conf`** se mostrará en pantalla. Identifique la sección que contiene las siguientes líneas:

```
# By default, exchange time
with everybody, but don't allow
configuration.
restrict -4 default kod notrap
nomodify nopeer noquery

restrict -6 default kod notrap
nomodify nopeer noquery
```

PASO A PASO /2 (cont.)

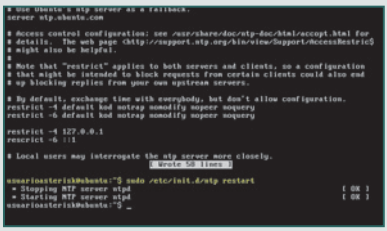
3



Para permitir que el demonio NTP se sincronice con una fuente externa, agregue a continuación las siguientes líneas: **# By default, exchange time with everybody, but don't allow configuration.**

```
restrict -4 default kod notrap nomodify novpeer noquery

restrict -6 default kod notrap nomodify nopeer

noqueryrestrict -4 127.0.0.1
```

4



En pantalla deberá ver lo que se que muestra a continuación: **# By default, exchange time with everybody, but don't allow configuration.**

```
restrict -4 default kod notrap nomodify novpeer noquery

restrict -6 default kod notrap nomodify nopeer

noqueryrestrict -4 127.0.0.1

restrict -6 ::1
```

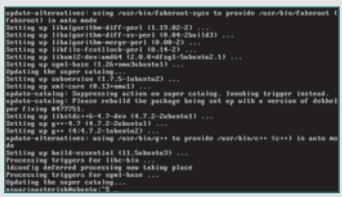
Para salir del editor, presionamos la combinación de teclas **CTRL + X** y, luego, pulsamos **Y** para guardar las modificaciones.

Es importante saber que no debemos cambiar el nombre del archivo sino aceptar el sugerido por el editor: **/etc/ntp.conf**. A continuación, debemos reiniciar el demonio para que tome los cambios efectuados. Esto se hace con el siguiente comando:

```
sudo /etc/init.d/ntp restart
```

PASO A PASO /3
Instalación de las dependencias de software

1

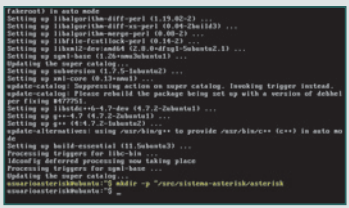


Ingrese el siguiente comando en el prompt del S.O.:

```
# sudo apt-get install build-essential
subversion \

libncurses5-dev libssl-dev libxml2-dev.
```

A continuación, se desplegará la instalación de tres librerías.

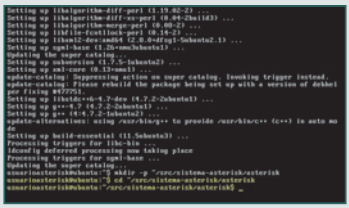
2



Cree la estructura de directorios:

#mkdir -p ~/src/sistema-asterisk/asterisk. En este directorio se almacenará la fuente de Asterisk para su posterior instalación.

3



Vaya al directorio creado:

```
# cd ~/src/sistema-asterisk/asterisk
```

```
A    1.8/menuselect/mxml/Makefile.in
A    1.8/menuselect/mxml/mxml.pc.in
A    1.8/menuselect/mxml/configure.in
A    1.8/menuselect/mxml/mxml-set.c
A    1.8/menuselect/mxml/ANNOUNCEMENT
A    1.8/menuselect/mxml/mxml.list.in
A    1.8/menuselect/mxml/README
A    1.8/menuselect/mxml/config.h.in
A    1.8/menuselect/mxml/mxml-search.c
A    1.8/menuselect/mxml/mxml-string.c
A    1.8/menuselect/mxml/mxml.h
A    1.8/menuselect/mxml/mxml-index.c
A    1.8/menuselect/mxml/mxml-attr.c
A    1.8/menuselect/mxml/mxml-private.c
A    1.8/menuselect/mxml/mxml-entity.c
A    1.8/menuselect/mxml/COPYING
A    1.8/menuselect/mxml/CHANGES
A    1.8/menuselect/mxml/mxml-file.c
A    1.8/menuselect/mxml/install-sh
 U   1.8/menuselect/mxml
Checked out external at revision 430.

Checked out revision 1110.
Checked out revision 382006.
usuarioasterisk@ubuntu:~/src/sistema-asterisk/asterisk$ _
```

Figura 5. Pantalla final, después de una descarga exitosa del software Asterisk; en este caso, la versión 1.8.

Ahora tenemos que instalar las dependencias de software requeridas por Asterisk.

En este directorio descargaremos el código del software Asterisk, para desde él iniciar las instalaciones correspondientes. Este tema lo veremos en el siguiente apartado.

DESCARGAR EL CÓDIGO DE ASTERISK

En el mundo Linux, en general, siempre hay varias maneras para obtener el código de una aplicación. La más popular entre los usuarios es a través del **Centro de software**, en el caso de un desktop; o bien su versión **CLI** (**Command Line Interface**), mediante el comando **apt-get**. En este caso, las versiones se obtienen de repositorios reconocidos por la comunidad Linux, pero que pueden estar algo desactualizados en comparación con las versiones estables que utiliza la comunidad Asterisk.

Un método que permite obtener el código original es el denominado **subversion**. Para esto, debemos ingresar el siguiente comando en el prompt del sistema operativo:

```
# svn co http://svn.asterisk.org/svn/asterisk/branches/1.8.
```

En cambio, para hacer la descarga alternativa de una versión específica tenemos que ingresar:

```
# svn co http://svn.asterisk.org/svn/asterisk/branches/1.8.X
```

INSTALAR EL SOFTWARE DIGIUM ASTERISK HARDWARE INTERFACE (DAHDI)

Digium Asterisk Hardware Interface (DAHDI) es un software que se requiere como interfaz del sistema operativo y el hardware de telefonía. Si bien en este trabajo no abordaremos la interconexión con el mundo exterior a través de hardware específico, este software contiene ciertas dependencias que pueden ser requeridas por algunas funciones de Asterisk que sí podríamos necesitar.

Es importante que la versión del kernel en uso coincida con la del código fuente del kernel

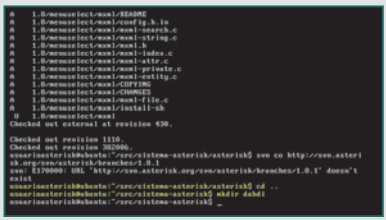
instalado. Para verificar la versión del kernel que se está ejecutando, usamos el siguiente comando:

```
sudo apt-get install linux-headers-`uname -r`.
```

Como el software DAHDI se actualiza con frecuencia, para tener la última versión, debemos consultar el sitio **http://donwloads.asterisk.org** e ingresar los identificadores correspondientes, que aquí indicamos como **id-version**.

PASO A PASO /4
Instalación del software DAHDI

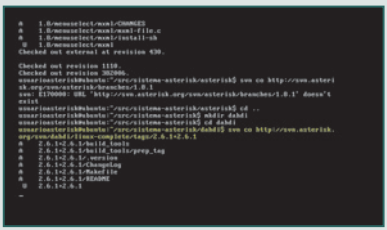
1



Cree el directorio DAHDI en Asterisk:

```
# cd ..

# mkdir dahdi
```

2

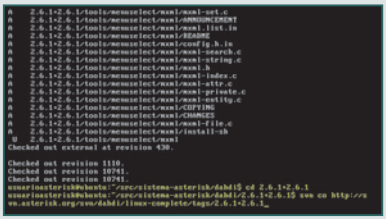


Descargue el software en el directorio creado: **# cd dahdi/**

```
# svn co http://svn.asterisk.
org/svn/dahdi/linux-complete/
tags/ "id-version-dhadi_
linux+id-version-dhadi_tools"
```

PASO A PASO /4 (cont.)

3

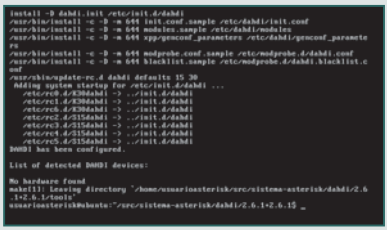


Cambie al directorio **/"id-version-dhadi_linux+id-version-dhadi_tools"**:

```
# cd "id-version-dhadi_
linux+id-version-dhadi_tools"
```

```
# svn co http://svn.asterisk.
org/svn/dahdi/linux-complete/
tags/ "id-version-dhadi_
linux+id-version-dhadi_tools"
```

4



Instale y configure el software:

```
# make
```

```
# sudo make install
```

```
# sudo make config
```

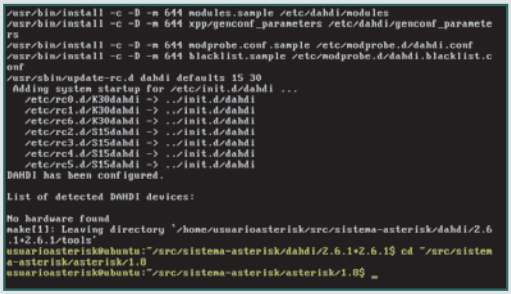
Es importante señalar que el identificador de versión de DAHDI está compuesto de dos partes, porque el software contiene tanto los drivers del sistema Linux, como las herramientas de configuración y gestión correspondientes.

Si volvemos al paso a paso anterior, podemos observar que esto lo hemos indicado de la siguiente manera:

"id-version-dhadi_linux+id-version-dhadi_tools".

PASO A PASO /5
Instalar el software Asterisk y los accesorios

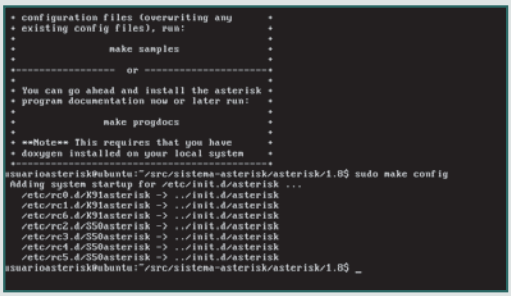
1



Cambie el directorio donde se descargó el código Asterisk: **`# cd ~/src/sistema-asterisk asterisk/1.8.`** Instale el software descargado, mediante los siguientes comandos:

```
# ./configure
# make
# sudo make install
# sudo make config
```

2

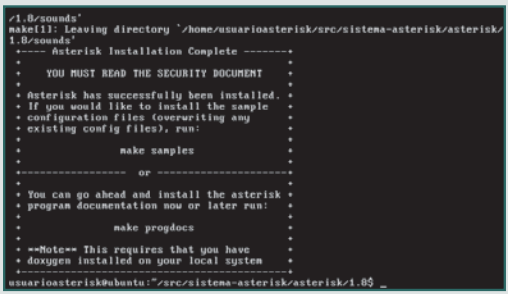


Instale la utilidad **`Menuselect`**: **`# cd ~/src/sistema-asterisk/asterisk/1.8/`**

```
# make menuselect
# sudo make install
```

PASO A PASO /5 (cont.)

3



Modifique los permisos en los directorios de instalación:

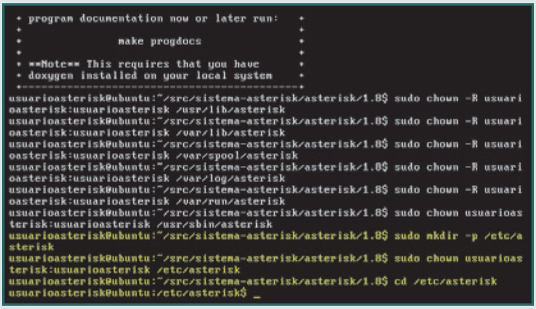
```
# sudo chown -R usuarioasterisk:usuarioasterisk /usr/lib/
asterisk/
# sudo chown -R usuarioasterisk:usuarioasterisk /var/lib/
asterisk/
# sudo chown -R usuarioasterisk:usuarioasterisk /var/spool/
asterisk/
# sudo chown -R usuarioasterisk:usuarioasterisk /var/log/
asterisk/
# sudo chown -R usuarioasterisk:usuarioasterisk /var/run/
asterisk/
# sudo chown usuarioasterisk:usuarioasterisk /usr/sbin/
asterisk
```

Si no existe, cree el directorio **/etc/asterisk**:

```
# sudo mkdir -p /etc/asterisk
# sudo chown usuarioasterisk:usuarioasterisk /etc/asterisk
# cd /etc/asterisk/
```

PASO A PASO /5 (cont.)

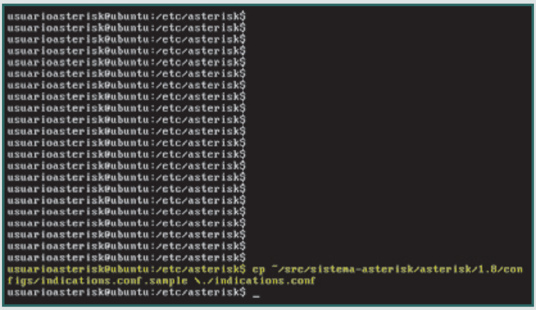
4



Copie el archivo `indications.conf.sample`:

```
# cp ~/src/sistema-asterisk/asterisk/1.8/configs/indications.conf.sample \
./indications.conf
```

5

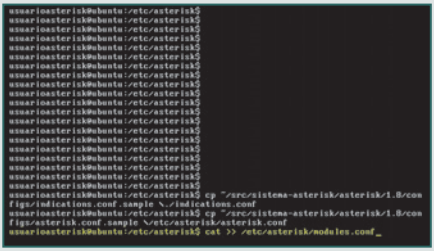


Copie el archivo `asterisk.conf.sample`:

```
# cp ~/src/sistema-asterisk/asterisk/1.8/configs/asterisk.conf.sample \
/etc/asterisk/asterisk.conf
```

PASO A PASO /5 (cont.)

6



Durante la instalación del S.O., creó un usuario distinto de root; ese usuario es el que debe emplear para llevar a cabo la instalación. Cree el archivo **modules.conf**:

```
# cat >> /etc/asterisk/modules.conf
```

Aquí podrá habilitar la carga de los módulos de software automáticamente (los que se encuentren en el directorio correspondiente) y deshabilitar los que sean innecesarios para la instalación. Configure los parámetros básicos de **musiconhold.conf**:

```
# cat >> musiconhold.conf
; musiconhold.conf
[default]
mode=files
directory=moh
Ctrl + D
```

Para finalizar con el procedimiento, deberemos guardar todos los cambios introducidos. En los capítulos que siguen, avanzaremos con la configuración de cada uno de los archivos que hemos instalado.

RESUMEN

En el presente capítulo, hemos visto la estructura de Asterisk, el versionado y cómo dimensionar una plataforma, para luego instalar el programa y los accesorios de software necesarios para componer un sistema básico, que configuraremos en capítulos posteriores.

Capítulo 2

Configuración de Asterisk

Aprenderemos a configurar las principales herramientas del sistema Asterisk.

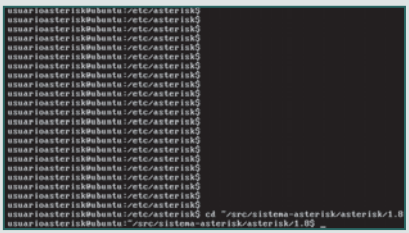
RedUSERS.com

Configurar archivos **de muestra**

Si cumplimos con éxito las tareas del capítulo anterior, en este punto deberíamos de tener un sistema Asterisk funcionando, pero sin que pueda prestar servicio alguno, pues nos falta realizar su configuración. Para esto, y como sucede en general con las aplicaciones Linux, deberemos editar archivos de configuración y completarlos con ciertos datos, que le darán forma al sistema bajo implementación. Para comenzar a familiarizarnos con ellos, vamos a generar los archivos de muestra que vienen con el sistema.

PASO A PASO /1
Generar archivos de muestra

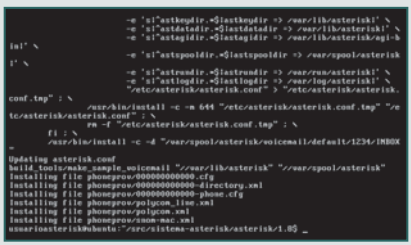
1



Vaya al directorio donde descargó Asterisk:

```
# cd ~/src/sistema-asterisk/
asterisk/1.8. (1.8. es la
versión descargada).
```

2



Genere los archivos de muestra:

```
# sudo make samples
```

Esto creará en el directorio **/etc/asterisk** los archivos de configuración que podremos emplear como referencia para nuestras propias configuraciones. Una vez creados, sugerimos que los explore, para tratar de familiarizarse con la nomenclatura y los formatos de las directivas empleadas.

A continuación, pasaremos a mover estos archivos a un directorio de reserva, que deberá ser creado para alojar los archivos originales, con el propósito de tenerlos para futuras referencias. Denominaremos a este directorio **/etc/asterisk/samples**.

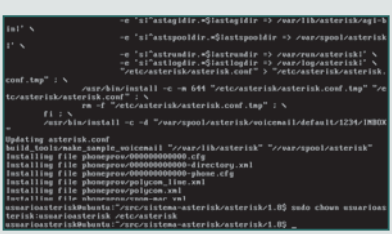
Debemos ser cuidadosos al correr el comando **make samples** en un sistema Asterisk productivo, puesto que sobrescribirá los archivos de configuración existentes y desarmará, literalmente, nuestro sistema de telefonía. Si esto sucede, deberemos recurrir a las copias de respaldo para restituir los archivos reemplazados por el comando y, así, recuperar la funcionalidad de la central telefónica Asterisk.

Resultará beneficioso y pedagógico crear desde cero los archivos de configuración que se requerirán. Esto permitirá tener una mejor comprensión del funcionamiento del sistema y, además, nos permitirá concentrarnos en los archivos que realmente necesita el sistema básico, y no distraernos con la gran cantidad de información y archivos que implica la configuración completa de Asterisk.

En principio, podemos utilizar los archivos **indications.conf** y **asterisk.conf** creados en el directorio fuente de Asterisk, copiándolos a **/etc/asterisk**. Veamos el siguiente procedimiento.

PASO A PASO /2
Copiar los archivos de muestra

1

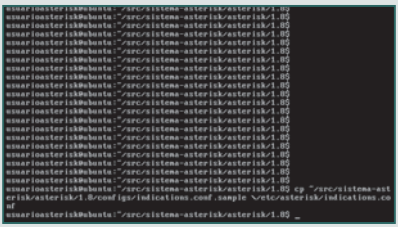


Cambie los permisos del directorio para el usuario administrador creado: **#sudo chown usuarioasterisk:usuarioasterisk /etc/asterisk**

#sudo chown usuarioasterisk:usuarioasterisk /dev/dahdi/transcode

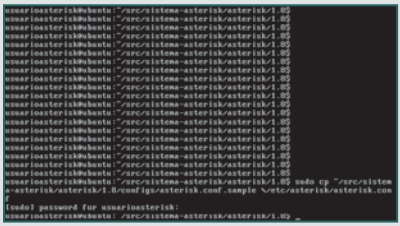
PASO A PASO /2 (cont.)

2



Copie el archivo **indications.conf**, creado en el paso a paso anterior, al directorio de configuraciones de Asterisk: **cp ~/src/sistema-asterisk/asterisk/1.8/configs/indications.conf.sample \/etc/asterisk/indications.conf**

3



Copie el archivo **asterisk.conf**, creado en el paso a paso anterior, al directorio de configuraciones de Asterisk: **cp ~/src/sistema-asterisk/asterisk/1.8/configs/asterisk.conf.sample:**

\/etc/asterisk/asterisk.conf

PASO A PASO /2 (cont.)

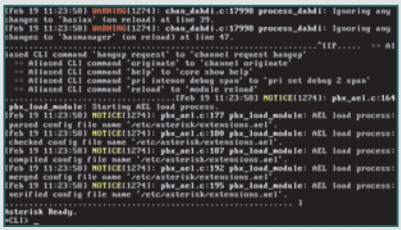
4

Edite el archivo **asterisk.conf**, busque la sección **[options]** y modifique los ítem:

runuser=usuarioasterisk

rungroup=usuarioasterisk

5



Al guardar el archivo resultante, estará en condiciones de ejecutar Asterisk en una configuración mínima, con la que poco podrá hacer, pero será el punto de partida. Emplee para esto el comando:

/usr/sbin/asterisk -c

Mientras Asterisk se inicia, veremos diferentes mensajes en pantalla. Al terminar el proceso, obtendremos una interfaz en modo línea de comando o CLI. Este comando inicia Asterisk en modo de ejecución de primer plano. Aquí podremos jugar con algunos comandos del programa, solo para verificar su funcionalidad; por ejemplo, revisar el estado de los módulos con el comando CLI **`module show`** o detener la ejecución del módulo con el comando **`core stop now`**.

Configurar los archivos del sistema

Como mencionamos en el **Capítulo 1**, los archivos de configuración fundamentales para contar con un sistema Asterisk operativo son los que detallamos a continuación:

- **`asterisk.conf`**
- **`modules.conf`**
- **`musiconhold.conf`**
- **`indications.conf`**

A continuación, describiremos cada uno de ellos en detalle y veremos su configuración.

ARCHIVO ASTERISK.CONF

Este archivo está compuesto por numerosas secciones. Algunas de ellas ya las vimos y, en general, diremos que se trata del archivo de configuración que gobierna el funcionamiento de la aplicación Asterisk como tal. Comúnmente, podremos utilizar el archivo de muestra que viene con el sistema, haciéndole los cambios sugeridos. Es más, podremos ejecutar Asterisk sin él, pero algunas de sus opciones nos resultarán muy útiles, y otras, necesarias.

En este archivo se indican los directorios de la estructura de Asterisk y su función, aunque es posible cambiar los predefinidos si fuera necesario. También se pueden parametrizar opciones de acceso remoto y compatibilidad entre versiones. Como con cualquier aplicación UNIX, podremos recurrir al comando **`man asterisk`** para revisar con detalle lo que estemos buscando.

En esta instancia, vamos a crear los archivos **`modules.conf`** y **`musiconhold.conf`**, y a configurar lo mínimo indispensable para iniciar nuestro sistema de manera funcional.

CENTRALES TELEFÓNICAS

Son dispositivos que realizan en forma automática el trabajo que, en los albores de la telefonía, efectuaban las telefonistas. En sus inicios, fueron grandes equipos electromecánicos, que evolucionaron a equipos basados en circuitos integrados a gran escala.

ARCHIVO MODULES.CONF

El archivo **modules.conf** debe ser creado mediante el editor preferido en el directorio **/etc/asterisk**. Este archivo se emplea para definir qué módulos deben ser cargados por Asterisk al iniciar el sistema y cuáles no. Por defecto, con el parámetro **autoload=yes**, Asterisk cargará todos los módulos que se encuentran en el directorio **/usr/lib/asterisk/modules**.

```
[modules]

autoload=yes
```

Ahora, en el mismo archivo debemos desactivar los módulos innecesarios:

```
; Resource modules

noload => res_speech.so

noload => res_phoneprov.so

noload => res_ael_share.so

noload => res_clialiases.so

noload => res_adsi.so

; PBX modules

noload => pbx_ael.so

noload => pbx_dundi.so

; Channel modules

noload => chan_oss.so

noload => chan_mgcp.so

noload => chan_skinny.so

noload => chan_phone.so

noload => chan_agent.so

noload => chan_unistim.so

noload => chan_alsa.so

; Application modules

noload => app_nbscat.so

noload => app_amd.so

noload => app_minivm.so

noload => app_zapateller.so

noload => app_ices.so

noload => app_sendtext.so

noload => app_speech_utils.so

noload => app_mp3.so

noload => app_flash.so

noload => app_getcpeid.so
```

```
noload => app_setcallerid.so
noload => app_adsiprog.so
noload => app_forkcdr.so
noload => app_sms.so
noload => app_morsecode.so
noload => app_followme.so
noload => app_url.so
noload => app_alarmreceiver.so
noload => app_disa.so
noload => app_dahdiras.so
noload => app_senddtmf.so
noload => app_sayunixtime.so
noload => app_test.so
noload => app_externalivr.so
noload => app_image.so
noload => app_dictate.so
noload => app_festival.so
```

Si queremos observar qué sucede ahora con el sistema que estamos implementando, podemos iniciar Asterisk otra vez, como se indica en el punto 5 del **Paso a Paso Copiar los archivos de muestra**, y observar qué módulos están cargados.

Ingresamos en el sistema con el usuario creado para administrar Asterisk (en nuestro ejemplo, **usuarioasterisk**) y escribimos los comandos:

```
# asterisk -c
CLI> module show
```

```
format_g719.so                 ITU G.719                                 0
app_meetme.so                  MeetMe conference bridge                  0
cel_custom.so                  Customizable Comma Separated Values CEL   0
func_extstate.so               Gets an extension's state in the dialpla 0
app_dumpchan.so                Dump Info About The Calling Channel       0
res_convert.so                 File format conversion CLI command        0
func_sysinfo.so                System information related functions      0
func_volume.so                 Technology independent volume control     0
chan_local.so                  Local Proxy Channel (Mote: used internal 0
func_audiohookinherit.so       Audiohook inheritance function            0
app_directed_pickup.so         Directed Call Pickup Application          0
147 modules loaded
*CLI> _
```

Figura 1. **Algunos de los módulos funcionales cargados en el sistema Asterisk.**

Veremos los módulos que han sido cargados, interpretando nuestras instrucciones dadas a través del archivo **modules.conf**.

ARCHIVO MUSICONHOLD.CONF

Todo sistema telefónico posee una serie de sonidos para indicar ciertas señales, como el tono de llamada entrante, el tono de ocupado o la música en espera. Estos sonidos deberán ser instalados de alguna manera en el sistema para que Asterisk pueda utilizarlos cuando sea necesario. El archivo **musiconhold.conf** define las clases para los sonidos de la música en espera, empleados en varias situaciones durante el flujo de una llamada. Para comenzar, debemos crear un archivo en **/etc/asterisk**, llamarlo **musiconhold.conf** y completarlo con las siguientes líneas:

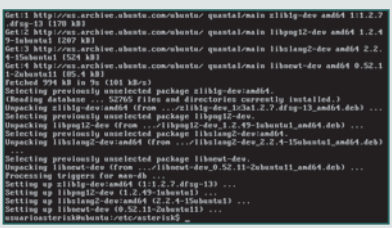
```
; Clase por defecto

[default]

mode=files

directory=moh
```

Los sonidos reales, digitalizados y codificados se deben instalar mediante la utilidad **Menuselect** de Asterisk. Esta se emplea para una gran cantidad de situaciones, desde el apoyo para el diagnóstico de problemas, hasta la instalación de paquetes definidos, como los sonidos en varias codificaciones y formatos. En general, se la utiliza para configurar qué módulos compilar e instalar; es decir, es otra manera de armar el sistema Asterisk.

La utilidad **Menuselect** debe ser instalada en nuestro sistema. Veremos cómo hacerlo en el siguiente Paso a Paso.

PASO A PASO /3
Instalación de Menuselect

1

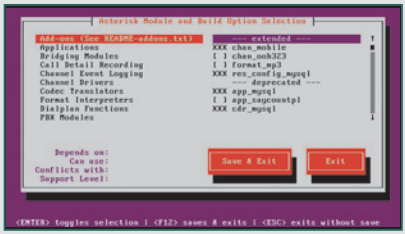


Instale la librería **libnewt** necesaria para habilitar la nueva presentación de la interfaz de **Menuselect**:

```
# sudo apt-get install
libnewt-dev
```

PASO A PASO /3 (cont.)

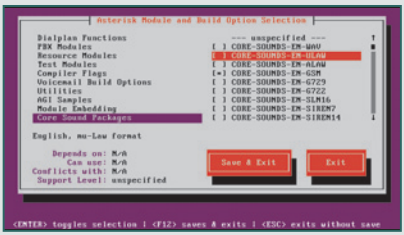
2



Vaya al directorio en que descargó Asterisk: **`# cd ~/src/sistema-asterisk/asterisk/1.8.x`**, e instale la utilidad:

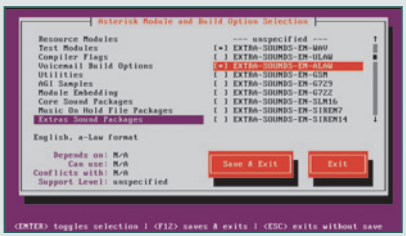
```
# cd menuselect
# make clean
# ./configure
# cd ..
# make menuselect
```

3



Se presentará una interfaz en dos colores, que podrá navegar con las teclas de dirección. Vaya a **`Core Sound Packages`** y presione la tecla **`FLECHA DERECHA`** o **`ENTER`** para desplegar el submenú de opciones disponibles.

4



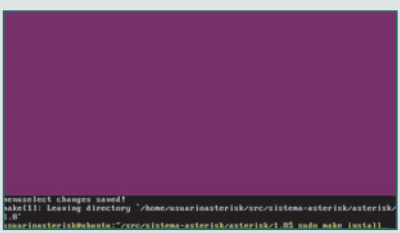
Seleccione **`CORE-SOUNDS-EN-WAV`**, vuelva a las opciones con **`FLECHA IZQ.`** y seleccione **`Extra Sound Packages`**. Aquí conviene seleccionar los sonidos en formato **`WAV`** y **`ALAW`**.

Es importante destacar que el tipo de formato de los archivos de sonido tendrá un impacto importante en el rendimiento del procesador al momento de su uso. Siempre debemos utilizar un formato compatible.

Una vez terminado el proceso de selección, ingresamos **X** para guardar y salir de la utilidad. Ahora solo nos queda instalar los paquetes de sonido. Para hacerlo, es necesario que el servidor tenga una conexión operativa a Internet.

PASO A PASO /4
Instalación de los paquetes de sonido

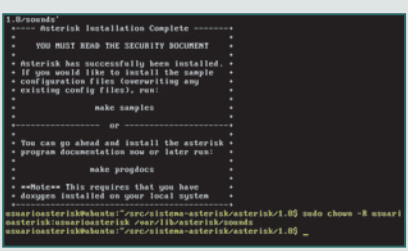
1



Instale los paquetes seleccionados con **Menuselect**:

```
# sudo make install.
```

2



Cambie los permisos del directorio para el usuario creado:

```
# sudo chown -R
usuarioasterisk:usuarioasterisk
/var/lib/asterisk/sounds/.
```

Música en espera: derechos de autor

Otro aspecto importante que debemos tener en cuenta con respecto a los sonidos es el **legal**. No es posible utilizar cualquier tema musical, ya sea descargado de Internet o tomado de un CD, adquirido o no, dado que la licencia que lo protege no permite su uso comercial, salvo pago de un canon que determina el dueño de los derechos del tema en cuestión. Este puede ser el autor, una sociedad de autores o algún particular o empresa que los haya adquirido legalmente. En caso de emplearlo sin abonar este canon, se estará infringiendo los derechos de autor y será aplicable la penalización que contemple la ley para ese caso. Por eso, es necesario revisar los derechos que protegen a los autores de la música que queremos utilizar o crear nuestros propios temas musicales.

Si localizamos temas con licencias tipo **CC**, tendremos que resolver el asunto del formato. Por lo mencionado anteriormente, el formato ideal es **WAV**, pero el más popular es **MP3**, así que tendremos que realizar una conversión de formatos antes de instalar el archivo en el sistema.

ARCHIVO INDICATION.CONF

Por último, el archivo **indications.conf** es, desde el punto de vista operativo, el más interesante, porque en él se definen los sonidos de señalización telefónica, asociándolos a los canales que se configuren en el sistema. Esto se realizará aplicando un **tonezone** a los canales, como parte de su configuración.

Así quedarán determinados el conjunto de sonidos por utilizar según la región o el país para el que estemos configurando el sistema. La línea a continuación muestra esta asignación:

```
Set(CHANNEL(tonezone)=[código_de_
país]) ;
```

El código de país coincidirá con el identificador que se ubica en los dominios **DNS**. Podemos modificar el archivo de muestra que viene instalado con Asterisk. Lo copiamos desde el directorio **/usr/src/sistema-asterisk/1.8/configs/indications.conf.sample** y, en la copia, modificamos el parámetro **country** en la sección **[general]**.

RESUMEN

En este capítulo creamos y configuramos los cuatro archivos básicos del sistema. Instalamos los grupos de sonidos empleados en la señalización y en el servicio de música en espera, y la utilidad Menuselect, empleada para definir los componentes del sistema Asterisk.

Capítulo 3

Configuración de canales y dispositivos

Veremos cómo Asterisk se comunica con los dispositivos de los usuarios para interconectarlos.

RedUSERS.com

Canales y dispositivos

Antes de comenzar con los detalles de la configuración de canales y dispositivos, debemos hacer una aclaración. Si bien Asterisk permite la interconexión de teléfonos analógicos, siempre que contemos con el hardware correspondiente, en este capítulo solo vamos a tratar la conexión de dispositivos IP, en particular, los denominados **softphones**. Ahora sí, entraremos en nuestro tema.

Los **canales** son una abstracción empleada por Asterisk para definir parámetros relacionados con los dispositivos que se conectan a él. Estos se implementan mediante archivos de configuración, donde se inscriben los parámetros correspondientes a los dispositivos telefónicos, ya sean teléfonos o troncales de interconexión entre centrales telefónicas públicas o privadas. En tanto, los **dispositivos** son los aparatos o el software telefónico que empleamos para realizar llamadas.

Asterisk se puede conectar con distintos tipos de interfaces y dispositivos, como:

- Circuitos digitales, como troncales **E1**.
- Protocolos de voz sobre IP, como **SIP**.
- Interfaces analógicas, como teléfonos analógicos o líneas telefónicas.
- Teléfonos IP, físicos y softphones.

La conexión se realiza a través de hardware que, a diferencia de los productos propietarios, es de tipo estándar y de bajo costo. En nuestro caso, solo emplearemos la **conexión de red** como interfaz para realizar la interconexión de dispositivos, dado que, como dijimos, solo nos ocuparemos de dispositivos IP. No obstante, mostraremos cómo configurar una interfaz que nos permita conectarnos a una línea telefónica convencional, con el propósito de realizar llamadas al exterior de la central Asterisk.

La configuración de los canales es un procedimiento relativamente sencillo y directo, pero también será necesario configurar los dispositivos telefónicos involucrados en las llamadas. La configuración se separa en dos grandes etapas:

- Configurar Asterisk con los parámetros del dispositivo.
- Configurar el dispositivo con los parámetros de Asterisk.

PUERTOS FXO Y FXS

Las siglas FXO/FXS se utilizan para definir las interfaces estándar de un teléfono o de una central telefónica. Un puerto FXO no genera tono de invitación a discar pero acepta uno, y uno FXS genera tono de invitación a discar sin aceptarlo en retorno.

Estas actividades se realizan sobre archivos de configuración; en el caso de Asterisk, sobre el archivo de configuración del canal utilizado por el dispositivo. En el caso del dispositivo, se trata de un archivo que se encuentra dentro de la estructura de software del mismo aparato o softphone.

Protocolo SIP

SIP (**Session Initiation Protocol**) es un protocolo de señalización telefónica utilizado en teléfonos **VoIP**, tanto físicos como software. Consiste en un protocolo peer-to-peer empleado para el establecimiento de las llamadas, las maniobras durante ellas (por ejemplo, transferencia) y su finalización en redes de telefonía IP. Este no trata con la información de voz de la llamada, los tonos de invitación de la central o la música en espera, porque de su transporte se encarga el protocolo **RTP** (**Real Time Protocol**), descripto en la **RFC 3550**. La definición **peer-to-peer** implica que un teléfono SIP necesita hacer una conexión directa con otro teléfono SIP, sin que una **PBX** se encuentre en el medio. Sin embargo, Asterisk no cumple con esta norma. Cuando una llamada se realiza entre dos terminales SIP a través de una central Asterisk, esta se encontrará en medio de la comunicación. Habrá dos llamadas: una entre la terminal llamante y Asterisk, y otra, entre Asterisk y la terminal llamada. La central conmutará la llamada, realizando un **bridge** de los dos canales involucrados. Los extremos de la comunicación SIP, denominados **agentes de usuario**, pueden ser cliente o servidor. El cliente es el que generará la llamada, en tanto que el servidor será el encargado de procesarla y producir una respuesta. Por ejemplo, cuando un teléfono IP hace una llamada a un softphone, genera un requerimiento y lo envía a un proxy SIP. Este se hace cargo del requerimiento, analiza el destino y lo conmuta hacia él. Una vez que los dos agentes han negociado con éxito el establecimiento de la llamada, la voz es transportada a través de RTP y enviada directamente entre los dos agentes. El proxy no se ocupa de los datos de voz, solo habla SIP.

Con respecto a su comportamiento en un entorno SIP, Asterisk se define como un agente de usuario **back-to-back** (B2BUA), dado que actúa como un agente de usuario SIP tanto en la recepción como en el reenvío de la llamada a otro extremo. Cuando el teléfono IP llama a un número de extensión, la llamada se establece entre él y Asterisk. Si la programación de Asterisk indica que hay que llamar a otro agente, este actúa otra vez

MÁS INFORMACIÓN SOBRE SIP

El protocolo SIP está descripto en la RFC 3621. Para profundizar la información sobre él, sería importante leer al menos las primeras 80 páginas de esta recomendación, que es una introducción. Podemos encontrar la RFC en www.ietf.org/rfc/rfc3261.txt.

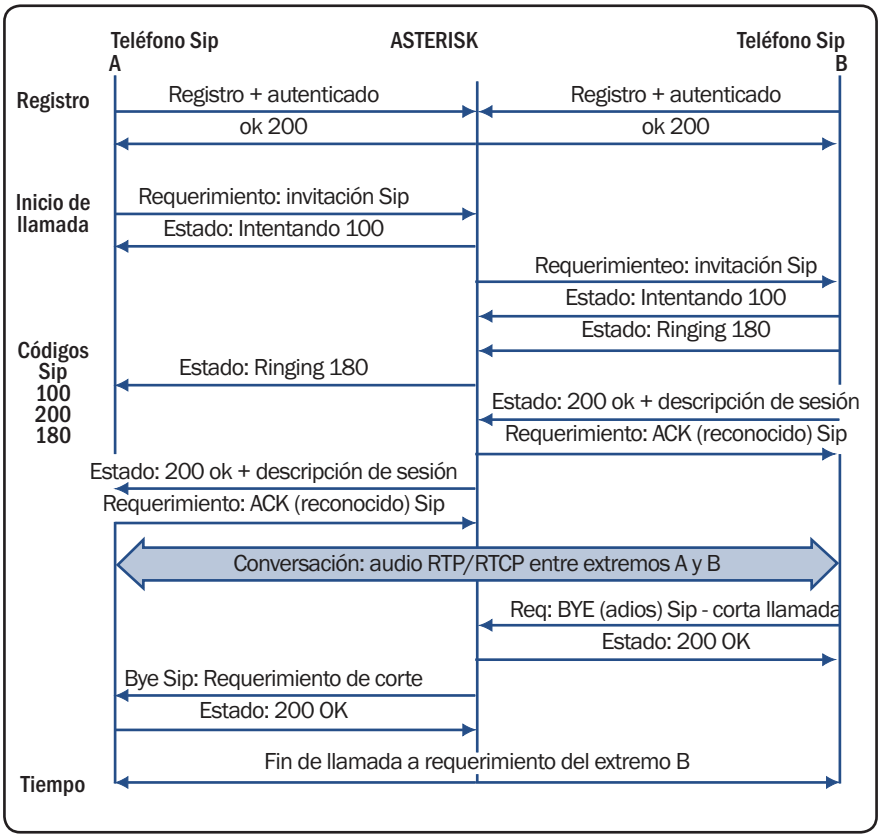


Figura 1. En el esquema se observa el diagrama de estados del protocolo.

como agente de usuario y establece otra conexión (canal) al softphone. Los datos irán directamente a través de Asterisk, así es que ambos terminales están hablando directamente con él.

Tendremos que configurar el dispositivo SIP para que envíe las llamadas a la central Asterisk, aun si este puede realizar y mantener la llamada directamente contra la terminal contraparte.

Infraestructura **requerida**

En instalaciones donde se empleen aparatos telefónicos IP conectados a la red, será de gran utilidad contar con una serie de servicios de red que harán que la puesta en servicio y el mantenimiento de los aparatos sean más simples. Los

servicios que se encargan de realizar estas funciones son **DHCP** y **FTP/TFTP**.

Por lo general, la configuración de los teléfonos IP puede ser muy compleja, hecho que se ve agravado por las interfaces empleadas, que distan de ser intuitivas. Además, la configuración depende del fabricante y suele ser propietaria. Así, si utilizamos varias marcas de teléfonos IP (algo que podemos hacer, dado que en este ambiente podemos alcanzar un alto grado de compatibilidad), deberemos desarrollar una estrategia distinta para cada una. Por todo esto, para simplificar la configuración de los teléfonos IP, es conveniente desarrollar una estrategia de autoconfiguración, para lo cual se requieren los dos servicios, cuyo funcionamiento explicaremos a continuación.

Los teléfonos IP vienen con una configuración por defecto que permite obtener su dirección IP de manera automática mediante el protocolo DHCP. Entonces, cuando conectamos un teléfono de este tipo a la red (como si se tratara de una PC), este comienza a requerir atención de un servicio DHCP para obtener su dirección. Una vez lograda, el teléfono se dispone a buscar el archivo de configuración y las actualizaciones de software en el servidor de archivos provisto por el servicio junto con la dirección IP y otros parámetros, como la puerta de enlace. Esta transferencia se realiza mediante el protocolo **TFTP** o **FTP**, según lo requiera el aparato.

De este modo, lo único que debemos hacer para asignarle un teléfono a un usuario es conectarlo en un puerto de red habilitado para tal efecto, como hacemos con una computadora. Su lógica interna y los servicios descriptos harán el resto.

Escapa al alcance de este libro analizar los pormenores de la instalación de los servicios, y diremos que no son necesarios en lo que a la telefonía atañe. En el caso de emplear un softphone en una computadora, este se configurará desde su propia interfaz.

Configuración básica de Asterisk para un teléfono SIP

Los teléfonos SIP y el sistema Asterisk tienen una gran cantidad de opciones de

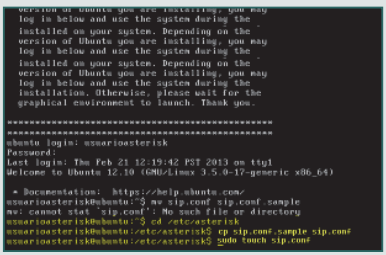
TRANSMISIÓN Y CONMUTACIÓN

El área de la telefonía se separa en dos grandes sectores: la transmisión y la conmutación. Asterisk está ubicado en el sector de conmutación, en tanto que habrá aspectos de transmisión asociados con el hardware que conectemos en el servidor.

configuración, que pueden generar cierta confusión al momento de poner un aparato en servicio. Sin embargo, Asterisk no requiere demasiado para conectarse con un teléfono SIP. Como primera tarea, lo que debemos hacer es configurar un canal SIP mediante la manipulación del archivo **sip.conf**. A continuación, en el Paso a Paso 1, explicamos cómo poner rápidamente en servicio un aparato telefónico IP.

PASO A PASO /1
Configuración básica de un canal SIP

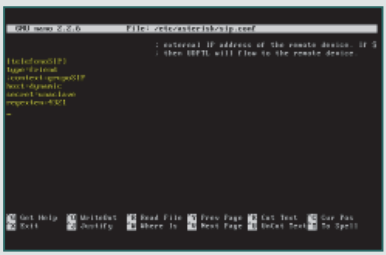
1



Copie el archivo de configuración de muestra: **# mv sip.conf sip.conf.sample**

 # touch sip.conf

2



Edite el nuevo archivo **sip.conf** y complételo con estos datos:

```
[general]
[telefonoSIP]
type=friend
context=grupoSIP
host=dynamic
secret=unaclave
```

Antes de continuar, tenemos que hacer algunas aclaraciones sobre las denominaciones usadas en este sistema. Para identificar el teléfono con un nombre, podemos utilizar cualquier cadena de caracteres. Una práctica muy difundida es identificar los teléfonos por su dirección MAC, dado que es un valor único, en tanto que el nombre de un usuario, el número de interno o una marca pueden resultar ambiguos y generar problemas y confusiones durante el mantenimiento de la red.

En Asterisk, lo único que importa en cuanto a denominaciones es el nombre del canal. No existen los conceptos de usuario, interno o extensión. Por otro lado, una **extensión** tiene un significado totalmente distinto del que estamos acostumbrados a manejar los usuarios de telefonía. En este caso, es el nombre que se le da a un grupo de instrucciones en el script de programación de la central, y puede ser tanto un número como una cadena de caracteres alfanumérica.

Ahora podemos seguir adelante. La configuración del canal SIP nos permitirá realizar y recibir llamadas en el aparato identificado como **teléfono SIP**, que pertenece al context **`grupo`** **`SIP`** (veremos más sobre el context en los Capítulos 4 y 5), con un registro dinámico de su dirección IP y con una clave de uso.

Es importante destacar que la tendencia actual es emplear softphones, ya sea como una aplicación en la computadora o en un smartphone. Un ejemplo de esto es el avance de la telefonía a través del popular **Skype**.

Un aparato de teléfono IP que emplee SIP tendrá su propia forma de configurarse a través de una interfaz, compuesta por una pantalla en el mismo teléfono y su teclado. No describiremos las diferentes posibilidades en que esto se presenta, porque excede el objetivo de este libro, pero sí diremos que tendremos que ajustar las definiciones del archivo **`sip.conf`** con los parámetros de configuración del teléfono; en especial, el nombre del dispositivo, que será empleado como credencial al momento de registrarse y solicitar hacer una llamada. Por esta razón es fundamental que este nombre sea único en la red.

El archivo **`sip.conf`** copiado al directorio **`/etc/asterisk`** contiene gran cantidad de opciones y documentación, pero las líneas más

PROTOCOLOS DE SEÑALIZACIÓN

Se llama protocolo de señalización telefónica al conjunto de reglas y procedimientos que permite establecer, mantener y finalizar una llamada. Hay numerosos protocolos, y SIP es uno de ellos. Otro famoso es el denominado Skinny, de CISCO.

importantes que permitirán un funcionamiento adecuado, junto con las agregadas, son:

```
[general]

context=default

allowoverlap=no

bindport=5060

bindaddr=0.0.0.0

srvlookup=yes
```

El parámetro **srvlookup** debe colocarse en **yes** para que la llamada SIP pueda realizarse utilizando los nombres de dominio de los usuarios SIP en Internet. La sección **[general]** se emplea para las configuraciones globales; también existen otras secciones para configuraciones puntuales.

Para ver el estado del canal SIP, podemos ingresar el siguiente comando en el CLI:

```
CLI> sip show settings
```

Dispositivo **softphone**

Los softphones se encuentran en gran cantidad de instalaciones Asterisk, ya que son simples y cumplen muy bien su función. Además, existe una amplia oferta en el mercado de estos productos de descarga libre.

Suponiendo que empleamos la dirección IP **1.1.1.1/24** para el servidor Asterisk y la dirección IP **1.1.1.10/24** para el softphone, y tomando el ejemplo de los parámetros que utilizamos en los apartados anteriores, los datos que necesitaremos para configurar un softphone serán los siguientes:

- **Display Name**: SuNombre
- **Username**: telefonoSIP
- **Autorization User**: telefonoSIP
- **Password**: ****** (clave)
- **Domain/Realm**: 1.1.1.1
- **SIP Proxy**: 1.1.1.1

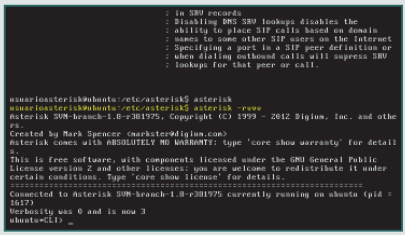
Si queremos observar el estado de registro del softphone que estamos configurando, podemos proceder de la siguiente manera:

CENTRAL TELEFÓNICA

La tarea principal de una central telefónica es encaminar las llamadas para interconectar el nodo que origina la llamada con su destino. De esta manera, una central conecta ambos extremos en forma dinámica.

PASO A PASO /2
Monitoreo de canal y registro

1



Arranque Asterisk en consola remota y nivel verbose 3:

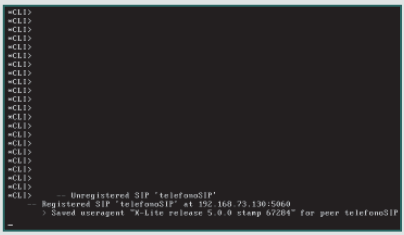
`# asterisk -rvvv.`

En el prompt, reinicie los componentes:

`> dialplan reload`

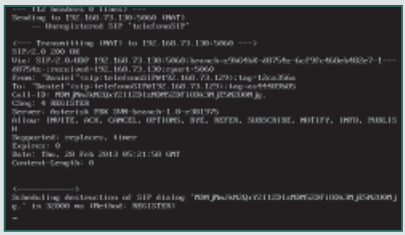
`> sip reload`

2



En la ventana principal del softphone verá la negociación con Asterisk para realizar el registro. Cuando lo logre, verá en el prompt del CLI de Asterisk lo que se muestra en la imagen.

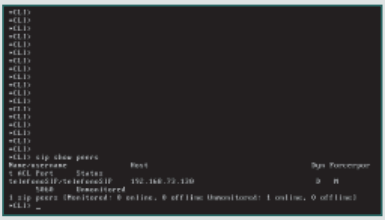
3



En caso de habilitar el parámetro **`sipdebug`** en el archivo de configuración **`sip.conf`**, el despliegue de información acerca del proceso de registro es mucho más rico. El mensaje **`SIP/2.0 200 OK`** muestra que la conversación progresa adecuadamente.

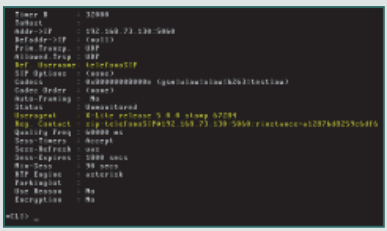
PASO A PASO /2 (cont.)

4



Para verificar el estado de la registración, utilice el comando: `> sip show peers`

5



Para obtener más detalles, emplee el comando: `> sip show peer telefonoSIP`

Canales y Dialplan

Hemos visto que la configuración de los canales en Asterisk se logra a través de los archivos de configuración, cuyo nombre coincide con el protocolo utilizado por el canal; para nuestro caso, el archivo **sip.conf**, que está ubicado en el directorio **/etc/asterisk**. En este archivo configuramos los parámetros relacionados con el canal SIP y los dispositivos SIP que empleamos en la instalación.

En una instalación comercial, encontraremos otros archivos de configuración de canales,

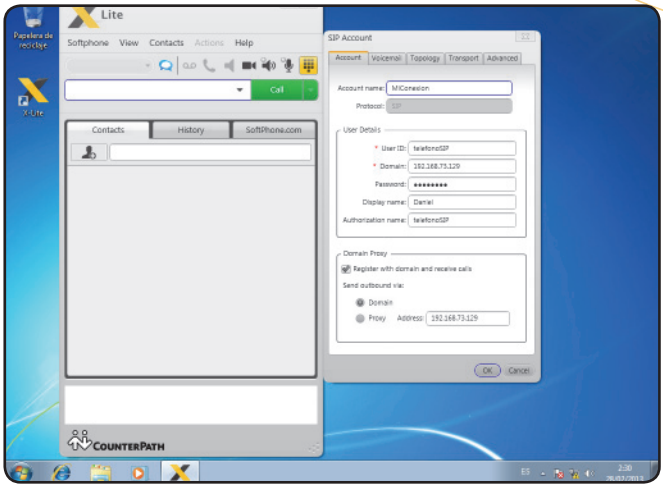


Figura 3. Una vista del `X-Lite` y su ventana de configuración básica con los parámetros utilizados en la instalación de prueba.

como **`IAX2`**, que es un protocolo de Asterisk con funciones similares a SIP. A través de estas configuraciones, Asterisk puede comunicarse con los dispositivos, lo que a su vez permite que estos se comuniquen entre ellos.

Los archivos de configuración de canales contienen la configuración del driver del canal (**`chan_sip.so`**) y se estructuran en secciones, cuyos identificadores se encierran entre corchetes. Por ejemplo, en la sección **`[general]`** está la configuración global del canal y por defecto de los dispositivos, que pueden ser reemplazados por otros específicos que figuren en la sección correspondiente, más abajo en la estructura del archivo.

Los parámetros de cada canal son evaluados por Asterisk en el orden que detallamos a continuación:

1. Evaluación de la sección específica.

2. Evaluación de la plantilla de la sección (si existe).

3. Evaluación de la sección **`[general]`**.

4. Asignación de los valores por defecto.

El archivo de configuración del canal interactúa con el script de control de la central (**Dialplan**), y así queda definido desde el punto de vista lógico de qué manera la llamada ingresa y sale de la central.

En nuestro caso, nos hemos concentrado en el canal SIP, dado que es uno de los más utilizados y estables, y tiene un gran soporte de parte de la comunidad Asterisk. Además, las tendencias actuales marcan que los dispositivos sean teléfonos IP y, si es posible, softphones.

No obstante, será necesario interconectar la central Asterisk con el mundo exterior para poder realizar y recibir llamadas. Esto requerirá la configuración de hardware específico y de otros canales, que no trataremos en este libro, pero que siguen la misma lógica explicada y no revisten un grado de dificultad que los haga inabordables para quienes tienen conocimientos de telefonía e informática.

Como vimos anteriormente, el archivo **sip.conf.sample** de Asterisk fue copiado con el nombre **sip.conf** en el directorio **/etc/asterisk**. A partir de este archivo, que tiene una gran cantidad de opciones de configuración, de las que normalmente bastarán las que están por defecto, construiremos nuestro archivo de configuración de canal SIP.

En el siguiente listado, podemos ver las opciones consideradas fundamentales a la hora de poner en funcionamiento el canal SIP y que serán necesarias, como datos, para la configuración de los dispositivos SIP:

[general]

context=unauthenticated; contexto por defecto para las llamadas entrantes

allowguest=no; deshabilitar las llamadas no autenticadas

srvlookup=yes; habilitar búsqueda DNS para llamadas salientes

udpbindaddr=0.0.0.0; escuchar UDP en todas las interfaces

tcpenable=no; deshabilitar TCP

[office-phone](!); crear una plantilla de dispositivos

type=friend; se valida con el nombre de usuario primero, y luego con su IP

context=LocalSets; punto de ingreso de las llamadas en el Dialplan

host=dynamic; el dispositivo se registra con Asterisk

nat=yes; está en red privada con NAT

secret=unaclave; clave de seguridad (use la propia)

dtmfmode=auto; acepta tonos DTMF y negocia en auto

disallow=all; inicializa los códecs aceptados

allow=alaw; códec de audio aceptado/requerido para definir nombres de dispositivo que usa la plantilla

[0A0B1111ED0B](office-phone)

[0A0BECDBAFB4](office-phone)

Los comentarios aclaran el significado elemental de cada parámetro y algunos de ellos ya han sido explicados. De los que no vimos, algunos merecen un párrafo aparte. Por ejemplo, el parámetro **allowguest=no** indica que no serán aceptadas las llamadas que no posean autenticación. Es una buena práctica definir un contexto para tratar por defecto las llamadas sin autenticación. Esta definición se hace, como se ve en el listado, definiendo un **context** en la sección **[general]**. De esta manera, podemos controlar las llamadas que se realizan desde dispositivos ajenos a nuestra instalación, para evitar que se efectúen utilizando nuestras líneas y a nuestro costo.

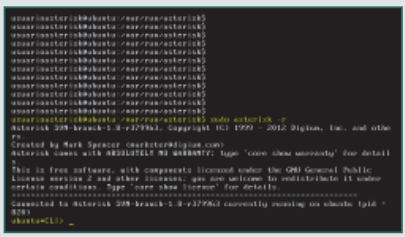
Otra cuestión interesante está relacionada con la creación de **plantillas** o **templates**, para no tener que escribir una y otra vez los mismos parámetros en situaciones como la definición de dispositivos, donde lo único que cambia es el nombre. Estas plantillas se definen con un signo de exclamación (**!**) a continuación del nombre de sección. Esto le indica a Asterisk que debe considerar esa definición como una plantilla, que deberá aplicar cuando sea invocada. En los próximos capítulos veremos algún ejemplo ilustrativo de esta característica, muy útil al programar las centrales Asterisk.

COMANDOS DE ASTERISK

Para finalizar este capítulo, conoceremos algunos comandos de Asterisk relacionados con la configuración de los canales. Para esto, debemos seguir el **Paso a Paso** que presentamos a continuación.

PASO A PASO /3
Configuración de canales

1

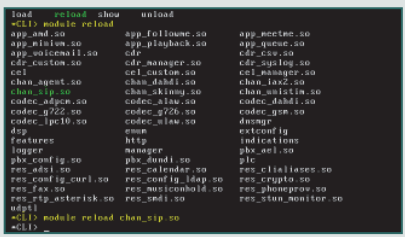


Conéctese con Asterisk a través del comando:

```
#sudo asterisk -r
```

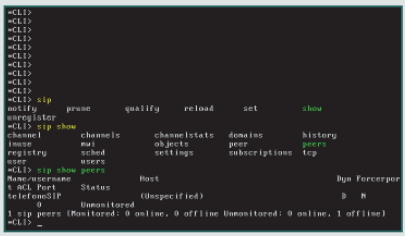
PASO A PASO /3 (cont.)

2



Reinicie el módulo del canal SIP mediante el comando: `CLI> module reload chan_sip.so`

3



Verifique el estado del canal configurado, ingresando el siguiente comando: `CLI> sip show peers`

RESUMEN

En este capítulo analizamos el concepto de canal y su relación con los dispositivos. Repasamos los componentes de red necesarios para una instalación con teléfonos IP. Exploramos la configuración del tipo de canal SIP y la de un softphone de uso popular. Finalmente, vimos algunos comandos Asterisk relacionados.

Capítulo 4

Programación Asterisk I

Veremos la programación básica y necesaria de Asterisk para crear un sistema funcional.

RedUSERS.com

Conceptos sobre **el Dialplan**

El motor de los sistemas de telefonía tradicionales y propietarios es un dominio cerrado y reservado para el fabricante y, hasta donde este quiera, a los socios de negocios, distribuidores y técnicos de soporte. Además, la mayoría de los parámetros que controlan su comportamiento vienen dados de fábrica y no es posible modificarlos, salvo mediante costosas actualizaciones del software del sistema.

Por el contrario, el corazón de Asterisk, el Dialplan, es abierto y totalmente configurable. Consiste en una especie de lenguaje de programación del tipo scripting, muy flexible y vasto, pero que posee cierta estructura, que es la que estudiaremos en este capítulo. Permite escribir un sistema manejado por eventos, en general externos, que especificará cómo serán tratadas por el sistema las llamadas telefónicas.

Su implementación física se realiza a través del archivo de configuración **extensions.conf**, que, como otros archivos de este tipo, se encuentra, por defecto, en el directorio **/etc/asterisk**.

La sintaxis del Dialplan está formada por cuatro elementos:

- **context** (contexto)
- **extensions** (extensiones)
- **priorities** (prioridades)
- **applications** (aplicaciones)

La estructura básica del Dialplan se compone de una secuencia de contextos, separados por la directiva **[context]**, donde la palabra **context** se reemplaza por el identificador, por ejemplo: **[internos]** puede ser el identificador del contexto que agrupe parámetros y directivas que se aplicarán al tratamiento de las llamadas dirigidas a los internos o entre estos.

La función principal de un contexto es agrupar los elementos, directivas, etcétera, que se encuentran en él y aislarlos de los otros contextos. De esta manera, podremos mantener la independencia entre ellos, y lo que suceda dentro de un contexto no tendrá impacto ni relación con lo que suceda en otro, salvo que lo definamos o programemos explícitamente. La cadena de caracteres que forma el identificador del contexto puede contener cualquier carácter alfanumérico, guiones medio y bajo, sin espacios, hasta un máximo de ochenta contando el NULL final. Los siguientes nombres están reservados para su uso por Astrerisk:

- **[general]**: agrupa la configuración de los parámetros que se aplicarán a todo el Dialplan.
- **[global]**: se hace cargo de las variables globales del script.
- **[default]**: se ocupa de los valores por defecto de los parámetros empleados por todos los segmentos del Dialplan.

A continuación del identificador de sección **[globals]**, se agrupan los parámetros que actúan como globales para el funcionamiento del Dialplan. Estos valores tienen una cobertura

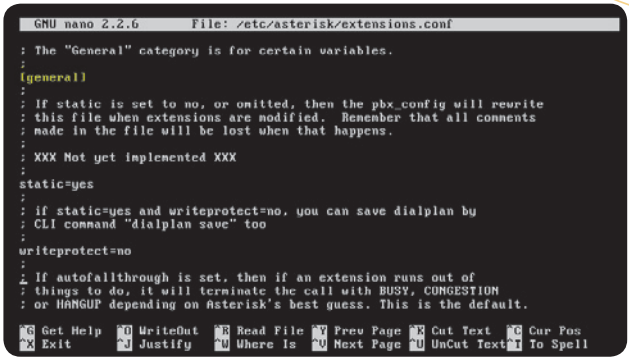


Figura 1. Debajo del identificador `[general]` se dan pautas para su configuración.

sobre todo lo que hagamos en el Dialplan, y algunos de ellos podrán redefinirse a nivel de los contextos particulares, con valores que solo tendrán cobertura dentro de ellos.

El archivo de configuración del Dialplan contiene una serie de preconfiguraciones establecidas. Entre ellas se encuentra el contexto **[default]**, que es el único que existe si no se declara otro. Las configuraciones de las extensiones se realizarán dentro de él, porque las aplicaciones, de manera predefinida, se refieren a él. Cabe destacar que, en muchas instalaciones, este contexto será más que suficiente para definir el ambiente telefónico utilizado. En nuestro caso, pusimos como ejemplo otro contexto, llamado **[internos]**, bajo el cual podremos incluir todas las configuraciones de prueba necesarias sin alterar el estado original del sistema.

Estos segmentos de script no forman contextos estrictamente hablando, pero agrupan elementos comunes y aislados del resto de la programación, así que, por extensión del

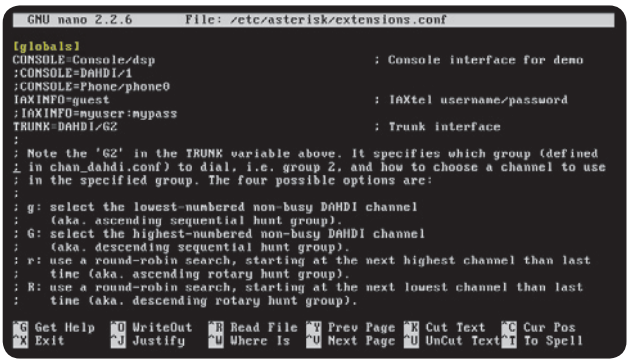


Figura 2. Las líneas que se encuentran comentadas pueden utilizarse quitando el punto y coma (;).

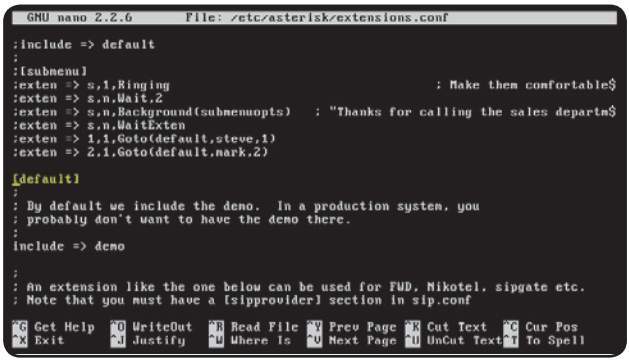


Figura 3. Aquí se muestra el identificador de la sección que marca el contexto por defecto donde se ubicarán las extensiones.

lenguaje empleado en el Dialplan, se los trata como contextos.

La segunda función para destacar de un contexto es mantener la seguridad: forma compartimentos de código estanco donde es posible separar las funcionalidades y privilegios dados, unos de otros. Mantener esta seguridad puede hacernos ahorrar dinero, pues permite asegurarnos de que los permisos otorgados para hacer determinados tipos de llamadas no estarán fuera de control.

Por último, debemos decir que la etiqueta **`[context]`** enlaza las definiciones realizadas en los archivos de configuración de canal con las correspondientes del Diaplan, es decir, en el archivo **`extensions.conf`**. Esta relación es muy importante porque permite definir el flujo de las llamadas a través del sistema Asterisk, determinando en qué parte del Dialplan está programada la lógica que atenderá una llamada que ha ingresado por un canal determinado. En el esquema de la Figura 4 puede apreciarse esta relación.

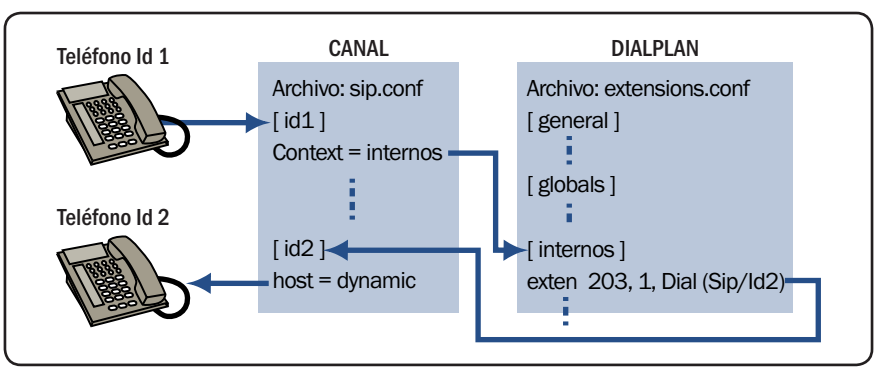


Figura 4. Comprender esta relación es clave para entender el funcionamiento de Asterisk.

Sintaxis del Dialplan

Dentro de cada contexto hay una secuencia de líneas estructuradas de la siguiente manera:

```
exten => nombre,prioridad,aplicación()
```

Como ejemplo, mostramos la siguiente línea:

```
exten => primera,1,Answer()
```

Aquí, el nombre de la extensión es **primera**, la prioridad es **1** y la aplicación es **Answer()**, que, como podremos imaginarnos, indica que hay que responder a esta llamada.

La directiva **exten=>** se refiere a **extensión**, que en Asterisk se emplea para identificar una línea de instrucción que contiene a los otros dos elementos de la sintaxis: la **prioridad** y la **aplicación**. Como vemos, la línea de programa tiene un nombre que a veces se confunde con el número que tenemos que digitar o discar para que el teléfono perteneciente a un determinado usuario comience a sonar, avisando que alguien está llamando. El nombre es cualquier cadena alfanumérica y sirve de etiqueta para identificar el paso del programa o Dialplan en cuestión.

Luego, separado por una coma, pondremos un número llamado **prioridad**. Esto también tiende a generar confusión, ya que no se trata de una prioridad para cursar la llamada ni de un identificador de privilegio, sino del número de orden del paso de programa en cuestión para una extensión dada.

En cada contexto, podemos tener muchas extensiones, y cada una estar compuesta, a su vez, por un determinado número de pasos, distinguidos por una prioridad, como vemos en la porción del Dialplan siguiente:

```
...

exten => primera,1,Answer()

exten => primera,2,aplicación1()

exten => primera,3,otra-aplicación()

exten => primera,4,Hangup()

...
```

ARCHIVOS DE CONFIGURACIÓN

Al implementar Asterisk, hemos instalado los archivos de configuración de muestra en el directorio: `/usr/src/sistemaasterisk/asterisk/1.8/configs`. **Podemos comenzar a trabajar a partir de los archivos allí instalados o crear los propios desde cero.**

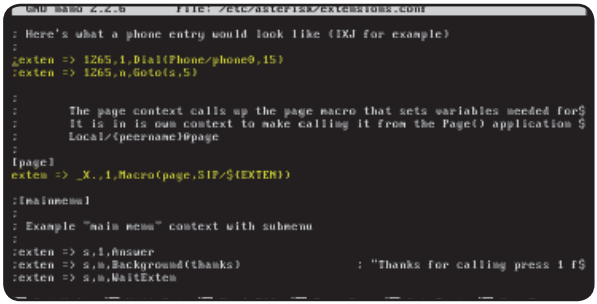


Figura 5. El archivo `extensions.conf`, donde se ven algunas configuraciones por defecto.

En este ejemplo, la extensión lleva por nombre **primera**, y su tratamiento se compone de cuatro pasos o prioridades, que serán ejecutadas en ese orden, para dar el tratamiento definido para esa extensión. Este script se ejecutará al ser disparado por una llamada, o sea, por el discado de un número de teléfono o interno que corresponderá a un destino al que se quiere llamar.

Antes de explicar el último elemento de la línea de programa del Dialplan, cabe destacar que, para facilitar la programación y evitar la redundancia de escribir una y otra vez **exten => primera**, con el inconveniente que acarrea la numeración consecutiva al momento de tener que insertar alguna línea intermedia, se ha provisto al lenguaje de dos opciones, que se muestran en la siguiente porción del Dialplan:

```
...

exten => primera,1,Answer()

same => n,aplicación1()

same => n,otra-aplicación()

same => n,Hangup()

...
```

same => repite lo que se ha definido en la línea anterior, evitando escribir lo mismo tantas veces como pasos hayamos diseñado. El parámetro **n** transforma en dinámica la prioridad de la línea, con lo cual permite intercalar o modificar la secuencia sin renumerar el resto de las líneas.

VARIABLES DE CANAL

Asterisk tiene muchas variables de canal predefinidas que pueden emplearse en el Dialplan. Para acceder a mayor información sobre ellas podemos consultar la wiki de Asterisk, en: https://wiki.asterisk.org/wiki/display/AST/Channel+Variables.

Algo importante es no olvidarnos de numerar con **1** la primera línea de programa de la extensión, o esto no servirá de nada. Asterisk calcula las prioridades incrementando en uno el resultado anterior, y por eso se requiere tener la referencia de la primera línea; de lo contrario, no podrá expandir el código como corresponde.

Como en todo lenguaje de programación, el Dialplan cuenta con estrategias para controlar el flujo del programa. Todas ellas, implementadas con directivas, necesitan conocer los puntos de acceso del flujo para poder dirigirse a él y retornar al punto de partida, si es necesario. Para esto, en programación es usual emplear el concepto de **label** o **etiquetas**, que identifican estos puntos destacados en el flujo de programa. Este concepto se implementa en el Dialplan agregando un parámetro más a la prioridad. El formato general para la extensión de ejemplo será el siguiente:

```
...

exten => primera,n(label),application()

...
```

De esta manera, podemos etiquetar una línea determinada con el propósito de referirnos a ella en otros pasos de la misma extensión, en pasos de otras extensiones o, incluso, desde otros contextos. Así es posible saltar de un punto a otro a la manera **GoTo**.

El último elemento que conforma la sintaxis, **aplicación()**, determina la acción que se realizará cuando se ejecute la línea en cuestión. Este elemento es el más rico y complejo de todos y, por lo tanto, le reservamos un apartado dedicado a su presentación y explicación. Antes examinaremos un poderoso concepto de todo lenguaje de programación, que facilita la codificación, permite programar con mayor elegancia, y ayuda a reducir el tamaño del programa, las variables y los patrones.

Variables en Asterisk

Una **variable** es una especie de contenedor de valores, donde estos pueden cambiar, pero el contenedor (su definición o identificación) no. Es un concepto que quienes han programado

INSTRUCCIONES PRECISAS

Asterisk requiere que se le explicite qué debe realizar y cómo debe proceder con cada situación que se le presente en cuanto al manejo de las llamadas; de otro modo, las desconectará. Este comportamiento es distinto de las centrales PBX propietarias.

alguna vez comprenderán rápidamente. Esta herramienta no se encuentra en la programación de centrales PBX propietarias y será muy útil durante la implementación de la lógica de atención de llamadas.

En general, lo que cambia entre los diferentes lenguajes es la forma de definir estas variables y la recuperación de los valores almacenados en ellas. También, como en muchos sistemas, hay nombres reservados por la misma plataforma para usos propios. Las variables del sistema están definidas en mayúsculas y no deben ser utilizadas por el programa de usuario. Veremos mediante ejemplos la sintaxis en Asterisk:

...

```
exten => 2000,1,Set(JOSE=SIP/0ADCBB
0AEFBC)

same => n,Dial(${JOSE})
```

...

En esta porción de código, en la extensión **2000**, hemos declarado e inicializado la variable **JOSE** (la variable **jose** es distinta de **JOSE**), y en la siguiente línea la hemos utilizado, invocando su contenido, como argumento destino de la aplicación **Dial()**, que veremos en detalle en el apartado correspondiente a las aplicaciones.

Suponemos que José tiene un terminal SIP y lo hemos identificado con la variable **JOSE**. Si José cambia su terminal, solo deberemos cambiar el valor asignado a la variable, y no, en cada lugar del Dialplan donde hayamos empleado la dirección de su terminal. Esto facilita el mantenimiento y disminuye la probabilidad de errores humanos en el programa.

Como vemos en la segunda línea del ejemplo, el contenido de la variable se invoca encerrando entre llaves su valor y anteponiendo el signo **$** al conjunto.

En Asterisk se definen tres tipos de variables, que detallamos a continuación:

- Variables globales
- Variables de canal
- Variables de entorno

ACCESO A ASTERISK I

Hay tres maneras de ejecutar Asterisk: `#asterisk` corre como un demonio y devuelve el control al shell de Linux; `#asterisk -c` se ejecuta y devuelve una consola local para su manejo (*CLI>); y `# asterisk -r` se conecta en forma remota a un Asterisk que se está ejecutando en otra máquina.

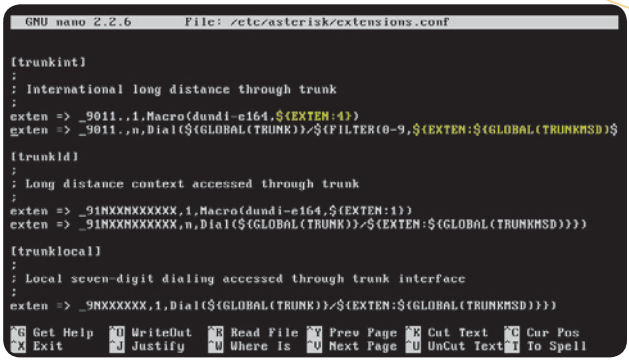


Figura 6. Ejemplos de uso de variables en el archivo `extensions.conf` preconfigurado.

Las **variables globales** son visibles por todos los canales definidos en Asterisk y deben declararse en el contexto **`[globals]`** al principio del archivo **`extensions.conf`**:

```
[globals]

JOSE=SIP/0ADCBB0AEFBC

...
```

Por el contrario, las **variables de canal** solo son visibles en los canales donde se han declarado y son válidas durante el desarrollo de la llamada que involucra estos canales solamente. Se inicializan mediante la aplicación **`Set()`**:

```
...

exten => 1032,1,Set(Intentos=10)

...
```

Las **variables de entorno** se refieren a las homónimas que encontramos en la plataforma Linux. Si bien no son muy utilizadas, Asterisk proporciona una manera de acceder a las variables de entorno de Linux, mediante la aplicación **`ENV()`**.

Su sintaxis es **`ENV(var)`**, donde **`var`** es la variable de entorno Linux referenciada, y su contenido es indicado como: **`${ENV(var)}`**.

ACCESO A ASTERISK II

Cuando invocamos la ejecución de Asterisk, tenemos la oportunidad de establecer el nivel de diálogo informativo (verbosidad). Por ejemplo, para invocar a Asterisk con una consola de comando local y nivel 2, existe el siguiente comando: `# asterisk -cvv`.

Patrones en Asterisk

Otra herramienta muy poderosa dentro de un lenguaje de programación es la posibilidad de definir en forma dinámica valores que no se conozcan a priori o que sean rangos extensos o definidos al momento de ejecutar una aplicación.

El emparejamiento de valores se define a través de un guión bajo ubicado delante de ciertos caracteres o términos, como podemos observar en la Tabla 1.

Para ejemplificar este concepto veremos, a continuación, algunos usos en el Dialplan:

...

```
exten => _ZXN,
Playback(archivo-de-audio)
```

...

Mediante el patrón **_ZXN**, hacemos referencia a una extensión que comience con cualquier número entre 1 y 9, siga con cualquier número entre 0 y 9, y finalice con cualquier número entre 2 y 9. Asterisk realiza la evaluación de

REFERENCIA A VALORES

CARÁCTER	DEFINICIÓN
. (punto)	Es un carácter comodín, que representa uno o más dígitos cualesquiera.
! (signo de exclamación)	Otro comodín, pero a diferencia del anterior, representa a ninguno o más caracteres cualesquiera.
X	Representa cualquier dígito entre 0 y 9.
Z	Representa cualquier dígito entre 1 y 9.
N	Representa cualquier dígito entre 2 y 9.
[03-7]	Representa una serie de dígitos o rango de ellos. Aquí representa el 0 y un número entre 3 y 7.

Tabla 1. **Estos caracteres permiten hacer referencia a valores desconocidos.**

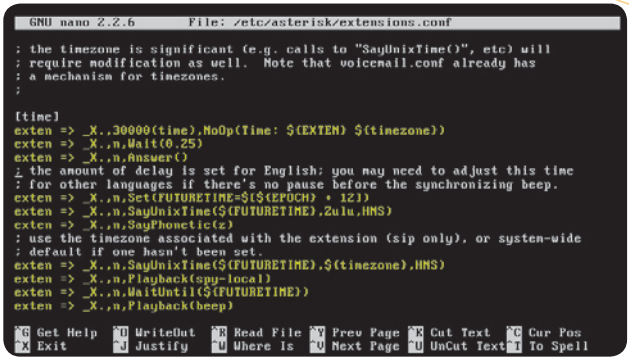


Figura 7. Ejemplos de uso de patrones en el archivo `extensions.conf` preconfigurado.

estos patrones de la misma manera que los motores de ruteo lo hacen con las rutas a las redes. Si dos patrones se parecen, Asterisk tomará el más específico.

Aplicaciones

Las **aplicaciones** son las rutinas que se ejecutan para realizar las acciones requeridas por la lógica del Dialplan. Su estructura recuerda a las llamadas a procedimiento del lenguaje C, dado que se invocan con un identificador seguido de paréntesis, dentro de los cuales, en caso de que existan, se sitúan los argumentos que deben pasarse a estas llamadas, separados por coma (**,**) o barra vertical (**|**). Los nombres que utilizamos en el apartado anterior son verdaderos identificadores del Dialplan de Asterisk.

En la siguiente porción del Dialplan (que puede ser lo único que tengamos en él, tratándose entonces de la lógica más simple que podamos implementar), vemos el uso de tres de las aplicaciones más simples (en el desarrollo de los próximos capítulos, introduciremos otras más):

...

`[internos]` ; nombre del contexto (será referenciado desde el archivo **`tipo_canal.conf`**)

...

```
exten => 70,1,Answer()

same => n,Playback(archivo-de-audio)

same => n,Hangup()
```

En la primera línea, cuando el canal asociado con el identificador del contexto donde se encuentra esta porción del Dialplan está llamando (**ringing** en la jerga telefónica), se emplea la aplicación **`Answer()`** (no requiere argumentos) para atenderlo y conectarlo. Esto es necesario si luego vamos a tomar más acciones sobre esta llamada, como en este caso.

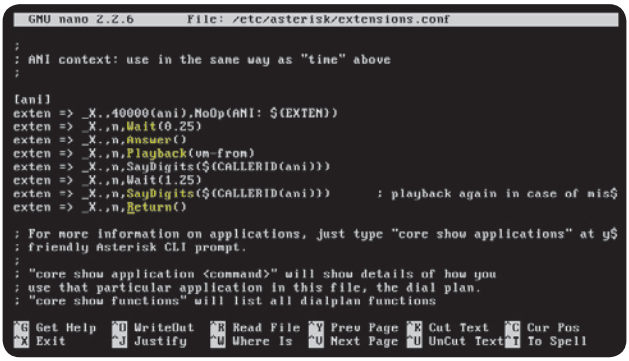


Figura 8. Ejemplos de uso de aplicaciones en el archivo `extensions.conf` preconfigurado.

En la siguiente línea, la aplicación **`Playback(ruta/archivo-de-audio)`** reproducirá sobre el canal conectado por la aplicación **`Answer()`** un archivo de audio designado como argumento. Este argumento puede indicarse como una ruta de directorios completa o como el nombre del archivo con extensión o sin ella. En caso de tener varios archivos de sonido iguales, pero con diferentes codificaciones (estos se manifestarán con diferentes extensiones en el nombre del archivo), Asterisk tomará el archivo más adecuado, sobre la base del uso de los recursos. En este caso, irá a buscarlos al directorio de sonidos por defecto, que en una instalación estándar es **`/var/lib/asterisk/sounds/`**.

Finalmente, en la última línea de este contexto se emplea la aplicación **`Hangup()`**, que en general no utiliza argumentos para desconectar el canal de la llamada en curso. Esto asegura una finalización adecuada de la llamada y evita dejar conectados los canales a la lógica del Dialplan, hecho que podría producir consecuencias desagradables en el funcionamiento del sistema. Por esto último, se recomienda el uso de esta aplicación al final del contexto para terminar la rutina incorporada en él.

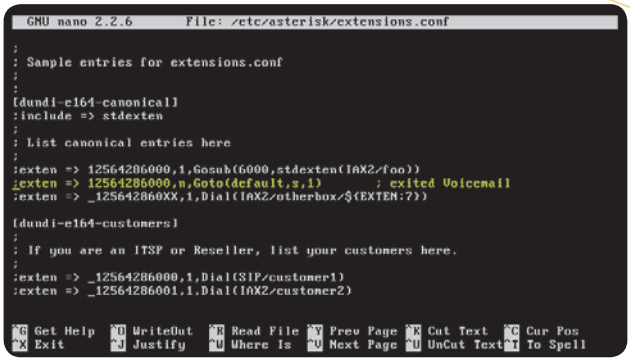
Saltos **incondicionales**

Tal como en otros lenguajes de programación, en el de Asterisk también encontramos sentencias para romper el flujo secuencial de instrucciones. Veamos el formato de esta instrucción:

```
same => n,Goto(context,extension,priority)
```

La sentencia **`Goto`** realiza un salto incondicional al contexto indicado como primer argumento y, dentro de él, a la extensión indicada en el segundo argumento y al número de secuencia de dicha extensión, indicada por el tercero. Si solo incluimos un argumento, el

Figura 9. Ejemplos de uso de saltos en el archivo `extensions.conf` preconfigurado.



salto se realizará al número de secuencia, indicado por ese argumento, de la extensión corriente, dentro del mismo contexto. Un segundo argumento daría cuenta de un salto a una secuencia de otra extensión, siempre dentro del mismo contexto.

Para ejemplificar el uso de esta sentencia, utilizaremos el siguiente listado de un Dialplan:

```
[internos]

...

exten => 70,1,Answer()

same => n,Playback(archivo-de-audio)

same => n,Hangup()

exten => 911,1,Goto(Exepciones,emergencias,1)

...

[Excepciones]

exten => emergencias,1,Answer()

same=> n,Playback(archivo-de-audio-recepción911)
```

CONVERSIÓN DE AUDIO

Cuando se debe reproducir un archivo de sonido, Asterisk lo convierte previamente a su formato nativo. Para hacerlo, selecciona el archivo, entre los disponibles, cuyo formato permita esta conversión con el mínimo uso de recursos de la CPU.

Una llamada que ingresa por el contexto **[internos]** a la extensión **911** será atendida en el contexto **[Excepciones]**, donde se reproducirá un mensaje (**archivo-de-audio-recepción911**) y, luego, seguirá el curso según la lógica diseñada para ella. El salto se ha realizado desde el contexto **[internos]** al contexto **[Excepciones]**, a la extensión cuyo nombre de identificador es **emergencias** y en la secuencia **1** de esta.

Aplicación **Dial()**

Podríamos decir que **Dial()** es la aplicación principal de Asterisk porque permite interconectar los extremos que pretenden comunicarse. Además, realiza la conversión de los distintos formatos y protocolos empleados por los canales (analógicos, digitales convencionales, voz sobre IP, etc.). Es una aplicación muy poderosa y versátil, por lo que requiere de algunos argumentos para cumplir su misión. Su sintaxis es la siguiente:

```
Dial(destino,timeout,opción,URI)
```

El primer argumento es indispensable, y representa el destino que se intenta alcanzar, es decir, el extremo al que se está llamando. Este argumento es compuesto y tiene la siguiente estructura:

```
tipo-de-transporte/
dirección-recurso-remoto
```

Como ejemplo, supongamos que al llamar a la extensión 701, queremos comunicarnos con un terminal telefónico IP o un softphone. El tipo de transporte o tecnología de canal empleado será **SIP**. La dirección del recurso estará dada, en este caso, por su identificación o nombre; en el caso de otras tecnologías, como la **DAHDI** para transporte analógico, será el número del canal empleado. El identificador, como vimos en el **Capítulo 3**, deberá ser algo único, que permita designar unívocamente al recurso, como, por ejemplo, su dirección física o **MAC**. Así es como se verá en el Dialplan:

```
exten => 701,1,Dial(SIP/0A00BF4E0C1)
```

Existe una estrategia muy utilizada que se denomina **cobertura** y consiste en llamar a múltiples terminales telefónicos simultáneamente

TRANSPORTE CON IAX2

Asterisk permite utilizar diferentes tipos de transporte, lo que posibilita la interconexión de distintas tecnologías. Un formato que permite la interconexión entre sistemas Asterisk es el denominado IAX2.

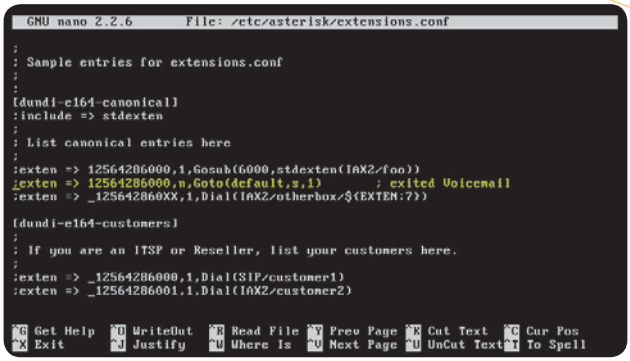


Figura 10. Distintas llamadas a la aplicación `Dial()`. Notemos los diferentes canales empleados.

con el objeto de ser atendidos por el primer agente libre que se presente en el destino llamado, por ejemplo, un centro de atención de reclamos. Con esta aplicación, sin mucho esfuerzo adicional, podremos implementar esta estrategia. La codificación se verá así:

```
exten => SoporteTecnico,1,Dial(SIP/00E
0AB00DCEF&SIP/0A00BF4E0C1)
```

Simplemente, debemos concatenar los destinos mediante el operador **&**. Lo que sucederá cuando se curse la llamada será que todos los destinos concatenados van a sonar (**ringear**, en la jerga telefónica) al mismo tiempo, y cuando uno de ellos sea atendido, los demás dejarán de ser alertados. No desarrollaremos este tema en detalle, pero para tener una idea de la potencia de esta aplicación, mostraremos un ejemplo para el caso de llamar a un terminal analógico externo a la central, empleando la aplicación **`Dial()`**:

```
exten => Soporte,1,Dial(DAH
DI/7/08109998018)
```

Esta llamada será cursada al grupo o canal 7 de un transporte **DAHDI** y al número telefónico 0810-999-8018. El encaminamiento de este destino es responsabilidad de las centrales públicas a las que deberemos interconectar nuestra central Asterisk mediante la tecnología

INFORMACIÓN DE SEÑALIZACIÓN

Algunos sistemas telefónicos solicitan información de señalización durante el establecimiento de las llamadas. Asterisk dispone de la aplicación Progress(), que provee información sobre el progreso de la llamada al canal origen, antes de conectarla con Answer().

adecuada, por ejemplo, una línea telefónica convencional o un troncal **E1**.

En el ejemplo de la estrategia de cobertura, los terminales telefónicos son alertados indefinidamente, y el programa queda estancado en ese punto hasta que algún agente atienda o corte la llamada. Para evitar esto, podemos indicar el tiempo máximo durante el cual Asterisk alertará a los terminales remotos, luego de lo cual se terminará la llamada y se procederá a procesar la siguiente secuencia en la extensión. Esto se efectúa mediante el segundo argumento de la aplicación **Dial()**, el **timeout**, expresado en segundos:

```
exten => 701,1,Dial(SIP/0A00BF4E0C1,8)
```

En nuestro ejemplo, hemos agregado un timeout de 8 segundos, luego del cual se cancelará la acción de la aplicación y se continuará con la siguiente secuencia.

Desde el punto de vista de la persona que llama, este comportamiento puede parecer un poco rudo, puesto que la llamada se cortará sin explicación alguna. Una alternativa más elegante es proporcionar alguna pista de por qué se ha decidido terminar con la llamada. También puede suceder que el destino que se desea alcanzar se encuentre ocupado. Cualquiera sea el motivo, Asterisk lo indicará a través de una variable de estado del sistema llamada **DIALSTATUS**, de la que nos ocuparemos en el próximo capítulo.

Este ejemplo muestra una manera en que podemos resolver esta situación:

```
...

exten => 701,1,Dial(SIP/0A00BF4E0C1,8)

same=>n,Playback(aviso-estoy-ocupado)

same => n,Hangup()

...
```

Luego de superados los 8 segundos, se cancelará la llamada, cesará el alerta al terminal **SIP** y se reproducirá un mensaje de ocupado.

Cabe destacar que podemos poner cualquier mensaje independientemente de la causa del cese del alerta. En el siguiente capítulo, veremos

SEGURIDAD DEL SISTEMA

Como en todo sistema, la seguridad debe tomarse seriamente en Asterisk. Podemos consultar las consideraciones básicas de seguridad para un sistema Asterisk de telefonía en: https://wiki.asterisk.org/wiki/display/AST/Important+Security+Considerations.

una solución más real a esta situación, mediante el uso del estado de la variable **DIALSTATUS**.

El argumento denominado **URI** (**Uniform Resource Identifier**) se emplea para incluir, por ejemplo, la dirección SIP, y no es muy utilizado todavía. En tanto que el argumento **opcion** toma varios valores. El valor **m** es muy popular, y habilita la reproducción de música en espera hacia el terminal llamante, mientras se aguarda que el extremo remoto atienda la llamada.

Si queremos saltear algún argumento, simplemente dejamos un espacio entre dos comas, donde va el argumento esperado, y este tomará el valor predefinido (si existe) o no quedará especificado. Esto es similar a lo que sucede en otros lenguajes de programación.

EJEMPLO DE CONTEXTO DEL DIALPLAN

Concluiremos este apartado con un resumen de las modificaciones que hemos explicado en un ejemplo de contexto del Dialplan.

Supongamos que estamos programando un nuevo contexto dentro de un Dialplan existente y agregamos los conceptos que acabamos de estudiar:

```
...

[internos]

...

exten => 70,1,Answer()

same => n,Playback(archivo-de-audio)

same => n,Hangup()

exten => 911,1,Goto(Exepciones,emerge
ncias,1)

exten => Soporte,1,Dial(DAH
DI/7/08109998018,10)

exten => SoporteTecnico,1,Dial(SIP/00E
0AB00DCEF&SIP/0A00BF4E0C1,30)

same => n,Playback(aviso-estamos-to-
dosOcupados-disculpeLasMolestias)

same => n,Hangup()

exten => 701,1,Dial(SIP/0A00BF4E0C1,8)
```

COMANDO ÚTIL

Para conocer el costo que lleva la conversión de archivos de sonido, en términos de recursos de cómputo, podemos utilizar el comando `show translation` de Asterisk, que reporta los milisegundos consumidos en la transcodificación de un segundo del audio origen.

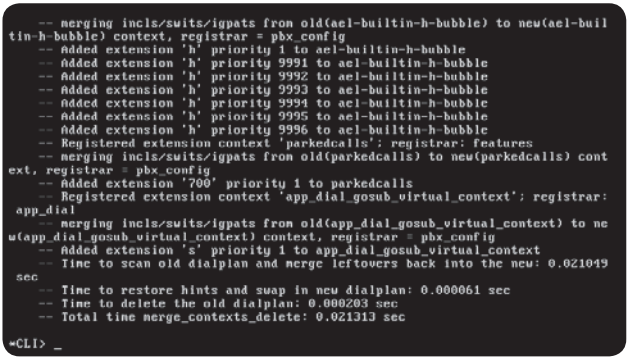


Figura 11. Esta salida podrá observarse iniciando Asterisk con el comando CLI, `asterisk -vvvvc`.

```
same=>n,Playback(aviso-estoy-ocupado)

same => n,Hangup()

...

[Excepciones]

exten => emergencias,1,Answer()

same => n,Playback(archivo-de-audio-recepción911)

same => n,Playback
(archivo-de-audio-saludo)

same => n,Hangup()

...
```

Una vez que hayamos guardado los cambios del archivo **extensions.conf**, para que tengan efecto, debemos recargar el Dialplan. Esto se puede hacer en caliente de dos formas. Desde el prompt de control de Asterisk (**CLI>**):

```
CLI> dialplan reload
```

O desde el shell de Linux:

```
$ sudo /usr/sbin/asterisk -rx "dialplan reload"
```

RESUMEN

El Dialplan, corazón de Asterisk, nos permite implementar la lógica de funcionamiento de la central telefónica. Hemos visto su estructura, sintaxis y componentes básicos, como así también la relación con otro componente principal de Asterisk, los canales.

Capítulo 5

Programación de Asterisk II

Abordaremos los detalles de las estructuras, las funciones y los saltos condicionales.

RedUSERS.com

Introducción

Asterisk presenta estructuras y sentencias similares a otros lenguajes de programación. Lo que en estos se denomina instrucción o sentencia, en Asterisk se implementa mediante las **aplicaciones**, algunas de las cuales hemos visto anteriormente, en el **Capítulo 4**. En este capítulo, estudiaremos otras aplicaciones más complejas, que permitirán darles una mayor riqueza a los programas que podamos implementar en nuestro trabajo.

Tanto este capítulo como el anterior no pretenden ser un listado exhaustivo de todas las estructuras y aplicaciones que proporciona Asterisk, sino solo mostrar la lógica que hay detrás de ellas y la manera de emplearlas. Debemos aclarar que es indispensable contar con ciertos conocimientos generales sobre programación para poder desarrollar con éxito un Dialplan funcional.

La documentación de Asterisk, proporcionada en el sitio web oficial, es el lugar adecuado para consultar acerca de todas las aplicaciones y las estructuras disponibles en la aplicación. Dado que la información se actualiza con frecuencia, sugerimos, decididamente, su revisión frecuente.

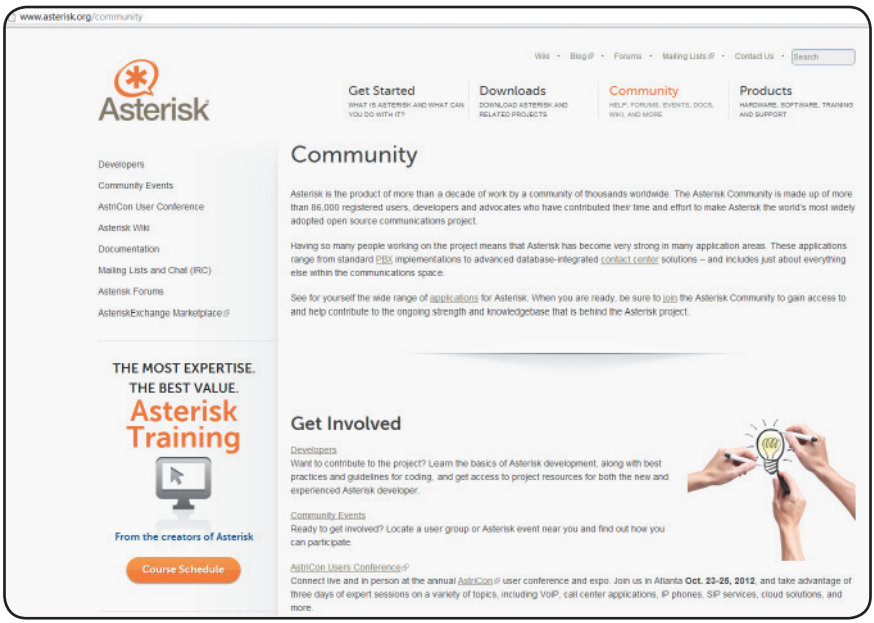


Figura 1. En www.asterisk.org **encontraremos información actualizada sobre las aplicaciones.**

Plantillas

Cuando trabajamos con los archivos de configuración y tenemos que repetir parámetros, podemos utilizar plantillas para simplificar nuestra tarea. Esta herramienta permite declarar una sección como una plantilla y, luego, utilizarla a modo de definición para incorporar los parámetros configurados en ella.

A continuación, presentamos un ejemplo clásico de plantilla, mencionado en muchos otros libros, sobre la configuración de los parámetros de terminales telefónicos, en este caso, teléfonos SIP:

```
...

[terminales](!) ; Este signo (!) transforma la sección en una plantilla

type=friend

context=internos

host=dynamic

disallow=all

allow=ulaw

dtmfmode=rfc2833

secret=unaclave

...
```

Agregando un signo de exclamación (**!**) luego del nombre de la sección, declaramos la sección como una plantilla. Ahora veremos cómo se utiliza en la siguiente porción del Dialplan:

```
...

[3030](terminales)

mailbox=3030

[2020](terminales)

mailbox=2020

[1010](terminales)

mailbox=1010

...
```

BASE DE DATOS ASTERISK (ASTDB)

Asterisk proporciona un mecanismo para almacenar valores que, luego, pueden ser empleados en el Dialplan. Se trata de una base de datos denominada `AstDB`, cuyo formato es `Berkeley DB1`.

Al emplear el nombre de la plantilla como atributo del nombre del contexto **[3030]**, incorporamos los parámetros definidos en aquella dentro de este. Ahora, si tenemos que hacer un cambio que afecte a todos los terminales, solo modificamos la plantilla, recargamos el Dialplan, y el cambio se habrá aplicado a todos los contextos donde se usó la plantilla en la declaración.

Directiva include

En ocasiones, resulta útil incorporar secciones o contextos definidos dentro de otros contextos, a modo de la expansión de librerías que permiten algunos lenguajes de programación, como C. Para hacerlo, Asterisk proporciona la directiva **include**.

Cuando dentro de un contexto invocamos esta directiva, seguida del nombre de otro contexto, podemos incluir las extensiones de este último en el primero. De esta manera, habilitamos sus extensiones para ser llamadas desde el primer contexto. La sintaxis para hacer esto es la siguiente:

```
include=> ventas
```

Donde **ventas** es el identificador del contexto que se desea incluir.

Asterisk evalúa, en primer lugar, las extensiones propias del contexto; si no tiene éxito, lo hace con las de los contextos incluidos, en el orden en que estos aparecen en la configuración.

Aplicación GoSub()

En general, cuando programamos, tenemos que realizar algunas tareas repetitivas en muchas partes del programa, y para esto empleamos las subrutinas. Asterisk proporciona la aplicación **GoSub()**, que implementa esta funcionalidad permitiendo el procesamiento de

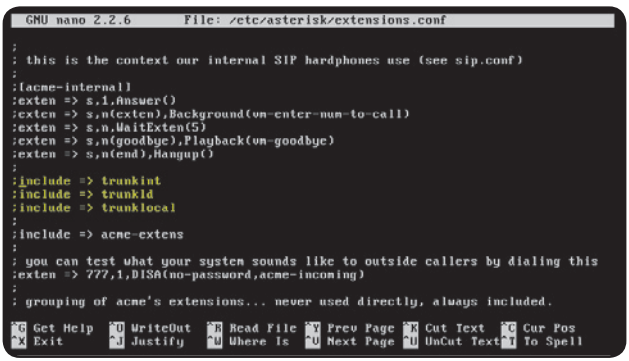


Figura 2. Esta directiva es muy potente y permite incluir código disponible en otras secciones.

una cápsula de código que realiza una determinada tarea, y devuelve un resultado o estado que luego es utilizado en el curso principal del programa.

Una subrutina se escribe como cualquier porción de código del Dialplan. Sin embargo, su código suele definirse en un contexto aparte, que se nombra de alguna manera particular para facilitar su reconocimiento. La porción de Dialplan que sigue nos muestra una subrutina y la llamada a ella desde el curso principal del Diaplan:

...

```
[subAviso] ; definimos un contexto aparte para la subrutina

exten => inicio,1,Dial(${Fede},10)

same => n,GotoIf($["${DIALSTATUS}" = "BUSY"]?ocupado:nodisp)

same => n(nodisp),Playback(no-disponible.wav)

same => n,Hangup()

same => n(ocupado),Palyback(ocupado.wav)

same => n,Hangup()

...

exten => 701,1,GoSub(subAviso,inicio,1())

...
```

Cuando se llama a la extensión **701**, se realiza una llamada a la subrutina **subAviso**, ingresando en ella en la primera secuencia de la extensión **inicio**. Dentro de los paréntesis se colocan los argumentos, que son los valores de las variables que la subrutina espera, si es que los hay. En este caso, la subrutina no espera ningún valor, por eso dejamos los paréntesis vacíos o, directamente, los omitimos.

En este ejemplo, la subrutina llama al canal definido en la variable **Fede**, alerta durante **10** segundos y, si no logra una respuesta, ejecuta el salto condicional sobre el valor de la variable **DIALSTATUS**. Según sea el valor de esta variable, **BUSY** u otro (no disponible), se reproducirá un mensaje y se desconectará la llamada.

VARIABLES DE CANAL

Una variable de gran utilidad es la denominada EXTEN. En ella se almacenan los dígitos que se marcan al llamar a una extensión. Esto tiene la ventaja de que permite emplear el número para un posterior procesamiento.

Si tenemos alguna experiencia con subrutinas, notaremos que falta un elemento importante. Cuando se llama a una subrutina, es para realizar una tarea, en general, repetitiva, y se espera que el control del flujo del programa sea devuelto a la instrucción siguiente a la sentencia en que se ha llamado a dicha subrutina. Entonces, necesitamos una forma de indicar que ha terminado la tarea, y que es necesario retornar y devolver o no un valor como respuesta.

Para ver este concepto de manera práctica y conocer el procedimiento que permite pasar argumentos a una subrutina, estudiaremos el siguiente código:

...

```
[subDial]

exten => inicio,1,Dial(${ARG1},${ARG2})

same => n,Return()

[subAviso]

exten => inicio,1,Playback(${ARG1})

same => n,Hangup()

...

exten => 701,1,GoSub(subDial,inicio,1
(${Fede},30))

same => n,GoSub(subAviso,inicio,1(no-
disponible.wav))
```

...

En este caso, llamar a la extensión **701** hace que se llame a la subrutina **subDial**, que ejecuta la aplicación **DIAL()**. En la llamada **GoSub()**, pasamos dos argumentos: el canal (**${Fede}**) y el tiempo de alerta al terminal remoto (**30** segundos). Transcurrido este tiempo sin obtener una respuesta, se procesa la aplicación **Return()**, que permite volver a

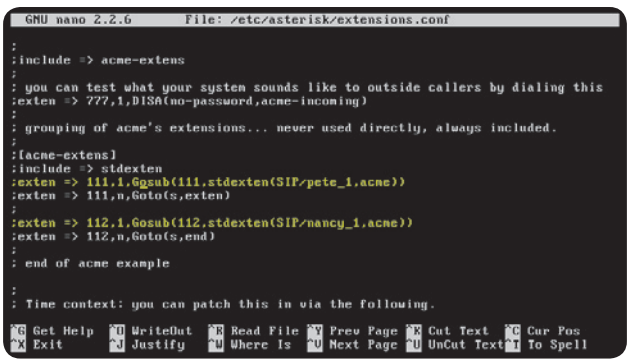


Figura 3. Se muestra la llamada con sus argumentos.

la secuencia que sigue a la llamada. Es decir, se ejecuta otra llamada, pero ahora a la subrutina **subAviso**, pasándole como único argumento el mensaje por reproducir. Finalmente, luego de reproducir este mensaje, la llamada es desconectada.

Notemos que los identificadores de variable **ARG1**, **ARG2**, .., **ARGn** están reservados para alojar los parámetros que se pasen a la subrutina llamada. Estos contenedores se llenarán con el valor pasado según su posición; es decir, el primer parámetro irá a **ARG1**, el segundo a **ARG2**, y así sucesivamente.

Para discriminar si el terminal remoto está ocupado o no disponible, empleamos la variable **DIALSTATUS**, que será posicionada en un valor cuando se ejecute la aplicación **Dial()** y no se obtenga respuesta desde el remoto. Así estará disponible cuando se ejecute la aplicación **Return()**, y su valor reflejará el estado en que se encuentra el terminal remoto.

Para utilizar esta información en nuestro código y realizar un salto condicional sobre él, podemos pasar este valor como respuesta de la aplicación **Return()**.

A continuación, veremos cómo quedará el código de ejemplo al incorporar este nuevo concepto:

```
...

[subDial]

exten => inicio,1,Dial(${ARG1},${ARG2})

same => n,Return(${DIALSTATUS})

[subAviso]

exten => inicio,1,Playback(${ARG1})

same => n,Hangup()

...

exten => 701,1,GoSub(subDial,inicio,1
(${Fede},20))

same => n,Set(MSG=${IF($[${GOSUB_
RETVAL}=BUSY]?ocupado:nodisp)})

same => n,GoSub(subAviso,inicio,1(${
MSG}))
```

MANIPULACIÓN DE VARIABLES

Al recuperar el valor de una variable, podemos manipularlo. Por ejemplo, si discamos **01115435207**00, Asterisk lo almacenará en la variable `EXTEN`. Luego, si en una sentencia se invoca el valor `${EXTEN:5:8}`, recuperamos el número **43520700** de la cadena original.

En la aplicación **Return()**, agregamos el contenido de la variable **DIALSTATUS** como valor de retorno; esto se hace poniendo este valor entre los paréntesis de la llamada. Este valor se almacenará en una variable de canal especial, definida en Asterisk para este fin: **GOSUB_RETVAL**. Cuando se llama a la extensión **701**, se invoca la subrutina **subDial**, y si no se obtiene respuesta del remoto, esta regresa y devuelve el estado del terminal remoto en la variable **GOSUB_RETVAL**.

La lógica es similar al ejemplo anterior, pero en este caso agregamos una prueba del valor de la variable **GOSUB_RETVAL** con la aplicación **IF()**. Si el valor de la variable es **BUSY**, entonces la variable **MSG** tomará los valores **ocupado** o **nodisp**. Estos son los nombres de los archivos de sonido que serán reproducidos en cada caso, lo cual se observa en la última línea de la extensión **701**. Allí se realiza la llamada a la subrutina **subAviso**, pasándole como argumento el nombre del archivo por reproducir en **ARG1**, que corresponderá al valor de la variable **MSG**, posicionada en la segunda línea del código principal del Dialplan de ejemplo.

Es importante destacar que este es un ejemplo clásico empleado para mostrar el potencial que posee esta estructura, y al igual que en otros lenguajes, facilita la construcción de programas, la lectura, el mantenimiento y la reutilización de código, y hace más compacto al código resultante.

Operadores y expresiones

Como muchos otros lenguajes de programación, Asterisk incluye **operadores**, que son un conjunto de símbolos conectores especiales para manipular variables. Los tipos principales de operadores son:

-Booleanos: evalúan si la expresión es verdadera o falsa. Estos son:

- **|**: se emplea para la operación **OR** (**o**)
- **&**: se utiliza para la operación **AND** (**y**)
- **>**: mayor que
- **<**: menor que
- **=**: igual a
- **>=**: mayor o igual a
- **<=**: menor o igual a
- **!=** distinto de

CANALES LOCALES

Al discar de manera simultánea varios canales con distintos parámetros, una forma de incorporarlos desde una única llamada a la aplicación `DIAL()` es a través del tipo de canal `Local`.

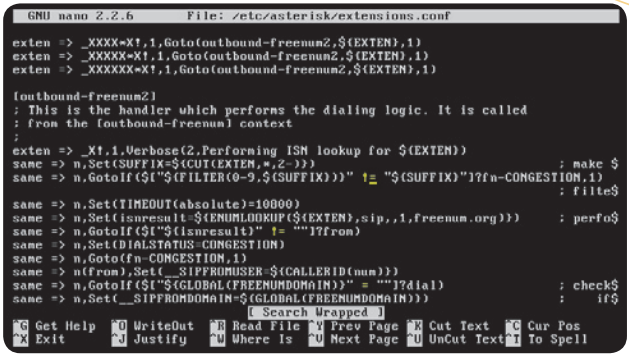


Figura 4.
Algunos de los operadores empleados en Asterisk.

-**Matemáticos**: realizan las operaciones de suma (**+**), resta (**-**), multiplicación (*****), división (**/**) y porcentaje (**%**).

Las **expresiones** son conjuntos de variables, operadores y valores que, evaluados en conjunto, devuelven un valor o resultado de la expresión. En el Dialplan estas expresiones tienen la siguiente sintaxis:

```
$[expresión]
```

Cuando el parser del Dialplan encuentra una expresión, la evalúa y reemplaza el símbolo por el valor. Para la evaluación de la expresión, primero sustituye las variables por su contenido y, después, realiza las operaciones indicadas por los operadores involucrados. Por ejemplo, consideremos el siguiente caso:

```
$[${cLlamadas} / ${nUsuarios}>=50]
```

Asterisk reemplazará las variables **cLlamadas** y **nUsuarios** por sus contenidos; luego, realizará la operación de división y, por último, verificará si es mayor o igual que **50**. Si lo es, la evaluación de la expresión dará como resultado **True**; en caso contrario, dará **False**.

Funciones

Asterisk contiene una estructura especial llamada **función**, que permite realizar cálculos con las expresiones y valores. La sintaxis de las funciones es:

```
NOMBRE_de_FUNCION(argumento)
```

Desde el punto de vista de la referencia, se la invoca como a una variable, y de la misma manera, a su valor:

```
${NOMBRE_de_FUNCION(argumento)}
```

Por ejemplo, la función **LEN()** permite calcular la longitud de la cadena de caracteres del argumento. Para conocer la lista de funciones

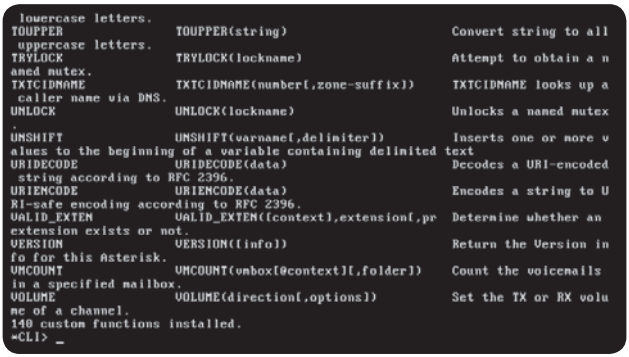


Figura 5. La última parte del despliegue de las funciones disponibles en Asterisk.

disponibles, podemos ingresar, en el prompt de Asterisk, el comando:

```
CLI> core show functions
```

Saltos condicionales

Así como en otros lenguajes contamos con algunas instrucciones para realizar un redireccionamiento del flujo del programa de acuerdo con el valor que arroje la evaluación de alguna condición, en esta aplicación disponemos de saltos condicionales, que explicaremos a continuación.

APLICACIÓN GOTOIF()

Asterisk dispone de la aplicación **GotoIf()**, que realiza un salto a un lugar indicado del Dialplan siempre que la condición evaluada sea verdadera (**True**). Veamos, en principio, la sintaxis que emplea:

```
GotoIf(expresión-a-evaluar?destino1:destino2)
```

Si la evaluación de la expresión da un valor verdadero (cualquier valor distinto de 0 o una cadena vacía d), el flujo del programa es dirigido hacia el punto identificado como **destino1**; en caso contrario, será redirigido al punto identificado como **destino2**.

Los identificadores de destino se componen de hasta tres elementos, y pueden ser:

- Una etiqueta de prioridad dentro de la misma extensión.
- Una extensión más una etiqueta de prioridad dentro del mismo contexto.
- Un contexto más una extensión más una etiqueta de prioridad dentro del Dialplan.

```
...

exten => 555,GotoIf($[${nCalls} - 20]?limite:credito)
```

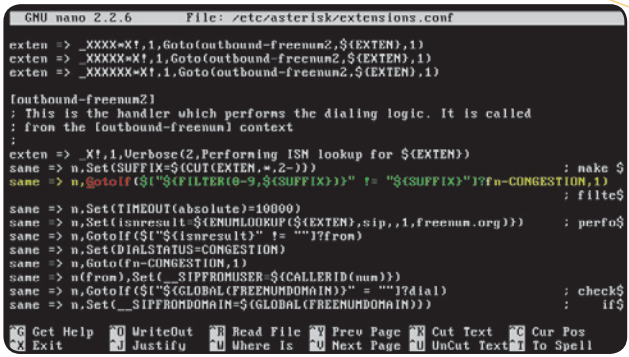


Figura 6. Esta aplicación es muy poderosa y puede ser muy compleja. Notemos las diferentes partes de su sintaxis.

```
same => n(limite),Playback(se-acabo-el-credito.wav)

same => n,Hangup()

same => n(credito),Playback(tiene-credito.wav)

same => n,Hangup()

...
```

En este ejemplo tenemos dos ramas definidas y terminadas por la aplicación **Hangup()**, luego de reproducir un mensaje acerca del crédito para hacer llamadas.

Cuando se llama a la extensión **555**, se evalúa el valor de la variable **nCalls**, y si este ha alcanzado el número **20**, se saltará a la etiqueta de prioridad/secuencia identificada como **limite**; en caso contrario, lo hará a la etiqueta de prioridad **credito**.

Si omitimos ingresar el destino etiquetado como **limite**, el programa funcionará igual, porque la próxima secuencia es, justamente, la que está etiquetada como **limite**.

IMPLEMENTACIÓN DE LAZOS

Veremos un ejemplo donde combinamos las aplicaciones **GotoIf()** y **Goto()** para construir lazos de ejecución:

SALTOS DE EJECUCIÓN

En programación es necesario contar con instrucciones de control del flujograma. Dos de las más empleadas son los saltos condicionales, que deciden sobre la evaluación de una condición, y los saltos incondicionales, que no requieren de este paso previo.

...

```
exten => 555,1,Set(nCalls=20)

same => n(entrada),GotoIf($[${nCalls} != 0]?:SinCredito)

same => n,SayNumber(${nCalls})

same => n,Set(COUNT=$[${nCalls} - 1])

same => n,Goto(entrada)

same => n(SinCredito),Hangup()
```

...

Cuando se llama a la extensión **555**, primero se establece el valor **20** para la variable **nCalls**. En la siguiente secuencia, se evalúa si el valor de esta variable es distinto de **0**; en caso afirmativo, se continúa con la siguiente secuencia, donde empleamos la aplicación **SayNumber()** para reproducir el valor de la variable en el terminal telefónico que realiza la consulta. A continuación, se decrementa el valor de la variable **nCalls** y se realiza un salto incondicional a la secuencia identificada con la etiqueta de prioridad **entrada**.

Cuando el valor de la variable **nCalls** llegue a **0**, la evaluación de la condición será **False** y el salto se realizará al punto identificado con la etiqueta de prioridad **SinCredito**, luego de lo cual la llamada será desconectada.

Podemos colocar en una misma secuencia/prioridad dos aplicaciones, que se ejecutarán según el orden de izquierda a derecha.

Debemos destacar que, cuando ponemos solo un argumento como destino en un **GotoIf()**, este se refiere a una etiqueta de prioridad, el segundo será una extensión, y el tercero le sumará un contexto.

APLICACIÓN GOTOIFTIME()

Por tratarse de una aplicación que trabaja con datos cuya naturaleza involucra al tiempo como una variable muy comprometida, Asterisk brinda una aplicación de salto condicional, que evalúa la fecha o, más precisamente, el System Time (hora del sistema), para decidir qué camino tomar. La sintaxis de esta aplicación es la siguiente:

ASTDB

En la DB de Asterisk, los datos se almacenan agrupándolos en lo que se conoce como familias, con valores identificados por claves. Una clave dada solo puede usarse una vez dentro de cada familia de datos.

```
GotoIfTime(hora,dia_de_la_
semana,día_del_mes,mes?etiqueta)
```

Esta aplicación encamina el flujo del programa hacia el punto identificado como **etiqueta**, si los argumentos suministrados coinciden con la fecha y la hora actuales, obtenidas desde el reloj del sistema. Veamos un ejemplo de su uso para comprender mejor la forma que tomarán los argumentos:

...

```
exten => entrada,1,GotoIfTime(*,*,1,
jan?cerrado,inicio,1)

same => n,GotoIfTime(09:00-17:59,mon-
fri,*,*?abierto,inicio,1)

same => n,GotoIfTime(09:00-
11:59,sat,*,*?abierto,inicio,1)

same => n,Goto(cerrado,inicio,1)
```

...

La primera secuencia de la extensión **entrada** contiene una llamada **GotoIfTime()**, donde el arreglo de argumentos se lee de la siguiente manera: "cualquier hora de cualquier día de la semana, el primer día del mes (los días se numeran del 1 al 31, y se puede usar el operador **&** para días no contiguos), en el mes de enero".

Si esto es cierto, se produce un salto a la primera secuencia de la extensión **inicio** dentro del contexto **[cerrado]**. Si la evaluación resulta falsa, entonces se continúa con la siguiente secuencia, donde se evalúa si estamos entre las **09:00** y las **18:00** horas (el día inicia a las 0:00 y finaliza a las 23:59) de cualquier día hábil (lunes a viernes) de la semana (si queremos poner días no contiguos, utilizamos el operador **&** entre ellos), en cualquier día del mes y cualquier mes del año. Si esto es cierto, entonces se saltará a la primera secuencia de la extensión **inicio**, pero del contexto **[abierto]**. Si la evaluación resulta falsa, se procede a comprobar si estamos en un día sábado, entre las **09:00** y las **12:00** horas. Si es cierto, se saltará al mismo punto del contexto **[abierto]**; en caso contrario, se hará un salto incondicional (**Goto()**) a la primera secuencia de la extensión **inicio** del contexto **[cerrado]**.

WIKI

Se queremos profundizar la información acerca del uso y las opciones del tipo de canal `Local`, podemos acceder a la siguiente dirección: https://wiki.asterisk.org/wiki/display/AST/Local+Channel.

```
structure based on the evaluation of the given time specification. After this
application completes, the pbx engine will continue dialplan execution at
the specified location in the dialplan. If the current time is within the
given time specification, the channel will continue at <labeliftrue>. Otherwise
the channel will continue at <labeliffalse>. If the label chosen by the
condition is omitted, no jump is performed, and execution passes to the next
instruction. If the target jump location is bogus, the same actions would
be taken as for 'Goto'. Further information on the time specification can
be found in examples illustrating how to do time-based context includes in
the dialplan.

[Syntax]
GotoIfTime(times,weekdays,mdays,months[,timezone]?[labeliftrue][:labeliffalse])

[Arguments]
labeliftrue
    Continue at <labeliftrue> if the condition is true. Takes the form
    similar to Goto() of [[context,]extension,]priority.
labeliffalse
    Continue at <labeliffalse> if the condition is false. Takes the form
    similar to Goto() of [[context,]extension,]priority.

[See Also]
GotoIf(), Goto(), IFTIME, TESTTIME
*CLI> _
```

Figura 7. Para acceder a la documentación en línea de una aplicación, empleamos el comando CLI `core show application <nombre-apl>`.

Con esta sencilla aplicación, podemos administrar el flujo de atención de las llamadas que se hacen en horario laboral o fuera de este.

Las etiquetas de identificación siguen las mismas reglas que en el caso de la aplicación **`GotoIf()`**, que analizamos en el apartado anterior.

RESUMEN

En este capítulo hemos explorado las aplicaciones que implementan los saltos condicionales, el procedimiento para manejar el código repetitivo, y algunas otras herramientas y aplicaciones fundamentales que nos ayudarán a implementar un Dialplan.

Capítulo 6

Aplicaciones con Asterisk

Veremos la implementación de algunas aplicaciones muy utilizadas en sistemas de atención telefónica.

RedUSERS.com

Introducción

En todos los ámbitos que involucran la atención de clientes o usuarios, la función de recepción de llamadas y toma de mensajes es clave para mantenernos comunicados. Históricamente, estas funciones son realizadas por una o más personas que trabajan en el área de recepción y que tienen responsabilidad sobre las agendas de una persona o grupo, en diferentes departamentos de una empresa o institución.

En este capítulo, veremos algunas aplicaciones que facilitan estas funciones y ayudan a mejorar la productividad de quienes se desempeñan en estas áreas. Incluso, en algunas organizaciones de pocos empleados, donde estos desarrollan múltiples tareas, estas aplicaciones permiten una atención profesional, sin la carga económica para la empresa y de trabajo para los empleados que significa dedicar personal y horas de labor a ellas.

Correo de voz

En primer lugar, veremos una de las aplicaciones más utilizadas por todos los sistemas de telefonía: el **correo de voz** o **casilla de mensajes**. Por lo general, dotar a los sistemas comerciales de esta herramienta requiere de un presupuesto elevado. Asterisk incorpora una aplicación que permite implementar esta funcionalidad en el Dialplan sin costo adicional (y como es de código abierto, esto significa con costo cero). Se denomina **`VoiceMail()`** y presenta las siguientes características generales:

- Saludos preparados por defecto o configurables.
- Notificación de que un mensaje de voz ha llegado a través de un correo. Además, presenta la opción de adjuntar al correo el mensaje como un archivo de sonido. Esta integración es una característica avanzada.
- Un terminal puede recibir mensajes de más de una casilla de mensajes.
- Una casilla de mensajes puede recibir mensajes de más de un terminal telefónico.
- No hay límites de licencias para la cantidad de casillas que se pueden habilitar.
- Cada casilla está implementada por una simple carpeta o directorio para organizar los correos.
- Saludos diferentes para los estados no-disponible y ocupado.
- Capacidad de reenvío de mensajes y difusión.
- Indicación de mensaje en espera para diferentes tipos de teléfonos.
- Capacidad de armar un directorio de empleados de la compañía para mejorar la atención de las llamadas entrantes.

SECCIONES Y CONFIGURACIÓN

Al igual que muchas otras aplicaciones, esta funcionalidad se configura a través de los parámetros incluidos en un archivo llamado **`voicemail.conf`**, que se encuentra en el interior del directorio **`/etc/asterisk/`**. Con la instalación de Asterisk, se incluye una versión de este archivo preconfigurado, que requiere de pocos cambios y detalles para lograr la implementación de la herramienta en nuestros sistemas.

Este archivo, como otros que hemos visto, presenta algunas secciones identificadas por nombres entre corchetes, que analizaremos a continuación.

Sección [general]

En la sección **[general]**, donde se definen las configuraciones globales de la aplicación, podemos indicar el formato de los archivos de audio que se van a utilizar. Los valores más empleados para este parámetro son **WAV** y **WAV49**.

También es posible habilitar, en la integración con un sistema de correo, el adjuntado de los mensajes de voz como archivos de sonido, codificados según lo que definamos en esta misma sección.

Otros parámetros que podemos configurar son los que explicamos a continuación:

- La duración mínima de un mensaje, en segundos, para que sea grabado.
- La longitud, en segundos, de los saltos de búsqueda de mensajes. Cuando se recuperan los mensajes, se puede avanzar o retroceder presionando las teclas asterisco (*) y numeral (#), respectivamente.
- La máxima cantidad de intentos de ingreso a la casilla.

Las opciones avanzadas de esta sección, además de definirse en forma global, pueden configurarse para cada casilla de mensajes; de esta manera, tendrán prioridad sobre la definición global.

Aconsejamos revisar estas opciones para ver si su valor por defecto entra en conflicto con los de la implementación. En general, los valores por defecto del archivo **voicemail.conf** son adecuados para la mayoría de las instalaciones simples, y solo algunos deberán modificarse en instalaciones complejas, en especial, aquellos relacionados con la seguridad o con un impacto sobre ella. Por ejemplo, el parámetro **dialout <context>**, que permite a los usuarios llamar desde sus casillas de mensajes.

Sección [zonemessages]

La sección **[zonemessages]** permite establecer el manejo de los mensajes en relación con

SEGURIDAD I

Asterisk incluye un script para mejorar la seguridad del sistema de mensajería de voz, que se encuentra en `/contrib/scripts/voicemailpwcheck.py`. **Para emplearlo, hay que moverlo a otro directorio, por ejemplo,** `/usr/local/bin` **y habilitar la opción** `externpasscheck=` **en el archivo** `voicemail.conf`.

su **time-zone** específico. Por ejemplo, es posible definir etiquetas para identificar las zonas de tiempo.

En el archivo de muestra, que se instaló junto con Asterisk, **/usr/src/sistema-asterisk/asterisk/1.8/configs/voicemail.conf.sample**, pueden verse la sintaxis y los detalles para usar esta sección.

Sección [default]

El contenido principal del archivo de configuración está ocupado por la sección de los contextos de las casillas de mensajes. Esta sección, denominada **[default]**, agrupa las definiciones de las casillas de mensajes creadas para el sistema de mensajería de voz.

Las aplicaciones que están relacionadas con **VoiceMail()** buscarán este contexto, a menos que se definan otros explícitamente. La mayoría de los usos que se hagan de la mensajería de voz no requieren segmentar las casillas de los usuarios, así que la opción por defecto será la seleccionada.

No necesitaremos hacer modificaciones y podremos crear las casillas de mensajes para nuestros usuarios dentro de este contexto. La sintaxis para hacerlo es la siguiente:

```
mailbox => password [,FirstName LastName [,email addr [,pager addr [,options[|options] ] ] ] ]
```

Veamos brevemente el significado de cada elemento del comando:

- **mailbox**: es el número de la casilla, que, en general, se corresponde con el número de extensión del terminal asociado.
- **password**: es la contraseña numérica que debe ingresar al usuario para poder acceder a la casilla.
- **FirstName LastName**: es el nombre del propietario de la casilla de mensajes.
- **email addr**: es la dirección de e-mail a la cual Asterisk envía las notificaciones.
- **pager addr**: es la dirección de correo del pager o del móvil del propietario de la casilla, donde Asterisk puede enviar una notificación.
- **options**: son las opciones por casilla, que tienen prioridad sobre las globales. Algunas opciones válidas son: **attach, servere-mail, tz, saycid, review, operator, callback, dialout** y **exitcontext**.

MULTICONFERENCIAS

En los sistemas PBX tradicionales, la capacidad de realizar multiconferencias entre extensiones es muy valorada. En estos casos, habilitar esta característica tiene costos muy elevados; en Asterisk, por supuesto, es gratuito.

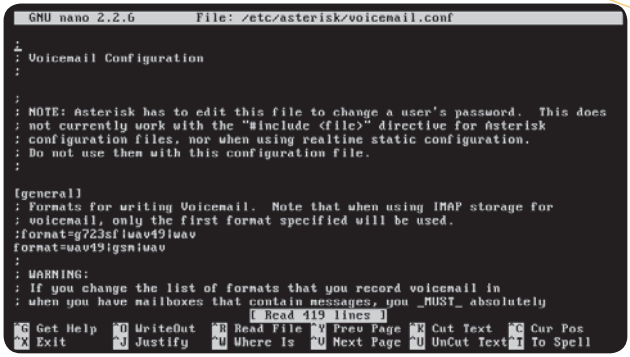


Figura 1. Mediante la configuración de `/etc/asterisk/voicemail.conf`, armamos el sistema de mensajería de voz.

Como ejemplo, en el listado siguiente, vemos una porción de un contexto del archivo de configuración:

```
...

[default]

701 => 25419,Pepe Totio,p.totio@lapropia.org

702 => 21678,Esteban Quito,e.quito@lapropia.org

[otroContexto]

800 => 6734,Tio Sam,t.sam@otroContexto.com

801 => 1987,JoseIngenieros,j.ingenieros@otroContexto.com,,attach=yes|maxmsg=200|tz=ARG

...
```

La integración entre el servicio de mensajes de voz y el sistema de e-mail requiere establecer el correo electrónico en la plataforma Linux sobre la que se ejecuta Asterisk.

A continuación, veremos en profundidad las aplicaciones que brinda Asterisk, para

SEGURIDAD II

Asterisk, por defecto, no realiza la validación de contraseñas en la aplicación de `VoiceMail` (casilla de mensajes). Si lo deseamos, podemos configurar esta aplicación para que le pase el requerimiento a un sistema externo.

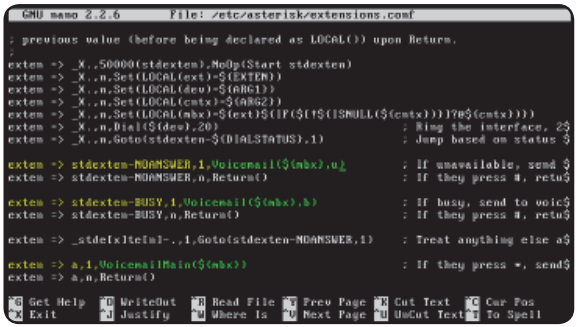


Figura 2.
El Dialplan preconfigurado muestra ejemplos de llamadas a esta aplicación.

implementar el servicio de mensajería de voz en el Dialplan.

APLICACIÓN VOICEMAIL()

La sintaxis de esta aplicación, tal como la utilizaremos en el Dialplan, es la siguiente:

```
VoiceMail(mailbox[@context][&mail-
box[@context][&...]][,options])
```

A la llamada debemos pasarle dos argumentos: la dirección de la casilla para almacenar el mensaje y un lista de opciones que detallamos a continuación:

- **b**: se utiliza para indicar el mensaje de terminal ocupado.
- **d([C])**: acepta dígitos para ser procesados en el contexto **[C]**. Si no se especifica el contexto, utiliza **[default]**.
- **g(#)**: solo para canales DAHDI, indica la ganancia en decibeles que se aplicará al audio durante la grabación del mensaje.
- **s**: para suprimir las instrucciones que se dan por defecto, después de reproducir el saludo.
- **u**: se utiliza para indicar el mensaje de terminal no disponible. Este es el comportamiento por defecto.
- **U**: indica que el mensaje debe ser marcado como urgente cuando se pasa una notificación por correo o para informar este estado cuando el usuario llama para recuperar sus mensajes.
- **p**: establece que el mensaje debe ser marcado como prioritario.

Como ejemplo, la siguiente porción de Dialplan muestra el uso de esta aplicación:

```
...

exten => 903,1,Dial(${Pepe},20)

same => n,GotoIf($["${DIALSTATUS}" =
"BUSY"]?ocupado:nodisp)

same => n(nodisp),VoiceMail(903@
default,u)

same => n,Hangup()
```

```
same => n(ocupado),VoiceMail(903@
default,b)

same => n,Hangup()

...
```

Cuando se llama a la extensión **903**, el Dialplan cursa una llamada al canal identificado con el contenido de la variable **Pepe**. Alerta durante **20** segundos y, si no obtiene respuesta, realiza una evaluación del valor de la variable **DIALSTATUS**. Si este es **BUSY**, salta a la línea identificada con la etiqueta **ocupado**, redirige la llamada a la casilla de mensajes correspondiente al usuario llamado y reproduce un mensaje de ocupado. En caso contrario, continúa con la siguiente secuencia identificada como **nodisp**, redirige la llamada a la casilla de mensajes correspondiente al usuario llamado y reproduce un mensaje de no disponible.

Los mensajes, por defecto, son almacenados en el sistema de archivos de Linux, en el directorio: `/var/spool/asterisk/voicemail/<contexto>/<casilla_mensajes>`.

Estructura del sistema de mensajes de voz:

```
./INBOX

./INBOX/msg0001.WAV

./INBOX/msg0001.txt

./INBOX/msg0002.WAV

./INBOX/msg0002.txt

./Old

./Old/msg0000.WAV

./Old/msg0000.txt

./ocupado.WAV

./nodisp.WAV
```

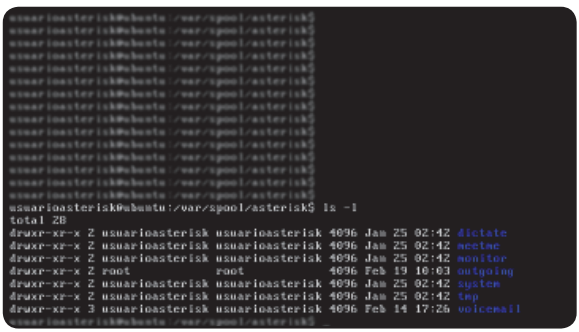


Figura 3. La instalación de Asterisk arma esta estructura para almacenar los mensajes de voz.

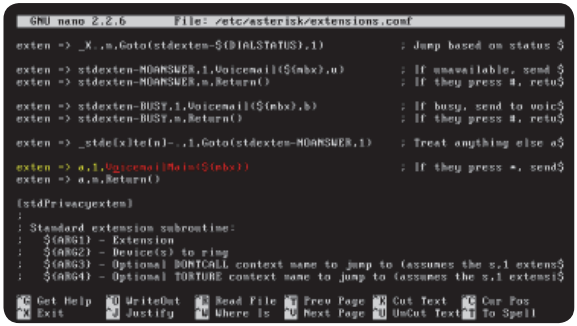


Figura 4. Otro ejemplo del archivo preconfigurado.

Vemos dos mensajes nuevos en la carpeta **INBOX**, un mensaje guardado en la carpeta **Old**, y los archivos de sonido con los avisos de ocupado y no disponible en formato **WAV**.

APLICACIÓN VOICEMAILMAIN()

Como sabemos, **VoiceMail()** permite dejar mensajes para los usuarios que no pueden o no quieren tomar las llamadas en tiempo real. Para cuando estos usuarios deseen escuchar los mensajes que les han dejado en sus casillas, Asterisk dispone de la aplicación **VoiceMailMain()**. Esta permite recuperar los mensajes, grabar saludos y modificar las opciones del servicio para quien lo solicite. Su sintaxis es la siguiente:

```
VoiceMailMain([mailbox][@context]
[,options])
```

Los argumentos son opcionales, pero si no se indica una dirección de casilla, se desplegará una solicitud para ingresar su número. En la siguiente lista mostramos los valores de las opciones.

- **g(#)**: aumenta la ganancia en decibeles para la reproducción de los mensajes.
- **p**: permite utilizar el parámetro **[mailbox]** como un prefijo del número de casilla.
- **s**: evita la confirmación de contraseña.
- **a(carpeta)**: inicia la sesión en una de las siguientes carpetas del sistema de mensajes de voz: **INBOX**, **Old**, **Work**, **Family**, **Friends**, **Cust1**, **Cust2**, **Cust3**, **Cust4**, **Cust5**.

UNIFIED MESSAGING

Cuando VoiceMail está integrado al e-mail, el conjunto se denomina mensajería unificada. En sistemas tradicionales, su implementación es costosa. Asterisk lo resuelve con un gateway entre el VoiceMail y el e-mail a través de un servidor IMAP.

APLICACIÓN DIRECTORY()

La aplicación **Directory()** emplea los nombres de las casillas de mensajes definidas en el archivo **voicemail.conf** para armar un directorio que les permitirá a los usuarios realizar un discado por nombre.

Su sintaxis es la siguiente:

```
Directory(contexto_
nombres,contexto_dial,opciones)
```

El primer argumento indica el contexto del sistema de mensajes, y el segundo, el contexto del Dialplan al cual pasar la llamada.

En la porción de Dialplan siguiente, vemos un ejemplo de uso de esta aplicación:

```
...

exten => 0,1,Directory(default,inte
rnos,f)

exten                              =>
5,1,Directory(default,internos)

...
```

Si los usuarios digitan el **0**, se les presentará un directorio de apellidos; y si digitan el **5**, uno con los nombres. Esta diferencia se indica con la opción **f** como tercer argumento. El nombre y el apellido están definidos en el archivo **voicemail.conf**, en la sección **[default]**. Una vez realizada la selección, la llamada se enviará al contexto **[internos]** del Dialplan.

CALIDAD DEL SONIDO

El sonido digitalizado se envía por las redes, junto con todos los otros tipos de datos, para alcanzar el servidor de mensajes de voz. Los datos de sonido tienen una naturaleza particular porque es información en tiempo real. Esto significa que perder un paquete con esta información tiene un efecto negativo sobre la calidad de la comunicación que se pretende establecer. En el caso del servicio de mensajes de voz, los mensajes quedarían entrecortados, ininteligibles o, directamente, perdidos. El factor determinante para que esto suceda es la variación en la latencia, más conocida como **jitter** de las redes de datos. Para contrarrestar este comportamiento natural de las redes de datos, Asterisk posee una opción que entra en funcionamiento al utilizar la aplicación. En la siguiente porción de código mostramos su uso:

ÓRGANOS DE CONTROL

En la Argentina, los órganos de gobierno de las comunicaciones son la Secretaría de Comunicaciones, autoridad de aplicación de las políticas de comunicaciones; y la Comisión Nacional de Comunicaciones, encargada del control de dichas políticas.

...

```
[Servicios]

exten => #40,1,Dial(Local/mensavoz@
Servicios/nj)

exten => mensavoz,1,VoiceMailMain()
```

...

La opción **j**, empleada en la llamada a un canal local, habilita un buffer de jitter entre el llamante y el servicio de mensajes de voz. Este buffer puede solucionar el problema que se presenta cuando uno o más paquetes con información de sonido llegan fuera de secuencia, se pierden o no llegan a tiempo, rellenando los espacios vacíos y acomodando el stream de sonido para que el almacenamiento o la reproducción tengan la calidad adecuada. La opción **n** permite indicar que la optimización no se realice fuera del camino de esta llamada.

Esta implementación es necesaria porque esta característica (conocida en inglés como **Jitterbuffer**) solo es soportada en el bridge entre dos canales. Como en el sistema de mensajes de voz solo tenemos un canal conectado a las aplicaciones, debemos pasar por un canal local para cumplir con este requerimiento.

Menús con Asterisk

Combinando otras aplicaciones que proporciona Asterisk, podemos construir sencillos menús que orientarán a la persona que llama para que encuentre lo que busca, y a nosotros nos permitirá mejorar el servicio.

En el ejemplo que presentamos a continuación, vemos una simple implementación de un menú con tres posibilidades:

...

```
[Menu-1]

exten => 89,1,Answer()

same => n,Background(principal)
```

SERVIDOR DE VOICEMAIL

En una implementación tradicional, el sistema de VoiceMail suele atenderse con un servidor dedicado a este servicio. Asterisk es capaz de funcionar como un sistema dedicado a ser servidor de VoiceMail para ser incorporado en las instalaciones de PBX propietarias.

```
same => n,WaitExten(10)

exten => 1,1,Playback(sonidos/1)

exten => 2,1,Playback(sonidos/2)

exten => 3,1,Hangout()

...
```

En el contexto **[Menu-1]** de esta porción del Dialplan, hemos utilizado dos nuevas aplicaciones: **Background()** y **WaitExten()**. La primera opera en forma similar a **Playback()**, empleada en otros ejemplos. La diferencia es que si la persona que llama introduce algún dígito mientras se está reproduciendo el mensaje, que le hemos pasado como argumento a la aplicación, esta detiene la reproducción y le pasa la llamada a la extensión indicada. En nuestro ejemplo, al llamar a la extensión **89**, se reproducirá el mensaje grabado en el archivo **principal**, donde se informarán las opciones posibles, mientras se esperan dígitos DTMF (son los sonidos que se escuchan cuando presionamos los números en un teclado telefónico) en **Background**. Si estos no llegan y deseamos darle tiempo al usuario para que ingrese su selección, una vez terminada la reproducción (puede suceder que el usuario necesite la última opción grabada en el aviso), podemos emplear la segunda aplicación mencionada, a la que le pasamos como argumento el tiempo en segundos que deseamos aguardar (en este caso, **10** segundos) antes de continuar con la siguiente secuencia.

En este menú de tres opciones, las dos primeras conducen a la reproducción de los números marcados (sus sonidos están grabados en archivos cuyo nombre es el número correspondiente en el directorio **/sonidos** dentro del directorio por defecto donde Asterisk guarda los sonidos) y la tercera, al corte de la comunicación. Si se ingresa cualquier otro número, el Dialplan no sabrá cómo proceder. Por eso, es importante planear las posibles ramificaciones antes de implementar esta característica.

Asterisk define algunas extensiones especiales para el manejo de situaciones excepcionales, como el tratamiento de información no válida cuando se espera por ella para realizar alguna acción en particular, como en nuestro ejemplo. Para estos casos, está definida la extensión identificada con la letra **i**, donde Asterisk, en

GRABACIÓN DE MENSAJES

La aplicación `Record()` toma como argumento el nombre de archivo con su extensión, porque permite grabar en diferentes formatos. La aplicación `Playback()` no requiere la extensión porque selecciona el mejor formato en términos de consumo de recursos de CPU.

vez de cancelarla, dirige la llamada a dicha extensión si los dígitos que se han ingresado no corresponden a ninguna extensión válida en el contexto considerado.

Otra extensión es la definida con la letra **t**, que se emplea cuando se ha superado el tiempo de espera del ingreso de dígitos por parte del usuario. El ejemplo siguiente muestra el uso de estas extensiones:

```
...

[Menu-1]

exten => 89,1,Answer()

same => n,Background(principal)

same => n,WaitExten(10)

exten => 1,1,Playback(sonidos/1)

exten => 2,1,Playback(sonidos/2)

exten => 3,1,Hangout()

exten=>i,1,Playback(entrada-invalida)

exten                                    =>
t,1,Playback(supero-el-tiempo)

same => n,Hangup()

...
```

Call parking

Esta funcionalidad se utiliza para poner en espera una llamada o para **estacionarla** de modo que pueda ser tomada desde otra extensión. Permite dotar al sistema de la capacidad de transferencia de llamadas entre internos. Su comportamiento se define en el archivo **features.conf**. Los parámetros más importantes, configurados en la sección **[general]** de este archivo, son los siguientes:

Figura 5. Los archivos `*.CONF` tienen información de configuración. Aquí vemos el de la funcionalidad `Call Parking`.

- **parktest**: es el nombre de la extensión en la que se estacionan las llamadas. Al transferir la llamada a esta extensión, el sistema le dirá en qué posición está. El valor por defecto es **700**.
- **parkpos**: define el número de ranuras de que dispone el sistema para alojar llamadas. Por ejemplo, para definir treinta posiciones, debemos configurar este parámetro en **701-730** .
- **context**: indica el nombre del contexto de estacionamiento de llamadas.
- **parkingtime**: indica el tiempo, en segundos, que la llamada puede estar estacionada. Si no se responde antes de este tiempo, el sistema devuelve la llamada a la extensión que la puso en ese estado.

Veamos un ejemplo de uso de esta característica:

```
...

[internos]

include => parkedcalls

exten => 404,1,Dial(SIP/Carlos,,tT)

exten => 405,1,Dial(SIP/Silvia,,tT)
```

Un usuario llama a la extensión **401** para hablar con **Carlos**, quien luego de un tiempo, necesita pasarle la llamada a **Silvia** para que complete la solución del problema que tiene el usuario. **Carlos** transfiere la llamada a la extensión **700** y recibe un mensaje que le dice que la llamada ha sido estacionada en la extensión **710**. **Carlos** llama ahora a **Silvia** a la extensión **405** y le avisa que tiene una llamada en espera del usuario en la extensión **710**; entonces **Silvia** llama a la extensión **710** y habla con el usuario. El usuario estuvo hablando con **Carlos**, quien le dijo que lo transferiría a una compañera que seguiría con su caso; luego, fue puesto en espera y, finalmente, fue atendido por **Silvia**, con quien continuó la conversación.

Atención **automatizada**

"Si quiere enviar un fax, presione uno, si quiere hablar con ventas presione dos, o espere y será atendido...". Una de las herramientas más comunes que poseen las implementaciones de sistemas de telefonía en general es la

MENSAJES EN VOICEMAIL

Asterisk crea un archivo por cada mensaje que se deja en el sistema de VoiceMail. El nombre del archivo es `MSGNNNN.TXT` (donde `NNNN` es un número de cuatro cifras); contiene información de contexto del mensaje y se utiliza para manejar la indicación de mensaje en espera.

denominada **autoatendedor**. Con esta funcionalidad nos encontramos al llamar a una empresa: un autómata que nos guía por numerosas opciones de un menú hasta encontrar lo que queremos.

Un sistema de atención automática presentará las siguientes características generales:

- Posibilidad de transferir a una extensión.
- Posibilidad de dejar un mensaje de voz.
- Posibilidad de transferir la llamada a una cola de espera.
- Reproducción de mensajes.
- Conexión con otros menús.
- Conexión con la recepción o desborde final.
- Repetición de las opciones.

Si bien estas características son algunas de las que ya hemos explicado durante la implementación de Asterisk, una de las cuestiones fundamentales de un sistema de autoatención pasa por su diseño.

También es importante aclarar que no debemos confundir esta función con otra muy difundida y empleada, el **IVR (Interactive Voice Response)**. La autoatención es parte del IVR, pero no es un IVR. Un IVR tiene un diseño más complejo y costoso, e involucra accesos a aplicaciones del backoffice, especialmente, a bases de datos. Con Asterisk también podemos implementar un IVR, pero este es un tema que queda fuera del alcance de este libro.

CONSIDERACIONES DE DISEÑO

Al diseñar un sistema de autoatención (AA), es importante tener en mente que este será utilizado por clientes/usuarios, y no solo debe conformar las expectativas técnicas y estéticas del implementador. No podemos olvidar que las personas llaman por teléfono para hablar con un ser humano y, en general, toleran estos sistemas porque no tienen opción. Por eso, una buena práctica es incluir la opción de transferir la llamada a un representante en algún punto del árbol, cercano al ingreso; por ejemplo, luego de pasar por las opciones más empleadas. Si estas son muchas, algunos clientes abandonarán la empresa y, si hay competencia, quizá no vuelvan a llamar. Entonces, otra buena práctica es que el sistema sea lo más simple posible.

No pretendemos ser exhaustivos en las consideraciones, pero podemos sugerir una

HISTORIA DE LA TELEFONÍA

Por el año 1854, el inventor italiano Antonio Meucci construyó un teléfono para comunicarse con su esposa, que, enferma, estaba postrada en la planta alta de su casa, en Nueva York. Como no tenía dinero suficiente, no pudo patentar su invento.

lista de los elementos que no pueden faltar en un **sistema de autoatención**:

- Transferencia a un operador humano.
- Saludos de acuerdo con la hora del día.
- Agradecimientos.
- Presentación de la compañía.
- Avisos con horarios de atención y cierre.
- Permitir la llamada a un interno, si este es conocido.
- Listado de extensiones de los departamentos clave (ventas, contaduría, etc.).
- Acceso a servicios como directorio de la empresa, envío de fax, etc.
- Música en espera con variaciones.
- Mensajes de publicidad de los productos de la compañía o de la compañía misma.
- Habilitar el retorno para escuchar las opciones.
- Permitir la recepción de mensajes de voz para el destino, si este se encuentra ocupado.
- Reproducción de mensajes de cortesía (ocupado, no disponible, etc.).

Si bien los saludos pueden cambiar según la hora del día o el día de la semana, no modifican la estructura del menú. También debemos saber que si se saluda al acceder al sistema y se retorna a escuchar las opciones, es más elegante no volver a reproducir los mensajes.

Cuando planeamos las opciones, tenemos que considerar que las personas no ponen mucha atención a lo que se dice, sino que están esperando conocer qué número deben presionar; por eso, es aconsejable informar primero el destino y después el número. Por ejemplo: "Para comunicarse con planta, presione 4".

Además, las opciones más importantes desde el punto de vista de quien llama deben estar al principio del árbol. Por ejemplo, si existe un directorio, podemos incluirlo al comienzo de la grabación para darlo a conocer al público, y dar las opciones de transferencia al informar las extensiones clave de la compañía.

Otro tema para considerar es el tratamiento de las entradas inválidas y los tiempos de espera excesiva o **timeouts** del sistema.

Una vez que finalizamos el diseño, podemos realizar su implementación en el Dialplan. Antes de escribir el código, necesitamos unos

COMPATIBILIDAD

Asterisk soporta el protocolo SMDI (Simplified Message Desk Interface), que permite la interoperabilidad entre sistemas de telefonía y sistemas de VoiceMail. Como es dependiente de las implementaciones propietarias, requiere cuidado al usarlo.

componentes esenciales que serán empleados al momento de programar la lógica. Estos elementos, denominados **prompts**, son los mensajes que utilizaremos en el sistema de autoatención.

PROMPTS

Los mensajes que vamos a utilizar en el sistema de autoatención son de vital importancia, porque serán la imagen de la empresa ante los usuarios que llamen. Es decir, es el primer contacto que tendrán los usuarios con la empresa.

El formato preferido para grabarlos es **WAV**, y la única combinación que funciona en Asterisk, hasta ahora, es **monoaural**, en **16 bits** y **8 KHz**.

Si bien podemos grabar los mensajes en estudio o con algún software como **Audacity**, la manera más directa y sencilla de hacerlo es utilizando la aplicación **Record()** de Asterisk. En el siguiente ejemplo, explicamos su funcionamiento:

...

```
[grabador]

exten => 665,1,Playback(inicio-grabacion)

exten => 665,n,Record(saludo-uno.wav)

exten => 665,n,Wait(2)

exten => 665,n,Playback(saludo-uno)

exten => 665,n,Hangup
```

...

Cuando se llama a la extensión **665**, este código reproduce un mensaje que alerta sobre el inicio del período de grabación; luego, graba un mensaje en formato **WAV**; al finalizar (por tiempo), espera 2 segundos y procede a repetir el mensaje recién grabado. Después, termina la llamada.

IMPLEMENTACIÓN EN EL DIALPLAN

A continuación, presentamos un ejemplo de aplicación de un sistema de autoatención que sigue los parámetros explicados anteriormente:

```
[Menu_Principal]
```

BUENA PRÁCTICA

Asterisk puede manejar el hecho de que un número de opción se solape con el de una extensión, pero esto genera una demora al esperar el lapso de tiempo de guarda del interdígito, antes de proceder. Por eso, aconsejamos utilizar extensiones con diferentes números que las opciones.

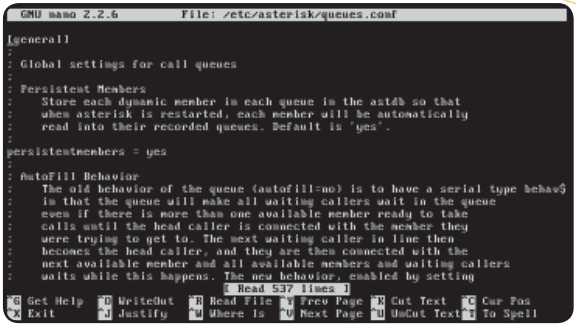


Figura 6. En este archivo se implementan las colas de espera.

```
exten => s,1,Answer()

same => n,Set(TIMEOUT(digit)=2)

same => n,Wait(1)

same => n,GotoIfTime(9:00-17:00,mon-fri,*,*?laboral:no_laboral)

same => n(no_laboral),Background(fuera_horario)

same => n,Goto(opciones_menu)

same => n(laboral),Background(en_horario)

same => n,Goto(opciones_menu)

same => n(opciones_menu),Background(menu_principal)

same => n,WaitExten(3)

same => n,Goto(0,1)

exten => 1,Goto(Filas,9001,1)

exten => 2,Goto(Filas,9002,1)

exten => 3,Background(dire_fax)

exten => n,Goto(s,opciones_menu)

exten => #,Directory()

exten => 0,Dial(SIP/operador)

exten => i,Playback(entrada-invalida)

same => i,Goto(s,opciones_menu)

same => t,Goto(0,1)

exten => _1XX,1,Verbose(1,LLamar a una extensión que inicia con '1')

same => n,Goto(Internos,${EXTEN},1)
```

Antes de cada selección, es posible agregar alguna locución orientativa. Por cuestiones de

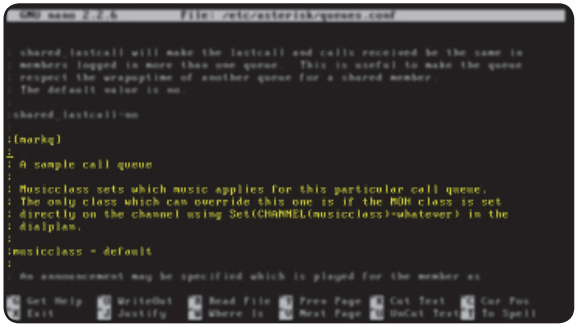


Figura 7. La definición de una cola se inicia con su nombre encerrado entre corchetes ([ ]).

legibilidad, hemos preferido no sobrecargar el código, por eso evitamos incorporar estas líneas, que puede generar confusión.

La mayoría de las aplicaciones que se emplean en esta porción de Dialplan fueron estudiadas en secciones y capítulos anteriores. Recordamos que la aplicación llamada **Wait(T)** permite incluir una demora en segundos igual a su argumento **T**.

A continuación, veremos brevemente qué significa el argumento **Filas**. Este argumento se refiere a una **Queue**. Las queues son las colas de espera que se pueden definir en Asterisk para estacionar las llamadas. En este caso se han definido, previamente, dos colas de espera: una para el área de ventas y otra para el área de soporte. El comportamiento y la configuración de estas colas se definen mediante el archivo **queues.conf**, y su definición en el Diaplan se muestra en la siguiente porción de código:

...

```
[Filas]

exten => 9001,1,Verbose(2,${CALLERID
(all)} ingreso a la fila de soporte)

same => n,Queue(soporte)

same => n,Hangup()
```

COLAS DE ESPERA

Para obtener más información sobre la configuración y el tratamiento de las queues o colas de espera, sugerimos consultar el sitio oficial de Asterisk, www.asterisk.org, o la wiki https://wiki.asterisk.org/wiki/display/AST/Using+queues.conf.

```
exten => 9002,1,Verbose(2,${CALLERID
(all)} ingreso a la fila de ventas)

same => n,Queue(ventas)

same => n,Hangup()

...

[internos]

include => Filas
```

Excede a los alcances de este libro el desarrollo del tema de las filas o colas de espera, pero para su comprensión basta saber que, a medida que las llamadas van llegando, se van estacionando en estas filas.

Estas tienen una estrategia FIFO (la primera que llega es la primera que se atiende), y en su configuración se establece el número máximo de llamadas que pueden contener (lo que usualmente se informa a los usuarios con un mensaje que dice: "Todos nuestros operadores se encuentran ocupados"). La llamada a las filas se realiza invocando a la aplicación **Queues(fila)**, cuyo argumento es la fila definida en el archivo **queues.conf**.

Es muy importante que realicemos diferentes pruebas exhaustivas con estas configuraciones a los efectos de determinar que, dentro del sistema de autoatención, únicamente es posible realizar aquellas acciones que han sido debidamente planeadas.

CAMINO DE LAS LLAMADAS

Las llamadas que ingresan al sistema son enviadas al contexto definido, cualquiera sea el canal por donde hayan entrado.

Esto quiere decir que la totalidad de las llamadas deben ser dirigidas al sistema de autoatención, independientemente del nombre del contexto y del nombre de la extensión que posean.

En la siguiente línea de código, mostramos una manera posible de hacerlo:

```
...

[RTC]

exten                    =>
```

IVR

Un sistema de respuesta de voz interactivo es mucho más que un sistema de autoatención. Entre otras particularidades, está la integración con una base de datos. Suele confundirse con un autómata que atiende en vez de una persona, pero no son lo mismo.

```
41810101,1,1Goto(Menu_Principal,s,1)
```

...

En este ejemplo podemos observar que aquellas llamadas que ingresan al número de extensión **41810101** son dirigidas al menú principal del sistema de autoatención. Lo mismo sucedería si en este código reemplazamos el número de extensión por uno diferente.

Aplicación **MeetMe()**

La aplicación **MeetMe ()** permite que varios usuarios formen una conferencia de audio. Sus características principales son:

- Capacidad de crear conferencias protegidas con contraseña.
- Administración de conferencias: eliminar participantes, bloquear conferencias, etc.
- Creación dinámica y estática de conferencias.

Para su configuración utilizamos el archivo **meetme.conf**, en el cual podemos definir las salas de conferencias virtuales y las contraseñas de ingreso a ellas, entre otras opciones.

A continuación, vemos la definición de una sala sin contraseña a la que se accede llamando a la extensión **500**:

```
[rooms]

conf => 500
```

Como siempre, cuando modificamos la configuración de aplicaciones o parámetros del sistema, es necesario recargar Asterisk para que pueda volver a leer los archivos de configuración.

La sintaxis de la aplicación es la siguiente:

```
MeetMe(sala,opciones,contraseña)
```

Para habilitar esta sala, debemos hacer la siguiente configuración en el Dialplan:

...

```
exten => 500,1,MeetMe(500,i,)
```

...

VISUALIZAR LAS OPCIONES

Para visualizar las opciones que pueden utilizarse con una aplicación Asterisk, podemos emplear el comando CLI (Core Show Application). Por ejemplo: `*CLI> core show application MeetMe.`

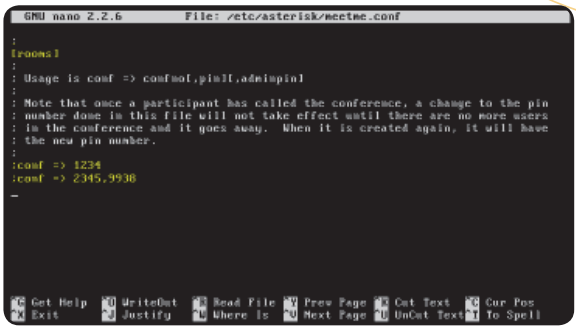


Figura 8. La implementación de las salas es realmente muy sencilla.

Cuando los usuarios llaman a la extensión **500**, si se ha configurado una contraseña, esta será requerida (en el ejemplo no se la solicita) y, luego, se los sumará a la conferencia.

La opción **i** permite que, cuando un usuario ingresa a la conferencia, los demás escuchen un mensaje de anuncio de su ingreso, virtualizando la situación de arribo de una persona a una sala real de conferencia.

En el ejemplo siguiente, veremos cómo limitar la cantidad de participantes que aceptamos en una sala, introduciendo el uso de la aplicación **MeetMeCount()**:

...

```
exten => 500,1,MeetMeCount(500,PART
ICIPANTES)

same => n,GotoIf($[${PARTICIPANTES}
<= 5]?ingresa:rebota,1)

same=>n(ingresa),MeetMe(500,i,Clave3)
```

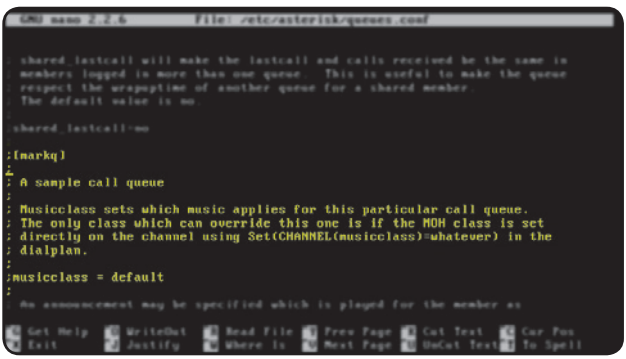


Figura 9. En el archivo preconfigurado, la sala 1234 se alcanza al llamar a la extensión 8600.

```
exten => rebota,1,Playback(sala_llena)

same => n,Hangup()

...
```

En esta porción de Dialplan, cuando se llama a la extensión definida para la sala en el archivo **meetme.conf**, mediante la aplicación **MeetMeCount()**, se asigna un valor igual al número de participantes en la sala **500** a la variable **PARTICIPANTES**. Esta se emplea en la siguiente secuencia para, luego de su evaluación, permitir o denegar el ingreso a la sala. Si el número de participantes es mayor que 5, se le informa al usuario que la sala está llena y se corta la llamada. Si es menor, se invoca la aplicación **MeetMe()**, y se le permite sumarse a la conferencia.

RESUMEN

Los servicios que en otras plataformas son costosos y complejos de implementar, con Asterisk resultan gratuitos y simples. En este capítulo explicamos algunos de los servicios clave de cualquier sistema de atención telefónica.

Capítulo 7

Interfaz de gestión

Vamos a explorar la AMI, la interfaz que permite la gestión de Asterisk.

RedUSERS.com

AMI

Asterisk Managment Interface, también conocidad por su sigla AMI, es una capa de software que permite monitorear en línea el sistema Asterisk y manejarlo, para hacerlo realizar diferentes acciones. Además, hace posible el desarrollo de aplicaciones, utilizando Asterisk como plataforma.

Mediante el archivo **manager.conf**, que se encuentra en el directorio **/etc/asterisk**, podemos habilitar y configurar esta funcionalidad. El listado que presentamos a continuación muestra los parámetros fundamentales por configurar y sus valores, para empezar a trabajar con esta herramienta:

```
[general]

enabled = yes

webenabled = yes

port=5038

bindaddr = 0.0.0.0

[cuenta]

secret=clave

read=system,call,log,verbose,agent,user,config,dtmf,reporting,cdr,dialplan

write=system,call,agent,user,config,command,reporting,originate
```

Esta característica no se encuentra habilitada por defecto, porque salvo que se precise desarrollar alguna aplicación que deba comunicarse con Asterisk, a nivel de software, no es necesaria y le pone cierta carga adicional al sistema.

No es una funcionalidad clave para la tarea de Asterisk, sino para su comunicación con otros productos o desarrollos de software, en especial, los de gestión.

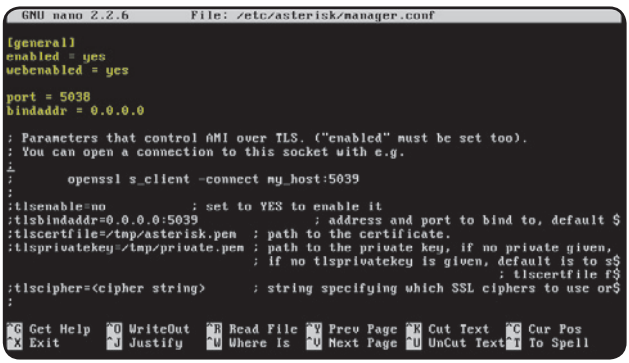


Figura 1. La API de Asterisk requiere que se la habilite antes de poder usarla.

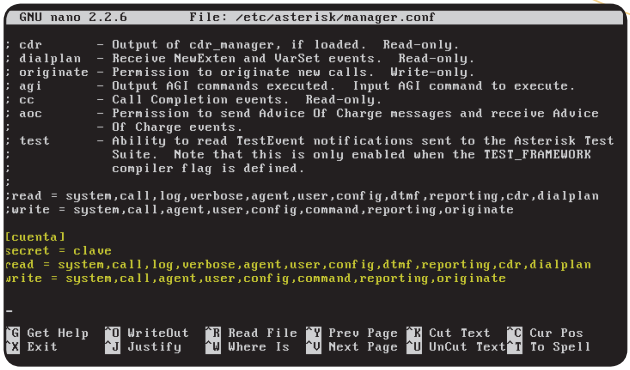


Figura 2. Creamos una cuenta para poder conectar con la API.

También, necesitamos revisar los valores del archivo **/etc/asterisk/http.conf** para confirmar que sus parámetros tengan los valores que mostramos a continuación:

```
[general]

enabled = yes

enablestatic=yes

bindaddr = 0.0.0.0

biindport=8088
```

Con esto, habremos habilitado el servidor HTTP incluido con el sistema Asterisk.

Una vez realizadas estas tareas de preparación, vamos a confirmar la conectividad con la AMI a través de los dos métodos más empleados: TCP y HTTP. A continuación, presentamos dos **Paso a Paso**, donde explicamos cada uno de estos procedimientos en detalle. Veremos que es algo muy simple.

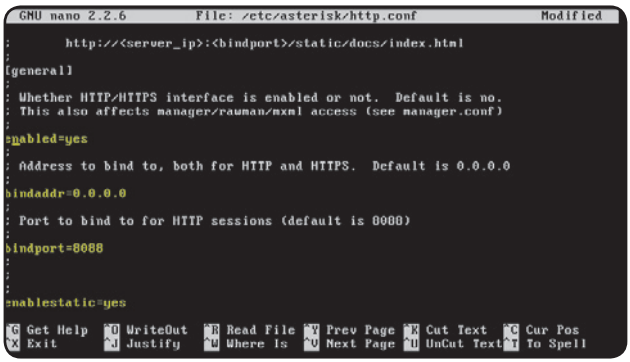
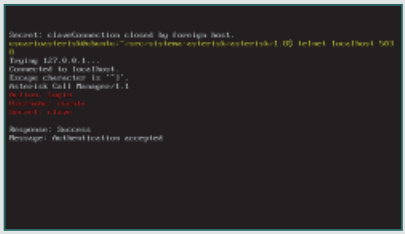


Figura 3. El servidor HTTP incluido en Asterisk se maneja desde el archivo `/etc/asterisk/http.conf`.

PASO A PASO /1
Conexión con AMI a través de TCP

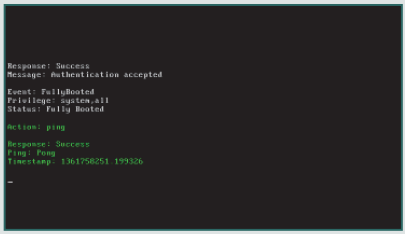
1



Conéctese a través del puerto 5038: **`# telnet localhost 5038`**. A continuación, ejecute un login:

```
Action: Login
Username: cuenta
Secret: clave
RETURN
RETURN
```

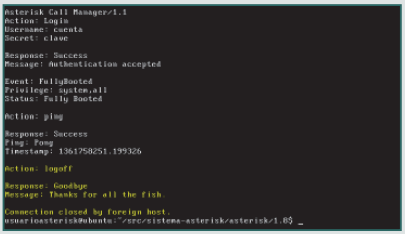
2



Ahora, ejecute un ping:

```
Action: ping
RETURN
RETURN
```

3



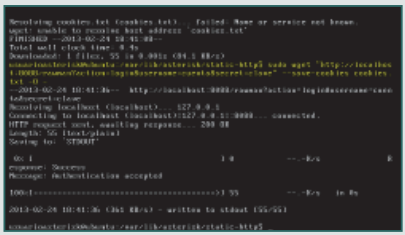
Para finalizar, ejecute un logoff:

```
Action: logoff
RETURN
RETURN
```

PASO A PASO /2
Conexión con AMI a través de HTTP

La conectividad AMI a través de HTTP permite la integración de llamadas de control a Asterisk desde un web service.

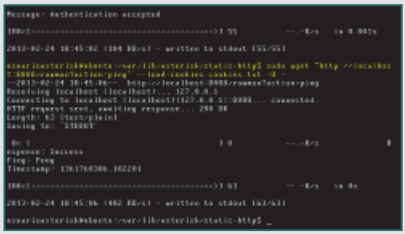
1



Ingrese el siguiente comando para hacer un login:

```
# wget "http://
localhost:8088/
rawman?action=login&
username=cuenta&secret=
clave"
--save-cookies cookies.
txt -0 -
```

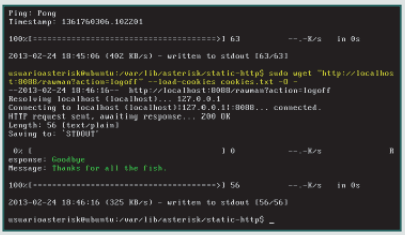
2



Ahora, ingrese el siguiente comando para realizar un ping:

```
# wget "http://
localhost:8088/
rawman?action=ping" --load-
cookies cookies.txt -0 -
```

3



Por último, ingrese el siguiente comando para efectuar un logoff:

```
# wget "http://localhost:8088/
rawman?action=logoff" --load-
cookies cookies.txt -0 -
```

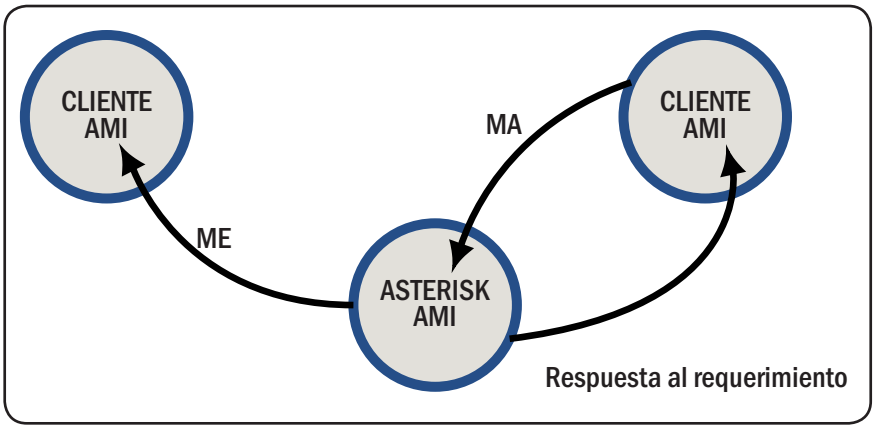


Figura 4. **Asterisk se reporta al cliente ante eventos del sistema y se le pueden solicitar acciones.**

La conectividad AMI a través de HTTP permite la integración de llamadas de control a Asterisk desde un web service.

Dentro del archivo de configuración **manager.conf**, se encuentra la sección **[general]**, donde, como en otras configuraciones, ingresamos los valores globales que afectarán el funcionamiento de AMI. Las restantes secciones son cuentas de usuario con sus contraseñas (en el ejemplo hemos configurado una cuenta con contraseña para probar la conectividad) y permisos. También será necesario configurar el servidor web incluido en Asterisk, para poder realizar conexiones por medio de este método.

PROTOCOLO

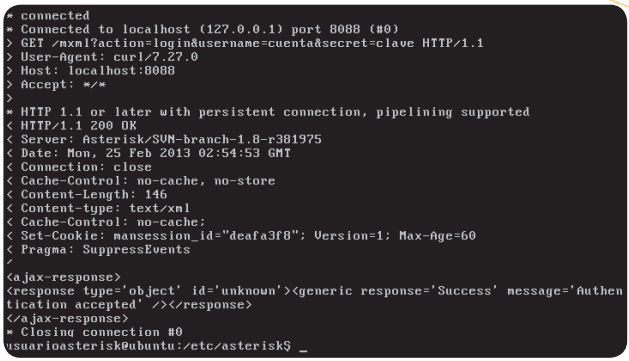
Entre el software cliente y la interfaz AMI de Asterisk se establece un diálogo que está reglamentado por un protocolo compuesto por dos tipos de mensajes: **Manager Events** (ME) y **Manager Actions** (MA).

Los Manager Events son mensajes que viajan desde la AMI al cliente y reportan eventos

MÁS SOBRE MANAGER.CONF

Para conocer más sobre las diferentes opciones que se pueden configurar en el archivo manager.conf podemos consultar la wiki: https://wiki.asterisk.org/wiki/dosearchsite.action?queryString=manager.conf&where=AST&typ

Figura 5.
Respuesta de AMI para la codificación `mxml`.



ocurridos en el sistema. Los Manager Actions son mensajes de respuesta que van desde el cliente hacia la AMI en Asterisk. El funcionamiento es similar, en el concepto, al del protocolo SNMP.

En la Figura 4 podemos ver la relación entre estas dos entidades y los tipos de mensajes que viajan entre ellas.

Cuando empleamos una conexión HTTP contra la AMI, se presentan dos formas de autenticación:

- `Action Login`
- `HTTP digest`

El primer caso corresponde al ejemplo empleado al principio del capítulo. Para el segundo método, AMI presenta tres tipos de codificación:

- `-/rawman`
- `-/manager`
- `-/mxml`

En el despliegue del siguiente comando podemos apreciar el tipo de codificación `/mxml`:

```
# curl -v "http://localhost:8088/
mxml?action=login&username=cuenta&
secret=clave"
```

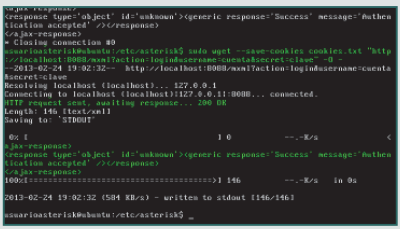
MÁS SOBRE HTTP.CONF

Para conocer más sobre las diferentes opciones que se pueden configurar en el archivo http.conf podemos consultar la wiki: https://wiki.asterisk.org/wiki/dosearchsite.action?queryString=http.conf&where=AST&type=&lastModified=&contributor=&contributorUsername=.

En el método de conexión HTTP, los ME no son asincrónicos, como en el método TCP; deben ser buscados a través de una consulta que tenemos que realizar explícitamente a la AMI de Asterisk. En el siguiente **Paso a Paso** vemos un ejemplo de este procedimiento:

PASO A PASO /3
Manager Events con AMI a través de HTTP

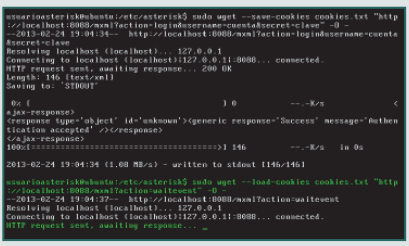
1



Ingrese el siguiente comando para hacer un login:

```
# wget
--save-cookies cookies.txt
\>"http://localhost:8088/mxml?act
ion=login&username=cuenta&secret
=clave" -O -
```

2



Ahora, ingrese el siguiente comando para efectuar un ping:

```
# wget
--load-cookies cookies.
txt "http://localhost:8088/
mxml?action=waitevent" -O -
```

RESUMEN

En este capítulo exploramos la interfaz de software que ofrece Asterisk para que los desarrolladores de aplicaciones puedan dialogar con el sistema. También conocimos los principales métodos de conexión: TCP y HTTP.

Capítulo 8

Gestión de Asterisk

Veremos algunas herramientas para monitorear Asterisk y obtener información que luego usaremos en su gestión.

RedUSERS.com

Introducción

La información que genera un sistema de telefonía se utiliza para su mantenimiento, la solución de problemas y el análisis de dimensionamiento. Pero también contamos con información para la facturación y la carga de las llamadas realizadas a través del sistema, para dar seguimiento y auditoría del tráfico cursado, y hacer análisis de llamadas para la atención de cuestiones legales, entre otras posibilidades. A continuación, veremos cómo podemos obtener todos estos datos.

Monitoreo

Resulta muy importante conocer el estado de salud de los equipos de comunicaciones en general, y de las conexiones entre ellos en particular. Con esta finalidad, existe el protocolo **SNMP**, que se ejecuta en una consola de gestión, y permite obtener una foto del estado de la red, tomada a intervalos regulares.

SNMP es un protocolo muy antiguo que integra el conjunto de protocolos TCP/IP, cuya función es interrogar al sistema acerca de determinados parámetros (disponibilidad, ancho de banda consumido, errores en las interfaces, etc.) para que este reporte los eventos que suceden en él. Tiene diferentes versiones en uso, pero la más reciente es la **V3**, que posee características de seguridad en la comunicación entre los agentes SNMP y el servidor SNMP, que no contienen las anteriores.

A continuación, en el **Paso a Paso 1**, veremos el procedimiento para habilitar este protocolo, en su versión 2, para que Asterisk pueda usarlo, y nosotros, desde una consola de gestión, podamos conocer el estado del sistema.

Nos concentraremos en habilitar la plataforma para que sea gestionada por una consola SNMP con la versión del protocolo 2: **SNMPv2**.

En primer lugar, instalaremos el software SNMP en la plataforma sobre la que se ejecuta Asterisk, en este caso, Ubuntu Server

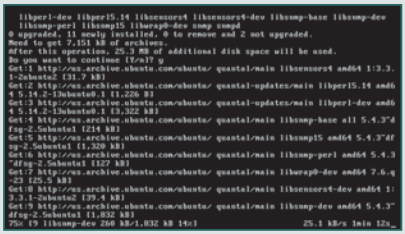
A continuación, una vez que instalamos el software SNMP en nuestra plataforma, tenemos que configurarlo. Para esto, debemos seguir el procedimiento explicado en el **Paso a Paso 2**. Veremos que es algo muy sencillo.

PLATAFORMA DE GESTIÓN

Está fuera del propósito de este libro la implementación y configuración de una plataforma de gestión. Si no disponemos de una, podemos instalar algún software de prueba; y si la tenemos, consultar la documentación para configurar el lado consola.

PASO A PASO /1
Instalación del software SNMP

1



Instale los paquetes necesarios a través del siguiente comando:

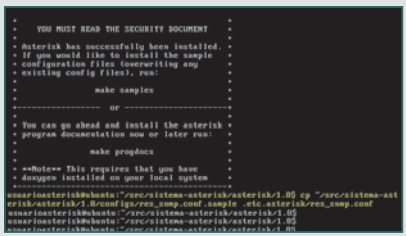
```
# Sudo apt-get install
snmp libsnmp-dev snmpd
```

2

Configure y verifique la disponibilidad del `res_snmp`:

```
# cd ~/src/sistema_asterisk/asterisk/1.8
# ./configure
# make menuselect
```

3

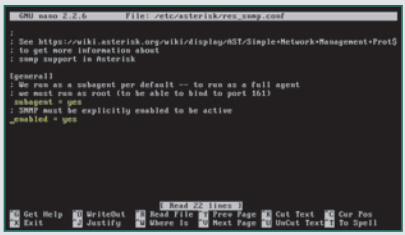


Realice la instalación mediante el comando: `# sudo make install.` Cree el archivo `res_snmp.conf:`

```
# cp ~/src/sistema_asterisk/
asterisk/1.8/configs/res_
snmp.conf.sample /etc/
asterisk/res_snmp.conf
```

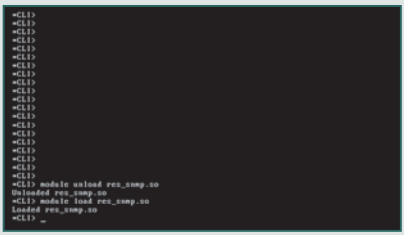
PASO A PASO /2
Configuración del SNMP

1



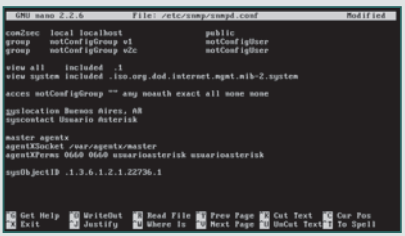
Edite el archivo **res_snmp.conf** mediante el comando: **# sudo nano res_snmp.conf**.A continuación, en la sección **[general]**, modifique los siguientes parámetros: **subagent=yes** **enabled=yes**

2



Inicie Asterisk a través del comando: **# asterisk -c.** Luego, recargue el módulo **res_snmp.so**: **CLI> module unload res_snmp.so** **CLI> module load res_snmp.so**

3



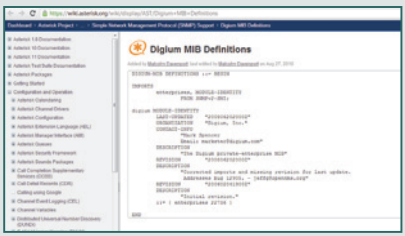
Cree el archivo de configuración SNMP, **snmpd.conf**: **#cd /etc/snmp** **# sudo cp snmpd.sample snmpd.conf**
Edite la configuración del **SNMPd**: **# sudo nano snmpd.conf**
En el archivo **snmpd.conf**, ingrese lo que muestra la imagen.

PASO A PASO /2 (cont.)

4

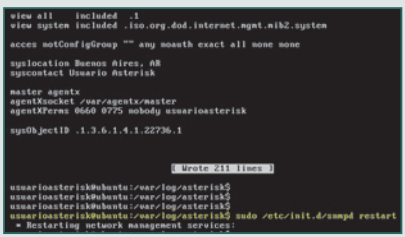
Acceda a **http:/wiki.asterisk.org/wiki/display/AST/Asterisk+MIB+Definitions** y copie el contenido a un archivo TXT: **/usr/share/snmp/mibs/asterisk-mib.txt**

5



Ahora ingrese en **http:/wiki.asterisk.org/wiki/display/AST/Digium+MIB+Definitions** y copie el contenido a un archivo TXT: **/usr/share/snmp/mibs/digium-mib.txt**

6



Importe las MIB:

```
# export MIBS=+ASTERISK-MIB
# export MIBS=+DIGIUM-MIB
```

Para finalizar, reinicie el demonio snmpd: **# sudo /etc/init.d/snmpd restart**

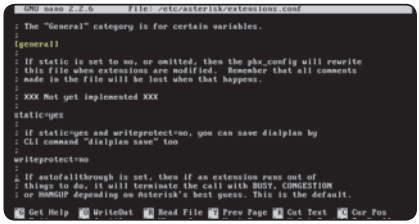


Figura 1. **Para usar las entradas del archivo debemos quitarle los comentarios.**

Luego de realizar esta configuración, podemos dar de alta el servidor Asterisk en una plataforma de gestión, utilizando **public** como valor de comunidad (**community**) y la dirección IP de nuestra interfaz de red.

Es importante aclarar que gran parte del proceso depende de los permisos bajo los que corren el demonio **snmpd** y la aplicación Asterisk.

Registro de **información**

Por lo general, la información de registro se encuentra en archivos que se conocen como **logs**. La mayoría de los sistemas suele contener esta clase de archivos y la funcionalidad para guardar en ellos la información pertinente. Por ejemplo, Linux, como sistema operativo, tiene un subsistema de **logging** destinado a recopilar diferente tipo de información, que abarca desde su funcionamiento hasta el historial de acceso de los usuarios.

Asterisk también cuenta con una herramienta para la creación, el mantenimiento y la consulta de estos **logs**, que a continuación veremos en detalle.

CONFIGURACIÓN Y ANÁLISIS DE EVENTOS

En Asterisk, el archivo para el registro de información se encuentra dentro del directorio **/etc/asterisk**, y su configuración consiste en indicar el tipo de información que se va a recopilar y el nombre de los archivos donde se almacenará. El nombre puede ser cualquier palabra, pero es conveniente que utilicemos alguna que refleje los datos que almacena el archivo. No olvidemos que estos subsistemas se configuran, en general, durante la instalación e implementación del sistema, y serán requeridos tiempo después. Tal vez otras personas tengan que

DEBUG I

Es la acción de reparar los problemas del sistema. Las sesiones de **debugging** son las que se llevan adelante cuando se está buscando la solución a algún inconveniente que tenemos en el sistema. No sirve para errores relacionados con el **Dialplan**.

revisarlos en busca de alguna pista para resolver un problema o para desarrollar alguna aplicación sobre Asterisk.

En la Tabla 1 detallamos los diferentes tipos de información que podemos incluir en los archivos de registro.

INFORMACIÓN DE REGISTRO

TIPO DE INFORMACIÓN	DESCRIPCIÓN
notice	Son mensajes que brindan información; por lo general, no requieren intervención.
warning	El evento que dispara estos mensajes puede alterar el normal funcionamiento del sistema. Su gravedad debe evaluarse sobre la base de la información contenida en el archivo, dado que a veces responden a configuraciones explícitas. Estos mensajes, en general, requieren intervención.
error	Estos mensajes informan de problemas en el sistema. Requieren intervención inmediata.
verbose	Es uno de los tipos preferidos por los implementadores. Permite registrar toda clase de inconvenientes, pero dejarlo activo sin utilizarlo puede causar problemas.
debug	Este tipo de información se utiliza durante el análisis de problemas. Su uso debería ser supervisado y solo emplearse cuando realizamos un análisis.
dtmf	Es un tipo específico de información involucrado en el encaminamiento interno de las llamadas.
fax	Otro tipo específico de información relacionada con la tecnología para atender un fax (recursos como res_fax_spandsp).
* (asterisco)	Esta selección hará que todos los tipos de mensajes se almacenen. Se podría emplear en caso de que no encontráramos la información necesaria, pero hay que ser cuidadosos.

Tabla 1. **Solo debemos almacenar la información que nos resulte de utilidad.**

Los tipos **debug** y **verbose** solo deben utilizarse dentro de sesiones de búsqueda y recolección de evidencia para el análisis de problemas, porque generan gran cantidad de información detallada y pueden llenar el disco en pocos días, dependiendo del tamaño del sistema.

El resto, salvo * (asterisco), puede usarse con discreción, pero siempre, como tarea del administrador de la plataforma. Es necesario controlar el tamaño de los archivos de logs en disco, porque aunque no detengan al sistema, pueden reducir su rendimiento a medida que crecen. Solo habilitemos el tipo de datos necesario y mínimo, y controlemos los archivos. Si estos crecen mucho o en forma acelerada, quitemos los tipos no estrictamente necesarios, o guardemos la información en un soporte de respaldo y borremos los archivos. El sistema volverá a crearlos.

La búsqueda y el análisis de los eventos en los archivos de registro se asemejan a la rutina que los administradores de sistemas realizan para resolver los inconvenientes de sus instalaciones. Lo importante es tener parámetros que permitan filtrar la información para reducir la búsqueda. Algunos de los más empleados son el horario y palabras clave como nombres de variables o subsistemas; todo lo que conduzca a reducir el tamaño de los datos por analizar. Las herramientas que se emplean para hacerlo van desde programas especializados en análisis de registros hasta los comandos de Linux: **less**, **tail** y **grep**.

Es posible enviar los eventos al registro central de la plataforma, el **syslog** o **rsyslog**. Para esto, debemos habilitar esta acción en los archivos **logger.conf** y **/etc/rsyslog.conf**, o **/etc/syslog.conf** y **/etc/rsyslog.d/50-default.conf**, si queremos redireccionar los mensajes a archivos específicos. En nuestro caso enviaremos todos los mensajes al archivo por defecto de syslog.

Los archivos **rsyslog.conf**, **syslog.conf** y **50-default.conf** no pertenecen al grupo de archivos de configuración de Asterisk, sino a la plataforma, en este caso, Ubuntu Server. Podemos ver los canales de registro que tenemos configurados en un sistema mediante el comando CLI **logger show channels**.

En La Figura 4 se muestra el canal de syslog Local7, que es otro de los canales de usuario para redireccionar los mensajes (van desde **Local0** hasta **Local7**).

DEBUG II

Cuando utilizamos el tipo de información debug, debemos recordar desactivarlo al finalizar, porque estos datos se guardan en un archivo que puede crecer hasta consumir todo el espacio de disco, lo que hará que el sistema se detenga abruptamente.

```
tail: cannot open `messages' for reading: No such file or directory
usuarioasterisk@ubuntu:/etc/asterisk$ cd /var/log/asterisk
usuarioasterisk@ubuntu:/var/log/asterisk$
usuarioasterisk@ubuntu:/var/log/asterisk$
usuarioasterisk@ubuntu:/var/log/asterisk$ tail -10 messages
[Feb 27 22:17:42] ERROR[1300] chan_sip.c: SIP TCP Server start failed. Not liste
ning on TCP socket.
[Feb 27 22:17:42] ERROR[1300] pbx_dundi.c: Unable to bind to 0.0.0.0 port 4520:
Address already in use
[Feb 27 22:17:43] NOTICE[1380] pbx_ael.c: Starting AEL load process.
[Feb 27 22:17:43] NOTICE[1380] pbx_ael.c: AEL load process: parsed config file n
ame '/etc/asterisk/extensions.ael'.
[Feb 27 22:17:43] NOTICE[1300] pbx_ael.c: AEL load process: checked config file
name '/etc/asterisk/extensions.ael'.
[Feb 27 22:17:43] NOTICE[1300] pbx_ael.c: AEL load process: compiled config file
 name '/etc/asterisk/extensions.ael'.
[Feb 27 22:17:43] NOTICE[1300] pbx_ael.c: AEL load process: merged config file n
ame '/etc/asterisk/extensions.ael'.
[Feb 27 22:17:43] NOTICE[1380] pbx_ael.c: AEL load process: verified config file
 name '/etc/asterisk/extensions.ael'.
[Feb 27 22:17:43] ERROR[1380] codec_dahdi.c: Failed to open /dev/dahdi/transcode
: Permission denied
[Feb 27 22:18:04] WARNING[1380] loader.c: Unload failed, 'res_' could not be fou
nd
usuarioasterisk@ubuntu:/var/log/asterisk$ _
```

Figura 2. Filtramos los últimos 10 registros en el archivo `messages`.

```
 GNU nano 2.2.6         File: /etc/asterisk/logger.conf

; this is another reason not to have debug mode on a production system unless
; you are in the process of debugging a specific issue.
;
;debug => debug
console => notice,warning,error,dtmf
;console => notice,warning,error,debug
messages => notice,warning,error
verbose => notice,warning,error,verbose
;full => notice,warning,error,debug,verbose,dtmf,fax

;syslog keyword : This special keyword logs to syslog facility
;
 syslog.local0 => notice,warning,error
;




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

Figura 3. Debemos habilitar la última línea para usar el syslog desde Asterisk.

```
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
registering pdu failed: 263!
.................. ]
Asterisk Ready.
*CLI> logger show channels
Channel                         Type      Status    Configuration
-------                         ----      ------    -------------
syslog.local17                  Syslog    Enabled     - NOTICE WARNING ERROR
/var/log/asterisk/verbose       File      Enabled     - NOTICE WARNING ERROR
ERBOSE
/var/log/asterisk/messages      File      Enabled     - NOTICE WARNING ERROR
                                Console   Enabled     - NOTICE WARNING ERROR
TMF

*CLI> _
```

Figura 4. El tipo de canal muestra el destino del mensaje generado por el logger de Asterisk.

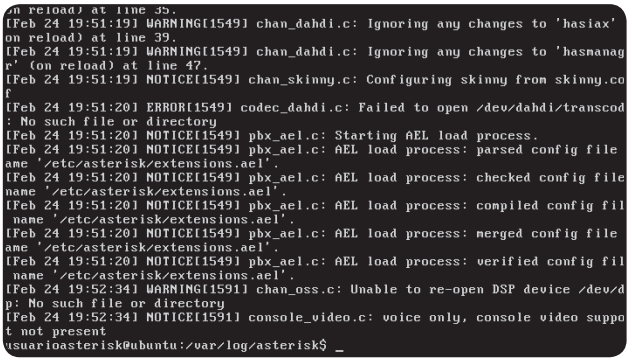


Figura 5. Los mensajes de Asterisk aparecen en el archivo syslog.

Otros **registros**

Asterisk también puede registrar el tratamiento de las llamadas. Estos registros se usan para analizar el funcionamiento interno del sistema, realizar el seguimiento de problemas, y recopilar información con el fin de facturar las llamadas y efectuar el **planeamiento de la capacidad**. A continuación, conoceremos algunos de estos registros.

REGISTRO CDR (CALL DETAIL RECORDS)

El registro **CDR** recopila eventos relacionados con la evolución de las llamadas. Si bien la información es bastante detallada, la organización y la segmentación de los eventos no son adecuadas para un eficiente análisis de problemas. En la siguiente dirección, podemos ver los campos utilizados en los CDR: **https://wiki.asterisk.org/wiki/display/AST/CDR+Fields**.

Aplicación CDR()

Esta aplicación nos permite acceder a los registros CDR y establecer los valores de sus campos desde el Dialplan. También podemos definir campos especiales, que no están establecidos por defecto.

Para configurarla, utilizamos el archivo **cdr.conf**, que, como los demás archivos de configuración,

MENUSELECT

Con la utilidad Menuselect, podemos configurar el sistema desde el punto de vista de la plataforma. Los componentes con el prefijo XXX son los que no se pudieron cargar por un problema de dependencias. Si los seleccionamos, veremos las dependencias y su estado.

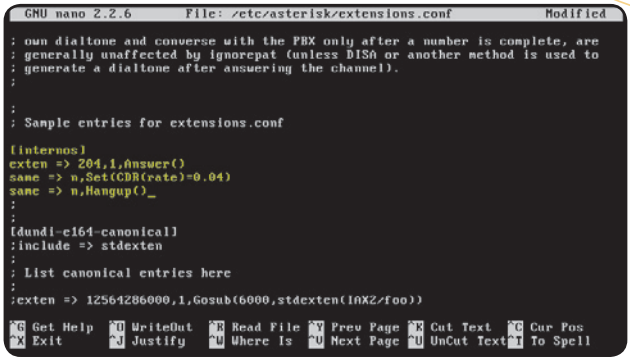


Figura 6. En el archivo `extensions.conf` encontramos usos sugeridos para la aplicación CDR().

posee identificadores de secciones. Los detalles de los parámetros del archivo pueden obtenerse en el sitio oficial de Asterisk. El archivo que viene configurado por defecto servirá para iniciar el servicio y registrar información básica de las llamadas.

Para que el servicio sea funcional, es necesario definir una plataforma donde almacenar los eventos. Para esto existen varias alternativas, pero solo veremos las más comunes. Las opciones para registrar los eventos generados por el subsistema CDR son las siguientes:

- `cdt_adaptive_odbc`
- `cdr_csv`
- `cdr_custom`
- `cdr_manager`
- `cdr_mysql`
- `cdr_odbc`
- `cdr_pgsql`
- `cdr_radius`
- `cdr_sqlite`
- `cdr_sqlite3_custom`
- `cdr_syslog`
- `cdr_tds`

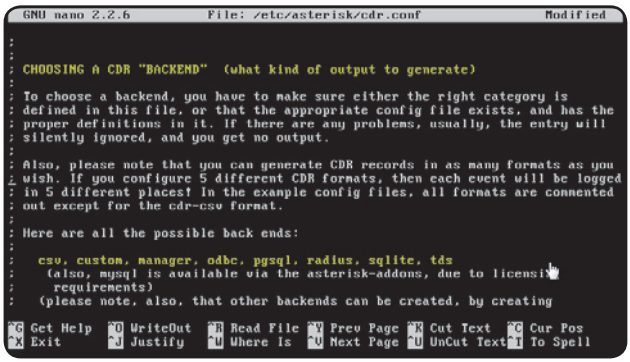


Figura 7. La forma de registrar los eventos CDR también se configura en el archivo `cdr.conf`.

Esta funcionalidad permite el registro en bases de datos de diferentes formatos. CSV (Comma Separated Values) es uno de los más utilizados; conoceremos algunas de sus características.

El archivo **cdr_csv** en el que se almacenan los registros se denomina **Master.csv** y se encuentra en el directorio **/var/log/asterisk/cdr-csv**. Una vez que se habilita el registro CDR, este formato queda operativo, sin necesidad de ninguna configuración, aunque existen opciones para controlar su comportamiento.

Para probar este almacenamiento, creamos la extensión **204** en el contexto **[internos]** y la llamamos con el comando **CLI> console dial 204@internos**. El resultado se observa en la Figura 8. Este formato puede recuperarse y es posible acceder a él con aplicaciones como Microsoft Excel.

REGISTRO CEL (CHANNEL EVENT LOGGING)

El sistema CEL nace a raíz de la necesidad de contar con una forma más detallada de registro de actividades. Si bien el CDR resulta muy útil y simple de implementar, solo es apropiado mientras la instalación tiene una tamaño reducido; a medida que se hace más compleja y se explotan más características, se hace necesario contar con herramientas que acompañen este desarrollo. El objetivo de este registro y su lógica de funcionamiento son similares a lo explicado para el CEL.

El archivo **cel.conf** es el que permite controlar el comportamiento del susbsistema y una serie de plataformas para registrar los eventos generados. Los formatos y plataformas admitidos son los que detallamos a continuación:

- **cel_odbc**
- **cel_custom**
- **cel_manager**
- **cel_pgsql**
- **cel_radius**
- **cel_sqlite3_custom**
- **cel_tds**

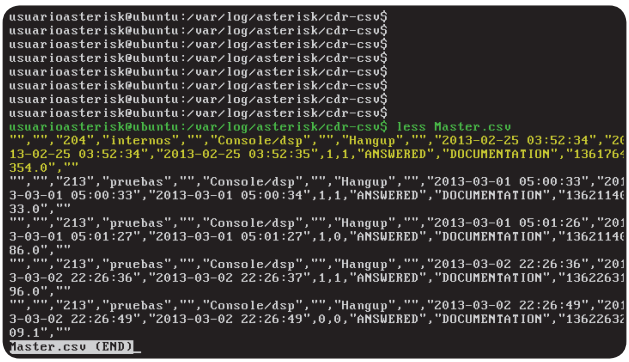
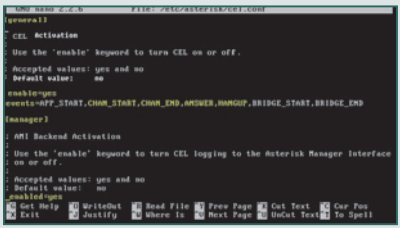


Figura 8. Los eventos se almacenan en formato CSV.

Los formatos **cel_manager** o **custom ver** emiten registros CEL sobre la AMI. Para poder utilizarlos, es necesario habilitar el uso de CEL y de AMI. En el siguiente Paso a Paso veremos en detalle cuál es el procedimiento que debemos seguir para poder habilitarlos.

PASO A PASO /3
Habilitar el backend cel_manager

1

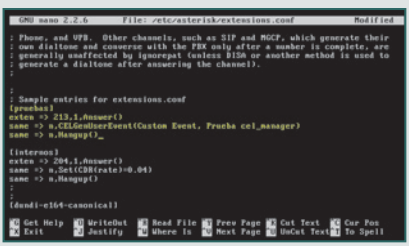


Edite el archivo **cel.conf** a través del comando: **# Sudo nano /etc/asterisk/cel.conf.**

Luego, habilite los siguientes parámetros en las secciones indicadas:

```
[general]
enable=yes
events=CHAN_START,CHAN_
END,ANSWER,HANGUP
[manager]
enabled=yes
```

2



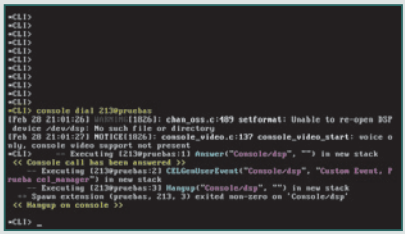
Genere una extensión de prueba en el Dialplan: **# sudo nano /etc/asterisk/extensions.conf.**

Agregue las siguientes líneas y guarde el archivo:

```
[pruebas]
exten => 213,1,Answer()
same =>
n,CELGenUserEvent(Custom
Event, Prueba cel_manager)
same => n,Hangup()
```

PASO A PASO /3 (cont.)

3



Ejecute Asterisk con consola local y nivel de verbose 3:

```
# asterisk -cvvv.
```

Por último, ejecute la extensión 213:

```
*CLI> console dial 213@pruebas
```

Los eventos CEL generados en esta prueba son almacenados por defecto en formato CSV.

Esto podemos verlo en la Figura 9, que muestra un despliegue del archivo **Master.csv**.

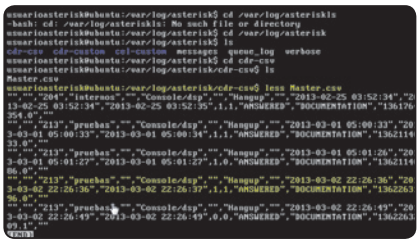
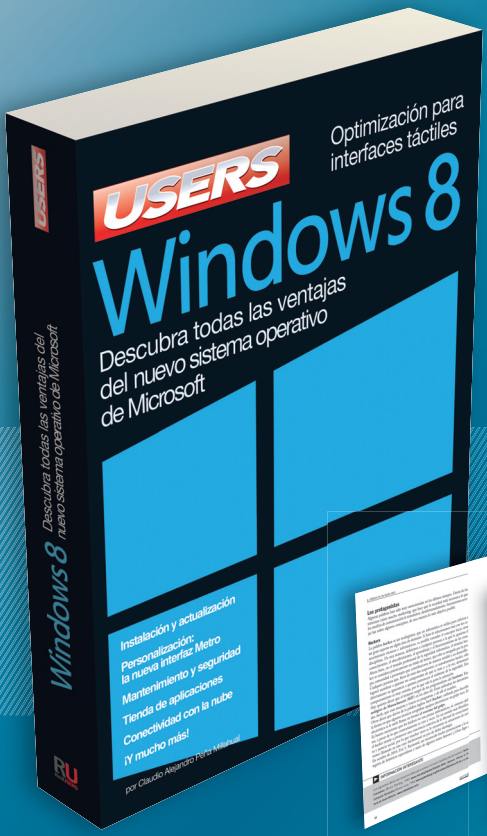


Figura 9. Los eventos CEL se almacenan en formato CSV.

RESUMEN

Para poder gestionar un sistema necesitamos monitorear su funcionamiento. Por eso, en este capítulo aprendimos a habilitar el monitoreo del sistema y el registro de información que se empleará, tanto para la gestión operativa y la planificación de capacidad, como para el mantenimiento y seguimiento de problemas.

DESCUBRA TODAS LAS VENTAJAS DEL NUEVO SISTEMA OPERATIVO DE MICROSOFT



Luego del lanzamiento de un sistema operativo sólido y veloz como Windows 7, Microsoft ha desarrollado un nuevo sistema que presenta una interfaz renovada, disponible tanto para equipos de escritorio y portátiles, como para tablets. Esta obra nos permitirá descubrir esta novedad, junto a otros aspectos en términos de seguridad y rendimiento, para aprovechar el potencial de Windows 8 al máximo.

» MICROSOFT / WINDOWS
» 192 PÁGINAS
» ISBN 978-987-1857-67-8

Técnico en REDES &SEGURIDAD