Instalación y Configuración de Asterisk

<Moisés Silva> moises.silva@gmail.com

TODO:

Tipo de Propuesta: Taller - 4 Horas

Track: Aplicaciones

Resumen:

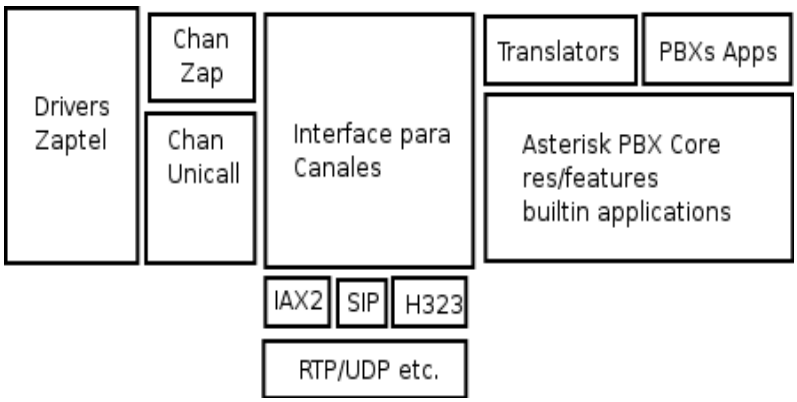
Es importante que las comunicaciones sean abiertas. Por lo tanto es importante apoyar el software libre. Mas aún el software libre que se encarga de hacer posible la comunicación. Asterisk se ha posicionado en pocos años como algo mas que un PBX; es un servidor de comunicaciones increiblemente flexible. Durante este taller mostraremos como configurar Asterisk para poner en marcha tu propio PBX. Explicaremos las diferencias y el papel que juegan los distintos protocolos que pueden ser utilizados para VoIP.

Durante esta sesión técnica se tocarán los siguientes puntos:

* Instalación de Asterisk en GNU/Linux (preferentemente Gentoo Linux)
* Configuración de extensiones SIP, IAX (si hay recursos, Zap, Unicall)
* Configuración de los patrones de marcado.
* Configuración de transferencia asistida y otros servicios (features.conf)
* Uso de los manejadores CDR.
* Uso apropiado de los CODECs
* Ejemplo de uso de diversas aplicaciones (Voicemail, MeetMe etc)
* Conexiones IAX2, tipos de autorización (RSA, plain, md5)

1. Instalación de Asterisk (y otros paquetes necesarios) (1 hora)

Asterisk es un PBX (Private Branch Exchange) basado completamente en software. Funcionalidad que durante mucho tiempo se hacia utilizando circuitos electrónicos de conmutación ha empezado a desarrollarse en software, volviendo los equipos mas flexibles, configurables y baratos. La instalación de Asterisk en GNU/Linux no difiere mucho de la instalación de cualquier otro servicio. Antes de instalarlo veremos como esta constituida la arquitectura general de Asterisk. En el siguiente diagrama he plasmado los conceptos que considero mas importantes y que tocaremos a lo largo de este taller.



Si nuestro deseo fuera unicamente instalar Asterisk para servicios puramente VoIP. La parte del diagrama que involucra chan_zap, chan_unicall y los drivers de zaptel desaparecerían. Sin embargo normalmente es necesario conectarnos a redes tradicionales como la red telefónica pública conmutada, RTPC o PSTN por sus siglas en inglés. Por esta razón necesitamos algunas dependencias mas. Los siguientes paquetes son necesarios para nuestra instalación.

sqlite 2.x (utilizaremos el sencillo manejador de CDR para SQLite)
zaptel-1.2.5 (drivers para el funcionamiento de tarjetas telefónicas PCI)
asterisk-1.2.7.1
kiax, kphone (o cualquier otro par de softphones SIP e IAX)

En gentoo basta utilizar los "use flags" apropiados y ejecutar el comando "emerge asterisk". Para ver las banderas disponibles y cuales se encuentran habilitadas ejecuta "emerge asterisk -pv", cualquier bandera extra que desees agregar o eliminar puedes hacerlo editando el archivo "/etc/portage/package.use". Dependiendo de las banderas gentoo decidira las dependencias adecuadas para Asterisk y los softphones.

Para instalarlo a mano, descomprime los archivos .tar.gz con el comando

tar -xvpzf <nombre del archivo>
Para el caso de zaptel el siguiente comando lo instalará:
make linux26
make install
Asterisk necesita:
make install

La compilación puede tomar minutos mas, minutos menos dependiendo de la velocidad de tu procesador, o de si compartes la compilación utilizando servicios como distcc. Mientras se compila, pasemos a revisar los archivos de configuración y directorios mas importantes que se instalarán.

/etc/asterisk

En este directorio se encuentran todos los archivos necesarios para configurar la gran cantidad de servicios que Asterisk provee. Tomaría mucho tiempo revisar todos estos servicios, por lo que nos concentraremos en los mas importantes.

asterisk.conf > configuraciones generales de la ubicación de directorios de configuraciones, módulos compilados, voicemails etc. En general es buena idea no modificar estas configuraciones, salvo casos especiales.

cdr.conf > Configuraciones referentes al "Call Detail Record". Los CDR son sumamente importantes para las compañias telefónicas. Modificar datos en este archivo puede repercutir en la integridad de los CDR si no estás seguro de lo que haces. Si tu instalación es unicamente de prueba, o los CDR no son materia importante, no hay problema.

codecs.conf > A menos que utilices SPEEX, o quieras hacer cosas especiales con la forma en la que los codecs se comportan, es mejor no modificar este archivo.

extconfig.conf > Archivo para mapear archivos de configuración hacia tablas en alguna base de datos, de forma que no es necesario guardar las configuraciones en archivos. Mas adelante hablaremos un poco mas acerca de esto.

extensions.conf > Tal vez el archivo mas importante de Asterisk. En este archivo se toman las decisiones de ruteo de las llamadas. Mas adelante veremos la sintaxis de este archivo y hablaremos sobre extensions.ael

features.conf > Este archivo es también muy importante. Permite habilitar y configurar servicios genéricos de un PBX como la transferencia asistida y monitoreo de llamadas.

iax.conf > Importante archivo para el funcionamiento del canal chan_iax que le permite a Asterisk interactuar con otros dispositivos IAX, incluyendo otros PBX Asterisk.

indications.conf > Configuraciones para los grupos de frequencias a utilizar para la indicacion del proceso de las llamadas. Los defaults suelen ser suficiente.

logger.conf > Que nivel de verbosidad deben tener los mensajes de log y a donde deben ser enviados.

manager.conf > Configuración del importante servicio AMI (Asterisk Manager Interface) que permite conectarnos a un socket TCP y manejar el PBX. De cierta forma se encuentra relacionado con el archivo http.conf, que provee de una interface para programar aplicaciones con AJAX que se comuniquen directamente con AMI.

modules.conf > Archivo sumamente importante. Determina que módulos serán cargados por Asterisk al iniciar. Es frecuente que cuando se instala asterisk por primera vez, no arranque debido a que no puede cargar un módulo para el que no tenemos soporte. Esto se soluciona comentando la línea del módulo en este archivo.

sip.conf > Análogo del archivo iax.conf para el protocolo SIP

zapata.conf > Configuración de los canales Zap. Las configuraciones de este archivo deben coincidir con el hardware instalado y la configuración del driver zaptel. Existe un archivo muy similar a este llamado unicall.conf, no incluido directamente con Asterisk.

/var/log/asterisk

Cuando hay problemas, este es el lugar en donde debemos buscar. En esta carpeta se encuentran los archivos de registro de las operaciones de Asterisk. Veamos que archivos podemos encontrar.

cdr,db > Este archivo se encuentra disponible si se cuenta con el CDR handler para la base de datos SQLite. El archivo contiene la base de datos de los registros de las llamadas.

event_log > Registro de eventos sucedidos en el PBX.

full > Creado con la intencion de contener todos los mensajes de debug del sistema.

messages > Contiene un listado de los mensajes de warning, debug y demas niveles de logeo.

queue_log > Archivo utilizado principalmente por la aplicacion app_queue.

/var/lib/asterisk

Directorio con archivos de audio, llaves RSA, scripts AGI (Asterisk Gateway Interface), base de datos astdb y archivos para el pequeño servidor HTTP para AJAM (Asynchronus Javascript Asterisk Manager). Aqui veremos una descripción de cada uno de los directorios, ya que los archivos pueden ser irrelevantes.

agi-bin/ > Aqui se contienen programas en C, PHP, Python o cualquier otro lenguaje con el que se pretenda interactuar desde Asterisk. Al final de este taller revisaremos con mas detalle AGI.

keys/ > Directorio que contiene llaves RSA para la autenticación de llamadas con el protocolo IAX2

sounds/ > Directorio con todos los sonidos que serán utilizados por aplicaciones como Playback() y Background()

mohmp3 > Archivos MP3 para MusicOnHold

Bien, ahora que la compilación ha terminado, y hemos terminado de revisar los archivos que fueron

instalados, podemos proceder a iniciar Asterisk. Varias distribuciones de GNU/Linux incluyen su propia forma de iniciar servicios. Para gentoo: /etc/init.d/asterisk start

Haciendolo a pata no es tan difícil. Iniciaremos el servicio como "root" (no intenten esto en casa!).

asterisk -vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvc

Si se equivocan en la cantidad de "v" vale queso y provocamos "kernel panic", asi que cuentenle bien todas las v, :p , ya hablando en serio, solo estamos usando dos opciones para iniciarlo. La "v" indica 1 incremento en la verbosidad (que tantos mensajes de notificaciones se observan en la consola), asi que mientras mas letras "v" más mensajes tenemos. La letra "c" nos proporciona una consola donde podemos ver mensajes de lo que sucede en el PBX y podemos ejecutar comandos de Asterisk. La linea de comandos en Asterisk es muy similar a bash, asi que es sencilla de usar. Si presionamos <TAB>, veremos los comandos disponibles, y desde luego también podemos completarlos. Ahora estrenemos nuestra linea de comandos ejecutando:

*CLI> sip show peers

Este comando nos muestra los "peers" disponibles en el sistema. No revisaremos con detalle lo que significa un "peer". De momento digamos que nos muestra las extensiones dadas de alta en el sistema que hablan el protocolo SIP. Existe un comando análogo para IAX2. Con esto ya tenemos un PBX instalado en nuestra computadora!. En la siguiente sección veremos como hacer algo útil con él :D

2. Patrones de marcado y configuración de extensiones SIP/IAX (1 hora)

Para los impacientes iremos directo a la acción y luego las explicaciones. Abre el archivo /etc/asterisk/extensions.conf con tu editor favorito (sugiero VIM ampliamente) y escribe:

```
[hello-world]
exten => _XX,1,Answer()
exten => _XX,2,Playback(hello-world)
exten => _XX,3,Hangup()
```

Guarda el archivo, abre sip.conf y escribe:

```
[general]
[general]
bindport=5060
bindaddr = 0.0.0.0
localnet = 192.168.1.0/24 ; aqui va la direccion de la red en la que te encuentras
allow=all
defaultcontext=unauthorized

[33]
type=friend
secret=verysecure
host=dynamic
nat=no
dtmfmode=info
qualify=yes
context=hello-world
```

Fijate que el parámetro "context" del registro [33] debe coincidir con el contexto que agregamos en extensions.conf. Ahora, como Asterisk guarda sus archivos de configuración en memoria al iniciar, los cambios que hemos hecho no han sido detectados. Para ello nos dirigimos a la consola y escribimos:

```
*CLI      > sip show peers
```

Y debe mostrarnos algo como esto:

Name/username	Host	Dyn	Nat	ACL	Port	Status
33	(Unspecified)	D			0	UNKNOWN

Esto nos indica que Asterisk ha detectado el nuevo registro de extensión. Ahora abre tu cliente SIP (kphone por ejemplo.), en kphone debes dirigirte a

"File > Identity..."

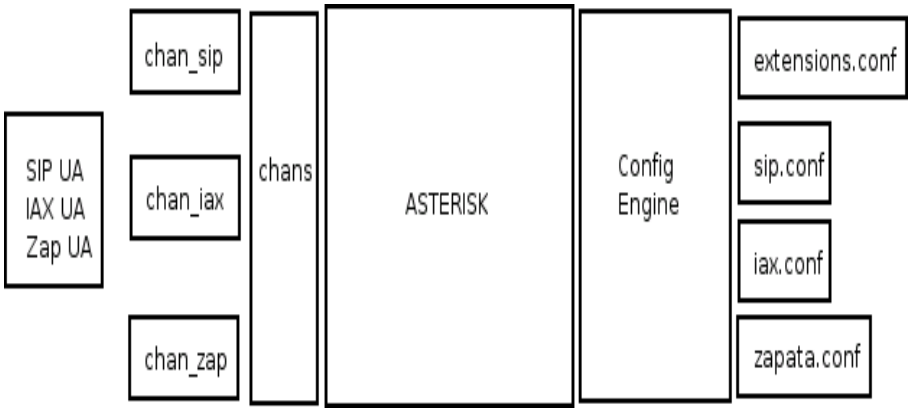
Un formulario te solicitará varios campos. Proporciona los siguientes:

Full Name: <tu nombre>
User Part of SIP URL: 33
Host Part of SIP URL: <la IP o nombre de dominio de la maquina que corre Asterisk>
Authentication User: 33

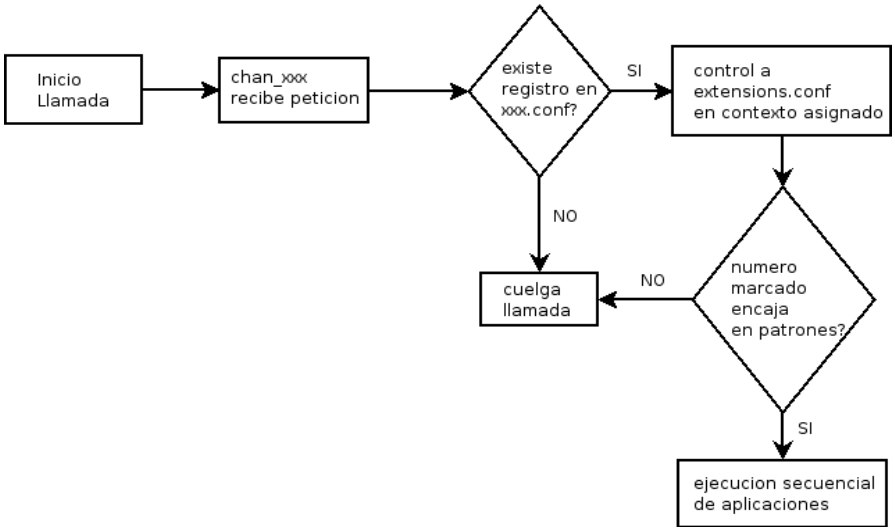
Es posible que KPhone requiera de reiniciarse para aplicar los cambios. Al reiniciar, si se encuentra bien configurado, te pedira el password para el registro. El password es el campo que indicamos como "secret" en la configuración del registro 33. Proporcionalo para finalizar el registro. En el momento en el que presiones "enter" después de haber ingresado el password, en Asterisk debe aparecer el intento de registro, con un mensaje indicando si fué exitoso, o falló (la causa mas común, teclear incorrectamente el password.).

Como usuario registrado puedes iniciar a hacer llamadas. Ahora teclea cualquier número de dos dígitos (66 por ejemplo) y presiona enter. Si tienes tu tarjeta de sonido bien configurada, o decidiste utilizar un teléfono IP en lugar de un softphone, debes escuchar un mensaje de una mujer de voz sexy diciendo: "Hello World". Y voilá, tenemos nuestro primer ruteo de llamada concretado.

Antes de continuar con un ruteo algo mas avanzado, observemos como Asterisk maneja la arquitectura de sus archivos de configuración.



Los canales chan_sip. chan_iax y chan_zap (asi como otras entidades en el sistema) tienen acceso a su archivo de configuración a través de Asterisk, quien a su vez tiene registrado un config engine que se encarga de obtener los datos. Este config engine puede acceder a una base de datos (postgresql, mysql etc) o directamente a los archivos contenidos en /etc/asterisk. Veamos ahora el flujo general de una llamada generada por un SIP UA (SIP User Agent).



En realidad es un poquito mas complejo que lo mostrado ;) , sin embargo para nuestros fines es suficiente

saber lo que el diagrama muestra. Para que una llamada pueda realizarse el UA (user agent) debe estar debidamente registrado. En el caso de Zap el registro es implicito puesto que la tarjeta PCI no pudo haber sido instalada ni conectada sola en la computadora :p

Una vez registrado el UA, puede proceder a solicitar el inicio de una llamada enviando un número. Esta petición es recibida por el módulo correspondiente de Asterisk (chan_zap, chan_sip, chan_iax, chan_xxx), quien a su vez revisa el archivo de configuración correspondiente (zapata.conf, sip.conf, iax.conf) para autorizar la llamada y decidir a que "contexto" de extensions.conf se delegará el ruteo de la llamada, así como otras configuraciones específicas de como tratar la llamada y al UA. Una vez delegado el ruteo al contexto, extensions.conf tendrá el completo control de la llamada, y es donde nosotros, como Administradores de Asterisk debemos decidir como rutear la llamada en base al número marcado y cualquier otro set de criterios que deseemos utilizar. Asi que, la primer pregunta que debemos hacernos es que deseamos hacer con nuestras llamadas y que numeración utilizaremos para hacerlo. Por ejemplo, simplificando un poco las cosas podemos decir que telmex decide en sus centrales que cuando reciba un numero iniciando con 01, seguido de 10 números, lo dirijira a un ruteo nacional, detectando posteriormente el código de estado, área hasta encontrar el destino final. Nosotros empezaremos con una red mas simple, sin involucrar otros servidores o conmutadores, así que pondremos una regla simple. Las extensiones internas de nuestro sistema tendrán dos dígitos. Asi que abrimos el archivo extensions.conf y escribimos:

```
[internal-extensions]
exten => _XX,1,Answer()
exten => _XX,2,Dial(SIP/${EXTEN})
exten => _XX,3,Hangup()
```

Los carcteres dentro de los corchetes indican el nombre del contexto. El inicio del contexto se encuentra justo debajo de su definición entre corchetes, y termina al iniciar otro contexto, o al terminar el archivo. Dentro del contexto deben encontrarse nuestras definiciones de ruteo (conocidas en Asterisk como extensiones), llamadas a aplicaciones, macros etc. El contexto que acabamos de definir unicamente permite, a los dispositivos configurados para usarlo, marcar números de longitud de 2 dígitos. La sintaxis para las extensiones es:

exten => <patron>, <prioridad>, <aplicación>

Patrón: La comprensión de los patrones es fundamental para organizar correctamente nuestro sistema y hacerlo funcionar de una forma optima y sencilla para nuestros usuarios. Al final de este documento se encuentra en el apéndice algo de teoría que tuve que analizar para desarrollar un ruteador de llamadas en PHP saltandome el reconocimiento de patrones de Asterisk, aqui solo veremos lo necesario para continuar. Un patrón de marcado en Asterisk se compone de una seríe de números y símbolos que representan rangos de números. De esta forma Asterisk decide que empezar a hacer con la llamada que está recibiendo.

Prioridad: Orden secuencial para la ejecución de aplicaciones de Asterisk sobre la llamada recibida.

Aplicación: Funcionalidad que se desea ejecutar sobre la llamada (conectar con otro dispositivo, reproducirle un

archivo de audio, colgarla etc.)

Analicemos el contexto [internal-extensions]. La primer linea dice:

exten => _XX,1,Answer()

Lo que indica que como primer aplicación a ejecutar para un número de dos dígitos Asterisk ejecutará la aplicación Answer(). Cada aplicación se ejecuta por un tiempo variable, dependiendo de objetivo de la aplicación. A la aplicación Answer solo le toma unos milisegundos. Su objetivo es simple, inicializar variables y datos importantes sobre la llamada. Es sumamente importante que siempre utilices esta aplicación al iniciar el ruteo de una llamada. El guión bajo antes de las dos X significa inicio de patrón. Es válido también omitir el guión bajo cuando se pondrá como patrón un número en particular, por ejemplo "exten => 12,1,Answer()".

La segunda línea del contexto dice:

exten => _XX,2,Dial(SIP/${EXTEN})

Esta línea indica que como segunda prioridad se ejecutará la aplicación Dial, recibiendo como primer argumento la cadena "SIP/${EXTEN}". La aplicación dial se encarga de conectar la llamada con otro punto. Sus argumentos pueden ser complicados, este es solo el caso más básico de conectar una llamada con otro punto. La cadena dice ${EXTEN} expande a ser el número marcado de 2 dígitos. De tal forma que si el UA envia el número 34, la aplicación Dial recibe el equivalente a Dial(SIP/34). Lo que provocaría que Asterisk intentara conectar la llamada con la extensión 34. Si no existen un registro en el archivo sip.conf con la extensión 34 registrada, la aplicación Dial inicializa una variable llamada ${DIALSTATUS} con el valor "CHANUNAVAIL" indicando que el canal solicitado no se encuentra disponible. El valor de esta variable es frecuentemente consultado después de las llamadas a Dial() para decidir que hacer con la llamada. Es frecuente también que si CHANUNAVAIL es el valor de la variable, a continuación se ejecute una aplicación como Playback() para reproducit un archivo de audio indicando que la extensión no existe o no se encuentra disponible.

Por último la línea

exten => _XX,3,Hangup()

le indica a Asterisk que como prioridad número 3 debe colgar la llamada.

De forma similar a como configuramos la extensión SIP, se configura la extensión IAX. En realidad el protocolo IAX es mejor conocido como IAX2, ya que la versión 1 del protocolo ya no se encuentra soportada. Sin embargo el archivo conserva el nombre iax.conf. Veamos una configuración típica.

La configuración de un cliente IAX muy similar a la configuración para un cliente SIP. Veamos un ejemplo de configuración.

```
[general]
bindport=4569
bindaddr = 0.0.0.0
allow=all
```

defaultcontext=unauthorized

[713]
type=friend
secret=verysecure
host=dynamic
context=hello-world

Existen sin embargo algunas pequeñas diferencias en algunos parámetros debido a que el protocolo IAX2 es enteramente diferente al protocolo SIP. Para que el teléfono SIP previamente configurado pudiera comunicarse con esta nueva extensión IAX tendriamos que modificar el contexto en extensions.conf de esta forma:

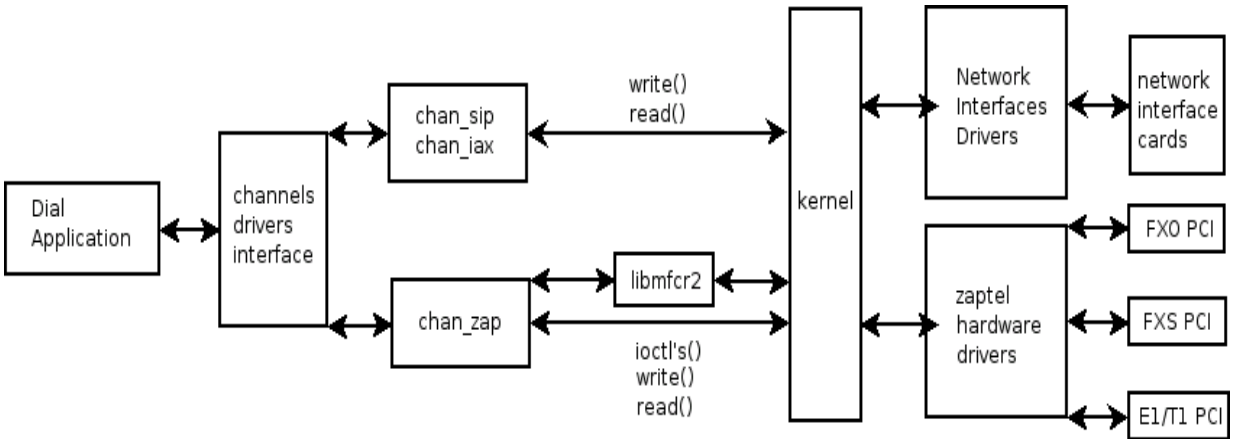
[internal-extensions]
exten => _XX,1,Answer()
exten => _XX,2,Dial(SIP/${EXTEN})
exten => _XX,3,Hangup()
exten => _XXX,1,Answer()
exten => _XXX,2,Dial(IAX2/${EXTEN})
exten => _XXX,3,Hangup()

3. Zaptel y Zapata. (1 hora o 30 minutos, depende de si se cuenta con tarjetas)

Al principio del documento mencionamos brevemente los canales Zap. Los canales Zap son la vía de comunicación entre Asterisk y el hardware PCI telefónico. El canal Zap se comunica directamente con el driver de las tarjetas telefónicas PCI (Zaptel). De tal suerte que cuando en un dial plan se escribe una línea como:

exten => _XXXXXXXX,2,Dial(Zap/1/${EXTEN})

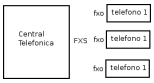
La petición de conexión es pasada por Asterisk hacia el módulo chan_zap que a su vez se comunica con el driver para indicarle que hable con el hardware para que la tarjeta PCI haga su trabajo telefónico para iniciar una llamada. Esta explicación la podemos apreciar mejor en el siguiente diagrama:



Tal vez en el diagrama he expresado una explicación que puede parecer mas profunda de lo necesario, sin embargo creo que no hace daño conocer como es que Asterisk encaja en todo el sistema operativo.

Ahora, tal vez sea mas sencillo entender por que es necesario que la configuración de zapata.conf coincida en algunos puntos con la configuración de los drivers de zaptel (/etc/zaptel.conf). En el diagrama se aprecian los módulos de zaptel mas importantes: FXO, FXS y E1/T1. Se encuentra fuera del objetivo de este documento describir a detalle cada uno de los módulos, sin embargo daré una descripción breve del objetivo y alcances de cada módulo.

FXS (Foreign Exchange Suscribers): Módulo capaz de generar señalización tipo FXS. La señalización FXS es propia de centrales telefónicas. Los puertos FXS son aquellos que proveen de tono de marcado proporcionando un nivel de voltaje y necesariamente conectados en su otro extremo hacia un puerto FXO. En nuestras casas, los puertos de los teléfonos son FXO y se encuentran conectados a los sockets de la pared, que van a dar hacia la central teléfonica que provee de señalización FXS.



FXO (Foreign Exchange Office): Módulo encargad de aceptar tono, enviar digitos y recibir llamadas. Usualmente destinados a ser nodos finales. La señalización FXO es propia de los teléfonos analógicos convencionales.

A continuación veremos una configuración clásica de una tarjeta FXO.

```
[channels]
echocancel=yes
echocancelwhenbridged=yes
echotraining=100
rxgain=18
txgain=2
relaxdtmf=yes
signalling=fxs_ks
callerid=no
usecallerid=no
restrictcid=no
threewaycalling=yes
transfer=yes
cancallforward=yes
callreturn=yes
group=2
immediate=yes
```

busydetect=no
callprogress=no
musiconhold=default
context=internal-extensions
channel=1-3

Hay una diferencia importante (aparte de la evidente diferencia de parámetros) entre el archivo de configuración zapata.con y los archivos sip.conf e iax.conf. Mientras en iax y sip el fin de un set de parámetros se encuentra determinado por el inicio de otro contexto. En zapata solo existe 1 contexto, el contexto [channels]. Y hacemos distinción entre parámetros que apliquen a un canal u otro en base al parámetro especial "channel". Cada vez que aparece "channel" se configura el canal o rango de canales especificado con los parámetros que están definidos antes. En este archivo hay un parámetro que es sumamente importante: "signalling". Este parámetro determina que señalización será utilizada con el puerto de la tarjeta PCI. Esta señalización debe ser coherente con la señalización declarada en el archivo de configuración del driver de la tarjeta. Veamos cual sería la configuración correcta de /etc/zaptel.conf para un archivo zapata como el anterior.

fxsks=1
fxsks=2
fxsks=3

Un archivo zaptel con estos 3 parámetros sería suficiente. Indicando que se utilizara señalización FXS kewl start, una variación de la señalización FXS con supervisión para desconexiones.

Como pudimos observar, hay una similitud muy importante entre zapata.conf, iax.conf y sip.conf. Esto es, todos contienen un parámetro conocido como "context" que determina donde iniciará el ruteo de las peticiones de llamada recibidas. La única diferencia es que nuestro contexto internal-extensions debe ser extendido un poco, debido a que a diferencia de SIP e IAX, las llamadas que recibamos por nuestra interface FXO no envian un número ruteable. La interface FXO unicamente recibe tono de timbrado, indicando que alguien quiere llamarnos, se supone que un teléfono convencional, ante tal señal timbraría y alguien lo contestaría. Sin embargo no conectamos un teléfono, contectamos un PBX. Como PBX es nuestra responsabilidad rutear la llamada. Tenemos basicamente dos opciones:

1. Enviar la llamada directamente a una extensión arbitraria
2. Realizar una contestación automática con una grabación que le pida a quien llama que marque la extensión deseada.

De momento solo observaremos la opción mas simple, la 1. Nuestro contexto internal-extensions quedaría asi:

[internal-extensions]
exten => s,1,Answer()
exten => s,2,Dial(SIP/33)
exten => s,3,Hangup()
exten => _XX,1,Answer()

```
exten => _XX,2,Dial(SIP/${EXTEN})
exten => _XX,3,Hangup()
exten => _XXX,1,Answer()
exten => _XXX,2,Dial(IAX2/${EXTEN})
exten => _XXX,3,Hangup()
```

Esta vez agregamos un patrón especial, "s". Este patrón especial indica que cuando no exista un número a rutear, se empezará en ese punto. Recapitulando. Cualquier dispositivo que tenga este contexto como inicio de ruteo tiene 3 posibilidades:

- marcar un número de 3 dígitos para extensiones IAX.
- marcar un número de 2 dígitos para extensiones SIP.
- cuando no se recibe número se marca a la extensión SIP 33 directamente.

4. Troncales IAX2. (1 hora)

En el lingo de las telecomunicaciones, una troncal es una conexión entre dos centrales telefónicas. Un PBX es una pequeña central telefónica. Por lo tanto si contamos con más de una computadora podemos crear un enlace telefónico entre ambas computadoras. En esta sección analizaremos como crear un enlace con una o mas centrales telefónicas mediante el protocolo IAX2.

Por si no lo había mencionado, IAX2 es el protocolo nativo de Asterisk, sus siglas significan Inter Asterisk Exchange. A diferencia de SIP, IAX es un protocolo pensado directamente en VoIP, por lo que no necesita puertos adicionales (SIP necesita conexiones adicionales RTP para llevar la voz, o el video). Decidí incluir unicamente troncales IAX debido a su simplicidad. Bien, para empezar necesitamos dos computadoras con Asterisk instalado. Llamaremos a la primer computadora "iaxprovider" y a la otra "iaxclient". Notese que los nombres son meramente ilustrativos, no existe un cliente y un servidor, ni un cliente ni un provider. En cuanto al protocolo se refiere ambos puntos son iguales. IAX2 soporta varios métodos de autenticación. Aqui usaremos el método de llave RSA, a mi parecer tal vez el más seguro. Empecemos por configurar "iaxprovider".

iax.conf en "iaxprovider"

```
[general]
port=4569
bindaddr=0.0.0.0

[iaxclient]
type=friend
host=dynamic
context=iaxclientcontext
auth=rsa
```

```
inkeys=iaxclientkey
notransfer=yes
```

extensions.conf en "iaxprovider"

```
[iaxclientcontext]
exten => _XX,1,Answer()
exten => _XX,2,Dial(SIP/${EXTENSION})
exten => _XX,3,Hangup()

[internal-extensions]
exten => _44XX,1,Answer()
exten => _44XX,2,Dial(IAX2/iaxprovider:[iaxproviderkey]@iaxclient.host.com/${EXTEN:2})
exten => _44XX,3,Hangup()
```

Como podrán ver unicamente le damos acceso a esta conexión a marcar extensiones tipo SIP. Ahora configuraremos la computadora "iaxclient"

iax.conf en "iaxclient"

```
[general]
port=4569
bindaddr=0.0.0.0

[iaxprovider]
type=friend
host=dynamic
context=iaxprovidercontext
auth=rsa
inkeys=iaxproviderkey
notransfer=yes
```

extensions.conf en "iaxclient"

```
[iaxprovidercontext]
exten => _XX,1,Answer()
exten => _XX,2,Dial(IAX2/${EXTENSION})
exten => _XX,3,Hangup()

[internal-extensions]
exten => _33XX,1,Answer()
```

exten => _33XX,2,Dial(IAX2/iaxclient:[iaxclientkey]@iaxprovider.host.com/${EXTEN:2})
exten => _33XX,3,Hangup()

Del lado del cliente solo damos acceso a extensiones IAX. Ahora, no hemos terminado queda pendiente un punto importante. La generación de las llaves RSA. Para ello primero analicemos lo que hemos hecho.

1. En la máquina iaxprovider creamos un contexto nuevo en iax.conf con el nombre iaxclient y en la máquina iaxclient creamos de igual forma un contexto, excepto que con el nombre iaxprovider. Estos contextos son las definiciones para darle entrada a otro dispositivo a nuestro sistema telefónico. Los otros puntos importantes dentro de estos contextos son context y inkeys, el primero determina donde se iniciara el ruteo de las llamadas recibidas por este host. El otro parámetro es el nombre de una llave RSA pública con la que autorizaremos al host a entrar a nuestro sistema. Las llaves RSA deben estar ubicadas en la carpeta /var/lib/asterisk/keys a menos que algo diferente se especifique en el archivo de asterisk.conf
2. Creamos un contexto de ruteo para recibir las llamadas de nuestros hosts. En iaxprovider unicamente damos acceso a extensiones SIP de 2 dígitos. En iaxclient damos a extensiones IAX de 2 digitos.
3. Creamos un contexto para poder realizar llamadas hacia el host en cuestion. La sintaxis para llamar a otro host iax es:
Dial(IAX2/<usuario>:<[llaveRSA]>@<host>/<extension deseada>)
Como se puede ver, la extensión deseada que especificamos es ${EXTEN:2} esto significa, la extensión marcada originalmente, quitandole dos dígitos de inicio. El 33 en iaxclient lo dejamos constante a manera de prefijo, pero no es enviado al iaxprovider. El 44 en iaxprovider de igual forma.

Una vez aclarado esto, procedemos a crear nuestro par de llaves. Una pública y una privada. Para ello necesitamos el programa openssl. En la computadora iaxclient ejecutamos:

openssl genrsa -out /var/lib/asterisk/keys/iaxclientkey.key 1024
openssl rsa -in /var/lib/asterisk/keys/iaxclientkey.key -pubout -out /var/lib/asterisk/keys/iaxclientkey.pub

En la computadora iaxprovider ejecutamos:

openssl genrsa -out /var/lib/asterisk/keys/iaxproviderkey.key 1024
openssl rsa -in /var/lib/asterisk/keys/iaxproviderkey.key -pubout -out /var/lib/asterisk/keys/iaxproviderkey.pub

Finalmente transferimos una copia de las llaves públicas entre los hosts. En iaxprovider:

scp /var/lib/asterisk/keys/iaxproviderkey.pub root@iaxclient.host.com:/var/lib/asterisk/keys/

En la máquina iaxclient:

scp /var/lib/asterisk/keys/iaxclientkey.pub root@iaxprovider.host.com:/var/lib/asterisk/keys/

Finalmente en asterisk ejecutamos "restart when convenient", con esto Asterisk leerá las llaves al iniciar.

Para provar es necesario configurar un par de extensiones, una en cada host e intentar marcar, desde iaxclient "33XX" donde XX es el número de la extensión de dos dígitos en iaxprovider. Para llamar desde iaxprovider se marca "44XX".

5. Otrás Aplicaciones. (posible tiempo sobrante indeterminado)

AJAM (Asynchronus Javascript Asterisk Manager)
ARA (Asterisk Real Time Architecture)
Aplicaciones Misc. Meetme(), VoiceMail(), ChanSpy(), diversas opciones en Dial(), CDR (1 hora)
AGI y AMI