ANEXO 5: SQL

Índice general

1. SQL RDBMS con PostgreSQL **1**

Capítulo 1

SQL RDBMS con PostgreSQL

Contents

1.1. DKLAB1 3

1.1.1. Instalación de PostgreSQL 3

1.1.2. Kerberización 4

Declaración del mecanismo de autenticación GSSAPI en pg_hba.conf; mapeo del principal kerberos al rol de PostgreSQL . . 5

Creación del principal de servicio para PostgreSQL y exportación a su keytab 5

1.1.3. Recarga de la configuración por parte del servidor 7

1.1.4. Tests con psql 7

1.2. DKLAB2 8

1.2.1. Instalación postgresql 9

1.2.2. Kerberización 9

Declaración del mecanismo de autenticación GSSAPI en pg_hba.conf; mapeo del principal kerberos al rol de PostgreSQL . . 9

Creación del principal de servicio para PostgreSQL y exportación a su keytab 10

1.2.3. Recarga de la configuración por parte del servidor 11

1.2.4. Test 11

1.3. Sincronización Master-Master: Bucardo 12

1.3.1. Bucardo en DKLAB1 14

Instalación de Bucardo 14

Soporte auth GSS-API para Bucardo 14

Creación de rol y db bucardo (script bucardo.schema) 16

Configuración general 16

Configuración específica para la replicación de cada tabla . . . 19

1.3.2. Bucardo en DKLAB2 19

Instalación de Bucardo 19

Soporte auth GSS-API para Bucardo 19

Creación de rol y db bucardo (script bucardo.schema) 21

Configuración general 21

Configuración específica para la replicación de cada tabla . . . 24

El otro contenedor de metadatos era el SQL RDBMS PostgreSQL. En nuestro despliegue sólo el servidor de mensajería instantánea, jabberd2, hará uso de él. Específicamente, lo usará para almacenar un grupo marcadamente volátil de sus metadatos.

Como hiciéramos con slapd, postgres será configurado para usar KERBEROS y su sistema de tickets para el subservicio de autenticación.

1.1. DKLAB1

1.1.1. Instalación de PostgreSQL

Actualizamos la lista de paquetes disponibles y buscamos que efectivamente tenemos disponible una versión de la rama 9.X

```
printf '\ndeb
http://backports.debian.org/debian-backports squeeze-backports main\n' \
  >> /etc/apt/sources.list


apt-get update && \
apt-cache search postgres | grep 9.
```

Nos aparece la 9.1, que instalamos finalmente:

```
apt-get install postgresql-9.1
```

Por defecto escucha en localhost; nosotros necesitamos que escuche como mínimo en la interfaz de red para la VPN:

```
vim /etc/postgresql/9.1/main/postgresql.conf
```

```
...
# - Connection Settings -
####fx:
#http://www.postgresql.org/docs/9.1/interactive/runtime-config-connection.html
#'*' significa escuchar en todas las interfaces; para nosotros esta' bien
#    pero quiza's se quiera ser ma's restrictivo en un entorno en
#    produccio'n, por ejemplo: '127.0.0.1,10.192.168.1'
listen_addresses = '*'
####endfx
#listen_addresses = 'localhost'         # what IP address(es) to listen on
...
```

Puesto que vamos a seguir haciendo modificaciones, esperaremos a recargar el sistema más tarde.

1.1.2. Kerberización

PostgreSQL permite utilizar KERBEROS [1] entre otros muchos mecanismos para la autenticación de roles. Nótese que con ello impedimos que viajen contraseñas en claro, si bien la documentación advierte que en el caso de postgresql no por ello encripta el tráfico subsiguiente (a diferencia de otros servicios, postgresql no usa el subprotocolo SASL (SASL-GSSAPI en este caso) el cual prevee poder encriptar las capas superiores, sino que usa simplemente a la librería GSSAPI para KERBEROS para autenticar usando tickets KRB5). Por ello, PostgreSQL sugiere adicionalmente usar TLS; en nuestro caso, como en PostgreSQL no guardaremos material sensible como contraseñas, no lo usaremos sin embargo. Así procedimos con slapd.

- Anotación: específicamente, postgresql en debian usará automáticamente TLS, pero con un certificado ajeno a nuestra CA, generado por el paquete ssl-cert (/var/lib/dpkg/info/ssl-cert.postinst) y compartido por otros procesos en el sistema. Así es más conveniente decir que el tráfico irá ofuscado, pues en teoría no puede dejar de considerarse desencriptable por terceras partes[2].

La configuración de kerberos conlleva dos fases que vemos a continuación:

[1] http://www.postgresql.org/docs/9.0/static/auth-methods.html#GSSAPI-AUTH
[2] http://www.philzimmermann.com/EN/essays/SnakeOil.html

Declaración del mecanismo de autenticación GSSAPI en pg_hba.conf; mapeo del principal kerberos al rol de PostgreSQL

```
cat <<EOF >> /etc/postgresql/9.1/main/pg_hba.conf
####fx:
host    all             all             0.0.0.0/0               gss \
map=krbprinc-pgrole include_realm=1
####endfx
EOF
```

Además de permitir la autentificación con GSSAPI para KERBEROS, hace falta declarar los mapeos de cada kerberos principal a su PostgreSQL rol ("rol" es el concepto de usuario en terminología de PostgreSQL). En el pg_hba.conf hemos anunciado que se permiten los mapeos etiquetados como "krbprinc-pgrole" de /etc/postgresql/9.1/main/pg_ident.conf. La sintaxis allí será algo como[3]:

```
krbprinc-pgrole <principal>@CASAFX.DYNDNS.ORG <usuario en posgtgresql>
```

Por ejemplo, para mapear el principal "root/admin" con el rol "postgres" (éste creado automáticamente durante la instalación), de forma que un usuario que presente un ticket de servicio de ese principal se le permita autenticarse para ese rol:

```
vim /etc/postgresql/9.1/main/pg_ident.conf
```

```
...
####fx:
krbprinc-pgrole root/admin@CASAFX.DYNDNS.ORG postgres
####endfx
```

Aún nos queda que el proceso postgres pueda autenticarse con el AS Kerberos, para lo cual debemos crear su principal y exportarlo convenientemente:

Creación del principal de servicio para PostgreSQL y exportación a su keytab

Según puede leerse postgresql.conf y su línea "#krb_srvname = postgres", el principal para PostgreSQL ha de tener la forma postgres/<FQDN>@<REALM>. Entonces:

[3] http://www.postgresql.org/docs/9.0/static/auth-username-maps.html

```
kadmin.local
 >addprinc -policy service -randkey postgres/dklab1.casafx.dyndns.org
 >ktadd -k /etc/keytab.d/postgresql.keytab \
        -norandkey postgres/dklab1.casafx.dyndns.org
 >quit

klist -ke /etc/keytab.d/postgresql.keytab

chown postgres:root /etc/keytab.d/postgresql.keytab
chmod go-rw /etc/keytab.d/postgresql.keytab
su postgres -m -c 'klist -t -K -k /etc/keytab.d/postgresql.keytab'
```

El propio postgresql.conf permite que se le indique la localización del keytab:

```
vim /etc/postgresql/9.1/main/postgresql.conf
```

```
...
# CONNECTIONS AND AUTHENTICATION
...
# Kerberos and GSSAPI
####fx:
krb_server_keyfile = '/etc/keytab.d/postgresql.keytab'
#krb_srvname = 'postgres'
#krb_server_hostname = 'dklab1.casafx.dyndns.org'
#krb_realm = 'CASAFX.DYNDNS.ORG'
#
#El tipo de auth en si' se configuro' en el pg_hba.conf; la u'ltima columna
#admiti'a 'gss <opts>', donde <opts> podi'a ser:
# map=<map-name>   # mapeos entre el principal y el pg rol (en pg_ident.conf)
# include_realm 1  # incluye el realm, lo tendremos en cuenta en el mapeo.
#http://www.postgresql.org/docs/9.1/static/client-authentication.html
####endfx
...
```

1.1.3. Recarga de la configuración por parte del servidor

```
invoke-rc.d postgresql restart
```

1.1.4. Tests con psql

El comando psql implementa un cliente para postgresql. Vamos allá:
Adquirimos las credenciales Kerberos de root/admin:

```
kdestroy
kinit -p root/admin
```

Intentamos un listado (-l) de las bases de datos en dklab1 (-h) ahora que tenemos el TGT:

```
klist

psql -h dklab1.casafx.dyndns.org -l
```

... no funciona, pues estamos pidiendo a psql que se autentique con el principal root/admin pero, al no especificar también un rol PostgreSQL, lo intenta con el nombre de usuario posix ("root" en este caso), rol que ni existe ni, si existiera, tiene un mapeo consentido desde el principal root/admin. Hay por tanto que explicitar el rol con "-U postgres":

```
psql -h dklab1.casafx.dyndns.org -U postgres -l

psql -h dklab1.casafx.dyndns.org -U postgres
 > \?
 > \l
 > \q


klist # veremos el ticket de servicio para postgresql en dklab1
```

Se puede añadir el argumento –dbname <coninfo>para explicitar con gran detalle a qué base de datos y cómo conectarnos, puede consultarse http://www.postgresql.org/docs/9.0/static/connect.html. Nota: si adicionalmente pasamos -W a psql, efectivamente nos dará un prompt para poner un password, pero no lo usaría.

Podemos anular el mapeo anterior tras comprobar que el sistema funciona:

```
sed -i 's/krbprinc-pgrole root
\/admin@CASAFX.DYNDNS.ORG postgres/#krbprinc-pgrole root
\/admin@CASAFX.DYNDNS.ORG postgres/g' \
    /etc/postgresql/9.1/main/pg_ident.conf
```

1.2. DKLAB2

El proceso es idéntico al de en dklab1.

1.2.1. Instalación postgresql

```
apt-get install postgresql-9.1


vim /etc/postgresql/9.1/main/postgresql.conf
```

```
...
# - Connection Settings -
####fx:
#http://www.postgresql.org/docs/9.1/interactive/runtime-config-connection.html
#'*' significa escuchar en todas las interfaces; para nosotros esta' bien
#    pero quiza's se quiera ser ma's restrictivo,
#    por ejemplo: '127.0.0.1,10.192.168.1'
listen_addresses = '*'
####endfx
...
```

1.2.2. Kerberización

Declaración del mecanismo de autenticación GSSAPI en pg_hba.conf; mapeo del principal kerberos al rol de PostgreSQL

```
cat <<EOF >> /etc/postgresql/9.1/main/pg_hba.conf
####fx:
host    all             all             0.0.0.0/0               gss \
map=krbprinc-pgrole include_realm=1
####endfx
EOF
```

Y el mapeo:

```
vim /etc/postgresql/9.1/main/pg_ident.conf
```

```
...
####fx:
krbprinc-pgrole root/admin@CASAFX.DYNDNS.ORG postgres
####endfx
```

Creación del principal de servicio para PostgreSQL y exportación a su keytab

Su principal ha de tener la forma postgres/<FQDN>@<REALM>:

```
kadmin.local
 >addprinc -policy service -randkey postgres/dklab2.casafx.dyndns.org
 >ktadd -k /etc/keytab.d/postgresql.keytab \
        -norandkey postgres/dklab2.casafx.dyndns.org
 >quit
klist -ke /etc/keytab.d/postgresql.keytab

chown postgres:root /etc/keytab.d/postgresql.keytab
chmod go-rw /etc/keytab.d/postgresql.keytab
su postgres -m -c 'klist -t -K -k /etc/keytab.d/postgresql.keytab'
```

Localización del keytab:

```
vim /etc/postgresql/9.1/main/postgresql.conf
```

```
...
# CONNECTIONS AND AUTHENTICATION
...
# Kerberos and GSSAPI
####fx:
krb_server_keyfile = '/etc/keytab.d/postgresql.keytab'
#krb_srvname = 'postgres'
#krb_server_hostname = 'dklab1.casafx.dyndns.org'
#krb_realm = 'CASAFX.DYNDNS.ORG'
#
#El tipo de auth en si' se configuro' en el pg_hba.conf; la u'ltima columna
#admiti'qa 'gss <opts>', donde <opts> podi'a ser:
# map=<map-name>   # mapeos entre el principal y el pg rol (en pg_ident.conf)
# include_realm 1  # incluye el realm, lo tendremos en cuenta en el mapeo.
#http://www.postgresql.org/docs/9.1/static/client-authentication.html
####endfx
```

1.2.3. Recarga de la configuración por parte del servidor

```
invoke-rc.d postgresql restart
```

1.2.4. Test

```
kdestroy
kinit -p root/admin
```

```
psql -h dklab2.casafx.dyndns.org -l -U postgres
 > \?
 > \l
 > \q


klist #veremos el ticket de servicio para postgresql en dklab2
```

Se puede añadir el argumento –dbname <coninfo>para explicitar con gran detalle a qué base de datos y cómo conectarnos, puede consultarse http://www.postgresql.org/docs/9.0/static/connect.html.

Y, por supuesto, podemos conectarnos desde dklab2 a dklab1:

```
psql -h dklab1.casafx.dyndns.org -l -U postgres


klist # veremos el ticket de servicio para postgresql en dklab1
      # adema's de en dklab2
```

Podemos anular el mapeo anterior tras comprobar que el sistema funciona:

```
sed -i 's/krbprinc-pgrole root
\/admin@CASAFX.DYNDNS.ORG postgres/#krbprinc-pgrole root
\/admin@CASAFX.DYNDNS.ORG postgres/g' \
    /etc/postgresql/9.1/main/pg_ident.conf
```

1.3. Sincronización Master-Master: Bucardo

Para los metadatos en LDAP tenemos la replicación Master-Master de slapd (ya desplegada). Para los que son gestionados por postgresql, éste aún no provee por sí solo un esquema de replicación similar al que tenemos en slapd. Tenemos entonces que instalar un software adicional, Bucardo, que se encargue de la replicación.

En general, nunca ha sido fácil desplegar servicios relacionados con la alta disponibilidad con postgresql. Existen numerosas opciones parciales implementadas en proyectos externos.

En nuestro caso particular, podemos delimitar nuestro problema a lo siguiente: queremos disfrutar de sincronización mutua entre los RDBMS, dklab1 y dklab2, para las tablas

de jabberd2. Existe una solución que se adapta a estos requisitos, sin abarcar mucho más y por tanto sin aportar más complejidad de la necesaria, ésto es Bucardo.

Bucardo ofrece replicación master-master pero asíncrona: la transacción SQL puede terminar antes de que la replicación se haya completado (en concreto, Bucardo usa triggers para lanzar la replicación). Al contrario que LDAP, las bases de datos SQL estaban orientadas a escrituras frecuentes, por tanto los defectos de sincronización temporales inherentes a Bucardo podrían teóricamente ocasionar problemas de pérdidas de datos, sin embargo:

- Jabberd2 (su proceso jabberd2-sm) no permite conectarse a varios pgsql (no existen registros SRV, tampoco se le pueden declarar varios en la configuración) como se investigó. Ésto implica que, por ejemplo, jabberd2-sm en dklab1 se conecta a postgresql en dklab1 y no a dklab2. A la vez:
- Aunque posible, no es frecuente compartir una misma cuenta jabber por varias personas de forma que se usase concurrentemente.
- Los datos fundamentales del servicio (roster, vcard) dependen de slapd, no de postgresql.
- Por otro lado, Bucardo necesita que las tablas tengan una PRIMARY KEY y sea no coincidente en ambos servidores (ejem: una par y otra impar), pero en nuestro caso no será difícil modificar el esquema (una docena de tablas) para cumplir con ésto.
- Las líneas principales de desarrollo de postgresql en este momento (9.0, 9.1, 9.2...) se centran precisamente en las capacidades de replicación.

Estos factores hacen que Bucardo nos parezca suficiente al menos mientras el desarrolo de la rama 9.X avanza hasta implementar su propia solución de replicación (hecho en 9.0) síncrona (hecho en 9.1) multimaster (por hacer aún, probablemente a costa de la incorporación del proyecto Postgres-XC: "Postgres-XC will soon be merged with PostgreSQL 9.X" según su desarrollador Michael Paquier[4]).

Existen alternativas a Bucardo [5], pero o parecen desactualizadas y sin soporte en Debian, u ofrecen más de lo que necesitamos a la vez que complican el despliegue (pgpool2[6]). Con una implementación autóctona a las puertas, preferimos algo simple que desplegar ahora, y hacer la inversión de aprendizaje definitiva después. En cualquier caso, la única alternativa real open source en el caso de que el desempeño de Bucardo no fuese el esperado y la solución autóctona se retrasase, diríase que es pgpool2.

Sólo puede haber un bucardo_ctl replicando en la red: o en dklab1 o en dklab2. Sin embargo dejaremos instalado Bucardo en ambos, guardando así la posibilidad de que ante hipotéticas labores de mantenimiento para Bucardo en dklab1, podamos manualmente

[4] http://michael.otacoo.com/postgresql-2/postgres-9-1-setup-a-synchronous-stand-by-server-in-5-minutes/

[5] http://wiki.postgresql.org/wiki/Replication,_Clustering,_and_Connection_Pooling

[6] http://pgpool.projects.postgresql.org/contrib_docs/simple_sr_setting2/index.html

sustituirlo temporalmente por Bucardo en dklab2 modificando /etc/default/bucardo e incluso /etc/bucardorc.

1.3.1. Bucardo en DKLAB1

Instalación de Bucardo

Versiones anteriores a Debian 7 "Wheezy" no funcionaban correctamente, pero ésto está ya solucionado y sólo tenemos que instalar:

```
apt-get install bucardo
```

Se ha creado una nueva cuenta local:

```
id bucardo
```

Soporte auth GSS-API para Bucardo

Bucardo necesita un principal en kerberos y las credenciales exportadas, más el mapeo de identidad en pg_ident.conf. Con ello, se facilitará que el TGT esté siempre y automáticamente disponible gracias a k5start e init.

```
kadmin.local
 >addprinc -policy service -randkey bucardo/dklab1.casafx.dyndns.org
 >ktadd -k /etc/keytab.d/bucardo.keytab bucardo/dklab1.casafx.dyndns.org
 >addprinc -policy service -randkey bucardo/dklab2.casafx.dyndns.org
 >ktadd -k /etc/keytab.d/bucardo.keytab2 bucardo/dklab2.casafx.dyndns.org
 >quit


chown bucardo:root /etc/keytab.d/bucardo.keytab
chmod go-rw /etc/keytab.d/bucardo.keytab
su bucardo -m -c 'klist -t -K -k /etc/keytab.d/bucardo.keytab'
```

Necesitamos que el principal para Bucardo pueda mapearse a otros roles en postgesql, no sólo Bucardo; efectivamente no hay ninguna restricción a ésto: "There is no restriction regarding how many database users a given operating system user [o su TGT] can

correspond to"[7].

```
vim /etc/postgresql/9.1/main/pg_ident.conf
```

```
...
####fx:
krbprinc-pgrole bucardo/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG bucardo
krbprinc-pgrole bucardo/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG jabberd2
krbprinc-pgrole bucardo/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG bucardo
krbprinc-pgrole bucardo/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG jabberd2
####endfx
```

```
vim /etc/inittab
```

(La línea KSB:... aparece dividida en 2 partes delimitadas por \ y cambio de línea. Realmente debe aparecer en una sóla línea, sin \ y cambio de línea:)

```
...
####fx:
KSB:2345:respawn:/usr/bin/k5start -U -f /etc/keytab.d/bucardo.keytab \
 -K 10 -l 24h -k /tmp/krb5cc_‘id -u bucardo‘ -o bucardo
####endfx
```

```
kill -HUP 1
sleep 5
ls -l /tmp/krb5cc_‘id -u bucardo‘


invoke-rc.d postgresql reload
```

[7] auth-username-maps

Creación de rol y db bucardo (script bucardo.schema)

Rol "bucardo" y db "bucardo" en postgresql:

```
su postgres -c 'psql --file /usr/share/bucardo/bucardo.schema'
```

...usamos "su postgres -c" y luego psql sin –host para que utilice autenticación ident tal como permite el pg_hba.conf.

Configuración general

```
vim /etc/bucardorc
```

```
dbport = 5432
####fx:
#-dbhost = localhost
dbhost = dklab1.casafx.dyndns.org
####endfx
dbname = bucardo
dbuser = bucardo
####fx:
#-dbpass = bucardo
####endfx
```

El paquete no trae un script para logrotate, a pesar de que no manda todos sus mensajes por syslog. Para solventar ésto podemos usar las opciones debugsyslog=1 y debugfile=0 en el /etc/bucardorc anterior, o bien crear el suyo[8]:

```
vim /etc/logrotate.d/bucardo
```

[8] http://web.archiveorange.com/archive/v/aQD0QzWpuylIzMVQN2Tk
http://web.archiveorange.com/archive/v/aQD0QoY2lQLhCnCTOKgz

```
/var/log/bucardo/* {
      daily
      missingok
      notifempty
      rotate 7
      create 644 bucardo bucardo
      compress
      copytruncate
      delaycompress
}
```

```
vim /etc/default/bucardo
```

```
####fx:
#-ENABLED=0
ENABLED=1
####endfx
```

Alternativamente a activarlo en /etc/default/bucardo, podemos no hacerlo allí y hacerlo de forma supervisada en inittab:

```
vim /etc/inittab
```

```
####fx:
#BUC:2345:respawn:/usr/local/bin/bucardo_overseer_wrap.sh
####endfx
```

... pero hasta que no estén configuradas las sincronizaciones, dejamos comentada la entrada BUC. Respecto al script en sí de supervisión, debemos diseñarlo y dejarlo en la localización señalada:

```
vim /usr/local/bin/bucardo_overseer_wrap.sh
```

```
#!/bin/bash
####fx:
#No reinventamos la rueda y tomamos la configuracio'n de los
#ficheros correspondientes:
for i in `grep '\w=' /etc/init.d/bucardo|grep -v DESC`; do eval $i; done

if ! test -d /var/run/bucardo; then
 mkdir /var/run/bucardo
 chown bucardo:bucardo /var/run/bucardo
fi
STAMP=`mktemp --tmpdir=' ' --dry-run --quiet | sed s@\ /tmp.@@g`
su bucardo --command "$DAEMON start --extraname=${STAMP}"
if ! test -f "$PIDFILE"; then
 exit 2
fi

BPID=`pgrep -f Master.*${STAMP}`
if test -z "${BPID}"
 then exit 3
fi

# El bucle while no deberi'a acceder al disco duro, asi' que uso
# /proc y los built-in test y read de bash:
while test -d /proc/$BPID; do
 read -t 3 # espera 3 segundos. -t so'lo esta' disponible en bash, no dash
done

sleep 1
exit 1
####endfx
```

```
chmod u+x /usr/local/bin/bucardo_overseer_wrap.sh
kill -HUP 1 # no tendra' efecto pues la entrada BUC esta' comentada au'n.
```

Finalmente:

```
su bucardo -c 'bucardo_ctl show all'
```

...usamos "su bucardo -c" para que el proceso bucardo_ctl pueda acceder al TGT (/tmp/krb5cc_‘id -u bucardo‘) del usuario local bucardo obtenido por k5start e init; ya con éste, usará la db "bucardo" como el rol "bucardo" (tal como dicta /etc/bucardorc) lo cual será permitido en virtud del TGT y el mapeo en pg_ident.conf.

Configuración específica para la replicación de cada tabla

Básicamente, le decimos a bucardo_ctl qué tablas, dónde y cómo se han de replicar. Éste entonces almacena esa información en su db bucardo, se conecta a las db que replicar y comienza a actuar. Sólo puede haber un bucardo_ctl replicando en la red, o en dklab1 o en dklab2 (en principio dklab1 tomará el mando, y dklab2 estará preparado de reserva).

Dejamos esta parte para cuando se instale y configure jabberd2.

1.3.2. Bucardo en DKLAB2

En dklab2, lo instalamos y configuramos pero no activamos el daemon, pues será el de dklab1 el que tome el control.

Instalación de Bucardo

```
apt-get install bucardo
```

Dependencias:
Se ha creado una nueva cuenta local:

```
id bucardo
```

Soporte auth GSS-API para Bucardo

Bucardo necesitaba un principal en kerberos y las credenciales exportadas, más el mapeo de identidad en pg_ident.conf. Con ello, se facilitará también que el TGT esté

siempre y automáticamente disponible gracias a k5start e inittab:

```
scp dklab1.casafx.dyndns.org:/etc/keytab.d/bucardo.keytab2 /etc/keytab.d/bucardo.k
ssh dklab1.casafx.dyndns.org "/bin/rm -f /etc/keytab.d/bucardo.keytab2"

chown bucardo:root /etc/keytab.d/bucardo.keytab
chmod go-rw /etc/keytab.d/bucardo.keytab
su bucardo -m -c 'klist -t -K -k /etc/keytab.d/bucardo.keytab'
```

Necesitamos que el principal para Bucardo pueda mapearse a otros roles en postgresql, no sólo bucardo; efectivamente no había ninguna restricción a ésto y por tanto:

```
vim /etc/postgresql/9.1/main/pg_ident.conf
```

```
...
####fx:
krbprinc-pgrole bucardo/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG bucardo
krbprinc-pgrole bucardo/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG jabberd2
krbprinc-pgrole bucardo/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG bucardo
krbprinc-pgrole bucardo/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG jabberd2
####endfx
```

```
vim /etc/inittab
```

(La línea KSB:... aparece dividida en 2 partes delimitadas por \ y cambio de línea. Realmente debe aparecer en una sóla línea, sin \ y cambio de línea:)

```
...
####fx:
KSB:2345:respawn:/usr/bin/k5start -U -f /etc/keytab.d/bucardo.keytab \
 -K 10 -l 24h -k /tmp/krb5cc_`id -u bucardo` -o bucardo
####endfx
```

```
kill -HUP 1
sleep 5
ls -l /tmp/krb5cc_‘id -u bucardo‘
```

Reiniciamos PostgreSQL:

```
invoke-rc.d postgresql reload
```

Creación de rol y db bucardo (script bucardo.schema)

Rol "bucardo" y db "bucardo" en postgresql:

```
su postgres -c ’psql --file /usr/share/bucardo/bucardo.schema’
```

Configuración general

```
vim /etc/bucardorc
```

```
dbport = 5432
####fx:
#-dbhost = localhost
dbhost = dklab1.casafx.dyndns.org
####endfx
dbname = bucardo
dbuser = bucardo
####fx:
#-dbpass = bucardo
####endfx
```

El paquete no trae un script para logrotate, a pesar de que no manda todos sus mensajes por syslog. Para solventar ésto podemos usar las opciones debugsyslog=1 y

debugfile=0 en el /etc/bucardorc anterior, o bien crear el suyo[9].

```
vim /etc/logrotate.d/bucardo
```

```
/var/log/bucardo/* {
      daily
      missingok
      notifempty
      rotate 7
      create 644 bucardo bucardo
      compress
      copytruncate
      delaycompress
}
```

```
view /etc/default/bucardo
```

```
ENABLED=0
```

En dklab1 especificamos además la forma de ejecutar Bucardo de forma supervisada por init.

```
vim /etc/inittab
```

```
####fx:
#BUC:2345:respawn:/usr/local/bin/bucardo_overseer_wrap.sh
####endfx
```

```
vim /usr/local/bin/bucardo_overseer_wrap.sh
```

[9] http://web.archiveorange.com/archive/v/aQD0QzWpuylIzMVQN2Tk
http://web.archiveorange.com/archive/v/aQD0QoY2lQLhCnCTOKgz

```
#!/bin/bash
####fx:
# No reinventamos la rueda y tomamos la configuracio'n de los
# ficheros correspondientes:
for i in `grep '\w=' /etc/init.d/bucardo|grep -v DESC`; do eval $i; done

if ! test -d /var/run/bucardo; then
 mkdir /var/run/bucardo
 chown bucardo:bucardo /var/run/bucardo
fi
STAMP=`mktemp --tmpdir=' ' --dry-run --quiet | sed s@\ /tmp.@@g`
su bucardo --command "$DAEMON start --extraname=${STAMP}"
if ! test -f "$PIDFILE"; then
 exit 2
fi

BPID=`pgrep -f Master.*${STAMP}`
if test -z "${BPID}"
 then exit 3
fi

# El while no deberi'a acceder al disco duro, asi' que uso /proc
# y los built-in test y read de bash:
while test -d /proc/$BPID; do
 read -t 3 # espera 3 segundos. -t so'lo esta' disponible en bash, no dash.
done

sleep 1
exit 1
####endfx
```

Finalmente:

```
su bucardo -c 'bucardo_ctl show all'
```

Según /etc/bucardorc usará la db "bucardo" como el rol "bucardo", así que necesitamos usar "su bucardo -c" para que las autentificaciones se produzcan satisfactoriamente y como se explicó.

Configuración específica para la replicación de cada tabla

Básicamente, le decimos a bucardo_ctl qué tablas, dónde y cómo se han de replicar. Éste entonces almacena esa información en su db bucardo, se conecta a las db que replicar y comienza a actuar. Sólo puede haber un bucardo_ctl replicando en la red, y en dklab2 sólo lo activaremos ante hipotéticas labores de mantenimiento de Bucardo en dklab1. LyX