ANEXO 8: XMPP

Índice general

Capítulo 1

Servicio de Mensajería Instantánea (y señalización VoIP) XMPP

Contents

1.1. Arquitectura de Jabberd2

Jabberd2 es un conjunto de procesos que cooperan entre sí:

Proceso	Función	¿varios?
jabberd2-c2s	A él se conectan los clientes. Los autentica.	Sí
jabberd2-sm	Maneja los sevicios del dominio (recabar roster's...)	Sí
jabberd2-s2s	Gestiona las conexiones a servidores de otros dominios	Depende
jabberd2-router	Relaciona todos los procesos entre sí	No

- Anotaciones:
  - c2s viene de "cliente a servidor". s2s de "servidor a servidor". sm de "session manager".
  - El valor "Depende" para s2s quiere decir que puedes tener a varios si cada uno se encarga de una ruta *distinta*.
  - Roster es el nombre de la lista de contactos de un usuario.

1.1.1. Otros componentes/pasarelas

El protocolo permite la aparición de pasarelas a otras redes de mensajería como MSN o IRC, así como la integración de componentes que provean comunicaciones multicuenta o el desarrollo de servicios basados en XMPP. Aunque estos componentes mejoran notablemente la experiencia del usuario, están fuera del alcance de este despliegue y no se instalarán. Baste decir que es sencillo incorporarlos en el siguiente sentido: la extensión XEP-0114[1] del protocolo hace que se descubran e incorporen automáticamente ante el servidor dado un puerto (7009 en Jabberd2).

Además son compatibles[2] con la infraestructura de cluster de Jabberd2 (puede haber varios en la red sirviendo un mismo subservicio). Su configuración suele ser un fichero xml similar a los de jabberd2; es aconsejable entender cómo funciona /etc/init.d/jabberd2 y que llama a los scripts de /etc/jabberd2/component.d, para así poder integrar estos componentes externos en la instalación de Debian. Los usuarios, por su parte, tienen en sus clientes a través de algún botón o menú, la funcionalidad "discovery" para listar estos componentes adicionales proveídos en su red y efectuar registros.

- PyMSNt[3] (apt-get install pymsnt), pasarela a MSN. Status: ok.
- PyIRCt[4], pasarela a IRC. Status: ok. Otra implementación: Jajcus[5].
- J2J[6], pasarela a jabber (permite tener los contactos de otra cuenta jabber en la actual, por ejemplo los contactos de facebook o gmail). Status: irregular.

[1] http://xmpp.org/extensions/xep-0114.html
[2] http://tomasz.sterna.tv/2009/06/transport-clustering-in-jabberd2/
[3] http://delx.net.au/projects/pymsnt/
[4] http://xmpppy.sourceforge.net/irc/
[5] https://github.com/Jajcus/jjigw/
[6] http://wiki.jrudevels.org/index.php/Eng:J2J

- Spectrum[7] es un proyecto nuevo que pretende suplir algunos de los anteriores y proveer otros (ej. pasarelas a plataformas de microblogging como twitter). Status: desconocido.
- MUconference[8] (apt-get install jabber-muc) permite crear salas multiusuario. Status: ok (si bien la versión anterior, 0.7, no hacía necesario confirmar la sala tras crearla). Es posible que este componente no se pueda clusterizar con Jabberd2. Otra implementación: Palaver[9].
- Idavoll[10] (funcionalidad pubsub[11] para desarrollar otros servicios encima).

1.2. DKLAB1

1.2.1. Instalación

Obtención de las fuentes y aplicación de parches

La versión más reciente empaquetada para Debian no es un paquete oficial, puede verse la historia completa en http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=547767. En lo que sigue detallaremos el proceso hasta tener un paquete instalable, comenzando por descargar las fuentes de este paquete no oficial:

```
apt-get install devscripts
apt-get install --no-install-recommends build-essential devscripts
mkdir /usr/src/jabberd2
cd /usr/src/jabberd2
dget \
http://www.darkskies.za.net/~norman/jabber/deb-src/jabberd2_2.2.9-0nr2.dsc
dpkg-source -x jabberd2_2.2.9-0nr2.dsc
cd jabberd2-2.2.9/
```

Parche sobre storage/storage_ldapvcard.c Tuvimos que crear este parche para que funcionase en nuestro despliegue:

[7] http://spectrum.im
[8] http://gna.org/projects/mu-conference/
[9] http://www.onlinegamegroup.com/projects/palaver
[10] http://idavoll.ik.nu/
[11] http://www.isode.com/whitepapers/xmpp-pubsub.html

- la creación al vuelo, buscando en LDAP, de tarjetas Vcard con información del usuario.
- la creación al vuelo, buscando en LDAP, de la lista de contactos (también llamada "roster").

```
cd storage

cp -a storage_ldapvcard.c storage_ldapvcard.c.orig
```

A continuación se presenta el parche en formato base64 (sin duda la forma más idónea para preservar su integridad si es copiado desde este documento):

```
vim storage_ldapvcard.c.patch.b64
```

```
LS0tIHN0b3JhZ2VfbGRhcHZjYXJkLmMJMjAwOS0wNy0wNSAyMzo1NDoxNi4wMDAwMDAwMDAgKzAy
MDAKKysrIHN0b3JhZ2VfbGRhcHZjYXJkLmMubW9kCTIwMTItMDItMDYgMDM6NDQ6NDEuMDAwMDAw
MDAwICswMTAwCkBAIC0yNDIsNyArMjQyLDE2IEBACiAgICAgICAgICAgICBhdHRyc192Y2FyZFtp
KytdID0gbGUubGRhcGVudHJ5OwogICAgICAgICB9IHdoaWxlICggbGUubGRhcGVudHJ5ICE9IE5V
TEwgKTsKIAotICAgICAgICBzbnByaW50ZihsZGFwZmlsdGVyLCAxMDI0LCAiKCYob2JqZWN0Q2xh
c3M9JXMpKCVzPSVzKSkiLCBkYXRhLT5vYmplY3RjbGFzcywgZGF0YS0+dWlkYXR0ciwgb3duZXIp
OworICAgICAgICAvLy8vZng6IHZjYXJkIGZpbHRlcgorICAgICAgICAvLyAgICAgIFF1aXRvIGxh
IHBhcnRlIEA8cmVhbGO+IHBhcmEgaGFjZXIgbGEgYnUnc3F1ZWRhLgorICAgICAgICAvL3NOcmR1
cCwgc3Ryc2VwIGVzdGEnbiBkZWZpbmlkYXMgZW4gPHNOcmluZy5oPgorICAgICAgICBjaGFyICp1
aWR0b2tlbiA9IHNOcmR1cCggb3duZXIgKTsKKyAgICAgICAgdW1kdG9rZW4gPSBzdHJzZXAoICZ1
aWR0b2tlbiwgIkAiKTsKKyAgICAgICAgbG9nX2RlYnVnKFpTkUsICJGWDogbGRhcHZjYXJkIHVp
ZD0lcyIsIHVpZHRva2VuKTsKKyAgICAgICAgLy8tc25wcmludGYobGRhcGZpbHRlciwgMTAyNCwg
IigmKG9iamVjdENsYXNzPSVzKSglcz0lcykpIiwgZGF0YS0+b2JqZWN0Y2xhc3MsIGRhdGEtPnVp
ZGF0dHIsIG93bmVyKTsKKyAgICAgICAgc25wcmludGYobGRhcGZpbHRlciwgMTAyNCwgIigmKG9i
amVjdENsYXNzPSVzKSglcz0lcykpIiwgZGF0YS0+b2JqZWN0Y2xhc3MsIGRhdGEtPnVpZGF0dHIs
IHVpZHRva2VuKTsKKyAgICAgICAgZnJlZSh1aWR0b2tlbik7CisgICAgICAgIC8vLy9lbmRmeAog
ICAgICAgICAgICBsb2dfZGVidWcoWk9ORSwgInNlYXJjaCBmaWx0ZXI6ICVzIiwgbGRhcGZpbHRlcik7
CiByZXRyeV92Y2FyZDoKICAgICAgICAgICAgaWYobGRhcF9zZWFyY2hfcyhyYXRhLT5sZCwgZGF0YS0+
YmFzZWRuLCBBMREFQX1NDT1BFX1NVQlRSRUUsIGxkYXBmaWx0ZXIsIGF0dHJzX3ZjYXJkLCAwLCAm
cmVzdWx0KSkQEAgLTMxMSw3ICszMjAsMTUgQEAKICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
dmFsaWRhdHRyICkgewogICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
MjU2LCAiKCYoJXM9KikoISglcz0wKSkpKCVzPTEpIiwgZGF0YS0+cHVibGlzaGVkYXR0ciwgZGF0
YS0+cHVibGlzaGVkYXR0ciwgZGF0YS0+dmFsaWRhdHRyKTsKICAgICAgICAgICAgICAgICAgICAg
c2UgewotICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAg
```

```
JXM9KikoISglcz0wKSkpIiwgZGF0YS0+cHVibGlzaGVkYXR0ciwgZGF0YS0+cHVibGlzaGVkYXR0
cik7CisgICAgICAgICAgICAgICAgICAgIC8vLy9meDogcm9zdGVyLXB1Ymxpc2ggcGFyY2lhbCAi
dmFsaWRmaWx0ZXIiCisgICAgICAgICAgICAgICAgICAgIC8vICAgICAgUXVpdG8gZXN0YSBwYXJ0
ZSBkZWwgZmlsdHJvIHB1ZXMgc2UgZXhwYW5kZSwKKyAgICAgICAgICAgICAgICAgICAgLy8gICAg
ICBubyBzZScgcG9yIHF1ZScsIGEgZXN0YSBleHByZXNpbydu
IGludmEnbGlkYToKKyAgICAgICAgICAgICAgICAgICAgLy8gICAgICAoJig/amFiYmVyUHVibGlzaGVkSXRlbT0qKSg/PWVycm9yKSkK
KyAgICAgICAgICAgICAgICAgICAgLy8gICAgICBhZGVtYSdzLCBlbiBtaSBkZXNwbGllZ3VlIHRv
ZG9zIGxvcyB1c3VhcmlvcyBzb24gCisgICAgICAgICAgICAgICAgICAgIC8vICAgICAgcHVibGlj
YWJsZXMgZW4gZWwgcm9zdGVyLCBubyBuZWNlc2l0byBlc3RlIGZpbHRybworICAgICAgICAgICAg
ICAgICAgICAvLyAgICAgIHkgcG9yIHRhbnRvIG5vIGludGVudG8gY29ycmVnaXJsby4KKyAgICAg
ICAgICAgICAgICAgICAgLy8tc25wcmludGYodmFsaWRmaWx0ZXIsIDI1NiwgIigmKCVzPSopKCEo
JXM9MCkpKSIsIGRhdGEtPnB1Ymxpc2hlZGF0dHIsIGRhdGEtPnB1Ymxpc2hlZGF0dHIpOworICAg
ICAgICAgICAgICAgICAgICAvLy8vZW5kZngKICAgICAgICAgICAgICAgICAgICAgICB9CiAgICAgICAg
ICB9CiAgICAgICAgICAgICAgICAgICAgCkBAIC0zNDIsMTggKzM1OSwzNCBAQAogICAgICAgICAg
ICAgIGxkYXBfbXNnZnJlZShyZXN1bHQpOwogICAgICAgICAgICAgICAgICAgIHJldHVybiBzdF9GQUlM
RUQ7CiAgICAgICAgICAgICAgICB9CiOKKyAgICAgICAgICAgICAgIC8vLy9meDoKKyAgICAgICAgICAgIGxv
Z19kZWJ1ZyhaT05FLCAiRW5jdWVudHJvIGVudHJhZGFzIGVuIGxkYXAgcGFyYSBlbCByb3N0ZXIs
IGNyZW8gZWwgb2JqZWN0IHNldCIpOworICAgICAgICAgICAgICAgLy8vL2VuZGZ4CiAgICAgICAgICAg
ICAqb3MgPSBvc19uZXcoKTsKIAogICAgICAgICAgICAgICAgICAgZG8geworCiAgICAgICAgICAgICAgICAg
dmFscyA9IChjaGFyICoqKWxkYXBfZ2V0X3ZhbHVlcyhkYXRhLT5sZCxlbnRyeSxkYXRhLT5ncm91
cGF0dHIpOwogICAgICAgICAgICAgICAgICAgIGlmKCBsZGFwX2NvdW50X3ZhbHVlcyh2YWxzKSA8PSAw
ICkgewogICAgICAgICAgICAgICAgICAgICBsZGFwX3ZhbHVlX2ZyZWUodmFscyk7CiAgICAgICAg
ICAgICAgICAgIGNvbnRpbnVlOwogICAgICAgICAgICAgICAgICAgIH0KKyAgICAgICAgICAgICAg
ICAvLy8vZng6CisgICAgICAgICAgICAgICAgICAgICAvLyBUcmVzIGRhdG9zIHF1ZSBjb25zZWd1aXIg
```

cGFyYSBlbCByb3N0ZXI6IGdyb3VwLCBqaWQsIG5hbWUKKyAgICAgICAgICAgICAgICAgIC8vCisg
ICAgICAgICAgICAgICAgICAvLyAxLiBHcm91cDoKKyAgICAgICAgICAgICAgICAgIC8vIChlbCBx
dWUgbWEncyBhYmFqbyB1c2UgdmFsc1swXSBpbXBsaWNhIHF1ZSB1c2Egc28nbG8gZWwKKyAgICAg
ICAgICAgICAgICAgIC8vIHByaW1lciBvYmpldG8sIGx1ZWdvIGRhIGlndWFsIHNpIG11bHRpdmFs
dWFkbykKKyAgICAgICAgICAgICAgICAvLy8vZW5kZngKICAgICAgICAgICAgICAgICBzdHJuY3B5
KGdyb3VwLHZhbHNbMF0sc2l6ZW9mKGdyb3VwKS0xKTsgZ3JvdXBbc2l6ZW9mKGdyb3VwKS0xXT0n
XDAnOwogICAgICAgICAgICAgICAgIGxkYXBfdmFsdWVfZnJlZSh2YWxzKTsKLQorICAgICAgICAg
ICAgICC8vLy9meDoKKyAgICAgICAgICAgICAgIC8vIDIuIEppZDoKKyAgICAgICAgICAgICAg
ICAgICC8vIENvZ2UgZWwgamlkIGRlc2RlIHVpZGF0dHIsIHBlcm8gYWhpJyBsbyB0ZW5nbyBz
aW4gCisgICAgICAgICAgICAgICAgICAvLyBlbCBAPHJlYWxtPiEgQ3VhbmRvIHJlY2FiZSBlbCB0
cmknbyBncm91cCxqaWQsbmFtZSB5IHZheWEKKyAgICAgICAgICAgICAgICAgIC8vIGEgYWxtYWNl
bmFybG8gZW4gZWwgcm9zdGVyLCB0ZW5kcmUnIG9wb3J0dW5pZGFkIGRlIGxpZGlhcgorICAgICAg
ICAgICAgICAgICAgLy8gY29uIGUnc3RvLgorICAgICAgICAgICAgICAgICAgIC8vLy9lbmRmeAogICAg
ICAgICAgICAgICAgIHZhbHMgPSAoY2hhciAqKilsZGFwX2dldF92YWx1ZXMoZGF0YS0+bGQsZW50
cnksZGF0YS0+dWlkYXR0cik7CiAgICAgICAgICAgICAgICAgaWYoIGxkYXBfY291bnRfdmFsdWVz
KHZhbHMpID09IDAgKSB7CiAgICAgICAgICAgICAgICAgICAgIGxkYXBfdmFsdWVfZnJlZSh2YWxz
KTsKQEAgLTM2MiwxMSArMzk1LDIwIEBACiAgICAgICAgICAgICAgICAgfQogICAgICAgICAgICAg
ICAgIHN0cm5jcHkoamlkLHZhbHNbMF0sc2l6ZW9mKGppZCktMSk7IGppZFtzaXplb2YoamlkKS0x
XT0nXDAnOwogICAgICAgICAgICAgICAgIGxkYXBfdmFsdWVfZnJlZSh2YWxzKTsKLQotICAgICAg
ICAgICAgICAgIHZhbHMgPSAoY2hhciAqKilsZGFwX2dldF92YWx1ZXMoZGF0YS0+bGQsZW50cnks
InNuIik7CisgICAgICAgICAgICAgICAgLy8vL2Z4OiAKKyAgICAgICAgICAgICAgICAgIC8vIDMu
IE5hbWU6CisgICAgICAgICAgICAgICAgICAvLyBQcmVmaWVybyBxdWUgaW50ZW50ZSB1c2FyICJk
aXNwbGF5TmFtZSIgeSSwgc2kgbm8sICJzbiIKKyAgICAgICAgICAgICAgICAgIC8vIGx1ZWdvIGhl
IGRlIGNhbWJpYXIgZWwgb3JkZW4gZWRpdGFuZG8gYXF1aScgeSAzIGxpJ25lYXMKKyAgICAgICAg

ICAgICAgICAgIC8vIG1hJ3MgYWJham86CisgICAgICAgICAgICAgICAgICAvLy12YWxzID0gKGNo
YXIgKiopbGRhcF9nZXRfdmFsdWVzKGRhdGEtPmxkLGVudHJ5LCJzbiIpOworICAgICAgICAgICAg
ICAgIHZhbHMgPSAoY2hhciAqKilsZGFwX2dldF92YWx1ZXMoZGF0YS0+bGQsZW50cnksImRpc3Bs
YXlOYW1lIik7CisgICAgICAgICAgICAgICAgLy8vL2VuZGZ4CiAgICAgICAgICAgICAgICAgaWYo
IGxkYXBfY291bnRfdmFsdWVzKHZhbHMpIDw9IDAgKSB7CiAgICAgICAgICAgICAgICAgICAgIGxk
YXBfdmFsdWVfZnJlZSh2YWxzKTsKLSAgICAgICAgICAgICAgICAgICAgdmFscyA9IChjaGFyICoq
KWxkYXBfZ2V0X3ZhbHVlcyhkYXRhLT5sZCxlbnRyeSwiZGlzcGxheU5hbWUiKTsKKyAgICAgICAg
ICAgICAgICAgLy8vL2Z4OgorICAgICAgICAgICAgICAgICAgICAvLy12YWxzID0gKGNoYXIg
KiopbGRhcF9nZXRfdmFsdWVzKGRhdGEtPmxkLGVudHJ5LCJkaXNwbGF5TmFtZSIpOworICAgICAg
ICAgICAgICAgICB2YWxzID0gKGNoYXIgKiopbGRhcF9nZXRfdmFsdWVzKGRhdGEtPmxkLGVu
dHJ5LCJzbiIpOworICAgICAgICAgICAgICAgICAgICAvLy8vZW5kZngKICAgICAgICAgICAg
ICAgICAgaWYoIGxkYXBfY291bnRfdmFsdWVzKHZhbHMpIDw9IDAgKSB7CiAgICAgICAgICAg
ICAgICAgICBzdHJuY3B5KG5hbWUsamlkLHNpemVvZihuYW1lKS0xKTsgbmFtZVtzaXplb2Yo
bmFtZSktMV09J1wwJzsKICAgICAgICAgICAgICAgICAgfSBlbHNlIHsKQEAgLTM4NCw3ICs0
MjYsMzYgQEAKICAgICAgICAgICAgICBsZGFwX3ZhbHVlX2ZyZWUodmFscyk7CiAKICAgICAg
ICAgICAgICBvID0gb3Nfb2JqZWN0X25ldygqb3MpOworICAgICAgICAgICAgIC8vLy9m
eDoKKyAgICAgICAgICAgICAgICAvLyBQYXJhIGFufmFkaXIgQDxyZWFsbT4gYSBsb3MgdXN1YXJp
b3MgZGVsIHJvc3RlciwKKyAgICAgICAgICAgICAgICAvLyB0ZW5nbyBxdWUgcmVzb2x2ZXIgcHJp
bWVybyBxdWUgb3duZXIgZXMgIiIgY3VhbmRvCisgICAgICAgICAgICAgICAgLy8gdHlwZSBlcyAi
cHVibGlzaGVkLXJvc3RlciIuCisgICAgICAgICAgICAgICAgLy8gTGEgZnVuY2lvJ24gX3Jvc3Rl
cl9wdWJsaXNoX3VzZXJfbG9hZCgpIGVuCisgICAgICAgICAgICAgICAgLy8gc20vbW9kX3Jvc3Rl
cl9wdWJsaXNoLmMgbGxhbWEgY29uIG93bmVyICIiIGEKKyAgICAgICAgICAgICAgICAvLyBzdG9y
YWdlX2dldCgpIHNtL3N0b3JhZ2UuYyBxdWUgbm9zIGxsYW1hIGEgbm9zb3Ryb3MsCisgICAgICAg
ICAgICAgICAgLy8gcG9yIHRhbnRvLCBjb24gb3duZXIgIiI7IHNpIGFsbGkgIGVuIG1vZF9yb3N0

```
ZXJfcHVibGlzaC5jCisgICAgICAgICAgICAgICAgLy8gY2FtYmlvICIiIHBvciBlbCBKSUQgY29u
IGppZF91c2VyKHVzZXItPmppZCksIGFxdWknIHB1ZWRvCisgICAgICAgICAgICAgICAgLy8gdXNh
ciBvd25lciBwYXJhIGNvbnNlZ3VpciBlbCBAPHJlYWxtPiB5IGNvbXBsZXRhciBlbCBjYW1wbwor
ICAgICAgICAgICAgIC8vIGppZCBkZWwgcm9zdGVyLgorICAgICAgICAgICAgIGxvZ19k
ZWJ1ZyhaT05FLCAiRlg6IG93bmVyPSVzIixvd25lcik7CisgICAgICAgICAgICAgICAgY2hhciAq
dG9rZW4gPSBzdHJkdXAoIG93bmVyICk7CisgICAgICAgICAgICAgICAgbG9nX2RlYnVnKFpPTkUs
ICJGWDogdG9rZW49JXMiLHRva2VuKTsKKyAgICAgICAgICAgICAgICBzdHJzZXAoICZ0b2tlbiwg
IkAiKTsKKyAgICAgICAgICAgICAgICBsb2dfZGVidWcoWk9ORSwgIkZYOiB0b2tlbl9wb3N0X2Zp
cnN0X3N0cnNlcD0lcyIsdG9rZW4pOworICAgICAgICAgICAgICAgIGNoYXIgKnJlYWxtdG9rZW4g
PSBzdHJzZXAoICZ0b2tlbiwgIkAiKTsKKyAgICAgICAgICAgICAgICBsb2dfZGVidWcoWk9ORSwg
IkZYOiByZWFsbXRva2VuPSVzIixyZWFsbXRva2VuKTsKKyAgICAgICAgICAgICAgICBzdHJuY2F0
KGppZCwiQCIsc2l6ZW9mKGppZCktMSk7IGppZFtzaXplb2YoamlkKS0xXT0nXDAnOworICAgICAg
ICAgICAgIGxvZ19kZWJ1ZyhaT05FLCAiRlg6IGppQD0lcyIsamlkKTsKKyAgICAgICAgICAgICAg
ICAgICBzdHJuY2F0KGppZCxyZWFsbXRva2VuLHNpemVvZihqaWQpLTEpOyBqaWRbc2l6ZW9mKGpp
ZCktMV09J1wwJzsKKyAgICAgICAgICAgICAgICBsb2dfZGVidWcoWk9ORSwgIkZYOiBqQGQ9JXMi
LGppZCk7CisKICAgICAgICAgICAgICAgICBvc19vYmplY3RfcHV0KG8sImppZCIsamlkLG9zX3R5
cGVfU1RSSU5HKTsKKworICAgICAgICAgICAgICAgIGZyZWUodG9rZW4pOworICAgICAgICAgICAg
ICAgIGxvZ19kZWJ1ZyhaT05FLCAiRlg6IHBvc3QgZnJlZShOb2tlbikiKTsKKyAgICAgICAgICAg
ICAgICAvLworICAgICAgICAgICAgICAgIC8vLW9zX29iamVjdF9wdXQobywiamlkIixqaWQsb3Nf
dHlwZV9TVFJJTkcpOworICAgICAgICAgICAgICAgIC8vLy9lbmRmeAogICAgICAgICAgICAgICAg
IG9zX29iamVjdF9wdXQobywiZ3JvdXAiLGdyb3VwLG9zX3R5cGVfU1RSSU5HKTsKICAgICAgICAg
ICAgICAgICBvc19vYmplY3RfcHV0KG8sIm5hbWUiLG5hbWUsb3NfdHlwZV9TVFJJTkcpOwogICAg
ICAgICAgICAgICAgIGl2YWw9MTsKQEAgLTQwMCw2ICs0NzEsNyBAQAogICAgICAgICAgICAgICAg
ICAgICBvc19mcmVlKGRhdGEtPmNhY2hlKTsKICAgICAgICAgICAgICAgICB9CiAgICAgICAgICAg
ICAgICAgZGF0YS0+Y2FjaGUgPSBvc19uZXcoKTsKKyAgICAgICAgICAgICAgICAvLy8vRlg6ICAK
ICAgICAgICAgICAgICAgICBvc19jb3B5KCpvcywgZGF0YS0+Y2FjaGUpOwogICAgICAgICAgICAg
ICAgIGRhdGEtPmNhY2hlX3RpbWUgPSB0aW1lKE5VTEwpOwogICAgICAgICAgICAgICAgfQo=
```


Podemos descomprimirlo así:

```
base64 -d storage_ldapvcard.c.patch.b64 > storage_ldapvcard.c.patch
```

Si lo inspeccionamos, debiera resultar algo como ésto:

```
view storage_ldapvcard.c.patch
```

```
--- storage_ldapvcard.c 2009-07-05 23:54:16.000000000 +0200
+++ storage_ldapvcard.c.mod     2012-02-06 03:44:41.000000000 +0100
@@ -242,7 +242,16 @@
             attrs_vcard[i++] = le.ldapentry;
         } while ( le.ldapentry != NULL );

-        snprintf(ldapfilter, 1024, "(&(objectClass=%s)(%s=%s))", data->objectclass, data->uidattr, owner);
+        ////fx: vcard filter
+        //      Quito la parte @<realm> para hacer la bu'squeda.
+        //strdup, strsep esta'n definidas en <string.h>
+        char *uidtoken = strdup( owner );
+        uidtoken = strsep( &uidtoken, "@");
+        log_debug(ZONE, "FX: ldapvcard uid=%s", uidtoken);
+        //-snprintf(ldapfilter, 1024, "(&(objectClass=%s)(%s=%s))", data->objectclass, data->uidattr, owner);
+        snprintf(ldapfilter, 1024, "(&(objectClass=%s)(%s=%s))", data->objectclass, data->uidattr, uidtoken);
+        free(uidtoken);
+        ////endfx
         log_debug(ZONE, "search filter: %s", ldapfilter);
 retry_vcard:
         if(ldap_search_s(data->ld, data->basedn, LDAP_SCOPE_SUBTREE, ldapfilter, attrs_vcard, 0, &result))
@@ -311,7 +320,15 @@
                 if( data->validattr ) {
                     snprintf(validfilter, 256, "(&(%s=*)(!(%s=0)))(%s=1)", data->publishedattr, data->publishedattr, data->validattr);
                 } else {
-                    snprintf(validfilter, 256, "(&(%s=*)(!(%s=0)))", data->publishedattr, data->publishedattr);
+                    ////fx: roster-publish parcial "validfilter"
+                    //      Quito esta parte del filtro pues se expande,
+                    //      no se' por que', a esta expresio'n inva'lida:
+                    //      (&(?jabberPublishedItem=*)(?=error))
+                    //      adema's, en mi despliegue todos los usuarios son
+                    //      publicables en el roster, no necesito este filtro
+                    //      y por tanto no intento corregirlo.
+                    //-snprintf(validfilter, 256, "(&(%s=*)(!(%s=0)))", data->publishedattr, data->publishedattr);
+                    ////endfx
                 }
             }
```

```
@@ -342,18 +359,34 @@
                 ldap_msgfree(result);
                 return st_FAILED;
             }
-
+            ////fx:
+            log_debug(ZONE, "Encuentro entradas en ldap para el roster, creo el object set");
+            ////endfx
             *os = os_new();

             do {
+
                 vals = (char **)ldap_get_values(data->ld,entry,data->groupattr);
                 if( ldap_count_values(vals) <= 0 ) {
                     ldap_value_free(vals);
                     continue;
                 }
+                ////fx:
+                  // Tres datos que conseguir para el roster: group, jid, name
+                  //
+                  // 1. Group:
+                  // (el que ma's abajo use vals[0] implica que usa so'lo el
+                  // primer objeto, luego da igual si multivaluado)
+                ////endfx
                 strncpy(group,vals[0],sizeof(group)-1); group[sizeof(group)-1]='\0';
                 ldap_value_free(vals);
-
+                ////fx:
+                  // 2. Jid:
+                  // Coge el jid desde uidattr, pero ahi' lo tengo sin
+                  // el @<realm>! Cuando recabe el tri'o group,jid,name y vaya
+                  // a almacenarlo en el roster, tendre' oportunidad de lidiar
+                  // con e'sto.
+                ////endfx
                 vals = (char **)ldap_get_values(data->ld,entry,data->uidattr);
                 if( ldap_count_values(vals) <= 0 ) {
                     ldap_value_free(vals);
@@ -362,11 +395,20 @@
                 }
                 strncpy(jid,vals[0],sizeof(jid)-1); jid[sizeof(jid)-1]='\0';
                 ldap_value_free(vals);
-
-                vals = (char **)ldap_get_values(data->ld,entry,"sn");
```

```
+                ////fx:
+                  // 3. Name:
+                  // Prefiero que intente usar "displayName" y, si no, "sn"
+                  // luego he de cambiar el orden editando aqui' y 3 li'neas
+                  // ma's abajo:
+                  //-vals = (char **)ldap_get_values(data->ld,entry,"sn");
+                vals = (char **)ldap_get_values(data->ld,entry,"displayName");
+                ////endfx
                 if( ldap_count_values(vals) <= 0 ) {
                     ldap_value_free(vals);
-                    vals = (char **)ldap_get_values(data->ld,entry,"displayName");
+                    ////fx:
+                    //-vals = (char **)ldap_get_values(data->ld,entry,"displayName");
+                    vals = (char **)ldap_get_values(data->ld,entry,"sn");
+                    ////endfx
                     if( ldap_count_values(vals) <= 0 ) {
                         strncpy(name,jid,sizeof(name)-1); name[sizeof(name)-1]='\0';
                     } else {
@@ -384,7 +426,36 @@
                 ldap_value_free(vals);

                 o = os_object_new(*os);
+                ////fx:
+                // Para an~adir @<realm> a los usuarios del roster,
+                // tengo que resolver primero que owner es "" cuando
+                // type es "published-roster".
+                // La funcio'n _roster_publish_user_load() en
+                // sm/mod_roster_publish.c llama con onwer "" a
+                // storage_get() sm/storage.c que nos llama a nosotros,
+                // por tanto, con owner ""; si alli' en mod_roster_publish.c
+                // cambio "" por el JID con jid_user(user->jid), aqui' puedo
+                // usar owner para conseguir el @<realm> y completar el campo
+                // jid del roster.
```

```
+               log_debug(ZONE, "FX: owner=%s",owner);
+               char *token = strdup( owner );
+               log_debug(ZONE, "FX: token=%s",token);
+               strsep( &token, "@");
+               log_debug(ZONE, "FX: token_post_first_strsep=%s",token);
+               char *realmtoken = strsep( &token, "@");
+               log_debug(ZONE, "FX: realmtoken=%s",realmtoken);
+               strncat(jid,"@",sizeof(jid)-1); jid[sizeof(jid)-1]='\0';
+               log_debug(ZONE, "FX: ji@=%s",jid);
+               strncat(jid,realmtoken,sizeof(jid)-1); jid[sizeof(jid)-1]='\0';
+               log_debug(ZONE, "FX: j@d=%s",jid);
+
                os_object_put(o,"jid",jid,os_type_STRING);
+
+               free(token);
+               log_debug(ZONE, "FX: post free(token)");
+               //
+               //-os_object_put(o,"jid",jid,os_type_STRING);
+               ////endfx
                os_object_put(o,"group",group,os_type_STRING);
                os_object_put(o,"name",name,os_type_STRING);
                ival=1;
@@ -400,6 +471,7 @@
                    os_free(data->cache);
                }
                data->cache = os_new();
+               ////FX:
                os_copy(*os, data->cache);
                data->cache_time = time(NULL);
            }
```

El parche se hizo con:

```
diff --unified --recursive --new-file \
     storage_ldapvcard.c storage_ldapvcard.c.mod \
     > storage_ldapvcard.c.patch
```

luego, para aplicarlo:

```
patch -p0 < storage_ldapvcard.c.patch
cd ..
```

Parche sobre sm/mod_roster_publish.c Este parche sigue buscando la creación al vuelo del roster desde LDAP, pero afecta ahora al fichero mod_roster_publish.c:

```
cd sm
cp -a mod_roster_publish.c mod_roster_publish.c.orig
```

En base64, el parche es:

```
vim mod_roster_publish.c.patch.b64
```


```
LS0tIG1vZF9yb3N0ZXJfcHVibGlzaC5jCTIwMDktMDctMDUgMjM6NTQ6MTUuMDAwMDAwMDAwICsw
MjAwCisrKyBtb2Rfcm9zdGVyX3B1Ymxpc2guYy5tb2QJMjAxMi0wMi0wNiAwMTo1NDoxMi4wMDAw
MDAwMDAgKzAxMDAKQEAgLTIxOSw3ICsyMTksMTEgQEAKIAogICAgICAgICBsb2dfZGVidWcoWk9O
RSwgInB1Ymxpc2hpbmcgcm9zdGVyIGZvciAlcyIsamlkX3VzZXIodXNlci0+amlkKSk7CiAgICAg
ICAgIC8qIGdldCBwdWJsaXNoZWQgcm9zdGVyICovCi0gICAgICAgIGlmKCBzdG9yYWdlX2dldCh1
c2VyLT5zbS0+c3QsICJwdWJsaXNoZWQtcm9zdGVyIiwgIiIsIE5VTEwsICZvcykgPT0gc3RfU1VD
Q0VTUyApIHsKKyAgICAgICAgLy8vL2Z4OgorICAgICAgICBsb2dfZGVidWcoWk9ORSwgIkZYOiBq
aWQ9JXMiLCBqaWRfdXNlcih1c2VyLT5qaWQpICk7CisgICAgICAgIGlmKCBzdG9yYWdlX2dldCh1
c2VyLT5zbS0+c3QsICJwdWJsaXNoZWQtcm9zdGVyIiwgamlkX3VzZXIodXNlci0+amlkKSwgTlVM
TCwgJm9zKSA9PSBzdF9TVUNDRVNTICkgewor
ICAgICAgICAvLy1pZiggc3RvcmFnZV9nZXQodXNl
ci0+c20tPnN0LCAicHVibGlzaGVkLXJvc3RlciIsICIiLCBOVUxMLCAmb3MpID09IHN0X1NVQ0NF
U1MgKSB7CisgICAgICAgIC8vLy9lbmRmeAogICAgICAgICAgICAgaWYob3NfaXRlcl9maXJzdChv
cykpIHsKICAgICAgICAgICAgICAgICAgICAvKiBpdGVyYXRlIG9uIHB1Ymxpc2hlZCByb3N0ZXIgKi8K
ICAgICAgICAgICAgICAgICBqaWQgPSBOVUxMOwo=
```


Y podemos decodificarlo con:

```
base64 -d mod_roster_publish.c.patch.b64 > mod_roster_publish.c.patch
```

Si lo inspeccionamos, su aspecto debiera ser algo como:

```
view mod_roster_publish.c.patch
```

```
--- mod_roster_publish.c        2009-07-05 23:54:15.000000000 +0200
+++ mod_roster_publish.c.mod    2012-02-06 01:54:12.000000000 +0100
@@ -219,7 +219,11 @@

         log_debug(ZONE, "publishing roster for %s",jid_user(user->jid));
         /* get published roster */
-        if( storage_get(user->sm->st, "published-roster", "", NULL, &os) == st_SUCCESS ) {
+        ////fx:
+        log_debug(ZONE, "FX: jid=%s", jid_user(user->jid) );
+        if( storage_get(user->sm->st, "published-roster", jid_user(user->jid), NULL, &os) == st_SUCCESS ) {
+        //-if( storage_get(user->sm->st, "published-roster", "", NULL, &os) == st_SUCCESS ) {
+        ////endfx
             if(os_iter_first(os)) {
                 /* iterate on published roster */
                 jid = NULL;
```

El parche se hizo con:

```
diff --unified --recursive --new-file \
     mod_roster_publish.c mod_roster_publish.c.mod \
     > mod_roster_publish.c.patch
```

luego para aplicarlo debemos:

```
patch -p0 < mod_roster_publish.c.patch
cd ..
```

Construcción e instalación del paquete

Sufijaremos con un código de revisión el paquete, ya que no coincide con el original. Después intentaremos compilar:

```
dch -l fxpatched \
              'Parcheado para hacer funcionar ldapvcard y roster-publishing'


dpkg-buildpackage -D
```

```
...
dpkg-checkbuilddeps: Unmet build dependencies: debhelper (>= 7) ...
...
```

El comando anterior se queja de que el SO no tiene todas las dependencias en tiempo de compilación. Nos avisa de en qué paquetes se encuentra, luego procedemos a instalarlos:

```
apt-get install --no-install-recommends \
            debhelper dpatch autotools-dev automake libtool libssl-dev \
            libgsasl7-dev libdb-dev libpam0g-dev libmysqlclient15-dev  \
            libpq-dev libldap2-dev libsqlite3-dev libidn11-dev         \
            libexpat1-dev
```

La librería libudns-dev no está en los repositorios de Debian 7 "Wheezy", pero es descargable de la rama de desarrollo de Debian, Sid. Es compatible con la libc6 en nuestro sistema (su única dependencia) luego no es necesario recompilarla:

- http://packages.debian.org/sid/libudns-dev
- http://packages.debian.org/sid/libudns0

```
wget \
http://ftp.es.debian.org/debian/pool/main/u/udns/libudns0_<version>_i386.deb \
    http://ftp.es.debian.org/debian/pool/main/u/udns/libudns-dev_<version>_i386.de


dpkg -i libudns-dev_*_i386.deb libudns0_*_i386.deb


mv *deb ../
```

Por último, hay un error en el fichero (de tipo Makefile) debian/rules, que yerra con el nombre de la opción para dar soporte SASL al binario:

```
configure: WARNING: unrecognized options: --disable-rpath, --enable-sasl
```

Lo correjimos editando no interactivamente con sed:

```
sed -i 's/--enable-sasl=gsasl/--with-sasl=gsasl/g' debian/rules
```

Ya podemos compilar. A este respecto, es extraordinariamente aconsejable compilar Jabberd2 con soporte de depuración. Este Makefile debian/rules lo hará si encuentra la variable

```
DEB_BUILD_OPTIONS="debug"
```

El resultado de tener soporte para depuración, será que los programas de Jabberd2 admitirán el flag -D y serán más verborreicos. Por otro lado, no queremos por parte del compilador (gcc) ningún tipo de optimización no predefinida, así que la variable CFLAGS estará vacía. Por tanto:

```
DEB_BUILD_OPTIONS=debug CFLAGS= CXXFLAGS= dpkg-buildpackage -D
```

```
...
        CFLAGS="$(CFLAGS)" ./configure --host=$(DEB_HOST_GNU_TYPE) \
                --build=$(DEB_BUILD_GNU_TYPE) \
                --prefix=/usr --sysconfdir=/etc/jabberd2 \
                --bindir=\$${prefix}/sbin \
                --program-prefix=jabberd2- --disable-rpath \
                --with-sasl=gsasl \
                --enable-ssl \
                --enable-mysql \
                --enable-pgsql \
                --enable-sqlite \
                --enable-db \
                --enable-ldap \
                --enable-pam \
                --enable-pipe \
                --enable-anon \
                --enable-fs \
                --debug -g2 -O2
...
```

Tras la compilación, instalamos el paquete y paramos los servicios que levante:

```
ls ../*deb


dpkg -i ../jabberd2_2.2.9-0nr2_i386.deb
invoke-rc.d jabberd2 stop


mkdir -p ~/deb
mv ../*deb ~/deb
cd ~
```

1.2.2. Soporte para centralización de metadatos: LDAP y SQL

Desechando la autenticación, proveída por KERBEROS, podemos usar PostgreSQL y OpenLDAP para almacenar el resto de metadatos de las cuentas. Pero, en el caso de LDAP, sólo tiene soporte para algunos metadatos, en concreto aquellos con los que se puede construir una vCARD[12] (el servicio XMPP incluye la posibilidad de que los usuarios se intercambien esta información[13]).

```
ls /usr/lib/jabberd2/ | egrep '(ldap|pgsql)' | grep --invert-match auth
```

Efectivamente Jabberd2 registra datos como, por ejemplo, si la cuenta se está usando ahora o no ("presencia"). Este metadato tan volátil no es aceptable para ser almacenado en OpenLDAP (un gestor de bases de datos pensado para lecturas) y sí para PostgreSQL. Otros metadatos como los representados en un archivo vCARD (nombre, apellidos, teléfono, correo, fotografía...) son más estables y candidatos a ser almacenados en LDAP. Así, el componente sm de Jabberd2 está diseñado para usar PostgreSQL, pero puede también construir al vuelo una tarjeta electrónica vCARD recabando los datos de la rama del árbol LDAP. Por ahora no se puede configurar el mapeo entre los atributos que tenemos en LDAP y los que busca Jabberd2, éstos están declarados en la estructura ldapvcard_entry_st en ${JABBERD2_SRC}/storage/storage_ldapvcard.c y coinciden con los de las clases estructurales person e inetOrgPerson de los esquemas de OpenLDAP (ésta última, la usada entre otras para crear la cuenta de ejemplo umea).

Existe un segundo uso de LDAP. El componente sm puede elaborar al vuelo un roster (esto es, la "lista de contactos") buscando en LDAP las cuentas definidas allí. Así,

[12] Tarjeta de visita, consúltese:
http://www.imc.org/pdi/
http://www.imc.org/pdi/vcard-21.txt
[13] http://xmpp.org/xmpp-protocols/xmpp-extensions/

si creamos una cuenta centralizada nueva, no hay necesidad de que el usuario mande subscripciones a todo el resto.

Como conclusión de todo lo dicho sobre Jabberd2 y LDAP, diremos que no tenemos que hacer nada adicional. No tenemos que importar un nuevo esquema a slapd para los metadatos que Jabberd2 vaya a buscar en ldap, sino dejar que reuse lo que ya tenemos.

En el caso de PostgreSQL, Jabberd2 provee un esquema para organizar sus metadatos en tablas y que por supuesto no tenemos cargado. Vamos entonces con PostgreSQL; registraremos su DB, crearemos el principal para que se use autenticación KERBEROS contra PostgreSQL y cargaremos su esquema SQL.

Creación en PostgreSQL de la base de datos jabberd2 y el rol jabberd2

```
grep ^[^#] /etc/postgresql/9.0/main/pg_hba.conf | grep postgres
```

Registramos la DB para Jabberd2, usando tabla unicode y nombre jabberd2. También creamos el rol PostgreSQL, de nombre jabberd2 también y sin contraseña (-P, pues usaremos KERBEROS). Podemos hacer su a la cuenta local postgres y autenticarnos (-U) como tal para llevar a cabo estas acciones:

```
# DB:
su postgres -c 'createdb -U postgres -E UNICODE jabberd2'
# Rol:
su postgres -c 'createuser -U postgres
                           --no-createrole --no-createdb --no-superuser
                           jabberd2'
```

Configuración autenticación KERBEROS para acceder a PostgreSQL

Mapeo KERBEROS principal ->PostgreSQL Rol El mapeo relaciona con el rol jabberd2 el principal del componente jabberd2-sm tanto de dklab1 como de dklab2 (el instance varía):

```
grep gss /etc/postgresql/9.0/main/pg_hba.conf


cat <<EOF >> /etc/postgresql/9.0/main/pg_ident.conf
####fx:
krbprinc-pgrole jabberd2-sm/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG jabberd2
krbprinc-pgrole jabberd2-sm/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG jabberd2
####endfx
EOF
```

Creación del principal para jabberd2-sm, exportación a keytab y configuración de k5start Tenemos aquí dos posibilidades: más tarde tendremos que kerberizar al componente jabberd2-c2s (client-to-server, al que se conectan los clientes para autenticarse, de forma que los clientes jabber puedan hacer SSO). Entonces, habrá que crear un principal de servicio que, según la documentación, deberá tener la forma xmpp/<FQDN>@<REALM>. Podemos usar ese principal para que ahora jabberd2-sm (session-manager, el componente que recaba los metadatos) se conecte como cliente al servicio postgresql, o podemos crearle el suyo propio. Para clarificar el proceso, hemos decidido crearle a jabberd2-sm el suyo propio. Lo que sí será común es que ambos se exportarán al mismo keytab, propiedad de jabber (usuario local con que se ejecutan todos los procesos de jabberd2).

```
kadmin.local
> addprinc -policy service -randkey jabberd2-sm/dklab1.casafx.dyndns.org
> ktadd -k /etc/keytab.d/jabberd2.keytab \
        -norandkey jabberd2-sm/dklab1.casafx.dyndns.org
> quit


chown jabber:root /etc/keytab.d/jabberd2.keytab
chmod go-rw /etc/keytab.d/jabberd2.keytab
su jabber -m -c 'klist -t -K -k /etc/keytab.d/jabberd2.keytab'


vim /etc/inittab
```

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

```
...
####fx:
KSJ:2345:respawn:/usr/bin/k5start -U -f /etc/keytab.d/jabberd2.keytab \
-K 10 -l 24h -k /tmp/krb5cc_‘id -u jabber‘ -o jabber
####endfx
```

```
kill -HUP 1
sleep 10
ls -l /tmp/krb5cc_‘id -u jabber‘
```

Test Conseguiremos un TGT de forma no interactiva a través del keytab. A continuación pediremos un listado de las DB's existentes (-l), indicando que el rol con el que queremos autenticarnos es jabberd2 (-U) y, gracias al TGT conseguido y los mapeos, la autenticación será un éxito y se nos devolverá el listado (consulta para la que estamos autorizados, debemos añadir, sin consultásemos las ACL del rol jabberd2).

```
kdestroy

# Conseguimos un TGT de forma no interactiva:
kinit -k -t /etc/keytab.d/jabberd2.keytab \
       jabberd2-sm/dklab1.casafx.dyndns.org

# Pedimos como rol jabberd2 un listado de las DB's existentes:
psql -h dklab1.casafx.dyndns.org -l -U jabberd2
klist
```

Importación del esquema SQL de Jabberd2; modificaciones para Bucardo

Jabberd2 viene con su propio esquema para PostgreSQL:

```
cd ~
mkdir sql
zcat /usr/share/doc/jabberd2/db-setup.pgsql.gz > sql/j2pgsql.sql
grep ^CREATE sql/j2pgsql.sql
```

Como vemos por el último grep, el esquema no es demasiado extenso. Ésto es importante, ya que debemos modificarlo para que nuestro mecanismo de replicación Master-Master pueda funcionar. Efectivametne, para Bucardo añadimos una Primary-Key en las tablas donde no la hay. Puesto que donde ocurre ésto, está definida una secuencia[14], uso esa secuencia como Primary-Key[15].

```
sed -i '142s/"xml" text NOT NULL/"xml" text NOT NULL, PRIMARY KEY
 ("object-sequence")/g' sql/j2pgsql.sql

sed -i 's/"node" text/"node" text, PRIMARY KEY
 ("object-sequence")/g' sql/j2pgsql.sql

sed -i 's/"block" integer/"block" integer, PRIMARY KEY
 ("object-sequence")/g' sql/j2pgsql.sql

sed -i 's/CREATE SEQUENCE "object-sequence";/CREATE SEQUENCE
 "object-sequence" START WITH 1 INCREMENT BY 2;/g' sql/j2pgsql.sql
```

Aprovechando el TGT conseguido durante el test anterior:

```
grep ^[^#] /etc/postgresql/9.0/main/pg_hba.conf
psql -h dklab1.casafx.dyndns.org -U jabberd2 jabberd2 -f sql/j2pgsql.sql
```

[14] Emulando a un campo serial además:
http://wiki.postgresql.org/wiki/FAQ/es#.C2.BFC.C3.B3mo_puedo_crear_un_campo_serial.2Fauto-creciente.3F

[15] Puede consultarse como informales referencias:
http://searchoracle.techtarget.com/answer/Should-sequence-numbers-be-used-as-primary-keys->
http://en.wikipedia.org/wiki/Surrogate_key->
http://en.wikipedia.org/wiki/Current_database.

1.2.3. Configuración de Jabberd2

Jabberd2 almacena su configuración en ficheros XML. Cada componente de Jabberd2 tiene el suyo. Sería conveniente mostrar un cuadro sinóptico que explique qué se configura en cada fichero.

Como se podrá observar, todos los componentes tienen un identificador (por ejemplo de la forma <componente>.jabberd2.<hostname>). El componente sm además registra los identificadores de dominio que gestiona (parte tras la @ del JID[16]) así como dónde se almacenan los metadatos. El componente c2s sin embargo es el que lidia con la autenticación (reino de autenticación, mecanismo), y tiene que saber qué componente (qué id) corresponde al componente sm de su reino.

Componente	Parámetro	Función
sm	id	Único, simplemente para identificar esta instancia.
	local id	La parte del JID, uno por cada host virtual (todos los puede manejar un sólo sm) si necesitas ya que cada vhost tenga una configuración !=, hay q lanzar varios procesos.
	storage type	Backend usado según /usr/lib/jabberd2/storage_*.
	auto-create	Si un usuario hace log-in satisfactoriamente, se le crea la cuenta XMPP automáticamente.
c2s	id	Único, simplemente para identificar esta instancia si vacío es jid y es default de entre los vhosts.
	auth type	Auth usada según /usr/lib/jabberd2/authreg_*) y mechanismos SASL[17] y tradicional[18].
	local id	Un FQDN que coincida con el local-id de su sm
	realm	Desde el punto de vista de la auth, un user pertenece a un REALM, no a su JID, de forma que varios JID pueden ser los mismos users (misma auth) si no dices nada es tu id. Si indicas "" y usas PAM, éste no mira el JID, sólo el nombre local, por cierto.
	user-accountreg	Si el usuario puede crear su cuenta vía XMPP
router	id	Único, simplemente para identificar esta instancia.
	clusters	Varios sm/transport para un sólo JID (el clustering component que controla quién atiende a qué cuenta no ha habido que implementarlo/integrarlo puesto que este componente ya lo hace[19]).
s2s	id	Único, simplemente para identificar esta instancia.

Antes de pasar a los ficheros de configuración en sí, tengamos en cuenta las siguientes reglas generales para editarlos:

- Son ficheros XML, luego hay que respetar las reglas del XML bien formado:
  - http://www.w3.org/TR/REC-xml/#dt-wellformed
- No admite tildes o caracteres no ASCII en los comentarios.

[16] http://xmpp.org/extensions/xep-0029.html

Certificados X.509 para comunicaciones entre componentes

Debemos encriptar las comunicaciones entre los componentes. Lo haremos utilizando TLS. Aprovecharemos la CA que hicimos cuando la necesitó OpenVPN.

"Router selects the destination instance using a hash of user JID (receiver in case of SM instances, sender in case of external components)".

```
cd /usr/local/lib/easy-rsa/2.0

bash

export KEY_EMAIL="jabbermaster@casafx.dyndns.org"
```

Declaramos read-only esta variable para no ser sobreescrita con el valor de cuando usamos OpenVPN:

```
declare -r KEY_EMAIL

source vars
```

Aprovechando la configuración que utilizamos cuando openvpn, queremos que los certificados de router, c2s y s2s lleven atributo subjectAltName (pues éstos actuarán como servicios a los que se conectarán clientes que pueden intentar verificar su nombre). Para ello utilizamos build-key-server (y no build-key) sabiendo que éste además incluirá el atributo "Netscape Cert Type: SSL Server" que Jabberd2 no tendrá en cuenta (como veremos, no tenemos constancia de que defina PMI alguna, excepto, y no es el caso, cuando los certificados se utilizan en el proceso de autentificación SASL EXTERNAL[20]. El subjectAltName que realmente comprobarán los clientes es, según se comprobó durante las pruebas, "DNS:casafx.dyndns.org" es decir el identificador de dominio.

[20] http://xmpp.org/extensions/xep-0178.html#c2s

```
export SAN="DNS:casafx.dyndns.org, DNS:dklab1.casafx.dyndns.org,
            DNS:jabber1.casafx.dyndns.org"
./build-key-server router-1.jabberd2.casafx.dyndns.org
#... pulsar RET para todas las preguntas; el passphrase
#    de la CA era "asdf"

# De la misma manera, para el resto de componentes:
./build-key sm-1.jabberd2.casafx.dyndns.org
./build-key-server c2s-1.jabberd2.casafx.dyndns.org
./build-key-server s2s-1.jabberd2.casafx.dyndns.org

# Para dklab2:
export SAN="DNS:casafx.dyndns.org, DNS:dklab2.casafx.dyndns.org,
            DNS:jabber2.casafx.dyndns.org"
./build-key-server router-2.jabberd2.casafx.dyndns.org
./build-key sm-2.jabberd2.casafx.dyndns.org
./build-key-server c2s-2.jabberd2.casafx.dyndns.org
./build-key-server s2s-2.jabberd2.casafx.dyndns.org
unset SAN
exit
```

```
ls keys/

cat keys/serial keys/index.txt

openssl req     -text -in keys/router-1.jabberd2.casafx.dyndns.org.csr \
        | ccze -A | less -R

openssl x509    -text -in keys/router-1.jabberd2.casafx.dyndns.org.crt \
        | ccze -A | less -R

openssl rsa     -text -in keys/router-1.jabberd2.casafx.dyndns.org.key \
        | ccze -A | less -R
```

El formato requerido por los componentes de jabberd2 es, según los comentarios de los ficheros de configuración:

"The certificates must be in PEM format and must be sorted starting with the subject's certificate (actual client or server certificate), followed by intermediate CA certificates if applicable, and ending at the highest level (root) CA".

No se describe, pues, una PMI en particular (atributos que use de ACL). Por otro lado, el formato PEM implica que sólo debe aparecer las líneas de BEGIN, END y los datos en base64 entre ambas[21]. Por tanto, creamos un directorio /etc/jabberd2/certs y almacenamos los certificados según todas las indicaciones anteriores:

[21] http://tools.ietf.org/html/rfc1421

```
ls -l /etc/ssl/certs/casafx-ca.crt


CERTDIR=/etc/jabberd2/certs
mkdir ${CERTDIR}
for i in router-1 sm-1 c2s-1 s2s-1 router-2 sm-2 c2s-2 s2s-2
 do LN=‘grep --line-number "BEGIN CERTIFICATE-----" \
         keys/${i}.jabberd2.casafx.dyndns.org.crt\
         | awk -F":" ’{print $1}’‘
     (sed -n "${LN},\$p" keys/${i}.jabberd2.casafx.dyndns.org.crt; \
      cat keys/ca.crt; \
      cat keys/${i}.jabberd2.casafx.dyndns.org.key) > ${CERTDIR}/${i}.pem
      chown root:jabber ${CERTDIR}/${i}.pem
      chmod 640         ${CERTDIR}/${i}.pem
done
ls -l ${CERTDIR}
```

Componente router

Se configura principalmente:

- ID de este componente: router-1.jabberd2.casafx.dyndns.org. El sufijo -1 alude a que es el componente router desplegado en dklab1. El ID es un simple identificador, no va a utilizarse para resoluciones DNS aunque lo hayamos basado en esa política de nombres.

El caso del programa router es un tanto especial: será el punto débil del sistema. Ante labores de mantenimiento del servidor que requieran parar el componente router, el servicio se vendrá abajo y, a lo sumo, podremos planificar su sustitución por el componente router en dklab2 haciendo que los demás componentes apunten a aquel router y reiniciándolos en el momento adecuado. Por otro lado, ante eventuales caídas no programadas, lo único que podemos hacer es que un proceso (como init) init supervise al componente router y lo vuelva a levantar rápidamente.

Tras la anotación anterior, vamos ya a editar su fichero de configuración:

```
chown jabber:root /etc/jabberd2/router*.xml
chmod o-r /etc/jabberd2/router*.xml


vim /etc/jabberd2/router.xml
```

```
...
<!-- ####fx:-->
     <!--
     #-<id>router</id>
     -->
     <id>router-1.jabberd2.casafx.dyndns.org</id>
<!-- ####endfx-->


<log>
...
</log>
```

```
<local>
 ...
     <pass>secret</pass>           <!-- default: secret -->
 ...
 <!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/router-1.pem</pemfile>
 <!-- ####endfx -->
</local>

<check>
 ...
<check>

<io>
...
</io>

<aliases>
...
</aliases>

<aci>
...
</aci>
```

Hay archivos adicionales, como el router-users.xml que controla qué otro componentes y con qué credencial puede conectarse al componente router. Nosotros usaremos siempre los valores por defecto establecidos allí (jabber:secret), si bien en un entorno en producción habría que cambiarlos allí y, acorde, en el resto de *.xml que vienen a continuación.

Componente sm

Aquí se configura principalmente:

- ID, nombre que identifica a este componente: sm-1.jabberd2.casafx.dyndns.org
- Local ID, la parte tras la @ en el nombre de las cuentas: casafx.dyndns.org
- Tipo de almacenamiento para los metadatos asociados a las cuentas: pgsql, ldap. No soporta el descubrimiento de los servidores LDAP usando registros SRV, desafortunadamente.
- Política de creación automática de cuenta si autenticación exitosa: activado.

```
chown jabber:root /etc/jabberd2/sm.xml

chmod o-r /etc/jabberd2/sm.xml


vim /etc/jabberd2/sm.xml
```

```
...
<!-- ####fx:-->
     <!--
     #-<id>sm</id>
     -->
     <id>sm-1.jabberd2.casafx.dyndns.org</id>
<!-- ####endfx-->

...

<router>
<!-- ####fx: NOTA: podri'amos tener la tentacio'n de crear varios bloques
     #router.../router, uno para dklab1 y otro para el de dklab2... no
     #funcionara', se usa sistema'ticamente so'lo el primero.
     #Igualmente pasara' en los bloques router del resto de componentes.
     #Actualmente, so'lo puede haber un componente router en la red.
     ####endfx
-->
    <ip>127.0.0.1</ip>             <!-- default: 127.0.0.1 -->
...
    <pass>secret</pass>            <!-- default: secret -->
...
<!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/sm-1.pem</pemfile>
<!-- ####endfx -->
    <retry>
```

```
<!-- ####fx: -->
     <!--
     #-<init>3</init>
     -->
     <init>-1</init>
<!-- ####endfx -->
     ...
<!-- ####fx: -->
     <!--
     #-<lost>3</lost>
     -->
     <lost>-1</lost>
<!-- ####endfx -->
    </retry>
...
</router>

<log>
...
</log>

<local>
 <!-- ####fx: identificador de dominio (la parte tras la arroba) -->
      <!--
      #-<id>localhost.localdomain</id>
      -->
      <id>casafx.dyndns.org</id>
 <!-- ####endfx -->
</local>
```

```
<storage>
...
   <!-- ####fx: -->
        <!--
        #-<driver>sqlite</driver>
        -->
        <driver>pgsql</driver>
   <!-- ####endfx -->

   <!-- ####fx: -->
        <driver type='vcard'>ldapvcard</driver>
        <driver type='published-roster'>ldapvcard</driver>
   <!-- ####endfx -->
   <!--
   <driver type='published-roster-groups'>ldapvcard</driver>
   -->
...
  <pgsql>
      <!-- ####fx:
      http://www.postgresql.org/docs/9.0/static/libpq-connect.html -->
          <!--
          #Tal como veni'a, usari'a socket unix y auth ident, pues no
          #indica host:
    #-<conninfo>dbname=jabberd2 user=jabberd2 password=secret</conninfo>
```

```
        #Si pongo varios host (o en gnl repito cualquier variable),
        #se usa so'lo el u'ltimo valor; si pongo varios <conninfo>,
        #se usa so'lo el primero.
        #No puede, pues, indicarse que elija de entre varios hosts,
        #desde aqui'.
        -->
        <conninfo>
        dbname=jabberd2 user=jabberd2 host=dklab1.casafx.dyndns.org
        </conninfo>
    <!-- ####endfx -->
...
</pgsql>

<ldapvcard>
    <!-- ####fx: -->
        <!--
        #Pretendi'a que usase los registros SRV, pero da error:
        #[error] ldapvcard: no uri specified in config file
        #-<uri>ldap://localhost/ ldaps://ldap.example.com/</uri>
        -->
        <uri>
         ldap://dklab1.casafx.dyndns.org
         ldap://dklab2.casafx.dyndns.org
        </uri>
    <!-- ####endfx -->
    ...
    <uidattr>uid</uidattr>
    <objectclass>posixAccount</objectclass>
```

```
    <!-- ####fx: -->
         <!--
         # No se debiera contrastar password alguna pues no lo hay al
         # usar gssapi
         #-<pwattr>userPassword</pwattr>
         -->
    <!-- ####endfx -->

    <!-- ####fx: -->
         <!--
         #-<basedn>o=Example Corp.</basedn>
         -->
          <basedn realm='casafx.dyndns.org'>
           ou=users,ou=accounts,dc=casafx,dc=dyndns,dc=org
          </basedn>
         <groupattr>departmentNumber</groupattr>
    <!-- ####endfx -->
    ...
    <publishedcachettl>60</publishedcachettl>
    <mapped-groups>
    ...
    </mapped-groups>
  </ldapvcard>
</storage>
```

```
<aci>
  <acl type='all'>
    <!-- ####fx: -->
         <!--
         #-<jid>admin@localhost.localdomain</jid>
         -->
         <jid>root/admin@casafx.dyndns.org</jid>
    <!-- ####endfx -->
  </acl>
...
</aci>

<modules>
...
</modules>

<discovery>
...
  <!-- ####fx: -->
       <serverinfo>
        <admin-addresses>
         <value>mailto:jabbermaster@casafx.dyndns.org</value>
        </admin-addresses>
       </serverinfo>
  <!-- ####endfx -->
</discovery>
```

```
<user>
   <!-- ####fx: -->
        <auto-create/>
   <!-- ####endfx -->
 ...

 <template>
    <publish>
    <!-- Uncomment <publish> if you wish to forcely publish
         roster template from LDAP on each user login.
         Si no defines <publish> para roster en ldap, puedes definir
         <roster> para roster.xml local.
     -->
     <!-- ####fx: -->
     <!-- #Si no defines <publish> para roster en LDAP, puedes definir
          #<roster> para roster.xml local.
       -->
          <publish realm='casafx.dyndns.org'>
     <!-- ####endfx -->
    ...
     <!-- ####fx: -->
          <!--
          # El nombre (que no el JID) de los contactos puede
          # ser cambiado por el usuario (y se almacenara' en
          # PostgreSQL), e'sto puede deshabilitarse
          # con <override-names>.
          -->
     <!-- ####endfx -->
```

```
        <!--
        <override-names/>
        -->

        <!-- ####fx: -->
             <force-groups>
              <prefix>departmentNumber</prefix>
              <!-- <suffix>(MyOrg)</suffix> -->
             </force-groups>
        <!-- ####endfx -->
    ...
     <!-- ####fx: -->
          </publish>
     <!-- ####endfx -->


     <!-- ####fx: -->
     <!-- # Si en el futuro queremos una cuenta especial no en LDAP,
          # como serviciotecnico@casafx.dyndns.org, que deba ser an~adida
          #a todo usuario automa'ticamente, podemos declararlo en un
          #fichero y activar la siguiente opcion.
          #(Este roster.xml existe y contiene algu'n ejemplo de uso).
     <!-- ####endfx -->
     <!--
     <roster>/etc/jabberd2/templates/roster.xml</roster>
     -->
 </template>
</user>
```

```
<amp>
...
</amp>

<offline>
...
</offline>

<roster>
...
</roster>

<status>
...
</status>
```

Componente c2s

Aquí se configura principalmente:

- ID, nombre que identifica a este componente: c2s-1.jabberd2.casafx.dyndns.org.
- Tipo de autenticación: SASL-GSSAPI.
- Local ID, la parte tras la @ en el nombre de las cuentas. Debe coincidir con lo que se especificó en <local>en sm.xml: casafx.dyndns.org.
- Tipo de autentificación: normalmente se elije uno de los módulos listados en /usr/lib/jabberd2. Pero no hay ninguno para SASL o directamente GSSAPI, ésto tiene como consecuencia que, en lo que a SASL-GSSAPI se refiere, no importa el módulo pongas, lo importante es que especifiques que SASL, compilado dinámicamente en jabberd2-c2s, anuncie el mecanismo GSSAPI a los clientes.

```
ls /usr/lib/jabberd2|grep '(sasl|gssapi)'

ldd /usr/sbin/jabberd2-c2s|grep gssapi
```

- Política de registro de cuenta por el usuario a través de su cliente: prohibido.
- Certificado para las conexiones con el cliente, por las deficiencias de GSASL.

Vamos a ello:

```
chown jabber:root /etc/jabberd2/c2s.xml
chmod o-r /etc/jabberd2/c2s.xml


vim /etc/jabberd2/c2s.xml
```

```
...
<!-- ####fx:-->
     <!--
     #-<id>c2s</id>
     -->
     <id>c2s-1.jabberd2.casafx.dyndns.org</id>
<!-- ####endfx-->
...
<router>
    <ip>127.0.0.1</ip>             <!-- default: 127.0.0.1 -->
...
    <pass>secret</pass>            <!-- default: secret -->
...
<!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/c2s-1.pem</pemfile>
<!-- ####endfx -->

    <retry>
<!-- ####fx: -->
     <!--
     #-<init>3</init>
     -->
     <init>-1</init>
<!-- ####endfx -->
      ...
```

```
<!-- ####fx: -->
     <!--
     #-<lost>3</lost>
     -->
     <lost>-1</lost>
<!-- ####endfx -->
     </retry>
...
</router>

<log>
...
</log>

 <local>
...
   <!-- ####fx: -->
        <!--
        #-<id register-enable='true'>localhost.localdomain</id> -->
        <id realm='casafx.dyndns.org'
          require-starttls='true'
          pemfile='/etc/jabberd2/certs/c2s-1.pem'
          register-enable='false'
          password-change='false'
        >casafx.dyndns.org</id>
   <!-- ####endfx -->

</local>
```

```
<io>
   <!-- ####fx: -->
        <!--
        #-<max_fds>1024</max_fds>
        # cat /proc/sys/fs/file-max reporta 11586 en la sesio'n qemu
        -->
        <max_fds>10240</max_fds>
   <!-- ####endfx -->
...
</io>

<stats>
...
</stats>

<pbx>
<!-- ####fx: se utilizan para, usando Asterisk AGI scripts, cambiar
             el estado en jabber, "answering a call..." etc.
     ####endfx
-->
...
</pbx>

<auth>
...
```

```
    <!-- ####fx: -->
    <!-- #Teniendo en cuenta que hemos declarado requiretls para este
         #REALM y que vamos a utilizar SASL-GSSAPI como u'nico
         #mecanismo, tenemos que:
         #- module no es reelevante porque usaremos SASL-GSSAPI
         # (da igual que deje sqlite, entonces).
         #- mecanismos no sasl ("traditional"): deshabilitados,
         #  a clientes antiguos que no inicien SASL no se les
         #  anunciara' nada, pues usamos SASL-GSSAPI.
         #- mecanismos sasl: gssapi
         #- Adicionalmente se an~aden a los anteriores los
         #mecanismos que se ofrecen so'lo si TLS fue establecido;
         #PLAIN viene preconfigurado asi'. Entonces so'lo vamos
         #a impedir que se ofrezca PLAIN, pues ya
         #declaramos SASL-GSSAPI en la seccio'n anterior.
         #
         #Nota: el cliente suele elegir, de entre los que se
         #le ofrecen, el mecanismo ma's seguro, a no ser que
         #no lo soporte o este' configurado expli'citamente
         #para otro mecanismo.
      -->
    <!-- ####endfx -->
<module>sqlite</module>
<mechanism>
    <!-- ####fx: -->
         <!--
         #-<traditional>  <plain/> <digest/>  </traditional>
         -->
    <!-- ####endfx -->
```

```
    <sasl>
      <!-- ####fx: -->
           <gssapi/>
           <!-- Eliminar estos mecanismos invalida el modulo que
           #los fuese a usar (sqlite)
           #-<plain/>
           #-<digest-md5/>
           -->
      <!-- ####endfx -->
    ...
    </sasl>
  </mechanism>

  <ssl-mechanisms>
    <!-- ####fx: -->
    <!-- #Eliminar estos mecanismos invalida el mo'dulo que
         #los fuese a usar (sqlite)
         #-<traditional>
         #-  <plain/>
         #-</traditional>
         #-<sasl>
         #-  <plain/>
         #-</sasl>
         -->
    <!-- ####endfx -->
  </ssl-mechanisms>
  ...
</auth>
```

Componente s2s

Aquí se configura principalmente:

- id, nombre que identifica a este componente: s2s-1.jabberd2.casafx.dyndns.org.
- certificado para las conexiones con otros servidores jabber.

Sólo puede haber un componente s2s que representa la ruta por defecto a otros sistemas federados con nosotros (es decir, que sirven cuentas que no llevan casafx.dyndns.org tras la arroba). puede haber otro s2s que represente una ruta en concreto, pero no otro s2s para la ruta por defecto. Nuestra estrategia será hacer que el componente s2s de dklab1 se registre como ruta por defecto en el componente router, y que, como éste, sea arrancado y supervisado por init para lidiar frente a hipotéticas caídas del componente. Ante labores de administración en dklab1, basta con sincronizar la descarga del s2s en dklab1 con la carga del s2s en dklab2 para que el componente router le deje tomar el relevo, sin más acciones por nuestra parte (afortunadamente s2s es, para lo que nos ocupa aquí, un cliente, luego los demás no apuntan a él en sus ficheros de configuración como ocurría con el router, y por ello hacer el relevo de routers conlleva más acciones como editar ficheros, mientras que para s2s no).

```
vim /etc/jabberd2/s2s.xml
```

```
<!-- ####fx:-->
     <!--
     #-<id>s2s</id>
     -->
     <id>s2s-1.jabberd2.casafx.dyndns.org</id>
<!-- ####endfx-->
...
<router>
    <ip>127.0.0.1</ip>             <!-- default: 127.0.0.1 -->
...
    <pass>secret</pass>            <!-- default: secret -->
...
<!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/s2s-1.pem</pemfile>
<!-- ####endfx -->
```

```
    <retry>
<!-- ####fx: -->
     <!--
     #-<init>3</init>
     -->
     <init>-1</init>
<!-- ####endfx -->
     ...
<!-- ####fx: -->
     <!--
     #-<lost>3</lost>
     -->
     <lost>-1</lost>
<!-- ####endfx -->
    </retry>
...
</router>

<log>
...
</log>

<local>
   <ip>0.0.0.0</ip>
   ...
<!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/s2s-1.pem</pemfile>
<!-- ####endfx -->
</local>
```

```
<io>
...
   <!-- ####fx: -->
        <!--
        #-<max_fds>1024</max_fds>
        # cat /proc/sys/fs/file-max reporta 11586 en la sesio'n qemu
        -->
        <max_fds>10240</max_fds>
   <!-- ####endfx -->
<io>

<check>
...
</check>

<stats>
...
</stats>

<lookup>
...
</lookup>
```

Kerberización de jabberd2-c2s

Al componente jabberd2-c2s se conectarán los clientes y, según vimos, éste está configurado para ofrecerles el mecanismo de autenticación sasl-gssapi. los clientes utilizarán su tgt y al kerberos kdc para conseguir un ticket de servicio jabber, por su lado jabberd2-c2s tiene que conseguir el propio ante el kerberos kdc previa autenticación. para que ésto ocurra necesita tener su propio principal y su "password" en forma de keytab.

El nombre del principal sebe ser de la forma xmpp/<FQDN>@<realm>[22], donde FQDN es el nombre con que se referirán los clientes a este host, y debe ser la salida de "hostname –fqdn" puesto que no existe en el c2s.xml una opción para configurarlo de otra manera y buscarlo en el keytab.

```
kadmin.local
> addprinc -policy service -randkey xmpp/dklab1.casafx.dyndns.org

> ktadd -k /etc/keytab.d/jabberd2.keytab \
        -norandkey xmpp/dklab1.casafx.dyndns.org
> quit

klist -ke /etc/keytab.d/jabberd2.keytab

chown jabber:root /etc/keytab.d/jabberd2.keytab
chmod go-rw /etc/keytab.d/jabberd2.keytab
su jabber -m -c 'klist -t -K -k /etc/keytab.d/jabberd2.keytab'
```

Debemos anunciar a jabberd2-c2s dónde está su keytab.

```
printf '
\n####fx:\nexport KRB5_KTNAME=/etc/keytab.d/jabberd2.keytab\n####endfx' \
 >> /etc/default/jabberd2
```

Nótese que este cambio no afectará al funcionamiento del otro componente, jabberd2-sm, y SASL-GSSAPI contra PostgreSQL: jabberd2-sm se conecta como cliente y por tanto la variable que le afecta es KRB5CCNAME (y no KRB5_KTNAME). Como además no encontrará esa variable definida, utilizará la localización por defecto /tmp/krb5cc_<uid>y, en virtud del proceso k5start en /etc/inittab, encontrará ahí su TGT para coseguir el ticket de servicio y conectarse a PostgreSQL. Sin embargo, para jabberd2-c2s necesitamos declarar este KRB5_KTNAME pues usamos una ruta no estándar para acceder al keytab. En el keytab, jabberd2-c2s está programado para buscar la llave para el principal xmmp/<FQDN>, mientras que k5start está programado para (por el problema que comentamos de longitud máxima de caracteres en inittab), coger la primera.

[22] http://trac.gajim.org/ticket/2465

Reinicio de Jabberd2

```
invoke-rc.d jabberd2 restart


ps aux | (sleep 1; grep jabberd2- )
```

```
jabber   /usr/sbin/jabberd2-router -c /etc/jabberd2/router.xml
jabber   /usr/sbin/jabberd2-sm     -c /etc/jabberd2/sm.xml
jabber   /usr/sbin/jabberd2-s2s    -c /etc/jabberd2/s2s.xml
jabber   /usr/sbin/jabberd2-c2s    -c /etc/jabberd2/c2s.xml
```

Si no aparecen los cuatro componentes, quizás haya un error de sintaxis en algún XML. por ejemplo si no está cargado jabberd2-c2s, podemos hacer:

```
export KRB5_KTNAME=/etc/keytab.d/jabberd2.keytab && \
su -m jabber -c '/usr/sbin/jabberd2-c2s -D -c /etc/jabberd2/c2s.xml'; \
unset KRB5_KTNAME
```

... y puede devolvernos dónde está el error de sintaxis (línea 390 en este ejemplo):

```
...
config_load: parse error at line 390: mismatched tag
```

Todos los componentes deberían de conectarse al componente router; la autenticación se encripta gracias a los certificados que creamos.

```
tail -n 20 usb/var/log/jabberd2/router.log | grep online
```

```
...
[s2s-1.jabberd2.casafx.dyndns.org] online (bound to 127.0.0.1, port 50152)
[sm-1.jabberd2.casafx.dyndns.org] online (bound to 127.0.0.1, port 50151)
[c2s-1.jabberd2.casafx.dyndns.org] online (bound to 127.0.0.1, port 50153)
...
```

Supervisión de jabberd2-router y jabberd2-s2s por init

Como se expuso, hasta la fecha no puede actuar más de un componente router y un componente s2s para Jabberd2. Por ello nos planteábamos tenerlos supervisados por init

de forma que sean relanzados si por algún motivo caen.

```
vim /etc/inittab
```

```
...
####fx:
#En el tercer campo, respawn hara' que vuelva a levantarse si cae.
J2R:2345:respawn:/usr/local/bin/jabberd2-router_init_overseer_wrap.sh
J2S:2345:respawn:/usr/local/bin/jabberd2-s2s_init_overseer_wrap.sh
####endfx
```

Esos scripts son una envoltura con las tareas que hacen los script de inicio:

```
vim /usr/local/bin/jabberd2-router_init_overseer_wrap.sh
```

```
#!/bin/sh
####fx:
#No reinventamos la rueda y tomamos la configuracio'n de
#los ficheros correspondientes:
for i in `grep '\w='  /etc/jabberd2/component.d/*router`;
    do eval $i
done

eval `egrep '(user|group)='  /etc/default/jabberd2`

start-stop-daemon -c ${user}:${group}            \
            --start --pidfile ${pidfile} --quiet \
            --exec ${command} -- -c ${conffile}
exitvalue=$?
sleep 1
exit $exitvalue
####endfx
```

```
chmod u+x /usr/local/bin/jabberd2-router_init_overseer_wrap.sh
```

```
vim /usr/local/bin/jabberd2-s2s_init_overseer_wrap.sh
```

```
#!/bin/sh
####fx:
#No reinventamos la rueda y tomamos la configuracio'n
#de los ficheros correspondientes:
for i in `grep '\w='  /etc/jabberd2/component.d/*s2s`; do eval $i; done
eval `egrep '(user|group)='  /etc/default/jabberd2`

start-stop-daemon -c ${user}:${group} \
            --start --pidfile ${pidfile} --quiet \
            --exec ${command} -- -c ${conffile}
exitvalue=$?
sleep 1
exit $exitvalue
####endfx
```

```
chmod u+x /usr/local/bin/jabberd2-s2s_init_overseer_wrap.sh


invoke-rc.d jabberd2 stop
```

Si es init el encargado de lanzarlos, deshabilitamos que lo haga el script de inicio:

```
vim /etc/default/jabberd2
```

```
...
# run router


####fx:
#-router_run=1
#Cargamos el componente a trave's de inittab, luego lo
#deshabilitamos aqui'.
router_run=0
####endfx
...
####fx:
#-s2s_run=1
#Cargamos el componente a trave's de inittab, luego lo
#deshabilitamos aqui'.
s2s_run=0
####endfx
```

```
kill -hup 1
invoke-rc.d jabberd2 start
```

Podemos probar a descargar el proceso jabberd2-router y ver cómo rápidamente es suplantado por otra instancia (es decir, un proceso con otro pid):

```
pgrep -l -f jabberd2-router.*xml && \
 pkill -f jabberd2-router.*xml && sleep 3 && \
 pgrep -l -f jabberd2-router.*xml
```

Además, el resto de componentes volverán a conectarse al router automáticamente.

- Anotación: al arranque, los componentes c2s y sm se cargarán antes que router porque el script /etc/init.d/jabberd2 será lanzado antes que los procesos de /etc/inittab (que se lanzan cuando el arranque ha, básicamente, terminado). Por ello es importante que los componentes estén configurados (sección <retry>en sus ficheros de

configuración) para reintentar la conexión con el router de forma indefinida (así lo hicimos) o, al menos, un número suficiente de veces para superar ese retraso.

Algunos comentarios respecto de la creación al vuelo del roster

Como se expuso, se hizo necesario inspeccionar y parchear el código fuente para que esta funcionalidad del componente sm estuviese presente. A continuación exponemos algunas de las conclusiones sobre su funcionamiento:

- La creación y entrega del roster al usuario se realiza, básicamente:
  - Se construye un roster parcial a partir de la tabla roster-items en PostgreSQL. El contenido de esta tabla proviene de los contactos a los que el usuario jabber pidió subscripción por su cuenta (sigue pudiendo hacerlo, a no ser que declaremos <check-remove-domain>en la configuración, etc). A ello se une, cuando se ha estado usando ldap (mod_roster_publish), lo que se descubrió en LDAP en una conexión anterior, pues automáticamente se copia allí en PostgreSQL.
  - Se construye un segundo roster parcial a través de los items recolectados en LDAP ahora.
  - Se unen los anteriores roster parciales, pero modificandolos de forma que cada item tenga asociado su grupo:
    1. Si usamos la opción <forcegroups>(con un filtro adecuado), entonces el grupo se añade según lo encontrado en LDAP al buscar por groupattr (el atributo de grupo que define los grupos) y hacer match el filtro.
    2. Si no usamos forcegroups o sí lo usamos pero su filtro no hace match, el grupo se añade según la tabla roster-groups actual en PostgreSQL, y el resultado de LDAP; claramente da la posibilidad de un item en varios grupos.
  - Ejemplo (gatt se refiere al atributo de grupo; top es el grupo padre al que se pertenece si no se pertenece a otro): Tenemos gatt=uid + forcegroups, lanzamos jabberd2-sm y, tras conectarse el cliente, inyecta en PostgreSQL los contactos (tabla roster-items) y grupos (tabla roster-groups). Entonces quito en todos estos casos al user aemu de PostgreSQL (por ejemplo con "TRUNCATE <table>;") y reconecto pidgin:
    - gatt=uid + forcegroups+filtro. Es decir, lo que tenía antes:
    - el usuario aemu nuevo se mete en grupo (viejo o nuevo porque se fuerza) uid.
      - el usuario umea viejo se mete en grupo (viejo o nuevo porque se fuerza) uid.
    - gatt=top + forcegroups+filtro. Es decir, cambio groupattr y dejo forcegroups:
      - el usuario aemu nuevo se mete en grupo nuevo top.

        - el usuario aemu nuevo se mete en grupo nuevo top.
    - gatt=top - forcegroups (o +forcegroups -filtro). Es decir, no podemos forzar grupo:
        - el usuario aemu nuevo se mete en grupo nuevo top.
        - el usuario umea viejo se queda en grupo viejo uid.

- Usamos como atributo de grupo finalmente a departmentNumber[23] (definido para el objeto inetOrgPerson). Su contenido es utf8[24]; multivaluable.
- No sabemos cómo indicar con LDAP pertenecer a varios grupos: no funcionó con groupattr multivaluado, ni strings separados por espacios (en givenname). La única posibilidad es en PostgreSQL: si en ldap se resuelve una asociación item-grupo distinta de la preexistente en PostgreSQL, ambas van al roster final (a no ser que, estando activado forcegroups, haga match su filtro).
- El nombre (que no el JID) de los contactos puede ser cambiado por el usuario (se almacenará en PostgreSQL), ésto puede deshabilitarse con <override-names>.
- Existe la posibilidad de guardar aparte el mapeo de usuarios y grupos, pero no lo usamos (véase published-roster-groups driver en el sm.xml).

1.3. DKLAB2

1.3.1. Preámbulo: Jabberd2 en dklab2; clúster

Nos preguntamos cuál es el rol de Jabberd2 en dklab2. A un router se pueden conectar varias instancias de los demás componentes (sm,c2s). Por tanto en dklab2 podríamos hacer que sus componentes sm,c2s utilizasen el componente router de dklab1, haciendo un cluster[25] con los componentes sm,c2s de allí. Ésto implica que se un usuario se conecta al c2s en dklab1, y otro usuario se conecta al c2s de dklab2, se pueden comunicar entre sí, es decir se "ven" a pesar de estar conectados a servidores en máquinas distintas, debido a la labor del componente router con el que se comunican ambos c2s.

Respecto al componente s2s, el componente router sólo admite una ruta hacia otros servidores federados, y por tanto sólo admite un componente s2s a este efecto: el primero que se conecte. Por tanto el componente s2s de dklab2 podría, a lo sumo, actuar de failover si hubiese problemas con el de dklab1. Soluciones sencillas serían, por ejemplo, una línea en inittab que llame en modo respawn a un script wrapper (lo que hicimos en dklab1 para el componente router), de forma que el s2s en dklab2 intenta registrarse en el router periódicamente hasta que, por ausencia del s2s en dklab1, lo consigue y le hace el relevo.

[23] http://wurley.demo.phpldapadmin.org/php5-SANDPIT/cmd.php?cmd%3Dschema&server_id%3D1&view%3Dat
[24] http://www.openldap.org/doc/admin24/schema.html
[25] http://tomasz.sterna.tv/2009/06/clustering-for-jabberd2/
(Tomasz Sterna es el principal desarrollador de Jabberd2)

Realmente, el s2s en dklab1 está supervisado por init, así que nosotros supondremos que las conexiones a cuentas ajenas a la organización no son prioritarias y que es aceptable el retardo de sustituir el s2s de dklab1 por el de dklab2 manualmente ante tareas de mantenimiento (que por alguna razón requiriesen incondicionalmente descargar s2s en dklab1). Si hubiese que desconectar dklab1, puesto que no podía haber dos routers en jabberd2, el servicio se interrumpiría.

1.3.2. Instalación

Podemos usar los paquetes que creamos en dklab1 (considerando que las arquitecturas del procesador en dklab1 y dklab2 lo permiten, como siempre).

```
mkdir -p ~/deb #/usr/src/jabberd2
               #cd /usr/src/jabberd2
scp dklab1.casafx.dyndns.org:/root/deb/*.deb ~/deb/
apt-get install --no-install-recommends \
        libgsasl7 libmysqlclient16 libntlm0 libpq5
dpkg -i ~/deb/*.deb
invoke-rc.d jabberd2 stop
cd ~
```

1.3.3. Soporte para centralización de metadatos: LDAP, SQL

Respecto a LDAP, la discusión que tuvimos en dklab1 sigue siendo válida: no tenemos que importar esquemas LDAP sino dejar que jabberd2-sm se conecte anónimamente y rehuse lo que ya tenemos.

Respecto a PostgreSQL, debemos no sólo volver a cargar el esquema, sino también a crear la base de datos, todo ello a pesar de que Bucardo replicará la base dedatos de Jabberd2. La razón es que Bucardo necesita que la base de datos preexista. A continuación, daremos soporte al componente jabberd2-sm para usar autenticación KERBEROS contra PostgreSQL como hiciéramos para el componente jabberd2-sm en dklab1.

Creación en PostgreSQL de la base de datos jabberd2 y el rol jabberd2

```
grep ^[^#] /etc/postgresql/9.0/main/pg_hba.conf | grep postgres
```

```
#DB usando tabla unicode
su -s /bin/sh postgres -c "createdb -U postgres -E UNICODE jabberd2"
#Rol jabberd2
su -s /bin/sh postgres -c "createuser -U postgres
                  --no-createrole --no-createdb --no-superuser jabberd2"
```

Configuración de autenticación KERBEROS para acceder a PostgreSQL

Mapeo KERBEROS principal ->PostgreSQL Rol Nos permitía mapear al rol jabberd2 tanto el principal de jabberd2-sm en dklab1 como en dklab2:

```
grep gss /etc/postgresql/9.0/main/pg_hba.conf


cat <<EOF >> /etc/postgresql/9.0/main/pg_ident.conf
####fx:
krbprinc-pgrole jabberd2-sm/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG jabberd2
krbprinc-pgrole jabberd2-sm/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG jabberd2
####endfx
EOF


invoke-rc.d postgresql reload
```

Creación del principal para jabberd2-sm, exportación a keytab y configuración de k5start Creamos principal jabberd2-sm/dklab2.casafx.dyndns.org, que jabberd2-sm usará para conseguir TGT y ticket del servicio PostgreSQL en dklab2. Para que jabberd2-sm tenga disponible y puntualmente renovado su TGT, utilizábamos k5start supervisado por init:

```
kadmin.local
> addprinc -policy service \
           -randkey jabberd2-sm/dklab2.casafx.dyndns.org
> ktadd -k /etc/keytab.d/jabberd2.keytab \
        -norandkey jabberd2-sm/dklab2.casafx.dyndns.org
> quit

chown jabber:root /etc/keytab.d/jabberd2.keytab
chmod go-rw /etc/keytab.d/jabberd2.keytab
su jabber -m -c 'klist -t -K -k /etc/keytab.d/jabberd2.keytab'

vim /etc/inittab
```

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

```
...
####fx:
KSJ:2345:respawn:/usr/bin/k5start -U -f /etc/keytab.d/jabberd2.keytab \
-K 10 -l 24h -k /tmp/krb5cc_`id -u jabber` -o jabber
####endfx
```

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

Test Pedimos un listado de las bases de datos en dklab2 y, aprovechando que está cargada, también en dklab1. Efectivamente tenemos el mapeo principal a rol configurados también en ambos. El TGT lo conseguimos no interactivamente a partir del keytab:

```
kdestroy
kinit -k -t /etc/keytab.d/jabberd2.keytab \
      jabberd2-sm/dklab2.casafx.dyndns.org

psql -h dklab2.casafx.dyndns.org -l -U jabberd2
psql -h dklab1.casafx.dyndns.org -l -U jabberd2
klist
```

Importación del esquema SQL de Jabberd2; modificaciones para Bucardo

```
cd ~
mkdir sql
zcat /usr/share/doc/jabberd2/db-setup.pgsql.gz > sql/j2pgsql.sql
grep ^CREATE sql/j2pgsql.sql
```

Nos aseguramos de que cada tabla tenga su Primary-Key, requisto de Bucardo:

```
sed -i '142s/"xml" text NOT NULL/"xml" text NOT NULL, PRIMARY KEY
 ("object-sequence")/g' sql/j2pgsql.sql
sed -i 's/"node" text/"node" text, PRIMARY KEY
 ("object-sequence")/g' sql/j2pgsql.sql
sed -i 's/"block" integer/"block" integer, PRIMARY KEY
 ("object-sequence")/g' sql/j2pgsql.sql
sed -i 's/CREATE SEQUENCE "object-sequence";/CREATE SEQUENCE
 "object-sequence" START WITH 2 INCREMENT BY 2;/g' sql/j2pgsql.sql
```

Aprovechando el TGT obtenido en el test anterior, cargamos el esquema:

```
grep ^[^#] /etc/postgresql/9.0/main/pg_hba.conf
klist


psql -h dklab2.casafx.dyndns.org -U jabberd2 jabberd2 -f sql/j2pgsql.sql
```

1.3.4. Configuración de Jabberd2

Recordamos la tabla sinóptica con las peculiaridades principales de cada fichero de configuración asociado a su componente, así como las reglas básicas para editarlos:

Componente	Parámetro	Función
sm	id	Único, simplemente para identificar esta instancia.
	local id	La parte del JID, uno por cada host virtual (todos los puede manejar un sólo sm) si necesitas ya que cada vhost tenga una configuración !=, hay q lanzar varios procesos.
	storage type	Backend usado según /usr/lib/jabberd2/storage_*.
	auto-create	Si un usuario hace log-in satisfactoriamente, se le crea la cuenta XMPP automáticamente.
c2s	id	Único, simplemente para identificar esta instancia si vacío es jid y es default de entre los vhosts.
	auth type	Auth usada según /usr/lib/jabberd2/authreg_*) y mechanismos SASL[26] y tradicional[27].
	local id	Un FQDN que coincida con el local-id de su sm
	realm	Desde el punto de vista de la auth, un user pertenece a un REALM, no a su JID, de forma que varios JID pueden ser los mismos users (misma auth) si no dices nada es tu id. Si indicas "" y usas PAM, éste no mira el JID, sólo el nombre local, por cierto.
	user-accountreg	Si el usuario puede crear su cuenta vía XMPP
router	id	Único, simplemente para identificar esta instancia.
	clusters	Varios sm/transport para un sólo JID (el clustering component que controla quién atiende a qué cuenta no ha habido que implementarlo/integrarlo puesto que este componente ya lo hace[28]).
s2s	id	Único, simplemente para identificar esta instancia.

- Son ficheros xml, hay que respetar las reglas del xml bien formado.
- No admite tildes o caracteres no ASCII en los comentarios.

Certificados X.509 para comunicaciones entre componentes

Movemos los ficheros con la cadena de certificados requerida por Jabberd2 que creamos en dklab1:

```
CERTDIR=/etc/jabberd2/certs
mkdir -p ${CERTDIR}
scp dklab1.casafx.dyndns.org:/etc/jabberd2/certs/*-2.pem ${CERTDIR}
for i in router-2 sm-2 c2s-2 s2s-2
 do chown root:jabber ${CERTDIR}/${i}.pem
    chmod 640         ${CERTDIR}/${i}.pem
done
ls -l ${CERTDIR}
```

Podemos borrarlos de allí:

```
ssh dklab1.casafx.dyndns.org "/bin/rm -f /etc/jabberd2/certs/*-2.pem"
```

Componente router

Lo configuraremos pero no lo usaremos mientras no haya problemas en dklab1. (Evidentemente los demás componentes en dklab2 apuntarán al router en dklab1, como veremos).

Así, deshabilitamos su carga al inicio modificando el /etc/default/jabberd2:

```
sed \
 -i 's/ROUTER_RUN=1/####fx:\n#-ROUTER_RUN=1\nROUTER_RUN=0\n####endfx/g' \
 /etc/default/jabberd2
```

```
chown jabber:root /etc/jabberd2/router*.xml
chmod o-r /etc/jabberd2/router*.xml


vim /etc/jabberd2/router.xml
```

```
...
<!-- ####fx:-->
     <!--
     #-<id>router</id>
     -->
     <id>router-2.jabberd2.casafx.dyndns.org</id>
<!-- ####endfx-->

<log>
...
</log>

<local>
 ...
     <pass>secret</pass>           <!-- default: secret -->
 ...
 <!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/router-2.pem</pemfile>
 <!-- ####endfx -->
</local>
```

```
<check>
 ...
<check>


<io>
...
</io>


<aliases>
...
</aliases>


<aci>
...
</aci>
```

Como mencionamos en dklab1, hay ficheros adicionales como el router-users.xml (que controla quién y con qué credencial accede al componente router). Nosotros usaremos siempre los valores por defecto establecidos allí (jabber:secret), si bien en un entorno en producción habría que cambiarlos allí y, acorde, en el resto de *.xml que vienen a continuación.

Componente sm

Se configura principalmente:

- ID, nombre que identifica a este componente: sm-2.jabberd2.casafx.dyndns.org
- Local ID, la parte tras la @ en el nombre de las cuentas: casafx.dyndns.org
- Tipo de almacenamiento para los metadatos asociados a las cuentas: pgsql, ldap. No soporta el descubrimiento de los servidores ldap usando registros SRV.
- Política de creación automática de cuenta si autenticación exitosa: activado.

```
chown jabber:root /etc/jabberd2/sm.xml
chmod o-r /etc/jabberd2/sm.xml


vim /etc/jabberd2/sm.xml
```

```
...
<!-- ####fx:-->
     <!--
     #-<id>sm</id>
     -->
     <id>sm-2.jabberd2.casafx.dyndns.org</id>
<!-- ####endfx-->


...


<router>
<!-- ####fx: NOTA: podri'amos tener la tentacio'n de crear varios
     #       bloques router.../router, uno para dklab1 y otro para
     #       el de dklab2... no funcionara', se usa sistema'ticamente
     #       so'lo el primero. Igualmente pasara' en los bloques
     #       router del resto de componentes. Actualmente, so'lo puede
     #       haber un componente router en la red.
     ####endfx
-->
```

```
<!-- ####fx: -->
     <!--
     # IP del componente router en dklab1
     #-<ip>127.0.0.1</ip>
     -->
    <ip>10.168.1.1</ip>             <!-- default: 127.0.0.1 -->
<!-- ####endfx -->


...
    <pass>secret</pass>           <!-- default: secret -->
...
<!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/sm-2.pem</pemfile>
<!-- ####endfx -->

    <retry>
<!-- ####fx: -->
     <!--
     #-<init>3</init>
     -->
     <init>-1</init>
<!-- ####endfx -->
     ...
```

```
<!-- ####fx: -->
     <!--
     #-<lost>3</lost>
     -->
     <lost>-1</lost>
<!-- ####endfx -->
    </retry>
...
</router>

<log>
...
</log>

<local>
 <!-- ####fx: la parte tras la arroba -->
      <!--
      #-<id>localhost.localdomain</id>
      -->
      <id>casafx.dyndns.org</id>
 <!-- ####endfx -->
 ...
</local>
```

```
<storage>
...
   <!-- ####fx: -->
        <!--
        #-<driver>sqlite</driver>
        -->
        <driver>pgsql</driver>
   <!-- ####endfx -->

   <!-- ####fx: -->
        <driver type='vcard'>ldapvcard</driver>
        <driver type='published-roster'>ldapvcard</driver>
   <!-- ####endfx -->

   <!--
   <driver type='published-roster-groups'>ldapvcard</driver>
   -->
...
```

```
<pgsql>
    <!-- ####fx:
      http://www.postgresql.org/docs/9.0/static/libpq-connect.html -->
    <!-- #Tal como veni'a, usari'a socket unix y auth ident, pues
         #no indica host:
  #-<conninfo>dbname=jabberd2 user=jabberd2 password=secret</conninfo>
         #Si pongo varios host (o en gnl repito cualquier variable),
         #se usa so'lo el u'ltimo valor; si pongo varios <conninfo>,
         #se usa so'lo el primero.
         #No puede, pues, indicarse que use uno de entre varios
         #hosts, desde aqui'.
        -->
        <conninfo>
         dbname=jabberd2 user=jabberd2 host=dklab2.casafx.dyndns.org
        </conninfo>
   <!-- ####endfx -->
...
</pgsql>
```

```
<ldapvcard>
   <!-- ####fx: -->
        <!--
        #Pretendimos que usase los registros SRV, pero da error:
        #[error] ldapvcard: no uri specified in config file
        #-<uri>ldap://localhost/ ldaps://ldap.example.com/</uri>
        -->
        <uri>
         ldap://dklab2.casafx.dyndns.org
         ldap://dklab1.casafx.dyndns.org
        </uri>
   <!-- ####endfx -->
   ...
   <uidattr>uid</uidattr>
   <objectclass>posixAccount</objectclass>
   <!-- ####fx: -->
        <!--
        #No se debiera contrastar password alguna pues no lo hay
        #al usar KERBEROS
        #-<pwattr>userPassword</pwattr>
        -->
   <!-- ####endfx -->
   <!-- ####fx: -->
        <!--
        #-<basedn>o=Example Corp.</basedn>
        -->
        <basedn>
         ou=users,ou=accounts,dc=casafx,dc=dyndns,dc=org</basedn>
        <groupattr>departmentNumber</groupattr>
   <!-- ####endfx -->
```

```
    ...
    <publishedcachettl>60</publishedcachettl>
    <mapped-groups>
    ...
    </mapped-groups>
  </ldapvcard>
</storage>

 <aci>
   <acl type='all'>
     <!-- ####fx: -->
          <!--
          #-<jid>admin@localhost.localdomain</jid>
          -->
          <jid>root/admin@casafx.dyndns.org</jid>
     <!-- ####endfx -->
   </acl>
 ...
 </aci>

 <modules>
 ...
 </modules>

 <discovery>
 ...
```

```
  <!-- ####fx: -->
       <serverinfo>
        <admin-addresses>
         <value>mailto:jabbermaster@casafx.dyndns.org</value>
        </admin-addresses>
       </serverinfo>
  <!-- ####endfx -->
</discovery>

<user>
  <!-- ####fx: -->
       <auto-create/>
  <!-- ####endfx -->
...
```

```
<template>
   <publish>
   <!-- Uncomment <publish> if you wish to forcely publish
      roster template from LDAP on each user login.
    -->
    <!-- ####fx: -->
    <!-- #Si no defines <publish> para roster en ldap, puedes
         #definir <roster> para roster.xml local.
         <publish realm='casafx.dyndns.org'>
    <!-- ####endfx -->
   ...
    <!-- ####fx: -->
         <!--
         #El nombre (que no el JID) de los contactos puede
         #ser cambiado por el usuario (y se almacenara'
         #en PostgreSQL), e'sto puede deshabilitarse
         # con <override-names>.
         -->
    <!-- ####endfx -->
      <!--
      <override-names/>
      -->
```

```
      <!-- ####fx: -->
           <force-groups>
            <prefix>departmentNumber</prefix>
            <!-- <suffix>(MyOrg)</suffix> -->
           </force-groups>
      <!-- ####endfx -->
   ...
    <!-- ####fx: -->
         </publish>
    <!-- ####endfx -->

    <!-- ####fx: -->
    <!-- #Si en el futuro queremos una cuenta especial no en ldap,
         #como serviciotecnico@casafx.dyndns.org, que deba ser an~adida
         #a todo usuario automa'ticamente, podemos declararlo en un
         #fichero y activar la siguiente opcio'n.
         #(Este roster.xml existe y contiene algu'n ejemplo de uso):
    <!-- ####endfx -->
    <!--
    <roster>/etc/jabberd2/templates/roster.xml</roster>
    -->
 </template>
</user>
```

```
<amp>
...
</amp>

<offline>
...
</offline>

<roster>
...
</roster>

<status>
...
</status>
```

Componente c2s

Configuramos principalmente:

- ID, nombre que identifica a este componente: c2s-2.jabberd2.casafx.dyndns.org.
- Tipo de autenticación: SASL-GSSAPI.
- Local ID, la parte tras la @ en el nombre de las cuentas. Debe coincidir con lo que se especificó en <local>en sm.xml: casafx.dyndns.org
- Tipo de autentificación: normalmente se elije uno de los módulos listados en /usr/lib/jabberd2. Pero no hay ninguno para sasl o directamente gssapi, ésto tiene como consecuencia que, en lo que a sasl-gssapi se refiere, no importa el módulo pongas, lo importante es que especifiques que sasl, compilado dinámicamente en jabberd2-c2s, anuncie el mecanismo gssapi a los clientes.

```
ls /usr/lib/jabberd2 | grep '(sasl|gssapi)'
```

```
ldd /usr/sbin/jabberd2-c2s | grep gssapi
```

- Política de registro de nueva cuenta por el usuario a través de su cliente: prohibido.
- Certificado para las conexiones con el cliente, por las deficiencias de gsasl.

```
chown jabber:root /etc/jabberd2/c2s.xml
chmod o-r /etc/jabberd2/c2s.xml


vim /etc/jabberd2/c2s.xml
```

```
...
<!-- ####fx:-->
     <!--
     #-<id>c2s</id>
     -->
     <id>c2s-2.jabberd2.casafx.dyndns.org</id>
<!-- ####endfx-->
...
<router>
<!-- ####fx: -->
     <!--
     # IP del componente router en dklab1
     #-<ip>127.0.0.1</ip>
     -->
    <ip>10.168.1.1</ip>            <!-- default: 127.0.0.1 -->
<!-- ####endfx -->
...
    <pass>secret</pass>           <!-- default: secret -->
...
<!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/c2s-2.pem</pemfile>
<!-- ####endfx -->
```

```
    <retry>
<!-- ####fx: -->
     <!--
     #-<init>3</init>
     -->
     <init>-1</init>
<!-- ####endfx -->
     ...
<!-- ####fx: -->
     <!--
     #-<lost>3</lost>
     -->
     <lost>-1</lost>
<!-- ####endfx -->
    </retry>
...
</router>

<log>
...
</log>

<local>
...
```

```
   <!-- ####fx: -->
        <!--
        #-<id register-enable='true'>localhost.localdomain</id> -->
        <id realm='casafx.dyndns.org'
          require-starttls='true'
          pemfile='/etc/jabberd2/certs/c2s-2.pem'
          register-enable='false'
          password-change='false'
        >casafx.dyndns.org</id>
   <!-- ####endfx -->

</local>

<io>
   <!-- ####fx: -->
        <!--
        #-<max_fds>1024</max_fds>
        # cat /proc/sys/fs/file-max reporta 11586 en la sesio'n qemu
        -->
        <max_fds>10240</max_fds>
   <!-- ####endfx -->
...
</io>

<stats>
...
</stats>
```

```
<pbx>
<!-- ####fx: se utilizan para, usando Asterisk AGI scripts,
     #       cambiar el estado en jabber, "answering a call..." etc.
     ####endfx
-->
...
</pbx>
<auth>
...
     <!-- ####fx: -->
     <!-- #Teniendo en cuenta que hemos declarado requiretls
          #para este realm y que vamos a utilizar sasl-gssapi
          #como u'nico mecanismo, tenemos que:
          #- module no es reelevante porque usaremos SASL-GSSAPI
          #(da igual que ponga sqlite, entonces)
          #- mecanismos no sasl ("traditional"): deshabilitados,
          #a clientes antiguos que no inicien SASL no se les
          #anunciara' nada, pues usamos SASL-GSSAPI.
          #- mecanismos sasl: gssapi
          #- Adicionalmente se an~aden a los anteriores los mecanismos
          #que se ofrecen so'lo si TLS fue establecido; PLAIN
          #viene preconfigurado asi'. Entonces so'lo vamos a impedir
          #que se ofrezca PLAIN, pues ya declaramos SASL-GSSAPI en
          #la seccio'n anterior.

          #Nota: el cliente suele elegir, de entre los que se
          #le ofrecen, el mecanismo ma's seguro, a no ser que no
          #lo soporte o este' configurado expli'citamente para
          #otro mecanismo.
       -->
```

```
    <!-- ####endfx -->
<module>sqlite</module>
<mechanism>
    <!-- ####fx: -->
        <!--
        #-<traditional>  <plain/> <digest/>  </traditional>
        -->
    <!-- ####endfx -->

    <sasl>
      <!-- ####fx: -->
           <gssapi/>
      <!-- #Eliminar los siguientes mecanismos invalida el mo'dulo
           #que los fuese a usar (sqlite)
           #-<plain/>
           #-<digest-md5/>
           -->
      <!-- ####endfx -->
    ...
    </sasl>
</mechanism>
```

```
  <ssl-mechanisms>
    <!-- ####fx: -->
         <!-- Eliminar estos mecanismos invalida el modulo
         #que los fuese a usar (sqlite)
         #-<traditional>
         #-  <plain/>
         #-</traditional>
         #-<sasl>
         #-  <plain/>
         #-</sasl>
         -->
    <!-- ####endfx -->
  </ssl-mechanisms>
  ...
</auth>
```

Componente s2s

Lo configuramos, pero no lo usamos excepto si se detectan problemas en el componente s2s de dklab1. Así, lo deshabilitamos:

```
sed \
 -i 's/S2S_RUN=1/####fx:\n#-S2S_RUN=1\nS2S_RUN=0\n####endfx/g' \
 /etc/default/jabberd2
```

En el s2s.xml se configura principalmente:

- ID, nombre que identifica a este componente: s2s-2.jabberd2.casafx.dyndns.org.

```
chown jabber:root /etc/jabberd2/s2s.xml
chmod o-r /etc/jabberd2/s2s.xml


vim /etc/jabberd2/s2s.xml
```

```
<!-- ####fx:-->
     <!--
     #-<id>s2s</id>
     -->
     <id>s2s-2.jabberd2.casafx.dyndns.org</id>
<!-- ####endfx-->
...
<router>
<!-- ####fx: -->
     <!--
     # IP del componente router en dklab1
     #-<ip>127.0.0.1</ip>
     -->
    <ip>10.168.1.1</ip>             <!-- default: 127.0.0.1 -->
<!-- ####endfx -->
...
    <pass>secret</pass>           <!-- default: secret -->
...
<!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/s2s-2.pem</pemfile>
<!-- ####endfx -->
```

```
    <retry>
<!-- ####fx: -->
     <!--
     #-<init>3</init>
     -->
     <init>-1</init>
<!-- ####endfx -->
     ...
<!-- ####fx: -->
     <!--
     #-<lost>3</lost>
     -->
     <lost>-1</lost>
<!-- ####endfx -->
    </retry>
...
</router>

<log>
...
</log>

<local>
   <ip>0.0.0.0</ip>
   ...
<!-- ####fx: -->
     <pemfile>/etc/jabberd2/certs/s2s-2.pem</pemfile>
<!-- ####endfx -->
</local>
```

```
<io>
...
   <!-- ####fx: -->
        <!--
        #-<max_fds>1024</max_fds>
        # cat /proc/sys/fs/file-max reporta 11586 en la sesio'n qemu
        -->
        <max_fds>10240</max_fds>
   <!-- ####endfx -->
<io>

<check>
...
</check>

<stats>
...
</stats>

<lookup>
...
</lookup>
```

Kerberización de jabberd2-c2s

Creamos su principal, lo exportamos, lo anunciamos. Con ello los clientes podrán hacer autenticación SASL-GSSAPI:

```
kadmin.local
> addprinc -policy service \
           -randkey xmpp/dklab2.casafx.dyndns.org
> ktadd -k /etc/keytab.d/jabberd2.keytab \
        -norandkey xmpp/dklab2.casafx.dyndns.org
> quit
klist -ke /etc/keytab.d/jabberd2.keytab


chown jabber:root /etc/keytab.d/jabberd2.keytab
chmod go-rw /etc/keytab.d/jabberd2.keytab
su jabber -m -c 'klist -t -K -k /etc/keytab.d/jabberd2.keytab'
```

```
printf  \
'\n####fx:\nexport KRB5_KTNAME=/etc/keytab.d/jabberd2.keytab\n####endfx\n' \
 >> /etc/default/jabberd2
```

Reinicio del servicio

```
invoke-rc.d jabberd2 restart


ps aux | (sleep 1; grep jabberd2- )
```

```
 jabber   /usr/sbin/jabberd2-sm     -c /etc/jabberd2/sm.xml
 jabber   /usr/sbin/jabberd2-s2s    -c /etc/jabberd2/s2s.xml
 jabber   /usr/sbin/jabberd2-c2s    -c /etc/jabberd2/c2s.xml
```

Si, por ejemplo no aparece jabberd2-c2s y sospechamos un error de sintaxis, recordamos que debíamos aprovechar el soporte de depuración con que se compiló Jabberd2 para conocer la localización exacta del error:

```
export KRB5_KTNAME=/etc/keytab.d/jabberd2.keytab && \
su -m jabber -c '/usr/sbin/jabberd2-c2s -D -c /etc/jabberd2/c2s.xml'; \
unset KRB5_KTNAME
```

Todos los componentes deberían de conectarse (de forma segura gracias a los certificados que creamos) al componente router en dklab1:

```
ssh dklab1.casafx.dyndns.org \
    'tail -n 20 /var/log/jabberd2/router.log' | grep online
```

```
...
[c2s-1.jabberd2.casafx.dyndns.org] online (bound to 127.0.0.1, port 58635)
[s2s-1.jabberd2.casafx.dyndns.org] online (bound to 127.0.0.1, port 58636)
[sm-1.jabberd2.casafx.dyndns.org] online (bound to 127.0.0.1, port 58637)
[casafx.dyndns.org] online (bound to 127.0.0.1, port 58637)

[c2s-2.jabberd2.casafx.dyndns.org] online (bound to 10.168.1.2, port 51313)
[sm-2.jabberd2.casafx.dyndns.org] online (bound to 10.168.1.2, port 51314)
[casafx.dyndns.org]:2 online (bound to 10.168.1.2, port 51314)
...
```

1.4. Replicación metadatos en PostgreSQL, configuración específica de Bucardo para Jabberd2

La configuración de Bucardo[29] para sincronizar una base de datos entre dos Master, tiene las siguientes peculiaridades:

- Necesitamos declarar:
  - Qué DB se sincronizan entre sí, en qué hosts (y puerto) están cada una y con qué rol de PostgreSQL se accede a ellas (será el rol bucardo pues tiene autorización de super-user).
  - Dentro de la DB, qué tablas. Ésto es llamado herd (manada) por Bucardo, y en nuestro caso sólo hay uno y las contiene a todas.

[29] http://bucardo.org/wiki/Bucardo/pgbench_example

    - Qué tipo de mecanismo de sincronización se aplica para ese grupo de tablas (herd). Puesto que cada tabla tiene su Primary-Key, podemos elegir el mecanismo swap[30] el cual opera a nivel de fila. Además, unque la sintaxis parezca (targetdb=jabberd2dklab2) que define un sentido de la sincronización dklab1 a dklab2, no es así y será bidireccional (Bucardo tiene tipos de sincronizaciones unidireccionales, pero tienen otro nombre, no es swap).
    - Deberíamo elegir también una política de solución de conflictos, por el carácter asíncrono de Bucardo (véase discusión a este respecto en el capítulo dedicado a PostgreSQL).

- Los datos anteriores se guardan en su base de datos. Nosotros haremos que Bucardo en dklab1 tenga su configuración en el PostgreSQL en dklab1.
- Si en algún momento Bucardo en dklab2 debiera sustituir a Bucardo en dklab1 y asumir el control, habría que apuntar (la configuración de Bucardo en dklab2) al PostgreSQL en dklab1, o volver a introducir la configuración tal como lo vamos a hacer a continuación en el PostgreSQL de dklab2. Adicionalmente, habría que activarlo en /etc/default/bucardo.

Vamos a ello. Como se ha explicado, el primer paso es declarar las dos DB (jabberd2 en dklab1 y jabberd2 en dklab2) que pretendemos sincronizar entre sí:

```
su bucardo -c 'bucardo_ctl add db jabberd2
                                  name=jabberd2dklab1
                                  port=5432
                                  host=dklab1.casafx.dyndns.org
                                  user=bucardo'
su bucardo -c 'bucardo_ctl add db jabberd2
                                  name=jabberd2dklab2
                                  port=5432
                                  host=dklab2.casafx.dyndns.org
                                  user=bucardo'


su bucardo -c 'bucardo_ctl list db'
```

Es necesario solucionar un conflicto entre la utilidad de bucardo_ctl y el esquema de autenticación KERBEROS de PostgreSQL. Parece ser que el script bucardo_ctl falla

[30] http://bucardo.org/wiki/Swap

sistemáticamente cuando trata de añadir las tablas, si la cuenta local postgres no puede usar el principal de bucardo. Suponemos que se debe a que bucardo_ctl no ha sido diseñado[31] con autenticaciones KERBEROS en mente. Debemos copiarle el TGT pues (sólo es necesario ahora durante la configuración, el servidor bucardo no lo necesita luego y por lo tanto borraremos la copia tras configurar).

```
cp /tmp/krb5cc_‘id -u bucardo‘ /tmp/krb5cc_‘id -u postgres‘
chown postgres /tmp/krb5cc_‘id -u postgres‘
```

Ya podemos seleccionar las tablas que serán sincronizadas:

```
su bucardo -c 'bucardo_ctl add all tables
                                    db=jabberd2dklab1
                                    herd=tablas_jabberd2'

su bucardo -c 'bucardo_ctl list herds'
su bucardo -c 'bucardo_ctl list tables'
```

Debemos proveer alguna política sobre resolución de conflictos, en otro caso bucardo no nos permitirá configurar una sincronización utilizando las tablas anteriores. La política puede ser[32]: "source" "target", "skip", "random", "abort" y "latest". Elegimos "latest" (el último cambio tiene preferencia ante conflictos) para todas las tablas de jabberd2. Como es posible (o no hemos encontrado cómo en la documentación), pedir a bucardo_ctl que lo haga por nosotros, modificamos la base de datos pertinente usando psql:

```
su bucardo -c psql\ -h\ dklab1\ -U\ bucardo\ bucardo\ -c\ \
   "UPDATE\ goat\ SET\ standard_conflict=\'latest\';"

su bucardo -c psql\ -h\ dklab1\ -U\ bucardo\ bucardo\ -c\ \
   "SELECT\ tablename,standard_conflict\ FROM\ goat;"
```

Por último, declaramos la sincronización entre el grupo de tablas anterior y dklab2:

[31] http://search.cpan.org/~turnstep/DBD-Pg-2.18.1/Pg.pm
[32] http://web.archiveorange.com/archive/v/aQD0Q90Ox68SYJW9RkkF

```
su bucardo -c 'bucardo_ctl add sync jabberd2_swap
                                    source=tablas_jabberd2
                                    targetdb=jabberd2dklab2
                                    type=swap'
su bucardo -c 'bucardo_ctl list syncs'
```

Borramos la copia del TGT que cedimos a postgres:

```
rm -f /tmp/krb5cc_‘id -u postgres‘
```

Activamos la línea de inittab que supervisa a Bucardo, y lanzamos a éste:

```
sed -i -e 's/^#BUC:/BUC:/g' /etc/inittab \
    && kill -HUP 1 && invoke-rc.d bucardo start
```

Podemos preguntarle al proceso por el estado de la instancia de sincronización "jabberd2_swap" que creamos:

```
su bucardo -c 'bucardo_ctl status jabberd2_swap'


su bucardo -c 'bucardo_ctl message "Bucardo + Jabberd2... Done."'
```

1.4.1. Test de la replicación de la DB jabberd2

Si en dklab2 insertamos un registro test en la tabla "active":

```
su jabber -m -c "psql -h dklab2.casafx.dyndns.org -U jabberd2 jabberd2
   -c \INSERT\ INTO\ active\ VALUES\ \(\'test\',1111,1111\)\;"
```

```
   INSERT 0 1
```

Inmediatamente se activa el Trigger y bucardo inicia la sincronización:

```
[20:28:22]  MCP Got notice "bucardo_kick_sync_jabberd2_swap"
            from 2415 on database jabberd2dklab2
[20:28:22]  CTL Got notice "bucardo_ctl_kick_jabberd2_swap" from 3380
[20:28:22]  MCP Sent a kick request to controller 3399
            for sync "jabberd2_swap"
[20:28:23]  KID Got a notice for jabberd2_swap:
            jabberd2dklab1 -> jabberd2dklab2
[20:28:23]  KID Source delta count for public."motd-message": 0
[20:28:24]  KID Target delta count for public."motd-message": 0
...
[20:28:24]  KID Source delta count for public.active: 0
[20:28:24]  KID Target delta count for public.active: 1
            `-----> un cambio en dklab2 (target) tabla active
...
[20:28:27]  KID Total source delta count: 0
[20:28:27]  KID Total target delta count: 1
[20:28:27]  KID Total delta count: 1
[20:28:27]  KID No conflict, target only for
                public.active."collection-owner": test
[20:28:27]  KID Action summary: 2:1
[20:28:27]  KID [1/1] public.active INSERT target to source pk test
[20:28:27]  KID Updating bucardo_track for public.active
            on jabberd2dklab2
[20:28:27]  KID Issuing final commit for source and target
[20:28:28]  KID Marking as done in the q table, notifying controller
[20:28:28]  KID Finished syncing.
            Time: 5.
            Updates: 0+0
            Inserts: 1+0
            Deletes: 0+0
            Sync: jabberd2_swap.
            Keepalive: 1
```

...reporta que ha tardado 5 segundos para hacer una inserción. Es conveniente apuntar que esta prueba se hizo sobre qemu (emulador "full-translation", luego el tiempo puede ser algo mejorable en otro tipo de entornos).

```
[20:28:28]  CTL Got notice
            "bucardo_syncdone_jabberd2_swap_jabberd2dklab2" from 3403
            (kid on database jabberd2dklab2)
[20:28:28]  CTL Sent notice "bucardo_syncdone_jabberd2_swap"
```

En dklab1, efectivamente, aparece el registro test:

```
su jabber -m -c "psql -h dklab1.casafx.dyndns.org -U jabberd2 jabberd2
   -c \SELECT\ *\ FROM\ active\ WHERE\ time=1111\;"
```

```
 collection-owner | object-sequence | time
------------------+-----------------+------
 test             |            1111 | 1111
```

Si ahora borramos la fila pero en dklab1, podremos ver cómo se propaga el cambio ahora en sentido contrario:

```
su jabber -m -c "psql -h dklab1.casafx.dyndns.org -U jabberd2 jabberd2
   -c \DELETE\ FROM\ active\ WHERE\ time=1111\;"
```

```
 DELETE 1
```

```
sleep 5


su jabber -m -c "psql -h dklab2.casafx.dyndns.org -U jabberd2 jabberd2
 -c \SELECT\ *\ FROM\ active\;"
```

1.5. Clientes XMPP

Antes de probar cualquier cliente[33], es conveniente asegurarse de que éstos pueden localizar el servicio utilizando los registros SRV:

```
dig @10.168.1.1 _xmpp-client._tcp.casafx.dyndns.org srv +short
dig @10.168.1.2 _xmpp-client._tcp.casafx.dyndns.org srv +short
```

- Anotación: además de xmpp-client había otro registro, xmpp-server. Éste era para que otros servidores federados con nosotros encuentren nuestros servidores (el protocolo lo permite automáticamente, basta que un usuario de otro dominio tenga en su roster una cuenta de el nuestro). Por tanto, atañe a los servidores, no a los clientes si bien es imprescindible para que usuarios de otros dominios se cominiquen con los nuestros:

```
dig @10.168.1.1 _xmpp-server._tcp.casafx.dyndns.org srv +short
dig @10.168.1.2 _xmpp-server._tcp.casafx.dyndns.org srv +short
```

Existen dos implementaciones típicas de SASL-GSSAPI (Cyrus libsasl2 y GNU libgsasl7; realmente hay alguna más como la de dovecot, no usada aún por otros proyectos), nos aseguramos que los clientes, usen la que usen, las tengan disponibles:

```
apt-get install libsasl2-modules-gssapi-mit libgsasl7
```

...ésto es especialmente importante en el caso de Cyrus, pues el módulo gssapi no se instala como dependencia sino como recomendación o sugerencia. En el caso de GNU es un paquete monolítico y debiera instalarse como dependencia de la aplicación.

1.5.1. Pidgin/Finch

Pidgin[34] es un cliente para diversos protocolos de mensajería instantánea, entre ellos el que nos interesa ahora, XMPP. El soporte de XMPP incluye a Jingle[35], la extensión para VoIP/Videoconferencia P2P que aprovecha XMPP como protocolo de señalización y otros para el stream multimedia (RTP/IAX/...; en el caso de Pidgin es RTP[36], otros como Coccinella optaron por IAX2[37]). Finch es la interfaz en modo texto para Pidgin.

[33] http://xmpp.org/xmpp-software/clients/
[34] http://www.pidgin.im/
[35] http://xmpp.org/extensions/xep-0166.html
Desde junio de 2011 es el estándar escogido por Google para su servicio IM Google talk:
http://xmpp.org/2011/06/the-future-is-jingle
[36] http://developer.pidgin.im/wiki/vv
http://tools.ietf.org/html/rfc3550.html
[37] http://coccinella.im/faq/voip

- Anotación: Efectivamente el programa pidgin es una aplicación con interfaz gráfica. Debe ser ejecutado allí donde la variable display apunte a un servidor gráfico, como por ejemplo en una Xterm o una sesión ssh -X convencional.

```
apt-get install --no-install-recommends pidgin finch \
                                        libsasl2-modules-gssapi-mit
```

```
login umea -p
# -p preserva entorno (excepto algunas variables relacionadas con NSS)
```

```
pidgin -d
```

El flag -d hace que pidgin muestre mensajes de depuración en consola y nos permita saber qué está haciendo. Si todo va bien, la interfaz gráfica muestra un menú de configuración que deberemos operar como se describe:

- Add account.
- Protocol: xmpp
- Username: umea
- Domain: casafx.dyndns.org
- Password: <blank>
- Remember password: <unset>
- Advanced tab:
  - require ssl/tls: <set>#por el problema de gsasl, etc
  - Force old (port 5223): <unset>
  - allow plaintext: <unset>
- +Add.

Debiera conectarse. Si tras conectarse no aparece nadie en el roster, se debe a que pidgin tiene activado ocultar contactos offline. Puede modificarse ésto en su menú View.

De forma similar, podemos hacer login con aemu, lanzar pidgin, configurar su cuenta y comenzar una conversación textual (o audiovisual si disponemos del hardware necesario) entre umea y aemu.

http://tools.ietf.org/html/rfc5456

Podemos utilizar alternativamente la interfaz en modo texto:

```
finch -d 2>/dev/null
```

- Los atajos de teclado de finch[38]:
  - M p para cambiar de "ventana", donde M es la tecla ALT o pulsar y soltar ESC
  - M c para cerrar alguna
  - M a para menú de acciones
  - F10 menú general
  - F11 menú de ventana
  - ...

Trazas

Vamos a aprovechar las trazas que deja pidgin cuando se ejecuta con el flag -d para conocer cómo hace uso de nuestra infraestructura:

- Tras añadir la cuenta intentará conectarse, para lo cual intenta descubrir el servicio a través de los registros SRV. Se le devuelve (aleatoriamente) dklab2 como primera localización. Como esta prueba se realizó con Jabberd2 en dklab2 desactivado, automáticamente pasa a probar con dklab1, sin intervención del usuario:

```
(16:58:33) dnssrv: querying srv record for casafx.dyndns.org:
            _xmpp-client._tcp.casafx.dyndns.org
(16:58:37) dnssrv: found 2 srv entries
(16:58:37) dns: dns query for 'dklab2.casafx.dyndns.org' queued
(16:58:39) proxy: attempting connection to 10.168.1.2
(16:58:39) jabber: unable to connect to server: connection refused.
           trying next srv record or connecting directly.
(16:58:39) dns: dns query for 'dklab1.casafx.dyndns.org' queued
```

Por su lado, en el log del componente c2s (/var/log/jabberd2/c2s.log) se observa:

```
16:58:39 [notice] [11] [10.168.1.1, port=44427] connect
```

Inicia la secuencia del protocolo XMPP:

[38] http://developer.pidgin.im/wiki/Using%20Finch

```
(16:58:40) jabber: sending (ssl) (umea@casafx.dyndns.org/):
           <stream:stream
            to='casafx.dyndns.org'
            xmlns='jabber:client'
            xmlns:stream='http://etherx.jabber.org/streams'
            version='1.0'>
```

El servidor le responde y le presenta los mecanismos para hacer autenticación (gssapi), que acepta.

```
(16:58:41) jabber: recv(ssl) (251):
<?xml version='1.0'?>
<stream:stream xmlns:stream='http://etherx.jabber.org/streams'
xmlns='jabber:client' from='casafx.dyndns.org'
version='1.0' id='x2obx3mnj6l3se2inli5lg08ii99m8yjanv8qiv9'
xmlns:ack='http://www.xmpp.org/extensions/xep-0198.html#ns'>
</stream:stream>

(16:58:42) jabber: recv(ssl) (357):
<stream:features xmlns:stream='http://etherx.jabber.org/streams'>

<address xmlns='http://affinix.com/jabber/address'>
10.168.1.1</address>
<mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
<mechanism>gssapi</mechanism>
</mechanisms>
<auth xmlns='http://jabber.org/features/iq-auth' />
<register xmlns='http://jabber.org/features/iq-register' />
</stream:features>

(16:58:42) sasl: mechs found: gssapi

(16:58:43) jabber: sending (ssl) (umea@casafx.dyndns.org/)
<auth
 xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
 mechanism='gssapi'>
password removed
</auth>
```

Comienza SASL-GSSAPI:

```
(16:58:46) jabber: recv (ssl)(272):
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
yigzbgkqhkig9xibagicag+bitcbhqadagefoqmcaq+iejb4oamc
arkicqrvfs+ez0zzyvkgvjp8f+tkfyulj8synmi3onf9neblcqfu
/t7lergzedczopox0xdcqg9+wozq3syc5cyradcc70g8vebkz5bo
6cycyogycrkce3opitnka2urekbaxopdpfwya0zlf67gxodel3dl
</challenge>
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl' />
 ...

(16:58:46) jabber: recv (ssl)(51):
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

El cliente es aceptado como umea@casafx.dyndns.org/<resource>

```
(16:58:47) jabber: sending (ssl)(umea@casafx.dyndns.org/):
<iq type='set' id='purpleeeed4def'>
<bind xmlns='urn:ietf:params:xml:ns:xmpp-bind' />
</iq>
<iq xmlns='jabber:client' id='purpleeeed4def'
type='result'>
<bind xmln="" s='urn:ietf:params:xml:ns:xmpp-bind'>
<jid>
umea@casafx.dyndns.org/a72486bd6ccf73d1aaf48a6eca87529f7c2d
828d</jid>
</bind>
</iq>
```

En /var/log/jabberd2/sm.log se observa:

```
16:58:47 [notice] session started:
     jid=umea@casafx.dyndns.org/a72486bd6ccf73d1aaf48a6eca87529f7c2d828d
```

Y en /var/log/jabberd2/c2s.log:

```
16:58:47 [notice] [11] sasl authentication succeeded:
     mechanism=gssapi; authzid=umea@casafx.dyndns.org
16:58:47 [notice] [11] bound:
     jid=umea@casafx.dyndns.org/a72486bd6ccf73d1aaf48a6eca87529f7c2d828d
```

El servidor enviará una descripción suya y de aquello que soporta, por ejemplo le dice que es un servidor jabber, que soporta vcard y el envío de roster, o cuál es la dirección de correo asociada al administrador:

```
(16:58:49) jabber: recv (ssl)(1505):
<iq xmlns='jabber:client' id='purpleeeed4df2'
from='casafx.dyndns.org'
to='umea@casafx.dyndns.org/a72486bd6ccf73d1aaf48a6eca87529f7c2d828d'
type='result'>
<query xmlns='ht tp://jabber.org/protocol/disco#info'>
<identity name='jabber im server' type='im'
category='server' />&lt;
feature var='jabber:iq:private'/&gt;
<feature var='presence' />
<feature var='jabber:iq:time' />
<feature var='jabber:iq:agents' />
<feature var='urn:xmpp:ping' />
<feature var='http://jabber.org/protocol/disco#inf o' />
<feature var='http://jabber.org/protocol/disco#items' />
<feature var='http://jabber.org/protocol/v acation' />
<feature var='urn:xmpp:time' />
<feature var='urn:xmpp:blocking' />
<feature var='vcard-temp' />
<feature var='jabber:iq:version' />
<feature var='message' />
<feature var='msgoffline' />
<feature var='ja bber:iq:privacy' />
<feature var='http://jabber.org/protocol/amp' />
<feature var='jabber:iq:roster' />
<feature var='iq' />
<feature var='jabber:iq:last' />
```

```
<x xmlns='jabber:x:data' type='result'>
<field type='hidden' var='form_type'>
<value>urn:xmpp:dataforms:softwareinfo</value>
</field>
<field var='software'>
<value>jabberd2</value>
</field>
<field var='software_version'>
<value>2.2.9</value>
</field>
<field var='os'>
<value>linux</value>
</field>
<field var='os_version'>
<value>i686</value>
</field>
</x>
<x xmlns='jabber:x:data' type='result'>
<field type='hidden' var='form_type'>
<value>http://jabber.org/network/serverinfo</value>
</field>
<field var='admin-addresses'>
<value>mailto:jabbermaster@casafx.dyndns.org</value>
</field>
</x></query>
</iq>
```

El cliente pide su vcard y su roster:

```
(16:58:49) jabber: sending
      (umea@casafx.dyndns.org/a72486bd6ccf73d1aaf48a6eca87529f7c2d828d):
<iq type='get' id='purpleeeed4df3'><vcard xmlns='vcard-temp'/></iq>


(16:58:49) jabber: sending
      (umea@casafx.dyndns.org/a72486bd6ccf73d1aaf48a6eca87529f7c2d828d):
<iq type='get' id='purpleeeed4df4'>
<query xmlns='jabber:iq:roster' />
</iq>
```

Ante desconexiones, una nueva consulta se hace al registro SRV, con la posibilidad de cambio a otro c2s si el anterior ya no está listado ahí o es devuelto en otro orden.

```
(19:49:36) connection: connection error on 0x8bb1768
           (reason: 0 description: server closed the connection)
(19:49:38) account: disconnecting account umea@casafx.dyndns.org/
           (0x88c6a20)
...
(19:50:06) autorecon: calling purple_account_connect
(19:50:06) account: Connecting to account umea@casafx.dyndns.org/.
...
(19:50:06) dnssrv: querying SRV record for casafx.dyndns.org:
           _xmpp-client._tcp.casafx.dyndns.org
(19:50:06) autorecon: done calling purple_account_connect
(19:50:06) dnssrv: found 2 SRV entries
(19:50:06) dns: DNS query for 'dklab2.casafx.dyndns.org' queued
```

Por último, al autenticarse exitosamente con los componentes c2s utilizando SASL-GSSAPI, dos nuevos tickets de servicio fueron conseguidos en el proceso por parte de pidgin (en concreto la librería Cyrus libsasl2 con la que se enlaza utilizó su módulo GSS-API libgssapiv2 para contactar con el kerberos KDC y conseguir esos ticket de servicio xmpp):

```
klist
```

```
...
xmpp/dklab1.casafx.dyndns.org@CASAFX.DYNDNS.ORG
xmpp/dklab2.casafx.dyndns.org@CASAFX.DYNDNS.ORG
```

- Pidgin (lanzado con -d) muestra en la consola los mensajes jabber desencriptados, sin embargo en su transmisión lo hacen encriptados en virtud del establecimiento de una capa de seguridad TLS o (si estuviese implementado completamente el protocolo en GNU Gsasl que utiliza Jabberd2), simplemente con SASL-GSSAPI[39]. Durante una conversación, puede usarse tshark para comprobarlo:

```
tshark -v -i ovpnCASAFX-SRV -d tcp.port==5222,jabber -R tcp.port==5222
```

```
...
Jabber XML Messaging
  eXtensible Markup Language
      [truncated] \027\003\001\000\240\332\362S\253\307\217\317\220=\237\322\235\213\0
\313\322\261W\303\370\370\222A\000|n\252\340\005\356\347\215\340\342\034\030A\241\024\t\
{\256\003L\316\232;\\253\337\344\243\360\006\
      [ ERROR: Unrecognized text ]
```

...sin TLS, por las deficiencias de Gsasl utilizada por Jabberd2, veríamos, en cambio:

```
Jabber XML Messaging
  eXtensible Markup Language
      <message>
          type='chat'
          id='purple5564abef'
          to='umea@casafx.dyndns.org'
          <active/>
              xmlns='http://jabber.org/protocol/chatstates'
          <body>
              HOLA
              </body>
          </message>
```

[39] http://tools.ietf.org/html/rfc4752#section-3.2

1.5.2. Otros clientes que soportan autenticación SASL-GSSAPI

Sólo Gajim parece soportar correctamente SASL-GSSAPI.

Coccinella (bug)

Era nuestra primera elección porque tiene soporte para pizarra interactiva además de VoIP a través de Jingle/IAX2, sin embargo un bug le impedió hacer SASL-GSSAPI. También tiene otras debilidades: no está preempaquetado para Debian y está escrito en Tk (lenguaje interpretado que puede hacerlo algo más lento, si bien es completamente multiplataforma).

Gajim

Gajim es un cliente gráfico multiplataforma (si bien el soporte para MAC es incipiente). Está escrito en python luego no se puede considerar ligero si bien funciona. Las últimas versiones parecen soportar VoIP (al menos en Linux):

- http://www.gajim.org/downloads.php?lang=es

Psi (bug)

Aneriormente (en Debian 6, donde no existía el paquete libqca2-plugin-cyrus-sasl pero se podía compilar desde las ramas de desarrollo, permitiendo entonces autenticación SASL-GSSAPI para Psi) no se producía el log-in y traceando la aplicación, parecía existir un bug. No conocemos el estado de este problema actualmente, pero el mensaje de error era por entonces:

Server krbtgt/DYNDNS.ORG@CASAFX.DYNDNS.ORG not found in Kerberos database

... la parte "/instance" de ese principal está incompleta cuando tiene 3 niveles, así que falla.

Modo texto (sólo finch)

Aparte de finch, no parece que conocidos y livianos clientes jabber en modo texto tengan soporte aún para autenticación SASL-GSSAPI. Ni centerim, ni freetalk ni el modo jabber.el de emacs.

Web (improbable el soporte de SASL-GSSAPI)

Existen clientes Web/AJAX como Jwchat[40], que encapsulan xmpp en http.

[40] http://blog.jwchat.org/jwchat/features/#supported

Sin embargo jabberd2 no implementa estas extensiones aún (ni BOSH/HTTP-binding[41] ni el obsoleto HTTP-polling[42]). Por ello se requeriría instalar un proxy como Punjab[43] (en python) o [44]Jabberhttpbind(java servlet).

No menos importante es que nuestro modelo de autenticación se basa en SASL-GSSAPI, difícil de integrar en un escenario web[45]. LyX

[41]http://xmpp.org/extensions/xep-0124.html
[42]http://xmpp.org/extensions/xep-0025.html
[43]http://punjab.sourceforge.net/
[44]http://zeank.in-berlin.de/jhb/
[45]http://blob.inf.ed.ac.uk/sxw/2008/09/19/integrating-jabber-web-interfaces-with-cosign-and-other-sso-technologies/