ejabberd 2.0.x

Installation and Operation Guide

October 2, 2008

ejabberd Development Team

Contents

Chapter 1

Introduction

ejabberd is a free and open source instant messaging server written in Erlang[1].

ejabberd is cross-platform, distributed, fault-tolerant, and based on open standards to achieve real-time communication.

ejabberd is designed to be a rock-solid and feature rich XMPP server.

ejabberd is suitable for small deployments, whether they need to be scalable or not, as well as extremely big deployments.

[1] http://www.erlang.org/

1.1 Key Features

`ejabberd` is:

- Cross-platform: `ejabberd` runs under Microsoft Windows and Unix derived systems such as Linux, FreeBSD and NetBSD.
- Distributed: You can run `ejabberd` on a cluster of machines and all of them will serve the same Jabber domain(s). When you need more capacity you can simply add a new cheap node to your cluster. Accordingly, you do not need to buy an expensive high-end machine to support tens of thousands concurrent users.
- Fault-tolerant: You can deploy an `ejabberd` cluster so that all the information required for a properly working service will be replicated permanently on all nodes. This means that if one of the nodes crashes, the others will continue working without disruption. In addition, nodes also can be added or replaced 'on the fly'.
- Administrator Friendly: `ejabberd` is built on top of the Open Source Erlang. As a result you do not need to install an external database, an external web server, amongst others because everything is already included, and ready to run out of the box. Other administrator benefits include:
  - Comprehensive documentation.
  - Straightforward installers for Linux, Mac OS X, and Windows.
  - Web Administration.
  - Shared Roster Groups.
  - Command line administration tool.
  - Can integrate with existing authentication mechanisms.
  - Capability to send announce messages.
- Internationalized: `ejabberd` leads in internationalization. Hence it is very well suited in a globalized world. Related features are:
  - Translated to 24 languages.
  - Support for IDNA[2].
- Open Standards: `ejabberd` is the first Open Source Jabber server claiming to fully comply to the XMPP standard.
  - Fully XMPP compliant.
  - XML-based protocol.
  - Many protocols supported[3].

[2] http://www.ietf.org/rfc/rfc3490.txt
[3] http://www.ejabberd.im/protocols

1.2 Additional Features

Moreover, `ejabberd` comes with a wide range of other state-of-the-art features:

- Modular
  - Load only the modules you want.
  - Extend `ejabberd` with your own custom modules.
- Security
  - SASL and STARTTLS for c2s and s2s connections.
  - STARTTLS and Dialback s2s connections.
  - Web Admin accessible via HTTPS secure access.
- Databases
  - Internal database for fast deployment (Mnesia).
  - Native MySQL support.
  - Native PostgreSQL support.
  - ODBC data storage support.
  - Microsoft SQL Server support.
- Authentication
  - Internal Authentication.
  - PAM, LDAP and ODBC.
  - External Authentication script.
- Others
  - Support for virtual hosting.
  - Compressing XML streams with Stream Compression (XEP-0138[4]).
  - Statistics via Statistics Gathering (XEP-0039[5]).
  - IPv6 support both for c2s and s2s connections.
  - Multi-User Chat[6] module with support for clustering and HTML logging.
  - Users Directory based on users vCards.
  - Publish-Subscribe[7] component with support for Personal Eventing via Pubsub[8].
  - Support for web clients: HTTP Polling[9] and HTTP Binding (BOSH)[10] services.
  - IRC transport.
  - Component support: interface with networks such as AIM, ICQ and MSN installing special tranports.

[4] http://www.xmpp.org/extensions/xep-0138.html
[5] http://www.xmpp.org/extensions/xep-0039.html
[6] http://www.xmpp.org/extensions/xep-0045.html
[7] http://www.xmpp.org/extensions/xep-0060.html
[8] http://www.xmpp.org/extensions/xep-0163.html
[9] http://www.xmpp.org/extensions/xep-0025.html
[10] http://www.xmpp.org/extensions/xep-0206.html

Chapter 2

Installing ejabberd

2.1 Installing ejabberd with Binary Installer

Probably the easiest way to install an ejabberd instant messaging server is using the binary installer published by ProcessOne. The binary installers of released ejabberd versions are available in the ProcessOne ejabberd downloads page: http://www.process-one.net/en/ejabberd/downloads

The installer will deploy and configure a full featured ejabberd server and does not require any extra dependencies.

In *nix systems, remember to set executable the binary installer before starting it. For example:

```
chmod +x ejabberd-2.0.0_1-linux-x86-installer.bin
./ejabberd-2.0.0_1-linux-x86-installer.bin
```

ejabberd can be started manually at any time, or automatically by the operating system at system boot time.

To start and stop ejabberd manually, use the desktop shortcuts created by the installer. If the machine doesn't have a graphical system, use the scripts 'start' and 'stop' in the 'bin' directory where ejabberd is installed.

The Windows installer also adds ejabberd as a system service, and a shortcut to a debug console for experienced administrators. If you want ejabberd to be started automatically at boot time, go to the Windows service settings and set ejabberd to be automatically started. Note that the Windows service is a feature still in development, and for example it doesn't read the file ejabberdctl.cfg.

On a *nix system, if you want ejabberd to be started as daemon at boot time, copy `ejabberd.init` from the 'bin' directory to something like `/etc/init.d/ejabberd` (depending on your distribution) and call `/etc/inid.d/ejabberd start` to start it.

If ejabberd doesn't start correctly in Windows, try to start it using the shortcut in desktop or start menu. If the window shows error 14001, the solution is to install: "Microsoft Visual

C++ 2005 SP1 Redistributable Package". You can download it from www.microsoft.com[1]. Then uninstall `ejabberd` and install it again.

If `ejabberd` doesn't start correctly and a crash dump is generated, there was a severe problem. You can try starting `ejabberd` with the script `bin/live.bat` in Windows, or with the command `bin/ejabberdctl live` in other Operating Systems. This way you see the error message provided by Erlang and can identify what is exactly the problem.

The `ejabberdctl` administration script is included in the `bin` directory. Please refer to the section 4.1 for details about `ejabberdctl`, and configurable options to fine tune the Erlang runtime system.

2.2 Installing ejabberd with Operating System specific packages

Some Operating Systems provide a specific `ejabberd` package adapted to the system architecture and libraries. It usually also checks dependencies and performs basic configuration tasks like creating the initial administrator account. Some examples are Debian and Gentoo. Consult the resources provided by your Operating System for more information.

Usually those packages create a script like `/etc/init.d/ejabberd` to start and stop `ejabberd` as a service at boot time.

2.3 Installing ejabberd with CEAN

CEAN[2] (Comprehensive Erlang Archive Network) is a repository that hosts binary packages from many Erlang programs, including `ejabberd` and all its dependencies. The binaries are available for many different system architectures, so this is an alternative to the binary installer and Operating System's `ejabberd` packages.

You will have to create your own `ejabberd` start script depending of how you handle your CEAN installation. The default `ejabberdctl` script is located into `ejabberd`'s priv directory and can be used as an example.

2.4 Installing ejabberd from Source Code

The canonical form for distribution of `ejabberd` stable releases is the source code package. Compiling `ejabberd` from source code is quite easy in *nix systems, as long as your system have all the dependencies.

[1] http://www.microsoft.com/
[2] http://cean.process-one.net/

2.4.1 Requirements

To compile `ejabberd` on a 'Unix-like' operating system, you need:

- GNU Make
- GCC
- Libexpat 1.95 or higher
- Erlang/OTP R10B-9 or newer.
- OpenSSL 0.9.6 or higher, for STARTTLS, SASL and SSL encryption. Optional, highly recommended.
- Zlib 1.2.3 or higher, for Stream Compression support (XEP-0138[3]). Optional.
- GNU Iconv 1.8 or higher, for the IRC Transport (mod_irc). Optional. Not needed on systems with GNU Libc.

2.4.2 Download Source Code

Released versions of `ejabberd` are available in the ProcessOne `ejabberd` downloads page: http://www.process-one.net/

Alternatively, the latest development version can be retrieved from the Subversion repository using this command:

```
svn co http://svn.process-one.net/ejabberd/trunk ejabberd
```

2.4.3 Compile

To compile `ejabberd` execute the commands:

```
./configure
make
```

The build configuration script allows several options. To get the full list run the command:

```
./configure --help
```

Some options that you may be interested in modifying:

`--prefix=/` Specify the path prefix where the files will be copied when running the make install command.

[3] http://www.xmpp.org/extensions/xep-0138.html

`--enable-pam` Enable the PAM authentication method (see section 3.1.4).

`--enable-odbc or --enable-mssql` Required if you want to use an external database. See section 3.2 for more information.

`--enable-full-xml` Enable the use of XML based optimisations. It will for example use CDATA to escape characters in the XMPP stream. Use this option only if you are sure your Jabber clients include a fully compliant XML parser.

`--disable-transient-supervisors` Disable the use of Erlang/OTP supervision for transient processes.

2.4.4 Install

To install `ejabberd` in the destination directories, run the command:

```
make install
```

Note that you probably need administrative privileges in the system to install `ejabberd`.

The files and directories created are, by default:

`/etc/ejabberd/` Configuration files:

- `ejabberd.cfg` ejabberd configuration file
- `ejabberdctl.cfg` Configuration file of the administration script
- `inetrc` Network DNS configuration

`/sbin/ejabberdctl` Administration script (see section 4.1)

`/var/lib/ejabberd/` `.erlang.cookie` Erlang cookie file (see section 5.3)

- `db` Mnesia database spool files
- `ebin` Binary Erlang files (*.beam)
- `priv` `bin` Binary C programs
  - `lib` Binary system libraries (*.so)
  - `msgs` Translated strings (*.msgs)

`/var/log/ejabberd/` Log files (see section 7.2):

- `ejabberd.log` ejabberd service log
- `sasl.log` Erlang/OTP system log

2.4.5 Start

You can use the `ejabberdctl` command line administration script to start and stop `ejabberd`.

Usage example:

```
ejabberdctl start

ejabberdctl status
Node ejabberd@localhost is started. Status: started
ejabberd is running

ejabberdctl stop
```

If `ejabberd` doesn't start correctly and a crash dump is generated, there was a severe problem. You can try starting `ejabberd` with the command `ejabberdctl live` to see the error message provided by Erlang and can identify what is exactly the problem.

Please refer to the section 4.1 for details about `ejabberdctl`, and configurable options to fine tune the Erlang runtime system.

2.4.6 Specific Notes for BSD

The command to compile `ejabberd` in BSD systems is:

```
gmake
```

2.4.7 Specific Notes for Sun Solaris

You need to have `GNU install`, but it isn't included in Solaris. It can be easily installed if your Solaris system is set up for blastwave.org[4] package repository. Make sure `/opt/csw/bin` is in your `PATH` and run:

```
pkg-get -i fileutils
```

If that program is called `ginstall`, modify the `ejabberd Makefile` script to suit your system, for example:

```
cat Makefile | sed s/install/ginstall/ > Makefile.gi
```

And finally install `ejabberd` with:

```
gmake -f Makefile.gi ginstall
```

[4] http://www.blastwave.org/

2.4.8 Specific Notes for Microsoft Windows

Requirements

To compile `ejabberd` on a Microsoft Windows system, you need:

- MS Visual C++ 6.0 Compiler
- Erlang/OTP R11B-5[5]
- Expat 2.0.0 or higher[6]
- GNU Iconv 1.9.2[7] (optional)
- Shining Light OpenSSL 0.9.8d or higher[8] (to enable SSL connections)
- Zlib 1.2.3 or higher[9]

Compilation

We assume that we will try to put as much library as possible into `C:\sdk\` to make it easier to track what is install for `ejabberd`.

1. Install Erlang emulator (for example, into `C:\sdk\erl5.5.5`).
2. Install Expat library into `C:\sdk\Expat-2.0.0` directory.

   Copy file `C:\sdk\Expat-2.0.0\Libs\libexpat.dll` to your Windows system directory (for example, `C:\WINNT` or `C:\WINNT\System32`)
3. Build and install the Iconv library into the directory `C:\sdk\GnuWin32`.

   Copy file `C:\sdk\GnuWin32\bin\lib*.dll` to your Windows system directory (more installation instructions can be found in the file README.woe32 in the iconv distribution).

   Note: instead of copying libexpat.dll and iconv.dll to the Windows directory, you can add the directories `C:\sdk\Expat-2.0.0\Libs` and `C:\sdk\GnuWin32\bin` to the `PATH` environment variable.
4. Install OpenSSL in `C:\sdk\OpenSSL` and add `C:\sdk\OpenSSL\lib\VC` to your path or copy the binaries to your system directory.
5. Install ZLib in `C:\sdk\gnuWin32`. Copy `C:\sdk\GnuWin32\bin\zlib1.dll` to your system directory. If you change your path it should already be set after libiconv install.
6. Make sure the you can access Erlang binaries from your path. For example: `set PATH=%PATH%;"C:\sdk\erl`

[5] http://www.erlang.org/download.html
[6] http://sourceforge.net/project/showfiles.php?group_id=10127&package_id=11277
[7] http://www.gnu.org/software/libiconv/
[8] http://www.slproweb.com/products/Win32OpenSSL.html
[9] http://www.zlib.net/

7. Depending on how you end up actually installing the library you might need to check and tweak the paths in the file configure.erl.
8. While in the directory `ejabberd\src` run:

   ```
   configure.bat
   nmake -f Makefile.win32
   ```

9. Edit the file `ejabberd\src\ejabberd.cfg` and run

   ```
   werl -s ejabberd -name ejabberd
   ```

2.5 Create a Jabber Account for Administration

You need a Jabber account and grant him administrative privileges to enter the `ejabberd` Web Admin:

1. Register a Jabber account on your `ejabberd` server, for example `admin1@example.org`. There are two ways to register a Jabber account:
   (a) Using `ejabberdctl` (see section 4.1):

       ejabberdctl register admin1 example.org FgT5bk3

   (b) Using a Jabber client and In-Band Registration (see section 3.3.15).
2. Edit the `ejabberd` configuration file to give administration rights to the Jabber account you created:

   ```
   {acl, admins, {user, "admin1", "example.org"}}.
   {access, configure, [{allow, admins}]}.
   ```

   You can grant administrative privileges to many Jabber accounts, and also to accounts in other Jabber servers.
3. Restart `ejabberd` to load the new configuration.
4. Open the Web Admin (`http://server:port/admin/`) in your favourite browser. Make sure to enter the *full* JID as username (in this example: `admin1@example.org`. The reason that you also need to enter the suffix, is because `ejabberd`'s virtual hosting support.

2.6 Upgrading `ejabberd`

To upgrade an ejabberd installation to a new version, simply uninstall the old version, and then install the new one. Of course, it is important that the configuration file and Mnesia database spool directory are not removed.

`ejabberd` automatically updates the Mnesia table definitions at startup when needed. If you also use an external database for storage of some modules, check if the release notes of the new ejabberd version indicates you need to also update those tables.

Chapter 3

Configuring ejabberd

3.1 Basic Configuration

The configuration file will be loaded the first time you start ejabberd. The content from this file will be parsed and stored in the internal ejabberd database. Subsequently the configuration will be loaded from the database and any commands in the configuration file are appended to the entries in the database.

Note that ejabberd never edits the configuration file. So, the configuration changes done using the Web Admin are stored in the database, but are not reflected in the configuration file. If you want those changes to be use after ejabberd restart, you can either edit the configuration file, or remove all its content.

The configuration file contains a sequence of Erlang terms. Lines beginning with a '%' sign are ignored. Each term is a tuple of which the first element is the name of an option, and any further elements are that option's values. If the configuration file do not contain for instance the 'hosts' option, the old host name(s) stored in the database will be used.

You can override the old values stored in the database by adding next lines to the configuration file:

```
override_global.
override_local.
override_acls.
```

With these lines the old global options (shared between all ejabberd nodes in a cluster), local options (which are specific for this particular ejabberd node) and ACLs will be removed before new ones are added.

3.1.1 Host Names

The option hosts defines a list containing one or more domains that ejabberd will serve.

Examples:

- Serving one domain:

```
{hosts, ["example.org"]}.
```

- Serving one domain, and backwards compatible with older `ejabberd` versions:

```
{host, "example.org"}.
```

- Serving two domains:

```
{hosts, ["example.net", "example.com"]}.
```

3.1.2 Virtual Hosting

Options can be defined separately for every virtual host using the `host_config` option. It has the following syntax:

```
{host_config, <hostname>, [<option>, <option>, ...]}.
```

Examples:

- Domain `example.net` is using the internal authentication method while domain `example.com` is using the LDAP server running on the domain `localhost` to perform authentication:

```
{host_config, "example.net", [{auth_method,   internal}]}.

{host_config, "example.com", [{auth_method,   ldap},
                              {ldap_servers,  ["localhost"]},
                              {ldap_uids,     [{"uid"}]},
                              {ldap_rootdn,   "dc=localdomain"},
                              {ldap_rootdn,   "dc=example,dc=com"},
                              {ldap_password, ""}]}.
```

- Domain `example.net` is using ODBC to perform authentication while domain `example.com` is using the LDAP servers running on the domains `localhost` and `otherhost`:

```
{host_config, "example.net", [{auth_method, odbc},
                              {odbc_server, "DSN=ejabberd;UID=ejabberd;PWD=ejabberd"}]}.

{host_config, "example.com", [{auth_method,   ldap},
                              {ldap_servers,  ["localhost", "otherhost"]},
                              {ldap_uids,     [{"uid"}]},
                              {ldap_rootdn,   "dc=localdomain"},
                              {ldap_rootdn,   "dc=example,dc=com"},
                              {ldap_password, ""}]}.
```

To define specific ejabberd modules in a virtual host, you can define the global `modules` option with the common modules, and later add specific modules to certain virtual hosts. To accomplish that, instead of defining each option in `host_config` with the syntax

```
{<option-name>, <option-value>}
```

use this syntax:

```
{{add, <option-name>}, <option-value>}
```

In this example three virtual hosts have some similar modules, but there are also other different modules for some specific virtual hosts:

```
%% This ejabberd server has three vhosts:
{hosts, ["one.example.org", "two.example.org", "three.example.org"]}.

%% Configuration of modules that are common to all vhosts
{modules,
 [
  {mod_roster,     []},
  {mod_configure,  []},
  {mod_disco,      []},
  {mod_private,    []},
  {mod_time,       []},
  {mod_last,       []},
  {mod_version,    []}
 ]}.

%% Add some modules to vhost one:
{host_config, "one.example.org",
 [{{add, modules}, [
                    {mod_echo,       [{host, "echo-service.one.example.org"}]}
                    {mod_http_bind,  []},
                    {mod_logxml,     []}
                   ]
  }
 ]}.

%% Add a module just to vhost two:
{host_config, "two.example.org",
 [{{add, modules}, [
                    {mod_echo,       [{host, "mirror.two.example.org"}]}
                   ]
  }
 ]}.
```

3.1.3 Listening Ports

The option `listen` defines for which addresses and ports `ejabberd` will listen and what services will be run on them. Each element of the list is a tuple with the following elements:

- Port number.
- Module that serves this port.
- Options to this module.

The available modules, their purpose and the options allowed by each one are:

`ejabberd_c2s` Handles c2s connections.
Options: `access`, `certfile`, `inet6`, `ip`, `max_stanza_size`, `shaper`, `starttls`, `starttls_required`, `tls`, `zlib`

`ejabberd_s2s_in` Handles incoming s2s connections.
Options: `inet6`, `ip`, `max_stanza_size`

`ejabberd_service` Interacts with external components[1] (as defined in the Jabber Component Protocol (XEP-0114[2]).
Options: `access`, `hosts`, `inet6`, `ip`, `shaper`, `service_check_from`

`ejabberd_http` Handles incoming HTTP connections.
Options: `certfile`, `http_bind`, `http_poll`, `inet6`, `ip`, `request_handlers`, `tls`, `web_admin`

This is a detailed description of each option allowed by the listening modules:

`{access, <access rule>}` This option defines access to the port. The default value is `all`.

`{certfile, Path}` Full path to a file containing the default SSL certificate. To define a certificate file specific for a given domain, use the global option `domain_certfile`.

`service_check_from` This option can be used with `ejabberd_service` only. It is used to disable control on the from field on packets send by an external components. The option can be either `true` or `false`. The default value is `true` which conforms to XEP-0114[3].

`{hosts, [Hostnames], [HostOptions]}` This option of `ejabberd_service` allows to define one or more hostnames of external Jabber components that provide a service. In `HostOptions` it is possible to define the password required to those components when attempt to connect to ejabberd: `{password, Secret}`. Note that you cannot define in a single `ejabberd_service` components of different services: add an `ejabberd_service` for each service, as seen in an example below.

[1] http://www.ejabberd.im/tutorials-transports
[2] http://www.xmpp.org/extensions/xep-0114.html
[3] http://www.xmpp.org/extensions/xep-0114.html

`http_bind` This option enables HTTP Binding (XEP-0124[4] and XEP-0206[5]) support. HTTP Bind enables access via HTTP requests to `ejabberd` from behind firewalls which do not allow outgoing sockets on port 5222.

Remember that you must also install and enable the module mod_http_bind.

If HTTP Bind is enabled, it will be available at `http://server:port/http-bind/`. Be aware that support for HTTP Bind is also needed in the Jabber client. Remark also that HTTP Bind can be interesting to host a web-based Jabber client such as JWChat[6] (check the tutorials to install JWChat with ejabberd and an embedded local web server[7] or Apache[8]).

`http_poll` This option enables HTTP Polling (XEP-0025[9]) support. HTTP Polling enables access via HTTP requests to `ejabberd` from behind firewalls which do not allow outgoing sockets on port 5222.

If HTTP Polling is enabled, it will be available at `http://server:port/http-poll/`. Be aware that support for HTTP Polling is also needed in the Jabber client. Remark also that HTTP Polling can be interesting to host a web-based Jabber client such as JWChat[10].

`inet6` Set up the socket for IPv6 instead of IPv4. Note: this option is not required for S2S outgoing connections, because when ejabberd attempts to establish a S2S outgoing connection it first tries IPv4, and if that fails it attempts with IPv6.

`{ip, IPAddress}` This option specifies which network interface to listen for. For example `{ip, {192, 168, 1, 1}}`.

`{max_stanza_size, Size}` This option specifies an approximate maximum size in bytes of XML stanzas. Approximate, because it is calculated with the precision of one block of readed data. For example `{max_stanza_size, 65536}`. The default value is `infinity`. Recommended values are 65536 for c2s connections and 131072 for s2s connections. s2s max stanza size must always much higher than c2s limit. Change this value with extreme care as it can cause unwanted disconnect if set too low.

`{request_handlers, [{Path, Module}]}` To define one or several handlers that will serve HTTP requests. The Path is a list of strings; so the URIs that start with that Path will be served by Module. For example, if you want `mod_foo` to serve the URIs that start with `/a/b/`, and you also want `mod_http_bind` to serve the URIs `/http-bind/`, use this option: `{request_handlers, [{["a", "b"], mod_foo}, {["http-bind"], mod_http_bind}]}`

`{service_check_from, true|false}` By enabling this option, `ejabberd` allows the component to send packets with any arbitrary domain in the 'from' attribute. Note that XEP-0114[11] requires that the domain must match the hostname of the component. Only enable this option if you are completely sure you need to enable it. Default value: false.

`{shaper, <access rule>}` This option defines a shaper for the port (see section 3.1.6). The default value is `none`.

[4] http://www.xmpp.org/extensions/xep-0124.html
[5] http://www.xmpp.org/extensions/xep-0206.html
[6] http://jwchat.sourceforge.net/
[7] http://www.ejabberd.im/jwchat-localserver
[8] http://www.ejabberd.im/jwchat-apache
[9] http://www.xmpp.org/extensions/xep-0025.html
[10] http://jwchat.sourceforge.net/
[11] http://www.xmpp.org/extensions/xep-0114.html

`starttls` This option specifies that STARTTLS encryption is available on connections to the port. You should also set the `certfile` option. You can define a certificate file for a specific domain using the global option `domain_certfile`.

`starttls_required` This option specifies that STARTTLS encryption is required on connections to the port. No unencrypted connections will be allowed. You should also set the `certfile` option. You can define a certificate file for a specific domain using the global option `domain_certfile`.

`tls` This option specifies that traffic on the port will be encrypted using SSL immediately after connecting. You should also set the `certfile` option.

`web_admin` This option enables the Web Admin for `ejabberd` administration which is available at `http://server:port/admin/`. Login and password are the username and password of one of the registered users who are granted access by the 'configure' access rule.

`zlib` This option specifies that Zlib stream compression (as defined in XEP-0138[12]) is available on connections to the port. Client connections cannot use stream compression and stream encryption simultaneously. Hence, if you specify both `tls` (or `ssl`) and `zlib`, the latter option will not affect connections (there will be no stream compression).

There are some additional global options:

`{s2s_use_starttls, true|false}` This option defines whether to use STARTTLS for s2s connections.

`{s2s_certfile, Path}` Full path to a file containing a SSL certificate.

`{domain_certfile, Domain, Path}` Full path to the file containing the SSL certificate for a specific domain.

`{s2s_default_policy, allow|deny}` The default policy for incoming and outgoing s2s connections to other Jabber servers. The default value is `allow`.

`{{s2s_host, Host}, allow|deny}` Defines if incoming and outgoing s2s connections with a specific remote host are allowed or denied. This allows to restrict ejabberd to only establish s2s connections with a small list of trusted servers, or to block some specific servers.

`{s2s_max_retry_delay, Seconds}` The maximum allowed delay for retry to connect after a failed connection attempt. Specified in seconds. The default value is 300 seconds (5 minutes).

For example, the following simple configuration defines:

- There are three domains. The default certificate file is `server.pem`. However, the c2s and s2s connections to the domain `example.com` use the file `example_com.pem`.
- Port 5222 listens for c2s connections with STARTTLS, and also allows plain connections for old clients.

[12] http://www.xmpp.org/extensions/xep-0138.html

- Port 5223 listens for c2s connections with the old SSL.
- Port 5269 listens for s2s connections with STARTTLS.
- Port 5280 listens for HTTP requests, and serves the HTTP Poll service.
- Port 5281 listens for HTTP requests, and serves the Web Admin using HTTPS as explained in section 4.2.

```
{hosts, ["example.com", "example.org", "example.net"]}.
{listen,
 [
  {5222, ejabberd_c2s, [
                        {access, c2s},
                        {shaper, c2s_shaper},
                        starttls, {certfile, "/etc/ejabberd/server.pem"},
                        {max_stanza_size, 65536}
                       ]},
  {5223, ejabberd_c2s, [
                        {access, c2s},
                        {shaper, c2s_shaper},
                        tls, {certfile, "/etc/ejabberd/server.pem"},
                        {max_stanza_size, 65536}
                       ]},
  {5269, ejabberd_s2s_in, [
                           {shaper, s2s_shaper},
                           {max_stanza_size, 131072}
                          ]},
  {5280, ejabberd_http, [
                         http_poll
                        ]},
  {5281, ejabberd_http, [
                         web_admin,
                         tls, {certfile, "/etc/ejabberd/server.pem"},
                        ]}
 ]
}.
{s2s_use_starttls, true}.
{s2s_certfile, "/etc/ejabberd/server.pem"}.
{domain_certfile, "example.com", "/etc/ejabberd/example_com.pem"}.
```

In this example, the following configuration defines that:

- c2s connections are listened for on port 5222 and 5223 (SSL) and denied for the user called ‘bad’.
- s2s connections are listened for on port 5269 with STARTTLS for secured traffic enabled. Incoming and outgoing connections of remote Jabber servers are denied, only two servers can connect: ”jabber.example.org” and ”example.com”.

- Port 5280 is serving the Web Admin and the HTTP Polling service. Note that it is also possible to serve them on different ports. The second example in section 4.2 shows how exactly this can be done.
- All users except for the administrators have a traffic of limit 1,000 Bytes/second
- The AIM transport[13] `aim.example.org` is connected to port 5233 with password '`aimsecret`'.
- The ICQ transport JIT (`icq.example.org` and `sms.example.org`) is connected to port 5234 with password '`jitsecret`'.
- The MSN transport[14] `msn.example.org` is connected to port 5235 with password '`msnsecret`'.
- The Yahoo! transport[15] `yahoo.example.org` is connected to port 5236 with password '`yahoosecret`'.
- The Gadu-Gadu transport[16] `gg.example.org` is connected to port 5237 with password '`ggsecret`'.
- The Jabber Mail Component[17] `jmc.example.org` is connected to port 5238 with password '`jmcsecret`'.
- The service custom has enabled the special option to avoiding checking the `from` attribute in the packets send by this component. The component can send packets in behalf of any users from the server, or even on behalf of any server.

```
{acl, blocked, {user, "bad"}}.
{access, c2s, [{deny, blocked},
               {allow, all}]}.
{shaper, normal, {maxrate, 1000}}.
{access, c2s_shaper, [{none, admin},
                      {normal, all}]}.
{listen,
 [{5222, ejabberd_c2s,     [{access, c2s}, {shaper, c2s_shaper}]},
  {5223, ejabberd_c2s,     [{access, c2s},
                            ssl, {certfile, "/path/to/ssl.pem"}]},
  {5269, ejabberd_s2s_in,  []},
  {5280, ejabberd_http,    [http_poll, web_admin]},
  {5233, ejabberd_service, [{hosts, ["aim.example.org"],
                             [{password, "aimsecret"}]}]},
  {5234, ejabberd_service, [{hosts, ["icq.example.org", "sms.example.org"],
                             [{password, "jitsecret"}]}]},
  {5235, ejabberd_service, [{hosts, ["msn.example.org"],
                             [{password, "msnsecret"}]}]},
  {5236, ejabberd_service, [{hosts, ["yahoo.example.org"],
                             [{password, "yahoosecret"}]}]},
  {5237, ejabberd_service, [{hosts, ["gg.example.org"],
```

[13] http://www.ejabberd.im/pyaimt
[14] http://www.ejabberd.im/pymsnt
[15] http://www.ejabberd.im/yahoo-transport-2
[16] http://www.ejabberd.im/jabber-gg-transport
[17] http://www.ejabberd.im/jmc

```
                            [{password, "ggsecret"}]}]},
  {5238, ejabberd_service, [{hosts, ["jmc.example.org"],
                            [{password, "jmcsecret"}]}]},
  {5239, ejabberd_service, [{hosts, ["custom.example.org"],
                            [{password, "customsecret"}]},
                           {service_check_from, false}]}
 ]
}.
{s2s_use_starttls, true}.
{s2s_certfile, "/path/to/ssl.pem"}.
{s2s_default_policy, deny}.
{{s2s_host,"jabber.example.org"}, allow}.
{{s2s_host,"example.com"}, allow}.
```

Note, that for jabberd 1.4- or WPJabber-based services you have to make the transports log and do XDB by themselves:

```
  <!--
     You have to add elogger and rlogger entries here when using ejabberd.
     In this case the transport will do the logging.
  -->

  <log id='logger'>
    <host/>
    <logtype/>
    <format>%d: [%t] (%h): %s</format>
    <file>/var/log/jabber/service.log</file>
  </log>

  <!--
     Some Jabber server implementations do not provide
     XDB services (for example, jabberd2 and ejabberd).
     xdb_file.so is loaded in to handle all XDB requests.
  -->

  <xdb id="xdb">
    <host/>
    <load>
      <!-- this is a lib of wpjabber or jabberd -->
      <xdb_file>/usr/lib/jabber/xdb_file.so</xdb_file>
      </load>
    <xdb_file xmlns="jabber:config:xdb_file">
      <spool><jabberd:cmdline flag='s'>/var/spool/jabber</jabberd:cmdline></spool>
    </xdb_file>
  </xdb>
```

3.1.4 Authentication

The option `auth_method` defines the authentication method that is used for user authentication:

```
{auth_method, [<method>]}.
```

The following authentication methods are supported by `ejabberd`:

- internal (default) — See section 3.1.4.
- external — There are some example authentication scripts[18].
- ldap — See section 3.2.5.
- odbc — See section 3.2.1, 3.2.3, 3.2.2 and 3.2.4.
- anonymous — See section 3.1.4.
- pam — See section 3.1.4.

Internal

`ejabberd` uses its internal Mnesia database as the default authentication method.

- `auth_method`: The value `internal` will enable the internal authentication method.

Examples:

- To use internal authentication on `example.org` and LDAP authentication on `example.net`:

  ```
  {host_config, "example.org", [{auth_method, [internal]}]}.
  {host_config, "example.net", [{auth_method, [ldap]}]}.
  ```

- To use internal authentication on all virtual hosts:

  ```
  {auth_method, internal}.
  ```

[18] http://www.ejabberd.im/extauth

SASL Anonymous and Anonymous Login

The anonymous authentication method can be configured with the following options. Remember that you can use the `host_config` option to set virtual host specific options (see section 3.1.2). Note that there also is a detailed tutorial regarding SASL Anonymous and anonymous login configuration[19].

- `auth_method`: The value `anonymous` will enable the anonymous authentication method.
- `allow_multiple_connections`: This value for this option can be either `true` or `false` and is only used when the anonymous mode is enabled. Setting it to `true` means that the same username can be taken multiple times in anonymous login mode if different resource are used to connect. This option is only useful in very special occasions. The default value is `false`.
- `anonymous_protocol`: This option can take three values: `sasl_anon`, `login_anon` or `both`. `sasl_anon` means that the SASL Anonymous method will be used. `login_anon` means that the anonymous login method will be used. `both` means that SASL Anonymous and login anonymous are both enabled.

Those options are defined for each virtual host with the `host_config` parameter (see section 3.1.2).

Examples:

- To enable anonymous login on all virtual hosts:

  ```
  {auth_method, [anonymous]}.
  {anonymous_protocol, login_anon}.
  ```

- Similar as previous example, but limited to `public.example.org`:

  ```
  {host_config, "public.example.org", [{auth_method, [anonymous]},
                                       {anonymous_protocol, login_anon}]}.
  ```

- To enable anonymous login and internal authentication on a virtual host:

  ```
  {host_config, "public.example.org", [{auth_method, [internal,anonymous]},
                                       {anonymous_protocol, login_anon}]}.
  ```

- To enable SASL Anonymous on a virtual host:

  ```
  {host_config, "public.example.org", [{auth_method, [anonymous]},
                                       {anonymous_protocol, sasl_anon}]}.
  ```

- To enable SASL Anonymous and anonymous login on a virtual host:

[19] http://support.process-one.net/doc/display/MESSENGER/Anonymous+users+support

```
{host_config, "public.example.org", [{auth_method, [anonymous]},
                                     {anonymous_protocol, both}]}.
```

- To enable SASL Anonymous, anonymous login, and internal authentication on a virtual host:

```
{host_config, "public.example.org", [{auth_method, [internal,anonymous]},
                                     {anonymous_protocol, both}]}.
```

PAM Authentication

`ejabberd` supports authentication via Pluggable Authentication Modules (PAM). PAM is currently supported in AIX, FreeBSD, HP-UX, Linux, Mac OS X, NetBSD and Solaris. PAM authentication is disabled by default, so you have to configure and compile `ejabberd` with PAM support enabled:

```
./configure --enable-pam && make install
```

Options:

`pam_service` This option defines the PAM service name. Default is `"ejabberd"`. Refer to the PAM documentation of your operation system for more information.

Example:

```
{auth_method, [pam]}.
{pam_service, "ejabberd"}.
```

Though it is quite easy to set up PAM support in `ejabberd`, PAM itself introduces some security issues:

- To perform PAM authentication `ejabberd` uses external C-program called `epam`. By default, it is located in `/var/lib/ejabberd/priv/bin/` directory. You have to set it root on execution in the case when your PAM module requires root privileges (`pam_unix.so` for example). Also you have to grant access for `ejabberd` to this file and remove all other permissions from it. Execute with root privileges:

  ```
  chown root:ejabberd /var/lib/ejabberd/priv/bin/epam
  chmod 4750 /var/lib/ejabberd/priv/bin/epam
  ```

- Make sure you have the latest version of PAM installed on your system. Some old versions of PAM modules cause memory leaks. If you are not able to use the latest version, you can `kill(1) epam` process periodically to reduce its memory consumption: `ejabberd` will restart this process immediately.

- `epam` program tries to turn off delays on authentication failures. However, some PAM modules ignore this behavior and rely on their own configuration options. You can create a configuration file `ejabberd.pam`. This example shows how to turn off delays in `pam_unix.so` module:

  ```
  #%PAM-1.0
  auth        sufficient  pam_unix.so likeauth nullok nodelay
  account     sufficient  pam_unix.so
  ```

  That is not a ready to use configuration file: you must use it as a hint when building your own PAM configuration instead. Note that if you want to disable delays on authentication failures in the PAM configuration file, you have to restrict access to this file, so a malicious user can't use your configuration to perform brute-force attacks.

- You may want to allow login access only for certain users. `pam_listfile.so` module provides such functionality.

3.1.5 Access Rules

ACL Definition

Access control in `ejabberd` is performed via Access Control Lists (ACLs). The declarations of ACLs in the configuration file have the following syntax:

```
{acl, <aclname>, {<acltype>, ...}}.
```

`<acltype>` can be one of the following:

`all` Matches all JIDs. Example:

```
{acl, all, all}.
```

`{user, <username>}` Matches the user with the name `<username>` at the first virtual host. Example:

```
{acl, admin, {user, "yozhik"}}.
```

`{user, <username>, <server>}` Matches the user with the JID `<username>@<server>` and any resource. Example:

```
{acl, admin, {user, "yozhik", "example.org"}}.
```

`{server, <server>}` Matches any JID from server `<server>`. Example:

```
{acl, exampleorg, {server, "example.org"}}.
```

`{user_regexp, <regexp>}` Matches any local user with a name that matches `<regexp>` on local virtual hosts. Example:

```
{acl, tests, {user_regexp, "^test[0-9]*$"}}.
```

{user_regexp, <regexp>, <server>} Matches any user with a name that matches <regexp> at server <server>. Example:

```
{acl, tests, {user_regexp, "^test", "example.org"}}.
```

{server_regexp, <regexp>} Matches any JID from the server that matches <regexp>. Example:

```
{acl, icq, {server_regexp, "^icq\\."}}.
```

{node_regexp, <user_regexp>, <server_regexp>} Matches any user with a name that matches <user_regexp> at any server that matches <server_regexp>. Example:

```
{acl, yohzik, {node_regexp, "^yohzik$", "^example.(com|org)$"}}.
```

{user_glob, <glob>}

{user_glob, <glob>, <server>}

{server_glob, <glob>}

{node_glob, <user_glob>, <server_glob>} This is the same as above. However, it uses shell glob patterns instead of regexp. These patterns can have the following special characters:

- * matches any string including the null string.
- ? matches any single character.
- [...] matches any of the enclosed characters. Character ranges are specified by a pair of characters separated by a '-'. If the first character after '[' is a '!', any character not enclosed is matched.

The following ACLs are pre-defined:

all Matches any JID.

none Matches no JID.

Access Rights

An entry allowing or denying access to different services looks similar to this:

```
{access, <accessname>, [{allow, <aclname>},
                         {deny, <aclname>},
                         ...
                        ]}.
```

When a JID is checked to have access to <accessname>, the server sequentially checks if that JID matches any of the ACLs that are named in the second elements of the tuples in the list. If it matches, the first element of the first matched tuple is returned, otherwise the value 'deny' is returned.

Example:

```
{access, configure, [{allow, admin}]}.
{access, something, [{deny, badmans},
                     {allow, all}]}.
```

The following access rules are pre-defined:

all Always returns the value 'allow'.

none Always returns the value 'deny'.

Limiting Opened Sessions with ACL

The special access max_user_sessions specifies the maximum number of sessions (authenticated connections) per user. If a user tries to open more sessions by using different resources, the first opened session will be disconnected. The error session replaced will be sent to the disconnected session. The value for this option can be either a number, or infinity. The default value is infinity.

The syntax is:

```
{access, max_user_sessions, [{<maxnumber>, <aclname>},
                              ...
                             ]}.
```

Examples:

- To limit the number of sessions per user to 10 for all users:

  ```
  {access, max_user_sessions, [{10, all}]}.
  ```

Several connections to a remote Jabber server with ACL

The special access max_s2s_connections specifies how many simultaneus S2S connections can be established to a specific remote Jabber server. The default value is 1. There's also available the access max_s2s_connections_per_node.

The syntax is:

```
{access, max_s2s_connections, [{<maxnumber>, <aclname>},
                                ...
                               ]}.
```

Examples:

- Allow up to 3 connections with each remote server:

  ```
  {access, max_s2s_connections, [{3, all}]}.
  ```

3.1.6 Shapers

Shapers enable you to limit connection traffic. The syntax of shapers is like this:

```
{shaper, <shapername>, <kind>}.
```

Currently only one kind of shaper called `maxrate` is available. It has the following syntax:

```
{maxrate, <rate>}
```

where `<rate>` stands for the maximum allowed incoming rate in bytes per second. When a connection exceeds this limit, `ejabberd` stops reading from the socket until the average rate is again below the allowed maximum.

Examples:

- To define a shaper named '`normal`' with traffic speed limited to 1,000 bytes/second:

  ```
  {shaper, normal, {maxrate, 1000}}.
  ```

- To define a shaper named '`fast`' with traffic speed limited to 50,000 bytes/second:

  ```
  {shaper, fast, {maxrate, 50000}}.
  ```

3.1.7 Default Language

The option `language` defines the default language of server strings that can be seen by Jabber clients. If a Jabber client do not support `xml:lang`, the specified language is used. The default value is `en`. In order to take effect there must be a translation file `<language>.msg` in `ejabberd`'s `msgs` directory.

Examples:

- To set Russian as default language:

  ```
  {language, "ru"}.
  ```

- To set Spanish as default language:

  ```
  {language, "es"}.
  ```

3.2 Database and LDAP Configuration

`ejabberd` uses its internal Mnesia database by default. However, it is possible to use a relational database or an LDAP server to store persistent, long-living data. `ejabberd` is very flexible: you can configure different authentication methods for different virtual hosts, you can configure different authentication mechanisms for the same virtual host (fallback), you can set different storage systems for modules, and so forth.

The following databases are supported by `ejabberd`:

- Microsoft SQL Server[20]
- Mnesia[21]
- MySQL[22]
- Any ODBC compatible database[23]
- PostgreSQL[24]

The following LDAP servers are tested with `ejabberd`:

- Active Directory[25] (see section 3.2.5)
- OpenLDAP[26]
- Normally any LDAP compatible server should work; inform us about your success with a not-listed server so that we can list it here.

3.2.1 MySQL

Although this section will describe `ejabberd`'s configuration when you want to use the native MySQL driver, it does not describe MySQL's installation and database creation. Check the MySQL documentation and the tutorial Using ejabberd with MySQL native driver[27] for information regarding these topics. Note that the tutorial contains information about `ejabberd`'s configuration which is duplicate to this section.

Moreover, the file mysql.sql in the directory src/odbc might be interesting for you. This file contains the `ejabberd` schema for MySQL. At the end of the file you can find information to update your database schema.

By default `ejabberd` opens 10 connections to the database for each virtual host. Use this option to modify the value:

[20] http://www.microsoft.com/sql/
[21] http://www.erlang.org/doc/apps/mnesia/index.html
[22] http://www.mysql.com/
[23] http://en.wikipedia.org/wiki/Open_Database_Connectivity
[24] http://www.postgresql.org/
[25] http://www.microsoft.com/activedirectory/
[26] http://www.openldap.org/
[27] http://support.process-one.net/doc/display/MESSENGER/Using+ejabberd+with+MySQL+native+driver

```
{odbc_pool_size, 10}.
```

You can configure an interval to make a dummy SQL request to keep alive the connections to the database. The default value is 'undefined', so no keepalive requests are made. Specify in seconds: for example 28800 means 8 hours.

```
{odbc_keepalive_interval, undefined}.
```

Driver Compilation

You can skip this step if you installed `ejabberd` using a binary installer or if the binary packages of `ejabberd` you are using include support for MySQL.

1. First, install the Erlang MySQL library[28]. Make sure the compiled files are in your Erlang path; you can put them for example in the same directory as your `ejabberd` .beam files.
2. Then, configure and install `ejabberd` with ODBC support enabled (this is also needed for native MySQL support!). This can be done, by using next commands:

   ```
   ./configure --enable-odbc && make install
   ```

Authentication

The option value name may be misleading, as the `auth_method` name is used for access to a relational database through ODBC, as well as through the native MySQL interface. Anyway, the first configuration step is to define the odbc `auth_method`. For example:

```
{host_config, "public.example.org", [{auth_method, [odbc]}]}.
```

The actual database access is defined in the option `odbc_server`. Its value is used to define if we want to use ODBC, or one of the two native interface available, PostgreSQL or MySQL.

To use the native MySQL interface, you can pass a tuple of the following form as parameter:

```
{mysql, "Server", "Database", "Username", "Password"}
```

`mysql` is a keyword that should be kept as is. For example:

```
{odbc_server, {mysql, "localhost", "test", "root", "password"}}.
```

Optionally, it is possible to define the MySQL port to use. This option is only useful, in very rare cases, when you are not running MySQL with the default port setting. The `mysql` parameter can thus take the following form:

[28] http://support.process-one.net/doc/display/CONTRIBS/Yxa

```
{mysql, "Server", Port, "Database", "Username", "Password"}
```

The Port value should be an integer, without quotes. For example:

```
{odbc_server, {mysql, "localhost", Port, "test", "root", "password"}}.
```

Storage

MySQL also can be used to store information into from several ejabberd modules. See section 3.3.1 to see which modules have a version with the '_odbc'. This suffix indicates that the module can be used with relational databases like MySQL. To enable storage to your database, just make sure that your database is running well (see previous sections), and replace the suffix-less or ldap module variant with the odbc module variant. Keep in mind that you cannot have several variants of the same module loaded!

3.2.2 Microsoft SQL Server

Although this section will describe ejabberd's configuration when you want to use Microsoft SQL Server, it does not describe Microsoft SQL Server's installation and database creation. Check the MySQL documentation and the tutorial Using ejabberd with MySQL native driver[29] for information regarding these topics. Note that the tutorial contains information about ejabberd's configuration which is duplicate to this section.

Moreover, the file mssql.sql in the directory src/odbc might be interesting for you. This file contains the ejabberd schema for Microsoft SQL Server. At the end of the file you can find information to update your database schema.

By default ejabberd opens 10 connections to the database for each virtual host. Use this option to modify the value:

```
{odbc_pool_size, 10}.
```

You can configure an interval to make a dummy SQL request to keep alive the connections to the database. The default value is 'undefined', so no keepalive requests are made. Specify in seconds: for example 28800 means 8 hours.

```
{odbc_keepalive_interval, undefined}.
```

[29] http://support.process-one.net/doc/display/MESSENGER/Using+ejabberd+with+MySQL+native+driver

Driver Compilation

You can skip this step if you installed ejabberd using a binary installer or if the binary packages of ejabberd you are using include support for ODBC.

If you want to use Microsoft SQL Server with ODBC, you need to configure, compile and install ejabberd with support for ODBC and Microsoft SQL Server enabled. This can be done, by using next commands:

```
./configure --enable-odbc --enable-mssql && make install
```

Authentication

The configuration of Microsoft SQL Server is the same as the configuration of ODBC compatible servers (see section 3.2.4).

Storage

Microsoft SQL Server also can be used to store information into from several ejabberd modules. See section 3.3.1 to see which modules have a version with the '_odbc'. This suffix indicates that the module can be used with relational databases like Microsoft SQL Server. To enable storage to your database, just make sure that your database is running well (see previous sections), and replace the suffix-less or ldap module variant with the odbc module variant. Keep in mind that you cannot have several variants of the same module loaded!

3.2.3 PostgreSQL

Although this section will describe ejabberd's configuration when you want to use the native PostgreSQL driver, it does not describe PostgreSQL's installation and database creation. Check the PostgreSQL documentation and the tutorial Using ejabberd with MySQL native driver[30] for information regarding these topics. Note that the tutorial contains information about ejabberd's configuration which is duplicate to this section.

Also the file pg.sql in the directory src/odbc might be interesting for you. This file contains the ejabberd schema for PostgreSQL. At the end of the file you can find information to update your database schema.

By default ejabberd opens 10 connections to the database for each virtual host. Use this option to modify the value:

```
{odbc_pool_size, 10}.
```

[30] http://support.process-one.net/doc/display/MESSENGER/Using+ejabberd+with+MySQL+native+driver

You can configure an interval to make a dummy SQL request to keep alive the connections to the database. The default value is 'undefined', so no keepalive requests are made. Specify in seconds: for example 28800 means 8 hours.

```
{odbc_keepalive_interval, undefined}.
```

Driver Compilation

You can skip this step if you installed `ejabberd` using a binary installer or if the binary packages of `ejabberd` you are using include support for PostgreSQL.

1. First, install the Erlang pgsql library from ejabberd-modules SVN repository[31]. Make sure the compiled files are in your Erlang path; you can put them for example in the same directory as your `ejabberd` .beam files.

2. Then, configure, compile and install `ejabberd` with ODBC support enabled (this is also needed for native PostgreSQL support!). This can be done, by using next commands:

   ```
   ./configure --enable-odbc && make install
   ```

Authentication

The option value name may be misleading, as the `auth_method` name is used for access to a relational database through ODBC, as well as through the native PostgreSQL interface. Anyway, the first configuration step is to define the odbc `auth_method`. For example:

```
{host_config, "public.example.org", [{auth_method, [odbc]}]}.
```

The actual database access is defined in the option `odbc_server`. Its value is used to define if we want to use ODBC, or one of the two native interface available, PostgreSQL or MySQL.

To use the native PostgreSQL interface, you can pass a tuple of the following form as parameter:

```
{pgsql, "Server", "Database", "Username", "Password"}
```

`pgsql` is a keyword that should be kept as is. For example:

```
{odbc_server, {pgsql, "localhost", "database", "ejabberd", "password"}}.
```

Optionally, it is possible to define the PostgreSQL port to use. This option is only useful, in very rare cases, when you are not running PostgreSQL with the default port setting. The `pgsql` parameter can thus take the following form:

[31] http://www.ejabberd.im/ejabberd-modules/

```
{pgsql, "Server", Port, "Database", "Username", "Password"}
```

The `Port` value should be an integer, without quotes. For example:

```
{odbc_server, {pgsql, "localhost", 5432, "database", "ejabberd", "password"}}.
```

Storage

PostgreSQL also can be used to store information into from several `ejabberd` modules. See section 3.3.1 to see which modules have a version with the '_odbc'. This suffix indicates that the module can be used with relational databases like PostgreSQL. To enable storage to your database, just make sure that your database is running well (see previous sections), and replace the suffix-less or ldap module variant with the odbc module variant. Keep in mind that you cannot have several variants of the same module loaded!

3.2.4 ODBC Compatible

Although this section will describe `ejabberd`'s configuration when you want to use the ODBC driver, it does not describe the installation and database creation of your database. Check the documentation of your database. The tutorial Using ejabberd with MySQL native driver[32] also can help you. Note that the tutorial contains information about `ejabberd`'s configuration which is duplicate to this section.

By default `ejabberd` opens 10 connections to the database for each virtual host. Use this option to modify the value:

```
{odbc_pool_size, 10}.
```

You can configure an interval to make a dummy SQL request to keep alive the connections to the database. The default value is 'undefined', so no keepalive requests are made. Specify in seconds: for example 28800 means 8 hours.

```
{odbc_keepalive_interval, undefined}.
```

Driver Compilation

You can skip this step if you installed `ejabberd` using a binary installer or if the binary packages of `ejabberd` you are using include support for ODBC.

1. First, install the Erlang MySQL library[33]. Make sure the compiled files are in your Erlang path; you can put them for example in the same directory as your `ejabberd` .beam files.

[32] http://support.process-one.net/doc/display/MESSENGER/Using+ejabberd+with+MySQL+native+driver
[33] http://support.process-one.net/doc/display/CONTRIBS/Yxa

2. Then, configure, compile and install `ejabberd` with ODBC support enabled. This can be done, by using next commands:

   ```
   ./configure --enable-odbc && make install
   ```

Authentication

The first configuration step is to define the odbc `auth_method`. For example:

```
{host_config, "public.example.org", [{auth_method, [odbc]}]}.
```

The actual database access is defined in the option `odbc_server`. Its value is used to defined if we want to use ODBC, or one of the two native interface available, PostgreSQL or MySQL.

To use a relational database through ODBC, you can pass the ODBC connection string as `odbc_server` parameter. For example:

```
{odbc_server, "DSN=database;UID=ejabberd;PWD=password"}.
```

Storage

An ODBC compatible database also can be used to store information into from several `ejabberd` modules. See section 3.3.1 to see which modules have a version with the '_odbc'. This suffix indicates that the module can be used with ODBC compatible relational databases. To enable storage to your database, just make sure that your database is running well (see previous sections), and replace the suffix-less or ldap module variant with the odbc module variant. Keep in mind that you cannot have several variants of the same module loaded!

3.2.5 LDAP

`ejabberd` has built-in LDAP support. You can authenticate users against LDAP server and use LDAP directory as vCard storage. Shared rosters are not supported yet.

Note that `ejabberd` treats LDAP as a read-only storage: it is possible to consult data, but not possible to create accounts, change password or edit vCard that is stored in LDAP.

Connection

Parameters:

`ldap_servers` List of IP addresses or DNS names of your LDAP servers. This option is required.

`ldap_port` Port to connect to your LDAP server. The initial default value is 389, so it is used when nothing is set into the configuration file. If you configure a value, it is stored in `ejabberd`'s database. Then, if you remove that value from the configuration file, the value previously stored in the database will be used instead of the default 389.

`ldap_rootdn` Bind DN. The default value is `""` which means 'anonymous connection'.

`ldap_password` Bind password. The default value is `""`.

Example:

```
{auth_method, ldap}.
{ldap_servers, ["ldap.example.org"]}.
{ldap_port, 389}.
{ldap_rootdn, "cn=Manager,dc=domain,dc=org"}.
{ldap_password, "secret"}.
```

Note that current LDAP implementation does not support SSL secured communication and SASL authentication.

Authentication

You can authenticate users against an LDAP directory. Available options are:

`ldap_base` LDAP base directory which stores users accounts. This option is required.

`ldap_uids` LDAP attribute which holds a list of attributes to use as alternatives for getting the JID. The value is of the form: `[{ldap_uidattr}]` or `[{ldap_uidattr, ldap_uidattr_format}]`. You can use as many comma separated tuples `{ldap_uidattr, ldap_uidattr_format}` that is needed. The default value is `[{"uid", "%u"}]`. The defaut `ldap_uidattr_format` is `"%u"`. The values for `ldap_uidattr` and `ldap_uidattr_format` are described as follow:

  `ldap_uidattr` LDAP attribute which holds the user's part of a JID. The default value is `"uid"`.

  `ldap_uidattr_format` Format of the `ldap_uidattr` variable. The format *must* contain one and only one pattern variable `"%u"` which will be replaced by the user's part of a JID. For example, `"%u@example.org"`. The default value is `"%u"`.

`ldap_filter` RFC 2254[34] LDAP filter. The default is `none`. Example: `"(&(objectClass=shadowAccount)(memb` `Users))"`. Please, do not forget to close brackets and do not use superfluous whitespaces. Also you *must not* use `ldap_uidattr` attribute in filter because this attribute will be substituted in LDAP filter automatically.

`ldap_local_filter` If you can't use `ldap_filter` due to performance reasons (the LDAP server has many users registered), you can use this local filter. The local filter checks an attribute in ejabberd, not in LDAP, so this limits the load on the LDAP directory. The default filter is: `undefined`. Example values:

[34] http://www.faqs.org/rfcs/rfc2254.html

```
{ldap_local_filter, {notequal, {"accountStatus",["disabled"]}}}.
{ldap_local_filter, {equal, {"accountStatus",["enabled"]}}}.
{ldap_local_filter, undefined}.
```

Examples

Common example Let's say `ldap.example.org` is the name of our LDAP server. We have users with their passwords in `"ou=Users,dc=example,dc=org"` directory. Also we have addressbook, which contains users emails and their additional infos in `"ou=AddressBook,dc=example,dc=org"` directory. Corresponding authentication section should looks like this:

```
%% Authentication method
{auth_method, ldap}.
%% DNS name of our LDAP server
{ldap_servers, ["ldap.example.org"]}.
%% Bind to LDAP server as "cn=Manager,dc=example,dc=org" with password "secret"
{ldap_rootdn, "cn=Manager,dc=example,dc=org"}.
{ldap_password, "secret"}.
%% Define the user's base
{ldap_base, "ou=Users,dc=example,dc=org"}.
%% We want to authorize users from 'shadowAccount' object class only
{ldap_filter, "(objectClass=shadowAccount)"}.
```

Now we want to use users LDAP-info as their vCards. We have four attributes defined in our LDAP schema: `"mail"` — email address, `"givenName"` — first name, `"sn"` — second name, `"birthDay"` — birthday. Also we want users to search each other. Let's see how we can set it up:

```
{modules,
 [
  ...
  {mod_vcard_ldap,
   [
    %% We use the same server and port, but want to bind anonymously because
    %% our LDAP server accepts anonymous requests to
    %% "ou=AddressBook,dc=example,dc=org" subtree.
    {ldap_rootdn, ""},
    {ldap_password, ""},
    %% define the addressbook's base
    {ldap_base, "ou=AddressBook,dc=example,dc=org"},
    %% uidattr: user's part of JID is located in the "mail" attribute
    %% uidattr_format: common format for our emails
    {ldap_uids, [{"mail", "%u@mail.example.org"}]},
    %% We have to define empty filter here, because entries in addressbook does not
    %% belong to shadowAccount object class
    {ldap_filter, ""},
```

```
    %% Now we want to define vCard pattern
    {ldap_vcard_map,
     [{"NICKNAME", "%u", []}, % just use user's part of JID as his nickname
      {"GIVEN", "%s", ["givenName"]},
      {"FAMILY", "%s", ["sn"]},
      {"FN", "%s, %s", ["sn", "givenName"]}, % example: "Smith, John"
      {"EMAIL", "%s", ["mail"]},
      {"BDAY", "%s", ["birthDay"]}]},
    %% Search form
    {ldap_search_fields,
     [{"User", "%u"},
      {"Name", "givenName"},
      {"Family Name", "sn"},
      {"Email", "mail"},
      {"Birthday", "birthDay"}]},
    %% vCard fields to be reported
    %% Note that JID is always returned with search results
    {ldap_search_reported,
     [{"Full Name", "FN"},
      {"Nickname", "NICKNAME"},
      {"Birthday", "BDAY"}]}
  ]},
  ...
 ]}.
```

Note that mod_vcard_ldap module checks for the existence of the user before searching in his information in LDAP.

Active Directory Active Directory is just an LDAP-server with predefined attributes. A sample configuration is shown below:

```
{auth_method, ldap}.
{ldap_servers, ["office.org"]}.    % List of LDAP servers
{ldap_base, "DC=office,DC=org"}. % Search base of LDAP directory
{ldap_rootdn, "CN=Administrator,CN=Users,DC=office,DC=org"}. % LDAP manager
{ldap_password, "*******"}. % Password to LDAP manager
{ldap_uids, [{"sAMAccountName"}]}.
{ldap_filter, "(memberOf=*)"}.

{modules,
 [
  ...
  {mod_vcard_ldap,
   [{ldap_vcard_map,
     [{"NICKNAME", "%u", []},
      {"GIVEN", "%s", ["givenName"]},
      {"MIDDLE", "%s", ["initials"]},
```

```
      {"FAMILY", "%s", ["sn"]},
      {"FN", "%s", ["displayName"]},
      {"EMAIL", "%s", ["mail"]},
      {"ORGNAME", "%s", ["company"]},
      {"ORGUNIT", "%s", ["department"]},
      {"CTRY", "%s", ["c"]},
      {"LOCALITY", "%s", ["l"]},
      {"STREET", "%s", ["streetAddress"]},
      {"REGION", "%s", ["st"]},
      {"PCODE", "%s", ["postalCode"]},
      {"TITLE", "%s", ["title"]},
      {"URL", "%s", ["wWWHomePage"]},
      {"DESC", "%s", ["description"]},
      {"TEL", "%s", ["telephoneNumber"]}]},
    {ldap_search_fields,
     [{"User", "%u"},
      {"Name", "givenName"},
      {"Family Name", "sn"},
      {"Email", "mail"},
      {"Company", "company"},
      {"Department", "department"},
      {"Role", "title"},
      {"Description", "description"},
      {"Phone", "telephoneNumber"}]},
    {ldap_search_reported,
     [{"Full Name", "FN"},
      {"Nickname", "NICKNAME"},
      {"Email", "EMAIL"}]}
  ]},
  ...
 ]}.
```

3.3 Modules Configuration

The option `modules` defines the list of modules that will be loaded after `ejabberd`'s startup. Each entry in the list is a tuple in which the first element is the name of a module and the second is a list of options for that module.

Examples:

- In this example only the module `mod_echo` is loaded and no module options are specified between the square brackets:

  ```
  {modules,
   [
    {mod_echo,      []}
   ]}.
  ```

- In the second example the modules `mod_echo`, `mod_time`, and `mod_version` are loaded without options. Remark that, besides the last entry, all entries end with a comma:

```
{modules,
 [
  {mod_echo,      []},
  {mod_time,      []},
  {mod_version,   []}
 ]}.
```

3.3.1 Modules Overview

The following table lists all modules included in `ejabberd`.

Module	Feature	Dependencies
mod_adhoc	Ad-Hoc Commands (XEP-0050[35])	
mod_announce	Manage announcements	recommends mod_adhoc
mod_caps	Entity Capabilities (XEP-0115[36])	
mod_configure	Server configuration using Ad-Hoc	mod_adhoc
mod_disco	Service Discovery (XEP-0030[37])	
mod_echo	Echoes Jabber packets	
mod_irc	IRC transport	
mod_last	Last Activity (XEP-0012[38])	
mod_last_odbc	Last Activity (XEP-0012[39])	supported DB (*)
mod_muc	Multi-User Chat (XEP-0045[40])	
mod_muc_log	Multi-User Chat room logging	mod_muc
mod_offline	Offline message storage (XEP-0160[41])	
mod_offline_odbc	Offline message storage (XEP-0160[42])	supported DB (*)
mod_privacy	Blocking Communication (XMPP IM)	
mod_privacy_odbc	Blocking Communication (XMPP IM)	supported DB (*)
mod_private	Private XML Storage (XEP-0049[43])	
mod_private_odbc	Private XML Storage (XEP-0049[44])	supported DB (*)
mod_proxy65	SOCKS5 Bytestreams (XEP-0065[45])	
mod_pubsub	Pub-Sub (XEP-0060[46]), PEP (XEP-0163[47])	mod_caps
mod_register	In-Band Registration (XEP-0077[48])	
mod_roster	Roster management (XMPP IM)	
mod_roster_odbc	Roster management (XMPP IM)	supported DB (*)
mod_service_log	Copy user messages to logger service	
mod_shared_roster	Shared roster management	mod_roster or mod_roster_odbc
mod_stats	Statistics Gathering (XEP-0039[49])	
mod_time	Entity Time (XEP-0090[50])	
mod_vcard	vcard-temp (XEP-0054[51])	
mod_vcard_ldap	vcard-temp (XEP-0054[52])	LDAP server
mod_vcard_odbc	vcard-temp (XEP-0054[53])	supported DB (*)
mod_version	Software Version (XEP-0092[54])	

- (*) This module requires a supported database. For a list of supported databases, see section 3.2.

You can see which database backend each module needs by looking at the suffix:

- No suffix, this means that the modules uses Erlang's built-in database Mnesia as backend.
- '_odbc', this means that the module needs a supported database (see 3.2) as backend.
- '_ldap', this means that the module needs an LDAP server as backend.

If you want to, it is possible to use a relational database to store pieces of information. You can do this by changing the module name to a name with an `_odbc` suffix in `ejabberd` config file. You can use a relational database for the following data:

- Last connection date and time: Use `mod_last_odbc` instead of `mod_last`.
- Offline messages: Use `mod_offline_odbc` instead of `mod_offline`.
- Rosters: Use `mod_roster_odbc` instead of `mod_roster`.
- Users' VCARD: Use `mod_vcard_odbc` instead of `mod_vcard`.
- Private XML storage: Use `mod_private_odbc` instead of `mod_private`.
- User rules for blocking communications: Use `mod_privacy_odbc` instead of `mod_privacy`.

You can find more contributed modules[55] on the `ejabberd` website. Please remember that these contributions might not work or that they can contain severe bugs and security leaks. Therefore, use them at your own risk!

3.3.2 Common Options

The following options are used by many modules. Therefore, they are described in this separate section.

`iqdisc`

Many modules define handlers for processing IQ queries of different namespaces to this server or to a user (e. g. to `example.org` or to `user@example.org`). This option defines processing discipline for these queries. Possible values are:

`no_queue` All queries of a namespace with this processing discipline are processed immediately. This also means that no other packets can be processed until this one has been completely processed. Hence this discipline is not recommended if the processing of a query can take a relatively long time.

[55] http://www.ejabberd.im/contributions

`one_queue` In this case a separate queue is created for the processing of IQ queries of a namespace with this discipline. In addition, the processing of this queue is done in parallel with that of other packets. This discipline is most recommended.

`{queues, N}` N separate queues are created to process the queries. The queries are thus process in parallel, but in a controlled way.

`parallel` For every packet with this discipline a separate Erlang process is spawned. Consequently, all these packets are processed in parallel. Although spawning of Erlang process has a relatively low cost, this can break the server's normal work, because the Erlang emulator has a limit on the number of processes (32000 by default).

Example:

```
{modules,
 [
  ...
  {mod_time, [{iqdisc, no_queue}]},
  ...
 ]}.
```

`host`

This option defines the Jabber ID of a service provided by an `ejabberd` module. The keyword "@HOST@" is replaced at start time with the real virtual host string.

This example configures the echo module to provide its echoing service in the Jabber ID `mirror.example.org`:

```
{modules,
 [
  ...
  {mod_echo, [{host, "mirror.example.org"}]},
  ...
 ]}.
```

However, if there are several virtual hosts and this module is enabled in all of them, the "@HOST@" keyword must be used:

```
{modules,
 [
  ...
  {mod_echo, [{host, "mirror.@HOST@"}]},
  ...
 ]}.
```

3.3.3 mod_announce

This module enables configured users to broadcast announcements and to set the message of the day (MOTD). Configured users can perform these actions with a Jabber client either using Ad-hoc commands or sending messages to specific JIDs.

The Ad-hoc commands are listed in the Server Discovery. For this feature to work, `mod_adhoc` must be enabled.

The specific JIDs where messages can be sent are listed bellow. The first JID in each entry will apply only to the specified virtual host `example.org`, while the JID between brackets will apply to all virtual hosts in ejabberd.

`example.org/announce/all (example.org/announce/all-hosts/all)` The message is sent to all registered users. If the user is online and connected to several resources, only the resource with the highest priority will receive the message. If the registered user is not connected, the message will be stored offline in assumption that offline storage (see section 3.3.10) is enabled.

`example.org/announce/online (example.org/announce/all-hosts/online)` The message is sent to all connected users. If the user is online and connected to several resources, all resources will receive the message.

`example.org/announce/motd (example.org/announce/all-hosts/motd)` The message is set as the message of the day (MOTD) and is sent to users when they login. In addition the message is sent to all connected users (similar to `announce/online`).

`example.org/announce/motd/update (example.org/announce/all-hosts/motd/update)` The message is set as message of the day (MOTD) and is sent to users when they login. The message is *not sent* to any currently connected user.

`example.org/announce/motd/delete (example.org/announce/all-hosts/motd/delete)` Any message sent to this JID removes the existing message of the day (MOTD).

Options:

`access` This option specifies who is allowed to send announcements and to set the message of the day (by default, nobody is able to send such messages).

Examples:

- Only administrators can send announcements:

```
{access, announce, [{allow, admins}]}.

{modules,
 [
  ...
  {mod_adhoc, []},
```

```
  {mod_announce, [{access, announce}]},
  ...
 ]}.
```

- Administrators as well as the direction can send announcements:

```
{acl, direction, {user, "big_boss", "example.org"}}.
{acl, direction, {user, "assistant", "example.org"}}.
{acl, admins, {user, "admin", "example.org"}}.

{access, announce, [{allow, admins},
                    {allow, direction}]}.

{modules,
 [
  ...
  {mod_adhoc, []},
  {mod_announce, [{access, announce}]},
  ...
 ]}.
```

Note that `mod_announce` can be resource intensive on large deployments as it can broadcast lot of messages. This module should be disabled for instances of `ejabberd` with hundreds of thousands users.

3.3.4 mod_disco

This module adds support for Service Discovery (XEP-0030[56]). With this module enabled, services on your server can be discovered by Jabber clients. Note that `ejabberd` has no modules with support for the superseded Jabber Browsing (XEP-0011[57]) and Agent Information (XEP-0094[58]). Accordingly, Jabber clients need to have support for the newer Service Discovery protocol if you want them be able to discover the services you offer.

Options:

`iqdisc` This specifies the processing discipline for Service Discovery (`http://jabber.org/protocol/disco#items` and `http://jabber.org/protocol/disco#info`) IQ queries (see section 3.3.2).

`extra_domains` With this option, extra domains can be added to the Service Discovery item list.

Examples:

- To serve a link to the Jabber User Directory on `jabber.org`:

[56] http://www.xmpp.org/extensions/xep-0030.html
[57] http://www.xmpp.org/extensions/xep-0011.html
[58] http://www.xmpp.org/extensions/xep-0094.html

```
{modules,
 [
  ...
  {mod_disco, [{extra_domains, ["users.jabber.org"]}]},
  ...
 ]}.
```

- To serve a link to the transports on another server:

```
{modules,
 [
  ...
  {mod_disco, [{extra_domains, ["icq.example.com",
                                "msn.example.com"]}]},
  ...
 ]}.
```

- To serve a link to a few friendly servers:

```
{modules,
 [
  ...
  {mod_disco, [{extra_domains, ["example.org",
                                "example.com"]}]},
  ...
 ]}.
```

3.3.5 mod_echo

This module simply echoes any Jabber packet back to the sender. This mirror can be of interest for `ejabberd` and Jabber client debugging.

Options:

`host` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`echo.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

Example: Mirror, mirror, on the wall, who is the most beautiful of them all?

```
{modules,
 [
  ...
  {mod_echo, [{host, "mirror.example.org"}]},
  ...
 ]}.
```

3.3.6 mod_irc

This module is an IRC transport that can be used to join channels on IRC servers.

End user information:

- A Jabber client with 'groupchat 1.0' support or Multi-User Chat support (XEP-0045[59]) is necessary to join IRC channels.
- An IRC channel can be joined in nearly the same way as joining a Jabber Multi-User Chat room. The difference is that the room name will be 'channel%irc.example.org' in case irc.example.org is the IRC server hosting 'channel'. And of course the host should point to the IRC transport instead of the Multi-User Chat service.
- You can register your nickame by sending 'IDENTIFY password' to nickserver!irc.example.org@irc.jabberserver.org.
- Entering your password is possible by sending 'LOGIN nick password' to nickserver!irc.example.org@irc.jabberserver.org.
- When using a popular Jabber server, it can occur that no connection can be achieved with some IRC servers because they limit the number of conections from one IP.

Options:

host This option defines the Jabber ID of the service. If the host option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix 'irc.'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

access This option can be used to specify who may use the IRC transport (default value: all).

default_encoding Set the default IRC encoding (default value: "koi8-r").

Examples:

- In the first example, the IRC transport is available on (all) your virtual host(s) with the prefix 'irc.'. Furthermore, anyone is able to use the transport. The default encoding is set to "iso8859-15".

```
{modules,
 [
  ...
  {mod_irc, [{access, all}, {default_encoding, "iso8859-15"}]},
  ...
 ]}.
```

- In next example the IRC transport is available with JIDs with prefix irc-t.net. Moreover, the transport is only accessible by paying customers registered on our domains and on other servers.

[59] http://www.xmpp.org/extensions/xep-0045.html

```
{acl, paying_customers, {user, "customer1", "example.net"}}.
{acl, paying_customers, {user, "customer2", "example.com"}}.
{acl, paying_customers, {user, "customer3", "example.org"}}.

{access, paying_customers, [{allow, paying_customers},
                            {deny, all}]}.

{modules,
 [
  ...
  {mod_irc, [{access, paying_customers},
             {host, "irc.example.net"}]},
  ...
 ]}.
```

3.3.7 mod_last

This module adds support for Last Activity (XEP-0012[60]). It can be used to discover when a disconnected user last accessed the server, to know when a connected user was last active on the server, or to query the uptime of the ejabberd server.

Options:

iqdisc This specifies the processing discipline for Last activity (jabber:iq:last) IQ queries (see section 3.3.2).

3.3.8 mod_muc

This module provides a Multi-User Chat (XEP-0045[61]) service. Users can discover existing rooms, join or create them. Occupants of a room can chat in public or have private chats.

Some of the features of Multi-User Chat:

- Sending public and private messages to room occupants.
- Inviting other users to a room.
- Setting a room subject.
- Creating password protected rooms.
- Kicking and banning occupants.

[60] http://www.xmpp.org/extensions/xep-0012.html
[61] http://www.xmpp.org/extensions/xep-0045.html

The MUC service allows any Jabber ID to register a nickname, so nobody else can use that nickname in any room in the MUC service. To register a nickname, open the Service Discovery in your Jabber client and register in the MUC service.

This module supports clustering and load balancing. One module can be started per cluster node. Rooms are distributed at creation time on all available MUC module instances. The multi-user chat module is clustered but the rooms themselves are not clustered nor fault-tolerant: if the node managing a set of rooms goes down, the rooms disappear and they will be recreated on an available node on first connection attempt.

Module options:

`host` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`conference.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

`access` You can specify who is allowed to use the Multi-User Chat service. By default everyone is allowed to use it.

`access_create` To configure who is allowed to create new rooms at the Multi-User Chat service, this option can be used. By default everybody is allowed to create rooms.

`access_persistent` To configure who is allowed to modify the 'persistent' room option. By default everybody is allowed to modify that option.

`access_admin` This option specifies who is allowed to administrate the Multi-User Chat service. The default value is `none`, which means that only the room creator can administer his room. The administrators can send a normal message to the service JID, and it will be shown in all active rooms as a service message. The administrators can send a groupchat message to the JID of an active room, and the message will be shown in the room as a service message.

`history_size` A small history of the current discussion is sent to users when they enter the room. With this option you can define the number of history messages to keep and send to users joining the room. The value is an integer. Setting the value to `0` disables the history feature and, as a result, nothing is kept in memory. The default value is `20`. This value is global and thus affects all rooms on the service.

`max_users` This option defines at the service level, the maximum number of users allowed per room. It can be lowered in each room configuration but cannot be increased in individual room configuration. The default value is 200.

`max_users_admin_threshold` This option defines the number of service admins or room owners allowed to enter the room when the maximum number of allowed occupants was reached. The default limit is 5.

`max_user_conferences` This option defines the maximum number of rooms that any given user can join. The default value is 10. This option is used to prevent possible abuses. Note that this is a soft limit: some users can sometimes join more conferences in cluster configurations.

`min_message_interval` This option defines the minimum interval between two messages send by an occupant in seconds. This option is global and valid for all rooms. A decimal value can be used. When this option is not defined, message rate is not limited. This feature can be used to protect a MUC service from occupant abuses and limit number of messages that will be broadcasted by the service. A good value for this minimum message interval is 0.4 second. If an occupant tries to send messages faster, an error is send back explaining that the message has been discarded and describing the reason why the message is not acceptable.

`min_presence_interval` This option defines the minimum of time between presence changes coming from a given occupant in seconds. This option is global and valid for all rooms. A decimal value can be used. When this option is not defined, no restriction is applied. This option can be used to protect a MUC service for occupants abuses. If an occupant tries to change its presence more often than the specified interval, the presence is cached by `ejabberd` and only the last presence is broadcasted to all occupants in the room after expiration of the interval delay. Intermediate presence packets are silently discarded. A good value for this option is 4 seconds.

`default_room_options` This module option allows to define the desired default room options. Note that the creator of a room can modify the options of his room at any time using a Jabber client with MUC capability. The available room options and the default values are:

`{allow_change_subj, true}` Allow occupants to change the subject.

`{allow_private_messages, true}` Occupants can send private messages to other occupants.

`{allow_query_users, true}` Occupants can send IQ queries to other occupants.

`{allow_user_invites, false}` Allow occupants to send invitations.

`{allow_visitor_nickchange, true}` Allow visitors to change nickname.

`{allow_visitor_status, true}` Allow visitors to send status text in presence updates. If disallowed, the `status` text is stripped before broadcasting the presence update to all the room occupants.

`{anonymous, true}` Occupants are allowed to see the real JIDs of other occupants.

`{logging, false}` The public messages are logged using `mod_muc_log`.

`{max_users, 200}` Maximum number of occupants in the room.

`{members_by_default, true}` The occupants that enter the room are participants by default, so they have 'voice'.

`{members_only, false}` Only members of the room can enter.

`{moderated, true}` Only occupants with 'voice' can send public messages.

`{password, ""}` Password of the room. You may want to enable the next option too.

`{password_protected, false}` The password is required to enter the room.

`{persistent, false}` The room persists even if the last participant leaves.

`{public, true}` The room is public in the list of the MUC service, so it can be discovered.

`{public_list, true}` The list of participants is public, without requiring to enter the room.

`{title, ""}` A human-readable title of the room.

All of those room options can be set to `true` or `false`, except `password` and `title` which are strings, and `max_users` that is integer.

Examples:

- In the first example everyone is allowed to use the Multi-User Chat service. Everyone will also be able to create new rooms but only the user `admin@example.org` is allowed to administrate any room. In this example he is also a global administrator. When `admin@example.org` sends a message such as 'Tomorrow, the Jabber server will be moved to new hardware. This will involve service breakdowns around 23:00 UMT. We apologise for this inconvenience.' to `conference.example.org`, it will be displayed in all active rooms. In this example the history feature is disabled.

```
{acl, admins, {user, "admin", "example.org"}}.

{access, muc_admins, [{allow, admins}]}.

{modules,
 [
  ...
  {mod_muc, [{access, all},
             {access_create, all},
             {access_admin, muc_admins},
             {history_size, 0}]},
  ...
 ]}.
```

- In the second example the Multi-User Chat service is only accessible by paying customers registered on our domains and on other servers. Of course the administrator is also allowed to access rooms. In addition, he is the only authority able to create and administer rooms. When `admin@example.org` sends a message such as 'Tomorrow, the Jabber server will be moved to new hardware. This will involve service breakdowns around 23:00 UMT. We apologise for this inconvenience.' to `conference.example.org`, it will be displayed in all active rooms. No `history_size` option is used, this means that the feature is enabled and the default value of 20 history messages will be send to the users.

```
{acl, paying_customers, {user, "customer1", "example.net"}}.
{acl, paying_customers, {user, "customer2", "example.com"}}.
{acl, paying_customers, {user, "customer3", "example.org"}}.
{acl, admins, {user, "admin", "example.org"}}.

{access, muc_admins, [{allow, admins},
                      {deny, all}]}.
{access, muc_access, [{allow, paying_customers},
                      {allow, admins},
                      {deny, all}]}.

{modules,
 [
  ...
  {mod_muc, [{access, muc_access},
             {access_create, muc_admins},
```

```
             {access_admin, muc_admins}]},
  ...
 ]}.
```

- In the following example, MUC anti abuse options are used. An occupant cannot send more than one message every 0.4 seconds and cannot change its presence more than once every 4 seconds. No ACLs are defined, but some user restriction could be added as well:

```
{modules,
 [
  ...
  {mod_muc, [{min_message_interval, 0.4},
             {min_presence_interval, 4}]},
  ...
 ]}.
```

- This example shows how to use `default_room_options` to make sure the newly created rooms have by default those options.

```
{modules,
 [
  ...
  {mod_muc, [{access, muc_access},
             {access_create, muc_admins},
             {default_room_options,
              [
               {allow_change_subj, false},
               {allow_query_users, true},
               {allow_private_messages, true},
               {members_by_default, false},
               {title, "New chatroom"},
               {anonymous, false}
              ]},
             {access_admin, muc_admins}]},
  ...
 ]}.
```

3.3.9 `mod_muc_log`

This module enables optional logging of Multi-User Chat (MUC) public conversations to HTML. Once you enable this module, users can join a room using a MUC capable Jabber client, and if they have enough privileges, they can request the configuration form in which they can set the option to enable room logging.

Features:

- Room details are added on top of each page: room title, JID, author, subject and configuration.

- The room JID in the generated HTML is a link to join the room (using XMPP URI[62]).
- Subject and room configuration changes are tracked and displayed.
- Joins, leaves, nick changes, kicks, bans and '/me' are tracked and displayed, including the reason if available.
- Generated HTML files are XHTML 1.0 Transitional and CSS compliant.
- Timestamps are self-referencing links.
- Links on top for quicker navigation: Previous day, Next day, Up.
- CSS is used for style definition, and a custom CSS file can be used.
- URLs on messages and subjects are converted to hyperlinks.
- Timezone used on timestamps is shown on the log files.
- A custom link can be added on top of each page.

Options:

`access_log` This option restricts which occupants are allowed to enable or disable room logging. The default value is `muc_admin`. Note for this default setting you need to have an access rule for `muc_admin` in order to take effect.

`cssfile` With this option you can set whether the HTML files should have a custom CSS file or if they need to use the embedded CSS file. Allowed values are `false` and an URL to a CSS file. With the first value, HTML files will include the embedded CSS code. With the latter, you can specify the URL of the custom CSS file (for example: 'http://example.com/my.css'). The default value is `false`.

`dirtype` The type of the created directories can be specified with this option. Allowed values are `subdirs` and `plain`. With the first value, subdirectories are created for each year and month. With the latter, the names of the log files contain the full date, and there are no subdirectories. The default value is `subdirs`.

`outdir` This option sets the full path to the directory in which the HTML files should be stored. Make sure the `ejabberd` daemon user has write access on that directory. The default value is `"www/muc"`.

`timezone` The time zone for the logs is configurable with this option. Allowed values are `local` and `universal`. With the first value, the local time, as reported to Erlang by the operating system, will be used. With the latter, GMT/UTC time will be used. The default value is `local`.

`spam_prevention` To prevent spam, the `spam_prevention` option adds a special attribute to links that prevent their indexation by search engines. The default value is `true`, which mean that nofollow attributes will be added to user submitted links.

`top_link` With this option you can customize the link on the top right corner of each log file. The syntax of this option is `{"URL", "Text"}`. The default value is `{"/", "Home"}`.

[62] http://www.xmpp.org/rfcs/rfc5122.html

Examples:

- In the first example any room owner can enable logging, and a custom CSS file will be used (http://example.com/my.css). The names of the log files will contain the full date, and there will be no subdirectories. The log files will be stored in /var/www/muclogs, and the time zone will be GMT/UTC. Finally, the top link will be `<a href="http://www.jabber.ru/">Jabber.ru</a>`.

```
{access, muc, [{allow, all}]}.

{modules,
 [
  ...
  {mod_muc_log, [
                 {access_log, muc},
                 {cssfile, "http://example.com/my.css"},
                 {dirtype, plain},
                 {outdir, "/var/www/muclogs"},
                 {timezone, universal},
                 {spam_prevention, true},
                 {top_link, {"http://www.jabber.ru/", "Jabber.ru"}}
                ]},
  ...
 ]}.
```

- In the second example only `admin1@example.org` and `admin2@example.net` can enable logging, and the embedded CSS file will be used. The names of the log files will only contain the day (number), and there will be subdirectories for each year and month. The log files will be stored in /var/www/muclogs, and the local time will be used. Finally, the top link will be the default `<a href="/">Home</a>`.

```
{acl, admins, {user, "admin1", "example.org"}}.
{acl, admins, {user, "admin2", "example.net"}}.

{access, muc_log, [{allow, admins},
                   {deny, all}]}.

{modules,
 [
  ...
  {mod_muc_log, [
                 {access_log, muc_log},
                 {cssfile, false},
                 {dirtype, subdirs},
                 {outdir, "/var/www/muclogs"},
                 {timezone, local}
                ]},
  ...
 ]}.
```

3.3.10 mod_offline

This module implements offline message storage. This means that all messages sent to an offline user will be stored on the server until that user comes online again. Thus it is very similar to how email works. Note that `ejabberdctl` has a command to delete expired messages (see section 4.1).

`user_max_messages` This option is use to set a max number of offline messages per user (quota). Its value can be either `infinity` or a strictly positive integer. The default value is `infinity`.

3.3.11 mod_privacy

This module implements Blocking Communication (also known as Privacy Rules) as defined in section 10 from XMPP IM. If end users have support for it in their Jabber client, they will be able to:

- Retrieving one's privacy lists.
- Adding, removing, and editing one's privacy lists.
- Setting, changing, or declining active lists.
- Setting, changing, or declining the default list (i.e., the list that is active by default).
- Allowing or blocking messages based on JID, group, or subscription type (or globally).
- Allowing or blocking inbound presence notifications based on JID, group, or subscription type (or globally).
- Allowing or blocking outbound presence notifications based on JID, group, or subscription type (or globally).
- Allowing or blocking IQ stanzas based on JID, group, or subscription type (or globally).
- Allowing or blocking all communications based on JID, group, or subscription type (or globally).

(from `http://www.xmpp.org/specs/rfc3921.html#privacy`)

Options:

`iqdisc` This specifies the processing discipline for Blocking Communication (`jabber:iq:privacy`) IQ queries (see section 3.3.2).

3.3.12 mod_private

This module adds support for Private XML Storage (XEP-0049[63]):

> Using this method, Jabber entities can store private data on the server and retrieve it whenever necessary. The data stored might be anything, as long as it is valid XML. One typical usage for this namespace is the server-side storage of client-specific preferences; another is Bookmark Storage (XEP-0048[64]).

Options:

`iqdisc` This specifies the processing discipline for Private XML Storage (`jabber:iq:private`) IQ queries (see section 3.3.2).

3.3.13 mod_proxy65

This module implements SOCKS5 Bytestreams (XEP-0065[65]). It allows `ejabberd` to act as a file transfer proxy between two XMPP clients.

Options:

`host` This option defines the hostname of the service. If this option is not set, the prefix '`proxy.`' is added to `ejabberd` hostname.

`name` Defines Service Discovery name of the service. Default is `"SOCKS5 Bytestreams"`.

`ip` This option specifies which network interface to listen for. Default is an IP address of the service's DNS name, or, if fails, `{127,0,0,1}`.

`port` This option defines port to listen for incoming connections. Default is 7777.

`auth_type` SOCKS5 authentication type. Possible values are `anonymous` and `plain`. Default is `anonymous`.

`access` Defines ACL for file transfer initiators. Default is `all`.

`max_connections` Maximum number of active connections per file transfer initiator. No limit by default.

`shaper` This option defines shaper for the file transfer peers. Shaper with the maximum bandwidth will be selected. Default is `none`.

Examples:

- The simpliest configuration of the module:

[63] http://www.xmpp.org/extensions/xep-0049.html
[64] http://www.xmpp.org/extensions/xep-0048.html
[65] http://www.xmpp.org/extensions/xep-0065.html

```
{modules,
 [
  ...
  {mod_proxy65, []},
  ...
 ]}.
```

- More complicated configuration.

```
{acl, proxy_users, {server, "example.org"}}.
{access, proxy65_access, [{allow, proxy_users}, {deny, all}]}.

{acl, admin, {user, "admin", "example.org"}}.
{shaper, normal, {maxrate, 10240}}. %% 10 Kbytes/sec
{access, proxy65_shaper, [{none, admin}, {normal, all}]}.

{modules,
 [
  ...
  {mod_proxy65, [{host, "proxy1.example.org"},
                 {name, "File Transfer Proxy"},
                 {ip, {200,150,100,1}},
                 {port, 7778},
                 {max_connections, 5},
                 {access, proxy65_access},
                 {shaper, proxy65_shaper}]},
  ...
 ]}.
```

3.3.14 `mod_pubsub`

This module offers a Publish-Subscribe Service (XEP-0060[66]). The functionality in `mod_pubsub` can be extended using plugins. The plugin that implements PEP (Personal Eventing via Pubsub) (XEP-0163[67]) is enabled in the default ejabberd configuration file, and it requires `mod_caps`.

Options:

`host` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`pubsub.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

`access_createnode` This option restricts which users are allowed to create pubsub nodes using ACL and ACCESS. The default value is `pubsub_createnode`.

`plugins` To specify which pubsub node plugins to use. If not defined, the default pubsub plugin is always used.

[66] http://www.xmpp.org/extensions/xep-0060.html
[67] http://www.xmpp.org/extensions/xep-0163.html

`nodetree` To specify which nodetree to use. If not defined, the default pubsub nodetree is used. Nodetrees are default and virtual. Only one nodetree can be used and is shared by all node plugins.

Example:

```
{modules,
 [
  ...
  {mod_pubsub, [
                {access_createnode, pubsub_createnode},
                {plugins, ["default", "pep"]}
               ]}
  ...
 ]}.
```

3.3.15 mod_register

This module adds support for In-Band Registration (XEP-0077[68]). This protocol enables end users to use a Jabber client to:

- Register a new account on the server.
- Change the password from an existing account on the server.
- Delete an existing account on the server.

Options:

`access` This option can be configured to specify rules to restrict registration. If a rule returns 'deny' on the requested user name, registration for that user name is denied. (there are no restrictions by default).

`welcome_message` Set a welcome message that is sent to each newly registered account. The first string is the subject, and the second string is the message body. In the body you can set a newline with the characters: `\n`

`registration_watchers` This option defines a list of JIDs which will be notified each time a new account is registered.

`iqdisc` This specifies the processing discipline for In-Band Registration (`jabber:iq:register`) IQ queries (see section 3.3.2).

This module reads also another option defined globably for the server: `{registration_timeout, Timeout}`. This option limits the frequency of registration from a given IP or username. So, a user can't register a new account from the same IP address or JID during this number of seconds

[68] http://www.xmpp.org/extensions/xep-0077.html

after previous registration. Timeout is expressed in seconds, and must be an integer. To disable this limitation, instead of an integer put a word like: `infinity`. Default value: 600 seconds.

Examples:

- Next example prohibits the registration of too short account names:

```
{acl, shortname, {user_glob, "?"}}.
{acl, shortname, {user_glob, "??"}}.
%% The same using regexp:
%%{acl, shortname, {user_regexp, "^..?$"}}.

{access, register, [{deny, shortname},
                    {allow, all}]}.

{modules,
 [
  ...
  {mod_register, [{access, register}]},
  ...
 ]}.
```

- The in-band registration of new accounts can be prohibited by changing the `access` option. If you really want to disable all In-Band Registration functionality, that is changing passwords in-band and deleting accounts in-band, you have to remove `mod_register` from the modules list. In this example all In-Band Registration functionality is disabled:

```
{access, register, [{deny, all}]}.

{modules,
 [
  ...
  %% {mod_register, [{access, register}]},
  ...
 ]}.
```

- Define the welcome message and two registration watchers. Also define a registration timeout of one hour:

```
{registration_timeout, 3600}.
{modules,
 [
  ...
  {mod_register,
   [
    {welcome_message, {"Welcome!", "Hi.\nWelcome to this Jabber server.\n Check http://ww
    {registration_watchers, ["admin1@example.org", "boss@example.net"]}
   ]},
  ...
 ]}.
```

3.3.16 mod_roster

This module implements roster management as defined in RFC 3921: XMPP IM[69].

Options:

`iqdisc` This specifies the processing discipline for Roster Management (`jabber:iq:roster`) IQ queries (see section 3.3.2).

3.3.17 mod_service_log

This module adds support for logging end user packets via a Jabber message auditing service such as Bandersnatch[70]. All user packets are encapsulated in a `<route/>` element and sent to the specified service(s).

Options:

`loggers` With this option a (list of) service(s) that will receive the packets can be specified.

Examples:

- To log all end user packets to the Bandersnatch service running on `bandersnatch.example.com`:

```
{modules,
 [
  ...
  {mod_service_log, [{loggers, ["bandersnatch.example.com"]}]},
  ...
 ]}.
```

- To log all end user packets to the Bandersnatch service running on `bandersnatch.example.com` and the backup service on `bandersnatch.example.org`:

```
{modules,
 [
  ...
  {mod_service_log, [{loggers, ["bandersnatch.example.com",
                                "bandersnatch.example.org"]}]},
  ...
 ]}.
```

[69] http://www.xmpp.org/specs/rfc3921.html#roster
[70] http://www.funkypenguin.info/project/bandersnatch/

3.3.18 mod_shared_roster

This module enables you to create shared roster groups. This means that you can create groups of people that can see members from (other) groups in their rosters. The big advantages of this feature are that end users do not need to manually add all users to their rosters, and that they cannot permanently delete users from the shared roster groups. A shared roster group can have members from any Jabber server, but the presence will only be available from and to members of the same virtual host where the group is created.

Shared roster groups can be edited *only* via the Web Admin. Each group has a unique identification and the following parameters:

Name The name of the group, which will be displayed in the roster.

Description The description of the group. This parameter does not affect anything.

Members A list of full JIDs of group members, entered one per line in the Web Admin. To put as members all the registered users in the virtual hosts, you can use the special directive: @all@. Note that this directive is designed for a small server with just a few hundred users.

Displayed groups A list of groups that will be in the rosters of this group's members.

Examples:

- Take the case of a computer club that wants all its members seeing each other in their rosters. To achieve this, they need to create a shared roster group similar to next table:

Identification	Group '`club_members`'
Name	Club Members
Description	Members from the computer club

Members	`member1@example.org`
	`member2@example.org`
	`member3@example.org`

Displayed groups	`club_members`

- In another case we have a company which has three divisions: Management, Marketing and Sales. All group members should see all other members in their rosters. Additionally, all managers should have all marketing and sales people in their roster. Simultaneously, all marketeers and the whole sales team should see all managers. This scenario can be achieved by creating shared roster groups as shown in the following table:

Identification	Group 'management'	Group 'marketing'	Group 'sales'
Name	Management	Marketing	Sales
Description			

Members	`manager1@example.org`
	`manager2@example.org`
	`manager3@example.org`
	`manager4@example.org`
		`marketeer1@example.org`
		`marketeer2@example.org`
		`marketeer3@example.org`
		`marketeer4@example.org`
			`saleswoman1@example.org`
			`salesman1@example.org`
			`saleswoman2@example.org`
			`salesman2@example.org`

Displayed groups	`management`
	`marketing`
	`sales`
		`management`
		`marketing`
			`management`
			`sales`

3.3.19 `mod_stats`

This module adds support for Statistics Gathering (XEP-0039[71]). This protocol allows you to retrieve next statistics from your `ejabberd` deployment:

- Total number of registered users on the current virtual host (users/total).
- Total number of registered users on all virtual hosts (users/all-hosts/total).
- Total number of online users on the current virtual host (users/online).
- Total number of online users on all virtual hosts (users/all-hosts/online).

Options:

`iqdisc` This specifies the processing discipline for Statistics Gathering (`http://jabber.org/protocol/stats`) IQ queries (see section 3.3.2).

As there are only a small amount of clients (for example Tkabber[72]) and software libraries with support for this XEP, a few examples are given of the XML you need to send in order to get the statistics. Here they are:

- You can request the number of online users on the current virtual host (`example.org`) by sending:

```
<iq to='example.org' type='get'>
  <query xmlns='http://jabber.org/protocol/stats'>
    <stat name='users/online'/>
  </query>
</iq>
```

- You can request the total number of registered users on all virtual hosts by sending:

[71] http://www.xmpp.org/extensions/xep-0039.html
[72] http://tkabber.jabber.ru/

```
<iq to='example.org' type='get'>
  <query xmlns='http://jabber.org/protocol/stats'>
    <stat name='users/all-hosts/total'/>
  </query>
</iq>
```

3.3.20 mod_time

This module features support for Entity Time (XEP-0090[73]). By using this XEP, you are able to discover the time at another entity's location.

Options:

`iqdisc` This specifies the processing discipline for Entity Time (`jabber:iq:time`) IQ queries (see section 3.3.2).

3.3.21 mod_vcard

This module allows end users to store and retrieve their vCard, and to retrieve other users vCards, as defined in vcard-temp (XEP-0054[74]). The module also implements an uncomplicated Jabber User Directory based on the vCards of these users. Moreover, it enables the server to send its vCard when queried.

Options:

`host` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`vjud.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

`iqdisc` This specifies the processing discipline for `vcard-temp` IQ queries (see section 3.3.2).

`search` This option specifies whether the search functionality is enabled (value: `true`) or disabled (value: `false`). If disabled, the option `host` will be ignored and the Jabber User Directory service will not appear in the Service Discovery item list. The default value is `true`.

`matches` With this option, the number of reported search results can be limited. If the option's value is set to `infinity`, all search results are reported. The default value is `30`.

`allow_return_all` This option enables you to specify if search operations with empty input fields should return all users who added some information to their vCard. The default value is `false`.

`search_all_hosts` If this option is set to `true`, search operations will apply to all virtual hosts. Otherwise only the current host will be searched. The default value is `true`. This option is available in `mod_vcard`, but not available in `mod_vcard_odbc`.

[73] http://www.xmpp.org/extensions/xep-0090.html
[74] http://www.xmpp.org/extensions/xep-0054.html

Examples:

- In this first situation, search results are limited to twenty items, every user who added information to their vCard will be listed when people do an empty search, and only users from the current host will be returned:

```
{modules,
 [
  ...
  {mod_vcard, [{search, true},
               {matches, 20},
               {allow_return_all, true},
               {search_all_hosts, false}]},
  ...
 ]}.
```

- The second situation differs in a way that search results are not limited, and that all virtual hosts will be searched instead of only the current one:

```
{modules,
 [
  ...
  {mod_vcard, [{search, true},
               {matches, infinity},
               {allow_return_all, true}]},
  ...
 ]}.
```

3.3.22 mod_vcard_ldap

`ejabberd` can map LDAP attributes to vCard fields. This behaviour is implemented in the `mod_vcard_ldap` module. This module does not depend on the authentication method (see 3.2.5).

Note that `ejabberd` treats LDAP as a read-only storage: it is possible to consult data, but not possible to create accounts, change password or edit vCard that is stored in LDAP.

The `mod_vcard_ldap` module has its own optional parameters. The first group of parameters has the same meaning as the top-level LDAP parameters to set the authentication method: `ldap_servers`, `ldap_port`, `ldap_rootdn`, `ldap_password`, `ldap_base`, `ldap_uids`, and `ldap_filter`. See section 3.2.5 for detailed information about these options. If one of these options is not set, `ejabberd` will look for the top-level option with the same name.

The second group of parameters consists of the following `mod_vcard_ldap`-specific options:

`host` This option defines the Jabber ID of the service. If the `host` option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix '`vjud.`'. The keyword "@HOST@" is replaced at start time with the real virtual host name.

`iqdisc` This specifies the processing discipline for `vcard-temp` IQ queries (see section 3.3.2).

`search` This option specifies whether the search functionality is enabled (value: `true`) or disabled (value: `false`). If disabled, the option `host` will be ignored and the Jabber User Directory service will not appear in the Service Discovery item list. The default value is `true`.

`matches` With this option, the number of reported search results can be limited. If the option's value is set to `infinity`, all search results are reported. The default value is `30`.

`ldap_vcard_map` With this option you can set the table that maps LDAP attributes to vCard fields. The format is: `[Name_of_vCard_field, Pattern, List_of_LDAP_attributes, ...]`. `Name_of_vcard_field` is the type name of the vCard as defined in RFC 2426[75]. `Pattern` is a string which contains pattern variables `"%u"`, `"%d"` or `"%s"`. `List_of_LDAP_attributes` is the list containing LDAP attributes. The pattern variables `"%s"` will be sequentially replaced with the values of LDAP attributes from `List_of_LDAP_attributes`, `"%u"` will be replaced with the user part of a JID, and `"%d"` will be replaced with the domain part of a JID. The default is:

```
[{"NICKNAME", "%u", []},
 {"FN", "%s", ["displayName"]},
 {"LAST", "%s", ["sn"]},
 {"FIRST", "%s", ["givenName"]},
 {"MIDDLE", "%s", ["initials"]},
 {"ORGNAME", "%s", ["o"]},
 {"ORGUNIT", "%s", ["ou"]},
 {"CTRY", "%s", ["c"]},
 {"LOCALITY", "%s", ["l"]},
 {"STREET", "%s", ["street"]},
 {"REGION", "%s", ["st"]},
 {"PCODE", "%s", ["postalCode"]},
 {"TITLE", "%s", ["title"]},
 {"URL", "%s", ["labeleduri"]},
 {"DESC", "%s", ["description"]},
 {"TEL", "%s", ["telephoneNumber"]},
 {"EMAIL", "%s", ["mail"]},
 {"BDAY", "%s", ["birthDay"]},
 {"ROLE", "%s", ["employeeType"]},
 {"PHOTO", "%s", ["jpegPhoto"]}]
```

`ldap_search_fields` This option defines the search form and the LDAP attributes to search within. The format is: `[Name, Attribute, ...]`. `Name` is the name of a search form field which will be automatically translated by using the translation files (see `msgs/*.msg` for available words). `Attribute` is the LDAP attribute or the pattern `"%u"`. The default is:

```
[{"User", "%u"},
 {"Full Name", "displayName"},
 {"Given Name", "givenName"},
 {"Middle Name", "initials"},
 {"Family Name", "sn"},
```

[75] http://www.ietf.org/rfc/rfc2426.txt

```
 {"Nickname", "%u"},
 {"Birthday", "birthDay"},
 {"Country", "c"},
 {"City", "l"},
 {"Email", "mail"},
 {"Organization Name", "o"},
 {"Organization Unit", "ou"}]
```

`ldap_search_reported` This option defines which search fields should be reported. The format is: `[Name, vCard_Name, ...]`. `Name` is the name of a search form field which will be automatically translated by using the translation files (see `msgs/*.msg` for available words). `vCard_Name` is the vCard field name defined in the `ldap_vcard_map` option. The default is:

```
[{"Full Name", "FN"},
 {"Given Name", "FIRST"},
 {"Middle Name", "MIDDLE"},
 {"Family Name", "LAST"},
 {"Nickname", "NICKNAME"},
 {"Birthday", "BDAY"},
 {"Country", "CTRY"},
 {"City", "LOCALITY"},
 {"Email", "EMAIL"},
 {"Organization Name", "ORGNAME"},
 {"Organization Unit", "ORGUNIT"}]
```

Examples:

- Let's say `ldap.example.org` is the name of our LDAP server. We have users with their passwords in `"ou=Users,dc=example,dc=org"` directory. Also we have addressbook, which contains users emails and their additional infos in `"ou=AddressBook,dc=example,dc=org"` directory. Corresponding authentication section should looks like this:

```
%% authentication method
{auth_method, ldap}.
%% DNS name of our LDAP server
{ldap_servers, ["ldap.example.org"]}.
%% We want to authorize users from 'shadowAccount' object class only
{ldap_filter, "(objectClass=shadowAccount)"}.
```

  Now we want to use users LDAP-info as their vCards. We have four attributes defined in our LDAP schema: `"mail"` — email address, `"givenName"` — first name, `"sn"` — second name, `"birthDay"` — birthday. Also we want users to search each other. Let's see how we can set it up:

```
{modules,
  ...
  {mod_vcard_ldap,
   [
```

```
    %% We use the same server and port, but want to bind anonymously because
    %% our LDAP server accepts anonymous requests to
    %% "ou=AddressBook,dc=example,dc=org" subtree.
    {ldap_rootdn, ""},
    {ldap_password, ""},
    %% define the addressbook's base
    {ldap_base, "ou=AddressBook,dc=example,dc=org"},
    %% uidattr: user's part of JID is located in the "mail" attribute
    %% uidattr_format: common format for our emails
    {ldap_uids, [{"mail","%u@mail.example.org"}]},
    %% We have to define empty filter here, because entries in addressbook does not
    %% belong to shadowAccount object class
    {ldap_filter, ""},
    %% Now we want to define vCard pattern
    {ldap_vcard_map,
     [{"NICKNAME", "%u", []}, % just use user's part of JID as his nickname
      {"FIRST", "%s", ["givenName"]},
      {"LAST", "%s", ["sn"]},
      {"FN", "%s, %s", ["sn", "givenName"]}, % example: "Smith, John"
      {"EMAIL", "%s", ["mail"]},
      {"BDAY", "%s", ["birthDay"]}]},
    %% Search form
    {ldap_search_fields,
     [{"User", "%u"},
      {"Name", "givenName"},
      {"Family Name", "sn"},
      {"Email", "mail"},
      {"Birthday", "birthDay"}]},
    %% vCard fields to be reported
    %% Note that JID is always returned with search results
    {ldap_search_reported,
     [{"Full Name", "FN"},
      {"Nickname", "NICKNAME"},
      {"Birthday", "BDAY"}]}
  ]}
  ...
}.
```

Note that mod_vcard_ldap module checks an existence of the user before searching his info in LDAP.

- ldap_vcard_map example:

```
{ldap_vcard_map,
 [{"NICKNAME", "%u", []},
  {"FN", "%s", ["displayName"]},
  {"CTRY", "Russia", []},
  {"EMAIL", "%u@%d", []},
  {"DESC", "%s\n%s", ["title", "description"]}
 ]},
```

- ldap_search_fields example:

```
{ldap_search_fields,
 [{"User", "uid"},
  {"Full Name", "displayName"},
  {"Email", "mail"}
 ]},
```

- ldap_search_reported example:

```
{ldap_search_reported,
 [{"Full Name", "FN"},
  {"Email", "EMAIL"},
  {"Birthday", "BDAY"},
  {"Nickname", "NICKNAME"}
 ]},
```

3.3.23 mod_version

This module implements Software Version (XEP-0092[76]). Consequently, it answers ejabberd's version when queried.

Options:

show_os Should the operating system be revealed or not. The default value is true.

iqdisc This specifies the processing discipline for Software Version (jabber:iq:version) IQ queries (see section 3.3.2).

[76] http://www.xmpp.org/extensions/xep-0092.html

Chapter 4

Managing an `ejabberd` server

4.1 `ejabberdctl`

4.1.1 Commands

The `ejabberdctl` command line administration script allows to start, stop and perform many other administrative tasks in a local or remote `ejabberd` server.

When `ejabberdctl` is executed without any parameter, it displays the available options. If there isn't an `ejabberd` server running, the available parameters are:

- `start` Start `ejabberd` in background mode. This is the default method.
- `debug` Attach an Erlang shell to an already existing `ejabberd` server. This allows to execute commands interactively in the `ejabberd` server.
- `live` Start `ejabberd` in live mode: the shell keeps attached to the started server, showing log messages and allowing to execute interactive commands.

If there is an `ejabberd` server running in the system, `ejabberdctl` shows all the available commands in that server. The more interesting ones are:

- `status` Check the status of the `ejabberd` server.
- `stop` Stop the `ejabberd` server which is running in the machine.
- `reopen-log` If you use a tool to rotate logs, you have to configure it so that this command is executed after each rotation.
- `backup, restore, install-fallback, dump, load` You can use these commands to create and restore backups.

`import-file, import-dir` These options can be used to migrate from other Jabber/XMPP servers. There exist tutorials to migrate from other software to ejabberd[1].

`delete-expired-messages` This option can be used to delete old messages in offline storage. This might be useful when the number of offline messages is very high.

The `ejabberdctl` script also allows the argument `--node NODENAME`. This allows to administer a remote node.

The `ejabberdctl` script can be configured in the file `ejabberdctl.cfg`. This file includes detailed information about each configurable option.

The `ejabberdctl` script returns a numerical status code. Success is represented by `0`, error is represented by `1`, and other codes may be used for specifical results. This can be used by other scripts to determine automatically if a command succedded or failed, for example using: `echo $?`

4.1.2 Erlang runtime system

`ejabberd` is an Erlang/OTP application that runs inside an Erlang runtime system. This system is configured using environment variables and command line parameters. The `ejabberdctl` administration script uses many of those possibilities. You can configure some of them with the file `ejabberdctl.cfg`, which includes detailed description about them. This section describes for reference purposes all the environment variables and command line parameters.

The environment variables:

`EJABBERD_CONFIG_PATH` Path to the ejabberd configuration file.

`EJABBERD_MSGS_PATH` Path to the directory with translated strings.

`EJABBERD_LOG_PATH` Path to the ejabberd service log file.

`EJABBERD_SO_PATH` Path to the directory with binary system libraries.

`HOME` Path to the directory that is considered `ejabberd`'s home. This path is used to read the file `.erlang.cookie`.

`ERL_CRASH_DUMP` Path to the file where crash reports will be dumped.

`ERL_INETRC` Indicates which IP name resolution to use. It is required if using `-sname`.

`ERL_MAX_PORTS` Maximum number of simultaneously open Erlang ports.

`ERL_MAX_ETS_TABLES` Maximum number of ETS and Mnesia tables.

The command line parameters:

[1] http://www.ejabberd.im/migrate-to-ejabberd

`-sname ejabberd` The Erlang node will be identified using only the first part of the host name, i. e. other Erlang nodes outside this domain cannot contact this node. This is the preferable option in most cases.

`-name ejabberd` The Erlang node will be fully identified. This is only useful if you plan to setup an `ejabberd` cluster with nodes in different networks.

`-kernel inetrc "/etc/ejabberd/inetrc"` Indicates which IP name resolution to use. It is required if using `-sname`.

`-detached` Starts the Erlang system detached from the system console. Useful for running daemons and backgrounds processes.

`-noinput` Ensures that the Erlang system never tries to read any input. Useful for running daemons and backgrounds processes.

`-pa /var/lib/ejabberd/ebin` Specify the directory where Erlang binary files (*.beam) are located.

`-s ejabberd` Tell Erlang runtime system to start the `ejabberd` application.

`-mnesia dir "/var/lib/ejabberd/db/nodename"` Specify the Mnesia database directory.

`-sasl sasl_error_logger {file, "/var/log/ejabberd/sasl.log"}` Path to the Erlang/OTP system log file.

`+K [true|false]` Kernel polling.

`-smp [auto|enable|disable]` SMP support.

`+P 250000` Maximum number of Erlang processes.

`-remsh ejabberd@localhost` Open an Erlang shell in a remote Erlang node.

Note that some characters need to be escaped when used in shell scripts, for instance `"` and `{}`. You can find other options in the Erlang manual page (`erl -man erl`).

4.2 Web Admin

The `ejabberd` Web Admin allows to administer most of `ejabberd` using a web browser.

This feature is enabled by default: a `ejabberd_http` listener with the option `web_admin` (see section 3.1.3) is included in the listening ports. Then you can open `http://server:port/admin/` in your favourite web browser. You will be asked to enter the username (the *full* Jabber ID) and password of an `ejabberd` user with administrator rights. After authentication you will see a page similar to figure 4.1.

Here you can edit access restrictions, manage users, create backups, manage the database, enable/disable ports listened for, view server statistics,. . .

Examples:

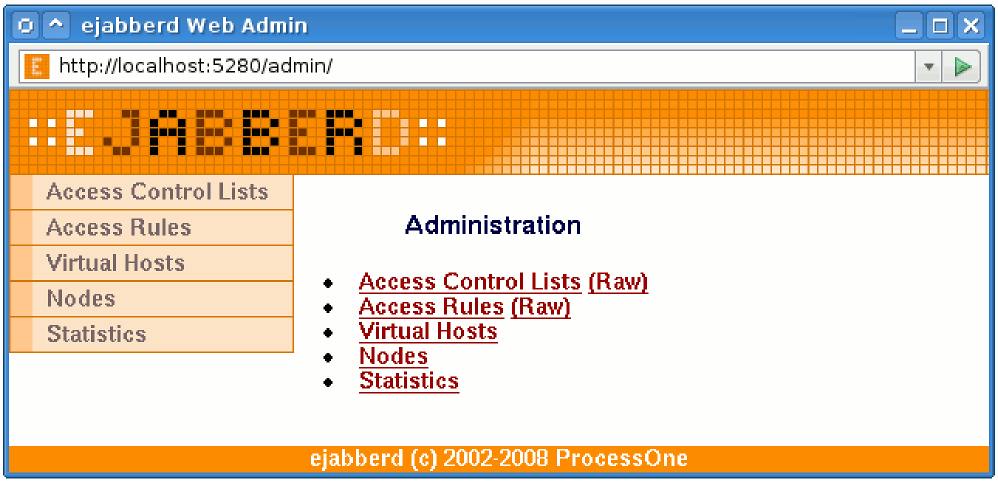

Figure 4.1: Top page from the Web Admin

- You can serve the Web Admin on the same port as the HTTP Polling interface. In this example you should point your web browser to `http://example.org:5280/admin/` to administer all virtual hosts or to `http://example.org:5280/admin/server/example.com/` to administer only the virtual host `example.com`. Before you get access to the Web Admin you need to enter as username, the JID and password from a registered user that is allowed to configure `ejabberd`. In this example you can enter as username '`admin@example.net`' to administer all virtual hosts (first URL). If you log in with '`admin@example.com`' on `http://example.org:5280/admin/server/example.com/` you can only administer the virtual host `example.com`.

```
{acl, admins, {user, "admin", "example.net"}}.
{host_config, "example.com", [{acl, admins, {user, "admin", "example.com"}}]}.
{access, configure, [{allow, admins}]}.

{hosts, ["example.org"]}.

{listen,
 [
  ...
  {5280, ejabberd_http, [http_poll, web_admin]},
  ...
 ]}.
```

- For security reasons, you can serve the Web Admin on a secured connection, on a port differing from the HTTP Polling interface, and bind it to the internal LAN IP. The Web Admin will be accessible by pointing your web browser to `https://192.168.1.1:5280/admin/`:

```
{hosts, ["example.org"]}.

{listen,
 [
```

```
  ...
  {5270, ejabberd_http,    [http_poll]},
  {5280, ejabberd_http,    [web_admin, {ip, {192, 168, 1, 1}},
                            tls, {certfile, "/usr/local/etc/server.pem"}]},
  ...
 ]}.
```

4.3 Ad-hoc Commands

If you enable `mod_configure` and `mod_adhoc`, you can perform several administrative tasks in `ejabberd` with a Jabber client. The client must support Ad-Hoc Commands (XEP-0050[2]), and you must login in the Jabber server with an account with proper privileges.

4.4 Change Computer Hostname

`ejabberd` uses the distributed Mnesia database. Being distributed, Mnesia enforces consistency of its file, so it stores the name of the Erlang node in it (see section 5.4). The name of an Erlang node includes the hostname of the computer. So, the name of the Erlang node changes if you change the name of the machine in which `ejabberd` runs, or when you move `ejabberd` to a different machine.

So, if you want to change the computer hostname where `ejabberd` is installed, you must follow these instructions:

1. In the old server, backup the Mnesia database using the Web Admin or `ejabberdctl`. For example:

   ```
   ejabberdctl backup /tmp/ejabberd-oldhost.backup
   ```

2. In the new server, restore the backup file using the Web Admin or `ejabberdctl`. For example:

   ```
   ejabberdctl restore /tmp/ejabberd-oldhost.backup
   ```

[2] http://www.xmpp.org/extensions/xep-0050.html

Chapter 5

Securing ejabberd

5.1 Firewall Settings

You need to take the following TCP ports in mind when configuring your firewall:

Port	Description
5222	Standard port for Jabber/XMPP client connections, plain or STARTTLS.
5223	Standard port for Jabber client connections using the old SSL method.
5269	Standard port for Jabber/XMPP server connections.
4369	Port used by EPMD for communication between Erlang nodes.
port range	Used for connections between Erlang nodes. This range is configurable.

5.2 epmd

epmd (Erlang Port Mapper Daemon)[1] is a small name server included in Erlang/OTP and used by Erlang programs when establishing distributed Erlang communications. `ejabberd` needs `epmd` to use `ejabberdctl` and also when clustering `ejabberd` nodes. This small program is automatically started by Erlang, and is never stopped. If `ejabberd` is stopped, and there aren't any other Erlang programs running in the system, you can safely stop `epmd` if you want.

`ejabberd` runs inside an Erlang node. To communicate with `ejabberd`, the script `ejabberdctl` starts a new Erlang node and connects to the Erlang node that holds `ejabberd`. In order for this communication to work, `epmd` must be running and listening for name requests in the port 4369. You should block the port 4369 in the firewall, so only the programs in your machine can access it.

If you build a cluster of several `ejabberd` instances, each `ejabberd` instance is called an `ejabberd` node. Those `ejabberd` nodes use a special Erlang communication method to build the cluster,

[1] http://www.erlang.org/doc/man/epmd.html

and EPMD is again needed listening in the port 4369. So, if you plan to build a cluster of `ejabberd` nodes you must open the port 4369 for the machines involved in the cluster. Remember to block the port so Internet doesn't have access to it.

Once an Erlang node solved the node name of another Erlang node using EPMD and port 4369, the nodes communicate directly. The ports used in this case are random. You can limit the range of ports when starting Erlang with a command-line parameter, for example:

```
erl ... -kernel inet_dist_listen_min 4370 inet_dist_listen_max 4375
```

5.3 Erlang Cookie

The Erlang cookie is a string with numbers and letters. An Erlang node reads the cookie at startup from the command-line parameter `-setcookie`. If not indicated, the cookie is read from the cookie file `$HOME/.erlang.cookie`. If this file does not exist, it is created immediately with a random cookie. Two Erlang nodes communicate only if they have the same cookie. Setting a cookie on the Erlang node allows you to structure your Erlang network and define which nodes are allowed to connect to which.

Thanks to Erlang cookies, you can prevent access to the Erlang node by mistake, for example when there are several Erlang nodes running different programs in the same machine.

Setting a secret cookie is a simple method to difficult unauthorized access to your Erlang node. However, the cookie system is not ultimately effective to prevent unauthorized access or intrusion to an Erlang node. The communication between Erlang nodes are not encrypted, so the cookie could be read sniffing the traffic on the network. The recommended way to secure the Erlang node is to block the port 4369.

5.4 Erlang node name

An Erlang node may have a node name. The name can be short (if indicated with the command-line parameter `-sname`) or long (if indicated with the parameter `-name`). Starting an Erlang node with -sname limits the communication between Erlang nodes to the LAN.

Using the option `-sname` instead of `-name` is a simple method to difficult unauthorized access to your Erlang node. However, it is not ultimately effective to prevent access to the Erlang node, because it may be possible to fake the fact that you are on another network using a modified version of Erlang `epmd`. The recommended way to secure the Erlang node is to block the port 4369.

5.5 Securing sensible files

`ejabberd` stores sensible data in the file system either in plain text or binary files. The file system permissions should be set to only allow the proper user to read, write and execute those files and directories.

`ejabberd configuration file: /etc/ejabberd/ejabberd.cfg` Contains the JID of administrators and passwords of external components. The backup files probably contain also this information, so it is preferable to secure the whole `/etc/ejabberd/` directory.

`ejabberd service log: /var/log/ejabberd/ejabberd.log` Contains IP addresses of clients. If the loglevel is set to 5, it contains whole conversations and passwords. If a logrotate system is used, there may be several log files with similar information, so it is preferable to secure the whole `/var/log/ejabberd/` directory.

`Mnesia database spool files: /var/lib/ejabberd/db/*` The files store binary data, but some parts are still readable. The files are generated by Mnesia and their permissions cannot be set directly, so it is preferable to secure the whole `/var/lib/ejabberd/db/` directory.

`Erlang cookie file: /var/lib/ejabberd/.erlang.cookie` See section 5.3.

Chapter 6

Clustering

6.1 How it Works

A Jabber domain is served by one or more `ejabberd` nodes. These nodes can be run on different machines that are connected via a network. They all must have the ability to connect to port 4369 of all another nodes, and must have the same magic cookie (see Erlang/OTP documentation, in other words the file `~ejabberd/.erlang.cookie` must be the same on all nodes). This is needed because all nodes exchange information about connected users, s2s connections, registered services, etc...

Each `ejabberd` node has the following modules:

- router,
- local router,
- session manager,
- s2s manager.

6.1.1 Router

This module is the main router of Jabber packets on each node. It routes them based on their destination's domains. It uses a global routing table. The domain of the packet's destination is searched in the routing table, and if it is found, the packet is routed to the appropriate process. If not, it is sent to the s2s manager.

6.1.2 Local Router

This module routes packets which have a destination domain equal to one of this server's host names. If the destination JID has a non-empty user part, it is routed to the session manager, otherwise it is processed depending on its content.

6.1.3 Session Manager

This module routes packets to local users. It looks up to which user resource a packet must be sent via a presence table. Then the packet is either routed to the appropriate c2s process, or stored in offline storage, or bounced back.

6.1.4 s2s Manager

This module routes packets to other Jabber servers. First, it checks if an opened s2s connection from the domain of the packet's source to the domain of the packet's destination exists. If that is the case, the s2s manager routes the packet to the process serving this connection, otherwise a new connection is opened.

6.2 Clustering Setup

Suppose you already configured `ejabberd` on one machine named (`first`), and you need to setup another one to make an `ejabberd` cluster. Then do following steps:

1. Copy `~ejabberd/.erlang.cookie` file from `first` to `second`.

   (alt) You can also add '`-cookie content_of_.erlang.cookie`' option to all '`erl`' commands below.

2. On `second` run the following command as the `ejabberd` daemon user, in the working directory of `ejabberd`:

   ```
   erl -sname ejabberd \
       -mnesia extra_db_nodes "['ejabberd@first']" \
       -s mnesia
   ```

   This will start Mnesia serving the same database as `ejabberd@first`. You can check this by running the command '`mnesia:info().`'. You should see a lot of remote tables and a line like the following:

   ```
   running db nodes   = [ejabberd@first, ejabberd@second]
   ```

3. Now run the following in the same '`erl`' session:

   ```
   mnesia:change_table_copy_type(schema, node(), disc_copies).
   ```

   This will create local disc storage for the database.

   (alt) Change storage type of the `scheme` table to 'RAM and disc copy' on the second node via the Web Admin.

4. Now you can add replicas of various tables to this node with '`mnesia:add_table_copy`' or '`mnesia:change_table_copy_type`' as above (just replace '`schema`' with another table name and '`disc_copies`' can be replaced with '`ram_copies`' or '`disc_only_copies`').

   Which tables to replicate is very dependant on your needs, you can get some hints from the command '`mnesia:info().`', by looking at the size of tables and the default storage type for each table on 'first'.

   Replicating a table makes lookups in this table faster on this node. Writing, on the other hand, will be slower. And of course if machine with one of the replicas is down, other replicas will be used.

   Also section 5.3 (Table Fragmentation) of Mnesia User's Guide[1] can be helpful.

   (alt) Same as in previous item, but for other tables.

5. Run '`init:stop().`' or just '`q().`' to exit from the Erlang shell. This probably can take some time if Mnesia has not yet transfered and processed all data it needed from `first`.

6. Now run `ejabberd` on `second` with almost the same config as on `first` (you probably do not need to duplicate '`acl`' and '`access`' options — they will be taken from `first`, and `mod_muc` and `mod_irc` should be enabled only on one machine in the cluster).

You can repeat these steps for other machines supposed to serve this domain.

6.3 Service Load-Balancing

6.3.1 Components Load-Balancing

6.3.2 Domain Load-Balancing Algorithm

`ejabberd` includes an algorithm to load balance the components that are plugged on an `ejabberd` cluster. It means that you can plug one or several instances of the same component on each `ejabberd` cluster and that the traffic will be automatically distributed.

The default distribution algorithm try to deliver to a local instance of a component. If several local instances are available, one instance is chosen randomly. If no instance is available locally, one instance is chosen randomly among the remote component instances.

If you need a different behaviour, you can change the load balancing behaviour with the option `domain_balancing`. The syntax of the option is the following:

```
{domain_balancing, "component.example.com", <balancing_criterium>}.
```

Several balancing criteria are available:

- `destination`: the full JID of the packet `to` attribute is used.

[1] http://www.erlang.org/doc/apps/mnesia/Mnesia_chap5.html#5.3

- `source`: the full JID of the packet `from` attribute is used.
- `bare_destination`: the bare JID (without resource) of the packet `to` attribute is used.
- `bare_source`: the bare JID (without resource) of the packet `from` attribute is used.

If the value corresponding to the criteria is the same, the same component instance in the cluster will be used.

6.3.3 Load-Balancing Buckets

When there is a risk of failure for a given component, domain balancing can cause service trouble. If one component is failing the service will not work correctly unless the sessions are rebalanced.

In this case, it is best to limit the problem to the sessions handled by the failing component. This is what the `domain_balancing_component_number` option does, making the load balancing algorithm not dynamic, but sticky on a fix number of component instances.

The syntax is the following:

```
{domain_balancing_component_number, "component.example.com", N}
```

Chapter 7

Debugging

7.1 Watchdog Alerts

ejabberd includes a watchdog mechanism. If a process in the ejabberd server consumes too much memory, a message is sent to the Jabber accounts defined with the option watchdog_admins in the ejabberd configuration file. Example configuration:

```
{watchdog_admins, ["admin2@localhost", "admin2@example.org"]}.
```

To remove watchdog admins, remove them in the option. To remove all watchdog admins, set the option with an empty list:

```
{watchdog_admins, []}.
```

7.2 Log Files

An ejabberd node writes two log files:

ejabberd.log is the ejabberd service log, with the messages reported by ejabberd code

sasl.log is the Erlang/OTP system log, with the messages reported by Erlang/OTP using SASL (System Architecture Support Libraries)

The option loglevel modifies the verbosity of the file ejabberd.log. The possible levels are:

0 No ejabberd log at all (not recommended)

1 Critical

2 Error

3 Warning

4 Info

5 Debug

For example, the default configuration is:

```
{loglevel, 4}.
```

7.3 Debug Console

The Debug Console is an Erlang shell attached to an already running `ejabberd` server. With this Erlang shell, an experienced administrator can perform complex tasks.

This shell gives complete control over the `ejabberd` server, so it is important to use it with extremely care. There are some simple and safe examples in the article Interconnecting Erlang Nodes[1]

To exit the shell, close the window or press the keys: control+c control+c.

[1] http://www.ejabberd.im/interconnect-erl-nodes

Appendix A

Internationalization and Localization

All built-in modules support the `xml:lang` attribute inside IQ queries. Figure A.1, for example, shows the reply to the following query:

```
<iq id='5'
    to='example.org'
    type='get'
    xml:lang='ru'>
  <query xmlns='http://jabber.org/protocol/disco#items'/>
</iq>
```

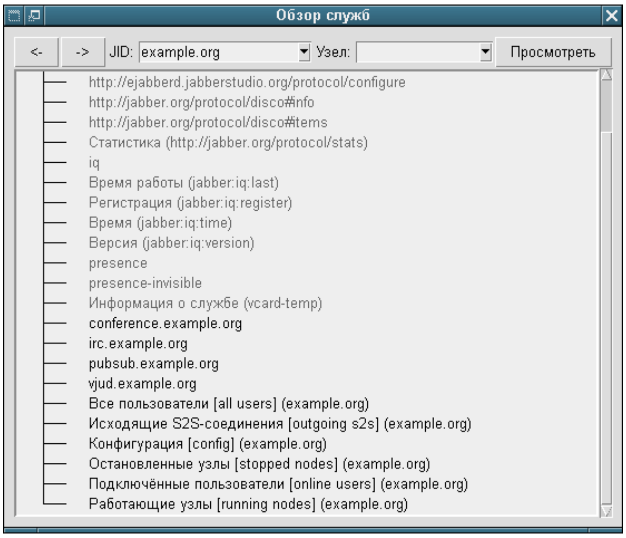

Figure A.1: Service Discovery when `xml:lang='ru'`

The Web Admin also supports the `Accept-Language` HTTP header.

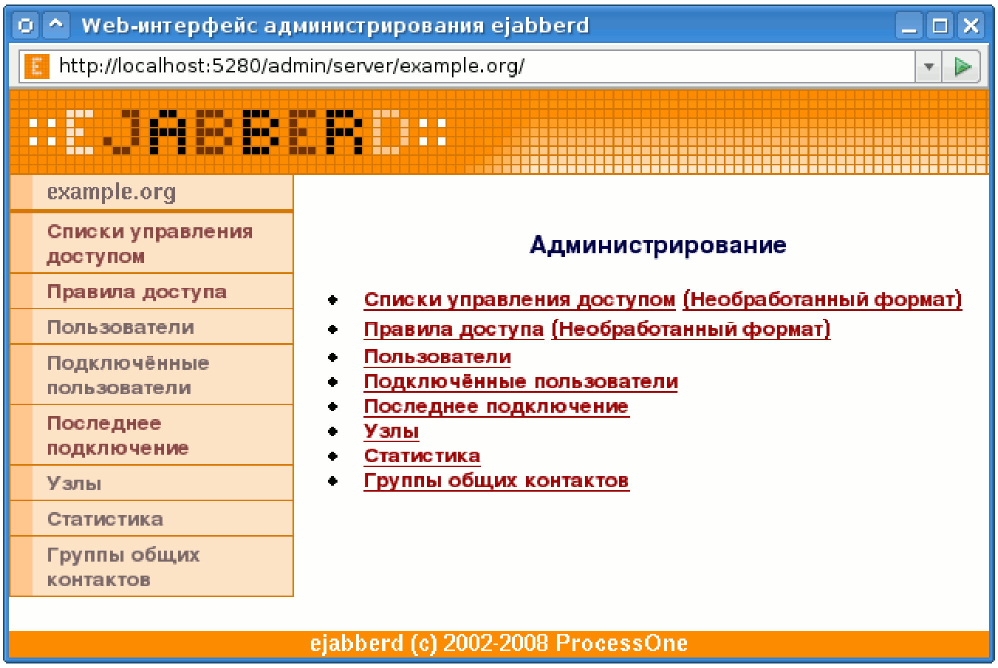

Figure A.2: Web Admin showing a virtual host when the web browser provides the HTTP header 'Accept-Language: ru'

Appendix B

Release Notes

Release notes are available from ejabberd Home Page[1]

[1] http://www.process-one.net/en/ejabberd/release_notes/

Appendix C

Acknowledgements

Thanks to all people who contributed to this guide:

- Alexey Shchepin (xmpp:aleksey@jabber.ru)
- Badlop (xmpp:badlop@jabberes.org)
- Evgeniy Khramtsov (xmpp:xram@jabber.ru)
- Florian Zumbiehl (xmpp:florz@florz.de)
- Michael Grigutsch (xmpp:migri@jabber.i-pobox.net)
- Mickael Remond (xmpp:mremond@process-one.net)
- Sander Devrieze (xmpp:s.devrieze@gmail.com)
- Sergei Golovan (xmpp:sgolovan@nes.ru)
- Vsevolod Pelipas (xmpp:vsevoload@jabber.ru)

Appendix D

Copyright Information

Index