Ejabberd Installation and Operation Guide

Alexey Shchepin
mailto:alexey@sevcom.net
xmpp:aleksey@jabber.ru

April 18, 2005

Contents

1 Introduction

`ejabberd` is a Free and Open Source fault-tolerant distributed Jabber server. It is written mostly in Erlang.

The main features of `ejabberd` are:

- Works on most of popular platforms: *nix (tested on Linux, FreeBSD and NetBSD) and Win32
- Distributed: You can run `ejabberd` on a cluster of machines to let all of them serve one Jabber domain.
- Fault-tolerance: You can setup an `ejabberd` cluster so that all the information required for a properly working service will be stored permanently on more than one node. This means that if one of the nodes crashes, then the others will continue working without disruption. You can also add or replace nodes "on the fly".
- Support for virtual hosting
- Built-in Multi-User Chat[1] service
- Built-in IRC transport
- Built-in Publish-Subscribe[2] service
- Built-in Jabber Users Directory service based on users vCards
- Built-in web-based administration interface
- Built-in HTTP Polling[3] service
- SSL support
- Support for LDAP authentication
- Ability to interface with external components (JIT, MSN-t, Yahoo-t, etc.)
- Migration from jabberd14 is possible
- Mostly XMPP-compliant
- Support for JEP-0030[4] (Service Discovery).
- Support for JEP-0039[5] (Statistics Gathering).
- Support for `xml:lang`

The misfeatures of `ejabberd` are:

- No support for authentication and STARTTLS in S2S connections
- Access rules can be defined only for global conext, not for specific virtual host

[1] `http://www.jabber.org/jeps/jep-0045.html`
[2] `http://www.jabber.org/jeps/jep-0060.html`
[3] `http://www.jabber.org/jeps/jep-0025.html`
[4] `http://www.jabber.org/jeps/jep-0030.html`
[5] `http://www.jabber.org/jeps/jep-0039.html`

2 Installation from Source

2.1 Installation Requirements

2.1.1 Unix

To compile `ejabberd`, you will need the following packages:

- GNU Make;
- GCC;
- libexpat 1.95 or later;
- Erlang/OTP R8B or later;
- OpenSSL 0.9.6 or later (optional).

2.1.2 Windows

To compile `ejabberd` in MS Windows environment, you will need the following packages:

- MS Visual C++ 6.0 Compiler
- Erlang/OTP R10B-1a[6]
- Expat 1.95.7[7]
- Iconv 1.9.1[8] (optional)
- Shining Light OpenSSL[9] (to enable SSL connections)

2.2 Obtaining

Stable `ejabberd` release can be obtained at http://www.jabberstudio.org/projects/ejabberd/releases/.

The latest alpha version can be retrieved from CVS.

```
export CVSROOT=:pserver:anonymous@jabberstudio.org:/home/cvs
cvs login
<press Enter when asked for a password>
cvs -z3 co ejabberd
```

[6]http://erlang.org/download/otp$_{w}in32_{R}10B - 1a.exe$
[7]http://prdownloads.sourceforge.net/expat/expat_win32bin_1_95_7.exe?download
[8]http://ftp.gnu.org/pub/gnu/libiconv/libiconv-1.9.1.tar.gz
[9]http://www.slproweb.com/products/Win32OpenSSL.html

2.3 Compilation

2.3.1 Unix

```
./configure
make
su
make install
```

This will install `ejabberd` to `/var/lib/ejabberd` directory, `ejabberd.cfg` to `/etc/ejabberd` directory and create `/var/log/ejabberd` directory for log files.

2.3.2 Windows

- Install Erlang emulator (for example, into `C:\Program Files\erl5.3`).
- Install Expat library into `C:\Program Files\Expat-1.95.7` directory.

  Copy file `C:\Program Files\Expat-1.95.7\Libs\libexpat.dll` to your Windows system directory (for example, `C:\WINNT` or `C:\WINNT\System32`)
- Build and install Iconv library into `C:\Program Files\iconv-1.9.1` directory.

  Copy file `C:\Program Files\iconv-1.9.1\bin\iconv.dll` to your Windows system directory.

  Note: Instead of copying libexpat.dll and iconv.dll to Windows directory, you can add directories `C:\Program Files\Expat-1.95.7\Libs` and `C:\Program Files\iconv-1.9.1\bin` to `PATH` environment variable.
- Being in `ejabberd\src` directory run:

  ```
  configure.bat
  nmake -f Makefile.win32
  ```
- Edit file `ejabberd\src\ejabberd.cfg` and run

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

2.4 Starting

To start `ejabberd`, use the following command:

```
erl -pa /var/lib/ejabberd/ebin -name ejabberd -s ejabberd
```

or

```
erl -pa /var/lib/ejabberd/ebin -sname ejabberd -s ejabberd
```

In the latter case Erlang node will be identified using only first part of host name, i. e. other Erlang nodes outside this domain can't contact this node.

Note that when using above command `ejabberd` will search for config file in current directory and will use current directory for storing user database and logging.

To specify path to config file, log files and Mnesia database directory, you may use the following command:

```
erl -pa /var/lib/ejabberd/ebin \
    -sname ejabberd \
    -s ejabberd \
    -ejabberd config \"/etc/ejabberd/ejabberd.cfg\" \
              log_path \"/var/log/ejabberd/ejabberd.log\" \
    -sasl sasl_error_logger \{file,\"/var/log/ejabberd/sasl.log\"\} \
    -mnesia dir \"/var/lib/ejabberd/spool\"
```

You can find other useful options in Erlang manual page (`erl -man erl`).

To use more than 1024 connections, you should set environment variable `ERL_MAX_PORTS`:

```
export ERL_MAX_PORTS=32000
```

Note that with this value `ejabberd` will use more memory (approximately 6MB more).

To reduce memory usage, you may set environment variable `ERL_FULLSWEEP_AFTER`:

```
export ERL_FULLSWEEP_AFTER=0
```

But in this case `ejabberd` can start to work slower.

3 Configuration

3.1 Initial Configuration

The configuration file is initially loaded the first time `ejabberd` is executed, when it is parsed and stored in a database. Subsequently the configuration is loaded from the database and any commands in the configuration file are appended to the entries in the database. The configuration file consists of a sequence of Erlang terms. Parts of lines after '`%`' sign are ignored. Each term is tuple, where first element is name of option, and other are option values. E. g. if this file does not contain a "host" definition, then old value stored in the database will be used.

To override old values stored in the database the following lines can be added in config:

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

With this lines old global or local options or ACLs will be removed before adding new ones.

3.1.1 Host Names

Option `hosts` defines a list of Jabber domains that `ejabberd` serves. E. g. to serve `example.org` and `example.com` domains add the following line in the config:

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

Option `host` defines one Jabber domain that `ejabberd` serves. E. g. to serve only `example.org` domain add the following line in the config:

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

It have the same effect as

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

3.1.2 Default Language

Option `language` defines default language of `ejabberd` messages, sent to users. Default value is `"en"`. In order to take effect there must be a translation file `<language>.msg` in `ejabberd` `msgs` directory. E. g. to use Russian as default language add the following line in the config:

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

3.1.3 Access Rules

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

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

`<acltype>` can be one of following:

`all` Matches all JIDs. Example:

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

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

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

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

```
{acl, admin, {user, "aleksey", "jabber.ru"}}.
```

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

```
{acl, jabberorg, {server, "jabber.org"}}.
```

`{user_regexp, <regexp>}` Matches local user with name that matches `<regexp>` at the first virtual host. Example:

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

`{user_regexp, <regexp>, <server>}` Matches user with name that matches `<regexp>` and from server `<server>`. Example:

```
{acl, tests, {user, "^test", "localhost"}}.
```

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

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

`{node_regexp, <user_regexp>, <server_regexp>}` Matches user with name that matches `<user_regexp>` and from server that matches `<server_regexp>`. Example:

```
{acl, aleksey, {node_regexp, "^aleksey$", "^jabber.(ru|org)$"}}.
```

`{user_glob, <glob>}`

`{user_glob, <glob>, <server>}`

`{server_glob, <glob>}`

`{node_glob, <user_glob>, <server_glob>}` This is same as above, but uses shell glob patterns instead of regexp. These patterns can have 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 `‘!’`, then any character not enclosed is matched.

The following ACLs are pre-defined:

`all` Matches all JIDs.

`none` Matches none JIDs.

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

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

When a JID is checked to have access to `<accessname>`, the server sequentially checks if this JID mathes one of the ACLs that are second elements in each tuple in list. If it is matched, then the first element of matched tuple is returned else "`deny`" is returned.

Example:

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

Following access rules pre-defined:

`all` Always returns "`allow`"

`none` Always returns "`deny`"

3.1.4 Shapers Configuration

With shapers is possible to bound connection traffic. The declarations of shapers in config file have following syntax:

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

Currently implemented only one kind of shaper: `maxrate`. It have following syntax:

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

where `<rate>` means maximum allowed incomig rate in bytes/second. E. g. to define shaper with name "`normal`" and maximum allowed rate 1000 bytes/s, add following line in config:

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

3.1.5 Listened Sockets

Option `listen` defines list of listened sockets and what services runned on them. Each element of list is a tuple with following elements:

- Port number;

- Module that serves this port;
- Options to this module.

Currently these modules are implemented:

`ejabberd_c2s` This module serves C2S connections.

The following options are defined:

`{access, <access rule>}` This option defines access of users to this C2S port. Default value is "`all`".

`{shaper, <access rule>}` This option is like previous, but use shapers instead of "`allow`" and "`deny`". Default value is "`none`".

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

`inet6` Set up the socket for IPv6.

`starttls` This option specifies that STARTTLS extension is available on connections to this port. You should also set "`certfile`" option.

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

`ssl` This option specifies that traffic on this port will be encrypted using SSL. You should also set "`certfile`" option. It is recommended to use `tls` option instead.

`{certfile, Path}` Path to a file containing the SSL certificate.

`ejabberd_s2s_in` This module serves incoming S2S connections.

`ejabberd_service` This module serves connections from Jabber services (i. e. that use the `jabber:component:accept` namespace).

The following additional options are defined for `ejabberd_service` (options `access`, `shaper`, `ip`, `inet6` are still valid):

`{host, Hostname, [HostOptions]}` This option defines hostname of connected service and allows to specify additional options, e. g. `{password, Secret}`.

`{hosts, [Hostnames], [HostOptions]}` The same as above, but allows to specify several hostnames.

`ejabberd_http` This module serves incoming HTTP connections.

The following options are defined:

`http_poll` This option enables HTTP Polling[10] support. It is available then at `http://server:port/http-pol`

`web_admin` This option enables web-based interface for `ejabberd` administration which is available at `http://server:port/admin/`, login and password should be equal to username and password of one of registered users who have permission defined in "configure" access rule.

For example, the following configuration defines that:

[10] http://www.jabber.org/jeps/jep-0025.html

- C2S connections are listened on port 5222 and 5223 (SSL) and denied for user "`bad`"
- S2S connections are listened on port 5269
- HTTP connections are listened on port 5280 and administration interface and HTTP Polling support are enabled
- All users except admins have traffic limit 1000 B/s
- AIM transport `aim.example.org` is connected to port 5233 with password "`aimsecret`"
- JIT transports `icq.example.org` and `sms.example.org` are connected to port 5234 with password "`jitsecret`"
- MSN transport `msn.example.org` is connected to port 5235 with password "`msnsecret`"
- Yahoo! transport `yahoo.example.org` is connected to port 5236 with password "`yahoosecret`"
- Gadu-Gadu transport `gg.example.org` is connected to port 5237 with password "`ggsecret`"
- ILE service `ile.example.org` is connected to port 5238 with password "`ilesecret`"

```
{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, [{host, "aim.example.org",
                              [{password, "aimsecret"}]}]},
  {5234, ejabberd_service, [{hosts, ["icq.example.org", "sms.example.org"],
                              [{password, "jitsecret"}]}]},
  {5235, ejabberd_service, [{host, "msn.example.org",
                              [{password, "msnsecret"}]}]},
  {5236, ejabberd_service, [{host, "yahoo.example.org",
                              [{password, "yahoosecret"}]}]},
  {5237, ejabberd_service, [{host, "gg.example.org",
                              [{password, "ggsecret"}]}]},
  {5238, ejabberd_service, [{host, "ile.example.org",
                              [{password, "ilesecret"}]}]}
 ]
}.
```

Note, that for jabberd14- or wpjabberd-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 jabberd 2.0 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>
```

3.1.6 Modules

Option `modules` defines the list of modules that will be loaded after `ejabberd` startup. Each list element is a tuple where first element is a name of a module and second is list of options to this module. See section **??** for detailed information on each module.

Example:

```
{modules,
 [{mod_register,  []},
  {mod_roster,    []},
  {mod_privacy,   []},
  {mod_configure, []},
  {mod_disco,     []},
  {mod_stats,     []},
  {mod_vcard,     []},
  {mod_offline,   []},
  {mod_announce,  [{access, announce}]},
  {mod_echo,      [{host, "echo.localhost"}]},
  {mod_private,   []},
  {mod_irc,       []},
  {mod_muc,       []},
  {mod_pubsub,    []},
  {mod_time,      [{iqdisc, no_queue}]},
  {mod_last,      []},
  {mod_version,   []}
 ]}.
```

3.2 Online Configuration and Monitoring

3.2.1 Web-based Administration Interface

To perform online reconfiguration of `ejabberd` you need to enable `ejabberd_http` listener with option `web_admin` (see section ??). After that you can open URL `http://server:port/admin/` with you favorite web-browser and enter username and password of an `ejabberd` user with administrator rights. E. g. with such config:

```
  ...
  {host, "example.org"}.
  ...
  {listen,
   [...
    {5280, ejabberd_http, [web_admin]},
    ...
   ]
  }.
```

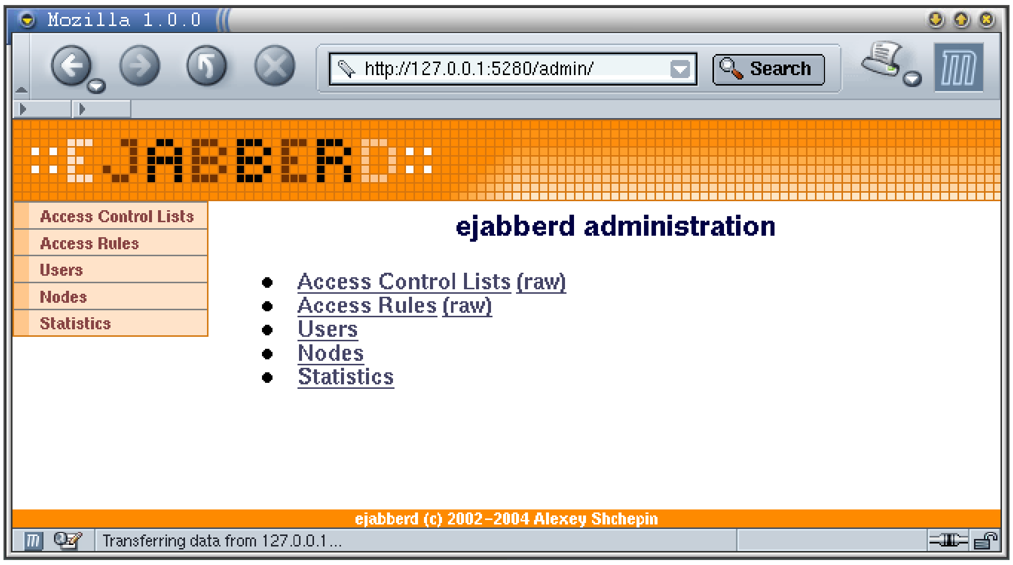
you should enter URL `http://example.org:5280/admin/`. After authentication you should see something like in figure ??. Here you can edit access restrictions, manage users, create backup files, manage DB, enable/disable listened ports, and view statistics.

3.2.2 `ejabberdctl` tool

It is possible to do some administration operations using `ejabberdctl` command-line tool. You can check available options running this command without arguments:

```
% ejabberdctl
Usage: ejabberdctl node command

Available commands:
  stop                          stop ejabberd
  restart                       restart ejabberd
  reopen-log                    reopen log file
  register user password        register a user
```



ejabberd administration

- Access Control Lists (raw)
- Access Rules (raw)
- Users
- Nodes
- Statistics

ejabberd (c) 2002–2004 Alexey Shchepin

Transferring data from 127.0.0.1...

Figure 1: Web-administration top page

`unregister user`	`unregister a user`
`backup file`	`store a database backup in file`
`restore file`	`restore a database backup from file`
`install-fallback file`	`install a database fallback from file`
`dump file`	`dump a database in a text file`
`load file`	`restore a database from a text file`
`registered-users`	`list all registered users`

```
Example:
  ejabberdctl ejabberd@host restart
```

4 Clustering

4.1 How it works

A Jabber domain is served by one or more `ejabberd` nodes. These nodes can be runned 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 have following modules:

- router;

- local router.
- session manager;
- S2S manager;

4.1.1 Router

This module is the main router of Jabber packets on each node. It routes them based on their destinations domains. It uses a global routing table. A domain of packet destination is searched in the routing table, and if it is found, then the packet is routed to appropriate process. If no, then it is sent to the S2S manager.

4.1.2 Local Router

This module routes packets which have a destination domain equal to this server name. If destination JID has a non-empty user part, then it is routed to the session manager, else it is processed depending on its content.

4.1.3 Session Manager

This module routes packets to local users. It searches to what user resource a packet must be sent via a presence table. Then packet is either routed to appropriate C2S process, or stored in offline storage, or bounced back.

4.1.4 S2S Manager

This module routes packets to other Jabber servers. First, it checks if an opened S2S connection from the domain of the packet source to the domain of packet destination is existing. If it is existing, then the S2S manager routes the packet to the process serving this connection, else a new connection is opened.

4.2 How to setup ejabberd cluster

Suppose you already setuped ejabberd on one of machines (`first`), and you need to setup another one to make `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 under '`ejabberd`' user in a directory where ejabberd will work later the following command:

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

This will start mnesia serving same DB as `ejabberd@first`. You can check this running "`mnesia:info().`" command. 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 DB.

   (alt) Change storage type of '`scheme`' table to "RAM and disc copy" on second node via web interface.

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`").

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

   Replicating of table makes lookup in this table faster on this node, but writing will be slower. And of course if machine with one of replicas is down, other replicas will be used.

   Also section "5.3 Table Fragmentation" here[11] can be useful.

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

5. Run "`init:stop().`" or just "`q().`" to exit from erlang shell. This probably can take some time if mnesia is not yet transfer and process all data it needed from `first`.

6. Now run ejabberd on `second` with almost the same config as on `first` (you probably don't 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 cluster).

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

A Built-in Modules

A.1 Common Options

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

[11] `http://www.erlang.se/doc/doc-5.4/lib/mnesia-4.2/doc/html/index.html`

A.1.1 `iqdisc`

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

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

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

`parallel` In this case for all packets with this discipline spawned separate Erlang process, so all these packets processed in parallel. Although spawning of Erlang process have relatively low cost, this can broke server normal work, because Erlang emulator have limit on number of processes (32000 by default).

Example:

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

A.1.2 `host`

This option explicitly defines hostname for the module which acts as a service.

Example:

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

A.1.3 `hosts`

This option explicitly defines a list of hostnames for the module which acts as a service.

Example:

```
{modules,
 [
  ...
  {mod_echo, [{hosts, ["echo.example.org", "echo.example.com"]}]},
  ...
 ]}.
```

A.2 mod_announce

This module adds support for broadcast announce messages and MOTD. When the module is loaded, it handles messages sent to the following JID's (suppose that main server has address `example.org`):

`example.org/announce/all` Message is sent to all registered users at `example.org`. If the user is online and connected to several resources, only resource with the highest priority will receive the message. If the registered user is not connected, the message will be stored offline (if offline storage is available).

`example.org/announce/online` Message is sent to all connected users at `example.org`. If the user is online and connected to several resources, all resources will receive the message.

`example.org/announce/all-hosts/online` Message is sent to all connected users at every virtual host. If the user is online and connected to several resources, all resources will receive the message.

`example.org/announce/motd` Message is set as MOTD (Message of the Day) and is sent to users at `example.org` as they login. In addition the message is sent to all connected users (similar to `announce/online` resource).

`example.org/announce/motd/update` Message is set as MOTD (Message of the Day) and is sent to users at `example.org` as they login. The message is *not sent* to all connected users.

`example.org/announce/motd/delete` Any message sent to this JID removes existing MOTD.

Options:

`access` Specifies who is allowed to send announce messages and set MOTD (default value is `none`).

Example:

```
% Only admins can send announcement messages:
{access, announce, [{allow, admin}]}.

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

A.3 `mod_configure`

Options:

`iqdisc` `ejabberd:config` IQ queries processing discipline (see ??).

A.4 `mod_disco`

This module adds support for JEP-0030[12] (Service Discovery).

Options:

`iqdisc` `http://jabber.org/protocol/disco#items` and `http://jabber.org/protocol/disco#info` IQ queries processing discipline (see ??).

`extra_domains` List of domains that will be added to server items reply

Example:

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

A.5 `mod_echo`

This module acts as a service and simply returns to sender any Jabber packet. Module may be useful for debugging.

Options:

`host` Defines hostname of the service (see ??).

`hosts` Defines hostnames of the service (see ??). If neither `host` nor `hosts` are not present, then prefix `echo.` is added to all `ejabberd` hostnames.

[12] http://www.jabber.org/jeps/jep-0030.html

A.6 mod_irc

This module implements IRC transport.

Options:

`host` Defines hostname of the service (see **??**).

`hosts` Defines hostnames of the service (see **??**). If neither `host` nor `hosts` are not present, then prefix `irc.` is added to all `ejabberd` hostnames.

`access` Specifies who is allowed to use IRC transport (default value is `all`).

Example:

```
{modules,
 [
  ...
  {mod_irc, [{access, all}]},
  ...
 ]}.
```

A.7 mod_last

This module adds support for JEP-0012[13] (Last Activity)

Options:

`iqdisc` `jabber:iq:last` IQ queries processing discipline (see **??**).

A.8 mod_muc

This module implements JEP-0045[14] (Multi-User Chat) service.

Options:

`host` Defines hostname of the service (see **??**).

`hosts` Defines hostnames of the service (see **??**). If neither `host` nor `hosts` are not present, then prefix `conference.` is added to all `ejabberd` hostnames.

`access` Specifies who is allowed to use MUC service (default value is `all`).

`access_create` Specifies who is allowed to create new rooms at MUC service (default value is `all`).

[13]http://www.jabber.org/jeps/jep-0012.html
[14]http://www.jabber.org/jeps/jep-0045.html

access_admin Specifies who is allowed to administrate MUC service (default value is `none`, which means that only creator may administer her room).

Example:

```
% Define admin ACL
{acl, admin, {user, "admin"}}

% Define MUC admin access rule
{access, muc_admin, [{allow, admin}]}

{modules,
 [
  ...
  {mod_muc, [{access, all},
             {access_create, all},
             {access_admin, muc_admin}]},
  ...
 ]}.
```

A.9 mod_offline

This module implements offline message storage.

A.10 mod_privacy

This module implements Privacy Rules as defined in XMPP IM (see http://www.jabber.org/ietf/).

Options:

iqdisc `jabber:iq:privacy` IQ queries processing discipline (see ??).

A.11 mod_private

This module adds support of JEP-0049[15] (Private XML Storage).

Options:

iqdisc `jabber:iq:private` IQ queries processing discipline (see ??).

[15] http://www.jabber.org/jeps/jep-0049.html

A.12 mod_pubsub

This module implements JEP-0060[16] (Publish-Subscribe Service).

Options:

`host` Defines hostname of the service (see ??).

`hosts` Defines hostnames of the service (see ??). If neither `host` nor `hosts` are not present, then prefix `pubsub.` is added to all `ejabberd` hostnames.

`served_hosts` Specifies which hosts are served by the service. If absent then only main `ejabberd` host is served.

Example:

```
{modules,
 [
  ...
  {mod_pubsub, [{served_hosts, ["example.com",
                                "example.org"]}]}
  ...
 ]}.
```

A.13 mod_register

This module adds support for JEP-0077[17] (In-Band Registration).

Options:

`access` Specifies rule to restrict registration. If this rule returns "deny" on requested user name, then registration is not allowed for it. (default value is `all`, which means no restrictions).

`iqdisc` `jabber:iq:register` IQ queries processing discipline (see ??).

Example:

```
% Deny registration for users with too short name
{acl, shortname, {user_glob, "?"}}.
{acl, shortname, {user_glob, "??"}}.
% Another variant: {acl, shortname, {user_regexp, "^..?$"}}.

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

[16] http://www.jabber.org/jeps/jep-0060.html
[17] http://www.jabber.org/jeps/jep-0077.html

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

A.14 mod_roster

This module implements roster management.

Options:

`iqdisc` `jabber:iq:roster` IQ queries processing discipline (see ??).

A.15 mod_service_log

This module adds support for logging of user packets via any jabber service. These packets encapsulated in ¡route/¿ element and sended to specified services.

Options:

`loggers` Specifies a list of services which will receive users packets.

Example:

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

A.16 mod_stats

This module adds support for JEP-0039[18] (Statistics Gathering).

Options:

`iqdisc` `http://jabber.org/protocol/stats` IQ queries processing discipline (see ??).

[18] http://www.jabber.org/jeps/jep-0039.html

A.17 mod_time

This module answers UTC time on `jabber:iq:time` queries.

Options:

`iqdisc` `jabber:iq:time` IQ queries processing discipline (see ??).

A.18 mod_vcard

This module implements simple Jabber User Directory (based on user vCards) and answers server vCard on `vcard-temp` queries.

Options:

`host` Defines hostname of the service (see ??).

`hosts` Defines hostnames of the service (see ??). If neither `host` nor `hosts` are not present, then prefix `vjud.` is added to all `ejabberd` hostnames.

`iqdisc` `vcard-temp` IQ queries processing discipline (see ??).

`search` Specifies whether search is enabled (value is `true`, default) or disabled (value is `false`) by the service. If `search` is set to `false`, option `host` is ignored and service does not appear in Jabber Discovery items.

`matches` Limits the number of reported search results. If value is set to `infinity` then all search results are reported. Default value is `30`.

`allow_return_all` Specifies whether search with empty input fields can return all known users. Default is `false`.

`search_all_hosts` If set in `true` then search returns matched items at all virtual hosts. Otherwise only current host items are returned. Default is `true`.

Example:

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

A.19 `mod_version`

This module answers `ejabberd` version on `jabber:iq:version` queries.

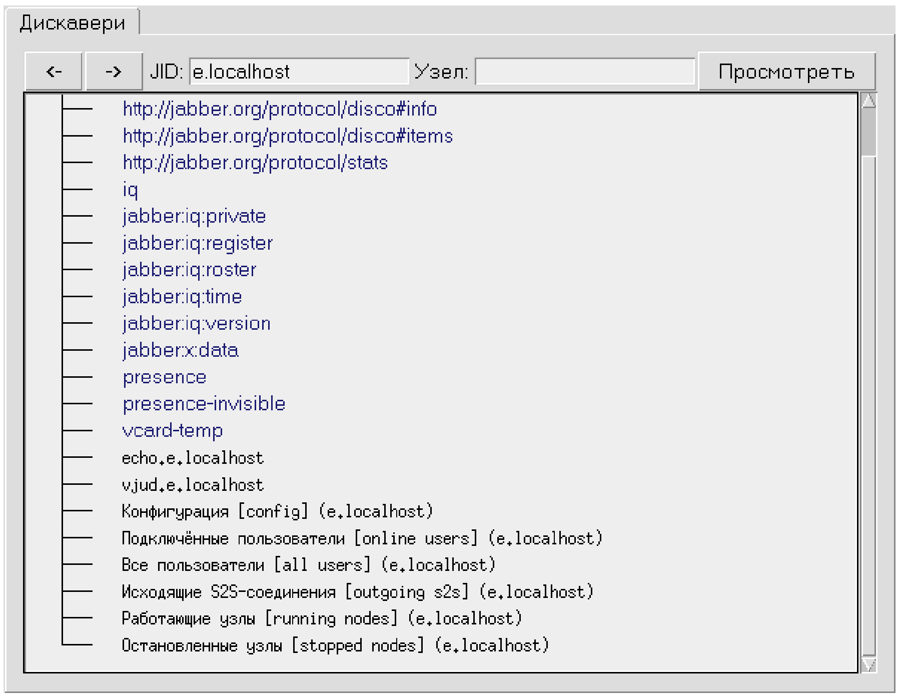
Options:

`iqdisc` `jabber:iq:version` IQ queries processing discipline (see ??).

B I18n/L10n

All built-in modules support `xml:lang` attribute inside IQ queries. E. g. on figure ?? showed the reply on the following query:

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



http://jabber.org/protocol/disco#info
http://jabber.org/protocol/disco#items
http://jabber.org/protocol/stats
iq
jabber:iq:private
jabber:iq:register
jabber:iq:roster
jabber:iq:time
jabber:iq:version
jabber:x:data
presence
presence-invisible
vcard-temp
echo.e.localhost
vjud.e.localhost
Конфигурация [config] (e.localhost)
Подключённые пользователи [online users] (e.localhost)
Все пользователи [all users] (e.localhost)
Исходящие S2S-соединения [outgoing s2s] (e.localhost)
Работающие узлы [running nodes] (e.localhost)
Остановленные узлы [stopped nodes] (e.localhost)

Figure 2: Discovery result when `xml:lang='ru'`

Also web-interface supports `Accept-Language` HTTP header (see figure ??, compare it with figure ??)

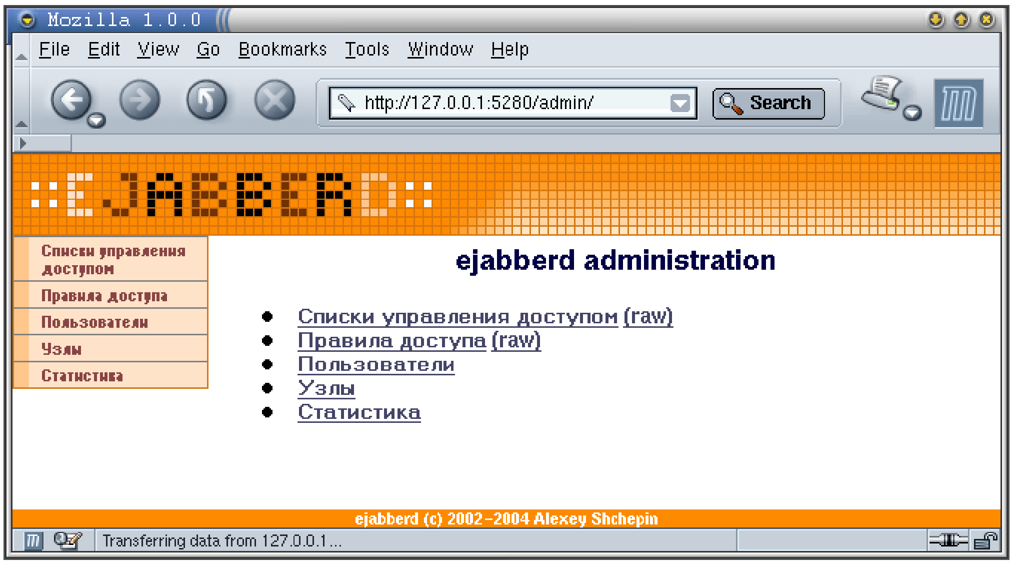

Mozilla 1.0.0
File Edit View Go Bookmarks Tools Window Help
http://127.0.0.1:5280/admin/
Search
::EJABBERD::
Списки управления доступом
Правила доступа
Пользователи
Узлы
Статистика
ejabberd administration
Списки управления доступом (raw)
Правила доступа (raw)
Пользователи
Узлы
Статистика
ejabberd (c) 2002-2004 Alexey Shchepin
Transferring data from 127.0.0.1...

Figure 3: Web-administration top page with HTTP header “`Accept-Language: ru`|’’