ejabberd Docs

ejabberd Community Server

Table of contents

Overview

Getting started 👋

Meet **ejabberd**, your superpowerful messaging framework

This web site is dedicated to help you use and develop for ejabberd XMPP messaging server.

ejabberd has been in development since 2002 and is used all over the world to power the largest XMPP deployments. This project is so versatile that you can deploy it and customize it for very large scale, no matter what your use case is.

This incredible power comes with a price. You need to learn how to leverage it. Fortunately, the goal of this website is to get you started on your path to mastery. Whether you are a sysadmin, an architect, a developer planning to extend it, or even a simple XMPP user, we have something for you here.

Overview

ejabberd is the de facto XMPP server in the world. The fact that it is used to power the largest deployments in the world should not intimidate you. ejabberd is equally suitable for small instances.

ejabberd has been designed from the ground-up, since 2002 for robust, enterprise deployment. The goal has always been to shoot for the moon and this is what made it a long-lasting success.

ejabberd is specifically designed for enterprise purposes: it is fault-tolerant, can utilise the resources of multiple clustered machines, and can easily scale when more capacity is required (by just adding a box/VM).

Designed at a moment when clients were mostly desktops that only supported a kind of HTTP polling call BOSH, the project managed to adapt to recent changes by introducing support for WebSockets, BOSH improvements, and a solid mobile stack.

It was developed at a time when XMPP was still known as "Jabber", but quickly adopted an evolution process in order to support the various versions of XMPP RFCs. It now encourages innovation and experimentation by supporting most, if not all, extensions produced by the XSF.

ejabberd relies on a dynamic community all over the world. To get an idea of existing contributions, you can check ejabberd main repository or the repository containing a great amount of contributed extensions.

This is possible thanks to a modular architecture based on a core router and an extremely powerful plugin mechanism that is getting richer every day.

Welcome to the beginning of your journey of ejabberd mastery!

Options to use ejabberd

ejabberd can be used in different ways. The most common one is to use ejabberd Community Edition. This is the standard Open Source version that everyone loves: highly scalable and flexible.

Fortunately, if you need more than just the ejabberd platform software, ProcessOne can help you with a commercial offering. Commercial offering come in two type of packaging:

- **ejabberd Business Edition**, including features for large companies (enhanced geodistributed companies and mobile support to develop own, rich clients) and world-class support, that can please even the most demanding businesses, with 24/7 options.

- Fluux.io being a way to access and benefit of all the features of ejabberd Business Edition at an attractive and scalable price. Fluux.io allows you to keep control of your data thanks to integration API you can implement on your backend to become a data provider for ejabberd SaaS.

Whatever approach you choose, you can hardly make the wrong choice with ejabberd! In every case you can easily integrate ejabberd with your existing application using:

- REST API and ejabberdctl command-line tool
- Mobile libraries for iOS: XMPPFramework, Jayme REST API
- Mobile libraries for Android: Smack, Retrofit
- Web library with WebSocket support and fallback to BOSH: Strophe

Architecture of an ejabberd service

ejabberd brings configurability, scalability and fault-tolerance to the core feature of XMPP – routing messages.

Its architecture is based on a set of pluggable modules that enable different features, including:

- One-to-one messaging
- Store-and-forward (offline messages)
- Contact list (roster) and presence
- Groupchat: MUC (Multi-User Chat)
- Messaging archiving with Message Archive Management (MAM)
- User presence extension: Personal Event Protocol (PEP) and typing indicator
- Privacy settings, through privacy list and simple blocking extensions
- User profile with vCards
- Full feature web support, with BOSH and websockets
- Stream management for message reliability on mobile (aka XEP-0198)
- Message Delivery Receipts (aka XEP-184)
- Last activity
- Metrics and full command-line administration
- and many many more.

The full list of supported protocol and extensions is available on Protocols Supported by ejabberd page.

This modular architecture allows high customisability and easy access to the required features.

ejabberd enables authenticating users using external or internal databases (Mnesia, SQL), LDAP or external scripts. It also allows connecting anonymous users, when required.

For storing persistent data, ejabberd uses Mnesia (the distributed internal Erlang database), but you can opt for SQL database like MySQL or PostgreSQL

And of course, thanks to its API, ejabberd can be customised to work with a database chosen by the customer.

Deploying and managing an ejabberd service

ejabberd can be deployed for a number of scenarios fitting end-user / developer / customer needs. The default installation setup consists of a single ejabberd node using Mnesia, so it does not require any additional configuration. This primary system is sufficient for fast deployment and connecting XMPP clients. It should be good enough for most of the small deployments (and even medium ones).

A more scalable solution would be deploying ejabberd with an external database for persistent data. As Mnesia is caching part of its data in ejabberd memory (actually in Erlang VM node), this kind of setup make your system more scalable and typically easier to integrate with your usual database. As a sysadmin, yes, you can use your standard backup process.

Those larger setup can run as a cluster of ejabberd nodes. This is a clustering mode where all nodes are active, so it can be use for fault-tolerance, but also to increase the capacity of your ejabberd deployment.

With such a deployment you can load balance the traffic to your cluster node using one of the following solution:

- traditional TCP/IP load balancer (beware of the cost of your solution, typical XMPP connections are persistent).
- DNS load balancing.
- Custom approach that requires client cooperation.

If deployed on a 16 GB RAM machine with at least 4 cores, a single ejabberd node can typically handle 200-300 K online users. This setup is suitable for systems with up to 10 nodes.

Note that your mileage may vary depending on your use case, the feature your are using and how clean the architecture design and the client is developed. That's why, if you plan to reach huge volume, it is recommended to start asking advices from day 1 to an ejabberd expert. Initial mistakes in the solution design are harder to fix once the project is in production.

If the service requires a cluster of more than 10 nodes, we recommend not relying on Mnesia clustering mode. Many solutions are available, the easiest and more inexpensive being to rely on ejabberd Software-as-a-Service approach.

ejabberd also allows connecting different clusters as parts of larger systems. This is a standard XMPP feature call server-to-server (aka s2s in XMPP lingo). It is used in geo-localised services handling massive traffic from all over the world. Special extension are also available from ProcessOne to handle geodistribution in an even more robust way.

To manage the users, rosters, messages and general settings, we provide a command-line tool, ejabberdctl. That command-line allows you to gather metrics from ejabberd to be able to monitor what is happening in your system, understand and even anticipate issues.

The main benefit of ejabberd is the ability to reach a command-line to type Erlang commands. This allows you to fix and troubleshoot most of the tricky situation and even update and reload code without stopping the service. This is a life saver for your uptime.

Welcome to the benefit of Erlang hot-code swapping!

ejabberd is more than XMPP

Thanks to the modular architecture of ejabberd, the platform is becoming a core component for messaging applications.

Messaging applications require to transfer more than text messages. ejabberd has grow a full set of media related features that makes ejabberd a great choice to support voice and video applications, but also to proxy various kind of media transfer (images, audio and video files for example).

As such, ejabberd support:

- Jingle, XMPP based voice protocol
- SIP (Session Initiation Protocol): Yes, you can pass SIP calls using ejabberd :)
- ICE (Interactive Connectivity Establishment: A Protocol for Network Address Translator (NAT) Traversal)
- STUN
- TURN
- Proxy65 media relay

This makes ejabberd the best XMPP server to support SIP and WebRTC based communication tools.

Helping us in the development process

With thousands of more or less official forks, the core ejabberd team, supported by ProcessOne, is constantly monitoring and reviewing improvements. We use our 15 years of experience to filter the best ideas or improvements to make sure ejabberd is always your most solid choice in term of scalability, robustness and manageability.

The best way to start developing for ejabberd is to clone, watch and star the project, to get in touch on our developer chatroom (ejabberd@conference.process-one.net) or to join ejabberd community on StackOverflow.

Features

`ejabberd` is a *free and open source* instant messaging server written in `Erlang/OTP` .

`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 large deployments.

Check also the features in ejabberd.im, ejabberd at ProcessOne, and the list of supported protocols in ProcessOne and XMPP.org.

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 all serving 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 can be added or replaced on the fly.
- *Administrator Friendly:* `ejabberd` is built on top of the Erlang programming language. As a result, if you wish, you can perform self-contained deployments. You are not required 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 and is well suited to build services available across the world. Related features are:
- Translated to 25 languages.
- Support for `IDNA` .
- *Open Standards:* `ejabberd` is the first Open Source Jabber server staking a claiming to full complyiance to the XMPP standard.
- Fully XMPP compliant.
- XML-based protocol.
- Many protocols supported.

Additional Features

`ejabberd` also 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.
- SQLite support.
- Authentication
- Internal Authentication.
- PAM, LDAP and SQL.
- External Authentication script.
- Others
- Support for virtual hosting.
- Compressing XML streams with Stream Compression (`XEP-0138`).
- Statistics via Statistics Gathering (`XEP-0039`).
- IPv6 support both for c2s and s2s connections.
- `Multi-User Chat` module with support for clustering and HTML logging.
- Users Directory based on users vCards.
- `Publish-Subscribe` component with support for `Personal Eventing via Pubsub` .
- Support for web clients: Support for XMPP subprotocol for WebSocket and `HTTP Binding (BOSH)` services.
- IRC transport.
- SIP support.
- Component support: interface with networks such as AIM, ICQ and MSN installing special transports.

Frequently Asked Questions

Development process

Why is there a commercial version of ejabberd?

Different needs for different users. Corporations and large scale deployments are very different from smaller deployments and community projects.

While we put a huge development effort to have a product that is on the edge of innovation with ejabberd community version, we are requested to provide a stable version with long term commitment and high level of quality, testing, audit, etc.

Maintaining such a version in parallel to the community version, along with extremely strong commitment in terms of availability and 24/7 support has a tangible cost. With ejabberd business edition we commit to a level of scalability and optimize the software until it is performing to the level agreed with the customer.

Covering all those costs, along with all our Research and Development cost on ejabberd community in general is the real reason we need a business edition.

The business edition is also a way for our customers to share the code between our customers only, thus retaining a huge competitive edge for a limited time (See next section).

So, even if you are not using our business edition, this is a great benefit for you as a user of the community edition and the reason you have seen so many improvements since 2002. Thanks to our business edition customers, ejabberd project itself is a major contributor to Erlang and Elixir community.

Does ProcessOne voluntarily hold some code in ejabberd community to push toward the business edition?

No. We never do that and have no plan doing so with the code we produce and we own.

However, when the code is paid by customer, they retain the ownership of the code. Part of our agreement is that the code produced for them will be limited to a restricted user base, ejabberd business edition until an agreed time expires, generally between 6 months and 1 year.

This is extremely important for both the users of the commercial edition and the users of the community edition:

- For the business edition customers, this is a way to keep their business advantage. This is fair as they paid for the development.
- This is also a great incentive for our customers as they benefit from development for other customers (I guess they agree for the reciprocal sharing of their own code with customers).
- This is fair for the community as the community edition users know they will benefit from new extremely advanced features in a relatively near future. For example, websocket module was contributed to ejabberd community as part of this process.

This is the model we have found to be fair to our broader user base and lets us produce an amazing code base that benefits all our users.

This dual model is the core strength of our approach and our secret sauce to make sure everyone benefits.

Performance

Is ejabberd the most scalable version?

Yes. Definitely. Despite claims that there is small change you can make to make it more scalable, we already performed the changes during the past year, that make those claims unfunded:

- ejabberd reduced memory consumption in 2013 by switching to binary representation of string instead of list. This can reduce given string by 8.
- We have improved the C code performance a lot, using new Erlang NIF. This provides better performance, removes bottlenecks
- We have a different clustering mechanism available to make sure we can scale to large clusters

This is a common misconception, but our feedback for production service on various customer sites shows that ejabberd is the most scalable version. Once it is properly configured, optimized and tuned, you can handle tens of millions of users on ejabberd systems.

... And we are still improving :)

As a reference, you should read the following blog post: ejabberd Massive Scalability: 1 Node — 2+ Million Concurrent Users

What are the tips to optimize performance?

Optimisation of XMPP servers performance, including ejabberd, is highly dependent on the use case. You really need to find your bottleneck(s) by monitoring the process queues, finding out what is your limiting factor, tune that and then move to the next one.

The first step is to make sure you run the latest ejabberd. Each new release comes with a bunch of optimisations and a specific bottleneck you are facing may have gone away in the latest version.

For perspective, ejabberd 15.07 is 2 to 3 times more efficient in memory, latency and CPU compared to ejabberd 2.1.

You should also make sure that you are using the latest Erlang version. Each release of Erlang comes with more optimisation regarding locks, especially on SMP servers, and using the latest Erlang version can also help tremendously.

Erlang support

Is ejabberd conforming to the best Erlang practices?

Yes. Our build system is primarily based on rebar. However, as we are multiplatform and need to run in many various environments, we rely on a toolchain that can detect required library dependencies using autotools.

This gives developers and admins the best of both worlds. A very flexible and very versatile build chain, that is very adequate to make sure ejabberd can be used in most operating systems and even integrated in Linux distributions.

ejabberd Use Cases

ejabberd is very versatile and is a solid choice to build messaging services across a large number of industries:

ejabberd

Mobile messaging

ejabberd's massive scalability makes it the most solid choice as the backbone for a very large number of mobile messaging services.

This includes:

- Chaatz
- Libon
- Nokia OVI Chat
- Roo Kids : Safe & fun instant messaging app for kids with minimum yet critical parental controls.
- Swisscom IO
- Versapp
- Whatsapp

Gaming

- Electronic Arts
- FACEIT
- Kixeye
- Machine Zone (Game of War)
- Nokia nGage
- Riot Games (League of Legends)

Voice and video messaging

- Nimbuzz
- ooVoo
- Sipphone
- WowApp

Internet of Things

- AeroFS
- IMA Teleassistance
- Nabaztag (Violet, Mindscape, then Aldebaran Robotics)

Telecom / Hosting

- Fastmail
- GMX
- Mailfence
- Orange

- SAPO - Portugal Telecom

Customer chat / CRM

- CoBrowser.net: Coder Interview.
- iAdvize
- LiveHelpercChat: Blog post: Full XMPP chat support for ejabberd

Media

- AFP
- BBC

Social media

- Facebook
- Nasza Klasa (NKTalk messenger)
- StudiVZ
- Sify
- Tuenti

Sport

- Major League of Baseball (MLB)

Education

- Apollo group
- Laureate

Push alerts

- Nokia push notifications
- Notify.me

Dating

- Grindr
- Meetic

Community sites

- Jabber.at
- Talkr.im

XMPP Use Cases

XMPP is a very versatile protocol designed to address many use cases of modern real-time messaging needs. However, it is also a very large protocol and it is difficult to understand at first sight all the use cases that XMPP adequately addresses.

This page is gathering XMPP specifications that make XMPP a good fit for a given use case of industry.

Realtime web

XMPP was designed before the advent of realtime web. However, it managed to adapt thanks to the following specifications:

- XMPP PubSub is defined in XEP-0060. This is a very powerful mechanism that defines channel based communication on top of the XMPP protocol itself. A server can handle millions of channels, called Pubsub nodes. Users interested in specific channels can subscribe to nodes. When data needs to be send to a given channel, authorized publishers can send data to that node. The XMPP server will then broadcast the content to all subscribers. This is very adequate for realtime web as it allows you to broadcast relevant events to web pages.
- WebSocket: XMPP over WebSocket is defined in RFC 7395. It is more efficient and more scalable than XMPP for web's previous specifications called BOSH. WebSocket being a true bidirectional channel, it allows lower latency messaging and is very reliable. Note that BOSH can still be used transparently along with WebSocket to support old web browsers.

Use cases: News, interactive web page, web chat, web games.

Supported by ejabberd: Yes.

As a special exception, the authors give permission to link this program with the OpenSSL library and distribute the resulting binary.

GNU GENERAL PUBLIC LICENSE

Version 2, June 1991

```
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA

Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
```

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this

License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
one line to give the program's name and an idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; either version 2
of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details
type `show w'.  This is free software, and you are welcome
to redistribute it under certain conditions; type `show c'
for details.
```

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright
interest in the program `Gnomovision'
(which makes passes at compilers) written
by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License.

Readme
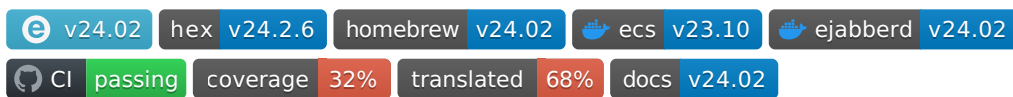
ejabberd is an open-source, robust, scalable and extensible realtime platform built using Erlang/OTP, that includes XMPP Server, MQTT Broker and SIP Service.

Check the features in ejabberd.im, ejabberd Docs, ejabberd at ProcessOne, and the list of supported protocols in ProcessOne and XMPP.org.

Installation

There are several ways to install ejabberd:

- Source code: compile yourself, see COMPILE
- Installers: ProcessOne Download and GitHub Releases for releases, GitHub Actions for master branch (run/deb/rpm for x64 and arm64)
- `ecs` container image: Docker Hub and Github Packages, see ecs README (for x64)
- `ejabberd` container image: Github Packages for releases and master branch, see CONTAINER (for x64 and arm64)
- Using your Operating System package
- Using the Homebrew package manager

Documentation

Please check the ejabberd Docs website.

When compiling from source code, you can get some help with:

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

Once ejabberd is installed, try:

```
ejabberdctl help
man ejabberd.yml
```

Development

Bug reports and features are tracked using GitHub Issues, please check CONTRIBUTING for details.

Translations can be improved online using Weblate or in your local machine as explained in Localization.

Documentation for developers is available in ejabberd docs: Developers.

There are nightly builds of ejabberd, both for `master` branch and for Pull Requests: - Installers: go to GitHub Actions: Installers, open the most recent commit, on the bottom of that commit page, download the `ejabberd-packages.zip` artifact. - `ejabberd` container image: go to ejabberd Github Packages

Security reports or concerns should preferably be reported privately, please send an email to the address: contact at process-one dot net or some other method from ProcessOne Contact.

For commercial offering and support, including ejabberd Business Edition and Fluux (ejabberd in the Cloud), please check ProcessOne ejabberd page.

Community

There are several places to get in touch with other ejabberd developers and administrators:

- ejabberd XMPP chatroom: ejabberd@conference.process-one.net
- GitHub Discussions
- Stack Overflow

License

ejabberd is released under the GNU General Public License v2 (see COPYING), and ejabberd translations under MIT License.

Install

Installation

There are several ways to install ejabberd Community Server, depending on your needs and your infrastructure.

Self-hosted

Container Images

- `ejabberd` and `ecs` Container Images – Ideal for Windows, macOS, Linux, ...

Binary Installers

- Linux RUN Installer – Suitable for various Linux distributions
- Linux DEB and RPM Installers – Specifically for DEB and RPM based Linux

Linux and *BSD

- Operating System Package – Tailored for System Operators

MacOS

- Homebrew – Optimized for MacOS

Source Code

- Source Code – Geared towards developers and advanced administrators

On-Premise (eBE)

- ejabberd Business Edition – Explore professional support and managed services on your infrastructure

Cloud Hosting (Fluux)

- Fluux.io – Opt for ejabberd hosting with a user-friendly web dashboard

Install ejabberd using a Container Image

There are two official container images of ejabberd that you can install using docker (or podman):

`ejabberd` Container Image

The `"ejabberd"` container image is available in the GitHub Container Registry. It is available for x64 and arm64, both for stable ejabberd releases and the `master` branch. Check the `"ejabberd"` container documentation.

For example, download the latest stable ejabberd release:

```
docker pull ghcr.io/processone/ejabberd
```

If you use Microsoft Windows 7, 10, or similar operating systems, check those tutorials:

- Install ejabberd on Windows 10 using Docker Desktop
- Install ejabberd on Windows 7 using Docker Toolbox

For bug reports and improvement suggestions, if you use the `"ecs"` container image please go to the docker-ejabberd GitHub Issues, if using the `"ejabberd"` container image from github please go to the ejabberd GitHub Issues

`ecs` Container Image

The `"ecs"` container image allows to get ejabberd stable releases in x64 machines. Check the `"ecs"` container documentation.

Download ejabberd with:

```
docker pull docker.io/ejabberd/ecs
```

`e` v24.02   🐳 ejabberd v24.02

`ejabberd` Container Image

`ejabberd` is an open-source, robust, scalable and extensible realtime platform built using Erlang/OTP, that includes XMPP Server, MQTT Broker and SIP Service.

This document explains how to use the `ejabberd` container image available in ghcr.io/processone/ejabberd, built using the files in `.github/container/`. This image is based in Alpine 3.19, includes Erlang/OTP 26.2 and Elixir 1.16.1.

Alternatively, there is also the `ecs` container image available in docker.io/ejabberd/ecs, built using the docker-ejabberd/ecs repository. Check the differences between `ejabberd` and `ecs` images.

If you are using a Windows operating system, check the tutorials mentioned in ejabberd Docs > Docker Image.

Start ejabberd

With default configuration

Start ejabberd in a new container:

```
docker run --name ejabberd -d -p 5222:5222 ghcr.io/processone/ejabberd
```

That runs the container as a daemon, using ejabberd default configuration file and XMPP domain "localhost".

Stop the running container:

```
docker stop ejabberd
```

Restart the stopped ejabberd container:

```
docker restart ejabberd
```

Start with Erlang console attached

Start ejabberd with an Erlang console attached using the `live` command:

```
docker run --name ejabberd -it -p 5222:5222 ghcr.io/processone/ejabberd live
```

That uses the default configuration file and XMPP domain "localhost".

Start with your configuration and database

Pass a configuration file as a volume and share the local directory to store database:

```
mkdir database
chown ejabberd database

cp ejabberd.yml.example ejabberd.yml

docker run --name ejabberd -it \
  -v $(pwd)/ejabberd.yml:/opt/ejabberd/conf/ejabberd.yml \
  -v $(pwd)/database:/opt/ejabberd/database \
  -p 5222:5222 ghcr.io/processone/ejabberd live
```

Notice that ejabberd runs in the container with an account named `ejabberd`, and the volumes you mount must grant proper rights to that account.

Next steps

Register the administrator account

The default ejabberd configuration does not grant admin privileges to any account, you may want to register a new account in ejabberd and grant it admin rights.

Register an account using the `ejabberdctl` script:

```
docker exec -it ejabberd ejabberdctl register admin localhost passw0rd
```

Then edit conf/ejabberd.yml and add the ACL as explained in ejabberd Docs: Administration Account

Check ejabberd log files

Check the content of the log files inside the container, even if you do not put it on a shared persistent drive:

```
docker exec -it ejabberd tail -f logs/ejabberd.log
```

Inspect the container files

The container uses Alpine Linux. Start a shell inside the container:

```
docker exec -it ejabberd sh
```

Open ejabberd debug console

Open an interactive debug Erlang console attached to a running ejabberd in a running container:

```
docker exec -it ejabberd ejabberdctl debug
```

CAPTCHA

ejabberd includes two example CAPTCHA scripts. If you want to use any of them, first install some additional required libraries:

```
docker exec --user root ejabberd apk add imagemagick ghostscript-fonts bash
```

Now update your ejabberd configuration file, for example:

```
docker exec -it ejabberd vi conf/ejabberd.yml
```

and add this option:

```
captcha_cmd: /opt/ejabberd-22.04/lib/captcha.sh
```

Finally, reload the configuration file or restart the container:

```
docker exec ejabberd ejabberdctl reload_config
```

If the CAPTCHA image is not visible, there may be a problem generating it (the ejabberd log file may show some error message); or the image URL may not be correctly detected by ejabberd, in that case you can set the correct URL manually, for example:

```
captcha_url: https://localhost:5443/captcha
```

For more details about CAPTCHA options, please check the CAPTCHA documentation section.

Advanced Container Configuration

Ports

This container image exposes the ports:

- `5222` : The default port for XMPP clients.
- `5269` : For XMPP federation. Only needed if you want to communicate with users on other servers.
- `5280` : For admin interface.
- `5443` : With encryption, used for admin interface, API, CAPTCHA, OAuth, Websockets and XMPP BOSH.
- `1883` : Used for MQTT
- `4369-4399` : EPMD and Erlang connectivity, used for `ejabberdctl` and clustering
- `5210` : Erlang connectivity when `ERL_DIST_PORT` is set, alternative to EPMD

Volumes

ejabberd produces two types of data: log files and database spool files (Mnesia). This is the kind of data you probably want to store on a persistent or local drive (at least the database).

The volumes you may want to map:

- `/opt/ejabberd/conf/` : Directory containing configuration and certificates
- `/opt/ejabberd/database/` : Directory containing Mnesia database. You should back up or export the content of the directory to persistent storage (host storage, local storage, any storage plugin)
- `/opt/ejabberd/logs/` : Directory containing log files
- `/opt/ejabberd/upload/` : Directory containing uploaded files. This should also be backed up.

All these files are owned by `ejabberd` user inside the container.

It's possible to install additional ejabberd modules using volumes, this comment explains how to install an additional module using docker-compose.

Commands on start

The ejabberdctl script reads the `CTL_ON_CREATE` environment variable the first time the container is started, and reads `CTL_ON_START` every time the container is started. Those variables can contain one ejabberdctl command, or several commands separated with the blankspace and `;` characters.

By default failure of any of commands executed that way would abort start, this can be disabled by prefixing commands with `!`

Example usage (or check the full example):

```
environment:
  - CTL_ON_CREATE=! register admin localhost asd
  - CTL_ON_START=stats registeredusers ;
             check_password admin localhost asd ;
             status
```

Clustering

When setting several containers to form a cluster of ejabberd nodes, each one must have a different Erlang Node Name and the same Erlang Cookie.

For this you can either: - edit `conf/ejabberdctl.cfg` and set variables `ERLANG_NODE` and `ERLANG_COOKIE` - set the environment variables `ERLANG_NODE_ARG` and `ERLANG_COOKIE`

Example to connect a local `ejabberdctl` to a containerized ejabberd: 1. When creating the container, export port 5210, and set `ERLANG_COOKIE` :

```
docker run --name ejabberd -it \
    -e ERLANG_COOKIE=`cat $HOME/.erlang.cookie` \
    -p 5210:5210 -p 5222:5222 \
    ghcr.io/processone/ejabberd
```

2. Set `ERL_DIST_PORT=5210` in ejabberdctl.cfg of container and local ejabberd 3. Restart the container 4. Now use `ejabberdctl` in your local ejabberd deployment

To connect using a local `ejabberd` script:

```
ERL_DIST_PORT=5210 _build/dev/rel/ejabberd/bin/ejabberd ping
```

Example using environment variables (see full example [docker-compose.yml](#)):

```
environment:
    - ERLANG_NODE_ARG=ejabberd@node7
    - ERLANG_COOKIE=dummycookie123
```

Build a Container Image

This container image includes ejabberd as a standalone OTP release built using Elixir. That OTP release is configured with:

- `mix.exs` : Customize ejabberd release
- `vars.config` : ejabberd compilation configuration options
- `config/runtime.exs` : Customize ejabberd paths
- `ejabberd.yml.template` : ejabberd default config file

Direct build

Build ejabberd Community Server container image from ejabberd master git repository:

```
docker buildx build \
    -t personal/ejabberd \
    -f .github/container/Dockerfile \
    .
```

Podman build

It's also possible to use podman instead of docker, just notice: - `EXPOSE 4369-4399` port range is not supported, remove that in Dockerfile - It mentions that `healthcheck` is not supported by the Open Container Initiative image format - to start with command `live` , you may want to add environment variable `EJABBERD_BYPASS_WARNINGS=true`

```
podman build \
    -t ejabberd \
    -f .github/container/Dockerfile \
    .

podman run --name eja1 -d -p 5222:5222 localhost/ejabberd

podman exec eja1 ejabberdctl status

podman exec -it eja1 sh

podman stop eja1

podman run --name eja1 -it -e EJABBERD_BYPASS_WARNINGS=true -p 5222:5222 localhost/ejabberd live
```

Package build for `arm64`

By default, `.github/container/Dockerfile` builds this container by directly compiling ejabberd, it is a fast and direct method. However, a problem with QEMU prevents building the container in QEMU using Erlang/OTP 25 for the `arm64` architecture.

Providing `--build-arg METHOD=package` is an alternate method to build the container used by the Github Actions workflow that provides `amd64` and `arm64` container images. It first builds an ejabberd binary package, and later installs it in the image. That method avoids using QEMU, so it can build `arm64` container images, but is extremely slow the first time it's used, and consequently not recommended for general use.

In this case, to build the ejabberd container image for arm64 architecture:

```
docker buildx build \
    --build-arg METHOD=package \
    --platform linux/arm64 \
    -t personal/ejabberd:$VERSION \
    -f .github/container/Dockerfile \
    .
```

Composer Examples

Minimal Example

This is the barely minimal file to get a usable ejabberd. Store it as `docker-compose.yml`:

```
services:
  main:
    image: ghcr.io/processone/ejabberd
    container_name: ejabberd
    ports:
      - "5222:5222"
      - "5269:5269"
      - "5280:5280"
      - "5443:5443"
```

Create and start the container with the command:

```
docker-compose up
```

Customized Example

This example shows the usage of several customizations: it uses a local configuration file, stores the mnesia database in a local path, registers an account when it's created, and checks the number of registered accounts every time it's started.

Download or copy the ejabberd configuration file:

```
wget https://raw.githubusercontent.com/processone/ejabberd/master/ejabberd.yml.example
mv ejabberd.yml.example ejabberd.yml
```

Create the database directory and allow the container access to it:

```
mkdir database
sudo chown 9000:9000 database
```

Now write this `docker-compose.yml` file:

```
version: '3.7'

services:

  main:
    image: ghcr.io/processone/ejabberd
    container_name: ejabberd
    environment:
      - CTL_ON_CREATE=register admin localhost asd
      - CTL_ON_START=registered_users localhost ;
                     status
    ports:
      - "5222:5222"
      - "5269:5269"
      - "5280:5280"
      - "5443:5443"
    volumes:
      - ./ejabberd.yml:/opt/ejabberd/conf/ejabberd.yml:ro
      - ./database:/opt/ejabberd/database
```

Clustering Example

In this example, the main container is created first. Once it is fully started and healthy, a second container is created, and once ejabberd is started in it, it joins the first one.

An account is registered in the first node when created (and we ignore errors that can happen when doing that - for example whenn account already exists), and it should exist in the second node after join.

Notice that in this example the main container does not have access to the exterior; the replica exports the ports and can be accessed.

```
version: '3.7'

services:

  main:
    image: ghcr.io/processone/ejabberd
    container_name: ejabberd
    environment:
      - ERLANG_NODE_ARG=ejabberd@main
      - ERLANG_COOKIE=dummycookie123
      - CTL_ON_CREATE=! register admin localhost asd

  replica:
    image: ghcr.io/processone/ejabberd
    container_name: replica
    depends_on:
      main:
        condition: service_healthy
    ports:
      - "5222:5222"
      - "5269:5269"
      - "5280:5280"
      - "5443:5443"
    environment:
      - ERLANG_NODE_ARG=ejabberd@replica
      - ERLANG_COOKIE=dummycookie123
      - CTL_ON_CREATE=join_cluster ejabberd@main
      - CTL_ON_START=registered_users localhost ;
                     status
```

version v23.10 | image size 39.5 MB | docker stars 67 | docker pulls 5.9M | Tests passing | Stars 94

`ecs` Container Image

ejabberd is an open-source XMPP server, robust, scalable and modular, built using Erlang/OTP, and also includes MQTT Broker and SIP Service.

This container image allows you to run a single node ejabberd instance in a container.

There is an Alternative Image in GitHub Packages, built using a different method and some improvements.

If you are using a Windows operating system, check the tutorials mentioned in ejabberd Docs > Docker Image.

Start ejabberd

With default configuration

You can start ejabberd in a new container with the following command:

```
docker run --name ejabberd -d -p 5222:5222 ejabberd/ecs
```

This command will run the container image as a daemon, using ejabberd default configuration file and XMPP domain "localhost".

To stop the running container, you can run:

```
docker stop ejabberd
```

If needed, you can restart the stopped ejabberd container with:

```
docker restart ejabberd
```

Start with Erlang console attached

If you would like to start ejabberd with an Erlang console attached you can use the `live` command:

```
docker run -it -p 5222:5222 ejabberd/ecs live
```

This command will use default configuration file and XMPP domain "localhost".

Start with your configuration and database

This command passes the configuration file using the volume feature and shares the local directory to store database:

```
mkdir database
docker run -d --name ejabberd -v $(pwd)/ejabberd.yml:/home/ejabberd/conf/ejabberd.yml -v $(pwd)/database:/home/ejabberd/database -p 5222:5222 ejabberd/ecs
```

Next steps

Register the administrator account

The default ejabberd configuration has already granted admin privilege to an account that would be called `admin@localhost`, so you just need to register such an account to start using it for administrative purposes. You can register this account using the `ejabberdctl` script, for example:

```
docker exec -it ejabberd bin/ejabberdctl register admin localhost passw0rd
```

Check ejabberd log files

Check the ejabberd log file in the container:

```
docker exec -it ejabberd tail -f logs/ejabberd.log
```

Inspect the container files

The container uses Alpine Linux. You can start a shell there with:

```
docker exec -it ejabberd sh
```

Open ejabberd debug console

You can open a live debug Erlang console attached to a running container:

```
docker exec -it ejabberd bin/ejabberdctl debug
```

CAPTCHA

ejabberd includes two example CAPTCHA scripts. If you want to use any of them, first install some additional required libraries:

```
docker exec --user root ejabberd apk add imagemagick ghostscript-fonts bash
```

Now update your ejabberd configuration file, for example:

```
docker exec -it ejabberd vi conf/ejabberd.yml
```

and add this option:

```
captcha_cmd: /home/ejabberd/lib/ejabberd-21.1.0/priv/bin/captcha.sh
```

Finally, reload the configuration file or restart the container:

```
docker exec ejabberd bin/ejabberdctl reload_config
```

If the CAPTCHA image is not visible, there may be a problem generating it (the ejabberd log file may show some error message); or the image URL may not be correctly detected by ejabberd, in that case you can set the correct URL manually, for example:

```
captcha_url: https://localhost:5443/captcha
```

For more details about CAPTCHA options, please check the CAPTCHA documentation section.

Use ejabberdapi

When the container is running (and thus ejabberd), you can exec commands inside the container using `ejabberdctl` or any other of the available interfaces, see Understanding ejabberd "commands"

Additionally, this container image includes the `ejabberdapi` executable. Please check the ejabberd-api homepage for configuration and usage details.

For example, if you configure ejabberd like this:

```
listen:
  -
    port: 5282
    module: ejabberd_http
    request_handlers:
      "/api": mod_http_api

acl:
  loopback:
    ip:
```

```
        - 127.0.0.0/8
        - ::1/128
        - ::FFFF:127.0.0.1/128

api_permissions:
  "admin access":
    who:
      access:
        allow:
          acl: loopback
    what:
      - "register"
```

Then you could register new accounts with this query:

```
docker exec -it ejabberd bin/ejabberdapi register --endpoint=http://127.0.0.1:5282/ --jid=admin@localhost --password=passw0rd
```

Advanced container configuration

Ports

This container image exposes the ports:

- `5222` : The default port for XMPP clients.

- `5269` : For XMPP federation. Only needed if you want to communicate with users on other servers.

- `5280` : For admin interface.

- `5443` : With encryption, used for admin interface, API, CAPTCHA, OAuth, Websockets and XMPP BOSH.

- `1883` : Used for MQTT

- `4369-4399` : EPMD and Erlang connectivity, used for `ejabberdctl` and clustering

Volumes

ejabberd produces two types of data: log files and database (Mnesia). This is the kind of data you probably want to store on a persistent or local drive (at least the database).

Here are the volume you may want to map:

- `/home/ejabberd/conf/` : Directory containing configuration and certificates

- `/home/ejabberd/database/` : Directory containing Mnesia database. You should back up or export the content of the directory to persistent storage (host storage, local storage, any storage plugin)

- `/home/ejabberd/logs/` : Directory containing log files

- `/home/ejabberd/upload/` : Directory containing uploaded files. This should also be backed up.

All these files are owned by ejabberd user inside the container. Corresponding `UID:GID` is `9000:9000` . If you prefer bind mounts instead of volumes, then you need to map this to valid `UID:GID` on your host to get read/write access on mounted directories.

Commands on start

The ejabberdctl script reads the `CTL_ON_CREATE` environment variable the first time the container is started, and reads `CTL_ON_START` every time the container is started. Those variables can contain one ejabberdctl command, or several commands separated with the blankspace and `;` characters.

By default failure of any of commands executed that way would abort start, this can be disabled by prefixing commands with `!`

Example usage (or check the full example):

```
    environment:
      - CTL_ON_CREATE=\! register admin localhost asd
      - CTL_ON_START=stats registeredusers ;
                     check_password admin localhost asd ;
                     status
```

Clustering

When setting several containers to form a cluster of ejabberd nodes, each one must have a different Erlang Node Name and the same Erlang Cookie. For this you can either: - edit `conf/ejabberdctl.cfg` and set variables `ERLANG_NODE` and `ERLANG_COOKIE` - set the environment variables `ERLANG_NODE_ARG` and `ERLANG_COOKIE`

Once you have the ejabberd nodes properly set and running, you can tell the secondary nodes to join the master node using the `join_cluster` API call.

Example using environment variables (see the full `docker-compose.yml` clustering example):

```
environment:
  - ERLANG_NODE_ARG=ejabberd@replica
  - ERLANG_COOKIE=dummycookie123
  - CTL_ON_CREATE=join_cluster ejabberd@main
```

Change Mnesia Node Name

To use the same Mnesia database in a container with a different hostname, it is necessary to change the old hostname stored in Mnesia.

This section is equivalent to the ejabberd Documentation Change Computer Hostname, but particularized to containers that use this ecs container image from ejabberd 23.01 or older.

Setup Old Container

Let's assume a container running ejabberd 23.01 (or older) from this ecs container image, with the database directory binded and one registered account. This can be produced with:

```
OLDCONTAINER=ejaold
NEWCONTAINER=ejanew

mkdir database
sudo chown 9000:9000 database
docker run -d --name $OLDCONTAINER -p 5222:5222 \
        -v $(pwd)/database:/home/ejabberd/database \
        ejabberd/ecs:23.01
docker exec -it $OLDCONTAINER bin/ejabberdctl started
docker exec -it $OLDCONTAINER bin/ejabberdctl register user1 localhost somepass
docker exec -it $OLDCONTAINER bin/ejabberdctl registered_users localhost
```

Methods to know the Erlang node name:

```
ls database/ | grep ejabberd@
docker exec -it $OLDCONTAINER bin/ejabberdctl status
docker exec -it $OLDCONTAINER grep "started in the node" logs/ejabberd.log
```

Change Mnesia Node

First of all let's store the Erlang node names and paths in variables. In this example they would be:

```
OLDCONTAINER=ejaold
NEWCONTAINER=ejanew
OLDNODE=ejabberd@95145ddee27c
NEWNODE=ejabberd@localhost
OLDFILE=/home/ejabberd/database/old.backup
NEWFILE=/home/ejabberd/database/new.backup
```

1. Start your old container that can still read the Mnesia database correctly. If you have the Mnesia spool files, but don't have access to the old container anymore, go to Create Temporary Container and later come back here.

2. Generate a backup file and check it was created:

```
docker exec -it $OLDCONTAINER bin/ejabberdctl backup $OLDFILE
ls -l database/*.backup
```

3. Stop ejabberd:

```
docker stop $OLDCONTAINER
```

4. Create the new container. For example:

```
docker run \
      --name $NEWCONTAINER \
      -d \
      -p 5222:5222 \
      -v $(pwd)/database:/home/ejabberd/database \
      ejabberd/ecs:latest
```

5. Convert the backup file to new node name:

```
docker exec -it $NEWCONTAINER bin/ejabberdctl mnesia_change_nodename $OLDNODE $NEWNODE $OLDFILE $NEWFILE
```

6. Install the backup file as a fallback:

```
docker exec -it $NEWCONTAINER bin/ejabberdctl install_fallback $NEWFILE
```

7. Restart the container:

```
docker restart $NEWCONTAINER
```

8. Check that the information of the old database is available. In this example, it should show that the account `user1` is registered:

```
docker exec -it $NEWCONTAINER bin/ejabberdctl registered_users localhost
```

9. When the new container is working perfectly with the converted Mnesia database, you may want to remove the unneeded files: the old container, the old Mnesia spool files, and the backup files.

Create Temporary Container

In case the old container that used the Mnesia database is not available anymore, a temporary container can be created just to read the Mnesia database and make a backup of it, as explained in the previous section.

This method uses `--hostname` command line argument for docker, and `ERLANG_NODE_ARG` environment variable for ejabberd. Their values must be the hostname of your old container and the Erlang node name of your old ejabberd node. To know the Erlang node name please check Setup Old Container.

Command line example:

```
OLDHOST=${OLDNODE#*@}
docker run \
      -d \
      --name $OLDCONTAINER \
      --hostname $OLDHOST \
      -p 5222:5222 \
      -v $(pwd)/database:/home/ejabberd/database \
      -e ERLANG_NODE_ARG=$OLDNODE \
      ejabberd/ecs:latest
```

Check the old database content is available:

```
docker exec -it $OLDCONTAINER bin/ejabberdctl registered_users localhost
```

Now that you have ejabberd running with access to the Mnesia database, you can continue with step 2 of previous section Change Mnesia Node.

Generating ejabberd release

Configuration

Image is built by embedding an ejabberd Erlang/OTP standalone release in the image.

The configuration of ejabberd Erlang/OTP release is customized with:

- `rel/config.exs` : Customize ejabberd release
- `rel/dev.exs` : ejabberd environment configuration for development release
- `rel/prod.exs` : ejabberd environment configuration for production release
- `vars.config` : ejabberd compilation configuration options
- `conf/ejabberd.yml` : ejabberd default config file

Build ejabberd Community Server base image from ejabberd master on Github:

```
docker build -t ejabberd/ecs .
```

Build ejabberd Community Server base image for a given ejabberd version:

```
./build.sh 18.03
```

Composer Examples

Minimal Example

This is the barely minimal file to get a usable ejabberd. Store it as `docker-compose.yml` :

```
services:
  main:
    image: ejabberd/ecs
    container_name: ejabberd
    ports:
      - "5222:5222"
      - "5269:5269"
      - "5280:5280"
      - "5443:5443"
```

Create and start the container with the command:

```
docker-compose up
```

Customized Example

This example shows the usage of several customizations: it uses a local configuration file, stores the mnesia database in a local path, registers an account when it's created, and checks the number of registered accounts every time it's started.

Download or copy the ejabberd configuration file:

```
wget https://raw.githubusercontent.com/processone/ejabberd/master/ejabberd.yml.example
mv ejabberd.yml.example ejabberd.yml
```

Create the database directory and allow the container access to it:

```
mkdir database
sudo chown 9000:9000 database
```

Now write this `docker-compose.yml` file:

```
version: '3.7'

services:
```

```
main:
  image: ejabberd/ecs
  container_name: ejabberd
  environment:
    - CTL_ON_CREATE=register admin localhost asd
    - CTL_ON_START=registered_users localhost ;
                   status
  ports:
    - "5222:5222"
    - "5269:5269"
    - "5280:5280"
    - "5443:5443"
  volumes:
    - ./ejabberd.yml:/home/ejabberd/conf/ejabberd.yml:ro
    - ./database:/home/ejabberd/database
```

Clustering Example

In this example, the main container is created first. Once it is fully started and healthy, a second container is created, and once ejabberd is started in it, it joins the first one.

An account is registered in the first node when created (and we ignore errors that can happen when doing that - for example when account already exists), and it should exist in the second node after join.

Notice that in this example the main container does not have access to the exterior; the replica exports the ports and can be accessed.

```
version: '3.7'

services:

  main:
    image: ejabberd/ecs
    container_name: main
    environment:
      - ERLANG_NODE_ARG=ejabberd@main
      - ERLANG_COOKIE=dummycookie123
      - CTL_ON_CREATE=\! register admin localhost asd
    healthcheck:
      test: netstat -nl | grep -q 5222
      start_period: 5s
      interval: 5s
      timeout: 5s
      retries: 120

  replica:
    image: ejabberd/ecs
    container_name: replica
    depends_on:
      main:
        condition: service_healthy
    ports:
      - "5222:5222"
      - "5269:5269"
      - "5280:5280"
      - "5443:5443"
    environment:
      - ERLANG_NODE_ARG=ejabberd@replica
      - ERLANG_COOKIE=dummycookie123
      - CTL_ON_CREATE=join_cluster ejabberd@main
      - CTL_ON_START=registered_users localhost ;
                     status
```

Binary Installers

Linux RUN Installer

The `*.run` binary installer will deploy and configure a full featured ejabberd server and does not require any extra dependencies. It includes a stripped down version of Erlang. As such, when using ejabberd installer, you do not need to install Erlang separately.

Those instructions assume installation on `localhost` for development purposes. In this document, when mentioning `ejabberd-YY.MM`, we assume `YY.MM` is the release number, for example 18.01.

Installation using the `*.run` binary installer:

1. Go to ejabberd GitHub Releases.

2. Download the `run` package for your architecture

3. Right-click on the downloaded file and select "Properties", click on the "Permissions" tab and tick the box that says "Allow executing file as program". Alternatively, you can set the installer as executable using the command line:

   ```
   chmod +x ejabberd-YY.MM-1-linux-x64.run
   ```

4. If the installer runs as superuser (by `root` or using `sudo`), it installs ejabberd binaries in `/opt/ejabberd-XX.YY/`; installs your configuration, Mnesia database and logs in `/opt/ejabberd/`, and setups an ejabberd service unit in `systemd`:

   ```
   sudo ./ejabberd-YY.MM-1-linux-x64.run
   ```

5. If the installer runs as a regular user, it asks the base path where ejabberd should be installed. In that case, the ejabberd service unit is not set in `systemd`, and `systemctl` cannot be used to start ejabberd; start it manually.

6. After successful installation by root, ejabberd is automatically started. Check its status with

   ```
   systemctl status ejabberd
   ```

7. Now that ejabberd is installed and running with the default configuration, it's time to do some basic setup: edit `/opt/ejabberd/conf/ejabberd.yml` and setup in the `hosts` option the domain that you want ejabberd to serve. By default it's set to the name of your computer on the local network.

8. Restart ejabberd completely using systemctl, or using ejabberdctl, or simply tell it to reload the configuration file:

   ```
   sudo systemctl restart ejabberd
   sudo /opt/ejabberd-22.05/bin/ejabberdctl restart
   sudo /opt/ejabberd-22.05/bin/ejabberdctl reload_config
   ```

9. Quite probably you will want to register an account and grant it admin rights, please check Next Steps: Administration Account.

10. Now you can go to the web dashboard at `http://localhost:5280/admin/` and fill the username field with the full account JID, for example `admin@domain` (or `admin@localhost` as above). Then fill the password field with that account's `password`. The next step is to get to know how to configure ejabberd.

11. If something goes wrong during the installation and you would like to start from scratch, you will find the steps to uninstall in the file `/opt/ejabberd-22.05/uninstall.txt`.

Linux DEB and RPM Installers

ProcessOne provides DEB and RPM all-in-one binary installers with the same content that the `*.run` binary installer mentioned in the previous section.

Those are self-sufficient packages that contain a minimal Erlang distribution, this ensures that it does not interfere with your existing Erlang version and is also a good way to make sure ejabberd will run with the latest Erlang version.

Those packages install ejabberd in `/opt/ejabberd-XX.YY/`. Your configuration, Mnesia database and logs are available in `/opt/ejabberd/`.

You can download directly the DEB and RPM packages from ejabberd GitHub Releases.

If you prefer, you can also get those packages from our official ejabberd packages repository.

Operating System Packages

Many operating systems provide specific ejabberd packages adapted to the system architecture and libraries. They usually also check dependencies and perform basic configuration tasks like creating the initial administrator account.

List of known ejabberd packages:

- Alpine Linux
- Arch Linux
- Debian
- Fedora
- FreeBSD
- Gentoo
- OpenSUSE
- NetBSD
- Ubuntu

Consult the resources provided by your Operating System for more information.

There's also an ejabberd snap to install ejabberd on several operating systems using `Snap` package manager.

Install 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.

Requirements

To compile ejabberd you need:

- GNU Make
- GCC
- Libexpat ≥ 1.95
- Libyaml ≥ 0.1.4
- Erlang/OTP ≥ 20.0. We recommend using Erlang OTP 25.3, which is the version used in the binary installers and container images.
- OpenSSL ≥ 1.0.0

Other optional libraries are:

- Zlib ≥ 1.2.3, For Zlib Stream Compression
- PAM library, for PAM Authentication
- ImageMagick's Convert program and Ghostscript fonts, for CAPTCHA challenges.
- Elixir ≥ 1.10.3, for Elixir Development. It is recommended Elixir 1.13.4 or higher and Erlang/OTP 23.0 or higher.

If your system splits packages in libraries and development headers, install the development packages too.

For example, in Debian:

```
apt-get install libexpat1-dev libgd-dev libpam0g-dev \
                libsqlite3-dev libwebp-dev libyaml-dev \
                autoconf automake erlang elixir rebar3
```

Download

There are several ways to obtain the ejabberd source code:

- Source code archive from ProcessOne Downloads
- Source code package from ejabberd GitHub Releases
- Latest development code from ejabberd Git repository

The latest development source code can be retrieved from the Git repository using the commands:

```
git clone git://github.com/processone/ejabberd.git
cd ejabberd
```

Compile

The general instructions to compile ejabberd are:

```
./autogen.sh
./configure
make
```

In this example, `./configure` prepares the installed program to run with a user called `ejabberd` that should exist in the system (it isn't recommended to run ejabberd with `root` user):

```
./autogen.sh
./configure --enable-user=ejabberd --enable-mysql
make
```

`make` gets the dependencies and compiles everything.

Note: To build ejabberd, you will need Internet access, as dependencies will be downloaded depending on the selected options.

./configure

The build configuration script supports many options. Get the full list:

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

Options details:

- `--bindir=/` : Specify the path to the user executables (where `epmd` and `iex` are available).
- `--prefix=/` : Specify the path prefix where the files will be copied when running the `make install` command.
- `--with-rebar=/` : Specify the path to rebar, rebar3 or mix

  🏷 added in 20.12 and improved in 24.02

- `--enable-user[=USER]` : Allow this normal system user to execute the ejabberdctl script (see section ejabberdctl), read the configuration files, read and write in the spool directory, read and write in the log directory. The account user and group must exist in the machine before running `make install` . This account needs a HOME directory, because the Erlang cookie file will be created and read there.
- `--enable-group[=GROUP]` : Use this option additionally to `--enable-user` when that account is in a group that doesn't coincide with its username.
- `--enable-all` : Enable many of the database and dependencies options described here, this is useful for Dialyzer checks: --enable-debug --enable-elixir --enable-mysql --enable-odbc --enable-pam --enable-pgsql --enable-redis --enable-sip --enable-sqlite --enable-stun --enable-tools --enable-zlib
- `--disable-debug` : Compile without `+debug_info` .
- `--enable-elixir` : Build ejabberd with Elixir extension support. Works only with rebar3, not rebar2. Requires to have Elixir installed. If interested in Elixir development, you may prefer to use `--with-rebar=mix`

  🏷 improved in 24.02

- `--disable-erlang-version-check` : Don't check Erlang/OTP version.
- `--enable-full-xml` : Use XML features in XMPP stream (ex: CDATA). This requires XML compliant clients).
- `--enable-hipe` : Compile natively with HiPE. This is an experimental feature, and not recommended.
- `--enable-lager` : Use lager Erlang logging tool instead of standard error logger.
- `--enable-latest-deps` : Makes rebar use latest versions of dependencies developed alongside ejabberd instead of version specified in rebar.config. Should be only used when developing ejabberd.
- `--enable-lua` : Enable Lua support, to import from Prosody.

  🏷 added in 21.04

- `--enable-mssql` : Enable Microsoft SQL Server support, this option requires --enable-odbc (see [Supported storages][18]).
- `--enable-mysql` : Enable MySQL support (see [Supported storages][18]).
- `--enable-new-sql-schema` : Use new SQL schema.
- `--enable-odbc` : Enable pure ODBC support.
- `--enable-pam` : Enable the PAM authentication method (see PAM Authentication section).
- `--enable-pgsql` : Enable PostgreSQL support (see [Supported storages][18]).
- `--enable-redis` : Enable Redis support to use for external session storage.
- `--enable-roster-gateway-workaround` : Turn on workaround for processing gateway subscriptions.
- `--enable-sip` : Enable SIP support.
- `--enable-sqlite` : Enable SQLite support (see [Supported storages][18]).
- `--disable-stun` : Disable STUN/TURN support.
- `--enable-system-deps` : Makes rebar use locally installed dependencies instead of downloading them.
- `--enable-tools` : Enable the use of development tools.

  🏷 changed in 21.04

- `--disable-zlib` : Disable Stream Compression (XEP-0138) using zlib.

make

This tool is used to compile ejabberd and perform other tasks:

- Get, update, compile dependencies; clean files
- System install, uninstall
- Build OTP production / development releases
- Development: edoc, options, translations, tags
- Testing: dialyzer, hooks, test, xref

Get the full list and details:

```
make help
```

Install

There are several ways to install and run the ejabberd compiled from source code: system install, building a production release, or building a development release.

System 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 created files and directories depend on the options provided to `./configure`, by default they are:

- `/etc/ejabberd/`: Configuration directory:
- `ejabberd.yml`: ejabberd configuration file (see File Format)
- `ejabberdctl.cfg`: Configuration file of the administration script (see Erlang Runtime System)
- `inetrc`: Network DNS configuration file for Erlang
- `/lib/ejabberd/`:
- `ebin/`: Erlang binary files (*.beam)
- `include/`: Erlang header files (*.hrl)
- `priv/`: Additional files required at runtime
- `bin/`: Executable programs
- `lib/`: Binary system libraries (*.so)
- `msgs/`: Translation files (*.msgs) (see Default Language)
- `/sbin/ejabberdctl`: Administration script (see ejabberdctl)
- `/share/doc/ejabberd/`: Documentation of ejabberd
- `/var/lib/ejabberd/`: Spool directory:
- `.erlang.cookie`: The Erlang cookie file
- `acl.DCD, ...`: Mnesia database spool files (*.DCD, *.DCL, *.DAT)
- `/var/log/ejabberd/`: Log directory (see Logging):
- `ejabberd.log`: ejabberd service log
- `erlang.log`: Erlang/OTP system log

Production Release

🏷 improved in 21.07

You can build an OTP release that includes ejabberd, Erlang/OTP and all the required erlang dependencies in a single tar.gz file. Then you can copy that file to another machine that has the same machine architecture, and run ejabberd without installing anything else.

To build that production release, run:

```
make prod
```

If you provided to `./configure` the option `--with-rebar` to use rebar3 or mix, this will directly produce a tar.gz that you can copy.

This example uses rebar3 to manage the compilation, builds an OTP production release, copies the resulting package to a temporary path, and starts ejabberd there:

```
./autogen.sh
./configure --with-rebar=rebar3
make
make prod
mkdir $HOME/eja-release
tar -xzvf _build/prod/ejabberd-*.tar.gz -C $HOME/eja-release
$HOME/eja-release/bin/ejabberdctl live
```

Development Release

new in 21.07

If you provided to `./configure` the option `--with-rebar` to use rebar3 or mix, you can build an OTP development release.

This is designed to run ejabberd in the local machine for development, manual testing... without installing in the system.

This development release has some customizations: uses a dummy certificate file, if you register the account admin@localhost it has admin rights...

This example uses Elixir's mix to manage the compilation, builds an OTP development release, and starts ejabberd there:

```
./autogen.sh
./configure --with-rebar=mix
make
make dev
_build/dev/rel/ejabberd/bin/ejabberdctl live
```

Specific notes

asdf

When Erlang/OTP (and/or Elixir) is installed using asdf (multiple runtime version manager), it is available only for your account, in `$HOME/.asdf/shims/erl`. In that case, you cannot install ejabberd globally in the system, and you cannot use the `root` account to start it, because that account doesn't have access to erlang.

In that scenario, there are several ways to run/install ejabberd:

- Run a development release locally without installing
- Copy a production release locally
- Use system install, but install it locally:

```
./autogen.sh
./configure --prefix=$HOME/eja-install --enable-user
make
make install
$HOME/eja-install/sbin/ejabberdctl live
```

BSD

The command to compile ejabberd in BSD systems is `gmake`.

You may want to check pkgsrc.se for ejabberd.

Up to ejabberd 23.04, some old scripts where included in ejabberd source for NetBSD compilation, and you can take a look to those files for reference in ejabberd `23.04/examples/mtr/` path.

macOS

If compiling from sources on Mac OS X, you must configure ejabberd to use custom OpenSSL, Yaml, iconv. The best approach is to use Homebrew to install your dependencies, then exports your custom path to let configure and make be aware of them.

```
brew install git erlang elixir openssl expat libyaml libiconv libgd sqlite rebar rebar3 automake autoconf
export LDFLAGS="-L/usr/local/opt/openssl/lib -L/usr/local/lib -L/usr/local/opt/expat/lib"
export CFLAGS="-I/usr/local/opt/openssl/include -I/usr/local/include -I/usr/local/opt/expat/include"
export CPPFLAGS="-I/usr/local/opt/openssl/include/ -I/usr/local/include -I/usr/local/opt/expat/include"
./configure
make
```

Check also the guide for Installing ejabberd development environment on OSX

Start

You can use the `ejabberdctl` command line administration script to start and stop ejabberd. Some examples, depending on your installation method:

- When installed in the system:

  ```
  ejabberdctl start
  /sbin/ejabberdctl start
  ```

- When built an OTP production release:

  ```
  _build/prod/rel/ejabberd/bin/ejabberdctl start
  _build/prod/rel/ejabberd/bin/ejabberdctl live
  ```

- Start interactively without installing or building OTP release:

  ```
  make relive
  ```

Install ejabberd on macOS

Homebrew

[Homebrew](#) is a package manager for macOS that aims to port the many Unix & Linux software that is not easily available or compatible. Homebrew installation is simple and the instruction is available on its website.

Check also the guide for [Installing ejabberd development environment on OSX](#)

The ejabberd configuration included in Homebrew's ejabberd has as default domain `localhost` , and has already granted administrative privileges to the account `admin@localhost` .

1. Once you have Homebrew installed, open Terminal. Run

   ```
   brew install ejabberd
   ```

   This should install the latest or at most the one-before-latest version of ejabberd. The installation directory should be reported at the end of this process, but usually the main executable is stored at `/usr/local/sbin/ejabberdctl` .

2. Start ejabberd in interactive mode, which prints useful messages in the Terminal.

   ```
   /usr/local/sbin/ejabberdctl live
   ```

3. Create the account `admin@localhost` with password set as `password` :

   ```
   /usr/local/sbin/ejabberdctl register admin localhost password
   ```

4. Now you can go to the web dashboard at `http://localhost:5280/admin/` and fill the username field with the full account JID, for example `admin@localhost` , then fill the password field with that account's `password` .

5. Without configuration there's not much to see here, therefore the next step is to get to know [how to configure ejabberd](#).

Installing ejabberd development environment on OSX

This short guide will show you how to compile ejabberd from source code on Mac OS X, and get users chatting right away.

Before you start

ejabberd is supported on Mac OS X 10.6.8 and later. Before you can compile and run ejabberd, you also need the following to be installed on your system:

- Gnu Make and GCC (the GNU Compiler Collection). To ensure that these are installed, you can install the Command Line Tools for Xcode, available via Xcode or from the Apple Developer website.
- Git
- Erlang/OTP 19.1 or higher. We recommend using Erlang 21.2.
- Autotools

Homebrew

An easy way to install some of the dependencies is by using a package manager, such as Homebrew – the Homebrew commands are provided here:

- Git: `brew install git`
- Erlang /OTP: `brew install erlang`
- Elixir: `brew install elixir`
- Autoconf: `brew install autoconf`
- Automake: `brew install automake`
- Openssl: `brew install openssl`
- Expat: `brew install expat`
- Libyaml: `brew install libyaml`
- Libiconv: `brew install libiconv`
- Sqlite: `brew install sqlite`
- GD: `brew install gd`
- Rebar: `brew install rebar rebar3`

You can install everything with a single command:

```
brew install erlang elixir openssl expat libyaml libiconv libgd sqlite rebar rebar3 automake autoconf
```

Installation

To build and install ejabberd from source code, do the following:

1. Clone the Git repository: `git clone git@github.com:processone/ejabberd.git`
2. Go to your ejabberd build directory: `cd ejabberd`
3. Run the following commands, assuming you want to install your ejabberd deployment into your home directory:

```
chmod +x autogen.sh
./autogen.sh
export LDFLAGS="-L/usr/local/opt/openssl/lib -L/usr/local/lib -L/usr/local/opt/expat/lib"
export CFLAGS="-I/usr/local/opt/openssl/include/ -I/usr/local/include -I/usr/local/opt/expat/include"
export CPPFLAGS="-I/usr/local/opt/openssl/include/ -I/usr/local/include -I/usr/local/opt/expat/include"
./configure --prefix=$HOME/my-ejabberd --enable-sqlite
make && make install
```

Note that the previous command reference the previously installed dependencies from Homebrew.

Running ejabberd

- From your ejabberd build directory, go to the installation directory: `cd $HOME/my-ejabberd`

- To start the ejabberd server, run the following command: `sbin/ejabberdctl start`

- To verify that ejabberd is running, enter the following: `sbin/ejabberdctl status` If the server is running, response should be as follow:

```
$ sbin/ejabberdctl status
The node ejabberd@localhost is started with status: started
ejabberd 14.12.40 is running in that node
```

- To connect to the ejabberd console after starting the server: `sbin/ejabberdctl debug`

- Alternatively, you can also run the server in interactive mode: `sbin/ejabberdctl live`

Registering a user

The default XMPP domain served by ejabberd right after the build is `localhost` . This is different from the IP address, DNS name of the server. It means remote users can connect to ejabberd even if it is running on your machine with `localhost` XMPP domain, by using your computer IP address or DNS name. This can prove handy in development phase to get more testers.

Adium

Adium is a popular XMPP client on OSX. You can use it

1. Launch Adium. If the Adium Setup Assistant opens, close it.
2. In the **Adium** menu, select **Preferences**, and then select the **Accounts** tab.
3. Click the **+** button and select **XMPP (Jabber)**.
4. Enter a Jabber ID (for example, "user1@localhost") and password, and then click **Register New Account**.
5. In the **Server** field, enter the following:
6. Users registering on the computer on which ejabberd is running: `localhost`
7. Users registering from a different computer: the ejabberd server's IP address
8. Click **Request New Account**.

After registration, the user will connect automatically.

Registered users wishing to add an existing account to Adium should enter the ejabberd server's IP address in the **Connect Server** field on the **Options** tab.

Command line

You can register a user with the `ejabberdctl` utility: `ejabberdctl register user domain password`

For example: `ejabberdctl register user1 localhost myp4ssw0rd`

Domains

To use your system's domain name instead of localhost, edit the following ejabberd configuration file: `$HOME/my-ejabberd/etc/ejabberd.yml` (point to the place of your real installation).

Note: You may find example `ejabberd.cfg` files. This is the old obsolete format for configuration file. You can ignore the and focus on the new and more user friendly Yaml format.

Find the line listing the hosts:

```
hosts:
  - "localhost"
```

Replace `localhost` with your XMPP domain name, for example:

```
hosts:
  - "example.org"
```

Save the configuration file and restart the ejabberd server. A user's Jabber ID will then use the domain instead of localhost, for example: `user1@example.org`

You can also configure multiple (virtual) domains for one server:

```
hosts:
  - "example1.org"
  - "example2.org"
```

Get chatting

Users that are registered on your server can now add their accounts in a chat application like Adium (specifying either the server's IP address or domain name), add each other as contacts, and start chatting.

Next Steps

Starting ejabberd

Depending on how you installed ejabberd, it may be started automatically by the operating system at system boot time.

You can use the `ejabberdctl` command line administration script to start and stop ejabberd, check its status and many other administrative tasks.

If you provided the configure option `--enable-user=USER` (see compilation `options`, you can execute `ejabberdctl` with either that system account or root.

Usage example:

```
prompt> ejabberdctl start

prompt> ejabberdctl status
The node ejabberd@localhost is started with status: started
ejabberd is running in that node

prompt> ejabberdctl stop
```

If ejabberd doesn't start correctly and a crash dump file is generated, there was a severe problem. You can try to start ejabberd in interactive mode with the command `bin/ejabberdctl live` to see the error messages provided by Erlang and identify the exact the problem.

The `ejabberdctl` administration script is included in the `bin` directory in the Linux Installers and Docker image.

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

Autostart on Linux

If you compiled ejabberd from source code or some other method that doesn't setup autostarting ejabberd, you can try this method.

On a *nix system, create a system user called 'ejabberd', give it write access to the directories `database/` and `logs/`, and set that as home.

If you want ejabberd to be started as daemon at boot time with that user, copy `ejabberd.init` from the `bin` directory to something like `/etc/init.d/ejabberd`. Then you can call `/etc/inid.d/ejabberd start` to start the server.

Or if you have a `systemd` distribution:

1. copy `ejabberd.service` to `/etc/systemd/system/`

2. run `systemctl daemon-reload`

3. run `systemctl enable ejabberd.service`

4. To start the server, you can run `systemctl start ejabberd`

When ejabberd is started, the processes that are started in the system are `beam` or `beam.smp`, and also `epmd`. For more information regarding `epmd` consult the section relating to epmd.

Administration Account

Some ejabberd installation methods ask you details for the first account, and take care to register that account and grant it administrative rights; in that case you can skip this section.

After installing ejabberd from source code or other methods, you may want to register the first XMPP account and grant it administrative rights:

1. Register an XMPP account on your ejabberd server. For example, if `example.org` is configured in the hosts section in your ejabberd configuration file, then you may want to register an account with JID `admin1@example.org`.

   There are two ways to register an XMPP account in ejabberd:

   • Using an XMPP client and In-Band Registration.

   • Using ejabberdctl:

   ```
   ejabberdctl register admin1 example.org password
   ```

2. Edit the ejabberd configuration file to give administration rights to the XMPP account you registered:

   ```
   acl:
     admin:
       user: admin1@example.org

   access_rules:
     configure:
       allow: admin
   ```

   You can grant administrative privileges to many XMPP accounts, and also to accounts in other XMPP servers.

3. Restart ejabberd to load the new configuration, or run the reload_config command.

4. Open the Web Admin page in your favourite browser. The exact address depends on your ejabberd configuration, and may be http://localhost:5280/admin/, https://localhost:5443/admin/, https://localhost:5280/admin/ ...

5. Your web browser shows a login window. Introduce the **full JID**, in this example `admin1@example.org`, and the account password. If the web address hostname is the same that the account JID, you can provide simply the username instead of the full JID: `admin1`. See Web Admin for details.

Configuring ejabberd

Now that you got ejabberd installed and running, it's time to configure it to your needs. You can follow on the Configuration section and take also a look at the Tutorials.

Configure

Configuring ejabberd

Here are the main entry points to learn more about ejabberd configuration. ejabberd is extremely powerful and can be configured in many ways with many options.

Do not let this complexity scare you. Most of you will be fine with default config file (or light changes).

Tutorials for first-time users:

- How to move to ejabberd XMPP server
- How to set up ejabberd video & voice calling (STUN/TURN)
- How to configure ejabberd to get 100% in XMPP compliance test

Detailed documentation in sections:

- File Format
- Basic Configuration: hosts, acl, logging...
- Authentication: auth_method
- Databases
- LDAP
- Listen Modules: c2s, s2s, http, sip, stun...
- Listen Options
- Top-Level Options
- Modules Options

There's also a copy of the old configuration document which was used up to ejabberd 20.03.

File format

Yaml File Format

`ejabberd` loads its configuration file during startup. This configuration file is written in `YAML` format, and its file name MUST have ".yml" or ".yaml" extension. This helps ejabberd to differentiate between this new format and the legacy configuration file format.

Please, consult `ejabberd.log` for configuration errors. `ejabberd` will report syntax related errors, as well as complains about unknown options and invalid values. Make sure you respect indentation (YAML is sensitive to this) or you will get pretty cryptic errors.

Note that `ejabberd` never edits the configuration file. If you are changing parameters at runtime from web admin interface, you will need to apply them to configuration file manually. This is to prevent messing up with your config file comments, syntax, etc.

Reload at Runtime

You can modify the `ejabberd` configuration file and reload it at runtime: the changes you made are applied immediately, no need to restart ejabberd. This applies to adding, changing or removing vhosts, listened ports, modules, ACLs or any other options.

How to do this?

1. Let's assume your ejabberd server is already running
2. Modify the configuration file
3. Run the reload_config command
4. ejabberd will read that file, check its YAML syntax is valid, check the options are valid and known...
5. If there's any problem in the configuration file, the reload is aborted and an error message is logged with details, so you can fix the problem.
6. If the file is right, it detects the changed options, and applies them immediately (add/remove hosts, add/remove modules, ...)

Legacy Configuration File

In previous `ejabberd` version the configuration file should be written in Erlang terms. The format is still supported, but it is highly recommended to convert it to the new YAML format with the convert_to_yaml API command using ejabberdctl.

If you want to specify some options using the old Erlang format, you can set them in an additional cfg file, and include it using the `include_config_file` option, see Include Additional Files for the option description and a related example in Restrict Execution with AccessCommands.

Include Additional Files

The option include_config_file in a configuration file instructs `ejabberd` to include other configuration files immediately.

This is a basic example:

```
include_config_file: /etc/ejabberd/additional.yml
```

In this example, the included file is not allowed to contain a `listen` option. If such an option is present, the option will not be accepted. The file is in a subdirectory from where the main configuration file is.

```
include_config_file:
  ./example.org/additional_not_listen.yml:
    disallow: [listen]
```

Please notice that options already defined in the main configuration file cannot be redefined in the included configuration files. But you can use host_config and append_host_config as usual (see Virtual Hosting).

In this example, `ejabberd.yml` defines some ACL for the whole ejabberd server, and later includes another file:

```
acl:
  admin:
    user:
      - admin@localhost
include_config_file:
  /etc/ejabberd/acl.yml
```

The file `acl.yml` can add additional administrators to one of the virtual hosts:

```
append_host_config:
  localhost:
    acl:
      admin:
        user:
          - bob@localhost
          - jan@localhost
```

Macros in Configuration File

In the `ejabberd` configuration file, it is possible to define a macro for a value and later use this macro when defining an option.

A macro is defined using the define_macro option.

This example shows the basic usage of a macro:

```
define_macro:
  LOG_LEVEL_NUMBER: 5
loglevel: LOG_LEVEL_NUMBER
```

The resulting option interpreted by `ejabberd` is: `loglevel: 5`.

This example shows that values can be any arbitrary YAML value:

```
define_macro:
  USERBOB:
    user:
      - bob@localhost
acl:
  admin: USERBOB
```

The resulting option interpreted by `ejabberd` is:

```
acl:
  admin:
    user:
      - bob@localhost
```

This complex example:

```
define_macro:
  NUMBER_PORT_C2S: 5222
  NUMBER_PORT_HTTP: 5280
listen:
  -
    port: NUMBER_PORT_C2S
    module: ejabberd_c2s
  -
    port: NUMBER_PORT_HTTP
    module: ejabberd_http
```

produces this result after being interpreted:

```
listen:
  -
    port: 5222
    module: ejabberd_c2s
  -
    port: 5280
    module: ejabberd_http
```

Basic Configuration

XMPP Domains

Host Names

`ejabberd` supports managing several independent XMPP domains on a single ejabberd instance, using a feature called virtual hosting.

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

Of course, the `hosts` list can contain just one domain if you do not want to host multiple XMPP domains on the same instance.

Examples:

- Serving one domain:

```
hosts: [example.org]
```

- Serving three domains:

```
hosts:
  - example.net
  - example.com
  - jabber.somesite.org
```

Virtual Hosting

When managing several XMPP domains in a single instance, those domains are truly independent. It means they can even have different configuration parameters.

Options can be defined separately for every virtual host using the `host_config` 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
  example.com:
    auth_method: ldap
    ldap_servers:
      - localhost
    ldap_uids:
      - uid
    ldap_rootdn: "dc=localdomain"
    ldap_password: ""
```

- Domain `example.net` is using SQL 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: sql
    sql_type: odbc
    sql_server: "DSN=ejabberd;UID=ejabberd;PWD=ejabberd"
  example.com:
    auth_method: ldap
    ldap_servers:
      - localhost
      - otherhost
    ldap_uids:
      - uid
    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` use `append_host_config` with the same syntax.

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:   {}

append_host_config:
  ## Add some modules to vhost one:
  one.example.org:
    modules:
      mod_muc:
        host: conference.one.example.org
      mod_ping: {}
  ## Add a module just to vhost two:
  two.example.org:
    modules:
      mod_muc:
        host: conference.two.example.org
```

Logging

`ejabberd` configuration can help a lot by having the right amount of logging set up.

There are several toplevel options to configure logging:

- `loglevel` : Verbosity of log files generated by ejabberd.

- `hide_sensitive_log_data` : Privacy option to disable logging of IP address or sensitive data.

- `log_modules_fully` : Modules that will log everything independently from the general `loglevel` option.

- `log_rotate_size`

- `log_rotate_count` : Setting count to N keeps N rotated logs. Setting count to 0 does not disable rotation, it instead rotates the file and keeps no previous versions around. Setting size to X rotate log when it reaches X bytes.

- `log_burst_limit_count`

- `log_burst_limit_window_time`

The values in default configuration file are:

```
log_rotate_size: 10485760
log_rotate_count: 1
```

For example, log warning and higher messages, but all c2s messages, and hide sensitive data:

```
loglevel: warning
hide_sensitive_log_data: true
log_modules_fully: [ejabberd_c2s]
```

Default Language

The `language` option defines the default language of server strings that can be seen by XMPP clients. If a XMPP client does not support `xml:lang` , `ejabberd` uses the language specified in this option.

The option syntax is:

language: Language : The default value is `en` . In order to take effect there must be a translation file `Language.msg` in `ejabberd` 's `msgs` directory.

For example, to set Russian as default language:

```
language: ru
```

The page Internationalization and Localization provides more details.

CAPTCHA

Some `ejabberd` modules can be configured to require a CAPTCHA challenge on certain actions, for instance mod_block_strangers, mod_muc, mod_register, and mod_register_web. If the client does not support CAPTCHA Forms (`XEP-0158`), a web link is provided so the user can fill the challenge in a web browser.

Example scripts are provided that generate the image using ImageMagick's Convert program and Ghostscript fonts. Remember to install those dependencies: in Debian install the `imagemagick` and `gsfonts` packages; in container images check their documentation for details.

The relevant top-level options are:

- `captcha_cmd` : `Path | Module` : Full path to a script that generates the image, or name of a module that supports generating CAPTCHA images (mod_ecaptcha, mod_captcha_rust). The default value disables the feature: `undefined`
- `captcha_url` : `URL | auto` : An URL where CAPTCHA requests should be sent, or `auto` to determine the URL automatically. The default value is `auto` .

And finally, configure `request_handlers` for the `ejabberd_http` listener with a path handled by `ejabberd_captcha` , where the CAPTCHA images will be served.

Example configuration:

```
hosts: [example.org]

captcha_cmd: /lib/ejabberd/priv/bin/captcha.sh
# captcha_cmd: /opt/ejabberd-23.01/lib/captcha.sh
# captcha_cmd: mod_ecaptcha

captcha_url: auto
## captcha_url: http://example.org:5280/captcha
## captcha_url: https://example.org:443/captcha
## captcha_url: http://example.com/captcha

listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /captcha: ejabberd_captcha
```

ACME

ACME is used to automatically obtain SSL certificates for the domains served by ejabberd, which means that certificate requests and renewals are performed to some CA server (aka "ACME server") in a fully automated mode.

Setting up ACME

In ejabberd, ACME is configured using the `acme` top-level option, check there the available options and example configuration.

The automated mode is enabled by default. However, some configuration of ejabberd is still required, because ACME requires HTTP challenges: an ACME remote server will connect to your ejabberd server on HTTP port 80 during certificate issuance.

For that reason you must have an `ejabberd_http` listener with TLS disabled handling an "ACME well known" path. For example:

```
listen:
  -
    module: ejabberd_http
```

```
    port: 5280
    tls: false
    request_handlers:
      /.well-known/acme-challenge: ejabberd_acme
```

Note that the ACME protocol **requires** challenges to be sent on port 80. Since this is a privileged port, ejabberd cannot listen on it directly without root privileges. Thus you need some mechanism to forward port 80 to the port defined by the listener (port 5280 in the example above). There are several ways to do this: using NAT, setcap (Linux only), or HTTP front-ends (e.g. `sslh`, `nginx`, `haproxy` and so on). Pick one that fits your installation the best, but **DON'T** run ejabberd as root.

If you see errors in the logs with ACME server problem reports, it's **highly** recommended to change `ca_url` option in the `acme` top-level option to the URL pointing to some staging ACME environment, fix the problems until you obtain a certificate, and then change the URL back and retry using `request-certificate` ejabberdctl command (see below). This is needed because ACME servers typically have rate limits, preventing you from requesting certificates too rapidly and you can get stuck for several hours or even days. By default, ejabberd uses Let's Encrypt authority. Thus, the default value of `ca_url` option is `https://acme-v02.api.letsencrypt.org/directory` and the staging URL will be `https://acme-staging-v02.api.letsencrypt.org/directory`:

```
acme:
  ## Staging environment
  ca_url: https://acme-staging-v02.api.letsencrypt.org/directory
  ## Production environment (the default):
  # ca_url: https://acme-v02.api.letsencrypt.org/directory
```

The automated mode can be disabled by setting `auto` option to `false` in the `acme` top-level option:

```
acme:
  auto: false
```

In this case automated renewals are still enabled, however, in order to request a new certificate, you need to run request_certificate API command:

```
ejabberdctl request-certificate all
```

If you only want to request certificates for a subset of the domains, run:

```
ejabberdctl request-certificate domain.tld,pubsub.domain.tld,server.com,conference.server.com,...
```

You can view the certificates obtained using ACME and list_certificates:

```
$ ejabberdctl list-certificates
domain.tld /path/to/cert/file1 true
server.com /path/to/cert/file2 false
```

The output is mostly self-explained: every line contains the domain, the corresponding certificate file, and whether this certificate file is used or not. A certificate might not be used for several reasons: mostly because ejabberd detects a better certificate (i.e. not expired, or having a longer lifetime). It's recommended to revoke unused certificates if they are not yet expired (see below).

At any point you can revoke a certificate using revoke_certificate: pick the certificate file from the listing above and run:

```
ejabberdctl revoke-certificate /path/to/cert/file
```

If the commands return errors, consult the log files for details.

ACME implementation details

In nutshell, certification requests are performed in two phases. Firstly, ejabberd creates an account at the ACME server. That is an EC private key. Secondly, a certificate is requested. In the case of a revocation, no account is used - only a certificate in question is needed. All information is stored under `acme` directory inside spool directory of ejabberd (typically `/var/lib/ejabberd`). An example content of the directory is the following:

```
$ tree /var/lib/ejabberd
/var/lib/ejabberd
├── acme
│   ├── account.key
│   └── live
│       ├── 251ce180d964e98a2f18b65504df2ab7c55943e2
```

```
|          └── 93816a8429ebbaa75574eb3f59d4a806b67d6917
...
```

Here, `account.key` is the EC private key used to identify the ACME account. You can inspect its content using `openssl` command:

```
openssl ec -text -noout -in /var/lib/ejabberd/acme/account.key
```

Obtained certificates are stored under `acme/live` directory. You can inspect any of the certificates using `openssl` command as well:

```
openssl x509 -text -noout -in /var/lib/ejabberd/acme/live/251ce180d964e98a2f18b65504df2ab7c55943e2
```

In the case of errors, you can delete the whole `acme` directory - ejabberd will recreate its content on next certification request. However, don't delete it too frequently - usually there is a rate limit on the number of accounts and certificates an ACME server creates. In particular, for Let's Encrypt the limits are described here.

Access Rights

This section describes new ACL syntax introduced in ejabberd 16.06. For old access rule and ACL syntax documentation, please refer to configuration document archive

ACL

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

```
acl:
  ACLName:
    ACLType: ACLValue
```

`ACLType: ACLValue` can be one of the following:

- `all` : Matches all JIDs. Example:

  ```
  acl:
    world: all
  ```

- `user: Username` : Matches the user with the name `Username` on any of the local virtual host. Example:

  ```
  acl:
    admin:
      user: yozhik
  ```

- `user: {Username: Server} | Jid` : Matches the user with the JID `Username@Server` and any resource. Example:

  ```
  acl:
    admin:
      - user:
          yozhik@example.org
      - user: peter@example.org
  ```

- `server: Server` : Matches any JID from server `Server` . Example:

  ```
  acl:
    exampleorg:
      server: example.org
  ```

- `resource: Resource` : Matches any JID with a resource `Resource` . Example:

  ```
  acl:
    mucklres:
      resource: muckl
  ```

- `shared_group: Groupname` : Matches any member of a Shared Roster Group with name `Groupname` in the virtual host. Example:

  ```
  acl:
    techgroupmembers:
      shared_group: techteam
  ```

- `shared_group: {Groupname: Server}` : Matches any member of a Shared Roster Group with name `Groupname` in the virtual host `Server` . Example:

  ```
  acl:
    techgroupmembers:
      shared_group:
        techteam: example.org
  ```

- `ip: Network` : Matches any IP address from the `Network` . Example:

  ```
  acl:
    loopback:
      ip:
        - 127.0.0.0/8
        - "::1"
  ```

- `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} | JidRegexp` : Matches any user with a name that matches `Regexp` at server `Server` . Example:

  ```
  acl:
    tests:
      user_regexp:
        - "^test1": example.org
        - "^test2@example.org"
  ```

- `server_regexp: Regexp` : Matches any JID from the server that matches `Regexp` . Example:

  ```
  acl:
    icq:
      server_regexp: "^icq\\."
  ```

- `resource_regexp: Regexp` : Matches any JID with a resource that matches `Regexp` . Example:

```
acl:
  icq:
    resource_regexp: "^laptop\\."
```

- **node_regexp: {UserRegexp: ServerRegexp}** : Matches any user with a name that matches `UserRegexp` at any server that matches `ServerRegexp` . Example:

```
acl:
  yozhik:
    node_regexp:
      "^yozhik$": "^example.(com|org)$"
```

- **user_glob: Glob** :
- **user_glob: {Glob: Server}** :
- **server_glob: Glob** :
- **resource_glob: Glob** :
- **node_glob: {UserGlob: ServerGlob}** : 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 `ACLName` are pre-defined:

- **all** : Matches any JID.
- **none** : Matches no JID.

Access Rules

The `access_rules` option is used to allow or deny access to different services. The syntax is:

```
access_rules:
  AccessName:
    - allow|deny: ACLRule|ACLDefinition
```

Each definition may contain arbitrary number of `- allow` or `- deny` sections, and each section can contain any number of acl rules (as defined in previous section, it recognizes one additional rule `acl: RuleName` that matches when acl rule named `RuleName` matches). If no rule or definition is defined, the rule `all` is applied.

Definition's `- allow` and `- deny` sections are processed in top to bottom order, and first one for which all listed acl rules matches is returned as result of access rule. If no rule matches `deny` is returned.

To simplify configuration two shortcut version are available: `- allow: acl` and `- allow` , example below shows equivalent definitions where short or long version are used:

```
access_rules:
  a_short: admin
  a_long:
    - acl: admin
  b_short:
    - deny: banned
    - allow
  b_long:
    - deny:
      - acl: banned
    - allow:
      - all
```

If you define specific Access rights in a virtual host, remember that the globally defined Access rights have precedence over those. This means that, in case of conflict, the Access granted or denied in the global server is used and the Access of a virtual host doesn't have effect.

Example:

```
access_rules:
  configure:
    - allow: admin
  something:
    - deny: someone
    - allow
  s2s_banned:
    - deny: problematic_hosts
    - deny:
      - acl: banned_forever
    - deny:
      - ip: 222.111.222.111/32
    - deny:
      - ip: 111.222.111.222/32
    - allow
  xmlrpc_access:
    - allow:
      - user: peter@example.com
    - allow:
      - user: ivone@example.com
    - allow:
      - user: bot@example.com
      - ip: 10.0.0.0/24
```

The following `AccessName` are pre-defined:

- `all` : Always returns the value '`allow`'.

- `none` : Always returns the value '`deny`'.

Shaper Rules

The `shaper_rules` top-level option declares shapers to use for matching user/hosts. The syntax is:

```
shaper_rules:
  ShaperRuleName:
    - Number|ShaperName: ACLRule|ACLDefinition
```

Semantic is similar to that described in Access Rights section, only difference is that instead using `- allow` or `- deny` , name of shaper or number should be used.

Examples:

```
shaper_rules:
  connections_limit:
    - 10:
      - user: peter@example.com
    - 100: admin
    - 5
  download_speed:
    - fast: admin
    - slow: anonymous_users
    - normal
  log_days: 30
```

Limiting Opened Sessions

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:

```
shaper_rules:
  max_user_sessions:
    - Number: ACLRule|ACLDefinition
```

This example limits the number of sessions per user to 5 for all users, and to 10 for admins:

```
shaper_rules:
  max_user_sessions:
```

```
        - 10: admin
        - 5
```

Connections to Remote Server

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

The syntax is:

```
shaper_rules:
  max_s2s_connections: MaxNumber
```

For example, let's allow up to 3 connections with each remote server:

```
shaper_rules:
  max_s2s_connections: 3
```

Shapers

The `shaper` top-level option defines limitations in the connection traffic. The basic syntax is:

```
shaper:
  ShaperName: 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.

This example defines a shaper with name `normal` that limits traffic speed to 1,000bytes/second, and another shaper with name `fast` that limits traffic speed to 50,000bytes/second:

```
shaper:
  normal: 1000
  fast: 50000
```

You can use the full syntax to set the `BurstSize` too:

```
shaper:
  ShaperName:
    rate: Rate
    burst_size: BurstSize
```

With `BurstSize` you can allow client to send more data, but its amount can be clamped reasonably. Each connection is allowed to send `BurstSize` of data before processing is delayed, and that amount is replenished by `Rate` each second, but never more than what `BurstSize` allows. This allows the client to send quite a bit of data at once, but still have limited amount of data to send on constant basis.

In this example, the `normal` shaper has `Rate` set to `1000` and the `BurstSize` takes that same value. The `not_normal` shaper has the same `Rate` that before, and sets a higher `BurstSize`:

```
shaper:
  normal: 1000
  not_normal:
    rate: 1000
    burst_size: 20000
```

Authentication

Supported Methods

The authentication methods supported by `ejabberd` are:

- `internal` — See section [Internal](#).
- `external` — See section [External Script](#).
- `ldap` — See section [LDAP](#).
- `sql` — See section [Relational Databases](#).
- `anonymous` — See section [Anonymous Login and SASL Anonymous](#).
- `pam` — See section [Pam Authentication](#).
- `jwt` — See section [JWT Authentication](#).

The top-level option [auth_method](#) defines the authentication methods that are used for user authentication. The option syntax is:

```
auth_method: [Method1, Method2, ...]
```

When the `auth_method` option is omitted, `ejabberd` relies on the default database which is configured in `default_db` option. If this option is not set neither, then the default authentication method will be `internal`.

Account creation is only supported by `internal`, `external` and `sql` auth methods.

General Options

The top-level option [auth_password_format](#) allows to store the passwords in SCRAM format, see the [SCRAM](#) section.

Other top-level options that are relevant to the authentication configuration: [disable_sasl_mechanisms](#), [fqdn](#).

Authentication caching is enabled by default, and can be disabled in a specific vhost with the option [auth_use_cache](#). The global authentication cache can be configured for all the authentication methods with the global top-level options: [auth_cache_missed](#), [auth_cache_size](#), [auth_cache_life_time](#). For example:

```
auth_cache_size: 1500
auth_cache_life_time: 10 minutes

host_config:
  example.org:
    auth_method: [internal]
  example.net:
    auth_method: [ldap]
    auth_use_cache: false
```

Internal

`ejabberd` uses its internal Mnesia database as the default authentication method. The value `internal` will enable the internal authentication method.

To store the passwords in SCRAM format instead of plaintext, see the [SCRAM](#) section.

Examples:

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

```
host_config:
  example.org:
    auth_method: [internal]
  example.net:
    auth_method: [ldap]
```

- To use internal authentication with hashed passwords on all virtual hosts:

```
auth_method: internal
auth_password_format: scram
```

External Script

In the `external` authentication method, ejabberd uses a custom script to perform authentication tasks. The server administrator can write that external authentication script in any programming language.

Please check some example scripts, and the details on the interface between ejabberd and the script in the Developers > Internals > External Authentication section.

Options:

- extauth_pool_name
- extauth_pool_size
- extauth_program

Please note that caching interferes with the ability to maintain multiple passwords per account. So if your authentication mechanism supports application-specific passwords, caching must be disabled in the host that uses this authentication method with the option auth_use_cache.

This example sets external authentication, specifies the extauth script, disables caching, and starts three instances of the script for each virtual host defined in ejabberd:

```
auth_method: [external]
extauth_program: /etc/ejabberd/JabberAuth.class.php
extauth_pool_size: 3
auth_use_cache: false
```

Anonymous Login and SASL Anonymous

The `anonymous` authentication method enables two modes for anonymous authentication:

`Anonymous login`: This is a standard login, that use the classical login and password mechanisms, but where password is accepted or preconfigured for all anonymous users. This login is compliant with SASL authentication, password and digest non-SASL authentication, so this option will work with almost all XMPP clients

`SASL Anonymous`: This is a special SASL authentication mechanism that allows to login without providing username or password (see `XEP-0175`). The main advantage of SASL Anonymous is that the protocol was designed to give the user a login. This is useful to avoid in some case, where the server has many users already logged or registered and when it is hard to find a free username. The main disadvantage is that you need a client that specifically supports the SASL Anonymous protocol.

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 Virtual Hosting):

- allow_multiple_connections
- anonymous_protocol

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_protoco: 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:

```
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
```

There are more configuration examples and XMPP client example stanzas in `Anonymous users support`.

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
- pam_userinfotype

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.

- If you use `pam_winbind` to authorize against a Windows Active Directory, then `/etc/nsswitch.conf` must be configured to use `winbind` as well.

JWT Authentication

`ejabberd` supports authentication using JSON Web Token (JWT). When enabled, clients send signed tokens instead of passwords, which are checked using a private key specified in the jwt_key option. JWT payload must look like this:

```
{
  "jid": "test@example.org",
  "exp": 1564436511
}
```

Options:

- jwt_key
- jwt_auth_only_rule
- jwt_jid_field

Example:

```
auth_method: jwt
jwt_key: /path/to/jwt/key
```

In this example, admins can use both JWT and plain passwords, while the rest of users can use only JWT.

```
# the order is important here, don't use [sql, jwt]
auth_method: [jwt, sql]

access_rules:
  jwt_only:
    deny: admin
    allow: all

jwt_auth_only_rule: jwt_only
```

Please notice that, when using JWT authentication, mod_offline will not work. With JWT authentication the accounts do not exist in the database, and there is no way to know if a given account exists or not.

For more information about JWT authentication, you can check a brief tutorial in the ejabberd 19.08 release notes.

SCRAM

The top-level option `auth_password_format` defines in what format the users passwords are stored: SCRAM format or plaintext format.

The top-level option `auth_scram_hash` defines the hash algorithm that will be used to scram the password.

ejabberd supports channel binding to the external channel, allowing the clients to use `-PLUS` authentication mechanisms.

In summary, depending on the configured options, ejabberd supports:

- `SCRAM_SHA-1(-PLUS)`
- `SCRAM_SHA-256(-PLUS)`
- `SCRAM_SHA-512(-PLUS)`

For details about the client-server communication when using SCRAM, refer to SASL Authentication and SCRAM.

Internal storage

When ejabberd starts with internal auth method and SCRAM password format configured:

```
auth_method: internal
auth_password_format: scram
```

and detects that there are plaintext passwords stored, they are automatically converted to SCRAM format:

```
[info] Passwords in Mnesia table 'passwd' will be SCRAM'ed
[info] Transforming table 'passwd', this may take a while
```

SQL Database

Please note that if you use SQL auth method and SCRAM password format, the plaintext passwords already stored in the database are not automatically converted to SCRAM format.

To convert plaintext passwords to SCRAM format in your database, use the convert_to_scram command:

```
ejabberdctl convert_to_scram example.org
```

Foreign authentication

Note on SCRAM using and foreign authentication limitations: when using the SCRAM password format, it is not possible to use foreign authentication method in ejabberd, as the real password is not known.

Foreign authentication are use to authenticate through various bridges ejabberd provide. Foreign authentication includes at the moment SIP and TURN auth support and they will not be working with SCRAM.

Database Configuration

Supported storages

`ejabberd` uses its internal Mnesia database by default. However, it is possible to use a relational database, key-value storage 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`:

- `Mnesia`
- `MySQL`. Check the tutorial Using ejabberd with MySQL
- Any `ODBC` compatible database
- `PostgreSQL`
- `MS SQL Server/SQL Azure`
- `SQLite`
- `Redis` (only for transient data)

Please check LDAP Configuration section for documentation about using LDAP.

Database Schema

When using external database backend, ejabberd does not create schema and tables by itself. If you plan to use MySQL, PostgreSQL, MS SQL or SQLite, you must create the schema before you run ejabberd.

- If installing ejabberd from sources, you will find sql script for your backend in the installation directory. By default: `/usr/local/lib/ejabberd/priv/sql`
- If installing ejabberd from Process-One installer, the init scripts are located in the ejabberd's installation path under `<base>/lib/ejabberd*/priv/sql`

If using MySQL or PostgreSQL, you can choose between the default or the new schemas.

See ejabberd SQL Database Schema for details on database schemas.

Virtual Hosting

Important note about virtual hosting: if you define several domains in ejabberd.yml (see section Host Names), you probably want that each virtual host uses a different configuration of database, authentication and storage, so that usernames do not conflict and mix between different virtual hosts. For that purpose, the options described in the next sections must be set inside a host_config for each vhost (see section Virtual Hosting). For example:

```
host_config:
  public.example.org:
    sql_type: pgsql
    sql_server: localhost
    sql_database: database-public-example-org
    sql_username: ejabberd
    sql_password: password
    auth_method: [sql]
```

Default database

You can simplify the configuration by setting the default database. This can be done with the `default_db` top-level option:

`default_db: mnesia|sql` : This will define the default database for a module lacking `db_type` option or if `auth_method` option is not set.

Relational Databases

Default and New Schemas

There are two database schemas available in ejabberd: the default schema is preferable when serving one massive domain, the new schema is preferable when serving many small domains.

The default schema stores only one XMPP domain in the database. The XMPP domain is not stored as this is the same for all the accounts, and this saves space in massive deployments. However, to handle several domains, you have to setup one database per domain and configure each one independently using host_config, so in that case you may prefer the new schema.

The new schema stores the XMPP domain in a new column `server_host` in the database entries, so it allows to handle several XMPP domains in a single ejabberd database. Using this schema is preferable when serving several XMPP domains and changing domains from time to time. However, if you have only one massive domain, you may prefer to use the default schema.

To use the new schema, edit the ejabberd configuration file and enable new_sql_schema top-level option:

```
new_sql_schema: true
```

Now, when creating the new database, remember to use the proper SQL schema! For example, if you are using MySQL and choose the default schema, use `mysql.sql` . If you are using PostgreSQL and need the new schema, use `pg.new.sql` .

If you already have a MySQL or PostgreSQL database with the default schema and contents, you can upgrade it to the new schema:

- *MySQL*: Edit the file `sql/mysql.old-to.new.sql` which is included with ejabberd, fill DEFAULT_HOST in the first line, and import that SQL file in your database. Then enable the `new_sql_schema` option in the ejabberd configuration, and restart ejabberd.
- *PostgreSQL*: First enable `new_sql_schema` and mod_admin_update_sql in your ejabberd configuration:

```
new_sql_schema: true
modules:
  mod_admin_update_sql: {}
```

then restart ejabberd, and finally execute the update_sql command:

```
ejabberdctl update_sql
```

SQL Options

The actual database access is defined in the options with `sql_` prefix. The values are used to define if we want to use ODBC, or one of the two native interface available, PostgreSQL or MySQL.

To configure SQL there are several top-level options:

- sql_type
- sql_server
- sql_port
- sql_database
- sql_username
- sql_password
- sql_ssl
- sql_ssl_verify
- sql_ssl_cafile
- sql_ssl_certfile
- sql_pool_size
- sql_keepalive_interval
- sql_odbc_driver
- sql_start_interval
- sql_prepared_statements
- update_sql_schema

Example of plain ODBC connection:

```
sql_server: "DSN=database;UID=ejabberd;PWD=password"
```

Example of MySQL connection:

```
sql_type: mysql
sql_server: server.company.com
sql_port: 3306 # the default
sql_database: mydb
sql_username: user1
sql_password: "**********"
sql_pool_size: 5
```

SQL Authentication

You can authenticate users against an SQL database, see the option `auth_method` in the Authentication section.

To store the passwords in SCRAM format instead of plaintext, see the SCRAM section.

SQL with SSL Connection

It's possible to setup SSL encrypted connections to PostgreSQL, MySQL and MsSQL by enabling the sql_ssl option in ejabberd's configuration file: `sql_ssl: true`

Please notice that ejabberd verifies the certificate presented by the SQL server against the CA certificate list. For that reason, if your SQL server uses a self-signed certificate, you need to setup sql_ssl_verify and sql_ssl_cafile, for example:

```
sql_ssl: true
sql_ssl_verify: false
sql_ssl_cafile: "/path/to/sql_server_cacert.pem"
```

This tells ejabberd to ignore problems from not matching any CA certificate from default list, and instead try to verify using the specified CA certificate.

SQL Storage

Several `ejabberd modules` have options called `db_type`, and can store their tables in an SQL database instead of internal.

In this sense, if you defined your database access using the SQL Options, you can configure a module to use your database by adding the option `db_type: sql` to that module.

Alternatively, if you want all modules to use your SQL database when possible, you may prefer to set SQL as your default database.

Redis

`Redis` is an advanced key-value cache and store. You can use it to store transient data, such as records for C2S (client) sessions. There are several options available:

- `redis_server: String` : A hostname of the Redis server. The default is `localhost` .
- `redis_port: Port` : The port where the Redis server is accepting connections. The default is 6379.
- `redis_password: String` : The password to the Redis server. The default is an empty string, i.e. no password.
- `redis_db: N` : Redis database number. The default is 0.
- `redis_connect_timeout: N` : A number of seconds to wait for the connection to be established to the Redis server. The default is 1 second.

Example configuration:

```
redis_server: redis.server.com
redis_db: 1
```

Microsoft SQL Server

For now, MS SQL is only supported in Unix-like OS'es. You need to have `unixODBC` installed on your machine, and your Erlang/OTP must be compiled with ODBC support. Also, in some cases you need to add machine name to `sql_username` , especially when you have `sql_server` defined as an IP address, e.g.:

```
sql_type: mssql
sql_server: 1.2.3.4
sql_username: user1@host
```

By default, ejabberd will use the `FreeTDS` driver. You need to have the driver file `libtdsodbc.so` installed in your library PATH on your system.

If the FreeTDS driver is not installed in a standard location, or if you want to use another ODBC driver, you can specify the path to the driver using the sql_odbc_driver option, available in ejabberd 20.12 or later. For example, if you want to use Microsoft ODBC Driver 17 for SQL Server:

```
sql_odbc_driver: "/opt/microsoft/msodbcsql17/lib64/libmsodbcsql-17.3.so.1.1"
```

Note that if you use a Microsoft driver, you may have to use an IP address instead of a host name for the `sql_server` option.

If hostname (or IP address) is specified in `sql_server` option, ejabberd will connect using a an ODBC DSN connection string constructed with:

- SERVER=sql_server
- DATABASE=sql_database
- UID=sql_username
- PWD=sql_password
- PORT=sql_port
- ENCRYPTION=required (only if sql_ssl is true)
- CLIENT_CHARSET=UTF-8

Since ejabberd 23.04, t is possible to use different connection options by putting a full ODBC connection string in `sql_server` (e.g. `DSN=database;UID=ejabberd;PWD=password`). The DSN must be configured in existing system or user odbc.ini file, where it can be configured as desired, using a driver from system odbcinst.ini. The sql_odbc_driver option will have no effect in this case.

If specifying an ODBC connection string, an ODBC connection string must also be specified for any other hosts using MS SQL DB, otherwise the auto-generated ODBC configuration will interfere.

LDAP Configuration

Supported storages

The following LDAP servers are tested with `ejabberd`:

- `Active Directory` (see section [Active Directory](#))
- `OpenLDAP`
- `CommuniGate Pro`

- Normally any LDAP compatible server should work; inform us about your success with a not-listed server so that we can list it here.

LDAP

`ejabberd` has built-in LDAP support. You can authenticate users against LDAP server and use LDAP directory as vCard storage.

Usually `ejabberd` treats LDAP as a read-only storage: it is possible to consult data, but not possible to create accounts or edit vCard that is stored in LDAP. However, it is possible to change passwords if `mod_register` module is enabled and LDAP server supports `RFC 3062`.

LDAP Connection

Two connections are established to the LDAP server per vhost, one for authentication and other for regular calls.

To configure the LDAP connection there are these top-level options:

- [ldap_servers](#)
- [ldap_encrypt](#)
- [ldap_tls_verify](#)
- [ldap_tls_cacertfile](#)
- [ldap_tls_depth](#)
- [ldap_port](#)
- [ldap_rootdn](#)
- [ldap_password](#)
- [ldap_deref_aliases](#)

Example:

```
auth_method: [ldap]
ldap_servers:
  - ldap1.example.org
ldap_port: 389
ldap_rootdn: "cn=Manager,dc=domain,dc=org"
ldap_password: "**********"
```

LDAP Authentication

You can authenticate users against an LDAP directory. Note that current LDAP implementation does not support SASL authentication.

To configure LDAP authentication there are these top-level options:

- ldap_base
- ldap_uids
- ldap_filter
- ldap_dn_filter

LDAP 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. The connection to the LDAP server is encrypted using TLS, and using the custom port 6123. 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
ldap_encrypt: tls
ldap_port: 6123
## 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:
    db_type: 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 their 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` with LDAP backend checks for the existence of the user before searching their 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:
    db_type: 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
```

Shared Roster in LDAP

Since mod_shared_roster_ldap has a few complex options, some of them are documented with more detail here:

Filters

`ldap_ufilter` : "User Filter" – used for retrieving the human-readable name of roster entries (usually full names of people in the roster). See also the parameters `ldap_userdesc` and `ldap_useruid` . If unspecified, defaults to the top-level parameter of the same name. If that one also is unspecified, then the filter is assembled from values of other parameters as follows (`[ldap_SOMETHING]` is used to mean "the value of the configuration parameter `ldap_SOMETHING` "):

```
(&(&([ldap_memberattr]=[ldap_memberattr_format])([ldap_groupattr]=%g))[ldap_filter])
```

Subsequently `%u` and `%g` are replaced with a *. This means that given the defaults, the filter sent to the LDAP server would be `(&(memberUid=*)(cn=*))` . If however the `ldap_memberattr_format` is something like `uid=%u,ou=People,o=org` , then the filter will be `(&(memberUid=uid=*,ou=People,o=org)(cn=*))` .

`ldap_filter` : Additional filter which is AND-ed together with *User Filter* and *Group Filter*. If unspecified, defaults to the top-level parameter of the same name. If that one is also unspecified, then no additional filter is merged with the other filters.

Note that you will probably need to manually define the *User* and *Group Filter* (since the auto-assembled ones will not work) if:

- your `ldap_memberattr_format` is anything other than a simple `%u` ,
- **and** the attribute specified with `ldap_memberattr` does not support substring matches.

An example where it is the case is OpenLDAP and *(unique)MemberName* attribute from the *groupOf(Unique)Names* objectClass. A symptom of this problem is that you will see messages such as the following in your `slapd.log` :

```
get_filter: unknown filter type=130
filter="(&(?=undefined)(?=undefined)(something=else))"
```

Control parameters

These parameters control the behaviour of the module.

`ldap_memberattr_format_re` : A regex for extracting user ID from the value of the attribute named by `ldap_memberattr` .

An example value `"CN=(\\w*),(OU=.*,)*DC=company,DC=com"` works for user IDs such as the following:

- `CN=Romeo,OU=Montague,DC=company,DC=com`
- `CN=Abram,OU=Servants,OU=Montague,DC=company,DC=com`
- `CN=Juliet,OU=Capulet,DC=company,DC=com`
- `CN=Peter,OU=Servants,OU=Capulet,DC=company,DC=com`

In case:

- the option is unset,
- or the `re` module in unavailable in the current Erlang environment,
- or the regular expression does not compile,

then instead of a regular expression, a simple format specified by `ldap_memberattr_format` is used. Also, in the last two cases an error message is logged during the module initialization.

Also, note that in all cases `ldap_memberattr_format` (and `*not*` the regex version) is used for constructing the default "User/Group Filter" — see section Filters.

Retrieving the roster

When the module is called to retrieve the shared roster for a user, the following algorithm is used:

1. [step:rfilter] A list of names of groups to display is created: the *Roster Filter* is run against the base DN, retrieving the values of the attribute named by `ldap_groupattr` .

2. Unless the group cache is fresh (see the `ldap_group_cache_validity` option), it is refreshed:

a. Information for all groups is retrieved using a single query: the *Group Filter* is run against the Base DN, retrieving the values of attributes named by `ldap_groupattr` (group ID), `ldap_groupdesc` (group "Display Name") and `ldap_memberattr` (IDs of group members).

b. group "Display Name", read from the attribute named by `ldap_groupdesc` , is stored in the cache for the given group

c. the following processing takes place for each retrieved value of attribute named by `ldap_memberattr` :

i. the user ID part of it is extracted using `ldap_memberattr_format(_re)` ,

ii. then (unless `ldap_auth_check` is set to `off`) for each found user ID, the module checks (using the `ejabberd` authentication subsystem) whether such user exists in the given virtual host. It is skipped if the check is enabled and fails. This step is here for historical reasons. If you have a tidy DIT and properly defined "Roster Filter" and "Group Filter", it is safe to disable it by setting `ldap_auth_check` to `off` — it will speed up the roster retrieval.

iii. the user ID is stored in the list of members in the cache for the given group.

3. For each item (group name) in the list of groups retrieved in step [step:rfilter]:

a. the display name of a shared roster group is retrieved from the group cache

b. for each IDs of users which belong to the group, retrieved from the group cache:

i. the ID is skipped if it's the same as the one for which we are retrieving the roster. This is so that the user does not have himself in the roster.

ii. the display name of a shared roster user is retrieved:

i. first, unless the user name cache is fresh (see the `ldap_user_cache_validity` option), it is refreshed by running the *User Filter*, against the Base DN, retrieving the values of attributes named by `ldap_useruid` and `ldap_userdesc` .

ii. then, the display name for the given user ID is retrieved from the user name cache.

Multi-Domain

By default, the module option `ldap_userjidattr` is set to the empty string, in that case the JID of the user's contact is formed by compounding UID of the contact `@` Host of the user owning the roster.

When the option `ldap_userjidattr` is set to something like `"mail"` , then it uses that field to determine the JID of the contact. This is useful if the ldap `mail` attribute contains the JID of the accounts.

Basically, it allows us to define a groupOfNames (e.g. xmppRosterGroup) and list any users, anywhere in the ldap directory by specifying the attribute defining the JID of the members.

This allows hosts/domains other than that of the roster owner. It is also more flexible, since the LDAP manager can specify the JID of the users without any assumptions being made. The only down side is that there must be an LDAP attribute (field) filled in for all Jabber/XMPP users.

Below is a sample, a relevant LDAP entry, and ejabberd's module configuration:

```
cn=Example Org Roster,ou=groups,o=Example Organisation,dc=acme,dc=com
objectClass: groupOfNames
objectClass: xmppRosterGroup
objectClass: top
xmppRosterStatus: active
member:
description: Roster group for Example Org
cn: Example Org Roster
uniqueMember: uid=john,ou=people,o=Example Organisation,dc=acme,dc=com
```

```
uniqueMember: uid=pierre,ou=people,o=Example Organisation,dc=acme,dc=com
uniqueMember: uid=jane,ou=people,o=Example Organisation,dc=acme,dc=com

uid=john,ou=people,o=Example Organisation,dc=acme,dc=com
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: inetOrgPerson
objectClass: mailUser
objectClass: sipRoutingObject
uid: john
givenName: John
sn: Doe
cn: John Doe
displayName: John Doe
accountStatus: active
userPassword: secretpass
IMAPURL: imap://imap.example.net:143
mailHost: smtp.example.net
mail: john@example.net
sipLocalAddress: john@example.net
```

Below is the sample ejabberd.yml module configuration to match:

```
mod_shared_roster_ldap:
  ldap_servers:
    - "ldap.acme.com"
  ldap_encrypt: tls
  ldap_port: 636
  ldap_rootdn: "cn=Manager,dc=acme,dc=com"
  ldap_password: "supersecretpass"
  ldap_base: "dc=acme,dc=com"
  ldap_filter: "(objectClass=*)"
  ldap_rfilter: "(&(objectClass=xmppRosterGroup)(xmppRosterStatus=active))"
  ldap_gfilter: "(&(objectClass=xmppRosterGroup)(xmppRosterStatus=active)(cn=%g))"
  ldap_groupattr: "cn"
  ldap_groupdesc: "cn"
  ldap_memberattr: "uniqueMember"
  ldap_memberattr_format_re: "uid=([a-z.]*),(ou=.*,)*(o=.*,)*dc=acme,dc=com"
  ldap_useruid: "uid"
  ldap_userdesc: "cn"
  ldap_userjidattr: "mail"
  ldap_auth_check: false
  ldap_user_cache_validity: 86400
  ldap_group_cache_validity: 86400
```

Configuration examples

Since there are many possible `DIT` layouts, it will probably be easiest to understand how to configure the module by looking at an example for a given DIT (or one resembling it).

FLAT DIT

This seems to be the kind of DIT for which this module was initially designed. Basically there are just user objects, and group membership is stored in an attribute individually for each user. For example in a layout like this, it's stored in the `ou` attribute:

```
dn: ou=flat,dc=nodomain
objectClass: organizationalUnit
ou: flat
```

```
dn: ou=people,ou=flat,dc=nodomain
objectClass: organizationalUnit
ou: people
```

```
dn: cn=anusiak,ou=people,ou=flat,dc=nodomain
cn: anusiak
objectClass: inetOrgPerson
sn: anusiak
uid: anusiak
displayName: Anusiak Wiejski
ou: lewa-lawka
```

```
dn: cn=czesio,ou=people,ou=flat,dc=nodomain
cn: czesio
sn: czesio
uid: czesio
displayName: Czeslaw Zombie
ou: prawa-lawka
objectClass: inetOrgPerson
```

```
dn: cn=konieczko,ou=people,ou=flat,dc=nodomain
cn: konieczko
objectClass: inetOrgPerson
sn: konieczko
uid: konieczko
displayName: Konieczko Madrala
ou: lewa-lawka
```

```
dn: cn=maslana,ou=people,ou=flat,dc=nodomain
cn: maslana
sn: maslana
uid: maslana
displayName: Maslana Dziany
ou: prawa-lawka
objectClass: inetOrgPerson
```

Such layout has a few downsides, including:

- information duplication – the group name is repeated in every member object

- difficult group management – information about group members is not centralized, but distributed between member objects

- inefficiency – the list of unique group names has to be computed by iterating over all users

This however seems to be a common DIT layout, so the module keeps supporting it. You can use the following configuration…

```
modules:
  mod_shared_roster_ldap:
    ldap_base: "ou=flat,dc=nodomain"
    ldap_rfilter: "(objectClass=inetOrgPerson)"
    ldap_groupattr: ou
    ldap_memberattr: cn
    ldap_filter: "(objectClass=inetOrgPerson)"
    ldap_userdesc: displayName
```

…to be provided with a roster upon connecting as user `czesio`, as shown in this figure:
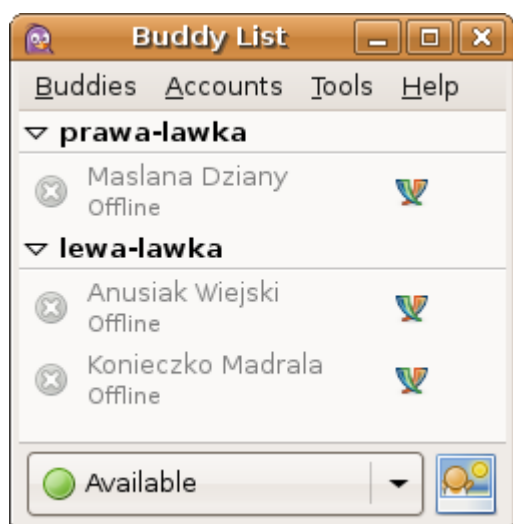
DEEP DIT

This type of DIT contains distinctly typed objects for users and groups – see the next figure. They are shown separated into different subtrees, but it's not a requirement.
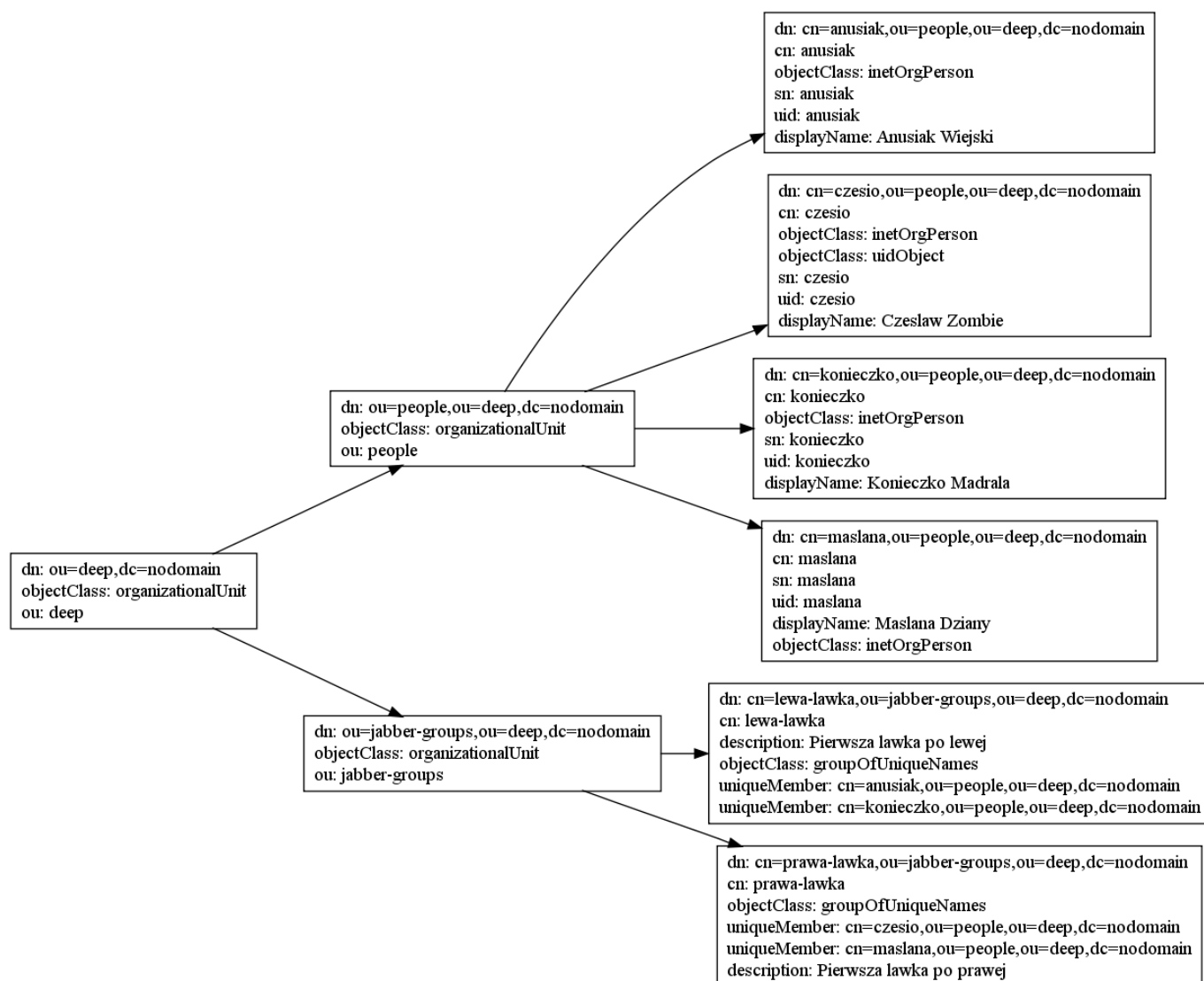
If you use the following example module configuration with it:

```
modules:
  mod_shared_roster_ldap:
    ldap_base: "ou=deep,dc=nodomain"
    ldap_rfilter: "(objectClass=groupOfUniqueNames)"
    ldap_filter: ""
    ldap_gfilter: "(&(objectClass=groupOfUniqueNames)(cn=%g))"
    ldap_groupdesc: description
    ldap_memberattr: uniqueMember
    ldap_memberattr_format: "cn=%u,ou=people,ou=deep,dc=nodomain"
    ldap_ufilter: "(&(objectClass=inetOrgPerson)(cn=%u))"
    ldap_userdesc: displayName
```

...and connect as user `czesio`, then `ejabberd` will provide you with the roster shown in this figure:

vCard in LDAP

Since LDAP may be complex to configure in mod_vcard, this section provides more details.

`ejabberd` can map LDAP attributes to vCard fields. This feature is enabled when the `mod_vcard` module is configured with `db_type:` `ldap`. Notice that it does not depend on the authentication method (see LDAP Authentication).

Usually `ejabberd` treats LDAP as a read-only storage: it is possible to consult data, but not possible to create accounts or edit vCard that is stored in LDAP. However, it is possible to change passwords if `mod_register` module is enabled and LDAP server supports `RFC 3062`.

This feature 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`, `ldap_deref_aliases` and `ldap_filter`. See section LDAP Authentication 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.

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:
    db_type: 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"}
    ## Now we want to define vCard pattern
    ldap_vcard_map:
      NICKNAME: {"%u": []} # just use user's part of JID as their 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` with LDAP backend checks an existence of the user before searching their info in LDAP.

- `ldap_vcard_map` example:

```
ldap_vcard_map:
  NICKNAME: {"%u": []} # just use user's part of JID as their nickname
  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
```

Listen Modules

This section describes the most recent ejabberd version. If you are using an old ejabberd release, please refer to the corresponding archived version of this page in the Archive.

Listen Options

The `listen` option defines for which ports, addresses and network protocols `ejabberd` will listen and what services will be run on them.

Each element of the list is an associative array with the following elements:

- **port**: *Number*

  Defines which port number to listen for incoming connections: it can be a Jabber/XMPP standard port or any other valid port number.

  Alternatively, set the option to a string in form `"unix:/path/to/socket"` to create and listen on a unix domain socket `/path/to/socket`.

- **ip**: *IpAddress*

  The socket will listen only in that network interface. Depending on the type of the IP address, IPv4 or IPv6 will be used.

  It is possible to specify a generic address (`"0.0.0.0"` for IPv4 or `"::"` for IPv6), so `ejabberd` will listen in all addresses. Note that on some operating systems and/or OS configurations, listening on `"::"` will mean listening for IPv4 traffic as well as IPv6 traffic.

  Some example values for IP address:

- `"0.0.0.0"` to listen in all IPv4 network interfaces. This is the default value when the option is not specified.

- `"::"` to listen in all IPv6 network interfaces

- `"10.11.12.13"` is the IPv4 address `10.11.12.13`

- `"::FFFF:127.0.0.1"` is the IPv6 address `::FFFF:127.0.0.1/128`

- **transport**: *tcp|udp*

  Defines the transport protocol. Default is `tcp`.

- **module**: *ModuleName*

  Listening module that serves this port

- Any other options for the socket and for the listening module, described later.

  For example:

  ```
  listen:
    -
      port: 5222
      ip: 127.0.0.1
      module: ejabberd_c2s
      starttls: true
    -
      port: 5269
      transport: tcp
      module: ejabberd_s2s_in
  ```

ejabberd_c2s

Handles c2s connections.

General listen options supported: access, cafile, ciphers, dhfile, max_fsm_queue, max_stanza_size, protocol_options, send_timeout, shaper, starttls, starttls_required, tls, tls_compression, tls_verify, zlib.

ejabberd_s2s_in

Handles incoming s2s connections.

General listen options supported: cafile, ciphers, dhfile, max_fsm_queue, max_stanza_size, protocol_options, send_timeout, shaper, tls, tls_compression.

ejabberd_service

Interacts with an `external component` as defined in XEP-0114: Jabber Component Protocol.

General listen options supported: access, cafile, certfile, check_from, ciphers, dhfile, global_routes, hosts, max_fsm_queue, max_stanza_size, password, protocol_options, send_timeout, shaper, shaper_rule, tls, tls_compression.

mod_mqtt

Support for MQTT requires configuring `mod_mqtt` both in the listen and the modules sections. Check the mod_mqtt module options, and the MQTT Support section.

General listen options supported: backlog, max_fsm_queue, max_payload_size, send_timeout, tls, tls_verify.

ejabberd_stun

`ejabberd` can act as a stand-alone STUN/TURN server, and this module handles STUN/TURN requests as defined in (`RFC 5389` / `RFC 5766` . In that role `ejabberd` helps clients with ICE (`RFC 5245` or Jingle ICE (`XEP-0176` support to discover their external addresses and ports and to relay media traffic when it is impossible to establish direct peer-to-peer connection.

General listen options supported: certfile, send_timeout, shaper, tls,

The specific `ejabberd_stun` configurable options are:

- **auth_realm**: *String*

  When `auth_type` is set to `user` and you have several virtual hosts configured you should set this option explicitly to the virtual host you want to serve on this particular listening port. Implies `use_turn`.

- **auth_type**: *user|anonymous*

  Which authentication type to use for TURN allocation requests. When type `user` is set, ejabberd authentication backend is used. For `anonymous` type no authentication is performed (not recommended for public services). The default is `user`. Implies `use_turn`.

- **shaper**: *Atom*

  For `tcp` transports defines shaper to use. The default is `none`.

- **server_name**: *String*

  Defines software version to return with every response. The default is the STUN library version.

- **turn_blacklist**: *String | [String,...]*

  Specify one or more IP addresses and/or subnet addresses/masks. The TURN server will refuse to relay traffic from/to blacklisted IP addresses. By default, loopback addresses (`127.0.0.0/8` and `::1/128`) are blacklisted.

- **turn_ipv4_address**: *String*

  The IPv4 address advertised by your TURN server. The address should not be NAT'ed or firewalled. There is not default, so you should set this option explicitly. Implies `use_turn`.

- **turn_ipv6_address**: *String*

  The IPv6 address advertised by your TURN server. The address should not be NAT'ed or firewalled. There is not default, so you should set this option explicitly. Implies `use_turn`.

- **turn_max_allocations**: *Integer|infinity*

  Maximum number of TURN allocations available from the particular IP address. The default value is 10. Implies `use_turn`.

- **turn_max_permissions**: *Integer|infinity*

  Maximum number of TURN permissions available from the particular IP address. The default value is 10. Implies `use_turn`.

- **turn_max_port**: *Integer*

  Together with `turn_min_port` forms port range to allocate from. The default is 65535. Implies `use_turn`.

- **turn_min_port**: *Integer*

  Together with `turn_max_port` forms port range to allocate from. The default is 49152. Implies `use_turn`.

- **use_turn**: *true|false*

  Enables/disables TURN (media relay) functionality. The default is `false`.

Example configuration with disabled TURN functionality (STUN only):

```
listen:
  -
    port: 3478
    transport: udp
    module: ejabberd_stun
  -
    port: 3478
    module: ejabberd_stun
  -
    port: 5349
    module: ejabberd_stun
    certfile: /etc/ejabberd/server.pem
```

Example configuration with TURN functionality. Note that STUN is always enabled if TURN is enabled. Here, only UDP section is shown:

```
listen:
  -
    port: 3478
    transport: udp
    use_turn: true
```

```
        turn_ipv4_address: 10.20.30.1
        module: ejabberd_stun
```

You also need to configure DNS SRV records properly so clients can easily discover a STUN/TURN server serving your XMPP domain. Refer to section `DNS Discovery of a Server` of `RFC 5389` and section `Creating an Allocation` of `RFC 5766` for details.

Example DNS SRV configuration for STUN only:

```
_stun._udp   IN SRV  0 0 3478 stun.example.com.
_stun._tcp   IN SRV  0 0 3478 stun.example.com.
_stuns._tcp  IN SRV  0 0 5349 stun.example.com.
```

And you should also add these in the case if TURN is enabled:

```
_turn._udp   IN SRV  0 0 3478 turn.example.com.
_turn._tcp   IN SRV  0 0 3478 turn.example.com.
_turns._tcp  IN SRV  0 0 5349 turn.example.com.
```

ejabberd_sip

`ejabberd` has built-in support to handle SIP requests as defined in `RFC 3261` .

To activate this feature, add the `ejabberd_sip` listen module, enable `mod_sip` module for the desired virtual host, and configure DNS properly.

To add a listener you should configure `ejabberd_sip` listening module as described in Listen section. If option `tls` is specified, option `certfile` must be specified as well, otherwise incoming TLS connections would fail.

General listen options supported: certfile, send_timeout, tls.

Example configuration with standard ports (as per `RFC 3261`):

```
listen:
  -
    port: 5060
    transport: udp
    module: ejabberd_sip
  -
    port: 5060
    module: ejabberd_sip
  -
    port: 5061
    module: ejabberd_sip
    tls: true
    certfile: /etc/ejabberd/server.pem
```

Note that there is no StartTLS support in SIP and `SNI` support is somewhat tricky, so for TLS you have to configure different virtual hosts on different ports if you have different certificate files for them.

Next you need to configure DNS SIP records for your virtual domains. Refer to `RFC 3263` for the detailed explanation. Simply put, you should add NAPTR and SRV records for your domains. Skip NAPTR configuration if your DNS provider doesn't support this type of records. It's not fatal, however, highly recommended.

Example configuration of NAPTR records:

```
example.com IN NAPTR 10  0 "s" "SIPS+D2T" "" _sips._tcp.example.com.
example.com IN NAPTR 20  0 "s" "SIP+D2T" "" _sip._tcp.example.com.
example.com IN NAPTR 30  0 "s" "SIP+D2U" "" _sip._udp.example.com.
```

Example configuration of SRV records with standard ports (as per `RFC 3261` :

```
_sip._udp   IN SRV  0 0 5060 sip.example.com.
_sip._tcp   IN SRV  0 0 5060 sip.example.com.
_sips._tcp  IN SRV  0 0 5061 sip.example.com.
```

> ⚠️ **Warning**
>
> SIP authentication does not support SCRAM. As such, it is not possible to use `mod_sip` to authenticate when ejabberd has been set to encrypt password with SCRAM.

ejabberd_http

Handles incoming HTTP connections.

With the proper request handlers configured, this serves HTTP services like ACME, API, BOSH, CAPTCHA, Fileserver, OAuth, RegisterWeb, Upload, WebAdmin, WebSocket, XML-RPC.

Options: cafile, ciphers, custom_headers, default_host, dhfile, protocol_options, request_handlers, send_timeout, tag, tls, tls_compression, and the trusted_proxies top-level option.

ejabberd_http_ws

This module enables XMPP communication over WebSocket connection as described in `RFC 7395`.

WEBSOCKET CONFIG

To enable WebSocket, simply add a handler to the `request_handlers` section of an `ejabberd_http` listener:

```
listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /xmpp: ejabberd_http_ws
```

This module can be configured using those top-level options:

- websocket_origin
- websocket_ping_interval
- websocket_timeout

WEBSOCKET DISCOVERY

With the example configuration previously mentioned, the WebSocket URL would be: `ws://localhost:5280/xmpp`

You may want to provide a `host-meta` file so clients can easily discover WebSocket service for your XMPP domain (see XEP-0156). One easy way to provide that file is using `mod_host_meta`.

TESTING WEBSOCKET

A test client can be found on Github: WebSocket test client

There is an example configuration for WebSocket and Converse.js in the ejabberd 21.12 release notes.

ejabberd_xmlrpc

Handles XML-RPC requests to execute ejabberd commands. It is configured as a request handler in ejabberd_http.

This is the minimum configuration required to enable the feature:

```
listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /xmlrpc: ejabberd_xmlrpc

api_permissions:
  "public commands":
    who:
```

```
      ip: 127.0.0.1/8
    what:
      - connected_users_number
```

Example Python3 script:

```
import xmlrpc.client
server = xmlrpc.client.ServerProxy("http://127.0.0.1:5280/xmlrpc/");
print(server.connected_users_number())
```

By default there is no restriction to who can execute what commands, so it is strongly recommended that you configure restrictions using API Permissions.

This example configuration adds some restrictions (only requests from localhost are accepted, the XML-RPC query must include authentication credentials of a specific account registered in ejabberd, and only two commands are accepted):

```
listen:
  -
    port: 5280
    ip: "::"
    module: ejabberd_http
    request_handlers:
      /xmlrpc: ejabberd_xmlrpc

api_permissions:
  "some XMLRPC commands":
    from: ejabberd_xmlrpc
    who:
      - ip: 127.0.0.1
      - user: user1@localhost
    what:
      - registered_users
      - connected_users_number
```

Example Python3 script for that restricted configuration:

```
import xmlrpc.client
server = xmlrpc.client.ServerProxy("http://127.0.0.1:5280/xmlrpc/");

params = {}
params['host'] = 'localhost'

auth = {'user': 'user1',
        'server': 'localhost',
        'password': 'mypass11',
        'admin': True}

def calling(command, data):
    fn = getattr(server, command)
    return fn(auth, data)

print(calling('registered_users', params))
```

Please notice, when using the old Python2, replace the two first lines with:

```
import xmlrpclib
server = xmlrpclib.Server("http://127.0.0.1:5280/xmlrpc/");
```

It's possible to use OAuth for authentication instead of plain password, see OAuth Support.

In ejabberd 20.03 and older, it was possible to configure `ejabberd_xmlrpc` as a listener, see the old document for reference and example configuration: Listening Module.

Just for reference, there's also the old `ejabberd_xmlrpc documentation` with example clients in other languages.

Examples

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.
- Port 5223 listens for c2s connections with the old SSL.
- Port 5269 listens for s2s connections with STARTTLS. The socket is set for IPv6 instead of IPv4.
- Port 3478 listens for STUN requests over UDP.
- Port 5280 listens for HTTP requests, and serves the HTTP-Bind (BOSH) service.
- Port 5281 listens for HTTP requests, using HTTPS to serve HTTP-Bind (BOSH) and the Web Admin as explained in Managing: Web Admin. The socket only listens connections to the IP address 127.0.0.1.

```
hosts:
  - example.com
  - example.org
  - example.net

certfiles:
  - /etc/ejabberd/server.pem
  - /etc/ejabberd/example_com.pem

listen:
  -
    port: 5222
    module: ejabberd_c2s
    access: c2s
    shaper: c2s_shaper
    starttls: true
    max_stanza_size: 65536
  -
    port: 5223
    module: ejabberd_c2s
    access: c2s
    shaper: c2s_shaper
    tls: true
    max_stanza_size: 65536
  -
    port: 5269
    ip: "::"
    module: ejabberd_s2s_in
    shaper: s2s_shaper
    max_stanza_size: 131072
  -
    port: 3478
    transport: udp
    module: ejabberd_stun
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /bosh: mod_bosh
  -
    port: 5281
    ip: 127.0.0.1
    module: ejabberd_http
    tls: true
    request_handlers:
      /admin: ejabberd_web_admin
      /bosh: mod_bosh

s2s_use_starttls: optional
outgoing_s2s_families:
  - ipv4
  - ipv6
outgoing_s2s_timeout: 10000
trusted_proxies: [127.0.0.1, 192.168.1.11]
```

In this example, the following configuration defines that:

- c2s connections are listened for on port 5222 (all IPv4 addresses) and on port 5223 (SSL, IP 192.168.0.1 and fdca: 8ab6:a243:75ef::1) and denied for the user called ' `bad` '.

- s2s connections are listened for on port 5269 (all IPv4 addresses) with STARTTLS for secured traffic strictly required, and the certificates are verified. Incoming and outgoing connections of remote XMPP servers are denied, only two servers can connect: "jabber.example.org" and "example.com".

- Port 5280 is serving the Web Admin and the HTTP-Bind (BOSH) service in all the IPv4 addresses. Note that it is also possible to serve them on different ports. The second example in section Managing: Web Admin shows how exactly this can be done. A request handler to serve MQTT over WebSocket is also defined.

- All users except for the administrators have a traffic of limit 1,000Bytes/second

- The `AIM transport` `aim.example.org` is connected to port 5233 on localhost IP addresses (127.0.0.1 and ::1) 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` `msn.example.org` is connected to port 5235 with password ' `msnsecret` '.

- The `Yahoo! transport` `yahoo.example.org` is connected to port 5236 with password ' `yahoosecret` '.

- The `Gadu-Gadu transport` `gg.example.org` is connected to port 5237 with password ' `ggsecret` '.

- The `Jabber Mail Component` `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
  trusted_servers:
    server:
      - example.com
      - jabber.example.org
  xmlrpc_bot:
    user:
      - xmlrpc-robot@example.org
shaper:
  normal: 1000
shaper_rules:
  c2s_shaper:
    - none: admin
    - normal
access_rules:
  c2s:
    - deny: blocked
    - allow
  xmlrpc_access:
    - allow: xmlrpc_bot
  s2s:
    - allow: trusted_servers
certfiles:
  - /path/to/ssl.pem
s2s_access: s2s
s2s_use_starttls: required_trusted
listen:
  -
    port: 5222
    module: ejabberd_c2s
    shaper: c2s_shaper
    access: c2s
  -
    ip: 192.168.0.1
    port: 5223
    module: ejabberd_c2s
    tls: true
    access: c2s
  -
    ip: "FDCA:8AB6:A243:75EF::1"
    port: 5223
    module: ejabberd_c2s
    tls: true
    access: c2s
  -
    port: 5269
    module: ejabberd_s2s_in
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
```

```
      /admin: ejabberd_web_admin
      /bosh: mod_bosh
      /mqtt: mod_mqtt
  -
    port: 4560
    module: ejabberd_xmlrpc
    access_commands: {}
  -
    ip: 127.0.0.1
    port: 5233
    module: ejabberd_service
    hosts:
      aim.example.org:
        password: aimsecret
  -
    ip: "::1"
    port: 5233
    module: ejabberd_service
    hosts:
      aim.example.org:
        password: aimsecret
  -
    port: 5234
    module: ejabberd_service
    hosts:
      icq.example.org:
        password: jitsecret
      sms.example.org:
        password: jitsecret
  -
    port: 5235
    module: ejabberd_service
    hosts:
      msn.example.org:
        password: msnsecret
  -
    port: 5236
    module: ejabberd_service
    password: yahoosecret
  -
    port: 5237
    module: ejabberd_service
    hosts:
      gg.example.org:
        password: ggsecret
  -
    port: 5238
    module: ejabberd_service
    hosts:
      jmc.example.org:
        password: jmcsecret
  -
    port: 5239
    module: ejabberd_service
    check_from: false
    hosts:
      custom.example.org:
        password: customsecret
```

Note, that for services based in jabberd14 or WPJabber 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 XMPP 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 jabberd14 -->
    <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>
```

Listen Options

This section describes the most recent ejabberd version. If you are using an old ejabberd release, please refer to the corresponding archived version of this page in the Archive.

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

access

AccessName

This option defines access to the port. The default value is `all` .

backlog

Value

The backlog value defines the maximum length that the queue of pending connections may grow to. This should be increased if the server is going to handle lots of new incoming connections as they may be dropped if there is no space in the queue (and ejabberd was not able to accept them immediately). Default value is 5.

cafile

Path

Path to a file of CA root certificates. The default is to use system defined file if possible.

This option is useful to define the file for a specific port listener. To set a file for all client listeners or for specific vhosts, you can use the `c2s_cafile` top-level option. To set a file for all server connections, you can use the `s2s_cafile` top-level option or the `ca_file` top-level option.

Please note: if this option is set in `ejabberd_c2s` or `ejabberd_s2s_in` and the corresponding top-level option is also set (`c2s_cafile` , `s2s_cafile`), then the top-level option is used, not this one.

certfile

Path

Path to the certificate file. Only makes sense when the `tls` options is set. If this option is not set, you should set the `certfiles` top-level option or configure ACME.

check_from

true | false

This option can be used with `ejabberd_service` only. `XEP-0114` requires that the domain must match the hostname of the component. If this option is set to `false` , `ejabberd` will allow the component to send stanzas with any arbitrary domain in the 'from' attribute. Only use this option if you are completely sure about it. The default value is `true` , to be compliant with `XEP-0114` .

ciphers

Ciphers

OpenSSL ciphers list in the same format accepted by ' `openssl ciphers` ' command.

Please note: if this option is set in `ejabberd_c2s` or `ejabberd_s2s_in` and the corresponding top-level option is also set (`c2s_ciphers` , `s2s_ciphers`), then the top-level option is used, not this one.

custom_headers

{Name: Value}

Specify additional HTTP headers to be included in all HTTP responses. Default value is: `[]`

default_host

undefined | HostName

If the HTTP request received by ejabberd contains the HTTP header `Host` with an ambiguous virtual host that doesn't match any one defined in ejabberd (see Host Names), then this configured HostName is set as the request Host. The default value of this option is: `undefined` .

dhfile

Path

Full path to a file containing custom parameters for Diffie-Hellman key exchange. Such a file could be created with the command `openssl dhparam -out dh.pem 2048` . If this option is not specified, default parameters will be used, which might not provide the same level of security as using custom parameters.

Please note: if this option is set in `ejabberd_c2s` or `ejabberd_s2s_in` and the corresponding top-level option is also set (`c2s_dhfile` , `s2s_dhfile`), then the top-level option is used, not this one.

global_routes

true | false

This option emulates legacy behaviour which registers all routes defined in `hosts` on a component connected. This behaviour is considered harmful in the case when it's desired to multiplex different components on the same port, so, to disable it, set `global_routes` to `false` .

The default value is `true` , e.g. legacy behaviour is emulated: the only reason for this is to maintain backward compatibility with existing deployments.

hosts

{Hostname: [HostOption, ...]}

The external Jabber component that connects to this `ejabberd_service` can serve one or more hostnames. As `HostOption` you can define options for the component; currently the only allowed option is the password required to the component 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. This option may not be necessary if the component already provides the host in its packets; in that case, you can simply provide the password option that will be used for all the hosts (see port 5236 definition in the example below).

max_fsm_queue

Size

This option specifies the maximum number of elements in the queue of the FSM (Finite State Machine). Roughly speaking, each message in such queues represents one XML stanza queued to be sent into its relevant outgoing stream. If queue size reaches the limit (because, for example, the receiver of stanzas is too slow), the FSM and the corresponding connection (if any) will be terminated and error message will be logged. The reasonable value for this option depends on your hardware configuration. This option can be specified for `ejabberd_service` and `ejabberd_c2s` listeners, or also globally for `ejabberd_s2s_out` . If the option is not specified for `ejabberd_service` or `ejabberd_c2s` listeners, the globally configured value is used. The allowed values are integers and 'undefined'. Default value: '10000'.

max_payload_size

Size

Specify the maximum payload size in bytes. It can be either an integer or the word `infinity` . The default value is `infinity` .

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 read 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.

password

Secret

Specify the password to verify an external component that connects to the port.

port

Port number, or unix domain socket path

🏷 improved in 20.07

Declares at which port/unix domain socket should be listening.

Can be set to number between `1` and `65535` to listen on TCP or UDP socket, or can be set to string in form `"unix:/path/to/socket"` to create and listen on unix domain socket `/path/to/socket` .

protocol_options

ProtocolOpts

List of general options relating to SSL/TLS. These map to `OpenSSL's set_options()` . The default entry is: `"no_sslv3|cipher_server_preference|no_compression"`

Please note: if this option is set in `ejabberd_c2s` or `ejabberd_s2s_in` and the corresponding top-level option is also set (`c2s_protocol_options` , `s2s_protocol_options`), then the top-level option is used, not this one.

request_handlers

{Path: Module}

To define one or several handlers that will serve HTTP requests in `ejabberd_http` . The Path is a string; 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_bosh` to serve the URIs `/bosh/` , use this option:

```
request_handlers:
  /a/b: mod_foo
  /bosh: mod_bosh
  /mqtt: mod_mqtt
```

send_timeout

Integer | infinity

🏷 new in 21.07

Sets the longest time that data can wait to be accepted to sent by OS socket. Triggering this timeout will cause the server to close it. By default it's set to 15 seconds, expressed in milliseconds: 15000

shaper

none | ShaperName

This option defines a shaper for the port (see section Shapers). The default value is `none` .

shaper_rule

none | ShaperRule

This option defines a shaper rule for `ejabberd_service` (see section Shapers). The recommended value is `fast` .

starttls

true | false

This option specifies that STARTTLS encryption is available on connections to the port. You should also set the `certfiles` top-level option or configure ACME.

This option gets implicitly enabled when enabling `starttls_required` or `tls_verify` .

starttls_required

true | false

This option specifies that STARTTLS encryption is required on connections to the port. No unencrypted connections will be allowed. You should also set the `certfiles` top-level option or configure ACME.

Enabling this option implicitly enables also the `starttls` option.

tag

String

Allow specifying a tag in a `listen` section and later use it to have a special `api_permissions` just for it.

For example:

```
listen:
  -
    port: 4000
```

```
    module: ejabberd_http
    tag: "magic_listener"

api_permissions:
  "magic_access":
    from:
      - tag: "magic_listener"
    who: all
    what: "*"
```

The default value is the empty string: `""` .

timeout

Integer

Timeout of the connections, expressed in milliseconds. Default: 5000

tls

true | false

This option specifies that traffic on the port will be encrypted using SSL immediately after connecting. This was the traditional encryption method in the early Jabber software, commonly on port 5223 for client-to-server communications. But this method is nowadays deprecated and not recommended. The preferable encryption method is STARTTLS on port 5222, as defined `RFC 6120: XMPP Core` , which can be enabled in `ejabberd` with the option `starttls` .

If this option is set, you should also set the `certfiles` top-level option or configure ACME.

The option `tls` can also be used in `ejabberd_http` to support HTTPS.

Enabling this option implicitly disables the `starttls` option.

tls_compression

true | false

Whether to enable or disable TLS compression. The default value is `false` .

Please note: if this option is set in `ejabberd_c2s` or `ejabberd_s2s_in` and the corresponding top-level option is also set (`c2s_tls_compression` , `s2s_tls_compression`), then the top-level option is used, not this one.

tls_verify

false | true

This option specifies whether to verify the certificate or not when TLS is enabled.

The default value is `false` , which means no checks are performed.

The certificate will be checked against trusted CA roots, either defined at the operation system level or defined in the listener `cafile` . If trusted, it will accept the jid that is embedded in the certificate in the `subjectAltName` field of that certificate.

Enabling this option implicitly enables also the `starttls` option.

use_proxy_protocol

true | false

Is this listener accessed by proxy service that is using proxy protocol for supplying real IP addresses to ejabberd server. You can read about this protocol in Proxy protocol specification. The default value of this option is `false` .

zlib

true | false

This option specifies that Zlib stream compression (as defined in `XEP-0138`) is available on connections to the port.

Top-Level Options

This section describes top level options of ejabberd 24.02. If you are using an old ejabberd release, please refer to the corresponding archived version of this page in the Archive. The options that changed in this version are marked with ⬤.

access_rules

{AccessName: {allow|deny: ACLRules|ACLName}}

This option defines Access Rules. Each access rule is assigned a name that can be referenced from other parts of the configuration file (mostly from *access* options of ejabberd modules). Each rule definition may contain arbitrary number of *allow* or *deny* sections, and each section may contain any number of ACL rules (see acl option). There are no access rules defined by default.

Example:

```
access_rules:
  configure:
    allow: admin
  something:
    deny: someone
    allow: all
  s2s_banned:
    deny: problematic_hosts
    deny: banned_forever
    deny:
      ip: 222.111.222.111/32
    deny:
      ip: 111.222.111.222/32
    allow: all
  xmlrpc_access:
    allow:
      user: peter@example.com
    allow:
      user: ivone@example.com
    allow:
      user: bot@example.com
      ip: 10.0.0.0/24
```

acl

{ACLName: {ACLType: ACLValue}}

The option defines access control lists: named sets of rules which are used to match against different targets (such as a JID or an IP address). Every set of rules has name *ACLName*: it can be any string except *all* or *none* (those are predefined names for the rules that match all or nothing respectively). The name *ACLName* can be referenced from other parts of the configuration file, for

example in [access_rules](#) option. The rules of *ACLName* are represented by mapping *{ACLType: ACLValue}*. These can be one of the following:

- **ip**: *Network*
  The rule matches any IP address from the *Network*.

- **node_glob**: *Pattern*
  Same as *node_regexp*, but matching is performed on a specified *Pattern* according to the rules used by the Unix shell.

- **node_regexp**: *user_regexp@server_regexp*
  The rule matches any JID with node part matching regular expression *user_regexp* and server part matching regular expression *server_regexp*.

- **resource**: *Resource*
  The rule matches any JID with a resource *Resource*.

- **resource_glob**: *Pattern*
  Same as *resource_regexp*, but matching is performed on a specified *Pattern* according to the rules used by the Unix shell.

- **resource_regexp**: *Regexp*
  The rule matches any JID with a resource that matches regular expression *Regexp*.

- **server**: *Server*
  The rule matches any JID from server *Server*. The value of *Server* must be a valid hostname or an IP address.

- **server_glob**: *Pattern*
  Same as *server_regexp*, but matching is performed on a specified *Pattern* according to the rules used by the Unix shell.

- **server_regexp**: *Regexp*
  The rule matches any JID from the server that matches regular expression *Regexp*.

- **user**: *Username*
  If *Username* is in the form of "user@server", the rule matches a JID against this value. Otherwise, if *Username* is in the form of "user", the rule matches any JID that has *Username* in the node part as long as the server part of this JID is any virtual host served by ejabberd.

- **user_glob**: *Pattern*
  Same as *user_regexp*, but matching is performed on a specified *Pattern* according to the rules used by the Unix shell.

- **user_regexp**: *Regexp*
  If *Regexp* is in the form of "regexp@server", the rule matches any JID with node part matching regular expression "regexp" as long as the server part of this JID is equal to "server". If *Regexp* is in the form of "regexp", the rule matches any JID with node part matching regular expression "regexp" as long as the server part of this JID is any virtual host served by ejabberd.

acme

Options

ACME configuration, to automatically obtain SSL certificates for the domains served by ejabberd, which means that certificate requests and renewals are performed to some CA server (aka "ACME server") in a fully automated mode. The *Options* are:

- **auto**: *true | false*
  Whether to automatically request certificates for all configured domains (that yet have no a certificate) on server start or configuration reload. The default is *true*.

- **ca_url**: *URL*
  The ACME directory URL used as an entry point for the ACME server. The default value is https://acme-v02.api.letsencrypt.org/directory - the directory URL of Let's Encrypt authority.

- **cert_type**: *rsa | ec*
  A type of a certificate key. Available values are *ec* and *rsa* for EC and RSA certificates respectively. It's better to have RSA certificates for the purpose of backward compatibility with legacy clients and servers, thus the default is *rsa*.

- **contact**: *[Contact, ...]*
  A list of contact addresses (typically emails) where an ACME server will send notifications when problems occur. The value of *Contact* must be in the form of "scheme:address" (e.g. "mailto:user@domain.tld"). The default is an empty list which means an ACME server will send no notices.

Example:

```
acme:
  ca_url: https://acme-v02.api.letsencrypt.org/directory
  contact:
    - mailto:admin@domain.tld
    - mailto:bot@domain.tld
  auto: true
  cert_type: rsa
```

allow_contrib_modules

true | false

Whether to allow installation of third-party modules or not. See ejabberd-contrib documentation section. The default value is *true*.

allow_multiple_connections

true | false

This option 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

login_anon | sasl_anon | both

Define what anonymous protocol will be used:

- *login_anon* means that the anonymous login method will be used.

- *sasl_anon* means that the SASL Anonymous method will be used.

- *both* means that SASL Anonymous and login anonymous are both enabled.

The default value is *sasl_anon*.

api_permissions

[Permission, ...]

Define the permissions for API access. Please consult the ejabberd Docs web → For Developers → ejabberd ReST API → API Permissions.

append_host_config

{Host: Options}

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, *append_host_config* option can be used.

auth_cache_life_time

timeout()

Same as cache_life_time, but applied to authentication cache only. If not set, the value from cache_life_time will be used.

auth_cache_missed

true | false

Same as cache_missed, but applied to authentication cache only. If not set, the value from cache_missed will be used.

auth_cache_size

pos_integer() | infinity

Same as cache_size, but applied to authentication cache only. If not set, the value from cache_size will be used.

auth_external_user_exists_check

true | false

🏷 added in 23.10

Supplement check for user existence based on *mod_last* data, for authentication methods that don't have a way to reliably tell if a user exists (like is the case for *jwt* and certificate based authentication). This helps with processing offline message for those users. The default value is *true*.

auth_method

[mnesia | sql | anonymous | external | jwt | ldap | pam, ...]

A list of authentication methods to use. If several methods are defined, authentication is considered successful as long as authentication of at least one of the methods succeeds. The default value is *[mnesia]*.

auth_opts

[Option, ...]

This is used by the contributed module *ejabberd_auth_http* that can be installed from the ejabberd-contrib Git repository. Please refer to that module's README file for details.

auth_password_format

plain | scram

🏷 improved in 20.01

The option defines in what format the users passwords are stored, plain text or in SCRAM format:

- *plain*: The password is stored as plain text in the database. This is risky because the passwords can be read if your database gets compromised. This is the default value. This format allows clients to authenticate using: the old Jabber Non-SASL (XEP-0078), SASL PLAIN, SASL DIGEST-MD5, and SASL SCRAM-SHA-1/256/512(-PLUS).
- *scram*: The password is not stored, only some information required to verify the hash provided by the client. It is impossible to obtain the original plain password from the stored information; for this reason, when this value is configured it cannot be changed to plain anymore. This format allows clients to authenticate using: SASL PLAIN and SASL SCRAM-SHA-1/256/512(-PLUS). The SCRAM variant depends on the auth_scram_hash option.

The default value is *plain*.

auth_scram_hash

sha | sha256 | sha512

Hash algorithm that should be used to store password in SCRAM format. You shouldn't change this if you already have passwords generated with a different algorithm - users that have such passwords will not be able to authenticate. The default value is *sha*.

auth_use_cache

true | false

Same as use_cache, but applied to authentication cache only. If not set, the value from use_cache will be used.

c2s_cafile

Path

Full path to a file containing one or more CA certificates in PEM format. All client certificates should be signed by one of these root CA certificates and should contain the corresponding JID(s) in *subjectAltName* field. There is no default value.

You can use host_config to specify this option per-vhost.

To set a specific file per listener, use the listener's cafile option. Please notice that *c2s_cafile* overrides the listener's *cafile* option.

c2s_ciphers

[Cipher, ...]

A list of OpenSSL ciphers to use for c2s connections. The default value is shown in the example below:

Example:

```
c2s_ciphers:
  - HIGH
```

```
    - "!aNULL"
    - "!eNULL"
    - "!3DES"
    - "@STRENGTH"
```

c2s_dhfile

Path

Full path to a file containing custom DH parameters to use for c2s connections. Such a file could be created with the command "openssl dhparam -out dh.pem 2048". If this option is not specified, 2048-bit MODP Group with 256-bit Prime Order Subgroup will be used as defined in RFC5114 Section 2.3.

c2s_protocol_options

[Option, ...]

List of general SSL options to use for c2s connections. These map to OpenSSL's *set_options()*. The default value is shown in the example below:

Example:

```
c2s_protocol_options:
  - no_sslv3
  - cipher_server_preference
  - no_compression
```

c2s_tls_compression

true | false

Whether to enable or disable TLS compression for c2s connections. The default value is *false*.

ca_file

Path

Path to a file of CA root certificates. The default is to use system defined file if possible.

For server connections, this *ca_file* option is overridden by the s2s_cafile option.

cache_life_time

timeout()

The time of a cached item to keep in cache. Once it's expired, the corresponding item is erased from cache. The default value is *1 hour*. Several modules have a similar option; and some core ejabberd parts support similar options too, see auth_cache_life_time, oauth_cache_life_time, router_cache_life_time, and sm_cache_life_time.

cache_missed

true | false

Whether or not to cache missed lookups. When there is an attempt to lookup for a value in a database and this value is not found and the option is set to *true*, this attempt will be cached and no attempts will be performed until the cache expires (see cache_life_time). Usually you don't want to change it. Default is *true*. Several modules have a similar option; and some core

ejabberd parts support similar options too, see auth_cache_missed, oauth_cache_missed, router_cache_missed, and sm_cache_missed.

cache_size

pos_integer() | infinity

A maximum number of items (not memory!) in cache. The rule of thumb, for all tables except rosters, you should set it to the number of maximum online users you expect. For roster multiply this number by 20 or so. If the cache size reaches this threshold, it's fully cleared, i.e. all items are deleted, and the corresponding warning is logged. You should avoid frequent cache clearance, because this degrades performance. The default value is *1000*. Several modules have a similar option; and some core ejabberd parts support similar options too, see auth_cache_size, oauth_cache_size, router_cache_size, and sm_cache_size.

captcha_cmd

Path | ModuleName

🏷 improved in 23.01

Full path to a script that generates CAPTCHA images. *@VERSION@* is replaced with ejabberd version number in *XX.YY* format. *@SEMVER@* is replaced with ejabberd version number in semver format when compiled with Elixir's mix, or XX.YY format otherwise. Alternatively, it can be the name of a module that implements ejabberd CAPTCHA support. There is no default value: when this option is not set, CAPTCHA functionality is completely disabled.

Examples:

When using the ejabberd installers or container image, the example captcha scripts can be used like this:

```
captcha_cmd: /opt/ejabberd-@VERSION@/lib/ejabberd-@SEMVER@/priv/bin/captcha.sh
```

captcha_host

String

Deprecated. Use captcha_url instead.

captcha_limit

pos_integer() | infinity

Maximum number of CAPTCHA generated images per minute for any given JID. The option is intended to protect the server from CAPTCHA DoS. The default value is *infinity*.

captcha_url

URL | auto | undefined

🏷 improved in 23.04

An URL where CAPTCHA requests should be sent. NOTE: you need to configure *request_handlers* for *ejabberd_http* listener as well. If set to *auto*, it builds the URL using a *request_handler* already enabled, with encryption if available. If set to *undefined*, it builds the URL using the deprecated captcha_host + /captcha. The default value is *auto*.

certfiles

[Path, ...]

The option accepts a list of file paths (optionally with wildcards) containing either PEM certificates or PEM private keys. At startup or configuration reload, ejabberd reads all certificates from these files, sorts them, removes duplicates, finds matching private keys and then rebuilds full certificate chains for the use in TLS connections. Use this option when TLS is enabled in either of ejabberd listeners: *ejabberd_c2s*, *ejabberd_http* and so on. NOTE: if you modify the certificate files or change the value of the option, run *ejabberdctl reload-config* in order to rebuild and reload the certificate chains.

Examples:

If you use Let's Encrypt certificates for your domain "domain.tld", the configuration will look like this:

```
certfiles:
  - /etc/letsencrypt/live/domain.tld/fullchain.pem
  - /etc/letsencrypt/live/domain.tld/privkey.pem
```

cluster_backend

Backend

A database backend to use for storing information about cluster. The only available value so far is *mnesia*.

cluster_nodes

[Node, ...]

A list of Erlang nodes to connect on ejabberd startup. This option is mostly intended for ejabberd customization and sophisticated setups. The default value is an empty list.

default_db

mnesia | sql

Default persistent storage for ejabberd. Modules and other components (e.g. authentication) may have its own value. The default value is *mnesia*.

default_ram_db

mnesia | redis | sql

Default volatile (in-memory) storage for ejabberd. Modules and other components (e.g. session management) may have its own value. The default value is *mnesia*.

define_macro

{MacroName: MacroValue}

Defines a macro. The value can be any valid arbitrary YAML value. For convenience, it's recommended to define a *MacroName* in capital letters. Duplicated macros are not allowed. Macros are processed after additional configuration files have been included, so it is possible to use macros that are defined in configuration files included before the usage. It is possible to use a *MacroValue* in the definition of another macro.

Example:

```
define_macro:
  DEBUG: debug
  LOG_LEVEL: DEBUG
  USERBOB:
    user: bob@localhost

loglevel: LOG_LEVEL

acl:
  admin: USERBOB
```

disable_sasl_mechanisms

[Mechanism, ...]

Specify a list of SASL mechanisms (such as *DIGEST-MD5* or *SCRAM-SHA1*) that should not be offered to the client. For convenience, the value of *Mechanism* is case-insensitive. The default value is an empty list, i.e. no mechanisms are disabled by default.

disable_sasl_scram_downgrade_protection

true | false

Allows to disable sending data required by *XEP-0474: SASL SCRAM Downgrade Protection*. There are known buggy clients (like those that use strophejs 1.6.2) which will not be able to authenticatate when servers sends data from that specification. This options allows server to disable it to allow even buggy clients connects, but in exchange decrease MITM protection. The default value of this option is *false* which enables this extension.

domain_balancing

{Domain: Options}

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 node and that the traffic will be automatically distributed. The algorithm to deliver messages to the component(s) can be specified by this option. For any component connected as *Domain*, available *Options* are:

- **component_number**: *2..1000*
  The number of components to balance.

- **type**: *random | source | destination | bare_source | bare_destination* How to deliver stanzas to connected components: *random* - an instance is chosen at random; *destination* - an instance is chosen by the full JID of the packet's *to* attribute; *source* - by the full JID of the packet's *from* attribute; *bare_destination* - by the bare JID (without resource) of the packet's *to* attribute; *bare_source* - by the bare JID (without resource) of the packet's *from* attribute is used. The default value is *random*.

Example:

```
domain_balancing:
  component.domain.tld:
    type: destination
    component_number: 5
  transport.example.org:
    type: bare_source
```

ext_api_headers

Headers

String of headers (separated with commas ,) that will be provided by ejabberd when sending ReST requests. The default value is an empty string of headers: "".

ext_api_http_pool_size

pos_integer()

Define the size of the HTTP pool, that is, the maximum number of sessions that the ejabberd ReST service will handle simultaneously. The default value is: *100*.

ext_api_path_oauth

Path

Define the base URI path when performing OAUTH ReST requests. The default value is: "*/oauth*".

ext_api_url

URL

Define the base URI when performing ReST requests. The default value is: "*http://localhost/api*".

extauth_pool_name

Name

Define the pool name appendix, so the full pool name will be *extauth_pool_Name*. The default value is the hostname.

extauth_pool_size

Size

The option defines the number of instances of the same external program to start for better load balancing. The default is the number of available CPU cores.

extauth_program

Path

Indicate in this option the full path to the external authentication script. The script must be executable by ejabberd.

fqdn

Domain

A fully qualified domain name that will be used in SASL DIGEST-MD5 authentication. The default is detected automatically.

hide_sensitive_log_data

true | false

A privacy option to not log sensitive data (mostly IP addresses). The default value is *false* for backward compatibility.

host_config

{Host: Options}

The option is used to redefine *Options* for virtual host *Host*. In the example below LDAP authentication method will be used on virtual host *domain.tld* and SQL method will be used on virtual host *example.org*.

Example:

```
hosts:
  - domain.tld
  - example.org

auth_method:
  - sql

host_config:
  domain.tld:
    auth_method:
      - ldap
```

hosts

[Domain1, Domain2, ...]

The option defines a list containing one or more domains that *ejabberd* will serve. This is a **mandatory** option.

include_config_file

[Filename, ...] | {Filename: Options}

Read additional configuration from *Filename*. If the value is provided in *{Filename: Options}* format, the *Options* must be one of the following:

- **allow_only**: *[OptionName, ...]*
  Allows only the usage of those options in the included file *Filename*. The options that do not match this criteria are not accepted. The default value is to include all options.
- **disallow**: *[OptionName, ...]*
  Disallows the usage of those options in the included file *Filename*. The options that match this criteria are not accepted. The default value is an empty list.

install_contrib_modules

[Module, ...]

added in 23.10

Modules to install from ejabberd-contrib at start time. The default value is an empty list of modules: *[]*.

jwt_auth_only_rule

AccessName

This ACL rule defines accounts that can use only this auth method, even if others are also defined in the ejabberd configuration file. In other words: if there are several auth methods enabled for this host (JWT, SQL, ...), users that match this rule can only use JWT. The default value is *none*.

jwt_jid_field

FieldName

By default, the JID is defined in the "*jid*" JWT field. In this option you can specify other JWT field name where the JID is defined.

jwt_key

FilePath

Path to the file that contains the JWK Key. The default value is *undefined*.

language

Language

The option defines the default language of server strings that can be seen by XMPP clients. If an XMPP client does not possess *xml:lang* attribute, the specified language is used. The default value is "*en*".

ldap_backups

[Host, ...]

A list of IP addresses or DNS names of LDAP backup servers. When no servers listed in ldap_servers option are reachable, ejabberd will try to connect to these backup servers. The default is an empty list, i.e. no backup servers specified. WARNING: ejabberd doesn't try to reconnect back to the main servers when they become operational again, so the only way to restore these connections is to restart ejabberd. This limitation might be fixed in future releases.

ldap_base

Base

LDAP base directory which stores users accounts. There is no default value: you must set the option in order for LDAP connections to work properly.

ldap_deref_aliases

never | always | finding | searching

Whether to dereference aliases or not. The default value is *never*.

ldap_dn_filter

{Filter: FilterAttrs}

This filter is applied on the results returned by the main filter. The filter performs an additional LDAP lookup to make the complete result. This is useful when you are unable to define all filter rules in *ldap_filter*. You can define "%u", "%d", "%s" and "%D" pattern variables in *Filter*: "%u" is replaced by a user's part of the JID, "%d" is replaced by the corresponding domain (virtual host), all "%s" variables are consecutively replaced by values from the attributes in *FilterAttrs* and "%D" is replaced by Distinguished Name from the result set. There is no default value, which means the result is not filtered. WARNING: Since this filter makes additional LDAP lookups, use it only as the last resort: try to define all filter rules in ldap_filter option if possible.

Example:

```
ldap_dn_filter:
  "(&(name=%s)(owner=%D)(user=%u@%d))": [sn]
```

ldap_encrypt

tls | none

Whether to encrypt LDAP connection using TLS or not. The default value is *none*. NOTE: STARTTLS encryption is not supported.

ldap_filter

Filter

An LDAP filter as defined in [RFC4515](). There is no default value. Example: "(&(objectClass=shadowAccount)(memberOf=XMPP Users))". NOTE: don't forget to close brackets and don't use superfluous whitespaces. Also you must not use "uid" attribute in the filter because this attribute will be appended to the filter automatically.

ldap_password

Password

Bind password. The default value is an empty string.

ldap_port

1..65535

Port to connect to your LDAP server. The default port is *389* if encryption is disabled and *636* if encryption is enabled.

ldap_rootdn

RootDN

Bind Distinguished Name. The default value is an empty string, which means "anonymous connection".

ldap_servers

[Host, ...]

A list of IP addresses or DNS names of your LDAP servers. The default value is *[localhost]*.

ldap_tls_cacertfile

Path

A path to a file containing PEM encoded CA certificates. This option is required when TLS verification is enabled.

ldap_tls_certfile

Path

A path to a file containing PEM encoded certificate along with PEM encoded private key. This certificate will be provided by ejabberd when TLS enabled for LDAP connections. There is no default value, which means no client certificate will be sent.

ldap_tls_depth

Number

Specifies the maximum verification depth when TLS verification is enabled, i.e. how far in a chain of certificates the verification process can proceed before the verification is considered to be failed. Peer certificate = 0, CA certificate = 1, higher level CA certificate = 2, etc. The value *2* thus means that a chain can at most contain peer cert, CA cert, next CA cert, and an additional CA cert. The default value is *1*.

ldap_tls_verify

false | soft | hard

This option specifies whether to verify LDAP server certificate or not when TLS is enabled. When *hard* is set, ejabberd doesn't proceed if the certificate is invalid. When *soft* is set, ejabberd proceeds even if the check has failed. The default is *false*, which means no checks are performed.

ldap_uids

[Attr] | {Attr: AttrFormat}

LDAP attributes which hold a list of attributes to use as alternatives for getting the JID, where *Attr* is an LDAP attribute which holds the user's part of the JID and *AttrFormat* must contain one and only one pattern variable "%u" which will be replaced by the user's part of the JID. For example, "%u@example.org". If the value is in the form of *[Attr]* then *AttrFormat* is assumed to be "%u".

listen

[Options, ...]

The option for listeners configuration. See the Listen Modules section for details.

log_burst_limit_count

Number

🏷 added in 22.10

The number of messages to accept in `log_burst_limit_window_time` period before starting to drop them. Default 500

log_burst_limit_window_time

Number

🏷 added in 22.10

The time period to rate-limit log messages by. Defaults to 1 second.

log_modules_fully

[Module, ...]

🏷 added in 23.01

List of modules that will log everything independently from the general loglevel option.

log_rotate_count

Number

The number of rotated log files to keep. The default value is *1*, which means that only keeps `ejabberd.log.0`, `error.log.0` and `crash.log.0`.

log_rotate_size

pos_integer() | infinity

The size (in bytes) of a log file to trigger rotation. If set to *infinity*, log rotation is disabled. The default value is *10485760* (that is, 10 Mb).

loglevel

none | emergency | alert | critical | error | warning | notice | info | debug

Verbosity of log files generated by ejabberd. The default value is *info*. NOTE: previous versions of ejabberd had log levels defined in numeric format (*0..5*). The numeric values are still accepted for backward compatibility, but are not recommended.

max_fsm_queue

Size

This option specifies the maximum number of elements in the queue of the FSM (Finite State Machine). Roughly speaking, each message in such queues represents one XML stanza queued to be sent into its relevant outgoing stream. If queue size reaches the limit (because, for example, the receiver of stanzas is too slow), the FSM and the corresponding connection (if any) will be terminated and error message will be logged. The reasonable value for this option depends on your hardware configuration. The allowed values are positive integers. The default value is *10000*.

modules

{Module: Options}

The option for modules configuration. See Modules section for details.

negotiation_timeout

timeout()

Time to wait for an XMPP stream negotiation to complete. When timeout occurs, the corresponding XMPP stream is closed. The default value is *120* seconds.

net_ticktime

timeout()

This option can be used to tune tick time parameter of *net_kernel*. It tells Erlang VM how often nodes should check if intra-node communication was not interrupted. This option must have identical value on all nodes, or it will lead to subtle bugs. Usually leaving default value of this is option is best, tweak it only if you know what you are doing. The default value is *1 minute*.

new_sql_schema

true | false

Whether to use *new* SQL schema. All schemas are located at https://github.com/processone/ejabberd/tree/24.02/sql. There are two schemas available. The default legacy schema stores one XMPP domain into one ejabberd database. The *new* schema can handle several XMPP domains in a single ejabberd database. Using this *new* schema is best when serving several XMPP domains and/or changing domains from time to time. This avoid need to manage several databases and handle complex configuration changes. The default depends on configuration flag *--enable-new-sql-schema* which is set at compile time.

oauth_access

AccessName

By default creating OAuth tokens is not allowed. To define which users can create OAuth tokens, you can refer to an ejabberd access rule in the *oauth_access* option. Use *all* to allow everyone to create tokens.

oauth_cache_life_time

timeout()

Same as cache_life_time, but applied to OAuth cache only. If not set, the value from cache_life_time will be used.

oauth_cache_missed

true | false

Same as cache_missed, but applied to OAuth cache only. If not set, the value from cache_missed will be used.

oauth_cache_rest_failure_life_time

timeout()

🏷 added in 21.01

The time that a failure in OAuth ReST is cached. The default value is *infinity*.

oauth_cache_size

pos_integer() | infinity

Same as cache_size, but applied to OAuth cache only. If not set, the value from cache_size will be used.

oauth_client_id_check

allow | db | deny

Define whether the client authentication is always allowed, denied, or it will depend if the client ID is present in the database. The default value is *allow*.

oauth_db_type

mnesia | sql

Database backend to use for OAuth authentication. The default value is picked from default_db option, or if it's not set, *mnesia* will be used.

oauth_expire

timeout()

Time during which the OAuth token is valid, in seconds. After that amount of time, the token expires and the delegated credential cannot be used and is removed from the database. The default is *4294967* seconds.

oauth_use_cache

true | false

Same as use_cache, but applied to OAuth cache only. If not set, the value from use_cache will be used.

oom_killer

true | false

Enable or disable OOM (out-of-memory) killer. When system memory raises above the limit defined in oom_watermark option, ejabberd triggers OOM killer to terminate most memory consuming Erlang processes. Note that in order to maintain functionality, ejabberd only attempts to kill transient processes, such as those managing client sessions, s2s or database connections. The default value is *true*.

oom_queue

Size

Trigger OOM killer when some of the running Erlang processes have messages queue above this *Size*. Note that such processes won't be killed if oom_killer option is set to *false* or if *oom_watermark* is not reached yet.

oom_watermark

Percent

A percent of total system memory consumed at which OOM killer should be activated with some of the processes possibly be killed (see oom_killer option). Later, when memory drops below this *Percent*, OOM killer is deactivated. The default value is *80* percents.

outgoing_s2s_families

[ipv6 | ipv4, ...]

🏷 changed in 23.01

Specify which address families to try, in what order. The default is *[ipv6, ipv4]* which means it first tries connecting with IPv6, if that fails it tries using IPv4. This option is obsolete and irrelevant when using ejabberd 23.01 and Erlang/OTP 22, or newer versions of them.

outgoing_s2s_ipv4_address

Address

🏷 added in 20.12

Specify the IPv4 address that will be used when establishing an outgoing S2S IPv4 connection, for example "127.0.0.1". The default value is *undefined*.

outgoing_s2s_ipv6_address

Address

🏷 added in 20.12

Specify the IPv6 address that will be used when establishing an outgoing S2S IPv6 connection, for example "::FFFF:127.0.0.1". The default value is *undefined*.

outgoing_s2s_port

1..65535

A port number to use for outgoing s2s connections when the target server doesn't have an SRV record. The default value is *5269*.

outgoing_s2s_timeout

timeout()

The timeout in seconds for outgoing S2S connection attempts. The default value is *10* seconds.

pam_service

Name

This option defines the PAM service name. Refer to the PAM documentation of your operation system for more information. The default value is *ejabberd*.

pam_userinfotype

username | jid

This option defines what type of information about the user ejabberd provides to the PAM service: only the username, or the user's JID. Default is *username*.

pgsql_users_number_estimate

true | false

Whether to use PostgreSQL estimation when counting registered users. The default value is *false*.

queue_dir

Directory

If queue_type option is set to *file*, use this *Directory* to store file queues. The default is to keep queues inside Mnesia directory.

queue_type

ram | file

Default type of queues in ejabberd. Modules may have its own value of the option. The value of *ram* means that queues will be kept in memory. If value *file* is set, you may also specify directory in queue_dir option where file queues will be placed. The default value is *ram*.

redis_connect_timeout

timeout()

A timeout to wait for the connection to be re-established to the Redis server. The default is *1 second*.

redis_db

Number

Redis database number. The default is *0*.

redis_password

Password

The password to the Redis server. The default is an empty string, i.e. no password.

redis_pool_size

Number

The number of simultaneous connections to the Redis server. The default value is *10*.

redis_port

1..65535

The port where the Redis server is accepting connections. The default is *6379*.

redis_queue_type

ram | file

The type of request queue for the Redis server. See description of queue_type option for the explanation. The default value is the value defined in queue_type or *ram* if the latter is not set.

redis_server

Hostname

A hostname or an IP address of the Redis server. The default is *localhost*.

registration_timeout

timeout()

This is a global option for module mod_register. It limits the frequency of registrations from a given IP or username. So, a user that tries to register a new account from the same IP address or JID during this time after their previous registration will receive an error with the corresponding explanation. To disable this limitation, set the value to *infinity*. The default value is *600 seconds*.

resource_conflict

setresource | closeold | closenew

NOTE: this option is deprecated and may be removed anytime in the future versions. The possible values match exactly the three possibilities described in XMPP Core: section 7.7.2.2. The default value is *closeold*. If the client uses old Jabber Non-SASL authentication (XEP-0078), then this option is not respected, and the action performed is *closeold*.

router_cache_life_time

timeout()

Same as cache_life_time, but applied to routing table cache only. If not set, the value from cache_life_time will be used.

router_cache_missed

true | false

Same as cache_missed, but applied to routing table cache only. If not set, the value from cache_missed will be used.

router_cache_size

pos_integer() | infinity

Same as cache_size, but applied to routing table cache only. If not set, the value from cache_size will be used.

router_db_type

mnesia | redis | sql

Database backend to use for routing information. The default value is picked from default_ram_db option, or if it's not set, *mnesia* will be used.

router_use_cache

true | false

Same as use_cache, but applied to routing table cache only. If not set, the value from use_cache will be used.

rpc_timeout

timeout()

A timeout for remote function calls between nodes in an ejabberd cluster. You should probably never change this value since those calls are used for internal needs only. The default value is *5* seconds.

s2s_access

Access

This Access Rule defines to what remote servers can s2s connections be established. The default value is *all*; no restrictions are applied, it is allowed to connect s2s to/from all remote servers.

s2s_cafile

Path

A path to a file with CA root certificates that will be used to authenticate s2s connections. If not set, the value of ca_file will be used.

You can use host_config to specify this option per-vhost.

s2s_ciphers

[Cipher, ...]

A list of OpenSSL ciphers to use for s2s connections. The default value is shown in the example below:

Example:

```
s2s_ciphers:
  - HIGH
  - "!aNULL"
  - "!eNULL"
  - "!3DES"
  - "@STRENGTH"
```

s2s_dhfile

Path

Full path to a file containing custom DH parameters to use for s2s connections. Such a file could be created with the command "openssl dhparam -out dh.pem 2048". If this option is not specified, 2048-bit MODP Group with 256-bit Prime Order Subgroup will be used as defined in RFC5114 Section 2.3.

s2s_dns_retries

Number

DNS resolving retries. The default value is *2*.

s2s_dns_timeout

timeout()

The timeout for DNS resolving. The default value is *10* seconds.

s2s_max_retry_delay

timeout()

The maximum allowed delay for s2s connection retry to connect after a failed connection attempt. The default value is *300* seconds (5 minutes).

s2s_protocol_options

[Option, ...]

List of general SSL options to use for s2s connections. These map to OpenSSL's *set_options()*. The default value is shown in the example below:

Example:

```
s2s_protocol_options:
  - no_sslv3
  - cipher_server_preference
  - no_compression
```

s2s_queue_type

ram | file

The type of a queue for s2s packets. See description of queue_type option for the explanation. The default value is the value defined in queue_type or *ram* if the latter is not set.

s2s_timeout

timeout()

A time to wait before closing an idle s2s connection. The default value is *1* hour.

s2s_tls_compression

true | false

Whether to enable or disable TLS compression for s2s connections. The default value is *false*.

s2s_use_starttls

true | false | optional | required

Whether to use STARTTLS for s2s connections. The value of *false* means STARTTLS is prohibited. The value of *true* or *optional* means STARTTLS is enabled but plain connections are still allowed. And the value of *required* means that only STARTTLS connections are allowed. The default value is *false* (for historical reasons).

s2s_zlib

true | false

Whether to use *zlib* compression (as defined in XEP-0138) or not. The default value is *false*. WARNING: this type of compression is nowadays considered insecure.

shaper

{ShaperName: Rate}

The option defines a set of shapers. Every shaper is assigned a name *ShaperName* that can be used in other parts of the configuration file, such as shaper_rules option. The shaper itself is defined by its *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. In the example below shaper *normal* limits the traffic speed to 1,000 bytes/sec and shaper *fast* limits the traffic speed to 50,000 bytes/sec:

Example:

```
shaper:
  normal: 1000
  fast: 50000
```

shaper_rules

{ShaperRuleName: {Number|ShaperName: ACLRule|ACLName}}

An entry allowing to declaring shaper to use for matching user/hosts. Semantics is similar to access_rules option, the only difference is that instead using *allow* or *deny*, a name of a shaper (defined in shaper option) or a positive number should be used.

Example:

```
shaper_rules:
  connections_limit:
    10:
      user: peter@example.com
    100: admin
    5: all
  download_speed:
    fast: admin
    slow: anonymous_users
    normal: all
  log_days: 30
```

sm_cache_life_time

timeout()

Same as cache_life_time, but applied to client sessions table cache only. If not set, the value from cache_life_time will be used.

sm_cache_missed

true | false

Same as cache_missed, but applied to client sessions table cache only. If not set, the value from cache_missed will be used.

sm_cache_size

pos_integer() | infinity

Same as cache_size, but applied to client sessions table cache only. If not set, the value from cache_size will be used.

sm_db_type

mnesia | redis | sql

Database backend to use for client sessions information. The default value is picked from default_ram_db option, or if it's not set, *mnesia* will be used.

sm_use_cache

true | *false*

Same as use_cache, but applied to client sessions table cache only. If not set, the value from use_cache will be used.

sql_connect_timeout

timeout()

A time to wait for connection to an SQL server to be established. The default value is *5* seconds.

sql_database

Database

An SQL database name. For SQLite this must be a full path to a database file. The default value is *ejabberd*.

sql_flags ⬤

[mysql_alternative_upsert]

🏷 added in 24.02

This option accepts a list of SQL flags, and is empty by default. *mysql_alternative_upsert* forces the alternative upsert implementation in MySQL.

sql_keepalive_interval

timeout()

An interval to make a dummy SQL request to keep alive the connections to the database. There is no default value, so no keepalive requests are made.

sql_odbc_driver

Path

🏷 added in 20.12

Path to the ODBC driver to use to connect to a Microsoft SQL Server database. This option only applies if the sql_type option is set to *mssql* and sql_server is not an ODBC connection string. The default value is: *libtdsodbc.so*

sql_password

Password

The password for SQL authentication. The default is empty string.

sql_pool_size

Size

Number of connections to the SQL server that ejabberd will open for each virtual host. The default value is 10. WARNING: for SQLite this value is *1* by default and it's not recommended to change it due to potential race conditions.

sql_port

1..65535

The port where the SQL server is accepting connections. The default is *3306* for MySQL, *5432* for PostgreSQL and *1433* for MS SQL. The option has no effect for SQLite.

sql_prepared_statements

true | false

🏷 added in 20.01

This option is *true* by default, and is useful to disable prepared statements. The option is valid for PostgreSQL and MySQL.

sql_query_timeout

timeout()

A time to wait for an SQL query response. The default value is *60* seconds.

sql_queue_type

ram | file

The type of a request queue for the SQL server. See description of queue_type option for the explanation. The default value is the value defined in queue_type or *ram* if the latter is not set.

sql_server

Host

🏷 improved in 23.04

The hostname or IP address of the SQL server. For sql_type *mssql* or *odbc* this can also be an ODBC connection string. The default value is *localhost*.

sql_ssl

true | false

🏷 improved in 20.03

Whether to use SSL encrypted connections to the SQL server. The option is only available for MySQL, MS SQL and PostgreSQL. The default value is *false*.

sql_ssl_cafile

Path

A path to a file with CA root certificates that will be used to verify SQL connections. Implies sql_ssl and sql_ssl_verify options are set to *true*. There is no default which means certificate verification is disabled. This option has no effect for MS SQL.

sql_ssl_certfile

Path

A path to a certificate file that will be used for SSL connections to the SQL server. Implies sql_ssl option is set to *true*. There is no default which means ejabberd won't provide a client certificate to the SQL server. This option has no effect for MS SQL.

sql_ssl_verify

true | false

Whether to verify SSL connection to the SQL server against CA root certificates defined in sql_ssl_cafile option. Implies sql_ssl option is set to *true*. This option has no effect for MS SQL. The default value is *false*.

sql_start_interval

timeout()

A time to wait before retrying to restore failed SQL connection. The default value is *30* seconds.

sql_type

mssql | mysql | odbc | pgsql | sqlite

The type of an SQL connection. The default is *odbc*.

sql_username

Username

A user name for SQL authentication. The default value is *ejabberd*.

trusted_proxies

all | [Network1, Network2, ...]

Specify what proxies are trusted when an HTTP request contains the header *X-Forwarded-For*. You can specify *all* to allow all proxies, or specify a list of IPs, possibly with masks. The default value is an empty list. Using this option you can know the real IP of the request, for admin purpose, or security configuration (for example using *mod_fail2ban*). IMPORTANT: The proxy MUST be configured to set the *X-Forwarded-For* header if you enable this option as, otherwise, the client can set it itself and as a result the IP value cannot be trusted for security rules in ejabberd.

update_sql_schema

true | false

🏷 added in 23.10

Allow ejabberd to update SQL schema. The default value is *false*.

use_cache

true | false

Enable or disable cache. The default is *true*. Several modules have a similar option; and some core ejabberd parts support similar options too, see auth_use_cache, oauth_use_cache, router_use_cache, and sm_use_cache.

validate_stream

true | false

Whether to validate any incoming XML packet according to the schemas of supported XMPP extensions. WARNING: the validation is only intended for the use by client developers - don't enable it in production environment. The default value is *false*.

version

string()

The option can be used to set custom ejabberd version, that will be used by different parts of ejabberd, for example by mod_version module. The default value is obtained at compile time from the underlying version control system.

websocket_origin

ignore | URL

This option enables validation for *Origin* header to protect against connections from other domains than given in the configuration file. In this way, the lower layer load balancer can be chosen for a specific ejabberd implementation while still providing a secure WebSocket connection. The default value is *ignore*. An example value of the *URL* is "https://test.example.org: 8081".

websocket_ping_interval

timeout()

Defines time between pings sent by the server to a client (WebSocket level protocol pings are used for this) to keep a connection active. If the client doesn't respond to two consecutive pings, the connection will be assumed as closed. The value of *0* can be used to disable the feature. This option makes the server sending pings only for connections using the RFC compliant protocol. For older style connections the server expects that whitespace pings would be used for this purpose. The default value is *60* seconds.

websocket_timeout

timeout()

Amount of time without any communication after which the connection would be closed. The default value is *300* seconds.

Modules Options

This section describes modules options of ejabberd 24.02. If you are using an old ejabberd release, please refer to the corresponding archived version of this page in the Archive. The modules that changed in this version are marked with ⬤.

mod_adhoc

This module implements XEP-0050: Ad-Hoc Commands. It's an auxiliary module and is only needed by some of the other modules.

Available options:

- **report_commands_node**: *true | false*
  Provide the Commands item in the Service Discovery. Default value: *false*.

mod_admin_extra

This module provides additional administrative commands.

Details for some commands:

- *ban-account*: This command kicks all the connected sessions of the account from the server. It also changes their password to a randomly generated one, so they can't login anymore unless a server administrator changes their password again. It is possible to define the reason of the ban. The new password also includes the reason and the date and time of the ban. See an example below.
- *pushroster*: (and *pushroster-all*) The roster file must be placed, if using Windows, on the directory where you installed ejabberd: `C:/Program Files/ejabberd` or similar. If you use other Operating System, place the file on the same directory where the .beam files are installed. See below an example roster file.
- *srg-create*: If you want to put a group Name with blank spaces, use the characters "' and '" to define when the Name starts and ends. See an example below.

The module has no options.

Examples:

With this configuration, vCards can only be modified with mod_admin_extra commands:

```
acl:
  adminextraresource:
    - resource: "modadminextraf8x,31ad"
access_rules:
  vcard_set:
    - allow: adminextraresource
modules:
  mod_admin_extra: {}
  mod_vcard:
    access_set: vcard_set
```

Content of roster file for *pushroster* command:

```
[{<<"bob">>, <<"example.org">>, <<"workers">>, <<"Bob">>},
{<<"mart">>, <<"example.org">>, <<"workers">>, <<"Mart">>},
{<<"Rich">>, <<"example.org">>, <<"bosses">>, <<"Rich">>}].
```

With this call, the sessions of the local account which JID is boby@example.org will be kicked, and its password will be set to something like *BANNED_ACCOUNT—20080425T21:45:07—2176635—Spammed_rooms*

```
ejabberdctl vhost example.org ban-account boby "Spammed rooms"
```

Call to srg-create using double-quotes and single-quotes:

```
ejabberdctl srg-create g1 example.org "'Group number 1'" this_is_g1 g1
```

mod_admin_update_sql

This module can be used to update existing SQL database from the default to the new schema. Check the section Default and New Schemas for details. Please note that only MS SQL, MySQL, and PostgreSQL are supported. When the module is loaded use update_sql API.

The module has no options.

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 an XMPP client either using Ad-hoc Commands or sending messages to specific JIDs.

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

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 below. 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 mod_offline) 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).

Available options:

- **access**: *AccessName*
  This option specifies who is allowed to send announcements and to set the message of the day. The default value is *none* (i.e. nobody is able to send such messages).

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

mod_avatar

The purpose of the module is to cope with legacy and modern XMPP clients posting avatars. The process is described in [XEP-0398: User Avatar to vCard-Based Avatars Conversion](#).

Also, the module supports conversion between avatar image formats on the fly.

The module depends on [mod_vcard](#), [mod_vcard_xupdate](#) and [mod_pubsub](#).

Available options:

- **convert**: *{From: To}*
  Defines image conversion rules: the format in *From* will be converted to format in *To*. The value of *From* can also be *default*, which is match-all rule. NOTE: the list of supported formats is detected at compile time depending on the image libraries installed in the system.
  Example:

  ```
  convert:
    webp: jpg
    default: png
  ```

- **rate_limit**: *Number*
  Limit any given JID by the number of avatars it is able to convert per minute. This is to protect the server from image conversion DoS. The default value is *10*.

mod_block_strangers

This module blocks and logs any messages coming from an unknown entity. If a writing entity is not in your roster, you can let this module drop and/or log the message. By default you'll just not receive message from that entity. Enable this module if you want to drop SPAM messages.

Available options:

- **access**: *AccessName*
  The option is supposed to be used when *allow_local_users* and *allow_transports* are not enough. It's an ACL where *deny* means the message will be rejected (or a CAPTCHA would be generated for a presence, if configured), and *allow* means the sender is whitelisted and the stanza will pass through. The default value is *none*, which means nothing is whitelisted.

- **allow_local_users**: *true | false*
  This option specifies if strangers from the same local host should be accepted or not. The default value is *true*.

- **allow_transports**: *true | false*
  If set to *true* and some server's JID is in user's roster, then messages from any user of this server are accepted even if no subscription present. The default value is *true*.

- **captcha**: *true | false*
  Whether to generate CAPTCHA or not in response to messages from strangers. See also section [CAPTCHA](#) of the Configuration Guide. The default value is *false*.

- **drop**: *true | false*
  This option specifies if strangers messages should be dropped or not. The default value is *true*.

- **log**: *true | false*
  This option specifies if strangers' messages should be logged (as info message) in ejabberd.log. The default value is *false*.

mod_blocking

The module implements [XEP-0191: Blocking Command](#).

This module depends on *mod_privacy* where all the configuration is performed.

The module has no options.

mod_bosh

This module implements XMPP over BOSH as defined in XEP-0124 and XEP-0206. BOSH stands for Bidirectional-streams Over Synchronous HTTP. It makes it possible to simulate long lived connections required by XMPP over the HTTP protocol. In practice, this module makes it possible to use XMPP in a browser without WebSocket support and more generally to have a way to use XMPP while having to get through an HTTP proxy.

Available options:

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **json**: *true | false*
  This option has no effect.

- **max_concat**: *pos_integer() | infinity*
  This option limits the number of stanzas that the server will send in a single bosh request. The default value is *unlimited*.

- **max_inactivity**: *timeout()*
  The option defines the maximum inactivity period. The default value is *30* seconds.

- **max_pause**: *pos_integer()*
  Indicate the maximum length of a temporary session pause (in seconds) that a client can request. The default value is *120*.

- **prebind**: *true | false*
  If enabled, the client can create the session without going through authentication. Basically, it creates a new session with anonymous authentication. The default value is *false*.

- **queue_type**: *ram | file*
  Same as top-level queue_type option, but applied to this module only.

- **ram_db_type**: *mnesia | sql | redis*
  Same as top-level default_ram_db option, but applied to this module only.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

Example:

```
listen:
  -
    port: 5222
    module: ejabberd_c2s
  -
    port: 5443
    module: ejabberd_http
    request_handlers:
      /bosh: mod_bosh

modules:
  mod_bosh: {}
```

mod_caps

This module implements XEP-0115: Entity Capabilities. The main purpose of the module is to provide PEP functionality (see mod_pubsub).

Available options:

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

mod_carboncopy

The module implements XEP-0280: Message Carbons. The module broadcasts messages on all connected user resources (devices).

The module has no options.

mod_client_state

This module allows for queueing certain types of stanzas when a client indicates that the user is not actively using the client right now (see XEP-0352: Client State Indication). This can save bandwidth and resources.

A stanza is dropped from the queue if it's effectively obsoleted by a new one (e.g., a new presence stanza would replace an old one from the same client). The queue is flushed if a stanza arrives that won't be queued, or if the queue size reaches a certain limit (currently 100 stanzas), or if the client becomes active again.

Available options:

- **queue_chat_states**: *true | false*
  Queue "standalone" chat state notifications (as defined in XEP-0085: Chat State Notifications) while a client indicates inactivity. The default value is *true*.

- **queue_pep**: *true | false*
  Queue PEP notifications while a client is inactive. When the queue is flushed, only the most recent notification of a given PEP node is delivered. The default value is *true*.

- **queue_presence**: *true | false*
  While a client is inactive, queue presence stanzas that indicate (un)availability. The default value is *true*.

mod_configure

The module provides server configuration functionality via XEP-0050: Ad-Hoc Commands. Implements many commands as defined in XEP-0133: Service Administration. This module requires mod_adhoc to be loaded.

The module has no options.

mod_conversejs

🏷 added in 21.12 and improved in 22.05

This module serves a simple page for the Converse XMPP web browser client.

To use this module, in addition to adding it to the *modules* section, you must also enable it in *listen → ejabberd_http →* request_handlers.

Make sure either *mod_bosh* or *ejabberd_http_ws* request_handlers are enabled.

When *conversejs_css* and *conversejs_script* are *auto*, by default they point to the public Converse client.

Available options:

- **bosh_service_url**: *auto | BoshURL*
  BOSH service URL to which Converse can connect to. The keyword *@HOST@* is replaced with the real virtual host name. If set to *auto*, it will build the URL of the first configured BOSH request handler. The default value is *auto*.

- **conversejs_css**: *auto | URL*
  Converse CSS URL. The keyword *@HOST@* is replaced with the hostname. The default value is *auto*.

- **conversejs_options**: *{Name: Value}*
  🏷 added in 22.05 Specify additional options to be passed to Converse. See Converse configuration. Only boolean, integer and string values are supported; lists are not supported.

- **conversejs_resources**: *Path*
  🏷 added in 22.05 Local path to the Converse files. If not set, the public Converse client will be used instead.

- **conversejs_script**: *auto | URL*
  Converse main script URL. The keyword *@HOST@* is replaced with the hostname. The default value is *auto*.

- **default_domain**: *Domain*
  Specify a domain to act as the default for user JIDs. The keyword *@HOST@* is replaced with the hostname. The default value is *@HOST@*.

- **websocket_url**: *auto | WebSocketURL*
  A WebSocket URL to which Converse can connect to. The keyword *@HOST@* is replaced with the real virtual host name. If set to *auto*, it will build the URL of the first configured WebSocket request handler. The default value is *auto*.

Examples:

Manually setup WebSocket url, and use the public Converse client:

```
listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /bosh: mod_bosh
      /websocket: ejabberd_http_ws
      /conversejs: mod_conversejs

modules:
  mod_bosh: {}
  mod_conversejs:
    websocket_url: "ws://@HOST@:5280/websocket"
```

Host Converse locally and let auto detection of WebSocket and Converse URLs:

```
listen:
  -
    port: 443
    module: ejabberd_http
    tls: true
    request_handlers:
      /websocket: ejabberd_http_ws
      /conversejs: mod_conversejs

modules:
  mod_conversejs:
    conversejs_resources: "/home/ejabberd/conversejs-9.0.0/package/dist"
```

Configure some additional options for Converse

```
modules:
  mod_conversejs:
    websocket_url: auto
    conversejs_options:
      auto_away: 30
```

```
        clear_cache_on_logout: true
        i18n: "pt"
        locked_domain: "@HOST@"
        message_archiving: always
        theme: dracula
```

mod_delegation

This module is an implementation of XEP-0355: Namespace Delegation. Only admin mode has been implemented by now. Namespace delegation allows external services to handle IQ using specific namespace. This may be applied for external PEP service.

Warning

Security issue: Namespace delegation gives components access to sensitive data, so permission should be granted carefully, only if you trust the component.

Note

This module is complementary to mod_privilege but can also be used separately.

Available options:

- **namespaces**: *{Namespace: Options}*
  If you want to delegate namespaces to a component, specify them in this option, and associate them to an access rule. The *Options* are:

- **access**: *AccessName*
  The option defines which components are allowed for namespace delegation. The default value is *none*.

- **filtering**: *Attributes*
  The list of attributes. Currently not used.

Examples:

Make sure you do not delegate the same namespace to several services at the same time. As in the example provided later, to have the *sat-pubsub.example.org* component perform correctly disable the *mod_pubsub* module.

```
access_rules:
  external_pubsub:
    allow: external_component
  external_mam:
    allow: external_component

acl:
  external_component:
    server: sat-pubsub.example.org

modules:
  mod_delegation:
    namespaces:
      urn:xmpp:mam:1:
        access: external_mam
      http://jabber.org/protocol/pubsub:
        access: external_pubsub
```

mod_disco

This module adds support for XEP-0030: Service Discovery. With this module enabled, services on your server can be discovered by XMPP clients.

Available options:

- **extra_domains**: *[Domain, ...]*
  With this option, you can specify a list of extra domains that are added to the Service Discovery item list. The default value is an empty list.

- **name**: *Name*
  A name of the server in the Service Discovery. This will only be displayed by special XMPP clients. The default value is *ejabberd*.

- **server_info**: *[Info, ...]*
  Specify additional information about the server, as described in XEP-0157: Contact Addresses for XMPP Services. Every *Info* element in the list is constructed from the following options:

- **modules**: *all | [Module, ...]*
  The value can be the keyword *all*, in which case the information is reported in all the services, or a list of ejabberd modules, in which case the information is only specified for the services provided by those modules.

- **name**: *Name*
  The field *var* name that will be defined. See XEP-0157 for some standardized names.

- **urls**: *[URI, ...]*
  A list of contact URIs, such as HTTP URLs, XMPP URIs and so on.

  Example:

```
server_info:
  -
    modules: all
    name: abuse-addresses
    urls: ["mailto:abuse@shakespeare.lit"]
  -
    modules: [mod_muc]
    name: "Web chatroom logs"
    urls: ["http://www.example.org/muc-logs"]
  -
    modules: [mod_disco]
    name: feedback-addresses
    urls:
      - http://shakespeare.lit/feedback.php
      - mailto:feedback@shakespeare.lit
      - xmpp:feedback@shakespeare.lit
  -
    modules:
      - mod_disco
      - mod_vcard
    name: admin-addresses
    urls:
      - mailto:xmpp@shakespeare.lit
      - xmpp:admins@shakespeare.lit
```

mod_fail2ban

The module bans IPs that show the malicious signs. Currently only C2S authentication failures are detected.

Unlike the standalone program, *mod_fail2ban* clears the record of authentication failures after some time since the first failure or on a successful authentication. It also does not simply block network traffic, but provides the client with a descriptive error message.

Warning

You should not use this module behind a proxy or load balancer. ejabberd will see the failures as coming from the load balancer and, when the threshold of auth failures is reached, will reject all connections coming from the load balancer. You can lock all your user base out of ejabberd when using this module behind a proxy.

Available options:

- **access**: *AccessName*
  Specify an access rule for whitelisting IP addresses or networks. If the rule returns *allow* for a given IP address, that address will never be banned. The *AccessName* should be of type *ip*. The default value is *none*.
- **c2s_auth_ban_lifetime**: *timeout()*
  The lifetime of the IP ban caused by too many C2S authentication failures. The default value is *1* hour.
- **c2s_max_auth_failures**: *Number*
  The number of C2S authentication failures to trigger the IP ban. The default value is *20*.

mod_host_meta

🏷 added in 22.05

This module serves small *host-meta* files as described in [XEP-0156: Discovering Alternative XMPP Connection Methods](#).

To use this module, in addition to adding it to the *modules* section, you must also enable it in *listen → ejabberd_http → request_handlers*.

Notice it only works if ejabberd_http has tls enabled.

Available options:

- **bosh_service_url**: *undefined | auto | BoshURL*
  BOSH service URL to announce. The keyword *@HOST@* is replaced with the real virtual host name. If set to *auto*, it will build the URL of the first configured BOSH request handler. The default value is *auto*.
- **websocket_url**: *undefined | auto | WebSocketURL*
  WebSocket URL to announce. The keyword *@HOST@* is replaced with the real virtual host name. If set to *auto*, it will build the URL of the first configured WebSocket request handler. The default value is *auto*.

Example:

```
listen:
  -
    port: 443
    module: ejabberd_http
    tls: true
    request_handlers:
      /bosh: mod_bosh
      /ws: ejabberd_http_ws
      /.well-known/host-meta: mod_host_meta
      /.well-known/host-meta.json: mod_host_meta

modules:
  mod_bosh: {}
  mod_host_meta:
    bosh_service_url: "https://@HOST@:5443/bosh"
    websocket_url: "wss://@HOST@:5443/ws"
```

mod_http_api

This module provides a ReST interface to call [ejabberd API](#) commands using JSON data.

To use this module, in addition to adding it to the *modules* section, you must also enable it in *listen → ejabberd_http → request_handlers*.

To use a specific API version N, when defining the URL path in the request_handlers, add a *vN*. For example: */api/v2: mod_http_api*

To run a command, send a POST request to the corresponding URL: *http://localhost:5280/api/<command_name>*

The module has no options.

Example:

```
listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /api: mod_http_api

modules:
  mod_http_api: {}
```

mod_http_fileserver

This simple module serves files from the local disk over HTTP.

Available options:

- **accesslog**: *Path*
  File to log accesses using an Apache-like format. No log will be recorded if this option is not specified.

- **content_types**: *{Extension: Type}*
  Specify mappings of extension to content type. There are several content types already defined. With this option you can add new definitions or modify existing ones. The default values are:

  Example:

  ```
  content_types:
    .css: text/css
    .gif: image/gif
    .html: text/html
    .jar: application/java-archive
    .jpeg: image/jpeg
    .jpg: image/jpeg
    .js: text/javascript
    .png: image/png
    .svg: image/svg+xml
    .txt: text/plain
    .xml: application/xml
    .xpi: application/x-xpinstall
    .xul: application/vnd.mozilla.xul+xml
  ```

- **custom_headers**: *{Name: Value}*
  Indicate custom HTTP headers to be included in all responses. There are no custom headers by default.

- **default_content_type**: *Type*
  Specify the content type to use for unknown extensions. The default value is *application/octet-stream*.

- **directory_indices**: *[Index, ...]*
  Indicate one or more directory index files, similarly to Apache's *DirectoryIndex* variable. When an HTTP request hits a directory instead of a regular file, those directory indices are looked in order, and the first one found is returned. The default value is an empty list.

- **docroot**: *Path*
  Directory to serve the files from. This is a mandatory option.

- **must_authenticate_with**: *[{Username, Hostname}, ...]*
  List of accounts that are allowed to use this service. Default value: *[]*.

Examples:

This example configuration will serve the files from the local directory */var/www* in the address *http://example.org:5280/pub/content/*. In this example a new content type *ogg* is defined, *png* is redefined, and *jpg* definition is deleted:

```
listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /pub/content: mod_http_fileserver

modules:
  mod_http_fileserver:
    docroot: /var/www
```

```
accesslog: /var/log/ejabberd/access.log
directory_indices:
  - index.html
  - main.htm
custom_headers:
  X-Powered-By: Erlang/OTP
  X-Fry: "It's a widely-believed fact!"
content_types:
  .ogg: audio/ogg
  .png: image/png
default_content_type: text/html
```

mod_http_upload

This module allows for requesting permissions to upload a file via HTTP as described in XEP-0363: HTTP File Upload. If the request is accepted, the client receives a URL for uploading the file and another URL from which that file can later be downloaded.

In order to use this module, it must be enabled in *listen → ejabberd_http →* request_handlers.

Available options:

- **access**: *AccessName*
  This option defines the access rule to limit who is permitted to use the HTTP upload service. The default value is *local*. If no access rule of that name exists, no user will be allowed to use the service.

- **custom_headers**: *{Name: Value}*
  This option specifies additional header fields to be included in all HTTP responses. By default no custom headers are included.

- **dir_mode**: *Permission*
  This option defines the permission bits of the *docroot* directory and any directories created during file uploads. The bits are specified as an octal number (see the chmod(1) manual page) within double quotes. For example: "0755". The default is undefined, which means no explicit permissions will be set.

- **docroot**: *Path*
  Uploaded files are stored below the directory specified (as an absolute path) with this option. The keyword @HOME@ is replaced with the home directory of the user running ejabberd, and the keyword @HOST@ with the virtual host name. The default value is "@HOME@/upload".

- **external_secret**: *Text*
  This option makes it possible to offload all HTTP Upload processing to a separate HTTP server. Both ejabberd and the HTTP server should share this secret and behave exactly as described at Prosody's mod_http_upload_external in the *Implementation* section. There is no default value.

- **file_mode**: *Permission*
  This option defines the permission bits of uploaded files. The bits are specified as an octal number (see the chmod(1) manual page) within double quotes. For example: "0644". The default is undefined, which means no explicit permissions will be set.

- **get_url**: *URL*
  This option specifies the initial part of the GET URLs used for downloading the files. The default value is *undefined*. When this option is *undefined*, this option is set to the same value as *put_url*. The keyword @HOST@ is replaced with the virtual host name. NOTE: if GET requests are handled by *mod_http_upload*, the *get_url* must match the *put_url*. Setting it to a different value only makes sense if an external web server or mod_http_fileserver is used to serve the uploaded files.

- **host**
  Deprecated. Use *hosts* instead.

- **hosts**: *[Host, ...]*
  This option defines the Jabber IDs of the service. If the *hosts* option is not specified, the only Jabber ID will be the hostname of the virtual host with the prefix "upload.". The keyword *@HOST@* is replaced with the real virtual host name.

- **jid_in_url**: *node | sha1*
  When this option is set to *node*, the node identifier of the user's JID (i.e., the user name) is included in the GET and PUT URLs generated by *mod_http_upload*. Otherwise, a SHA-1 hash of the user's bare JID is included instead. The default value is *sha1*.

- **max_size**: *Size*
  This option limits the acceptable file size. Either a number of bytes (larger than zero) or *infinity* must be specified. The default value is *104857600*.

- **name**: *Name*
  A name of the service in the Service Discovery. This will only be displayed by special XMPP clients. The default value is "HTTP File Upload".

- **put_url**: *URL*
  This option specifies the initial part of the PUT URLs used for file uploads. The keyword @HOST@ is replaced with the virtual host name. NOTE: different virtual hosts cannot use the same PUT URL. The default value is "https://@HOST@:5443/upload".

- **rm_on_unregister**: *true | false*
  This option specifies whether files uploaded by a user should be removed when that user is unregistered. The default value is *true*.

- **secret_length**: *Length*
  This option defines the length of the random string included in the GET and PUT URLs generated by *mod_http_upload*. The minimum length is 8 characters, but it is recommended to choose a larger value. The default value is *40*.

- **service_url**
  Deprecated.

- **thumbnail**: *true | false*
  This option specifies whether ejabberd should create thumbnails of uploaded images. If a thumbnail is created, aelement that contains the download and some metadata is returned with the PUT response. The default value is *false*.

- **vcard**: *vCard*
  A custom vCard of the service that will be displayed by some XMPP clients in Service Discovery. The value of *vCard* is a YAML
  map constructed from an XML representation of vCard. Since the representation has no attributes, the mapping is
  straightforward.

  Example:

  ```
  # This XML representation of vCard:
  #   <vCard xmlns='vcard-temp'>
  #     <FN>Conferences</FN>
  #     <ADR>
  #       <WORK/>
  #       <STREET>Elm Street</STREET>
  #     </ADR>
  #   </vCard>
  #
  # is translated to:
  vcard:
    fn: Conferences
    adr:
      -
        work: true
        street: Elm Street
  ```

Example:

```
listen:
  -
    port: 5443
    module: ejabberd_http
    tls: true
    request_handlers:
      /upload: mod_http_upload

modules:
  mod_http_upload:
    docroot: /ejabberd/upload
    put_url: "https://@HOST@:5443/upload"
```

mod_http_upload_quota

This module adds quota support for mod_http_upload.

This module depends on *mod_http_upload*.

Available options:

- **access_hard_quota**: *AccessName*
  This option defines which access rule is used to specify the "hard quota" for the matching JIDs. That rule must yield a positive
  number for any JID that is supposed to have a quota limit. This is the number of megabytes a corresponding user may upload.
  When this threshold is exceeded, ejabberd deletes the oldest files uploaded by that user until their disk usage equals or falls
  below the specified soft quota (see *access_soft_quota*). The default value is *hard_upload_quota*.

- **access_soft_quota**: *AccessName*
  This option defines which access rule is used to specify the "soft quota" for the matching JIDs. That rule must yield a positive
  number of megabytes for any JID that is supposed to have a quota limit. See the description of the *access_hard_quota* option
  for details. The default value is *soft_upload_quota*.

- **max_days**: *Days*
  If a number larger than zero is specified, any files (and directories) older than this number of days are removed from the
  subdirectories of the *docroot* directory, once per day. The default value is *infinity*.

Examples:

Please note that it's not necessary to specify the *access_hard_quota* and *access_soft_quota* options in order to use the quota feature. You can stick to the default names and just specify access rules such as those in this example:

```
shaper_rules:
  soft_upload_quota:
    1000: all # MiB
  hard_upload_quota:
    1100: all # MiB

modules:
  mod_http_upload: {}
  mod_http_upload_quota:
    max_days: 100
```

mod_jidprep

This module allows XMPP clients to ask the server to normalize a JID as per the rules specified in RFC 6122: XMPP Address Format. This might be useful for clients in certain constrained environments, or for testing purposes.

Available options:

- **access**: *AccessName*
  This option defines which access rule will be used to control who is allowed to use this service. The default value is *local*.

mod_last

This module adds support for XEP-0012: Last Activity. 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.

Available options:

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

mod_legacy_auth

The module implements XEP-0078: Non-SASL Authentication.

Note

This type of authentication was obsoleted in 2008 and you unlikely need this module unless you have something like outdated Jabber bots.

The module has no options.

mod_mam

This module implements XEP-0313: Message Archive Management and XEP-0441: Message Archive Management Preferences. Compatible XMPP clients can use it to store their chat history on the server.

Available options:

- **access_preferences**: *AccessName*
  This access rule defines who is allowed to modify the MAM preferences. The default value is *all*.

- **assume_mam_usage**: *true | false*
  This option determines how ejabberd's stream management code (see mod_stream_mgmt) handles unacknowledged messages when the connection is lost. Usually, such messages are either bounced or resent. However, neither is done for messages that were stored in the user's MAM archive if this option is set to *true*. In this case, ejabberd assumes those messages will be retrieved from the archive. The default value is *false*.

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **clear_archive_on_room_destroy**: *true | false*
  Whether to destroy message archive of a room (see mod_muc) when it gets destroyed. The default value is *true*.

- **compress_xml**: *true | false*
  When enabled, new messages added to archives are compressed using a custom compression algorithm. This feature works only with SQL backends. The default value is *false*.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **default**: *always | never | roster*
  The option defines default policy for chat history. When *always* is set every chat message is stored. With *roster* only chat history with contacts from user's roster is stored. And *never* fully disables chat history. Note that a client can change its policy via protocol commands. The default value is *never*.

- **request_activates_archiving**: *true | false*
  If the value is *true*, no messages are stored for a user until their client issue a MAM request, regardless of the value of the *default* option. Once the server received a request, that user's messages are archived as usual. The default value is *false*.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

- **user_mucsub_from_muc_archive**: *true | false*
  When this option is disabled, for each individual subscriber a separate mucsub message is stored. With this option enabled, when a user fetches archive virtual mucsub, messages are generated from muc archives. The default value is *false*.

mod_matrix_gw ●

🏷 added in 24.02

Matrix gateway.

Available options:

- **host**: *Host*
  This option defines the Jabber IDs of the service. If the *host* option is not specified, the Jabber ID will be the hostname of the virtual host with the prefix "*matrix.*". The keyword *@HOST@* is replaced with the real virtual host name.

- **key**: *string()*
  Value of the matrix signing key, in base64.

- **key_name**: *string()*
  Name of the matrix signing key.

- **matrix_domain**: *Domain*
  Specify a domain in the Matrix federation. The keyword *@HOST@* is replaced with the hostname. The default value is *@HOST@*.

- **matrix_id_as_jid**: *true | false*
  If set to *false*, all packets failing to be delivered via an XMPP server-to-server connection will then be routed to the Matrix gateway by translating a Jabber ID *user@matrixdomain.tld* to a Matrix user identifier *@user:matrixdomain.tld*. When set to *true*, messages must be explicitly sent to the matrix gateway service Jabber ID to be routed to a remote Matrix server. In this case, to send a message to Matrix user *@user:matrixdomain.tld*, the client must send a message to the JID *user%matrixdomain.tld@matrix.myxmppdomain.tld*, where *matrix.myxmppdomain.tld* is the JID of the gateway service as set by the *host* option. The default is *false*.

Example:

```
listen:
  -
    port: 8448
    module: ejabberd_http
    tls: true
    request_handlers:
      "/_matrix": mod_matrix_gw

modules:
  mod_matrix_gw:
    key_name: "key1"
    key: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
    matrix_id_as_jid: true
```

mod_metrics

This module sends events to external backend (by now only grapherl is supported). Supported events are:

- sm_register_connection
- sm_remove_connection
- user_send_packet
- user_receive_packet
- s2s_send_packet
- s2s_receive_packet
- register_user
- remove_user
- offline_message

When enabled, every call to these hooks triggers a counter event to be sent to the external backend.

Available options:

- **ip**: *IPv4Address*
  IPv4 address where the backend is located. The default value is *127.0.0.1*.
- **port**: *Port*
  An internet port number at which the backend is listening for incoming connections/packets. The default value is *11111*.

mod_mix

🏷 added in 16.03 and improved in 19.02

This module is an experimental implementation of XEP-0369: Mediated Information eXchange (MIX). It's asserted that the MIX protocol is going to replace the MUC protocol in the future (see mod_muc).

To learn more about how to use that feature, you can refer to our tutorial: Getting started with MIX

The module depends on mod_mam.

Available options:

- **access_create**: *AccessName*
  An access rule to control MIX channels creations. The default value is *all*.
- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.
- **host**
  Deprecated. Use *hosts* instead.
- **hosts**: *[Host, ...]*
  This option defines the Jabber IDs of the service. If the *hosts* option is not specified, the only Jabber ID will be the hostname of the virtual host with the prefix "mix.". The keyword *@HOST@* is replaced with the real virtual host name.
- **name**: *Name*
  A name of the service in the Service Discovery. This will only be displayed by special XMPP clients. The default value is *Channels*.

mod_mix_pam

This module implements XEP-0405: Mediated Information eXchange (MIX): Participant Server Requirements. The module is needed if MIX compatible clients on your server are going to join MIX channels (either on your server or on any remote servers).

Note

mod_mix is not required for this module to work, however, without *mod_mix_pam* the MIX functionality of your local XMPP clients will be impaired.

Available options:

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

mod_mqtt

This module adds support for the MQTT protocol version *3.1.1* and *5.0*. Remember to configure *mod_mqtt* in *modules* and *listen* sections.

Available options:

- **access_publish**: *{TopicFilter: AccessName}*
  Access rules to restrict access to topics for publishers. By default there are no restrictions.

- **access_subscribe**: *{TopicFilter: AccessName}*
  Access rules to restrict access to topics for subscribers. By default there are no restrictions.

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **match_retained_limit**: *pos_integer() | infinity*
  The option limits the number of retained messages returned to a client when it subscribes to some topic filter. The default value is *1000*.

- **max_queue**: *Size*
  Maximum queue size for outgoing packets. The default value is *5000*.

- **max_topic_aliases**: *0..65535*
  The maximum number of aliases a client is able to associate with the topics. The default value is *100*.

- **max_topic_depth**: *Depth*
  The maximum topic depth, i.e. the number of slashes (*/*) in the topic. The default value is *8*.

- **queue_type**: *ram | file*
  Same as top-level queue_type option, but applied to this module only.

- **ram_db_type**: *mnesia*
  Same as top-level default_ram_db option, but applied to this module only.

- **session_expiry**: *timeout()*
  The option specifies how long to wait for an MQTT session resumption. When *0* is set, the session gets destroyed when the underlying client connection is closed. The default value is *5* minutes.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

mod_mqtt_bridge

This module adds ability to synchronize local MQTT topics with data on remote servers It can update topics on remote servers when local user updates local topic, or can subscribe for changes on remote server, and update local copy when remote data is updated. It is available since ejabberd 23.01.

Available options:

- **replication_user**: *JID*
  Identifier of a user that will be assigned as owner of local changes.

- **servers**: *{ServerUrl: {publish: [TopicPairs, subscribe: [TopicPairs], authentication: [AuthInfo]}}}*
  Declaration of data to share, must contain *publish* or *subscribe* or both, and *authentication* section with username/password field or certfile pointing to client certificate. Accepted urls can use schema mqtt, mqtts (mqtt with tls), mqtt5, mqtt5s (both to trigger v5 protocol), ws, wss, ws5, wss5. Certificate authentication can be only used with mqtts, mqtt5s, wss, wss5.

Example:

```
modules:
  mod_mqtt_bridge:
    servers:
      "mqtt://server.com":
        publish:
          "localA": "remoteA" # local changes to 'localA' will be replicated on remote server as 'remoteA'
          "topicB": "topicB"
        subscribe:
          "remoteB": "localB" # changes to 'remoteB' on remote server will be stored as 'localB' on local server
        authentication:
          certfile: "/etc/ejabberd/mqtt_server.pem"
    replication_user: "mqtt@xmpp.server.com"
```

mod_muc

This module provides support for XEP-0045: Multi-User Chat. Users can discover existing rooms, join or create them. Occupants of a room can chat in public or have private chats.

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 XMPP client and register in the MUC service.

It is also possible to register a nickname in a room, so nobody else can use that nickname in that room. If a nick is registered in the MUC service, that nick cannot be registered in any room, and vice versa: a nick that is registered in a room cannot be registered at 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.

Available options:

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

- **access_admin**: *AccessName*
  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 their 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.

- **access_create**: *AccessName*
  To configure who is allowed to create new rooms at the Multi-User Chat service, this option can be used. The default value is *all*, which means everyone is allowed to create rooms.

- **access_mam**: *AccessName*
  To configure who is allowed to modify the *mam* room option. The default value is *all*, which means everyone is allowed to modify that option.

- **access_persistent**: *AccessName*
  To configure who is allowed to modify the *persistent* room option. The default value is *all*, which means everyone is allowed to modify that option.

- **access_register**: *AccessName*
  🏷 improved in 23.10 This option specifies who is allowed to register nickname within the Multi-User Chat service and rooms. The default is *all* for backward compatibility, which means that any user is allowed to register any free nick in the MUC service and in the rooms.

- **cleanup_affiliations_on_start**: *true | false*
  🏷 added in 22.05 Remove affiliations for non-existing local users on startup. The default value is *false*.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **default_room_options**: *Options*

  Define the default room options. Note that the creator of a room can modify the options of his room at any time using an XMPP client with MUC capability. The *Options* are:

- **allow_change_subj**: *true | false*
  Allow occupants to change the subject. The default value is *true*.

- **allow_private_messages_from_visitors**: *anyone | moderators | nobody* Visitors can send private messages to other occupants. The default value is *anyone* which means visitors can send private messages to any occupant.

- **allow_query_users**: *true | false*
  Occupants can send IQ queries to other occupants. The default value is *true*.

- **allow_subscription**: *true | false*
  Allow users to subscribe to room events as described in Multi-User Chat Subscriptions. The default value is *false*.

- **allow_user_invites**: *true | false*
  Allow occupants to send invitations. The default value is *false*.

- **allow_visitor_nickchange**: *true | false*
  Allow visitors to change nickname. The default value is *true*.

- **allow_visitor_status**: *true | false*
  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. The default value is *true*.

- **allow_voice_requests**: *true | false*
  Allow visitors in a moderated room to request voice. The default value is *true*.

- **allowpm**: *anyone | participants | moderators | none*
  Who can send private messages. The default value is *anyone*.

- **anonymous**: *true | false*
  The room is anonymous: occupants don't see the real JIDs of other occupants. Note that the room moderators can always see the real JIDs of the occupants. The default value is *true*.

- **captcha_protected**: *true | false*
  When a user tries to join a room where they have no affiliation (not owner, admin or member), the room requires them to fill a CAPTCHA challenge (see section CAPTCHA in order to accept their join in the room. The default value is *false*.

- **description**: *Room Description*
  Short description of the room. The default value is an empty string.

- **enable_hats**: *true | false*
  Allow extended roles as defined in XEP-0317 Hats. The default value is *false*.

- **lang**: *Language*
  Preferred language for the discussions in the room. The language format should conform to RFC 5646. There is no value by default.

- **logging**: *true | false*
  The public messages are logged using mod_muc_log. The default value is *false*.

- **mam**: *true | false*
  Enable message archiving. Implies mod_mam is enabled. The default value is *false*.

- **max_users**: *Number*
  Maximum number of occupants in the room. The default value is *200*.

- **members_by_default**: *true | false*
  The occupants that enter the room are participants by default, so they have "voice". The default value is *true*.

- **members_only**: *true | false*
  Only members of the room can enter. The default value is *false*.

- **moderated**: *true | false*
  Only occupants with "voice" can send public messages. The default value is *true*.

- **password**: *Password*
  Password of the room. Implies option *password_protected* set to *true*. There is no default value.

- **password_protected**: *true | false*
  The password is required to enter the room. The default value is *false*.

- **persistent**: *true | false*

  The room persists even if the last participant leaves. The default value is *false*.

- **presence_broadcast**: *[moderator | participant | visitor, ...]* List of roles for which presence is broadcasted. The list can contain one or several of: *moderator*, *participant*, *visitor*. The default value is shown in the example below:

  Example:

  ```
  presence_broadcast:
      - moderator
      - participant
      - visitor
  ```

- **public**: *true | false*

  The room is public in the list of the MUC service, so it can be discovered. MUC admins and room participants will see private rooms in Service Discovery if their XMPP client supports this feature. The default value is *true*.

- **public_list**: *true | false*

  The list of participants is public, without requiring to enter the room. The default value is *true*.

- **pubsub**: *PubSub Node*

  XMPP URI of associated Publish/Subscribe node. The default value is an empty string.

- **title**: *Room Title*

  A human-readable title of the room. There is no default value

- **vcard**: *vCard*

  A custom vCard for the room. See the equivalent mod_muc option.The default value is an empty string.

- **voice_request_min_interval**: *Number*

  Minimum interval between voice requests, in seconds. The default value is *1800*.

- **hibernation_timeout**: *infinity | Seconds*

  Timeout before hibernating the room process, expressed in seconds. The default value is *infinity*.

- **history_size**: *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 a non-negative 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 affects all rooms on the service. NOTE: modern XMPP clients rely on Message Archives (XEP-0313), so feel free to disable the history feature if you're only using modern clients and have *mod_mam* module loaded.

- **host**

  Deprecated. Use *hosts* instead.

- **hosts**: *[Host, ...]*

  This option defines the Jabber IDs of the service. If the *hosts* option is not specified, the only Jabber ID will be the hostname of the virtual host with the prefix "conference.". The keyword *@HOST@* is replaced with the real virtual host name.

- **max_captcha_whitelist**: *Number*

  🏷 added in 21.01   This option defines the maximum number of characters that Captcha Whitelist can have when configuring the room. The default value is *infinity*.

- **max_password**: *Number*

  🏷 added in 21.01   This option defines the maximum number of characters that Password can have when configuring the room. The default value is *infinity*.

- **max_room_desc**: *Number*

  This option defines the maximum number of characters that Room Description can have when configuring the room. The default value is *infinity*.

- **max_room_id**: *Number*

  This option defines the maximum number of characters that Room ID can have when creating a new room. The default value is *infinity*.

- **max_room_name**: *Number*

  This option defines the maximum number of characters that Room Name can have when configuring the room. The default value is *infinity*.

- **max_rooms_discoitems**: *Number*
  When there are more rooms than this *Number*, only the non-empty ones are returned in a Service Discovery query. The default value is *100*.

- **max_user_conferences**: *Number*
  This option defines the maximum number of rooms that any given user can join. The default value is *100*. 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.

- **max_users**: *Number*
  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**: *Number*
  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_users_presence**: *Number*
  This option defines after how many users in the room, it is considered overcrowded. When a MUC room is considered overcrowded, presence broadcasts are limited to reduce load, traffic and excessive presence "storm" received by participants. The default value is *1000*.

- **min_message_interval**: *Number*
  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**: *Number*
  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.

- **name**: *string()*
  The value of the service name. This name is only visible in some clients that support XEP-0030: Service Discovery. The default is *Chatrooms*.

- **preload_rooms**: *true | false*
  Whether to load all persistent rooms in memory on startup. If disabled, the room is only loaded on first participant join. The default is *true*. It makes sense to disable room preloading when the number of rooms is high: this will improve server startup time and memory consumption.

- **queue_type**: *ram | file*
  Same as top-level queue_type option, but applied to this module only.

- **ram_db_type**: *mnesia | sql*
  Same as top-level default_ram_db option, but applied to this module only.

- **regexp_room_id**: *string()*
  This option defines the regular expression that a Room ID must satisfy to allow the room creation. The default value is the empty string.

- **room_shaper**: *none | ShaperName*
  This option defines shaper for the MUC rooms. The default value is *none*.

- **user_message_shaper**: *none | ShaperName*
  This option defines shaper for the users messages. The default value is *none*.

- **user_presence_shaper**: *none | ShaperName*
  This option defines shaper for the users presences. The default value is *none*.

- **vcard**: *vCard*
  A custom vCard of the service that will be displayed by some XMPP clients in Service Discovery. The value of *vCard* is a YAML map constructed from an XML representation of vCard. Since the representation has no attributes, the mapping is straightforward.

  Example:

```
# This XML representation of vCard:
#   <vCard xmlns='vcard-temp'>
#     <FN>Conferences</FN>
#     <ADR>
#       <WORK/>
#       <STREET>Elm Street</STREET>
#     </ADR>
#   </vCard>
#
# is translated to:
vcard:
  fn: Conferences
  adr:
    -
      work: true
      street: Elm Street
```

mod_muc_admin

This module provides commands to administer local MUC services and their MUC rooms. It also provides simple WebAdmin pages to view the existing rooms.

This module depends on mod_muc.

Available options:

- **subscribe_room_many_max_users**: *Number*
  added in 22.05 How many users can be subscribed to a room at once using the *subscribe_room_many* command. The default value is *50*.

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 XMPP 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).
- 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.

The module depends on mod_muc.

Available options:

- **access_log**: *AccessName*
  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**: *Path | URL*
  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. Allowed values are either *Path* to local file or an *URL* to a remote file. By default a predefined CSS will be embedded into the HTML page.

- **dirname**: *room_jid | room_name*
  Configure the name of the room directory. If set to *room_jid*, the room directory name will be the full room JID. Otherwise, the room directory name will be only the room name, not including the MUC service name. The default value is *room_jid*.

- **dirtype**: *subdirs | plain*
  The type of the created directories can be specified with this option. If set to *subdirs*, subdirectories are created for each year and month. Otherwise, the names of the log files contain the full date, and there are no subdirectories. The default value is *subdirs*.

- **file_format**: *html | plaintext*
  Define the format of the log files: *html* stores in HTML format, *plaintext* stores in plain text. The default value is *html*.

- **file_permissions**: *{mode: Mode, group: Group}*
  Define the permissions that must be used when creating the log files: the number of the mode, and the numeric id of the group that will own the files. The default value is shown in the example below:

  Example:

  ```
  file_permissions:
    mode: 644
    group: 33
  ```

- **outdir**: *Path*
  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*.

- **spam_prevention**: *true | false*
  If set to *true*, a special attribute is added 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.

- **timezone**: *local | universal*
  The time zone for the logs is configurable with this option. If set to *local*, the local time, as reported to Erlang emulator by the operating system, will be used. Otherwise, UTC time will be used. The default value is *local*.

- **top_link**: *{URL: Text}*
  With this option you can customize the link on the top right corner of each log file. The default value is shown in the example below:

  Example:

  ```
  top_link:
    /: Home
  ```

- **url**: *URL*
  A top level *URL* where a client can access logs of a particular conference. The conference name is appended to the URL if *dirname* option is set to *room_name* or a conference JID is appended to the *URL* otherwise. There is no default value.

mod_muc_occupantid

🏷 added in 23.10

This module implements XEP-0421: Anonymous unique occupant identifiers for MUCs.

When the module is enabled, the feature is enabled in all semi-anonymous rooms.

The module has no options.

mod_muc_rtbl

🏷 added in 23.04

This module implement Real-time blocklists for MUC rooms.

It works by observing remote pubsub node conforming with specification described in https://xmppbl.org/.

Available options:

- **rtbl_node**: *PubsubNodeName*
  Name of pubsub node that should be used to track blocked users. The default value is *muc_bans_sha256*.

- **rtbl_server**: *Domain*
  Domain of xmpp server that serves block list. The default value is *xmppbl.org*

mod_multicast

This module implements a service for XEP-0033: Extended Stanza Addressing.

Available options:

- **access**: *Access*
  The access rule to restrict who can send packets to the multicast service. Default value: *all*.

- **host**
  Deprecated. Use *hosts* instead.

- **hosts**: *[Host, ...]*
  This option defines the Jabber IDs of the service. If the *hosts* option is not specified, the only Jabber ID will be the hostname of the virtual host with the prefix "multicast.". The keyword *@HOST@* is replaced with the real virtual host name. The default value is *multicast.@HOST@*.

- **limits**: *Sender: Stanza: Number*
  Specify a list of custom limits which override the default ones defined in XEP-0033. Limits are defined per sender type and stanza type, where:

- *sender* can be: *local* or *remote*.

- *stanza* can be: *message* or *presence*.

- *number* can be a positive integer or *infinite*.
  Example:

```
# Default values:
local:
  message: 100
  presence: 100
remote:
  message: 20
  presence: 20
```

- **name**
  Service name to provide in the Info query to the Service Discovery. Default is "*Multicast*".

- **vcard**
  vCard element to return when queried. Default value is *undefined*.

Example:

```
# Only admins can send packets to multicast service
access_rules:
```

```
   multicast:
    - allow: admin

# If you want to allow all your users:
access_rules:
  multicast:
    - allow

# This allows both admins and remote users to send packets,
# but does not allow local users
acl:
  allservers:
    server_glob: "*"
access_rules:
  multicast:
    - allow: admin
    - deny: local
    - allow: allservers

modules:
  mod_multicast:
    host: multicast.example.org
    access: multicast
    limits:
      local:
        message: 40
        presence: infinite
      remote:
        message: 150
```

mod_offline

This module implements XEP-0160: Best Practices for Handling Offline Messages and XEP-0013: Flexible Offline Message Retrieval. 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. A user is considered offline if no session presence priority > 0 are currently open.

Note

ejabberdctl has a command to delete expired messages (see chapter Managing an ejabberd server in online documentation.

Available options:

- **access_max_user_messages**: *AccessName*
  This option defines which access rule will be enforced to limit the maximum number of offline messages that a user can have (quota). When a user has too many offline messages, any new messages that they receive are discarded, and a <resource-constraint/> error is returned to the sender. The default value is *max_user_offline_messages*.

- **bounce_groupchat**: *true | false*
  This option is use the disable an optimization that avoids bouncing error messages when groupchat messages could not be stored as offline. It will reduce chat room load, without any drawback in standard use cases. You may change default value only if you have a custom module which uses offline hook after *mod_offline*. This option can be useful for both standard MUC and MucSub, but the bounce is much more likely to happen in the context of MucSub, so it is even more important to have it on large MucSub services. The default value is *false*, meaning the optimization is enabled.

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **store_empty_body**: *true | false | unless_chat_state*
  Whether or not to store messages that lack a <body/> element. The default value is *unless_chat_state*, which tells ejabberd to store messages even if they lack the <body/> element, unless they only contain a chat state notification (as defined in XEP-0085: Chat State Notifications.

- **store_groupchat**: *true | false*
  Whether or not to store groupchat messages. The default value is *false*.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

- **use_mam_for_storage**: *true | false*
  This is an experimental option. Enabling this option, *mod_offline* uses the *mod_mam* archive table instead of its own spool table to retrieve the messages received when the user was offline. This allows client developers to slowly drop XEP-0160 and rely on XEP-0313 instead. It also further reduces the storage required when you enable MucSub. Enabling this option has a known drawback for the moment: most of flexible message retrieval queries don't work (those that allow retrieval/deletion of messages by id), but this specification is not widely used. The default value is *false* to keep former behaviour as default.

Examples:

This example allows power users to have as much as 5000 offline messages, administrators up to 2000, and all the other users up to 100:

```
acl:
  admin:
    user:
      - admin1@localhost
      - admin2@example.org
  poweruser:
    user:
      - bob@example.org
      - jane@example.org

shaper_rules:
  max_user_offline_messages:
    - 5000: poweruser
    - 2000: admin
    - 100

modules:
  ...
  mod_offline:
    access_max_user_messages: max_user_offline_messages
  ...
```

mod_ping

This module implements support for [XEP-0199: XMPP Ping](#) and periodic keepalives. When this module is enabled ejabberd responds correctly to ping requests, as defined by the protocol.

Available options:

- **ping_ack_timeout**: *timeout()*
  How long to wait before deeming that a client has not answered a given server ping request. NOTE: when [mod_stream_mgmt](#) is loaded and stream management is enabled by a client, this value is ignored, and the `ack_timeout` applies instead. The default value is *undefined*.

- **ping_interval**: *timeout()*
  How often to send pings to connected clients, if option *send_pings* is set to *true*. If a client connection does not send or receive any stanza within this interval, a ping request is sent to the client. The default value is *1* minute.

- **send_pings**: *true | false*
  If this option is set to *true*, the server sends pings to connected clients that are not active in a given interval defined in *ping_interval* option. This is useful to keep client connections alive or checking availability. The default value is *false*.

- **timeout_action**: *none | kill*
  What to do when a client does not answer to a server ping request in less than period defined in *ping_ack_timeout* option: *kill* means destroying the underlying connection, *none* means to do nothing. NOTE: when [mod_stream_mgmt](#) is loaded and stream management is enabled by a client, killing the client connection doesn't mean killing the client session - the session will be kept alive in order to give the client a chance to resume it. The default value is *none*.

Example:

```
modules:
  mod_ping:
    send_pings: true
    ping_interval: 4 min
    timeout_action: kill
```

mod_pres_counter

This module detects flood/spam in presence subscriptions traffic. If a user sends or receives more of those stanzas in a given time interval, the exceeding stanzas are silently dropped, and a warning is logged.

Available options:

- **count**: *Number*
  The number of subscription presence stanzas (subscribe, unsubscribe, subscribed, unsubscribed) allowed for any direction (input or output) per time defined in *interval* option. Please note that two users subscribing to each other usually generate 4 stanzas, so the recommended value is *4* or more. The default value is *5*.

- **interval**: *timeout()*
  The time interval. The default value is *1* minute.

Example:

```
modules:
  mod_pres_counter:
    count: 5
    interval: 30 secs
```

mod_privacy

This module implements [XEP-0016: Privacy Lists](#).

Note

Nowadays modern XMPP clients rely on [XEP-0191: Blocking Command](#) which is implemented by *mod_blocking* module. However, you still need *mod_privacy* loaded in order for [mod_blocking](#) to work.

Available options:

- **cache_life_time**: *timeout()*
  Same as top-level [cache_life_time](#) option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level [cache_missed](#) option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level [cache_size](#) option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level [default_db](#) option, but applied to this module only.

- **use_cache**: *true | false*
  Same as top-level [use_cache](#) option, but applied to this module only.

mod_private

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

Using this method, XMPP entities can store private data on the server, retrieve it whenever necessary and share it between multiple connected clients of the same user. The data stored might be anything, as long as it is a valid XML. One typical usage is storing a bookmark of all user's conferences ([XEP-0048: Bookmarks](#)).

It also implements the bookmark conversion described in [XEP-0402: PEP Native Bookmarks](#), see the command [bookmarks_to_pep](#) API.

Available options:

- **cache_life_time**: *timeout()*
  Same as top-level [cache_life_time](#) option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level [cache_missed](#) option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level [cache_size](#) option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level [default_db](#) option, but applied to this module only.

- **use_cache**: *true | false*
  Same as top-level [use_cache](#) option, but applied to this module only.

mod_privilege

This module is an implementation of [XEP-0356: Privileged Entity](#). This extension allows components to have privileged access to other entity data (send messages on behalf of the server or on behalf of a user, get/set user roster, access presence information, etc.). This may be used to write powerful external components, for example implementing an external [PEP](#) or [MAM](#) service.

By default a component does not have any privileged access. It is worth noting that the permissions grant access to the component to a specific data type for all users of the virtual host on which *mod_privilege* is loaded.

Make sure you have a listener configured to connect your component. Check the section about listening ports for more information.

Warning

Security issue: Privileged access gives components access to sensitive data, so permission should be granted carefully, only if you trust a component.

Note

This module is complementary to mod_delegation, but can also be used separately.

Available options:

- **message**: *Options*
  This option defines permissions for messages. By default no permissions are given. The *Options* are:

- **outgoing**: *AccessName*
  The option defines an access rule for sending outgoing messages by the component. The default value is *none*.

- **presence**: *Options*
  This option defines permissions for presences. By default no permissions are given. The *Options* are:

- **managed_entity**: *AccessName*
  An access rule that gives permissions to the component to receive server presences. The default value is *none*.

- **roster**: *AccessName*
  An access rule that gives permissions to the component to receive the presence of both the users and the contacts in their roster. The default value is *none*.

- **roster**: *Options*
  This option defines roster permissions. By default no permissions are given. The *Options* are:

- **both**: *AccessName*
  Sets read/write access to a user's roster. The default value is *none*.

- **get**: *AccessName*
  Sets read access to a user's roster. The default value is *none*.

- **set**: *AccessName*
  Sets write access to a user's roster. The default value is *none*.

Example:

```
modules:
  mod_privilege:
    roster:
      get: all
    presence:
      managed_entity: all
    message:
      outgoing: all
```

mod_proxy65

This module implements XEP-0065: SOCKS5 Bytestreams. It allows ejabberd to act as a file transfer proxy between two XMPP clients.

Available options:

- **access**: *AccessName*
  Defines an access rule for file transfer initiators. The default value is *all*. You may want to restrict access to the users of your server only, in order to avoid abusing your proxy by the users of remote servers.

- **auth_type**: *anonymous | plain*
  SOCKS5 authentication type. The default value is *anonymous*. If set to *plain*, ejabberd will use authentication backend as it would for SASL PLAIN.

- **host**
  Deprecated. Use *hosts* instead.

- **hostname**: *Host*
  Defines a hostname offered by the proxy when establishing a session with clients. This is useful when you run the proxy behind a NAT. The keyword *@HOST@* is replaced with the virtual host name. The default is to use the value of *ip* option. Examples: *proxy.mydomain.org*, *200.150.100.50*.

- **hosts**: *[Host, ...]*
  This option defines the Jabber IDs of the service. If the *hosts* option is not specified, the only Jabber ID will be the hostname of the virtual host with the prefix "proxy.". The keyword *@HOST@* is replaced with the real virtual host name.

- **ip**: *IPAddress*
  This option specifies which network interface to listen for. The default value is an IP address of the service's DNS name, or, if fails, *127.0.0.1*.

- **max_connections**: *pos_integer() | infinity*
  Maximum number of active connections per file transfer initiator. The default value is *infinity*.

- **name**: *Name*
  The value of the service name. This name is only visible in some clients that support [XEP-0030: Service Discovery](#). The default is "SOCKS5 Bytestreams".

- **port**: *1..65535*
  A port number to listen for incoming connections. The default value is *7777*.

- **ram_db_type**: *mnesia | redis | sql*
  Same as top-level [default_ram_db](#) option, but applied to this module only.

- **recbuf**: *Size*
  A size of the buffer for incoming packets. If you define a shaper, set the value of this option to the size of the shaper in order to avoid traffic spikes in file transfers. The default value is *65536* bytes.

- **shaper**: *Shaper*
  This option defines a shaper for the file transfer peers. A shaper with the maximum bandwidth will be selected. The default is *none*, i.e. no shaper.

- **sndbuf**: *Size*
  A size of the buffer for outgoing packets. If you define a shaper, set the value of this option to the size of the shaper in order to avoid traffic spikes in file transfers. The default value is *65536* bytes.

- **vcard**: *vCard*
  A custom vCard of the service that will be displayed by some XMPP clients in Service Discovery. The value of *vCard* is a YAML map constructed from an XML representation of vCard. Since the representation has no attributes, the mapping is straightforward.

Example:

```
acl:
  admin:
    user: admin@example.org
  proxy_users:
    server: example.org

access_rules:
  proxy65_access:
    allow: proxy_users

shaper_rules:
  proxy65_shaper:
```

```
      none: admin
    proxyrate: proxy_users

shaper:
  proxyrate: 10240

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
    recbuf: 10240
    sndbuf: 10240
```

mod_pubsub

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

Available options:

- **access_createnode**: *AccessName*
  This option restricts which users are allowed to create pubsub nodes using *acl* and *access*. By default any account in the local ejabberd server is allowed to create pubsub nodes. The default value is: *all*.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **default_node_config**: *List of Key:Value*
  To override default node configuration, regardless of node plugin. Value is a list of key-value definition. Node configuration still uses default configuration defined by node plugin, and overrides any items by value defined in this configurable list.

- **force_node_config**: *List of Node and the list of its Key:Value*
  Define the configuration for given nodes. The default value is: *[]*.

  Example:

  ```
  force_node_config:
      ## Avoid buggy clients to make their bookmarks public
    storage:bookmarks:
        access_model: whitelist
  ```

- **host**
  Deprecated. Use *hosts* instead.

- **hosts**: *[Host, ...]*
  This option defines the Jabber IDs of the service. If the *hosts* option is not specified, the only Jabber ID will be the hostname of the virtual host with the prefix "pubsub.". The keyword *@HOST@* is replaced with the real virtual host name.

- **ignore_pep_from_offline**: *false | true*
  To specify whether or not we should get last published PEP items from users in our roster which are offline when we connect. Value is *true* or *false*. If not defined, pubsub assumes true so we only get last items of online contacts.

- **last_item_cache**: *false | true*
  To specify whether or not pubsub should cache last items. Value is *true* or *false*. If not defined, pubsub does not cache last items. On systems with not so many nodes, caching last items speeds up pubsub and allows you to raise the user connection rate. The cost is memory usage, as every item is stored in memory.

- **max_item_expire_node**: *timeout() | infinity*
  🏷 added in 21.12   Specify the maximum item epiry time. Default value is: *infinity*.

- **max_items_node**: *non_neg_integer() | infinity*
  Define the maximum number of items that can be stored in a node. Default value is: *1000*.

- **max_nodes_discoitems**: *pos_integer() | infinity*
  The maximum number of nodes to return in a discoitem response. The default value is: *100*.

- **max_subscriptions_node**: *MaxSubs*
  Define the maximum number of subscriptions managed by a node. Default value is no limitation: *undefined*.

- **name**: *Name*
  The value of the service name. This name is only visible in some clients that support XEP-0030: Service Discovery. The default is *vCard User Search*.

- **nodetree**: *Nodetree*
  To specify which nodetree to use. If not defined, the default pubsub nodetree is used: *tree*. Only one nodetree can be used per host, and is shared by all node plugins.

- *tree* nodetree store node configuration and relations on the database. *flat* nodes are stored without any relationship, and *hometree* nodes can have child nodes.

- *virtual* nodetree does not store nodes on database. This saves resources on systems with tons of nodes. If using the *virtual* nodetree, you can only enable those node plugins: *[flat, pep]* or *[flat]*; any other plugins configuration will not work. Also, all nodes will have the default configuration, and this can not be changed. Using *virtual* nodetree requires to start from a clean database, it will not work if you used the default *tree* nodetree before.

- **pep_mapping**: *List of Key:Value*
  In this option you can provide a list of key-value to choose defined node plugins on given PEP namespace. The following example will use *node_tune* instead of *node_pep* for every PEP node with the tune namespace:

Example:

```
modules:
  ...
    mod_pubsub:
      pep_mapping:
        http://jabber.org/protocol/tune: tune
  ...
```

- **plugins**: *[Plugin, ...]*
To specify which pubsub node plugins to use. The first one in the list is used by default. If this option is not defined, the default plugins list is: *[flat]*. PubSub clients can define which plugin to use when creating a node: add *type='plugin-name'* attribute to the *create* stanza element.

- *flat* plugin handles the default behaviour and follows standard XEP-0060 implementation.

- *pep* plugin adds extension to handle Personal Eventing Protocol (XEP-0163) to the PubSub engine. When enabled, PEP is handled automatically.

- **vcard**: *vCard*
A custom vCard of the server that will be displayed by some XMPP clients in Service Discovery. The value of *vCard* is a YAML map constructed from an XML representation of vCard. Since the representation has no attributes, the mapping is straightforward.

Example:

```
# This XML representation of vCard:
#   <vCard xmlns='vcard-temp'>
#     <FN>Conferences</FN>
#     <ADR>
#       <WORK/>
#       <STREET>Elm Street</STREET>
#     </ADR>
#   </vCard>
#
# is translated to:
vcard:
  fn: Conferences
  adr:
    -
      work: true
      street: Elm Street
```

Examples:

Example of configuration that uses flat nodes as default, and allows use of flat, hometree and pep nodes:

```
modules:
  mod_pubsub:
    access_createnode: pubsub_createnode
    max_subscriptions_node: 100
    default_node_config:
      notification_type: normal
      notify_retract: false
      max_items: 4
    plugins:
      - flat
      - pep
```

Using relational database requires using mod_pubsub with db_type *sql*. Only flat, hometree and pep plugins supports SQL. The following example shows previous configuration with SQL usage:

```
modules:
  mod_pubsub:
    db_type: sql
    access_createnode: pubsub_createnode
    ignore_pep_from_offline: true
    last_item_cache: false
    plugins:
      - flat
      - pep
```

mod_push

This module implements the XMPP server's part of the push notification solution specified in XEP-0357: Push Notifications. It does not generate, for example, APNS or FCM notifications directly. Instead, it's designed to work with so-called "app servers" operated by third-party vendors of mobile apps. Those app servers will usually trigger notification delivery to the user's mobile device using platform-dependent backend services such as FCM or APNS.

Available options:

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **include_body**: *true | false | Text*
  If this option is set to *true*, the message text is included with push notifications generated for incoming messages with a body. The option can instead be set to a static *Text*, in which case the specified text will be included in place of the actual message body. This can be useful to signal the app server whether the notification was triggered by a message with body (as opposed to other types of traffic) without leaking actual message contents. The default value is "New message".

- **include_sender**: *true | false*
  If this option is set to *true*, the sender's JID is included with push notifications generated for incoming messages with a body. The default value is *false*.

- **notify_on**: *messages | all*
  added in 23.10 If this option is set to *messages*, notifications are generated only for actual chat messages with a body text (or some encrypted payload). If it's set to *all*, any kind of XMPP stanza will trigger a notification. If unsure, it's strongly recommended to stick to *all*, which is the default value.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

mod_push_keepalive

This module tries to keep the stream management session (see mod_stream_mgmt) of a disconnected mobile client alive if the client enabled push notifications for that session. However, the normal session resumption timeout is restored once a push notification is issued, so the session will be closed if the client doesn't respond to push notifications.

The module depends on mod_push.

Available options:

- **resume_timeout**: *timeout()*
  This option specifies the period of time until the session of a disconnected push client times out. This timeout is only in effect as long as no push notification is issued. Once that happened, the resumption timeout configured for mod_stream_mgmt is restored. The default value is *72* hours.

- **wake_on_start**: *true | false*
  If this option is set to *true*, notifications are generated for **all** registered push clients during server startup. This option should not be enabled on servers with many push clients as it can generate significant load on the involved push services and the server itself. The default value is *false*.

- **wake_on_timeout**: *true | false*
  If this option is set to *true*, a notification is generated shortly before the session would time out as per the *resume_timeout* option. The default value is *true*.

mod_register

This module adds support for XEP-0077: In-Band Registration. This protocol enables end users to use an XMPP 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.

This module reads also the top-level registration_timeout option defined globally for the server, so please check that option documentation too.

Available options:

- **access**: *AccessName*
  Specify rules to restrict what usernames can be registered. If a rule returns *deny* on the requested username, registration of that user name is denied. There are no restrictions by default.

- **access_from**: *AccessName*
  By default, *ejabberd* doesn't allow the client to register new accounts from s2s or existing c2s sessions. You can change it by defining access rule in this option. Use with care: allowing registration from s2s leads to uncontrolled massive accounts creation by rogue users.

- **access_remove**: *AccessName*
  Specify rules to restrict access for user unregistration. By default any user is able to unregister their account.

- **allow_modules**: *all | [Module, ...]*
  🏷 added in 21.12 List of modules that can register accounts, or *all*. The default value is *all*, which is equivalent to something like *[mod_register, mod_register_web]*.

- **captcha_protected**: *true | false*
  Protect registrations with CAPTCHA. The default is *false*.

- **ip_access**: *AccessName*
  Define rules to allow or deny account registration depending on the IP address of the XMPP client. The *AccessName* should be of type *ip*. The default value is *all*.

- **password_strength**: *Entropy*
  This option sets the minimum Shannon entropy for passwords. The value *Entropy* is a number of bits of entropy. The recommended minimum is 32 bits. The default is *0*, i.e. no checks are performed.

- **redirect_url**: *URL*
  This option enables registration redirection as described in XEP-0077: In-Band Registration: Redirection.

- **registration_watchers**: *[JID, ...]*
  This option defines a list of JIDs which will be notified each time a new account is registered.

- **welcome_message**: *{subject: Subject, body: Body}*
  Set a welcome message that is sent to each newly registered account. The message will have subject *Subject* and text *Body*.

mod_register_web

This module provides a web page where users can:

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

This module supports CAPTCHA to register a new account. To enable this feature, configure the top-level captcha_cmd and top-level captcha_url options.

As an example usage, the users of the host *localhost* can visit the page: *https://localhost:5280/register/* It is important to include the last / character in the URL, otherwise the subpages URL will be incorrect.

This module is enabled in *listen → ejabberd_http →* request_handlers, no need to enable in *modules*. The module depends on mod_register where all the configuration is performed.

The module has no options.

Example:

```
listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /register: mod_register_web

modules:
  mod_register: {}
```

mod_roster

This module implements roster management as defined in RFC6121 Section 2. The module also adds support for XEP-0237: Roster Versioning.

Available options:

- **access**: *AccessName*
  This option can be configured to specify rules to restrict roster management. If the rule returns *deny* on the requested user name, that user cannot modify their personal roster, i.e. they cannot add/remove/modify contacts or send presence subscriptions. The default value is *all*, i.e. no restrictions.

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **store_current_id**: *true | false*
  If this option is set to *true*, the current roster version number is stored on the database. If set to *false*, the roster version number is calculated on the fly each time. Enabling this option reduces the load for both ejabberd and the database. This option does not affect the client in any way. This option is only useful if option *versioning* is set to *true*. The default value is *false*. IMPORTANT: if you use mod_shared_roster or mod_shared_roster_ldap, you must set the value of the option to *false*.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

- **versioning**: *true | false*
  Enables/disables Roster Versioning. The default value is *false*.

Example:

```
modules:
  mod_roster:
    versioning: true
    store_current_id: false
```

mod_s2s_dialback

The module adds support for XEP-0220: Server Dialback to provide server identity verification based on DNS.

Warning

DNS-based verification is vulnerable to DNS cache poisoning, so modern servers rely on verification based on PKIX certificates. Thus this module is only recommended for backward compatibility with servers running outdated software or non-TLS servers, or those with invalid certificates (as long as you accept the risks, e.g. you assume that the remote server has an invalid certificate due to poor administration and not because it's compromised).

Available options:

- **access**: *AccessName*
  An access rule that can be used to restrict dialback for some servers. The default value is *all*.

Example:

```
modules:
  mod_s2s_dialback:
    access:
      allow:
        server: legacy.domain.tld
        server: invalid-cert.example.org
      deny: all
```

mod_service_log

This module forwards copies of all stanzas to remote XMPP servers or components. Every stanza is encapsulated into <forwarded/> element as described in XEP-0297: Stanza Forwarding.

Available options:

- **loggers**: *[Domain, ...]*
  A list of servers or connected components to which stanzas will be forwarded.

Example:

```
modules:
  mod_service_log:
    loggers:
      - xmpp-server.tld
      - component.domain.tld
```

mod_shared_roster

This module enables you to create shared roster groups: groups of accounts 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 XMPP server, but the presence will only be available from and to members of the same virtual host where the group is created. It still allows the users to have / add their own contacts, as it does not replace the standard roster. Instead, the shared roster contacts are merged to the relevant users at retrieval time. The standard user rosters thus stay unmodified.

Shared roster groups can be edited via the Web Admin, and some API commands called *srg_**. Each group has a unique name and those parameters:

- Label: Used in the rosters where this group is displayed.

- Description: of the group, which has no effect.

- Members: A list of JIDs of group members, entered one per line in the Web Admin. The special member directive *@all@* represents all the registered users in the virtual host; which is only recommended for a small server with just a few hundred users. The special member directive *@online@* represents the online users in the virtual host. With those two directives, the actual list of members in those shared rosters is generated dynamically at retrieval time.

- Displayed: A list of groups that will be in the rosters of this group's members. A group of other vhost can be identified with *groupid@vhost*.

This module depends on mod_roster. If not enabled, roster queries will return 503 errors.

Available options:

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql*
  Same as top-level default_db option, but applied to this module only.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

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 this one:

```
Name: club_members
Label: 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 lists:

```
First list:
Name: management
Label: Management
Description: Management
Members: manager1@example.org, manager2@example.org
Displayed: management, marketing, sales

Second list:
Name: marketing
Label: Marketing
Description: Marketing
Members: marketeer1@example.org, marketeer2@example.org, marketeer3@example.org
Displayed: management, marketing

Third list:
Name: sales
Label: Sales
Description: Sales
Members: salesman1@example.org, salesman2@example.org, salesman3@example.org
Displayed: management, sales
```

mod_shared_roster_ldap

This module lets the server administrator automatically populate users' rosters (contact lists) with entries based on users and groups defined in an LDAP-based directory.

Note

mod_shared_roster_ldap depends on *mod_roster* being enabled. Roster queries will return *503* errors if *mod_roster* is not enabled.

The module accepts many configuration options. Some of them, if unspecified, default to the values specified for the top level of configuration. This lets you avoid specifying, for example, the bind password in multiple places.

- Filters: *ldap_rfilter*, *ldap_ufilter*, *ldap_gfilter*, *ldap_filter*. These options specify LDAP filters used to query for shared roster information. All of them are run against the ldap_base.

- Attributes: *ldap_groupattr*, *ldap_groupdesc*, *ldap_memberattr*, *ldap_userdesc*, *ldap_useruid*. These options specify the names of the attributes which hold interesting data in the entries returned by running filters specified with the filter options.

- Control parameters: *ldap_auth_check*, *ldap_group_cache_validity*, *ldap_memberattr_format*, *ldap_memberattr_format_re*, *ldap_user_cache_validity*. These parameters control the behaviour of the module.

- Connection parameters: The module also accepts the connection parameters, all of which default to the top-level parameter of the same name, if unspecified. See LDAP Connection section for more information about them.

Check also the Configuration examples section to get details about retrieving the roster, and configuration examples including Flat DIT and Deep DIT.

Available options:

- **cache_life_time**

  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**

  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**

  Same as top-level cache_size option, but applied to this module only.

- **ldap_auth_check**: *true | false*

  Whether the module should check (via the ejabberd authentication subsystem) for existence of each user in the shared LDAP roster. Set to *false* if you want to disable the check. Default value is *true*.

- **ldap_backups**

  Same as top-level ldap_backups option, but applied to this module only.

- **ldap_base**

  Same as top-level ldap_base option, but applied to this module only.

- **ldap_deref_aliases**

  Same as top-level ldap_deref_aliases option, but applied to this module only.

- **ldap_encrypt**

  Same as top-level ldap_encrypt option, but applied to this module only.

- **ldap_filter**

  Additional filter which is AND-ed together with "User Filter" and "Group Filter". For more information check the LDAP Filters section.

- **ldap_gfilter**

  "Group Filter", used when retrieving human-readable name (a.k.a. "Display Name") and the members of a group. See also the parameters *ldap_groupattr*, *ldap_groupdesc* and *ldap_memberattr*. If unspecified, defaults to the top-level parameter of the same name. If that one also is unspecified, then the filter is constructed exactly like "User Filter".

- **ldap_groupattr**

  The name of the attribute that holds the group name, and that is used to differentiate between them. Retrieved from results of the "Roster Filter" and "Group Filter". Defaults to *cn*.

- **ldap_groupdesc**

  The name of the attribute which holds the human-readable group name in the objects you use to represent groups. Retrieved from results of the "Group Filter". Defaults to whatever *ldap_groupattr* is set.

- **ldap_memberattr**

  The name of the attribute which holds the IDs of the members of a group. Retrieved from results of the "Group Filter". Defaults to *memberUid*. The name of the attribute differs depending on the objectClass you use for your group objects, for example: *posixGroup → memberUid*; *groupOfNames → member*; *groupOfUniqueNames → uniqueMember*.

- **ldap_memberattr_format**

  A globbing format for extracting user ID from the value of the attribute named by *ldap_memberattr*. Defaults to *%u*, which means that the whole value is the member ID. If you change it to something different, you may also need to specify the User and Group Filters manually; see section Filters.

- **ldap_memberattr_format_re**

  A regex for extracting user ID from the value of the attribute named by *ldap_memberattr*. Check the LDAP Control Parameters section.

- **ldap_password**

  Same as top-level ldap_password option, but applied to this module only.

- **ldap_port**

  Same as top-level ldap_port option, but applied to this module only.

- **ldap_rfilter**

  So called "Roster Filter". Used to find names of all "shared roster" groups. See also the *ldap_groupattr* parameter. If unspecified, defaults to the top-level parameter of the same name. You must specify it in some place in the configuration, there is no default.

- **ldap_rootdn**
  Same as top-level ldap_rootdn option, but applied to this module only.

- **ldap_servers**
  Same as top-level ldap_servers option, but applied to this module only.

- **ldap_tls_cacertfile**
  Same as top-level ldap_tls_cacertfile option, but applied to this module only.

- **ldap_tls_certfile**
  Same as top-level ldap_tls_certfile option, but applied to this module only.

- **ldap_tls_depth**
  Same as top-level ldap_tls_depth option, but applied to this module only.

- **ldap_tls_verify**
  Same as top-level ldap_tls_verify option, but applied to this module only.

- **ldap_ufilter**
  "User Filter", used for retrieving the human-readable name of roster entries (usually full names of people in the roster). See also the parameters *ldap_userdesc* and *ldap_useruid*. For more information check the LDAP Filters section.

- **ldap_uids**
  Same as top-level ldap_uids option, but applied to this module only.

- **ldap_userdesc**
  The name of the attribute which holds the human-readable user name. Retrieved from results of the "User Filter". Defaults to *cn*.

- **ldap_userjidattr**
  The name of the attribute which is used to map user id to XMPP jid. If not specified (and that is default value of this option), user jid will be created from user id and this module host.

- **ldap_useruid**
  The name of the attribute which holds the ID of a roster item. Value of this attribute in the roster item objects needs to match the ID retrieved from the *ldap_memberattr* attribute of a group object. Retrieved from results of the "User Filter". Defaults to *cn*.

- **use_cache**
  Same as top-level use_cache option, but applied to this module only.

mod_sic

This module adds support for XEP-0279: Server IP Check. This protocol enables a client to discover its external IP address.

Warning

The protocol extension is deferred and seems like there are no clients supporting it, so using this module is not recommended and, furthermore, the module might be removed in the future.

The module has no options.

mod_sip

This module adds SIP proxy/registrar support for the corresponding virtual host.

Note

It is not enough to just load this module. You should also configure listeners and DNS records properly. For details see the section about the ejabberd_sip listen module in the ejabberd Documentation.

Available options:

- **always_record_route**: *true | false*
  Always insert "Record-Route" header into SIP messages. With this approach it is possible to bypass NATs/firewalls a bit more easily. The default value is *true*.

- **flow_timeout_tcp**: *timeout()*
  The option sets a keep-alive timer for SIP outbound TCP connections. The default value is *2* minutes.

- **flow_timeout_udp**: *timeout()*
  The options sets a keep-alive timer for SIP outbound UDP connections. The default value is *29* seconds.

- **record_route**: *URI*
  When the option *always_record_route* is set to *true* or when SIP outbound is utilized, ejabberd inserts "Record-Route" header field with this *URI* into a SIP message. The default is a SIP URI constructed from the virtual host on which the module is loaded.

- **routes**: *[URI, ...]*
  You can set a list of SIP URIs of routes pointing to this SIP proxy server. The default is a list containing a single SIP URI constructed from the virtual host on which the module is loaded.

- **via**: *[URI, ...]*
  A list to construct "Via" headers for inserting them into outgoing SIP messages. This is useful if you're running your SIP proxy in a non-standard network topology. Every *URI* element in the list must be in the form of "scheme://host:port", where "transport" must be *tls*, *tcp*, or *udp*, "host" must be a domain name or an IP address and "port" must be an internet port number. Note that all parts of the *URI* are mandatory (e.g. you cannot omit "port" or "scheme").

Example:

```
modules:
  mod_sip:
    always_record_route: false
    record_route: "sip:example.com;lr"
    routes:
      - "sip:example.com;lr"
      - "sip:sip.example.com;lr"
    flow_timeout_udp: 30 sec
    flow_timeout_tcp: 1 min
    via:
      - tls://sip-tls.example.com:5061
      - tcp://sip-tcp.example.com:5060
      - udp://sip-udp.example.com:5060
```

mod_stats

This module adds support for XEP-0039: Statistics Gathering. This protocol allows you to retrieve the following statistics from your ejabberd server:

- 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).

Note

The protocol extension is deferred and seems like even a few clients that were supporting it are now abandoned. So using this module makes very little sense.

The module has no options.

mod_stream_mgmt

This module adds support for XEP-0198: Stream Management. This protocol allows active management of an XML stream between two XMPP entities, including features for stanza acknowledgments and stream resumption.

Available options:

- **ack_timeout**: *timeout()*
  A time to wait for stanza acknowledgments. Setting it to *infinity* effectively disables the timeout. The default value is *1* minute.

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only. The default value is *48 hours*.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **max_ack_queue**: *Size*
  This option specifies the maximum number of unacknowledged stanzas queued for possible retransmission. When the limit is exceeded, the client session is terminated. The allowed values are positive integers and *infinity*. You should be careful when setting this value as it should not be set too low, otherwise, you could kill sessions in a loop, before they get the chance to finish proper session initiation. It should definitely be set higher that the size of the offline queue (for example at least 3 times the value of the max offline queue and never lower than *1000*). The default value is *5000*.

- **max_resume_timeout**: *timeout()*
  A client may specify the period of time until a session times out if the connection is lost. During this period of time, the client may resume its session. This option limits the period of time a client is permitted to request. It must be set to a timeout equal to or larger than the default *resume_timeout*. By default, it is set to the same value as the *resume_timeout* option.

- **queue_type**: *ram | file*
  Same as top-level queue_type option, but applied to this module only.

- **resend_on_timeout**: *true | false | if_offline*
  If this option is set to *true*, any message stanzas that weren't acknowledged by the client will be resent on session timeout. This behavior might often be desired, but could have unexpected results under certain circumstances. For example, a message that was sent to two resources might get resent to one of them if the other one timed out. Therefore, the default value for this option is *false*, which tells ejabberd to generate an error message instead. As an alternative, the option may be set to *if_offline*. In this case, unacknowledged messages are resent only if no other resource is online when the session times out. Otherwise, error messages are generated.

- **resume_timeout**: *timeout()*
  This option configures the (default) period of time until a session times out if the connection is lost. During this period of time, a client may resume its session. Note that the client may request a different timeout value, see the *max_resume_timeout* option. Setting it to *0* effectively disables session resumption. The default value is *5* minutes.

mod_stun_disco

🏷 added in 20.04

This module allows XMPP clients to discover STUN/TURN services and to obtain temporary credentials for using them as per XEP-0215: External Service Discovery.

Available options:

- **access**: *AccessName*
  This option defines which access rule will be used to control who is allowed to discover STUN/TURN services and to request temporary credentials. The default value is *local*.

- **credentials_lifetime**: *timeout()*
  The lifetime of temporary credentials offered to clients. If ejabberd's built-in TURN service is used, TURN relays allocated using temporary credentials will be terminated shortly after the credentials expired. The default value is *12 hours*. Note that restarting the ejabberd node invalidates any temporary credentials offered before the restart unless a *secret* is specified (see below).

- **offer_local_services**: *true | false*
  This option specifies whether local STUN/TURN services configured as ejabberd listeners should be announced automatically. Note that this will not include TLS-enabled services, which must be configured manually using the *services* option (see below). For non-anonymous TURN services, temporary credentials will be offered to the client. The default value is *true*.

- **secret**: *Text*
  The secret used for generating temporary credentials. If this option isn't specified, a secret will be auto-generated. However, a secret must be specified explicitly if non-anonymous TURN services running on other ejabberd nodes and/or external TURN *services* are configured. Also note that auto-generated secrets are lost when the node is restarted, which invalidates any credentials offered before the restart. Therefore, it's recommended to explicitly specify a secret if clients cache retrieved credentials (for later use) across service restarts.

- **services**: *[Service, ...]*
  The list of services offered to clients. This list can include STUN/TURN services running on any ejabberd node and/or external services. However, if any listed TURN service not running on the local ejabberd node requires authentication, a *secret* must be specified explicitly, and must be shared with that service. This will only work with ejabberd's built-in STUN/TURN server and with external servers that support the same REST API For Access To TURN Services. Unless the *offer_local_services* is set to *false*, the explicitly listed services will be offered in addition to those announced automatically.

- **host**: *Host*
  The hostname or IP address the STUN/TURN service is listening on. For non-TLS services, it's recommended to specify an IP address (to avoid additional DNS lookup latency on the client side). For TLS services, the hostname (or IP address) should match the certificate. Specifying the *host* option is mandatory.

- **port**: *1..65535*
  The port number the STUN/TURN service is listening on. The default port number is 3478 for non-TLS services and 5349 for TLS services.

- **restricted**: *true | false*
  This option determines whether temporary credentials for accessing the service are offered. The default is *false* for STUN/STUNS services and *true* for TURN/TURNS services.

- **transport**: *tcp | udp*
  The transport protocol supported by the service. The default is *udp* for non-TLS services and *tcp* for TLS services.

- **type**: *stun | turn | stuns | turns*
  The type of service. Must be *stun* or *turn* for non-TLS services, *stuns* or *turns* for TLS services. The default type is *stun*.

Example:

```
services:
  -
    host: 203.0.113.3
    port: 3478
    type: stun
    transport: udp
    restricted: false
  -
    host: 203.0.113.3
    port: 3478
    type: turn
    transport: udp
    restricted: true
  -
    host: 2001:db8::3
    port: 3478
    type: stun
    transport: udp
    restricted: false
  -
    host: 2001:db8::3
```

```
        port: 3478
        type: turn
        transport: udp
        restricted: true
      -
        host: server.example.com
        port: 5349
        type: turns
        transport: tcp
        restricted: true
```

mod_time

This module adds support for XEP-0202: Entity Time. In other words, the module reports server's system time.

The module has no options.

mod_vcard

This module allows end users to store and retrieve their vCard, and to retrieve other users vCards, as defined in XEP-0054: vcard-temp. 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.

Available options:

- **allow_return_all**: *true | false*
  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*.

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **db_type**: *mnesia | sql | ldap*
  Same as top-level default_db option, but applied to this module only.

- **host**
  Deprecated. Use *hosts* instead.

- **hosts**: *[Host, ...]*
  This option defines the Jabber IDs of the service. If the *hosts* option is not specified, the only Jabber ID will be the hostname of the virtual host with the prefix "vjud.". The keyword *@HOST@* is replaced with the real virtual host name.

- **matches**: *pos_integer() | infinity*
  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*.

- **name**: *Name*
  The value of the service name. This name is only visible in some clients that support XEP-0030: Service Discovery. The default is *vCard User Search*.

- **search**: *true | false*
  This option specifies whether the search functionality is enabled or not. If disabled, the options *hosts*, *name* and *vcard* will be ignored and the Jabber User Directory service will not appear in the Service Discovery item list. The default value is *false*.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

- **vcard**: *vCard*
  A custom vCard of the server that will be displayed by some XMPP clients in Service Discovery. The value of *vCard* is a YAML map constructed from an XML representation of vCard. Since the representation has no attributes, the mapping is straightforward.

  Example:

```
# This XML representation of vCard:
#
#   <vCard xmlns='vcard-temp'>
#     <FN>Conferences</FN>
#     <ADR>
#       <WORK/>
#       <STREET>Elm Street</STREET>
#     </ADR>
#   </vCard>
#
# is translated to:
#
vcard:
  fn: Conferences
  adr:
    -
      work: true
      street: Elm Street
```

Available options for *ldap* backend:

- **ldap_backups**

  Same as top-level ldap_backups option, but applied to this module only.

- **ldap_base**

  Same as top-level ldap_base option, but applied to this module only.

- **ldap_deref_aliases**

  Same as top-level ldap_deref_aliases option, but applied to this module only.

- **ldap_encrypt**

  Same as top-level ldap_encrypt option, but applied to this module only.

- **ldap_filter**

  Same as top-level ldap_filter option, but applied to this module only.

- **ldap_password**

  Same as top-level ldap_password option, but applied to this module only.

- **ldap_port**

  Same as top-level ldap_port option, but applied to this module only.

- **ldap_rootdn**

  Same as top-level ldap_rootdn option, but applied to this module only.

- **ldap_search_fields**: *{Name: Attribute, ...}*

  This option defines the search form and the LDAP attributes to search within. *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*.

  Examples:

  The default is:

  ```
  User: "%u"
  "Full Name": displayName
  "Given Name": givenName
  "Middle Name": initials
  "Family Name": sn
  Nickname: "%u"
  Birthday: birthDay
  Country: c
  City: l
  Email: mail
  "Organization Name": o
  "Organization Unit": ou
  ```

- **ldap_search_reported**: *{SearchField: VcardField}, ...}*

  This option defines which search fields should be reported. *SearchField* is the name of a search form field which will be automatically translated by using the translation files (see *msgs/*.msg* for available words). *VcardField* is the vCard field name defined in the *ldap_vcard_map* option.

  Examples:

  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
  ```

- **ldap_servers**

  Same as top-level ldap_servers option, but applied to this module only.

- **ldap_tls_cacertfile**

  Same as top-level ldap_tls_cacertfile option, but applied to this module only.

- **ldap_tls_certfile**

  Same as top-level ldap_tls_certfile option, but applied to this module only.

- **ldap_tls_depth**
  Same as top-level ldap_tls_depth option, but applied to this module only.

- **ldap_tls_verify**
  Same as top-level ldap_tls_verify option, but applied to this module only.

- **ldap_uids**
  Same as top-level ldap_uids option, but applied to this module only.

- **ldap_vcard_map**: *{Name: {Pattern, LDAPattributes}, ...}*
  With this option you can set the table that maps LDAP attributes to vCard fields. *Name* is the type name of the vCard as defined in RFC 2426. *Pattern* is a string which contains pattern variables *%u*, *%d* or *%s*. *LDAPattributes* 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.

  Examples:

  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]}
  ```

Available options for *mnesia* backend:

- **search_all_hosts**: *true | false*
  Whether to perform search on all virtual hosts or not. The default value is *true*.

mod_vcard_xupdate

The user's client can store an avatar in the user vCard. The vCard-Based Avatars protocol (XEP-0153) provides a method for clients to inform the contacts what is the avatar hash value. However, simple or small clients may not implement that protocol.

If this module is enabled, all the outgoing client presence stanzas get automatically the avatar hash on behalf of the client. So, the contacts receive the presence stanzas with the *Update Data* described in XEP-0153 as if the client would had inserted it itself. If the client had already included such element in the presence stanza, it is replaced with the element generated by ejabberd.

By enabling this module, each vCard modification produces a hash recalculation, and each presence sent by a client produces hash retrieval and a presence stanza rewrite. For this reason, enabling this module will introduce a computational overhead in servers with clients that change frequently their presence. However, the overhead is significantly reduced by the use of caching, so you probably don't want to set *use_cache* to *false*.

The module depends on mod_vcard.

Note

Nowadays XEP-0153 is used mostly as "read-only", i.e. modern clients don't publish their avatars inside vCards. Thus in the majority of cases the module is only used along with mod_avatar for providing backward compatibility.

Available options:

- **cache_life_time**: *timeout()*
  Same as top-level cache_life_time option, but applied to this module only.

- **cache_missed**: *true | false*
  Same as top-level cache_missed option, but applied to this module only.

- **cache_size**: *pos_integer() | infinity*
  Same as top-level cache_size option, but applied to this module only.

- **use_cache**: *true | false*
  Same as top-level use_cache option, but applied to this module only.

mod_version

This module implements XEP-0092: Software Version. Consequently, it answers ejabberd's version when queried.

Available options:

- **show_os**: *true | false*
  Should the operating system be revealed or not. The default value is *true*.

Advanced

Advanced ejabberd Administration

- Clustering ejabberd

- Managing an ejabberd server

- MQTT Support

- Securing ejabberd

- Troubleshooting ejabberd

- Unattended installation

- XMPP Extensions, and how to support them

Architecture

This section contains information to help your understand ejabberd architecture and will explain how to integrate ejabberd properly into your overall infrastructure.

Overview

ejabberd is a configurable system where modules can be enabled or disabled based on customer requirements. Users can connect not only from a regular PC but also from mobile devices and from the web. User data can be stored internally in Mnesia or in one of the support SQL or NoSQL backend. Users can be totally managed by your own backend through a ReST interface.

ejabberd internal architecture is organised around its router. Most of the other elements are plugins that can be adapted, enhanced or replaced to build a custom solution tailored to your needs.

ejabberd support a core concept of XMPP: Federation. Federation is a mechanism allowing different independent XMPP servers and clusters to communicate with each other.

Here is a high level diagram of ejabberd internal architecture:
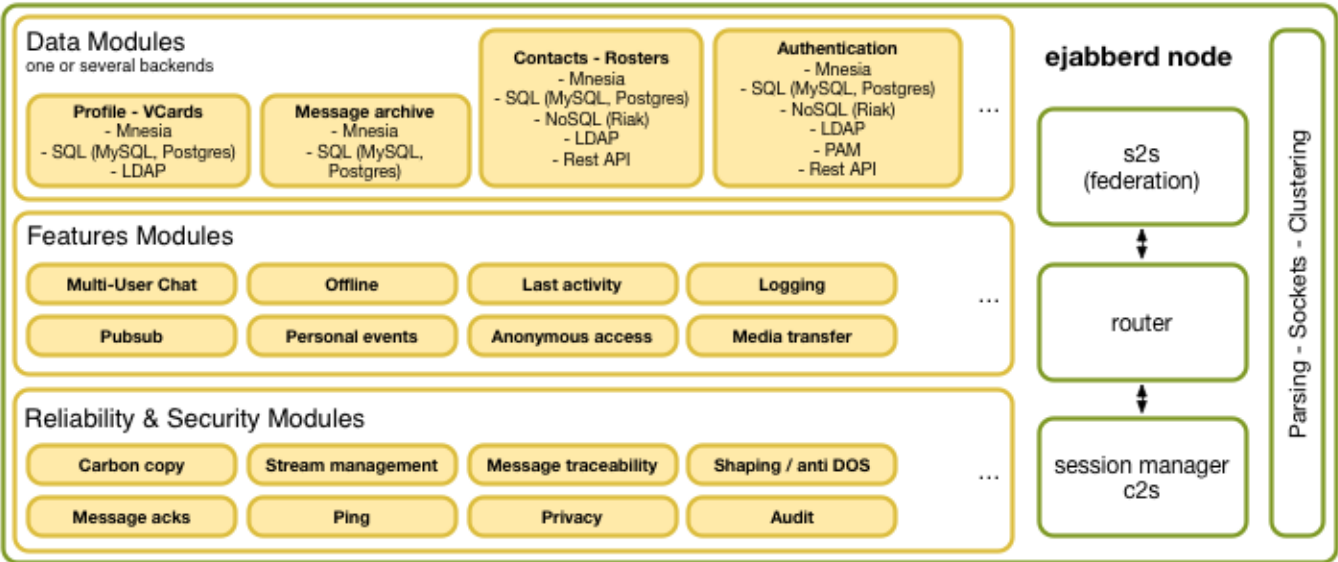
Typical large scale deployments

Here is a diagram for a typical ejabberd large scale deployment. It can scale massively and rely on several back-ends.
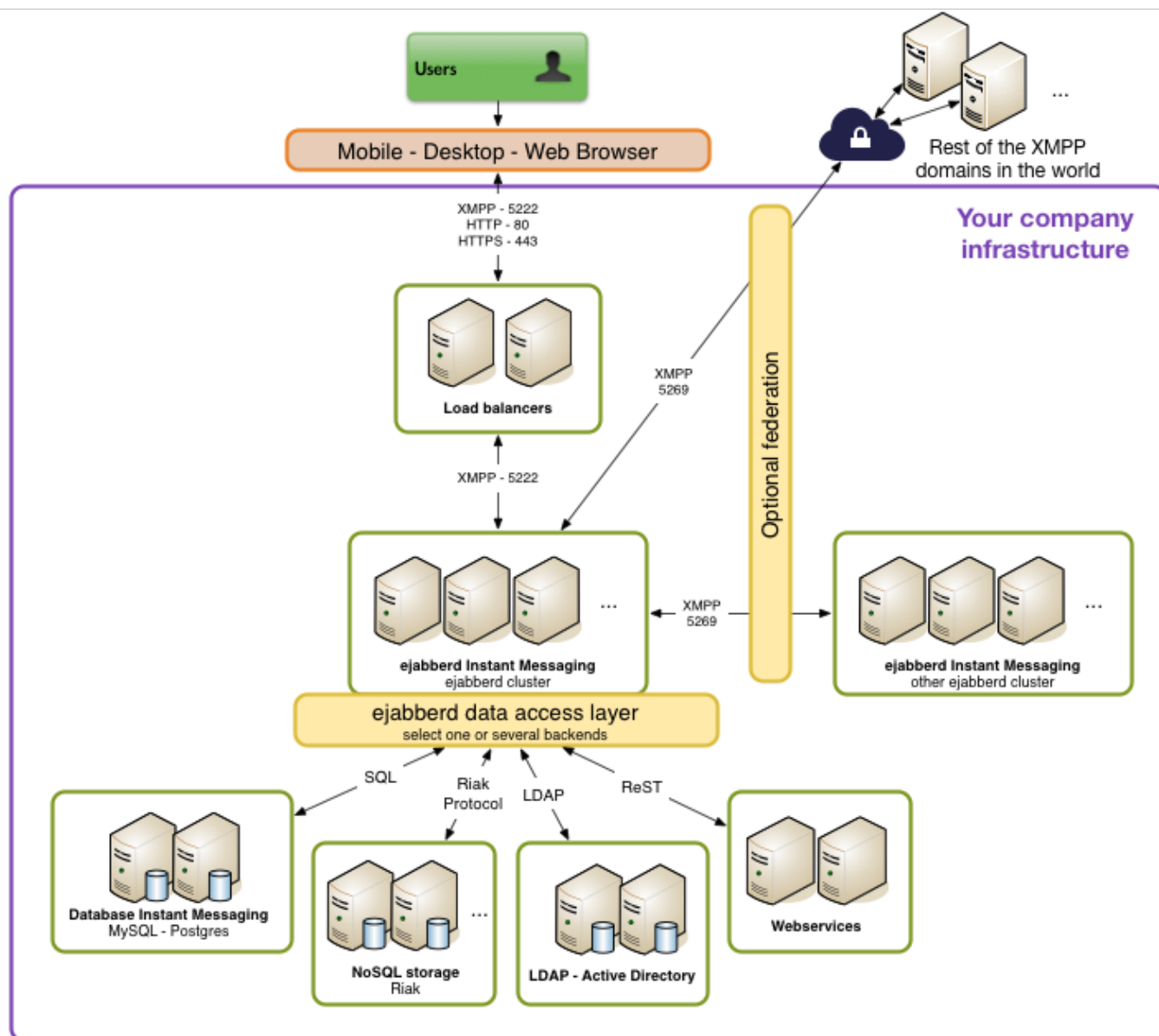
Typical large scale ejabberd deployment

Note that ejabberd ejabberd support a core concept of XMPP: Federation. Federation is a mechanism allowing different independent XMPP servers and clusters to communicate with each other. This is a purely optional layer, but it can help integrate with the rest of the world. It is also sometimes internally by companies to group users in subsidiaries or regions.

Virtual hosting

If you need to manage several small XMPP domains, ejabberd supports virtual hosting. It means you can host as many domain as you want on a single ejabberd deployment.

Instances can be made to be totally independent and invisible for each other if needed (or they can communicate as they would through federation).

Clustering

Purpose

The purpose of ejabberd clustering is to be able to use several servers for a single or small group of large domains, for fault-tolerance and scalability.

Note that you do not necessarily need clustering if you want to run two large domains independently. You may simply want to run two different independent servers.

However, to build reliable service and support large user base, clustering is a must have feature.

How it Works

A XMPP 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

Router

This module is the main router of XMPP 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.

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.

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.

s2s Manager

This module routes packets to other XMPP 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.

Before you get started

Before you start implementing clustering, there are a few things you need to take into account:

- Cluster should be set up in a single data center: The clustering in ejabberd Community Edition relies on low latency networking. While it may work across regions, it is recommended that you run an ejabberd cluster in a single Amazon region.

- Clustering relies on Erlang features and Mnesia shared schemas. Before getting started, it is best to get familiar with the Erlang environment as this guide will heavily reference Erlang terms.

Clustering Setup

Adding a node to a cluster

Suppose you have already configured `ejabberd` on one node named `ejabberd01`. Let's create an additional node (`ejabberd02`) and connect them together.

1. Copy the `/home/ejabberd/.erlang.cookie` file from `ejabberd01` to `ejabberd02`.

   Alternatively you could pass the `-setcookie <value>` option to all `erl` commands below.

1. Make sure your new ejabberd node is properly configured. Usually, you want to have the same `ejabberd.yml` config file on the new node that on the other cluster nodes.

2. Adding a node to the cluster is done by starting a new `ejabberd` node within the same network, and running join_cluster from a cluster node. On the `ejabberd02` node for example, as ejabberd is already started, run the following command as the `ejabberd` daemon user, using the ejabberdctl script:

   ```
   ejabberdctl --no-timeout join_cluster 'ejabberd@ejabberd01'
   ```

   This enables ejabberd's internal replications to be launched across all nodes so new nodes can start receiving messages from other nodes and be registered in the routing tables.

Removing a node from the cluster

To remove a node from the cluster, it just needs to be shut down. There is no specific delay for the cluster to figure out that the node is gone, the node is immediately removed from other router entries. All clients directly connected to the stopped node are disconnected, and should reconnect to other nodes.

If the cluster is used behind a load balancer and the node has been removed from the load balancer, no new clients should be connecting to that node but established connections should be kept, thus allowing to remove a node smoothly, by stopping it after most clients disconnected by themselves. If the node is started again, it's immediately attached back to the cluster until it has been explicitly removed permanently from the cluster.

To permanently remove a running node from the cluster, the leave_cluster command must be run as the `ejabberd` daemon user, from one node of the cluster:

```
ejabberdctl leave_cluster 'ejabberd@ejabberd02'
```

The removed node must be running while calling leave_cluster to make it permanently removed. It's then immediately stopped.

Restarting cluster nodes

Ejabberd Community Server uses mnesia internal database to manage cluster and internode synchronization. As a result, you may restart ejabberd nodes as long as there is at least one running node. If you stop the last running node of a cluster, you MUST restart that node first in order to get a running service back.

Service Load-Balancing

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 attempts to deliver to a local instance of a component. If several local instances are available, one instance is chosen at random. If no instance is available locally, one instance is randomly chosen among the remote component instances.

If you need a different behaviour, you can change the load balancing behaviour with the domain_balancing option.

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 `component_number` option does, making the load balancing algorithm not dynamic, but sticky on a fix number of component instances. Check domain_balancing top-level option documentation for details.

Managing an ejabberd server

ejabberdctl

With the `ejabberdctl` command line administration script you can execute `ejabberdctl commands` (described in the next section, ejabberdctl Commands) and also many general `ejabberd commands` (described in section ejabberd Commands). This means you can start, stop and perform many other administrative tasks in a local or remote `ejabberd` server (by providing the argument `-node NODENAME`).

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

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 specific results. This can be used by other scripts to determine automatically if a command succeeded or failed, for example using: `echo $?`

To restrict what commands can be executed; see API Permissions.

Bash Completion

If you use Bash, you can get Bash completion for ejabberdctl commands names.

Some methods to enable that feature:

- Copy the file `tools/ejabberdctl.bc` to the directory `/etc/bash_completion.d/` (in Debian, Ubuntu, Fedora and maybe others)
- Or add to your `$HOME/.bashrc` a line similar to:

```
source /path/to/ejabberd/tools/ejabberdctl.bc
```

When ejabberd is running in the machine, type `ejabberdctl` in a console and press the `TAB` key.

The first time this is used, the list of commands is extracted from ejabberd and stored in a file in `/tmp/`. The next time, that file is reused for faster responses.

ejabberdctl Commands

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 the `ejabberdctl commands` described below and all the `ejabberd commands` available in that server (see List of ejabberd Commands).

The `ejabberdctl commands` are:

- `help` : Get help about ejabberdctl or any available command. Try `ejabberdctl help help` .

- `status` : Check the status of the `ejabberd` server.

- `stop` : Stop the `ejabberd` server.

- `restart` : Restart the `ejabberd` server.

- `mnesia` : Get information about the Mnesia database.

ejabberd Commands

Please go to the API section.

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.

`EJABBERD_DOC_PATH` : Path to the directory with ejabberd documentation.

`EJABBERD_PID_PATH` : Path to the PID file that ejabberd can create when started.

`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_EPMD_ADDRESS` : IP address where epmd listens for connections (see epmd).

`ERL_INETRC` : Indicates which IP name resolution to use. If using `-sname` , specify either this option or `-kernel inetrc filepath` .

`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:

`-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. If using `-sname` , specify either this option or `ERL_INETRC` .

`-kernel inet_dist_listen_min 4200 inet_dist_listen_min 4210` : Define the first and last ports that `epmd` can listen to (see epmd).

`-kernel inet_dist_use_interface { 127,0,0,1 }` : Define the IP address where this Erlang node listens for other nodes connections (see epmd).

-detached : Starts the Erlang system detached from the system console. Useful for running daemons and background processes.

-noinput : Ensures that the Erlang system never tries to read any input. Useful for running daemons and background 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/' : Specify the Mnesia database directory.

-sasl sasl_error_logger {file, /var/log/ejabberd/erlang.log} : Path to the Erlang/OTP system log file. SASL here means "System Architecture Support Libraries" not "Simple Authentication and Security Layer".

+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.

-hidden : The connections to other nodes are hidden (not published). The result is that this node is not considered part of the cluster. This is important when starting a temporary `ctl` or `debug` 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`).

Web Admin

The `ejabberd` Web Admin allows to administer some parts of `ejabberd` using a web browser: accounts, Shared Roster Groups, manage the Mnesia database, create and restore backups, view server statistics, ...

Basic Setup

1. If not done already, register an account and grant administration rights to it (see Administration Account):

```
acl:
  admin:
    user: admin1@example.org
access_rules:
  configure:
    allow: admin
```

2. Make sure `ejabberd_web_admin` is available in request_handlers of a ejabberd_http listener. If you want to use HTTPS, enable tls. For example:

```
listen:
  -
    port: 5443
    ip: "::"
    module: ejabberd_http
    tls: true
    request_handlers:
      /admin: ejabberd_web_admin
```

3. Open the Web Admin page in your favorite web browser. The exact address depends on your configuration; in this example the address is: `https://example.net:5443/admin/`

4. In the login window provide the **full Jabber ID: admin1@example.org** and password. If the web address hostname is the same that the account JID, you can provide simply the username instead of the full JID: `admin1` .

5. You're good! You can now use the Web Admin.

Advanced Configuration

There are two access rules supported:

- `configure` determines what accounts can access the Web Admin and make changes.

- `webadmin_view` grants only view access: those accounts can browse the Web Admin with read-only access.

Example configurations:

- 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`. The account `reviewer@example.com` can browse that vhost in read-only mode.

```
acl:
  admin:
    user:
      - admin: example.net

host_config:
  example.com:
    acl:
      admin:
        user:
          - admin: example.com
      viewers:
        user:
          - reviewer: example.com

access:
  configure:
    admin: allow
  webadmin_view:
    viewers: allow

hosts:
  - example.org

listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /admin: ejabberd_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://192.168.1.1:5282/admin/`:

```
hosts:
  - example.org
listen:
  -
    port: 5280
    module: ejabberd_http
  -
    ip: "192.168.1.1"
    port: 5282
    module: ejabberd_http
    certfile: "/usr/local/etc/server.pem"
    tls: true
    request_handlers:
      /admin: ejabberd_web_admin
```

Certain pages in the ejabberd Web Admin contain a link to a related section in the ejabberd Installation and Operation Guide. In order to view such links, a copy in HTML format of the Guide must be installed in the system. The file is searched by default in `/share/doc/ejabberd/guide.html`. The directory of the documentation can be specified in the environment variable `EJABBERD_DOC_PATH`. See section Erlang Runtime System.

Ad-hoc Commands

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

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 Erlang Node Name). 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.

You have two ways to use the old Mnesia database in an ejabberd with new node name: put the old node name in `ejabberdctl.cfg` , or convert the database to the new node name.

Those example steps will backup, convert and load the Mnesia database. You need to have either the old Mnesia spool dir or a backup of Mnesia. If you already have a backup file of the old database, you can go directly to step 5. You also need to know the old node name and the new node name. If you don't know them, look for them by executing `ejabberdctl` or in the ejabberd log files.

Before starting, setup some variables:

```
OLDNODE=ejabberd@oldmachine
NEWNODE=ejabberd@newmachine
```

```
OLDFILE=/tmp/old.backup
NEWFILE=/tmp/new.backup
```

1. Start ejabberd enforcing the old node name:

   ```
   ejabberdctl --node $OLDNODE start
   ```

2. Generate a backup file:

   ```
   ejabberdctl --node $OLDNODE backup $OLDFILE
   ```

3. Stop the old node:

   ```
   ejabberdctl --node $OLDNODE stop
   ```

4. Make sure there aren't files in the Mnesia spool dir. For example:

   ```
   mkdir /var/lib/ejabberd/oldfiles
   mv /var/lib/ejabberd/*.* /var/lib/ejabberd/oldfiles/
   ```

5. Start ejabberd. There isn't any need to specify the node name anymore:

   ```
   ejabberdctl start
   ```

6. Convert the backup to new node name using mnesia_change_nodename:

   ```
   ejabberdctl mnesia_change_nodename $OLDNODE $NEWNODE $OLDFILE $NEWFILE
   ```

7. Install the backup file as a fallback using install_fallback:

   ```
   ejabberdctl install_fallback $NEWFILE
   ```

8. Stop ejabberd:

   ```
   ejabberdctl stop
   ```

   You may see an error message in the log files, it's normal, so don't worry:

   ```
   Mnesia(ejabberd@newmachine):
   ** ERROR ** (ignoring core)
   ** FATAL ** A fallback is installed and Mnesia must be restarted.
     Forcing shutdown after mnesia_down from ejabberd@newmachine...
   ```

9. Now you can finally start ejabberd:

   ```
   ejabberdctl start
   ```

10. Check that the information of the old database is available: accounts, rosters... After you finish, remember to delete the temporary backup files from public directories.

Securing ejabberd

Firewall Settings

You need to take the following ports in mind when configuring your firewall. The ports may change depending on your ejabberd configuration. Most of them are TCP ports, except the explicitely mentioned ones:

Port	Description
5222	Jabber/XMPP client connections, plain or STARTTLS
5223	Jabber client connections using the old SSL method
5269	Jabber/XMPP incoming server connections
5280/5443	HTTP/HTTPS for Web Admin and many more (ejabberd_http)
1883/8883	MQTT/MQTTS service (mod_mqtt)
3478/5349	STUN+TURN/STUNS+TURNS service (ejabberd_stun)
3478 UDP	''
49152-65535 range UDP	STUN+TURN service (ejabberd_stun), configure with `turn_min_port` and `turn_max_port`
5060/5061	SIP service (ejabberd_sip)
7777	SOCKS5 file transfer proxy (mod_proxy65)
4369	EPMD (see epmd) listens for Erlang node name requests
random port range	Used by epmd for connections between Erlang nodes, configure with `inet_dist_listen_min` and `inet_dist_listen_max`
5210	Erlang connectivity when `ERL_DIST_PORT` is set, alternative to EPMD

epmd

`epmd (Erlang Port Mapper Daemon)` 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 in such a way that only the programs in your machine can access it, or configure the option `ERL_EPMD_ADDRESS` in the file `ejabberdctl.cfg`.

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, 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 by default are random, but can be configured in the file `ejabberdctl.cfg`. The Erlang command-line parameter used internally is, for example:

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

It is also possible to configure in `ejabberdctl.cfg` the network interface where the Erlang node will listen and accept connections. The Erlang command-line parameter used internally is, for example:

```
erl ... -kernel inet_dist_use_interface "{127,0,0,1}"
```

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 file `$HOME/.erlang.cookie`.

If this file does not exist, it is created immediately with a random cookie in the user `$HOME` path. This means the user running ejabberd must have a `$HOME`, and have write access to that path. So, when you create a new account in your system for running ejabberd, either allow it to have a `$HOME`, or set as `$HOME` a path where ejabberd will have write access. Depending on your setup, examples could be:

```
adduser --home /usr/local/var/lib/ejabberd ejabberd
```

or

```
adduser --home /var/lib/ejabberd ejabberd
```

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.

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.

Securing Sensitive Files

`ejabberd` stores sensitive 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.yml`: 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 in `/var/lib/ejabberd/`: 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/` directory.

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

Troubleshooting ejabberd

Log Files

An `ejabberd` node writes three log files:

- `ejabberd.log` : is the ejabberd service log, with the messages reported by `ejabberd` code
- `error.log` : is the file accumulating error messages from `ejabberd.log`
- `crash.log` : is the Erlang/OTP log, with the crash messages reported by Erlang/OTP using SASL (System Architecture Support Libraries)

The option `loglevel` modifies the verbosity of the file ejabberd.log. The syntax:

`loglevel: Level` : The standard form to set a global log level.

The possible `Level` 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`

By default `ejabberd` rotates the log files when they get grown above a certain size. The exact value is controlled by the log_rotate_size top-level option.

However, you can rotate the log files manually. You can either use an external tool for log rotation and the reopen_log API command to reopen the log files, or the rotate_log API command to perform both steps (please refer to section ejabberd Commands).

The log_rotate_count toplevel option defines the number of rotated files to keep by the reopen_log API command. Every such file has a numeric suffix.

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`

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

Too many db tables

When running ejabberd, the log shows this error:

`** Too many db tables **`

The number of concurrent ETS and Mnesia tables is limited. If this error occurs, it means that you have reached this limit.

For a solution, please read the section about ERL_MAX_ETS_TABLES on the Performance Tuning page.

Upgrade Procedure for ejabberd

This document contains administration procedure for each version upgrade. Only upgrade from version N to N+1 is documented and supported. If you upgrade from an older version than previous one, you have to review all upgrade notes and apply each steps one by one for the possible database changes. You also have to stop your old ejabberd server, and start the new one.

Until release note explicitly state you must restart the server for upgrade, you should be able to run soft upgrade using a cluster. If you don't have cluster, upgrade from older release than previous one, or have explicit note soft upgrade does not work, then you have to fallback to standalone upgrade process.

Generic upgrade process

This is the simplest process, and require service restart.

- read the corresponding upgrade notes
- apply the required changes in database from the upgrade note.
- stop old node
- archive content of mnesia database directory (database, i.e. `/opt/ejabberd-XX.YY/database` , `/usr/local/var/lib/ejabberd` , ...)
- install new version
- extract database archive in new path
- if systemctl is used to manage ejabberd, copy the new service file and reload systemctl:

  ```
  cp ejabberd-21.12/bin/ejabberd.service /etc/systemd/system/
  systemctl daemon-reload
  ```

- start new node

Soft upgrade process

This process needs you to run in cluster, with at least two nodes. In this case, we assume you run node A and B with version N, and will upgrade to version N+1.

- read the corresponding upgrade notes, make sure it does not explicitly states "soft upgrade is not supported".
- apply the required changes in database from the upgrade note.
- make sure node A is running
- run leave_cluster on node B
- stop old node B
- install new version on B's host
- start new node B
- run join_cluster on node B, passing node A as parameter
- make sure both nodes are running and working as expected
- run leave_cluster on node A
- stop old node A
- install new version on A's host
- start new node A
- run join_cluster on node A, passing node B as parameter

Module update process

Instead of upgrading all ejabberd to a brand new version, maybe you just want to update a few modules with bugfixes... in that case you can update only specific modules.

This process is only recommended for bugfixes that involve functional changes, and do not involve structural or memory changes (those ones are usually detected and applied at server start only).

How to do this?

1. Apply the fixes to your source code, compile and reinstall ejabberd, so the new `*.beam` files replace the old ones
2. In the ejabberd Web Admin go to `Nodes` -> your node -> `Update`
3. This will detect what `*.beam` files have changed in the installation
4. Select which modules you want to update now, and click `Update`
5. This will load into memory the corresponding `*.beam` files

   If you prefer to use commands, check update_list + update.

   Notice this does not restart modules or any other tasks. If the fix you plan to apply requires a module restart, you can use this alternative: restart_module.

Note on database schema upgrade

`ejabberd` automatically updates the Mnesia table definitions at startup when needed. If you also use an external database (like MySQL, ...) for storage of some modules, check in the corresponding upgrade notes of the new ejabberd version if you need to update those tables yourself manually.

Specific version upgrade notes

The corresponsing ugprade notes are available in the release notes of each release, and also available in the Archive section:

- Upgrading from ejabberd 23.10 to 24.02
- Upgrading from ejabberd 23.04 to 23.10
- Upgrading from ejabberd 23.01 to 23.04
- Upgrading from ejabberd 22.10 to 23.01
- Upgrading from ejabberd 22.05 to 22.10
- Upgrading from ejabberd 21.12 to 22.05
- Upgrading from ejabberd 21.07 to 21.12
- Upgrading from ejabberd 21.04 to 21.07
- Upgrading from ejabberd 21.01 to 21.04
- Upgrading from ejabberd 19.08 to 20.01
- Upgrading from ejabberd 19.05 to 19.08
- Upgrading from ejabberd 19.02 to 19.05
- Upgrading from ejabberd 18.12 to 19.02
- Upgrading from ejabberd 18.09 to 18.12
- Upgrading from ejabberd 18.06 to 18.09
- Upgrading from ejabberd 18.04 to 18.06
- Upgrading from ejabberd 18.03 to 18.04
- Upgrading from ejabberd 18.01 to 18.03
- Upgrading from ejabberd 17.11 to 18.01
- Upgrading from ejabberd 17.09 to 17.11
- Upgrading from ejabberd ≥17.06 and ≤17.08 to 17.09
- Upgrading from ejabberd 17.03 or 17.04 to 17.06
- Upgrading from ejabberd ≥16.08 and ≤17.01 to 17.03
- Upgrading from ejabberd 16.06 to 16.08
- Upgrading from ejabberd 16.04 to 16.06
- Upgrading from ejabberd 16.03 to 16.04
- Upgrading from ejabberd 16.02 to 16.03
- Upgrading from ejabberd 15.11 to 16.02
- Upgrading from ejabberd 2.1.1x to 16.02

ejabberd and XMPP tutorials

Learning ejabberd and XMPP through videos and hands-on tutorials.

Text tutorials

In the ProcessOne's blog Tutorial tag you will find tutorials about:

- How to setup MariaDB, MQTT, PubSub, STUN/TURN, WebSocket.
- Elixir: Part 1, Part 2, Embed in Phoenix, Embed in Elixir app.
- Useful configuration steps
- Configuration for Office IM
- Configuration for XMPP compliance test
- Using a local development trusted CA on MacOS

In the so-called ejabberd book there are also old archived ejabberd tutorials.

Architecture

- Understanding ejabberd SaaS architecture

  Excerpt from XMPP Academy #1 starting at 1m33s.
- What are ejabberd backends? What backends are available in ejabberd and how do they work?

  Excerpt from XMPP Academy #2 starting at 2m05s.
- ejabberd backends architecture

  Excerpt from XMPP Academy #2 starting at 14m00s.
- What are ejabberd session backends and how to use them to scale?

  Excerpt from XMPP Academy #2 starting at 19m42s.

XMPP on mobile devices (smartphones)

- Mobile XMPP support on ejabberd SaaS and Business Edition: Standby, push and detached modes

  Excerpt from XMPP Academy #1 starting at 16m44s.
- How does Apple and Google Push support work on ejabberd SaaS and ejabberd Business Edition?

  Excerpt from XMPP Academy #3 starting at 1m20s.
- What is the relationship between ejabberd Push support and XEP-0357: Push Notifications?

  Excerpt from XMPP Academy #3 starting at 22m34s.
- What are message carbons and how do they work?

  Excerpt from XMPP Academy #2 starting at 27m30s.
- Demo: learning message carbons with Psi XMPP console

  Excerpt from XMPP Academy #2 starting at 29m51s.

XMPP for the Web

- ejabberd roadmap: announcing OAuth2 support

  Excerpt from XMPP Academy #1 starting at 27m43s.

- What is the impact of Websocket on Web chat performance?

  Excerpt from XMPP Academy #3 starting at 25m02s.

Multi-User Chat

- Why do avatars / carbons not work in MUC rooms? What is special about MUC rooms?

  Excerpt from XMPP Academy #2 starting at 34m15s.

Developer tools and techniques

- What are the typical tools for quick XMPP prototyping?

ejabberd and XMPP server-side implementation

- How does ejabberd internally store messages which are not yet delivered?
- How are privacy lists managed in ejabberd?
- Why do we seem to find duplicate in Message Archive Management backend?

  Excerpt from XMPP Academy #3 starting at 32m20s.

Getting started with MIX

MIX stands for Mediated Information eXchange and defined in MIX-CORE (XEP-0369), MIX-PRESENCE (XEP-0403) and MIX-PAM (XEP-0405). More concretely, ejabberd supports MIX 0.14.1.

It is a work in progress extension for the XMPP protocol to build a group messaging protocol that does not rely on the presence mechanism. It is designed to overcome the limitation of Multi-User Chat (XEP-0045) , in a context where most clients are mobile clients.

To do so, MIX is built on top of PubSub (XEP-0060) and use different nodes per channel to separate event types. There is five nodes to support five different types of event for each MIX channel:

- Messages
- Presence
- Participant list changes
- Subject update
- Conversion configuration changes

This is a work in progress, but this is a very important task and we are happy to provide the very first server implementation of the Mix protocol to get up to speed on that specification.

Here is a short walk through what can already be done.

Also note that the specification can (and will) change significantly before it becomes stable. These examples are based on XEP-0369 v0.1.

Configuration

Configuration is simple:

- Install a recent ejabberd version (19.02 or newer)
- You need to add mod_mix and mod_mix_pam in ejabberd configuration, modules section:

```
modules:
  mod_mix: {}
  mod_mix_pam: {}
```

- Make sure you have PubSub enabled. Default configuration is fine:

```
modules:
  mod_pubsub:
    access_createnode: pubsub_createnode
    plugins:
      - "flat"
      - "pep"
```

- The examples assume you have this virtual host:

```
hosts:
  - shakespeare.example
```

Usage

There is no client supporting MIX yet so here is how it works directly at XMPP stream level.

Here are real-life examples from playing with our MIX implementation:

Creating a MIX Channel

First of all, create a new MIX channel following 7.3.2 Creating a Channel:

```
<iq id='lx09df27'
    to='mix.shakespeare.example'
    type='set'>
  <create channel='coven' xmlns='urn:xmpp:mix:core:0'/>
</iq>
```

Joining a MIX Channel

Now tell your server that you want your account to join that MIX channel, using MIX-PAM: 2.7 Joining a Channel:

```
<iq type='set'
    to='hag66@shakespeare.example'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <client-join xmlns='urn:xmpp:mix:pam:0'
               channel='coven@mix.shakespeare.example'>
    <join xmlns='urn:xmpp:mix:core:0'>
      <nick>third witch</nick>
      <subscribe node='urn:xmpp:mix:nodes:messages'></subscribe>
      <subscribe node='urn:xmpp:mix:nodes:presence'></subscribe>
      <subscribe node='urn:xmpp:mix:nodes:participants'></subscribe>
      <subscribe node='urn:xmpp:mix:nodes:subject'></subscribe>
      <subscribe node='urn:xmpp:mix:nodes:config'></subscribe>
    </join>
  </client-join>
</iq>
```

You receive IQ that confirms success:

```
<iq type="result"
    from="hag66@shakespeare.example"
    to="hag66@shakespeare.example/MacBook-Pro-de-Mickael"
    id="E6E10350-76CF-40C6-B91B-1EA08C332FC7">
  <client-join xmlns='urn:xmpp:mix:pam:0'>
    <join xmlns="urn:xmpp:mix:core:0"
          jid='d79d011852b97adfaad6#coven@mix.shakespeare.example'>
      <nick>third witch</nick>
      <subscribe node="urn:xmpp:mix:nodes:messages"></subscribe>
      <subscribe node="urn:xmpp:mix:nodes:presence"></subscribe>
      <subscribe node="urn:xmpp:mix:nodes:participants"></subscribe>
      <subscribe node="urn:xmpp:mix:nodes:subject"></subscribe>
      <subscribe node="urn:xmpp:mix:nodes:config"></subscribe>
    </join>
  </client-join>
</iq>
```

Subscribers on the participants node for that channel will also receive the new list of participants (so, including ourselves in that case):

```
<message from="coven@mix.shakespeare.example"
         type="headline"
         to="hag66@shakespeare.example/MacBook-Pro-de-Mickael">
  <event xmlns="http://jabber.org/protocol/pubsub#event">
    <items node="urn:xmpp:mix:nodes:participants">
      <item id="3d1766e2bd1b02167104f350f84b0668f850ef92">
        <participant xmlns="urn:xmpp:mix:core:0" jid="hag66@shakespeare.example"></participant>
      </item>
    </items>
  </event>
</message>
```

Setting a nick

You may want to set a nick for this channel (see 7.1.4 Setting a Nick):

```
<iq type='set'
    to='coven@mix.shakespeare.example'
    id='7nve413p'>
  <setnick xmlns='urn:xmpp:mix:core:0'>
    <nick>thirdwitch</nick>
  </setnick>
</iq>
```

Note: Support for MIX nickname registration is not implemented in ejabberd.

Sending and receiving messages

You can now start chatting with your peers, by publishing on the message node (see 7.1.6 Sending a Message):

```
<message to='coven@mix.shakespeare.example'
         id='92vax143g'
         type='groupchat'>
  <body>Harpier cries: 'tis time, 'tis time.</body>
</message>
```

The message is received by all subscribers on the message node on that MIX channel:

```
<message
 to='hag77@shakespeare.example'
 from='coven@mix.shakespeare.example/19be8c262ed618e078b7'
 type='groupchat'
 id='1625493702877370'>
  <mix xmlns='urn:xmpp:mix:core:0'>
    <nick>thirdwitch</nick>
    <jid>hag66@shakespeare.example</jid>
  </mix>
  <body>Harpier cries: &apos;tis time, &apos;tis time.</body>
</message>
```

Querying participants list

A participant can always get list of participants with a PubSub query on node items for the channel (see 6.6 Determining the Participants in a Channel):

```
<iq type='get'
    to='coven@mix.shakespeare.example'
    id='mix4'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='urn:xmpp:mix:nodes:participants'></items>
  </pubsub>
</iq>
```

The channel will reply with list of participants:

```
<iq to='hag66@shakespeare.example/tka1'
    from='coven@mix.shakespeare.example'
    type='result'
    id='kl2fax27'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <items node='urn:xmpp:mix:nodes:participants'>
      <item id='19be8c262ed618e078b7'>
        <participant nick='thirdwitch'
 jid='hag66@shakespeare.example'
 xmlns='urn:xmpp:mix:core:0'/>
      </item>
      <item id='6be2b26cbf4d7108f1fb'>
        <participant jid='hag77@shakespeare.example'
 xmlns='urn:xmpp:mix:core:0'/>
      </item>
    </items>
  </pubsub>
</iq>
```

Caveats

At the moment it is unclear from XEP-0369 example how you match a message you receive to a participant. We are going to improve our implementation in the following way:

1. Add a participant id on the item tag when broadcasting new participant.

2. Add the participant id on the published items.

3. Add the participant id in participants list on the publisher

   Another issue is that the current specification and implementation will have trouble scaling and offer plenty of opportunities for "Denial of Service" attacks. This is something that will change in the future as the specification matures. However, currently, do not deploy or rely on this implementation for large-scale production services. The work is still an experiment to progress on the specifications by offering client developers to give real life feedback on a reference implementation of the current specification.

Conclusion

We are only at the beginning of MIX. However, we are excited to have reached a point where it is already usable in some cases.

It is still missing on administrative tasks, right management, user invitations, relationship with MAM archiving and probably a lot more. And we need consolidations on participants message attribution. However, we want to iterate fast with client developers to prototype implementation changes and have meaningful and real life feedback to improve XEP-0359.

Send us your feedback !

MQTT Support

Benefits

ejabberd is a multiprotocol server that supports MQTT out of the box since ejabberd Business Edition 4.0 and ejabberd Community Server 19.02

There are major benefits in using MQTT service embedded in ejabberd:

1. MQTT service relies on ejabberd infrastructure code, that has been battle tested since 15+ years, like the clustering engine. ejabberd MQTT service has been tested on large scale and can support millions of concurrent connections highly efficiently. ejabberd MQTT is rock-solid and highly scalable.

2. The ejabberd APIs and modules can be reused in MQTT. Authentication, virtual hosting, database backends, ... They both work with XMPP and MQTT. You can also share your security policy, as defined in the configuration file between the two protocols.

3. You can leverage existing skills and plugins you have written for ejabberd, like for example custom authentication.

4. You can deploy services that take advantage of both protocols and have them interoperate with each other, on a single platform, with a single tool.

5. ejabberd supports MQTT 5: it is a state of the art, modern MQTT server. And it also supports MQTT 3.1.1 in case you want to use previous clients.

   In summary:

   - You can switch between XMPP and MQTT as you wish, even use both protocols on the same infrastructure.
   - You will save on infrastructure, given the high-performance of the platform.
   - You get support on solution design for real-time infrastructure and can get help choosing between XMPP and MQTT, from a vendor that has no interest in selling one protocol more than another.

   ejabberd Business Edition offers a different clustering than eCS. Using MQTT with ejabberd Business Edition means you can leverage:

   - The clustering engine of eBE will be used for the MQTT service. It means that you have a more scalable cluster, that supports geoclustering. With geoclustering, you can deploy a single MQTT service across different datacenters, spread in different regions. You can deploy a truly global service.
   - The backend integration that are supported in ejabberd Business Edition will be available in MQTT. You have no need to develop support for new API.

Basic Setup

Maybe you already have MQTT enabled in your ejabberd server, as it comes enabled by default in many distributions.

MQTT support in ejabberd is enabled by adding `mod_mqtt` to the list of `listen` and the list of `modules` like this:

```
listen:
  -
    port: 1883
    module: mod_mqtt
    backlog: 1000

modules:
  mod_mqtt: {}
```

The listener on port 1883 is MQTT over cleartext TCP/IP connection; you can later setup encryption, WebSocket, and encrypted WebSocket.

For available options you can consult the mod_mqtt listener and the mod_mqtt module.

Test Setup

Start ejabberd server and you can connect to ejabberd MQTT service with your preferred MQTT client.

Let's use the clients included with mosquitto, available in Debian, Brew and many others (see mosquitto downloads).

First of all register several accounts and subscribe one to the topic `test/1` with:

```
ejabberdctl register author localhost Pass
ejabberdctl register user1 localhost Pass

mosquitto_sub -u user1@localhost -P Pass -t "test/1" -d -v

Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: test/1, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
```

Then go to another terminal or window and publish something on that topic:

```
mosquitto_pub -u author@localhost -P Pass -t "test/1" -d -m "ABC"

Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q0, r0, m1, 'test/1', ... (3 bytes))
Client (null) sending DISCONNECT
```

You will see the message received and displayed in the `mosquitto_sub` window:

```
Client (null) received PUBLISH (d0, q0, r0, m0, 'test/1', ... (3 bytes))
test/1 ABC
```

Access Control

The mod_mqtt module provides two options for access control:

- `access_subscribe` to restrict access for subscribers,
- and `access_publish` to restrict access for publishers.

Both options accept mapping `filter: rule` where `filter` is an MQTT topic filter and `rule` is the standard ejabberd Access Rule.

As an example, let's say only `author@localhost` is allowed to publish to topic "/test/1/" and its subtopics, while only `user1@localhost` is allowed to subscribe to this topic and its subtopics, and nobody else can publish or subscribe to anything else. The configuration will look something like this:

```
acl:
  publisher:
    user: author@localhost
  subscriber:
    user: user1@localhost

modules:
  mod_mqtt:
    access_publish:
      "test/1/#":
        - allow: publisher
        - deny
      "#":
        - deny
    access_subscribe:
      "test/1/#":
        - allow: subscriber
        - deny
      "#":
        - deny
```

Encryption

Self-Signed Certificate

If you have already setup encryption in ejabberd, you can bypass this step.

If you want to use TLS, you may want to create a self-signed certificate (at least to get started). The following page is a nice guide: Mosquitto SSL Configuration -MQTT TLS Security.

Here is a summary of the steps, adapted for ejabberd MQTT:

```
openssl genrsa -des3 -out ca.key 4096
openssl req -new -x509 -days 1826 -key ca.key -out ca.crt
openssl genrsa -out server.key 4096
openssl req -new -out server.csr -key server.key
openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 360
cat server.crt server.key > mqtt.pem
```

Now copy `mqtt.pem` to the path with ejabberd configuration files, and configure accordingly:

```
certfiles:
  - "/etc/ejabberd/mqtt.pem"
```

Configure Encryption

Add a new listener with `tls` option in the port number 8883 (the standard for encrypted MQTT):

```
listen:
  -
    port: 1883
    module: mod_mqtt
    backlog: 1000
  -
    port: 8883
    module: mod_mqtt
    backlog: 1000
    tls: true
```

The listener on port 1883 is MQTT over cleartext TCP/IP connection. The listener on port 8883 is MQTT over TLS. You can enable both or only one of them depending on your needs.

Test Encryption

You can repeat the commands from previous test, appending `-p 8883` to use the encrypted port. If you are using a self-signed certificate as explained previously, you will also have to append `--cafile server.crt`. For example:

```
mosquitto_sub -u user1@localhost -P Pass -t "test/1" -d -v -p 8883 --cafile server.crt
```

WebSocket

Setup WS

Add `mod_mqtt` as a request_handler on the ejabberd_http listener:

```
listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /mqtt: mod_mqtt
```

This configuration maps the path `/mqtt` to the MQTT WebSocket handler on the main ejabberd HTTP listener.

You can enable listeners independently, for example enable only the WebSocket listener and not the TCP/IP ones.

Test WS

Our beloved mosquitto client does not support MQTT over WebSocket, so you may have to find some capable MQTT client. For example, in MQTTX, setup in the login window:

- Host: `ws:// localhost`
- Port: 5280
- Path: `/mqtt`

If you need an example on how to use MQTTJS library, you can check our small example project: mqttjs-demo

Encrypted WS

To enable encryption on WebSocket, enable `tls` like this:

```
listen:
  -
    port: 5281
    ip: "::"
    module: ejabberd_http
    tls: true
    request_handlers:
      /mqtt: mod_mqtt
```

For testing this in the MQTTX client:

- Host: `wss:// localhost`
- Port: 5281
- Path: `/mqtt`
- SSL/TLS: true
- Certificate: CA signed server
- If you used a self-signed certificate, you will have to disable SSL Secure

Setting vCards / Avatars for MUC rooms

ejabberd supports the ability to set vCard for MUC rooms. One of the most common use case is to be able to define an avatar for your own MUC room.

How does it work?

To be allowed to set vCard for a given room, you need to be owner of that room.

To set up vCard avatar for your MUC room, you first need to make sure you convert your avatar image to base64 encoding, so that you can pass it on XMPP stream.

If you want to convert it manually from command line, you can use `base64` tool. For example:

```
base64 muc_logo.png > muc_logo.b64
```

However, when coding the client, you can more likely directly do the proper image base64 encoding in your code.

Setting up MUC vCard

To set the MUC vCard, you can send a vcard-temp set request, as defined in [XEP-0054: vcard-temp](), but directly addressed to your MUC room. For example, assuming my room id is `test@conference.localhost`:

```
<iq id='set1'
    type='set'
    to='test@conference.localhost'>
<vCard xmlns='vcard-temp'>
    <PHOTO>
      <TYPE>image/png</TYPE>
      <BINVAL>
iVBORw0KGgoAAAANSUhEUgAAAEAAAABACAYAAACqaXHeAAAABGdBTUEAALGPC/
xhBQAAACBjSFJNAAB6JgAAgIQAAPoAAACA6AAAdTAAAOpgAAA6mAAAF3CcuLE8AAAACXBIWXMAAAsTAAALEwEAmpwYAAAB1WlUWHRYTUw6Y29tLmFkb2JlLnhtcAAAAAAAPHg6eG1wbWV0YSB4bWxuczp4PSJhZG
nY3m0022bxIyItAIBBArZRUUbCoKCgVnWJxxlataO1rlKl2amec1urUdmrp2KE6nXFgGIWpQGmlzCDaoTDUBwoMVcFCeOQFhDwhCXlsNrub3fT7z7l3H8kmm4RUWsNJ7uvcc//zf//r/
OectfSxYBwX6zjGrqBfEcDltID/Be+7rBZgsVgup/yvuIBI4LJawGVX//gUQB8iY0/C56EFlWj0Szfi+7+Znow8ThAi5N84GRA1Hek3MvRY/
huJkEg4Eq90GAuwtItVbypFayosCM14bIFo0LqtxTK0ZweDfvZr5WHDmAogGAwq3q3W2AwEg33weHrQ6fYgM90Fh80uzHGgEARILKBh840E2dcXVIBMwcm1t7cbXn8LvN4L8PnOwx+4yLo2+PwNSHfNx8QJi2C1J
g7j5AsJsQCOhW0pzyx6T8LcjJukWBF6sZtQA0cA4jVvEpra229k5UVdfhyLEaHPi0BusPngM+7BR98GjEzKVzsGX1A5h7bWkUYPWga3i2INDrRlPLj9UQJcaUkHA1BXEXUlPmIS31GqQ6i2GzJYU+c3fX4nzrbrS
IGr8tQV+wCn18nlp0EBMy5+meDPcbsQuYGje1HQgEUVFVi/
f3H8Xf9hzFrjfq2IGXhwNZC11o9VP0B9x4ee1teORbS5DmSoliQD0YJ90U07sqcLK6lIIt4RsfgsFapT2JKzYCSU56FJkZy5Hhmou2jkO0lhXopablndVazO8cdC0RvIXXOvXttKL9yM66Ubmc2IRpNSMSgPi4aF
ugzPrFqOqVPyVW9CwzZIgNTsxDoTNFqJuEm9FIHYbC7e+wiogfc5vE/g/SnjY0oFoqgS+P2VyMpcrcDrl2HNG43jCyDS5Ls9Xry54z2sfGYXcLoHhYsy4aAaznl6UdsTwLSUBFTWUAuVXry++T7cf+
+tNFcHtaQtxxSg2flwr6JLEbTWO4UhAiHrFksGr208JMZIEZOXvopooZWwJ85FUf531BvTutRDxGnIIGiajDBeW9eEF9Zsw/rfHwZumIDS0hRUU909dL4JNPf0RAsqd1/
A9cvz8cet92Fe2WzVjbiKTWx1BIU2pfFGfaPNN1wloC+GH3kn/Fosibzz8J4RP+9ljiDZhlBi8zCoAEzw4kv/Lq/Csic3onZPK2bekYcL1PapDr/qPD/
ZhgY+t7zfgmdfXIhVjy1Dbo74rfjnyMHLdzqp4Y1WvVQNr6j2Wey3kdF/FlwperTRlhObREwBRIL/9LNTKFuxjlYWxKw7J+CEAZyiRq7DioaabnFHbH/
7Ydxz5wKl7cFMXmsoPiplAbH5HbJWU25krMqjC5xATe1qTC9+Hon29EGtIKZdmIyeZLAre2i9BGHMmJGKE+0+wa00k51oRVNjD+5dXIhj//wRlt/
1VQVeRXm6jI7CYX5NoYRr4tzFl9MQBEQIhdjvXIPqs6sRCHrJj1UJof9HAwQgjIrPt7R1YNXzm4EzPpQWOVFBzUvSozyRp2QbOTzhw5T8dMwunaLoKn/vF+WFnqTAZgAMSGoWp6g+4rSJ/7qZ2eMsXGx/
EbX1m1RzLYRo6lECEM2bjG7Y8g/s2VyD2fPTlb9bCZhDri7E3uILIp2Jzh9+fRgHPy430tCvhY6ANoGL4BqbWrD+T++g+YIOXCKYwcolKV9zQm37eTTBbs9A04XH0HrxU6O7OAKQVuL3Tz/
xPjIXZuMsI73klP3Z7Q70IZNuIGXb2weZiQWU80Qq5i+g5ejodGPnrv248aGX8b2VHD6HUUJBcBhtYzfRc5O+vjbG0WwVS+ubXlETpP6uELIAU/
tiolt3fES6QWRxeHNzGBustNIK7Del4aUXjuDY8RrVLMFmU9e2i534+56D+PaTr+KepW/gzLvtmPd1MnPp6h2MnX71phAqGZtK0NW9Aa3tH/drI9mEUczAV326Hqs3HOVY70KbT/w1NscCpIN5/
gyXHRVBH3bsOoRZjAVi6h9wXvD69kPY++fTQKETVy3NRnlzDy4yYZLx+XMt5FPsV/htbdvJmeAC3ouShBGLFoAJXhg7cqwaqOpAaUkeTnXKwoER+ORlRFE4aOK1BIXr07HunXLUNa3H2v3ngH3NQEkapi/
ORReHz4teCtJD11B/EUT+67da2pJKSy7m9ryEbs8qpDiLqAhtISEXEKAyhH1WfpZsyUIFL/r7IdlkDsRoy3S4uxdr15wAuvwovSMfhZOTUcn40SgNVIltSbGIjzYPiEVL1/VQkdmMU0F09wi+cFEuIDiFPQ9z/
eM154G8RPQwyKnKcNuYd2JaXoYJO0eJgpvTqOg+ZTnSWFvP8IGbHaggaD6MyVXS5iSl1B5vQxRFHQMMCXi9PtS1MLNLt8HHYUwkIOehinpPjH7enKGZq0KpCOx43w50d+RCG5yWvJFAzuSM50CvHoZFOcJhyAX4p
+k3edXxlAECoB40kc0nh32iQYM3UeUCyQm2nHtrEls4IOdbnDZy5ixkEiNc4LE4T85eXIULG0BDAhmbj7nmmlsYEebt5dWED8IRlEb64exMARFI10tjCY7VsKZLAqWoqUbZQFSPWvGZHzju9PQfLwbeUkU2aiioV
i+mBOlPaFahQumb5KLEhxJuGZJ+5m1gBUM7UVVxDriGcJ8j6FNFyMHebBJUMk8hkJFgZW0/
OGBqRsjp9YLFyKYgZHqjzSeCTzGE7RsctiuQq+3gruBf6U65RLjA9FMGEkUQKQFip9pd9fM3sa3tr5Te7qnIed7TPIvZpAGGRiXYS0m4LqZDJlHpIcqqyytw/
8H6YLkJC05aIG0CNUeXTw4OZL3CJJHC3WMovz/
3JuwC5E8aSfcG1C5jcyK4yGrFPhCKKmAKTq7jvnY8PmDjzywF+BBRORyilV1yBzBFFygOBR6yWvtDlRnkiEAqifQDOo9uAkJ006ARHqQxQyKmYrI7HF0h4SmujNahWWJbfvX+StdCirWrMJ/
jhsCXmYPvk15je5McELhUG3xvRylpbmlm178eAvdyEpPUElkj7pJ6KkE317ux9fKU7FKz+7l8EmjTuzvWpFyGwWFK0QUUlxIZK4WaKsSeXjZgu5CmEL/
L2d6HKf5p2SgGLeZnVwUaMSdY3LSEe0KILgcrQyZ5MhFwVWxO/
LuRRWhulTtnIzdfqg4PlxeEFEHiKLMCvLW7LCc8ei6zBl3Xs4w0WNrHQ7WsWWI4pDVNUdQLrTji9z59eZzHR6iBIbvHygtWhPcHHI+tIAClbuCJ9rlOpIAQgvifyykOJr5G8ByjmS/
YA7wc+RjwIDvNCNXQa4gNlMmDT38d7ZfRBnuNE5aVEW6mRxo1/xsy1Fr1ygmz+AEAHIjpEIUYnKkJcoXA2rAzQfSVBbnZ7BSb08y0p1AoczjxKRthTOWvlkQTH78FJZNYpI/sTXUJh/
PxXnHFLzqjFPgwpAXEC2tCqqz2HlC7uBualq810wGlyYFovc5AS0pYpfWkJCk28F7GiK/
o5xo18JB7BU0s4jQI8CLjylpvwQhbmPI80wnFgBrx859RhTADIUCgBZH1izdidwvAfTFmdyL5BRWUAZGpWdIRkhTvyrXdbJkTg7M9SHYRSh57G6Ud33tRM4+yQfTufD/
LnLI9z7564U44SU4YKXtgMEoIOfHio2vbkXr/7uMGYsmYgKrg+qQsAFBJ7CYbGijsNSuRtPPVuGMw3t2H7kQmhOoRuP7VmA+cgG52zISHsOEzKWUuNzlLlLT/
L7AbGSsKXE7z9qFBDNmxsjsqR915KNyL45U6320CW54TDm4zB4up5j83H+AmNeJrb94mtquPR6/
TjFX4rIXCLFGZ1vx2dj6BZm0HR76rjB8RED8TzGmaIQUBGMlJEAN3sMCcAc9uTF3g8+we0Pb8LUEieS+IMHWR0ONHF8r5LAwyRpWT4eX1GGZUvmYfKkXJNW6GoyHKoYk5vI4KMJXgpwkyUlAJPhHvr81u3v8rc8G
k+lYe4AqVbkMJfdU3EbdcX44ayGcwSpyJ3ot4Cl29N1kw6ow1+JlNDXU3QEnDHoh8Lzb5PCLVzC2v72x9i977jmFmcw2QmlYcL+bmZPLKQk5NJv5NEI8xepMuEa/+/7iiAPgqAS9vUvp9jt/
ykZahfdCiNU+sitLHQwOUWV5QL9GfGBKvqBbDR4IsA3MQaCoICtn/5IgHtj818DuUB4wGsCTryqjOeyJpxdn9FAONM4QPgXrGAASIZZxVXLGCcKXwA3CsWMEAk46ziP3wnyrgPINtbAAAAAElFTkSuQmCC
      </BINVAL>
    </PHOTO>
</vCard>
</iq>
```

Please, note that you have to set the mime type of the image properly to help the client displaying it.

You can of course add other fields to the vCard if needed.

After that IQ `set` stanza, the server will reply with success:

```
<iq from="test@conference.localhost"
    type="result"
    to="owner@localhost/r"
    id="set1">
 <vCard xmlns="vcard-temp"/>
</iq>
```

The MUC room also broadcasts a notification about non-privacy related configuration change to users that are currently in the room:

```
<message from="test@conference.localhost"
         type="groupchat"
         to="owner@localhost/r"
         id="17095969463368094420">
 <x xmlns="http://jabber.org/protocol/muc#user">
  <status code="104"/>
 </x>
</message>
```

Retrieving a MUC room vCard

Any user can retrieve the MUC vCard but sending a vcard-temp get IQ to the room itself:

```
<iq to='test@conference.localhost'
    id='get1'
    type='get'>
 <vCard xmlns='vcard-temp'/>
</iq>
```

Server will reply by sending back the vCard:

```
<iq from="test@conference.localhost"
    type="result"
    to="user@localhost/r"
    id="get1">
 <vCard xmlns="vcard-temp">
   <PHOTO>
     <TYPE>image/png</TYPE>
     <BINVAL>
```

iVBORw0KGgoAAAANSUhEUgAAAEAAAABACAYAAACqaXHeAAAABGdBTUEAALGPC/
xhBQAAACBjSFJNAAB6JgAAgIQAAPoAAACA6AAAdTAAAOpgAAA6mAAAF3CculE8AAAACXBIWXMAAAsTAAALEwEAmpwYAAAB1WlUWHRYTUw6Y29tLmFkb2JlLnhtcAAAAAAPHg6eG1wbWV0YSB4bWxuczp4PSJhZG
nY3m0022bxIyItAIBBArZRUUbCoKCgVnWJxxlataO1rlKl2amec1urUdmrp2KE6nXFgGIWpQGmlzCDaoTDUBwoMVcFCeOQFhDwhCXlsNrub3fT7z7l3H8kmm4RUWsNJ7uvcc//zf//r/
OectfSxYBwX6zjGrqBfEcDltID/Be+7rBZgsVgup/yvuIBI4LJawGVX//gUQB8iY0/C56EFlWj0Szfi+7+Znow8ThAi5N84GRA1Hek3MvRY/
huJkEg4Eq90GAuwtItVbypFayosCM14bIFoOLqtxTK0ZweDfvZr5WHDmAogGAwq3q3W2AwEg33weHrQ6fYgM90Fh8OuzHGgEARILKBh840E2dcXVIBMwcm1t7cbXn8LvN4L8PnOwx+4yLo2+PwNSHfNx8QJi2C1J
g7j5AsJsQCOhW0pzyx6T8LcjJukWBF6sZtQA0cA4jVvEpra229k5UVdfhyLEaHPi0BusPngM+7BR98GjEzKVzsGX1A5h7bWkUYPWga3i2INDrRlPLj9UQJcaUkHA1BXEXUlPmIS31GqQ6i2GzJYU+c3fX4nzrbrS
IGr8tQV+wCn18nlp0EBMy5+meDPcbsQuYGje1HQgEUVFVi/
f3H8Xf9hzFrjfq2IGXhwNZC11o9VP0B9x4ee1teORbS5DmSoliQD0YJ9OUO7sqcLK6lIIt4RsfgsFapT2JKzYCSU56FJkZy5Hhmou2jkO0lhXopablndVazO8cdC0RvIXXOvXttKL9yM66Ubmc2IRpNSMSgPi4aF
ugzPrFqOqVPyVW9CwzZIgNTsxDoTNFqJuEm9FIHYbC7e+wiogfc5vE/g/SnjY0oFoqgS+P2VyMpcrcDrl2HNG43jCyDS5Ls9Xry54z2sfGYXcLoHhYsy4aAaznl6UdsTwLSUBFTWUAuVXry++T7cf+
+tNFcHtaQtxxSg2flwr6JLEbTWO4UhAiHrFksGr208JMZIEZOXvopooZWwJ85FUf531BvTutRDxGnIIGiajDBeW9eEF9Zsw/rfHwZumIDS0hRUU9O9dL4JNPf0RAsqd1/
A9cvz8cet92Fe2WzVjbiKTWx1BIU2pfFGfaPNN1wloC+GH3kn/Fosibzz8J4RP+9ljiDZhlBi8zCoAEzw4kv/Lq/Csic3onZPK2bekYcL1PapDr/qPD/
ZhgY+t7zfgmdfXIhVjy1Dbo74rfjnyMHLdzqp4Y1WvVQNr6j2Wey3kdF/FlwperTRlhObREwBRIL/9LNTKFuxjlYWxKw7J+CEAZyiRq7DioaabnFHbH/
7Ydxz5wKl7cFMXmsoPiplAbH5HbJWU25krMqjC5xATe1qTC9+Hon29EGtIKZdmIyeZLAre2i9BGHMmJGKE+0+wa00k51oRVNjD+5dXIhj//wRlt/
1VQVeRXm6jI7CYX5NoYRr4tzFl9MQBEQIhWjvXIPqs6sRCHrJj1UJof9HAwQgjIrPt7R1YNXzm4EzPpQWOVFBzUvSozyRp2QbOTzhw5T8dMwunaLoKn/vF+WFnqTAZgAMSGoWp6g+4rSJ/7qZ2eMsXGx/
EbX1m1RzLYRo6lECEM2bjG7Y8g/s2VyD2fPTlb9bCZhDri7E3uILIp2Jzh9+fRgHPy43OtCvhY6ANoGL4BqbWrD+T++g+YIOXCKYwcolKV9zQm37eTTBbs9A04XH0HrxU6O7OAKQVuL3Tz/
xPjIXZuMsI73klP3Z7Q70IZNuIGXb2weZiQWU8OQq5i+g5ejodGPnrv248aGX8b2VHD6HUUJBcBhtYzfRc5O+vjbG0WwVS+ubXlETpP6uELIAU/
tiolt3fES6QWRxeHNzGBustNIK7Del4aUXjuDY8RrVLMFmU9e2i534+56D+PaTr+KepW/gzLvtmPd1MnPp6h2MnX71phAqGZtK0NW9Aa3tH/drI9mEUczAV326Hqs3HOVY70KbT/w1NscCpIN5/
gyXHRVBH3bsOoRZjAVi6h9wXvD69kPY++fTQKETVy3NRnlzDy4yYZLx+XMt5FPsV/htbdvJmeAC3ouShBGLFoAJXhg7cqwaqOpAaUkeTnXKwoER+ORlRFE4aOK1BIXr07HunXLUNa3H2v3ngH3NQEkapi/
ORReHz4teCtJD11B/EUT+67da2pJKSy7m9ryEbs8qpDiLqAhtISEXEKAyhH1WfpZsyUIFL/r7IdlkDsRoy3S4uxdr15wAuvwovSMfhZOTUcn40SgNVIltSbGIjzYPiEVL1/VQkdmMU0F09wi+cFEuIDiFPQ9z/
eM154G8RPQwyKnKcNuYd2JaXoYJO0eJgpvTqOg+ZTnSWFvP8IGbHaggaD6MyVXS5iSl1B5vQxRFHQMMCXi9PtS1MLNLt8HHYUwkIOehinpPjH7enKGZq0KpCOx43w5Od+RCG5yWvJFAzuSM50CvHoZFOcJhyAX4p
+k3edXxlAECoB4Okc0nh32iQYM3UeUCyQm2nHtrEls4IOdbnDZy5ixkEiNc4LE4T85eXIULG0BDAhmbj7nmmlsYEebt5dWED8IRlEb64exMARFI10tjCY7VsKZLAqWoqUbZQFSPWvGZHzju9PQfLwbeUkU2aiioV
i+mBOlPaFahQumb5KLEhxJuGZJ+5m1gBUM7UVVxDriGcJ8j6FNFyMHebBJUMk8hkJFgZW0/
OGBqRsjp9YLFyKYgZHqjzSeCTzGE7RsctiuQq+3gruBf6U65RLjA9FMGEkUQKQFip9pd9fM3sa3tr5Te7qnIed7TPIvZpAGGRiXYS0m4LqZDJlHpIcqqyytw/
8H6YLkJC05aIG0CNUeXTw4OZL3CJJHC3WMovz/
3JuwC5E8aSfcG1C5jcyK4yGrFPhCKKmAKTq7jvnY8PmDjzywF+BBRORyilV1yBzBFFygOBR6yWvtDlRnkiEAqifQDOo9uAkJ006ARHqQxQyKmYrI7HF0h4SmujNahWWJbfvX+StdCirWrMJ/
jhsCXmYPvk15je5McELhUG3xvRylpbmlm178eAvdyEpPUElkj7pJ6KkE317ux9fKU7FKz+7l8EmjTuzvWpFyGwWFK0QUUlxIZK4WaKsSeXjZgu5CmEL/
L2d6HKf5p2SgGLeZnVwUaMSdY3LSEe0KILgcrQyZ5MhFwVWxO/
LuRRWhulTtnIzdfqg4PlxeEFEHiKLMCvLW7LCc8ei6zBl3Xs4w0WNrHQ7WsWWI4pDVNUdQLrTji9z59eZzHR6iBIbvHygtWhPcHHI+tIAClbuCJ9rlOpIAQgvifyykOJr5G8ByjmS/
YA7wc+RjwIDvNCNXQa4gNlMmDT38d7ZfRBnuNE5aVEW6mRxo1/xsy1Fr1ygmz+AEAHIjpEIUYnKkJcoXA2rAzQfSVBbnZ7BSb08y0p1AoczjxKRthTOWvlkQTH78FJZNYpI/sTXUJh/
PxXnHFLzqjFPgwpAXEC2tCqqz2HlC7uBualq81OwGlyYFovc5AS0pYpFWkJCk28F7GiK/
o5xo18JB7BU0s4jQI8CLjylpvwQhbmPI8OwnFgBrx859RhTADIUCgBZH1izdidwvAfTFmdyL5BRWUAZGpWdIRkhTvyrXdbJkTg7M9SHYRSh57G6Ud33tRM4+yQfTufD/
LnLI9z7564U44SU4YKXtgMEoIOfHio2vbkXr/7uMGYsmYgKrg+qQsAFBJ7CYbGijsNSuRtPPVuGMw3t2H7kQmhOoRuP7VmA+cgG52zISHsOEzKWUuNzlLlLT/
L7ADGSsKXE7z9qFBDNmxsjsqR915KNyL45U632OCW54TDm4zB4up5j83H+AmNeJrb94mtquPR6/
TjFX4rIXCLFGZ1vx2dj6BZm0HR76rjB8RED8TzGmaIQUBGMlJEAN3sMCcAc9uTF3g8+we0Pb8LUEieS+IMHWR0ONHF8r5LAwyRpWT4eX1GGZUvmYfKkXJNW6GoyHKoYk5vI4KMJXgpwkyUlAJPhHvr81u3v8rc8G
k+lYe4AqVbkMJfdU3EbdcX44ayGcwSpyJ3ot4Cl29N1kw6ow1+JlNDXU3QEnDHoh8Lzb5PCLVzC2v72x9i977jmFmcw2QmlYcL+bmZPLKQk5NJv5NEI8xepMuEa/+/7iiAPgqAS9vUvp9jt/
ykZahfdCiNU+sitLHQwOUWV5QL9GfGBKvqBbDR4IsA3MQaCoICtn/5IgHtj818DuUB4wGsCTryqjOeyJpxdn9FAONM4QPgXrGAASIZZxVXLGCcKXwA3CsWMEAk46ziP3wnyrgPINtbAAAAAElFTkSuQmCC

```
     </BINVAL>
   </PHOTO>
 </vCard>
</iq>
```

Using ejabberd with MySQL

ejabberd is bundled with a native Erlang driver to use MySQL as a backend for persistent storage. Using MySQL as backend is thus extremely straightforward.

ejabberd installation

ejabberd packages and binary installers contain all the modules needed to connect to your MySQL server. You have no extra module to install anymore.

If you are building ejabberd from source, make sure that you configure ejabberd to include MySQL module. It can be done by passing option `--enable-mysql` to `configure` script. For example:

```
cd ejabberd-source
./configure --enable-mysql
```

MySQL installation

You need a MySQL server that you can point your ejabberd configuration to. The database does not have to be on the same server than ejabberd.

Requirements

ejabberd uses FULLTEXT indexes with InnoDB. Thus, you need MySQL 5.6 or greater to use with ejabberd.

Note: If you do not store message archive in database however, you can try using older 5.5 version. You may need to adapt MySQL database schema to cope with those older MySQL versions.

MySQL on Linux

This documentation will not get into the details of making MySQL running on Linux for production. It is dependent on Linux distribution and system administrators preferences and habits.

It is also well documented, so it should not be an issue.

Amazon RDS compliance

ejabberd is fully compliant with MySQL on Amazon RDS.

You just need to make sure to use MySQL version 5.6 or greater when you create your database.

MySQL on OSX with Homebrew

For testing / development, it is common to start experimenting with MySQL with Homebrew installation.

Here is how to get started to help with setup up environment.

With Homebrew properly installed, you can use the following command to install MySQL:

```
brew install mysql
```

You can then follow instruction to finish the installation, for example by running `mysql_secure_installation`.

You can manually start server with:

```
mysql.server start
```

To connect to your local MySQL server using `mysql` command-line, assuming you kept the default set up, use:

```
mysql -uroot
```

To stop it, use:

```
mysql.server stop
```

MySQL on Windows with Bash

On Windows you can install MySQL easily like on Linux using Ubuntu Bash:

```
sudo apt-get install mysql-server-5.6
```

After configuration, you can start MySQL with:

```
sudo /etc/init.d/mysql start
```

You can connect on the database with your created admin password:

```
mysql -uroot -ppassword
```

MySQL database creation

Create ejabberd user and database

MySQL admins should use this procedure and grant rights to a dedicated `ejabberd` user (replace password with your desired password):

```
echo "GRANT ALL ON ejabberd.* TO 'ejabberd'@'localhost' IDENTIFIED BY 'password';" | mysql -h localhost -u root
```

You can then create a dedicated `ejabberd` database (use password created earlier):

```
echo "CREATE DATABASE ejabberd;" | mysql -h localhost -u ejabberd -p
```

You should now be able to connect to `ejabberd` database with user `ejabberd` (use password defined on GRANT command):

```
mysql -h localhost -u ejabberd -p -D ejabberd

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.7.11 Homebrew

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Decide which SQL schema to use

Read carefully the Default and New Schemas section and decide which schema is preferable in your case: the default or the new schema.

Then modify the `ejabberd.yml` configuration file to setup your desired option value:

```
new_sql_schema: true
```

Use automatic schema update

Since ejabberd 23.10, ejabberd can take care to create the tables automatically the first time it starts with an empty database, and also takes care to update the database schema when you upgrade ejabberd to a newer version.

That feature works both for default and new SQL schema, for MySQL, PostgreSQL and SQLite.

To enable automatic database schema creation and update, simply add in your `ejabberd.yml` configuration file:

```
update_sql_schema: true
```

In that case, you don't need to load the database schema manually: no need to read the next section.

Load database schema manually

MySQL default schema is defined in a file called `mysql.sql`, and the new schema is `mysql.new.sql`. Some tables of the schema are described in: ejabberd SQL database schema documentation.

Those schema files can be found:

- Git repository and source code package: `/sql/` directory
- When installed from source code or binary installer, the SQL schemas are copied to `PREFIX/lib/ejabberd-VERSION/priv/sql`

Load the schema in your `ejabberd` database with the command:

```
mysql -h localhost -D ejabberd -u ejabberd -p < mysql.sql
```

To make sure all looks fine, you can show the list of SQL tables:

```
echo "SHOW TABLES;" | mysql -h localhost -D ejabberd -u ejabberd -p --table

mysql: [Warning] Using a password on the command line interface can be insecure.
+------------------------+
| Tables_in_ejabberd     |
+------------------------+
| archive                |
| archive_prefs          |
| caps_features          |
| last                   |
| motd                   |
| muc_registered         |
| muc_room               |
| privacy_default_list   |
| privacy_list           |
| privacy_list_data      |
| private_storage        |
| pubsub_item            |
| pubsub_node            |
| pubsub_node_option     |
| pubsub_node_owner      |
| pubsub_state           |
| pubsub_subscription_opt |
| roster_version         |
| rostergroups           |
| rosterusers            |
| sm                     |
| spool                  |
| sr_group               |
| sr_user                |
| users                  |
| vcard                  |
| vcard_search           |
| vcard_xupdate          |
+------------------------+
```

Your database is now ready to connect with ejabberd.

ejabberd configuration

Setup MySQL connection

In `ejabberd.yml`, define your database parameters:

```
sql_type: mysql
sql_server: "localhost"
sql_database: "ejabberd"
sql_username: "ejabberd"
sql_password: "password"
## If you want to specify the port:
sql_port: 3306
```

Those parameters are mandatory if you want to use MySQL with ejabberd.

Authentication use MySQL

If you decide to store user password in ejabberd, you need to tell ejabberd to use MySQL instead of internal database for authentication.

You thus need to change ejabberd configuration `auth_method` to replace `internal` authentication with `sql`:

```
auth_method: sql
```

If you restart ejabberd, it should connect to your database for authentication. In case it does not work as expected, check your config file syntax and log files (`ejabberd.log` , `error.log` , `crash.log`)

For example, you can create a user in database with `ejabberdctl`:

```
/sbin/ejabberdctl register "testuser" "localhost" "passw0rd"

User testuser@localhost successfully registered
```

You should now be able to connect XMPP users based on MySQL user base.

Modules use MySQL

At this stage, only the authentication / user base has been moved to MySQL. For data managed by modules, ejabberd still uses the Mnesia internal database by default; you can decide to use MySQL on a module-by-module basis.

For each modules that support SQL backend, you can pass option `db_type: sql` to use your configured MySQL database. Switch can be done on a module by module basis. For example, if you want to store contact list in MySQL, you can do:

```
modules:
  mod_roster:
    db_type: sql
```

However, if you want to use MySQL for all modules that support MySQL, you can simply use global option `default_db: sql`:

```
default_db: sql
```

Note: even if you move all the persistent data you can to MySQL, Mnesia will still be started and used to manage clustering.

Migrating data from internal to MySQL

To migrate your data, once you have setup your sql service, you can move most of the data to your database.

You need to take precautions before you launch the migration:

1. Before you launch migration from internal database, make sure you have made a proper backup.
2. Always try the migration first on an instance created from your data backup, to make sure the migration script will work fine on your dataset.
3. Then, when doing final migration, make sure your instance is not accepting connections by blocking incoming connections, for example with firewall rules (block port 5222, 5269 and 5280 as default).

When you are ready, you can:

1. Connect to a running ejabberd:

```
./ejabberdctl debug
```

2. Alternatively, use `ejabberdctl live` to launch ejabberd with an Erlang shell attached.

3. Launch the migration command `ejd2sql:export/2` from Erlang shell. First parameter is the XMPP domain name you want to migrate (i.e `localhost`). Second parameter `sql` tells ejabberd to export to configured MySQL database. For example:

```
ejd2sql:export(<<"localhost">>, sql).
```

You should be set now.

Converting database from default to new schema

Please check the section Default and New Schemas.

Getting further

To get further you can read the ejabberd Configuration section about Databases.

Development

ejabberd for Developers

As a developer, you can customize ejabberd to design almost every type of XMPP related type of solutions.

As a starting point, we recommend that you get extremely familiar with both the core XMPP protocol itself and its extensions.

From that, once you understand well XMPP, you can tame ejabberd to build your dream messaging system.

Getting started

Source code

ejabberd source is available on Github: ejabberd

You will need to get familiar with it to start learning about ejabberd module writing. The first place to start? You should read the time module. This is one of the simplest possible module for ejabberd.

Another great source of inspiration and knowledge is to read the source code of the many contributed ejabberd modules. Many of them are available from ejabberd-contribs repository.

For a complete overview of ejabberd source code and its dependencies, please refer to ejabberd and related repositories

Development Environment

The first step to develop for ejabberd is to install and configure your development environment:

- Check the Source Code Installation section
- If using Emacs, install erlang-mode in your operating system
- If using OSX, check the OSX development environment section
- For Visual Studio Code and alternatives, check the Developing ejabberd with VSCode section

Customizing ejabberd

- ejabberd development guide
- ejabberd modules development

ejabberd developer guide

Introduction

`ejabberd` is a *free and open source* instant messaging server written in `Erlang/OTP`.

`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.

Goals

This guide is a brief explanation of ejabberd internals. It is not intended to be a comprehensive ejabberd's internal API documentation. You still need to read and understand ejabberd's source code. However, the guide is believed to help you understanding ejabberd's code faster: it provides entry points from where to start reading relevant parts of the code and ignore irrelevant ones. Note that there is absolutely no need to know every line of code of ejabberd, but some parts are crucial to understand.

Requirements

In order to read and understand the guide you must be pretty fluent with Erlang programming language and understand basics of the XMPP protocol: there is no detailed explanation of Erlang syntax and/or features and it's assumed that you're familiar with such terms as `xml stream`, `stanza`, `c2s`, `s2s` and so on. If you see these words for the first time in your life you're unlikely to understand the guide.

Version

The guide describes ejabberd 17.03. Previous and next versions can differ drastically from the one described herein.

Coding style convention

NOTE: this section is only relevant for ejabberd contributors. If you're hacking ejabberd for internal needs, you are free to choose whatever coding style you like.

ejabberd follows Erlang Coding Standards & Guidelines or at least tries to do so: there is still a lot of poorly written legacy code (which is being leisurely rewritten), but the new code should be written with keeping these rules in mind. In some cases the rules can be bypassed, but the reason doing so should be really weighty. The rules shouldn't be ignored just because a contributor doesn't like them.

The typical coding style rules found violated in contributors' code are:

- 100 column per line: in fact we have defined 80 columns as a soft and 100 columns as a hard limit, which means most of your lines should be no longer than 80 characters and the rest must never be longer than 100 characters.
- no deep nesting
- no boolean parameters in case control
- only CamelCase variables name
- no macros
- no case-catch

It's worth noting that the code itself should be indented using Emacs indentation style (that is the standard indentation style for Erlang programs). If you're not using Emacs for ejabberd development, indent the code using it first before making a PR/commit.

Start-up procedure

ejabberd is written as a standard OTP application, so the startup module can be found in `src/ejabberd.app.src` or, if ejabberd is compiled, in `ebin/ejabberd.app` file: that is, `ejabberd_app.erl` module from where `start/2` function is called by Erlang application controller. This function makes some initialization (such as logger, mnesia, configuration file, etc.) and ends up by starting the main ejabberd supervisor - `ejabberd_sup`. Thus, for further startup order refer to `ejabberd_sup.erl` module (this is a simple list-like module with supervisor childspecs).

WARNING: only "core stuff" should be attached to `ejabberd_sup`. For attaching modules use gen_mod's supervisor (via `gen_mod:start_child/3,4` functions), for attaching database backend modules use `ejabberd_backend_sup` supervisor, etc.

Once `ejabberd_sup` is started, ejabberd application is considered to be started.

Core

The ejabberd core is not well-defined. Moreover, the described core layers are pure abstraction grouping several modules together by some criteria for better understanding of ejabberd internal processing rules.

Network Layer

Once ejabberd is started, some external events should obviously make it doing something. Besides explicit administrative commands, the most relevant such events are incoming connections. Incoming connections are handled inside `Network Layer`. The layer implemented by `ejabberd_listener.erl`, `ejabberd_receiver.erl` and `ejabberd_socket.erl` modules.

NOTE: `ejabberd_listner.erl` is able to handle raw TCP and UDP connections, however only XMPP connections are described here.

Once a connection is accepted by `ejabberd_listener.erl`, an instance (a process) of `ejabberd_receiver.erl` is started and it becomes the socket owner, where it performs the following operations:

- Throttles a connection using shapers from `shaper.erl` module
- Performs TLS decoding using fast_tls library
- Performs stream decompression using ezlib library
- Parses incoming raw XML data into `#xmlel{}` packets using fast_xml library

`ejabberd_socket.erl` does the same but in a reverse order, i.e. it performs stream compression and/or TLS encoding, serializes `#xmlel{}` packets into raw XML data and puts them into a socket (note that shapers do not apply for outgoing data).

Once `xmlel{}` packet is constructed by `ejabberd_receiver.erl` it's passed to `XMPP Stream Layer`.

XMPP Stream Layer

`XMPP Stream Layer` is represented by `xmpp_stream_in.erl` and `xmpp_stream_out.erl` modules. An instance (i.e. a process) of `xmpp_stream_in.erl` is started along with an instance of `ejabberd_receiver.erl` and all incoming `#xmlel{}` packets are passed from the latter to the former. `xmpp_stream_in.erl` module does the following:

- Encodes/decodes `#xmlel{}` packets using xmpp library from/to internal structures (records) defined in xmpp_codec.hrl.
- Performs negotiation of *inbound* XMPP streams
- Performs STARTTLS negotiation (if needed)
- Performs compression negotiation (if needed)
- Performs SASL authentication

NOTE: `XMPP Stream Layer` was only introduced in ejabberd 17.03. Prior to this XMPP stream negotiation was handled inside `ejabberd_c2s.erl`, `ejabberd_s2s_in.erl`, `ejabberd_service.erl` and `ejabberd_s2s_out.erl`. This has lead to unmaintainable monolithic spaghetti code with a lot of code duplication between these modules. It's believed introducing `xmpp_stream_in.erl` and `xmpp_stream_out.erl` modules now solves this problem.

During these procedures `xmpp_stream_in.erl` calls functions from its callback modules, i.e. the modules of `xmpp_stream_in` behaviour: `ejabberd_c2s.erl`, `ejabberd_s2s_in.erl` or `ejabberd_service.erl`, depending on the stream namespace.

`xmpp_stream_out.erl` does the same but for *outbound* XMPP streams. The only its callback module is `ejabberd_s2s_out.erl`.

NOTE: `xmpp_stream_in.erl` shares the same process and state with its callback modules, i.e. functions from `xmpp_stream_in.erl` and functions from `ejabberd_c2s/s2s_in/service.erl` modules are evaluated inside the same process. This is also true for `xmpp_stream_out.erl` and `ejabberd_s2s_out.erl`. The state is represented by a `map()` in both cases.

EJABBERD_C2S, EJABBERD_S2S_IN AND EJABBERD_SERVICE

These are modules of `xmpp_stream_in` behaviour. The only purpose of these modules is to provide callback functions for `xmpp_stream_in.erl` module. Examples of such callback functions are:

- `tls_enabled/1` : tells whether or not TLS is enabled in the configuration
- `check_password_fun/1` : provides a function for SASL authentication
- `handle_authenticated_packet/2` : what to do with packets after authentication is completed

Roughly, they represent an intermediate (or "glue") code between `XMPP Stream Layer` and `Routing Layer` for inbound XMPP streams.

`ejabberd_s2s_out.erl` is described elsewhere

Routing Layer

EJABBERD_ROUTER

`ejabberd_router.erl` module is the main dispatcher of XMPP stanzas.

It's pretty small and straightforward module whose the only task is to find the "route" for a stanza. `ejabberd_router.erl` only operates with `#message{}`, `#presence{}` and `#iq{}` packets (defined in xmpp_codec.hrl), so please note, that it is not possible to route arbitrary `#xmlel{}` packets or any other Erlang terms through `ejabberd_router`.

The only valid routes are:

- *local* route: stanzas of this route type are destined to the local server itself, i.e. stanzas with `to` attribute in the form of `domain.com` or `domain.com/resource`, where `domain.com` is a virtual host serviced by ejabberd. `ejabberd_router` passes such stanzas to `ejabberd_local.erl` module via `ejabberd_local:route/1` function call.
- *session manager* route: stanzas of this route type are destined to local users, i.e. stanzas with `to` attribute in the form of `user@domain.com` or `user@domain.com/resource` where `domain.com` is a virtual host serviced by ejabberd. `ejabberd_router` passes such stanzas to `ejabberd_sm.erl` module via `ejabberd_sm:route/1` function call.
- *registered* route: if a stanza is not destined to local virtual host, ejabberd first checks if there is a "registered" route for the stanza, i.e. a domain registered via `ejabberd_router:register_route/2` function. For doing this it looks up the routing table and if there is a process `Pid` registered on this domain, ejabberd routes the stanza as `Pid ! {route, Stanza}`. The routing table is backend-dependent and is implemented in the corresponding backend module such as `ejabberd_router_mnesia.erl`.
- *s2s* route: if a stanza is neither destined to local virtual host nor to registered route, `ejabberd_router` passes it to `ejabberd_s2s.erl` module via `ejabberd_s2s:route/1` function call.

Mentioned modules are explained in more details in the following sections. You're encouraged to inspect exported functions of `ejabberd_router.erl`, because most likely you will use some of them.

EJABBERD_LOCAL

`ejabberd_local.erl` handles stanzas destined to the local server itself. For `#message{}` and `#presence{}` it only calls hooks, while for `#iq{}` it finds the corresponding "IQ handler" by looking up its internal table to find a correspondence between a namespace of IQ's child element and the handler. Once the handler (an erlang function) is found, it passes further IQ processing to `gen_iq_handler.erl` via `gen_iq_handler:handle/5` call.

`ejabberd_local.erl` is also able to send IQ requests and to process responses for them. This is implemented in `ejabberd_local:route_iq/2,3` functions. This is also the most notable function of the module. Calling to other functions is not recommended.

EJABBERD_SM

`ejabberd_sm.erl` handles stanzas destined to local users. For `#message{}`, `#presence{}` and full-JID `#iq{}` it looks up its internal table (aka `session` table) for the corresponding `ejabberd_c2s` process and, if the process is found, it routes the stanza to this process via `ejabberd_c2s:route/2` call.

Bare-JID `#iq{}` stanzas are processed in a similar way as in `ejabberd_local.erl`. The internal `session` table is backend-dependent and is implemented in the corresponding backend module: `ejabberd_sm_mnesia.erl`, `ejabberd_sm_redis.erl` and so on.

The most notable functions of the module are:

- `get_user_resources/2`
- `dirty_get_sessions_list/0`
- `dirty_get_my_sessions_list/0`
- `get_vh_session_list/1`
- `get_vh_session_number/1`
- `get_vh_by_backend/1`
- `get_session_pid/3`
- `get_user_info/2`
- `get_user_info/3`
- `get_user_ip/3`
- `is_existing_resource/3`

ROUTE-REGISTERED PROCESSES

Any process can register a route to itself. It's done by calling to `ejabberd_router:route/2` function. Note that a route should be unregistered via `ejabberd_router:unregister_route/1` function if the registering process terminates or the route is no longer needed. Once a route is registered to a process, this process will receive Erlang messages in the form of `{route, Stanza}`.

NOTE: `from` and `to` fields are always set in the `Stanza`, so it's safe to assume that `xmpp:get_from(Stanza)` and `xmpp:get_to(Stanza)` always return `#jid{}` and never `undefined`.

Refer to the code of `mod_muc.erl` or `ejabberd_service.erl` for an example of a route-registered process.

EJABBERD_S2S AND EJABBERD_S2S_OUT

If a stanza is destined neither to local virtual host not to a route-registered process, it's passed to `ejabberd_s2s.erl` module via `ejabberd_s2s:route/1` function call. `ejabberd_s2s` in its turn will look up the internal table (currently it's `s2s` Mnesia table) for the `ejabberd_s2s_out` process and, if found, passes the stanza to this process or, otherwise, will start new `ejabberd_s2s_out` process.

`ejabberd_s2s_out.erl` handles *outbound* XMPP S2S streams. This is the only callback module of `xmpp_stream_out` behaviour.

Adding new functionality

There are two common ways to add new functionality to ejabberd:

- using IQ Handlers
- using hooks

Here is a rule of thumb on which way to choose:

- if you want to handle newly introduced IQs (that is, to generate replies for them), use IQ handlers
- if you want to modify ejabberd behaviour along the way of a stanza passing through all layers or want to "listen" for some internal events (like ejabberd configuration change), use hooks.

IQ Handlers

An `IQ Handler` is a function processing an IQ stanza (internally represented as `#iq{}` record). There are two types of IQ handlers: `local` and `sm`.

- `local` IQ handler is a function processing IQs coming from `ejabberd_local`, that is, an IQ destined to the local server itself as described in ejabberd_local.

- `sm` IQ handler is a function processing IQs coming from `ejabberd_sm`, that is, a bare-JID IQ destined to a local user as described in ejabberd_sm.

An IQ handler is registered as:

```
gen_iq_handler:add_iq_handler(Type :: ejabberd_local | ejabberd_sm,
                              Host :: binary(),
                              Namespace :: binary(),
                              Module :: module(),
                              Function :: atom()) -> ok
```

where:

- `Type` is `ejabberd_local` for `local` handlers or `ejabberd_sm` for `sm` handlers
- `Host` is a virtual host for which the IQ is to be processed
- `Namespace` is an XML namespace of IQ's child element

Once registered, matching `IQ` stanzas are handled by calling `Module:Function(IQ)`. The result should be in the form of `#iq{}` or `ignore`. When `#iq{}` is returned, it's treated as a reply and routed back to the IQ originator, otherwise, if `ignore` is returned, the further processing stops.

NOTE: `from` and `to` fields are always set in the `IQ`, so it's safe to assume that `xmpp:get_from(IQ)` and `xmpp:get_to(IQ)` always return `#jid{}` and never `undefined`.

If a handler is no longer needed it should be unregistered as:

```
gen_iq_handler:remove_iq_handler(Type :: ejabberd_local | ejabberd_sm,
                                 Host :: binary(),
                                 Namespace :: binary()) -> ok
```

with the same meaning of the arguments.

Hooks

When ejabberd is processing an arbitrary event (incoming IQ, outgoing presence, configuration change, etc), it is convenient to consider some of them notable. In order for someone to be notified of such events, ejabberd executes "hooks". A hook is represented by a unique name. All functions associated with the hook's name will be called in some specified order.

NOTE: The conception of hooking is not ejabberd specific, see Hooking Wikipedia page for a general description.

For example, when a packet is received on a client connection, ejabberd runs `user_send_packet` hook. Several modules need to listen for an event represented by this hook (that is, a packet and a C2S state), so they associate their internal functions with it: `mod_ping.erl` associates `user_send/1` function, `mod_privacy.erl` associates `user_send_packet/1` function and so on. The event is passed as an argument to the "hooked" functions, thus, the function from `mod_ping.erl` will be called as `mod_ping:user_send({Stanza, C2SState})`, the function from `mod_privacy.erl` will be called as `mod_privacy:user_send_packet({Stanza, C2SState})` and so on.

There are two types of hooks: *with* an accumulator and *without* an accumulator.

- a hook with an accumulator, as its name suggests, accumulates some state during execution of a list of associated functions: the first argument of the hooked function will always be an accumulator and the function must return the new value for the accumulator (whether it's modified or not) in the form of `NewAcc` or `{stop, NewAcc}`. If `{stop, NewAcc}` is returned, a hook is considered evaluated and next functions in its associated list are not called. Otherwise, the new value `NewAcc` is passed to the next function in the associated list. An example of hooks with accumulator are: `disco_info`, `filter_packet`, `muc_process_iq` and so on.
- a hook without accumulator doesn't accumulate anything during execution of a list of associated functions: the returning values of such functions are simply ignored unless `stop` is returned. In the latter case, evaluation of next functions in the associated list is not performed. An example of hooks without accumulator are: `config_reloaded`, `component_init` and so on.

Both types of hooks have *local* or *global* scope.

- a hook with local scope is associated with particular virtual host and is run only when an event is matching this host. Most of the hooks have local scope.
- a hook with global scope is not associated with any virtual host and is run for an event matching any hosts. A very few hooks have global scope.

A function gets associated with a local hook as follows (the type of a hook doesn't matter):

```
ejabberd_hooks:add(Hook :: atom(),
                   Host :: binary(),
                   Module :: module(),
                   Function :: atom(),
                   Seq :: integer() -> ok
```

where:

- `Hook` is a hook name
- `Host` is a virtual host
- `Seq` is a sequence number. This number defines position of the function in the list to maintain execution order. Functions with lower sequence number are executed *before* those with bigger sequence number. For functions with the same sequence number the order is unspecified. A function associated with an accumulating hook is called as `Module:Function(Acc, Arg1, Arg2, ...)` where `Acc` is an accumulator value, `Arg1`, `Arg2`, ... - arguments of the hook. Recall that such function must return a new accumulator value (whether it's modified or not) in the form of `NewAcc` or `{stop, NewAcc}` where `NewAcc` is the new accumulator value. A function associated with a hook without an accumulator is called as `Module:Function(Arg1, Arg2, ...)`. All returning values except `stop` are ignored.

WARNING: a `Function` with the corresponding arity should be exported by a `Module`

A function for a global hook gets associated as follows (the type of a hook doesn't matter):

```
ejabberd_hooks:add(Hook :: atom(),
                   Module :: module(),
                   Function :: atom(),
                   Seq :: integer()) -> ok
```

with the same meaning of the arguments. Note that `Host` argument is omitted in this case.

For any types of hooks, if an association is no longer needed, it can be deleted by calling `ejabberd_hooks:delete/5,6` functions with exactly the same arguments used to create an association.

In some cases a new hook should be introduced. There is no need to explicitly register the new hook, one only needs to run a hook in the required place. The following functions can be used for this:

- for local hooks with accumulator: `ejabberd_hooks:run_fold(Hook, Host, Acc, Args)`. The function returns a new accumulator value.

- for local hooks without accumulator: `ejabberd_hooks:run(Hook, Host, Args)`. The function always returns `ok`.

- for global hooks with accumulator: `ejabberd_hooks:run_fold(Hook, Acc, Args)`. The function returns a new accumulator value.

- for global hooks without accumulator: `ejabbed_hooks:run(Hook, Args)`. The function always returns `ok`.

where `Args` is a list of arguments (other variables have the same meaning as above).

There is a helper script that you can use to check hook correctness and find mishooked functions. The script also generates a module `src/hooks_type_test.erl` from where you can learn about existing hooks and check execution order. You can place your code inside `src` directory (if any), and run:

```
make hooks
```

Modules

gen_mod behaviour

As you might know, ejabberd is a modular software. The best method to add new functionality to it is to write a new module. For doing this one should create an Erlang module of `gen_mod` behaviour:

```
%% file mod_foo.erl
-module(mod_foo).
...
-behaviour(gen_mod).
...
```

Several callbacks should be defined in the module:

- `Module:start(Host, Opts)` where `Host` is a virtual host where the module is about to start and `Opts` is an option list (typically defined in the `modules` section of `ejabberd.yml`). The function is executed when a module is being started. It is intended to initialize a module. This is a good place to register hooks and IQ handlers, as well as to create an initial state of a module (if needed). The function should return either `ok` or `{ok, pid()}`.

- `Module:stop(Host)` where `Host` is a virtual host. The function is executed when a module is being stopped. It is intended to make some module cleanup: most likely unregistering hooks and IQ handlers. The returning value is ignored

- `Module:reload(Host, NewOpts, OldOpts)` where `NewOpts` and `OldOpts` is the new and old options list respectively. The function is called every time a module is being reloaded. This is the only optional callback, thus, if undefined, the module will be reloaded by calling sequentially `Module:stop/1` and `Module:start/2`.

- `Module:depends(Host, Opts)` where the meaning of the arguments is the same. The function is called to build modules dependencies on startup. The function must return a list of type `[{module(), DependencyType}]`, where `DependencyType` is one of `hard` or `soft`. The `hard` dependency means the module is non-functional if the other module is not loaded. The `soft` dependency means the module has suboptimal functionality if the other module is not loaded.

- `Module:mod_opt_type(Option)`. The function is used to process configuration options of `Module`. The function has the same meaning as `Module:opt_type/1` callback described in Configuration validation section.

Stateful modules

While some modules don't need to maintain an internal state ("stateless" modules), others are required to do this ("stateful" modules). The common practice is to implement a stateful module as a `gen_server` process. There is a couple of helpers to deal with such modules:

- `gen_mod:start_child(Module, Host, Opts)` where `Module` is a name of a stateful module. This function should be called as the last function inside of `Module:start/2`. It will create a `gen_server` process with a registered name and will attach it to `ejabberd_gen_mod_sup` supervisor.

- `gen_mod:stop_child(Module, Host)` should be used inside of `Module:stop/1` function and will terminate the corresponding registered `gen_server` process.

- `gen_mod:get_module_proc(Host, Module)` can be used to obtain a registered name of a stateful module (i.e. its `gen_server`'s name).

WARNING: don't forget to set `process_flag(trap_exit, true)` inside `Module:init/1` callback function, otherwise, `Module:terminate/2` callback will never be called when a module is being stopped.

WARNING: keeping module's configuration options in an internal state is not recommended. Use `gen_mod:get_module_opt/4,5` functions to retrieve the options: in this case you don't need to re-initialize options in the state inside `Module:reload/3` callback.

If a stateful module is intended to maintain a state in the form of a table, `ETS` can be used for this. In this case there is no need to implement it as a `gen_server` process. But make sure you're not calling `ets:new/2` several times for several virtual hosts (`badarg` will be raised in this case). E.g., the following code is *incorrect*:

```
start(Host, Opts) ->
    ...
    ets:new(some_table, named_table, ...]),
    ...
```

The correct code will look something like that:

```
start(Host, Opts) ->
    ...
    try ets:new(some_table, [named_table, ...])
    catch _:badarg -> ok end,
    ...
```

There is a plenty of examples of modules: pick up any file starting with `mod_` inside `src` directory.

gen_mod module

Module `gen_mod.erl` has various useful functions to work with modules, the most notable are:

- `is_loaded/2`: whether or not the module in question is loaded at a given virtual host

- `get_opt/3,4`: gets a value of an option from module's options list (see description of `ejabberd_config:get_option/3` function from Fetching configuration options for details)

- `get_module_opt/4,5`: the same as above, but an option is referenced by a virtual host and a module.

Configuration

ejabberd has quite powerful configuration processor - `ejabberd_config.erl`. It performs configuration file parsing and validation.

Validation

In order to validate options `ejabberd_config` has to install feedback with the rest of the code. For doing this, it provides `ejabberd_config` behaviour with a single callback function: `Module:opt_type/1`. The callback accepts an option name as an `atom()` and must return either validating function if an option is known for the `Module` or a list of available options (as a list of atoms). A validating function is a `fun()` of a single argument - the value of the option. The validating function must return any new value for the option (whether it's modified or not) or should crash if the value doesn't match expected format. Here is an example:

```
%% file: some.erl
-module(some).
-behaviour(ejabberd_config).
```

```
-export([opt_type/1]).
...
opt_type(max_connections_number) ->
    %% max_connections_number should be non-negative integer
    %% if the condition is satisfied, return this integer
    %% fail with function_clause otherwise
    fun(I) when is_integer(I), I>=0 -> I end;
opt_type(_) ->
    %% only max_connections_number is known
    [max_connections_number].
```

NOTE: `gen_mod` behaviour defines a very similar callback - `Module:mod_opt_type/1` with the same meaning of arguments and returning values, except the callback is called to validate the `Module`'s specific options (i.e. options defined in the corresponding subsection of the `modules` section of a configuration file).

Fetching options

The most notable function of the module is:

```
get_option(Option :: atom() | {atom(), binary() | global},
           ValidatingFun :: fun(),
           Default :: term()) -> Value :: term().
```

The function is used to get a value `Value` of a configuration option `Option`. The `ValidatingFun` is a validating function described in the previous section and `Default` is the default value if the option is not defined in the config.

Using XMPP library

xmpp module

Prior to version 16.12, ejabberd used to operate with `#xmlel{}` packets directly: `fast_xml` API functions have been used for manipulating with `#xmlel{}` packets (such as `fast_xml:get_subtag/2`, `fast_xml:get_attr_s/2`, `fast_xml:get_path_s/2` and so on) as well as some functions from `jlib.erl` module.

This is now deprecated and actually not possible. Instead, the new API functions are used from brand new xmpp library.

NOTE: although direct calling of `fast_xml` API is deprecated, there are still two useful functions: `fxml_stream:parse_element/1` and `fxml:element_to_binary/1`. You can use these functions for (de)serialization of data stored on disc or in a database.

The library is built on top of `XMPP Codec`: a number of decoding/encoding modules automatically generated by Fast XML generator from the specification file xmpp_codec.spec. The goal is to avoid manual processing of XML trees and, instead, using well-typed auto-generated structures defined in xmpp_codec.hrl. Every particular XML packet within some namespace has to have a specification defined in `xmpp_codec.spec`. The advantage of such approach is that you tell the generator *what* to parse instead of taming `fast_xml` library *how* to parse.

NOTE: describing how to write XMPP codec specification is out of scope of this guide

WARNING: you should never use functions from `xmpp_codec.erl` module directly: use functions from `xmpp.erl` module. The same is true for header files: do **NOT** include `xmpp_codec.hrl` -- include `xmpp.hrl` instead

XMPP CODEC

Once a raw XML packet is parsed by `ejabberd_receiver.erl` into `#xmlel{}` record, it's passed to `xmpp_stream_in.erl` module, where decoding of `#xmlel{}` into `xmpp_element()` format (i.e. into well-known record type defined in `xmpp_codec.hrl`) is performed (refer to XMPP Stream Layer section for details). At that level "lazy" decoding is applied: only top-level element is decoded. For example, an `xmlel()` packet

```
#xmlel{name = <<"message">>,
       attrs = [{<<"type">>,<<"chat">>}],
       children = [#xmlel{name = <<"composing">>,
                          attrs = [{<<"xmlns">>,
                                    <<"http://jabber.org/protocol/chatstates">>}],
                          children = []}]}
```

is decoded into the following `xmpp_element()`:

```
#message{id = <<>>,type = chat,lang = <<>>,from = undefined,
         to = undefined,subject = [],body = [],thread = undefined,
```

```
        sub_els = [#xmlel{name = <<"composing">>,
                       attrs = [{<<"xmlns">>,
                                <<"http://jabber.org/protocol/chatstates">>}],
                       children = []}],
        meta =
```

Note that the sub-element is still in `xmlel()` format. This "semi-decoded" packet is then passed upstream (at the Routing Layer). Thus, a programmer should explicitly decode sub-elements if needed. To accomplish this one can use the following function:

```
xmpp:decode(El :: xmlel(), Namespace :: binary(), [Option]) -> xmpp_element()`
```

where the only supported `Option` is `ignore_els` : with this option lazy decoding is performed. By default, full decoding is applied, i.e. all known sub-elements get decoded. `Namespace` is a "top-level" namespace: it should be provided only if `<<"xmlns">>` attribute is omitted in `El` , otherwise decoding would fail (see below).

There is also `xmpp:decode(El :: xmlel()) -> xmpp_element()` function, which is a short-hand for `xmpp:decode(El, ?NS_CLIENT, [])` (where `?NS_CLIENT` is a predefined namespace for `<<"jabber:client">>` , see Namespaces section).

Both functions might **fail** with `{xmpp_codec, Why}` exception. The value of `Why` can be used to format the failure reason into human readable description using `xmpp:format_error/1` function, e.g., using sub-element from example `#message{}` above, we can write:

```
try xmpp:decode(El) of
    #chatstate{} = ChatState -> process_chatstate(ChatState)
catch _:{xmpp_codec, Why} ->
    Text = xmpp:format_error(Why),
    ?ERROR_MSG("failed to decode element: ~s", [Txt])
end
```

To apply reverse operation use `xmpp:encode/2` functions:

```
xmpp:encode(Pkt :: xmpp_element(), Namespace :: binary()) -> El :: xmlel()
```

There is also `xmpp:encode(Pkt :: xmpp_element()) -> El :: xmlel()` function which is a short-hand for `xmpp:encode(Pkt, <<>>)` .

`Namespace` is a "top-level" namespace: it is used to tell the codec whether to include `<<"xmlns">>` attribute into resulting `#xmlel{}` element or not -- if the `Pkt` is within the same `Namespace` , `<<"xmlns">>` attribute will be omitted in the result. For example:

```
> rr(xmpp).
...
> Msg.
#message{id = <<>>,type = chat,lang = <<>>,from = undefined,
        to = undefined,subject = [],body = [],thread = undefined,
        sub_els = [#chatstate{type = composing}],
        meta =
> xmpp:encode(Msg).
#xmlel{name = <<"message">>,
       attrs = [{<<"type">>,<<"chat">>},
               {<<"xmlns">>,<<"jabber:client">>}],
       children = [#xmlel{name = <<"composing">>,
                       attrs = [{<<"xmlns">>,
                                <<"http://jabber.org/protocol/chatstates">>}],
                       children = []}]}
> xmpp:encode(Msg, <<"jabber:client">>).
#xmlel{name = <<"message">>,
       attrs = [{<<"type">>,<<"chat">>}],
       children = [#xmlel{name = <<"composing">>,
                       attrs = [{<<"xmlns">>,
                                <<"http://jabber.org/protocol/chatstates">>}],
                       children = []}]}
```

NOTE: `xmpp:encode/1,2` functions would never fail as long as the provided input is a valid `xmpp_element()` with valid values of its record fields. Use dialyzer checks of your code for validation.

NOTE: there is no need to explicitly decode a sub-element of an IQ passed into an IQ handler because decoding is performed inside `gen_iq_handler.erl` module and a handler actually will never receive malformed sub-elements.

Luckily, there is a helper function for sub-elements decoding, described in the next section and in a lot of cases it's more convenient to use it.

GETTING SUB-ELEMENTS

Once a programmer gets a stanza in `xmpp_element()` format, (s)he might want to get its subelement. To accomplish this the following function can be used:

```
xmpp:get_subtag(Stanza :: stanza(), Tag :: xmpp_element()) -> Pkt :: xmpp_element() | false
```

This function finds a `Tag` by its well-known record inside sub-elements of the `Stanza`. It automatically performs decoding (if needed) and returns either found `xmpp_element()` or `false` if no elements have matched. Note that the function doesn't fail if some of sub-elements are invalid.

Example:

```
> rr(xmpp).
...
> Msg.
#message{id = <<>>,type = chat,lang = <<>>,from = undefined,
         to = undefined,subject = [],body = [],thread = undefined,
         sub_els = [#xmlel{name = <<"composing">>,
                           attrs = [{<<"xmlns">>,
                                     <<"http://jabber.org/protocol/chatstates">>}],
                           children = []}],
         meta =
> xmpp:get_subtag(Msg, #chatstate{type = composing}).
#chatstate{type = composing}
> xmpp:get_subtag(Msg, #chatstate{type = inactive}).
false
> xmpp:get_subtag(Msg, #disco_info{}).
false
```

SETTING AND REMOVING SUB-ELEMENTS

In order to inject a sub-element into or delete one from arbitrary `stanza()` one can use `xmpp:set_subtag/2` and `xmpp:remove_subtag/2` respectively.

FROM AND TO

Every `stanza()` element has `from` and `to` record fields. In order to get/set them one can manipulate with these record fields directly, e.g. via `Msg#message.from` or `Pres#presence.to` expressions, or, use `xmpp:get_from/1`, `xmpp:get_to/1`, `xmpp:set_from/2`, `xmpp:set_to/2` and `xmpp:set_from_to/3` functions, depending on which approach is more convenient in the current situation.

NOTE: although in general `from` and `to` fields may have `undefined` values, these fields are always filled with correct `#jid{}` records at XMPP Stream Layer, thus, it is safe to assume that the fields always possess valid `#jid{}` values.

METADATA

Every `stanza()` element has `meta` field represented as a `map()`. It's useful when there is a need to attach some metadata to the stanza before routing it further. A programmer can manipulate with this field directly using maps module, or use `xmpp:get_meta/1,2,3`, `xmpp:set_meta/2`, `xmpp:put_meta/3`, `xmpp:update_meta/3` and `xmpp:del_meta/2` functions, which is almost always more convenient (except pattern matching).

TEXT ELEMENTS

Some `xmpp_element()`s has fields defined in `[#text{}]` format. The example is `#message.body` and `#presence.status` fields. To avoid writing a lot of extracting code the following functions can be used: `xmpp:mk_text/1,2` to convert some binary text written in some language into `[#text{}]` term, or `xmpp:get_text/1,2` to extract binary text from the `[#text{}]` element by a language.

GENERATING ERRORS

In order to generate stanza errors or stream errors `xmpp:err_/0,2` or `xmpp:serr_*/0,2` can be used respectively, such as `xmpp:err_service_unavailable()` or `xmpp:serr_not_authorized()`. If a stanza should be bounced back with an error, `xmpp:make_error/2` function can be used

NAMESPACES

There are many predefined macros for XML namespaces in ns.hrl. However, this file must **NOT** be included, as it's already included in `xmpp.hrl`.

A function `xmpp:get_ns/1` can be used to retrieve a namespace from `xmpp_element()` or from `xmlel()` directly:

```
> rr(xmpp).
...
> xmpp:get_ns(#message{}).
<<"jabber:client">>.
> xmpp:get_ns(xmpp:encode(#presence{})).
<<"jabber:client">>.
```

jid module

`jid.erl` module provides functions to work with XMPP addresses (aka "JIDs"). There are two common types of internal representation of JIDs:

- `jid()`: a JID is represented by a record `#jid{}` defined in jid.hrl
- `ljid()`: a JID is represented by a tuple `{User, Server, Resource}` where `User`, `Server` and `Resource` are stringprepped version of a nodepart, namepart and resourcepart of a JID respectively. This representation is useful to use for JIDs comparison and when a JID should be used as a key (in a Mnesia database, ETS table, etc.)

The most notable functions in this module are:

- `decode(Input :: binary()) -> jid()`: decodes binary data into `jid()`. Fails with `{bad_jid, Input}` otherwise.
- `encode(JID :: jid() | ljid()) -> binary()`: encodes `JID` into binary data
- `remove_resource(JID :: jid() | ljid()) -> jid() | ljid()`: removes resource part of a `JID`
- `replace_resource(JID :: jid() | ljid(), Resource :: binary()) -> jid() | ljid()`: replaces resource part of a `JID`
- `tolower(JID :: jid() | ljid()) -> ljid()`: transforms `JID` into `ljid()` representation
- `make(LJID :: ljid() | jid()) -> jid()`: transforms `LJID` into `jid()` representation

Inspect exported functions of jid.erl for more details.

External Authentication

You can configure ejabberd to use as authentication method an external script, as described in the Administrator section: External Script.

Let's see the interface between ejabberd and your script, and several example scripts. There are also several old example scripts.

Extauth Interface

The external authentication script follows the Erlang port driver API.

That script is supposed to do these actions, in an infinite loop:

- read from stdin: AABBBBBBBBBB.....
- A: 2 bytes of length data (a short in network byte order)
- B: a string of length found in A that contains operation in plain text operation are as follows:

- auth:User:Server:Password (check if a username/password pair is correct)
- isuser:User:Server (check if it's a valid user)
- setpass:User:Server:Password (set user's password)
- tryregister:User:Server:Password (try to register an account)
- removeuser:User:Server (remove this account)
- removeuser3:User:Server:Password (remove this account if the password is correct)

- write to stdout: AABB
- A: the number 2 (coded as a short, which is bytes length of following result)
- B: the result code (coded as a short), should be 1 for success/valid, or 0 for failure/invalid

As you noticed, the `:` character is used to separate the fields. This is possible because the User and Server fields can't contain the `:` character; and Password can have that character, but is always the last field. So it is always possible to parse the input characters unambiguously.

Perl Example Script

This is a simple example Perl script; for example if the file is copied to the path `/etc/ejabberd/check_pass_null.pl` then configure ejabberd like this:

```
auth_method: [external]
extauth_program: /etc/ejabberd/check_pass_null.pl
```

Content of `check_pass_null.pl`:

```perl
#!/usr/bin/perl

use Unix::Syslog qw(:macros :subs);

my $domain = $ARGV[0] || "example.com";

while(1)
  {
    # my $rin = '',$rout;
    # vec($rin,fileno(STDIN),1) = 1;
    # $ein = $rin;
    # my $nfound = select($rout=$rin,undef,undef,undef);

    my $buf = "";
    syslog LOG_INFO,"waiting for packet";
    my $nread = sysread STDIN,$buf,2;
    do { syslog LOG_INFO,"port closed"; exit; } unless $nread == 2;
    my $len = unpack "n",$buf;
    my $nread = sysread STDIN,$buf,$len;

    my ($op,$user,$host,$password) = split /:/,$buf;
    #$user =~ s/\./\//og;
    my $jid = "$user\@$domain";
    my $result;

    syslog(LOG_INFO,"request (%s)", $op);

 SWITCH:
      {
$op eq 'auth' and do
  {
          $result = 1;
  },last SWITCH;

$op eq 'setpass' and do
  {
          $result = 1;
  },last SWITCH;

      $op eq 'isuser' and do
        {
          # password is null. Return 1 if the user $user\@$domain exitst.
          $result = 1;
        },last SWITCH;

      $op eq 'tryregister' and do
        {
          $result = 1;
        },last SWITCH;

      $op eq 'removeuser' and do
        {
          # password is null. Return 1 if the user $user\@$domain exitst.
          $result = 1;
        },last SWITCH;

      $op eq 'removeuser3' and do
        {
          $result = 1;
        },last SWITCH;
    };
    my $out = pack "nn",2,$result ? 1 : 0;
    syswrite STDOUT,$out;
  }

closelog;
```

Python Example Script

Example Python script:

```python
#!/usr/bin/python

import sys
import struct

def read_from_stdin(bytes):
  if hasattr(sys.stdin, 'buffer'):
    return sys.stdin.buffer.read(bytes)
  else:
    return sys.stdin.read(bytes)

def read():
    (pkt_size,) = struct.unpack('>H', read_from_stdin(2))
    pkt = sys.stdin.read(pkt_size)
    cmd = pkt.split(':')[0]
    if cmd == 'auth':
        u, s, p = pkt.split(':', 3)[1:]
        if u == "wrong":
            write(False)
        else:
            write(True)
    elif cmd == 'isuser':
        u, s = pkt.split(':', 2)[1:]
        if u == "wrong":
            write(False)
        else:
            write(True)
    elif cmd == 'setpass':
        u, s, p = pkt.split(':', 3)[1:]
        write(True)
    elif cmd == 'tryregister':
        u, s, p = pkt.split(':', 3)[1:]
        write(True)
    elif cmd == 'removeuser':
        u, s = pkt.split(':', 2)[1:]
        write(True)
    elif cmd == 'removeuser3':
        u, s, p = pkt.split(':', 3)[1:]
        write(True)
    else:
        write(False)
    read()

def write(result):
    if result:
        sys.stdout.write('\x00\x02\x00\x01')
    else:
        sys.stdout.write('\x00\x02\x00\x00')
    sys.stdout.flush()

if __name__ == "__main__":
    try:
        read()
    except struct.error:
        pass
```

PubSub overview

This document describes ejabberd's PubSub architecture to understand how to write custom plugins.

`XEP-0060` (PubSub) is more than 100 pages of specifications, with 12 very detailed use cases with many possibles options and possible situations:

- Subscribe
- Unsubscribe
- Configure subscription
- Retrieve items
- Publish item
- Delete item
- Create node
- Configure node
- Delete node
- Purge node
- Manage subscriptions
- Manage affiliations

`XEP-0163` (PEP) is based on PubSub XEP-0248 (deprecated) for Collection Nodes and uses generic PubSub functionality, specified in XEP-0060.

History

Initial implementation made by Aleksey Shchepin, ability to organise nodes in a tree added by Christophe Romain in 2007. First attempt to create a flexible API for plugins started in 2007, and improved until 2015.

Implementation

PubSub service comes in several parts:

- A poll of iq handlers handled by ejabberd router
- A sending process
- A core router to perform high level actions for every use case
- Plugins to handle nodes, affiliations/subscriptions, and items at lower level and interface with data backend

Nodetree plugins

They handles storage and organisation of PubSub nodes. Called on get, create and delete node. Default implementation includes three plugins:

- tree: (default) both internal and odbc backend.
- virtual: no backend, no configurable nodes.
- dag: handles `XEP-0248` .

If all nodes shares same configuration, I/O on pubsub_node can be avoided using virtual nodetree.

Node plugins

They handle affiliations, subscriptions and items. They provide default node configuration and features. Called on every pubsub use cases. Each plugin is responsible of checks to handle all possibles cases and reply action result to PubSub engine to let it handle the routing. The most common plugins available in default installation are:

- flat: (default) all nodes are in a flat namespace, there are no parent/child nodes

- hometree: all nodes are organized as in a filesystem under /home/hostname/user/...

- pep: handles `XEP-0163`

- dag: handles `XEP-0248` .

- public, private, ... which are derivate of flat, with different default node configuration.

NODE_FLAT

`node_flat` is the default plugin, without node hierarchy, which handles standard PubSub case. The default node configuration with this plugin is:

```
[{deliver_payloads, true},
 {notify_config, false},
 {notify_delete, false},
 {notify_retract, true},
 {purge_offline, false},
 {persist_items, true},
 {max_items, 10},
 {subscribe, true},
 {access_model, open},
 {roster_groups_allowed, []},
 {publish_model, publishers},
 {notification_type, headline},
 {max_payload_size, 60000},
 {send_last_published_item, on_sub_and_presence},
 {deliver_notifications, true},
 {presence_based_delivery, false}].
```

NODE_HOMETREE

node_hometree use exact same features as flat plugin, but organise nodes in a tree following same scheme as path in filesystem. Every user can create nodes in its own home. Each node can contain items and/or sub-nodes. Example:

```
/home/user
/home/domain/user
/home/domain/user/my_node
/home/domain/user/my_node/child_node
```

NODE_PEP

node_pep handles XEP-0163: Personal Eventing Protocol It do not persist items, just keeping last item in memory cache. Node names are raw namespace attached to a given bare JID. Every user can have its own node with a common namespace sharing with others.

NODE_DAG

node_dag handles XEP-0248: PubSub Collection Nodes Contribution from Brian Cully. Every node takes places in a tree and is either a collection node (have only sub-nodes) or a leaf node (contains only items). No restriction on the tree structure

Plugin design

Due to complexity of XEP-0060, PubSub engine do successive calls to nodetree and node plugins in order to check validity, perform corresponding action and return result or appropriate error to users. Plugin design follows this requirement and divide actions by type of data to allow transient backend implementation without any PubSub engine change.
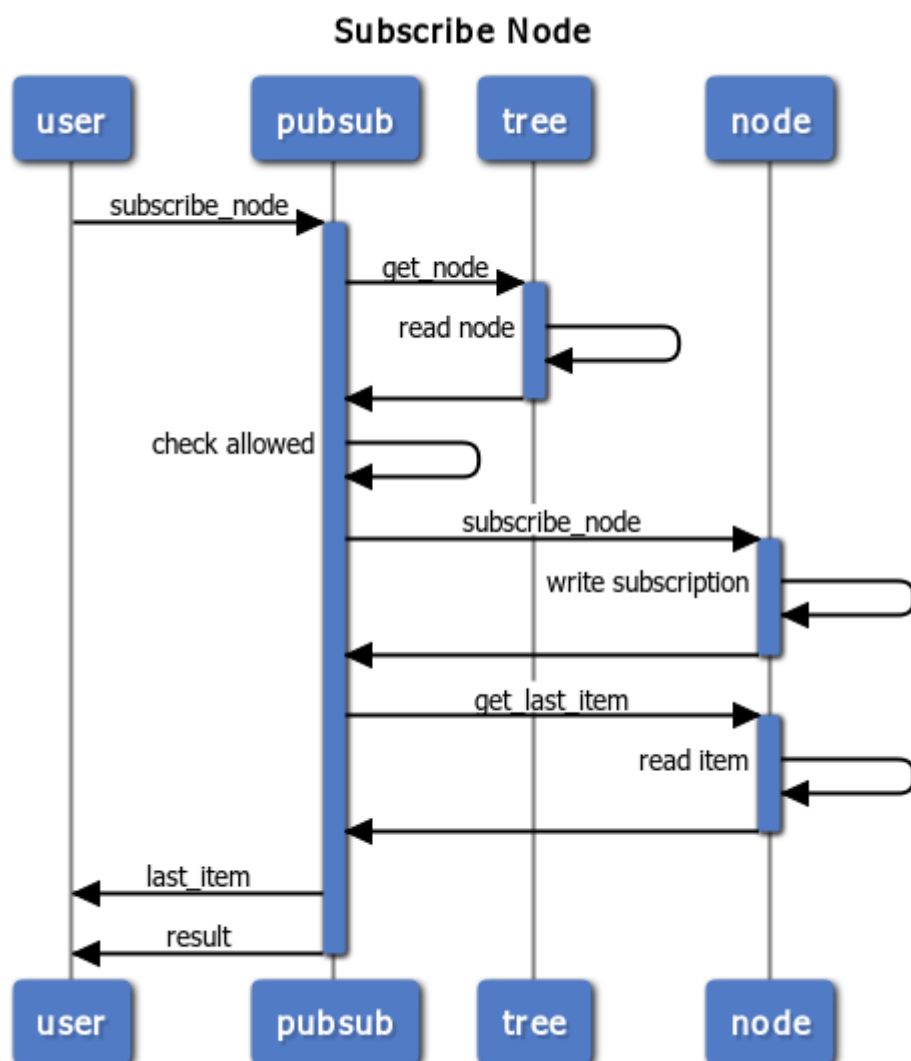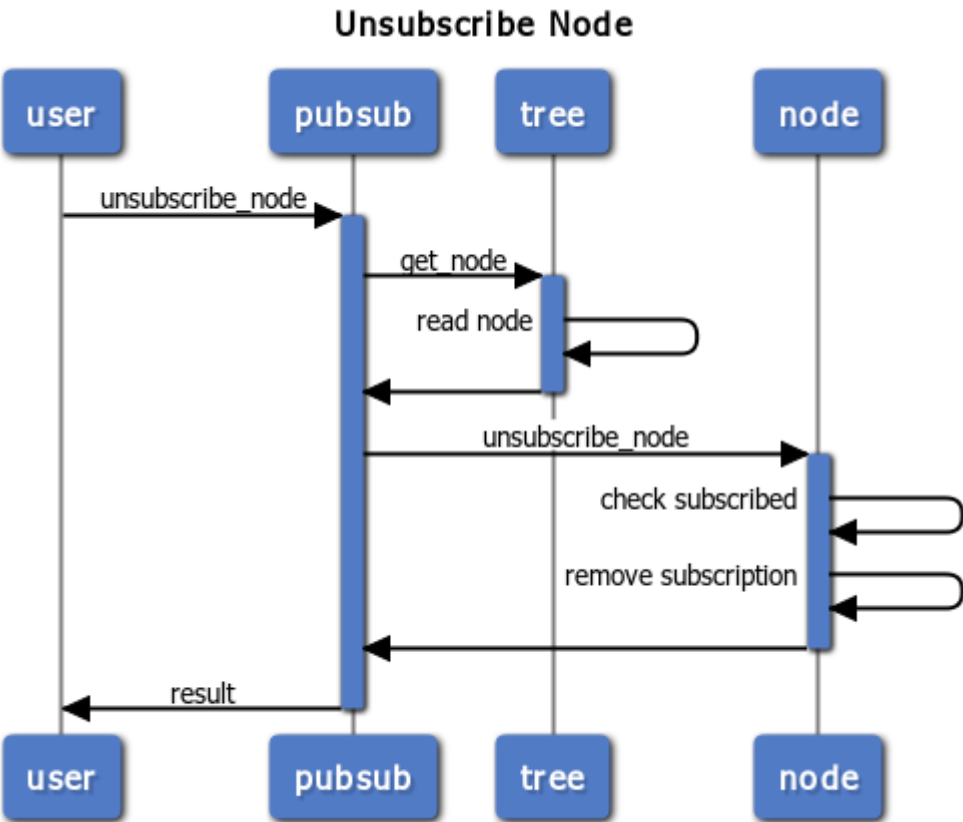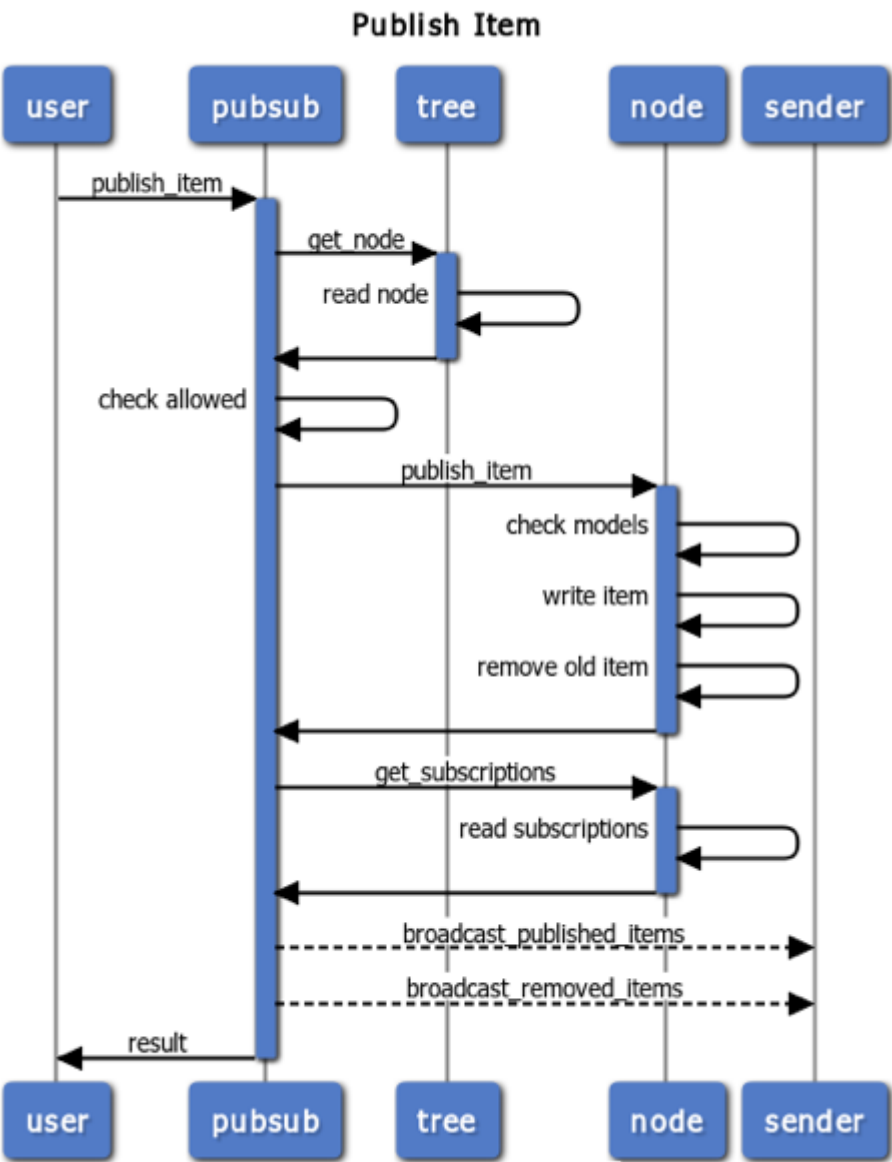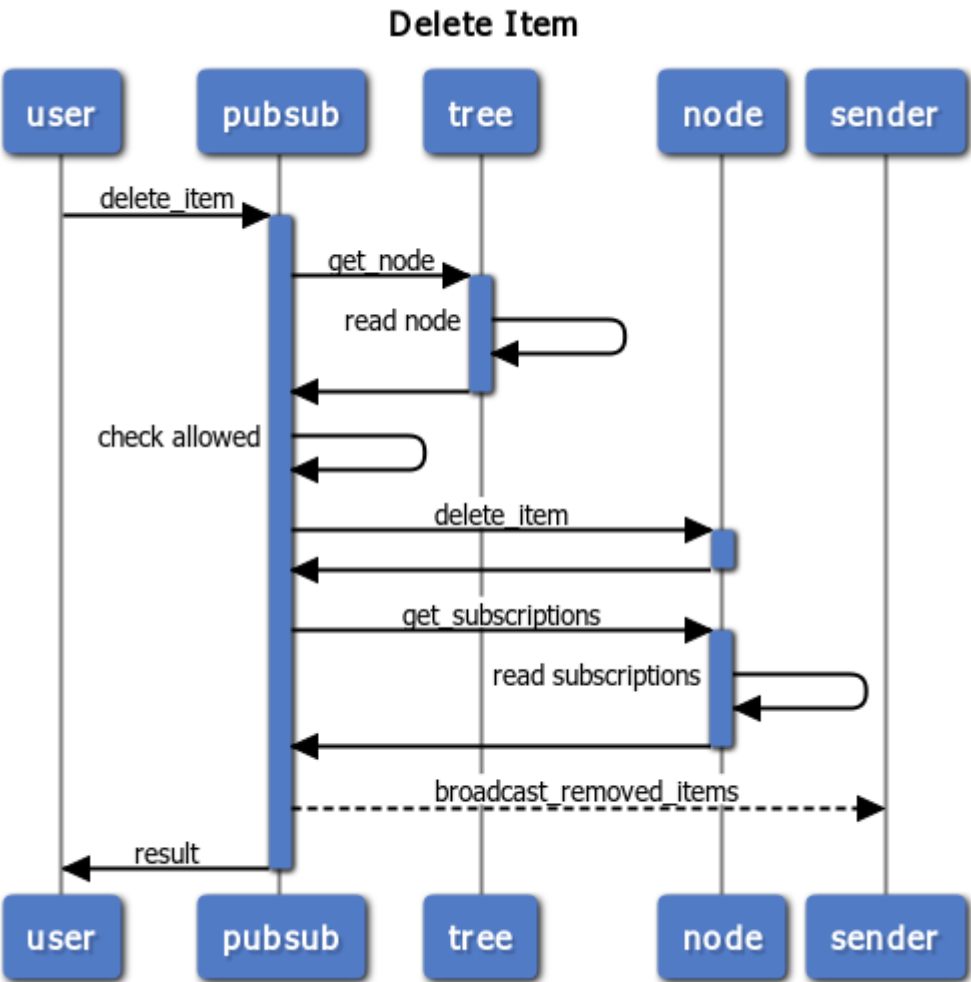
Create Node

Create Node

Delete Node

Delete Node

Subscribe

Subscribe Node

Unsubscribe
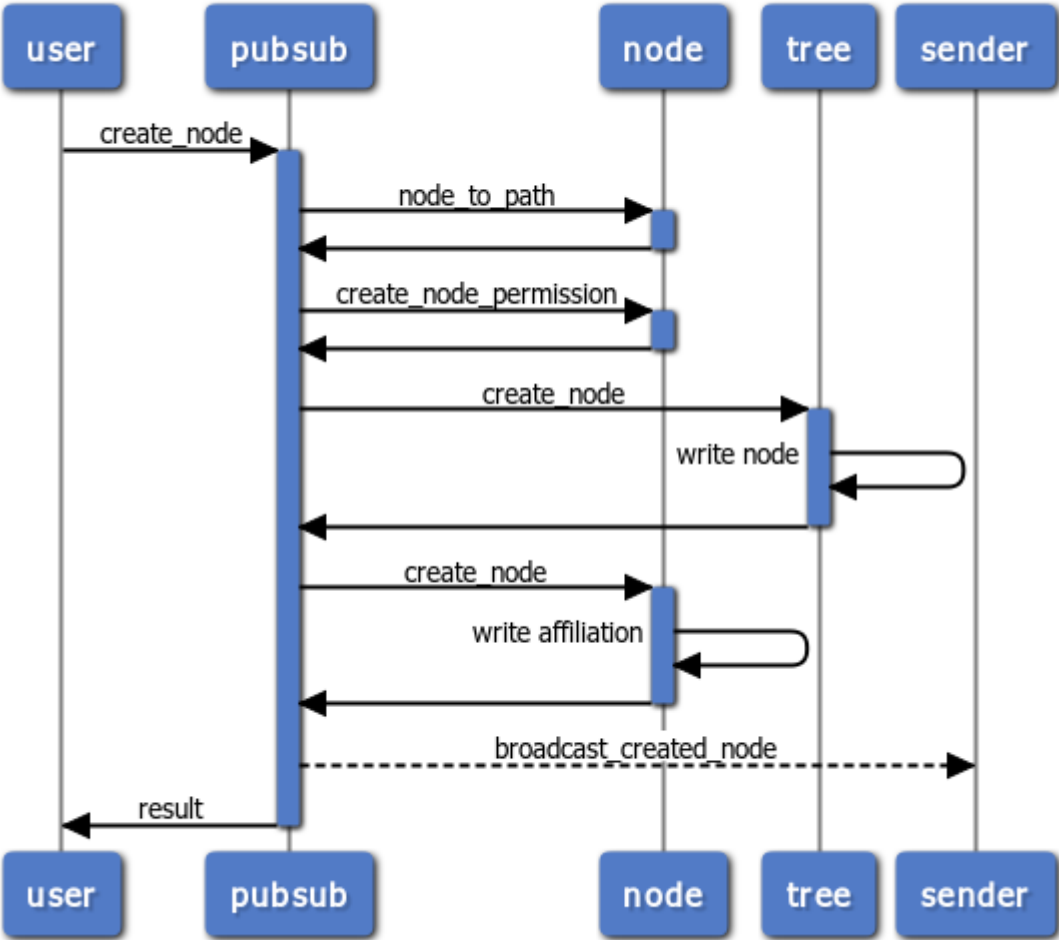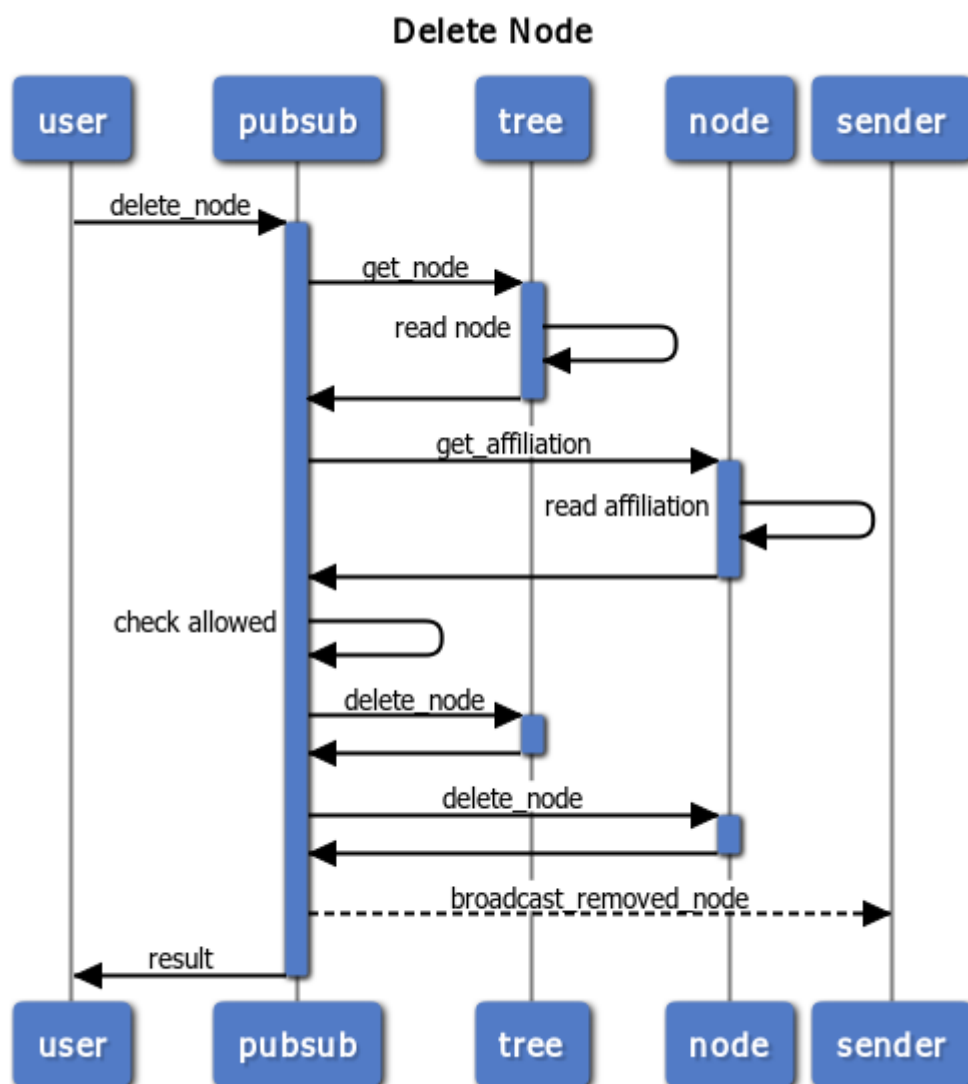
Unsubscribe Node

Publish item
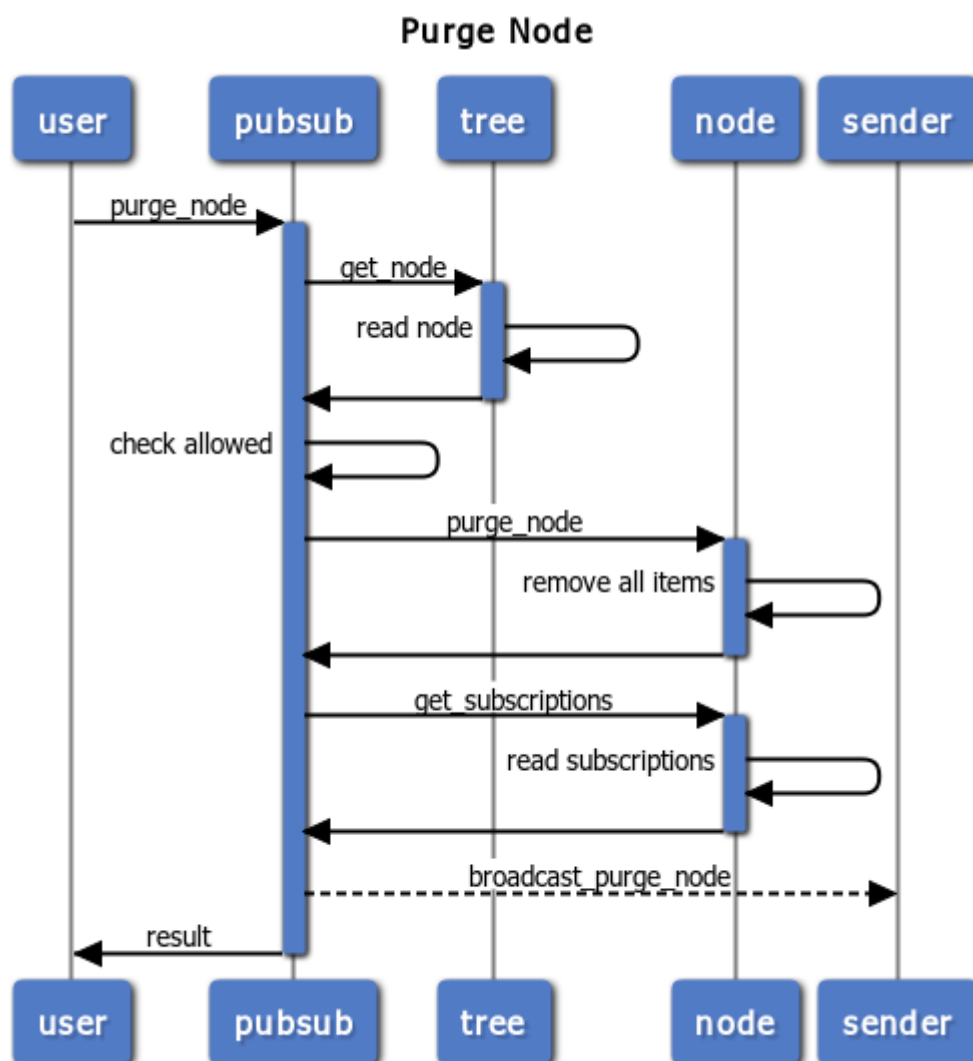
Publish Item
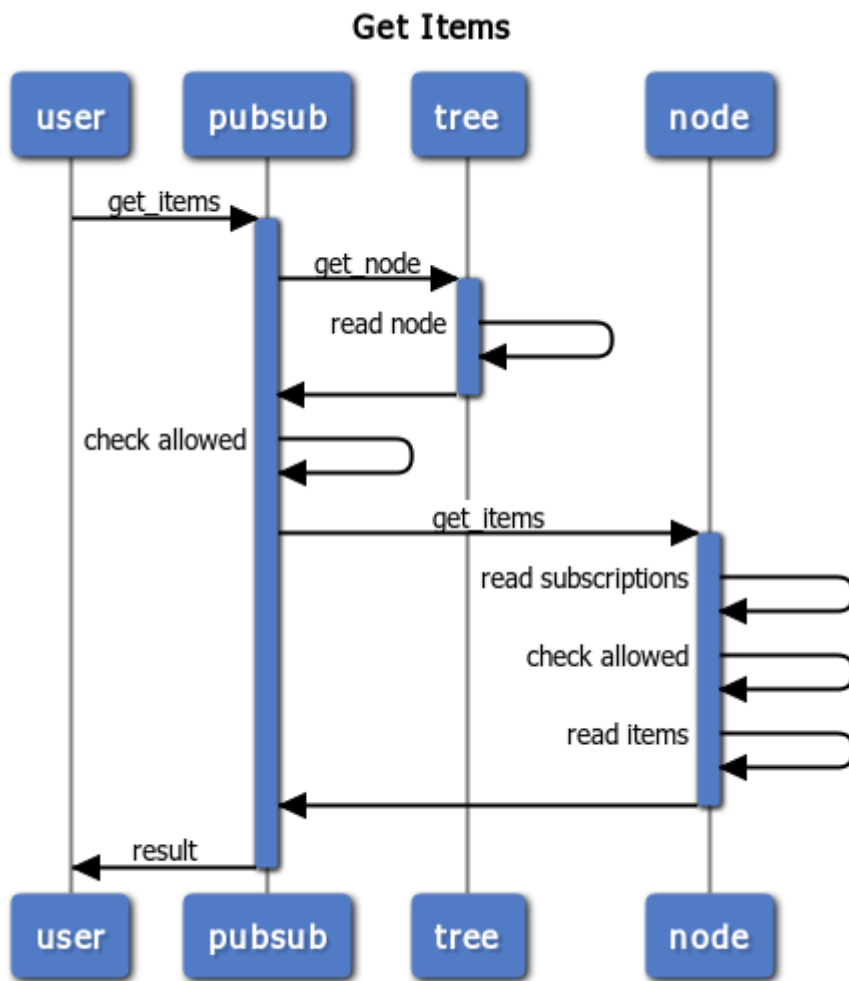
Delete item

Delete Item

Purge Node

Purge Node

Get item

Available backends

Flat, hometree and PEP supports mnesia and SQL backends. Any derivated plugin can support the same (public, private, club, buddy...). Adding backend does not require any PubSub engine change. Plugin just need to comply API. Business Edition also supports optimized ets and mdb.

Customisation

To write your own plugin, you need to implement needed functions:

```
[init/3, terminate/2, options/0, features/0,
 create_node_permission/6, create_node/2, delete_node/1,
 purge_node/2, subscribe_node/8, unsubscribe_node/4,
 publish_item/6, delete_item/4, remove_extra_items/3,
 get_entity_affiliations/2, get_node_affiliations/1,
 get_affiliation/2, set_affiliation/3,
 get_entity_subscriptions/2, get_node_subscriptions/1,
 get_subscriptions/2, set_subscriptions/4,
 get_pending_nodes/2, get_states/1, get_state/2,
 set_state/1, get_items/7, get_items/3, get_item/7,
 get_item/2, set_item/1, get_item_name/3, node_to_path/1,
 path_to_node/1]
```

Generic function must call their corresponding partner in `node_flat`.

Simple plugin would just call `node_flat` and override some defaults such as:

- options/0 and features/0 to match your needs. This triggers the way PubSub controls calls to your plugins.
- create_node_permission/6 for example to check an LDAP directory against an access flag
- Write your own tests on publish or create node, forbids explicit access to items, etc...

Clustering

ejabberd's implementation tends to cover most generic and standard uses. It's good for common use, but far from optimal for edges or specific cases. Nodes, affiliations, subscriptions and items are stored in a replicated database. Each ejabberd node have access to all the data. Each ejabberd node handles part of the load, but keep locking database cluster wide on node records write (pubsub_node) Affiliations, subscriptions and items uses non blocking write (pubsub_state and pubsub_item)

Roster versioning

Roster versioning as implemented currently by ejabberd is a simplified approach to roster versioning.

This is an all-or-nothing approach that does not support the granular diff as explained in RFC-6121.

Our implementation conforms to version 0.6 of XEP-0237, sending the full roster in case of change or empty result if the roster did not change.

As a result, as a client developer, when implementing support for roster versioning, you should expect both the traditional form for returning the roster, with version (iq result) and the incremental roster changes (iq set).

Example

As a summary, here is how you should expect it to work.

First, you can check that the feature is advertised in the `stream:features` as `urn:xmpp:features:rosterver`:

```
<stream:features>
 <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind"/>
 <session xmlns="urn:ietf:params:xml:ns:xmpp-session">
  <optional/>
  </session>
  <c xmlns="http://jabber.org/protocol/caps" node="http://www.process-one.net/en/ejabberd/" ver="/lmQr0llUEtX/pIt+6BDAbnIT/U=" hash="sha-1"/>
  <sm xmlns="urn:xmpp:sm:2"/>
  <sm xmlns="urn:xmpp:sm:3"/>
  <ver xmlns="urn:xmpp:features:rosterver"/>
</stream:features>
```

You can then bootstrap the use of roster versioning using empty `ver` attribute when sending your roster `get` iq:

```
<iq id='roster1' to='myuser@domain.com' type='get'>
 <query xmlns='jabber:iq:roster' ver=''/>
</iq>
```

In return, you get a full roster with the current version:

```
<iq from="myuser@domain.com" type="result" xml:lang="en" to="myuser@domain.com/resource" id="roster1">
 <query xmlns='jabber:iq:roster' ver="81cb523a7b77c7011552be85a3dde55189297590">
  <item subscription="both" jid="contact@domain.com">
   <group>Test</group>
  </item>
  ...
 </query>
</iq>
```

The client can store this version to send subsequent roster queries.

If client send a roster query with reference version it received get an empty iq result meaning the roster did not change:

```
<iq id="roster2" to="myuser@domain.com" type="get">
 <query xmlns='jabber:iq:roster' ver='81cb523a7b77c7011552be85a3dde55189297590'/>
</iq>
```

```
<iq from="myuser@domain.com" type="result" xml:lang="en" to="myuser@domain.com/resource" id="roster2"/>
```

If client send roster query with any other reference version, it will receive the full roster again in the roster iq result.

ejabberd Stanza Routing

Message Routing

In case of a message sent from User A to User B, both of whom are served by the same domain, the flow of the message through the system is as follows:

1. User A's `ejabberd_receiver` receives the stanza and passes it to `ejabberd_c2s` .

2. After some consistency check, `user_send_packet` is called if the stanza is correct.

3. The stanza is matched against any privacy lists in use and, in case of being allowed, routed by `ejabberd_router:route/3` .

4. `ejabberd_router:route/3` runs the `filter_packet` hook. `filter_packet` hook can drop of modify the stanza.

5. `ejabberd_router` will then consult the routing table to know what do to next. It is easier to understand by looking at an example of actual routing table content:

```
(ejabberd@localhost)2> ets:tab2list(route).
[{route,<<"pubsub.localhost">>,
  {apply_fun,#Fun<ejabberd_router.2.122122122>}},
{route,<<"muc.localhost">>,
  {apply_fun,#Fun<mod_muc.2.122122123>}},
{route,<<"localhost">>,{apply,ejabberd_local,route}}]
```

In that case, user is local so we need to route to same domain (in our case localhost). We then can see that we have to call `ejabberd_local:route` to route the message to local user. As both user are local (no server-to-server involved), it matches our expectations.

1. `ejabberd_local` routes the stanza to `ejabberd_sm` given it's got at least a bare JID as the recipient.

2. `ejabberd_sm` determines the available resources of User B, takes into account their session priorities and whether the message is addressed to a particular resource or a bare JID and appropriately replicates (or not) the message and sends it to the recipient's `ejabberd_c2s` process(es).

   In case no resources are available for delivery (hence no `ejabberd_c2s` processes to pass the message to), `offline_message_hook` is run to delegate offline message storage.

1. `ejabberd_c2s` verifies the stanza against any relevant privacy lists and sends it on the user socket if it does exist. In the case of ejabberd Business Edition and ejabberd Saas, session can be detached and push notifications can be used as a fallback. `user_receive_packet` hook is run to notify the rest of the system about stanza delivery to User B.

   Here is a broader diagram, including server-to-server routing:
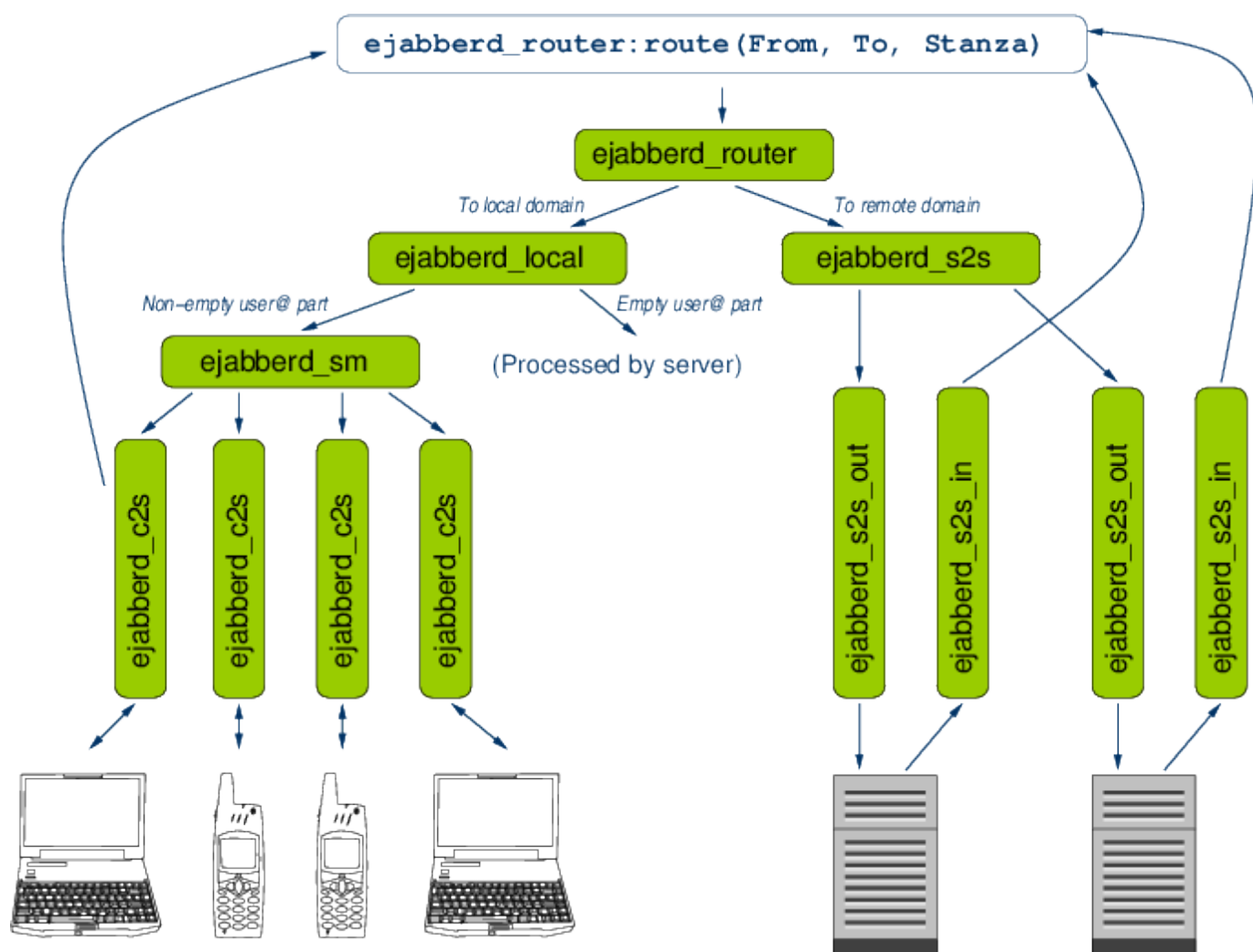
ejabberd SQL Database Schema

We present the tables that might be in use, depending on your server configuration, together with a short explanation of the fields involved and their intended use. Tables are presented roughly grouped by related functionality.

Consider this document a work in progress, not all tables are documented yet.

Latest version of database schema are available in ejabberd Github repository:

- MySQL schema
- Postgres schema
- SQLite schema
- MS SQL Server schema. This schema need testing / feedback and possibly improvement from SQL Server users.

Authentication

Table `users`

Contains the information required to authenticate users.

Field	Type	Usage
username	string	User
password	string	User password, can be hashed
created_at	timestamp	When the user account was created

The password are hashed if you use SCRAM authentication. In that case the next fields are also defined

Field	Type	Usage
serverkey	string	support for salted passwords
salt	string	support for salted passwords
iterationcount	integer	support for salted passwords

Rosters

Table `rosterusers`

This is a quite complex table, used as a store for a quite complex protocol that is the one defined to manage rosters and subscriptions on rfc6121.

In the common case of two users adding each other as contacts, entries in the roster table follows a series of steps as they moves from a subscription request to the final approval and bi-directional subscription being established. This process can be initiated either by the user, or by the (possible remote) peer. Also need to account for the case where the user, or the contact, might not be online at the moment of the subscription request is made.

Steps are further complicated by the fact that entries in the roster aren't required to have corresponding subscriptions. For details of the meaning of the different fields, refer to the protocol itself, as these are mostly a direct mapping of it.

Note: If you manage users contacts from outside the roster workflow of XMPP (for example your site backends perform the linking between users), it is likely that you only need to care about the username, jid and nick fields, and set the subscription field to be always 'B' for a mutual link between users.

Field	Type	Usage
username	string	User
jid	string	Contact jid
nick	string	Contact nickname
subscription	char	'B'=both | 'T'=To | 'F'=From | 'N'=none
ask	char	'S'=subscribe | 'U'=unsubscribe | B='both' | 'O'=out | 'I'=in | 'N'=none
askmessage	string	Message to be displayed on the subscription request
server	char	'N' for normal users contacts
subscribe	string	
type	string	"item"
created_at	timestamp	Creation date of this roster entry

Table `rostergroups`

Table `sr_group`

Table `sr_user`

Messages

Table `spool`

Messages sent to users that are offline are stored in this table. Do not confuse this with general message archiving: messages are only temporarily stored in this table, removed as soon as the target user is back online and the pending messages delivered to it.

Field	Type	Usage
username	string	User
xml	blob	Raw packet
seq	integer	Unique, autoincrement sequence number.
created_at	timestamp	When the message was stored

The seq field is used for sorting, and to easily identify a particular user message.

Table `privacy_list_data`

The table is used to store privacy rules.

The table is a direct translation of the XMPP packet used to set privacy lists. For more details, please read XEP-0016: Privacy Lists, Syntax and Semantics. Here is an example packet coming from privacy list specification:

```
<item
    type='[jid|group|subscription]'
    value='bar'
    action='[allow|deny]'
    order='unsignedInt'>
  [<message/>]
```

```
        [<presence-in/>]
        [<presence-out/>]
        [<iq/>]
</item>
```

The table fields are defined as follow:

Field	Type	Usage
id	int	Privacy list rule id.
t	char	Privacy rule type: 'j' for jid, 'g' for group and 's' for subscription.
value	string	Privacy list value for match, whose content depends on privacy list rule type.
action	char	Privacy list action: 'd' for deny and 'a' for allow.
ord	int	Order for applying the privacy list rule.
match_all	boolean (0 or 1)	If true (1), means any packet types will be matched. Other matches should be false (0).
match_iq	boolean (0 or 1)	If true (1), means iq packets will be matched by rule.
match_message	boolean (0 or 1)	If true (1), means message packets type will be matched by rule.
match_presence_in	boolean (0 or 1)	If true (1), means inbound presence packets type will be matched by rule.
match_presence_out	boolean (0 or 1)	If true (1), means outbound packets type will be matched by rule.

Multiuser Chat Rooms

Table `muc_room`

It is used to store *persistent* rooms, that is, rooms that must be automatically started with the server.

Field	Type	Usage
name	string	Room name
host	string	Hostname of the conference component
opts	string	Room options, encoded as erlang terms
created_at	timestamp	Creation date

The opts field is legible, but not mean to be modified directly. It contents depends on the implementation of mod_muc. It contains the room configuration and affiliations.

Table `muc_registered`

Contains a map of user to nicknames. When a user register a nickname with the conference module, that nick is reserved and can't be used by anyone else, in any room from that conference host.

Field	Type	Usage
jid	string	User jid
host	string	Hostname of the conference component
nick	string	Room options, encoded as erlang terms
created_at	timestamp	Creation date

Table `room_history`

In ejabberd Business Edition, this table is used if persistent room history is enabled. If so, recent room history is saved to the DB before ejabberd is stopped, allowing the recent history to survive server restarts.

Field	Type	Usage
room	string	Room jid
nick	string	Nickname that sent the message
packet	string	XML stanza with the message
have_subject	boolean	True if the message stanza had subject
created_at	timestamp	Creation date
size	integer	Size in bytes of the xml packet

Table `muc_online_room`

This table is used to store rooms that actually exists in the memory of the server.

Field	Type	Usage
name	string	Room name
host	string	Hostname of the conference component
node	string	Erlang node where the room is
pid	string	Pid of the thread running the room

Table `muc_online_users`

This table is used to store MucSub subscriptions.

Field	Type	Usage
username	string	User
server	string	Hostname of the user
resource	string	User resource
name	string	Name of the room
host	string	Hostname of the conference component
node	string	Erlang node

Table `muc_room_subscribers`

This table is used to store MucSub subscriptions.

Field	Type	Usage
room	string	Room name
host	string	Hostname of the conference component
jid	string	User jid
nick	string	User nick
nodes	string	MucSub nodes
created_at	timestamp	Creation date

VCard

Table `vcard`

The table is used to store raw vCard content for delivery of the vCard "as is".

The table fields are defined as follow:

Field	Type	Usage
username	string	Owner of the Vcard
vcard	text	Raw Vcard
created_at	timestamp	Record creation date

Table `vcard_search`

The table is used to store vCard index on a few of the Vcard field used for vCard search in users directory.

You can learn more about the vCard specification on Wikipedia vCard page.

The table fields are defined as follow:

Field	Type	Usage
username	string	Raw username for display
lusername	string	Lowercase username for search
fn	string	Raw fullname for display
lfn	string	Lowercase fullname for search
family	string	Raw family name for display
lfamilly	string	Lowercase family name for search
given	string	Raw given name for display
lgiven	string	Lowercase given name for search
middle	string	Raw middle name for display
lmiddle	string	Lowercase middle name for search
nickname	string	Raw nickname for display
lnickname	string	Lowercase nickname for search
bday	string	Raw birthday for display
lbday	string	Lowercase and processed birthday for search
ctry	string	Raw country for display
lctry	string	Lowercase country for search
locality	string	Raw city for display
llocality	string	Lowercase city for search
email	string	Raw email for display
lemail	string	Lowercase email for search
orgname	string	Raw organisation name for display
lorgname	string	Lowercase organisation name for search
orgunit	string	Raw organisation department name for display
lorgunit	string	Lowercase organisation department for search

Others

Table `last`

This table is used to store the last time the user was seen online. It is defined as follow:

Field	Type	Usage
username	string	User
seconds	string	Timestamp for the last time the user was seen online
state	string	Why user got disconnected. Usually is empty

Note that the table is *not* updated while the user has the session open.

Table `caps_features`

Ejabberd uses this table to keep a list of the entity capabilities discovered.

Field	Type	Usage
node	string	Node
subnode	string	Subnode
feature	string	Entity feature
created_at	timestamp	Creation date

The subnode field correspond to the 'ver' ("verification string") of XEP-0115. There is one entry in this table for each feature advertised by the given (node,subnode) pair.

Table `private_storage`

Used for user private data storage.

Field	Type	Usage
username	string	User
namespace	string	XEP-0049 namespace of the stored data
data	string	Raw xml
created_at	timestamp	Creation date

External authentication

There are examples of external authentication scripts in many different languages in the page: `https://ejabberd.im/extauth`

Main contribution repository

Check also the contributions hosted in the `ejabberd-contrib Github repository` .

ejabberd API libraries

Here is a ejabberd API implementations allowing to ease ejabberd integration with your own backends:

- `Pyejabberd` : Client library for ejabberd XMLRPC API, in Python, by Dirkmoors, MIT license. See `https://pypi.python.org/pypi/pyejabberd` and `https://github.com/dirkmoors/pyejabberd`

Old / obsolete contributions

Finally, there is an old list of contributions that were developed for ejabberd 2.x in: `https://ejabberd.im/contributions`

Contributing to ejabberd

We'd love for you to contribute to our source code and to make ejabberd even better than it is today! Here are the guidelines we'd like you to follow:

- Code of Conduct
- Questions and Problems
- Issues and Bugs
- Feature Requests
- Issue Submission Guidelines
- Pull Request Submission Guidelines
- Signing the CLA

Code of Conduct

Help us keep ejabberd community open-minded and inclusive. Please read and follow our Code of Conduct.

Questions, Bugs, Features

Got a Question or Problem?

Do not open issues for general support questions as we want to keep GitHub issues for bug reports and feature requests. You've got much better chances of getting your question answered on dedicated support platforms, the best being Stack Overflow.

Stack Overflow is a much better place to ask questions since:

- there are thousands of people willing to help on Stack Overflow
- questions and answers stay available for public viewing so your question / answer might help someone else
- Stack Overflow's voting system assures that the best answers are prominently visible.

To save your and our time, we will systematically close all issues that are requests for general support and redirect people to the section you are reading right now.

Other channels for support are: - ejabberd XMPP room: ejabberd@conference.process-one.net - ejabberd XMPP room logs - ejabberd Mailing List

Found an Issue or Bug?

If you find a bug in the source code, you can help us by submitting an issue to our GitHub Repository. Even better, you can submit a Pull Request with a fix.

Missing a Feature?

You can request a new feature by submitting an issue to our GitHub Repository.

If you would like to implement a new feature then consider what kind of change it is:

- **Major Changes** that you wish to contribute to the project should be discussed first in an GitHub issue that clearly outlines the changes and benefits of the feature.
- **Small Changes** can directly be crafted and submitted to the GitHub Repository as a Pull Request. See the section about Pull Request Submission Guidelines.

Issue Submission Guidelines

Before you submit your issue search the archive, maybe your question was already answered.

If your issue appears to be a bug, and hasn't been reported, open a new issue. Help us to maximize the effort we can spend fixing issues and adding new features, by not reporting duplicate issues.

The "new issue" form contains a number of prompts that you should fill out to make it easier to understand and categorize the issue.

Pull Request Submission Guidelines

By submitting a pull request for a code or doc contribution, you need to have the right to grant your contribution's copyright license to ProcessOne. Please check ProcessOne CLA for details.

Before you submit your pull request consider the following guidelines:

- Search GitHub for an open or closed Pull Request that relates to your submission. You don't want to duplicate effort.
- Create the development environment
- Make your changes in a new git branch:

```
git checkout -b my-fix-branch master
```

* Test your changes and, if relevant, expand the automated test suite. * Create your patch commit, including appropriate test cases. * If the changes affect public APIs, change or add relevant documentation. * Commit your changes using a descriptive commit message.

```
git commit -a
```

Note: the optional commit `-a` command line option will automatically "add" and "rm" edited files.

- Push your branch to GitHub:

```
git push origin my-fix-branch
```

- In GitHub, send a pull request to `ejabberd:master`. This will trigger the automated testing. We will also notify you if you have not yet signed the contribution agreement.
- If you find that the tests have failed, look into the logs to find out if your changes caused test failures, the commit message was malformed etc. If you find that the tests failed or times out for unrelated reasons, you can ping a team member so that the build can be restarted.
- If we suggest changes, then:
- Make the required updates.
- Test your changes and test cases.
- Commit your changes to your branch (e.g. `my-fix-branch`).
- Push the changes to your GitHub repository (this will update your Pull Request).

  You can also amend the initial commits and force push them to the branch.

```
git rebase master -i
git push origin my-fix-branch -f
```

  This is generally easier to follow, but separate commits are useful if the Pull Request contains iterations that might be interesting to see side-by-side.

That's it! Thank you for your contribution!

Signing the Contributor License Agreement (CLA)

Upon submitting a Pull Request, we will ask you to sign our CLA if you haven't done so before. It's a quick process, we promise, and you will be able to do it all online

You can read ProcessOne Contribution License Agreement in PDF.

This is part of the legal framework of the open-source ecosystem that adds some red tape, but protects both the contributor and the company / foundation behind the project. It also gives us the option to relicense the code with a more permissive license in the future.

Contributor Covenant Code of Conduct

Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at the email address: conduct AT process-one.net. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at http://contributor-covenant.org/version/1/4

Contributors

We would like to thanks official ejabberd source code contributors:

- Sergey Abramyan
- Badlop
- Ludovic Bocquet
- Emilio Bustos
- Thiago Camargo
- Juan Pablo Carlino
- Paweł Chmielowski
- Gabriel Gatu
- Tsukasa Hamano
- Konstantinos Kallas
- Evgeny Khramtsov
- Ben Langfeld
- Peter Lemenkov
- Anna Mukharram
- Johan Oudinet
- Pablo Polvorin
- Mickaël Rémond
- Matthias Rieber
- Rafael Roemhild
- Christophe Romain
- Jérôme Sautret
- Sonny Scroggin
- Alexey Shchepin
- Shelley Shyan
- Radoslaw Szymczyszyn
- Stu Tomlinson
- Christian Ulrich
- Holger Weiß

Please, if you think we are missing your contribution, do not hesitate to contact us at ProcessOne. In case you do not want to appear in this list, please, let us know as well.

Thanks !

Understanding ejabberd and its dependencies

We wanted to make sure that ejabberd is modular and that parts that can be of interest for other Erlang projects can be reused.

Not only we are massive open source contributors to Erlang community and ecosystem, but we are also trying to help even more by reviewing your pull requests. Do not hesitate to submit some on any of the many repositories mentioned here.

ejabberd dependencies

ejabberd code based is split among several repositories so effectively ejabberd code is much more than what is in its primary repository.

Required dependencies

The main ejabberd repository is processone/ejabberd

There is hundreds of forks, but we actively maintain ejabberd to make it the most reliable and up to date version. This is thus your best starting point.

When you build ejabberd yourself, the build chain will download a few Erlang dependencies:

- processone/cache_tab: Flexible in-memory Erlang caching module.
- processone/iconv: Native iconv driver for Erlang. This is use for fast character encoding conversion.
- processone/fast_xml: Fast native Expat based Erlang XML parsing library. XML is the core of XMPP so we needed to provide the fast and more robust XML parsing stack as possible. It means that if you are looking for a great XML parser, reusing p1_xml is probably a great idea.
- processone/stringprep: Fast and efficient Erlang Stringprep native driver. Stringprep is heavily used in XMPP to define encoding rules of various XMPP objects.
- processone/fast_yaml: Native Erlang interface to libyaml, for fast robust YAML parsing. This is needed by our new config file format.
- processone/ezlib: Native zlib driver for Erlang. Used for fast / efficient stream compression.
- processone/fast_tls: Erlang native driver for TLS / SSL. It is build for performance and is more scalable that Erlang SSL driver. If your Erlang server is handling heavy load and is using TLS, we strongly recommend you check / compare with this driver.
- processone/esip: ProcessOne SIP protocol support to add SIP-based voice capabilities to ejabberd.
- processone/stun: Implementation of Session Traversal Utilities for NAT. It is used for XMPP and SIP media capabilities, to help client discover their visible IP address and allow them to get in touch through NAT. This is basically useful for voice and file transfers.
- processone/p1_utils: This is extra Erlang modules developed for ejabberd but that can be useful in other Erlang applications.
- processone/p1_logger: Logger wrapper to allow selecting your preferred logger at build time.
- basho/lager: Erlang logger module.
- DeadZen/goldrush: Small Erlang app that provides fast event stream processing. It is used by lager.
- vaxelfel/eHyperLogLog: HyperLogLog, a distinct values estimator, in Erlang. Used for analytics.

Optional dependencies

Then, we use a few other dependent repositories that may be used if you have enabled support in the configure script.

Here are the dependencies for relational database support:

- processone/mysql: Pure Erlang MySQL driver.
- processone/pgsql: Pure Erlang PostgreSQL driver

Here are the dependencies for Elixir support:

- elixir-lang/elixir: Used to write ejabberd modules in Elixir programming language.
- yrashk/rebar_elixir_plugin: Plugin for rebar build tool to support Elixir modules compilation.

After you have build ejabberd from source, you will find all the dependencies downloaded and build in the *deps* directory.

As you see, there is much more Erlang code produce at ProcessOne and contributed to the Erlang community than just ejabberd repository.

ejabberd contributions

This is not dependencies, but simply modules that you may find nice to add to your ejabberd deployment.

We attempted to gather some of the more useful ejabberd modules in a contribution repository to ease discovery.

This repository is available on Github: ejabberd-contribs

We are thinking about a better approach for ejabberd contributions discovery. More on that soon.

ejabberd Docs Source Code

The ejabberd Community Server has its source code available in the ejabberd git repository. Its documentation is published in the ejabberd Docs website, and its source code is available in the docs git repository.

This is a community effort and you are welcome to submit issues or pull requests in order to improve the docs and benefit the ejabberd community.

This documentation site is built using MkDocs and Material for MkDocs.

Installation

To build the site you need Python 3.6 or later, then install the dependencies:

pip

```
mkdir -p .venv
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

> **info**
>
> From now on, remember to run `source .venv/bin/activate` before running any `mkdocs [...]` command.

> **Tip**
>
> You can freeze the dependencies to a file using `pip freeze > requirements.txt`.

Debian

You could install most dependencies using APT:

```
apt-get install mkdocs \
                mkdocs-material \
                mkdocs-material-extensions \
                mkdocs-redirects \
                python3-bs4
```

> **Warning**
>
> Unfortunately Debian doesn't package `mkdocs-with-pdf`, so you should remove `with-pdf` plugin from `mkdocs.yml`.

Building

Now you can start a small webserver that builds the site dynamically:

```
mkdocs serve
```

or build the site into static html files in the `site/` directory:

```
mkdocs build
```

Testing

To verify the internal URLs in the site:

```
TEST=true mkdocs serve
```

To verify the internal URLs and also the external links:

```
TEST=true TEST_EXTERNAL=true mkdocs serve
```

Updating content

Some pages in this documentation are extracted from a running ejabberd node:

- admin/configuration/toplevel.md
- admin/configuration/modules.md
- developer/ejabberd-api/admin-api.md
- developer/ejabberd-api/admin-tags.md

To update those pages, install ejabberd, start it and run `make all` in this repository. This gets documentation from ejabberd, processes it to obtain markdown content and moves the files to this repository.

Additionally, there are several other pages that are markdown files copied from ejabberd git repository and docker-ejabberd git repository. Those repositories must be available next to docs.ejabberd.im before running `make all`.

Markdown Shorthands

When editing ejabberd source code to document top-level options, modules or API commands, there is some additional syntax supported to generate HTML URLs:

For example, this text in the ejabberd source code:

```
_`mod_muc_admin`_
_`bookmarks_to_pep`_ API
_`default_db`_
_`basic.md#captcha|CAPTCHA`_
https://xmpp.org/extensions/xep-0045.html[XEP-0045]
```

gets converted into this markdown:

```
[mod_muc_admin](../../admin/configuration/modules.md#mod_muc_admin)
[bookmarks_to_pep](../../developer/ejabberd-api/admin-api.md#bookmarks_to_pep) API
[default_db](toplevel.md#default_db)
[CAPTCHA](basic.md#captcha)
[XEP-0045](https://xmpp.org/extensions/xep-0045.html)
```

There are example usage of those shorthands in ejabberd, for example in `mod_muc.erl`.

ejabberd for Elixir Developers

🏷 improved in 21.07

Building ejabberd with Mix

You can build ejabberd with Elixir `mix` tool. This allows ejabberd to use Elixir libraries and ejabberd modules written in Elixir.

Please note: Elixir 1.10.3 or higher is required to build a release. Also, if using Erlang/OTP 24, then Elixir 1.11.4 or higher is required.

1. Make sure you have the requirements installed. On MacOS you need to use Homebrew and set up your environment.

2. Clone ejabberd project from Github:

```
git clone https://github.com/processone/ejabberd.git
cd ejabberd
```

3. Compile ejabberd:

```
./autogen.sh
./configure --with-rebar=mix
make
```

4. Build a development release:

```
make dev
```

5. There are many ways to start ejabberd, using the `ejabberdctl` or `ejabberd` scripts:

```
_build/prod/rel/ejabberd/bin/ejabberdctl iexlive
_build/prod/rel/ejabberd/bin/ejabberdctl live
_build/prod/rel/ejabberd/bin/ejabberd start_iex
```

6. You should see that ejabberd is properly started:

```
Erlang/OTP 23 [erts-11.1.8] [source] [64-bit] [smp:2:2] [ds:2:2:10] [async-threads:1]

2021-08-03 13:37:36.561603+02:00 [info] Loading configuration from /home/bernar/e/git/ejabberd/_build/dev/rel/ejabberd/etc/ejabberd/ejabberd.yml
2021-08-03 13:37:37.541688+02:00 [info] Configuration loaded successfully
...
2021-08-03 13:37:40.201590+02:00 [info] ejabberd 21.7.9 is started in the node ejabberd@atenea in 3.86s
2021-08-03 13:37:40.203678+02:00 [info] Start accepting TCP connections at [::]:5222 for ejabberd_c2s

Interactive Elixir (1.10.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(ejabberd@localhost)1>
```

7. Now that ejabberd starts correctly, adapt to your needs the default ejabberd configuration file located at `_build/dev/rel/ejabberd/etc/ejabberd/ejabberd.yml` For example, enable this example Elixir ejabberd module:

```
modules:
  'ModPresenceDemo': {}
  mod_adhoc: {}
```

Embed ejabberd in an elixir app

ejabberd is available as an Hex.pm application: ejabberd on hex.pm.

This means you can build a customized XMPP messaging platform with Elixir on top of ejabberd by leveraging ejabberd code base in your app and providing only your custom modules. This makes the management of your ejabberd plugins easier and cleaner.

To create your own application depending on ejabberd, you can go through the following steps:

1. Create a new Elixir app using `mix` :

```
mix new ejapp
cd ejapp
```

2. Add ejabberd package as a dependency in your `mix.exs` file:

```
defmodule Ejapp.MixProject do
  defp deps do
    [
      {:ejabberd, "~> 21.7"}
    ]
  end
end
```

3. Compile everything:

```
mix do deps.get, compile
```

4. Create paths and files for ejabberd:

```
mkdir config
mkdir logs
mkdir mnesia
wget -O config/ejabberd.yml https://raw.githubusercontent.com/processone/ejabberd/master/ejabberd.yml.example
```

5. Define those paths in `config/config.exs` :

```
import Config
config :ejabberd,
  file: "config/ejabberd.yml",
  log_path: 'logs/ejabberd.log'
config :mnesia,
  dir: 'mnesia/'
```

6. Start your app, ejabberd will be started as a dependency:

```
iex -S mix
```

7. You should see that ejabberd is properly started:

```
Erlang/OTP 23 [erts-11.1.8] [source] [64-bit] [smp:2:2] [ds:2:2:10] [async-threads:1]

Compiling 1 file (.ex)
Generated ejapp app

17:58:35.955 [info]  Loading configuration from config/ejabberd.yml

17:58:36.459 [info]  Configuration loaded successfully
...
17:58:39.897 [info]  ejabberd 21.7.0 is started in the node :nonode@nohost in 4.07s
...
17:58:39.908 [info]  Start accepting TCP connections at [::]:5222 for :ejabberd_c2s

Interactive Elixir (1.10.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)>
```

8. Register user from Elixir console:

```
:ejabberd_auth.try_register("test", "localhost", "passw0rd")
```

9. You are all set, you can now connect with an XMPP client !

Call elixir code in erlang code

It's possible to use Elixir libraries in an Erlang module, both the ones included in Elixir, or any other you add as a dependency.

This simple example invokes Elixir's String.duplicate/2 function as shown in one of its documentation examples, and uses the result in the ejabberd vCard nickname field:

```
--- a/src/mod_vcard.erl
+++ b/src/mod_vcard.erl
@@ -209,6 +209,7 @@ process_local_iq(#iq{type = get, to = To, lang = Lang} = IQ) ->
```

```
        VCard = case mod_vcard_opt:vcard(ServerHost) of
     undefined ->
         #vcard_temp{fn = <<"ejabberd">>,
+        nickname = 'Elixir.String':duplicate(<<"abc">>, 2),
         url = ejabberd_config:get_uri(),
         desc = misc:get_descr(Lang, ?T("Erlang XMPP Server")),
         bday = <<"2002-11-16">>};
```

Notice that the elixir code:

```
String.duplicate("abc", 2)
```

is written in erlang as:

```
'Elixir.String':duplicate(<<"abc">>, 2),
```

Check Erlang/Elixir Syntax: A Crash Course for details.

Use elixir library in erlang code

This example demonstrates how to add an elixir library as a dependency in ejabberd, and use it in an ejabberd module written in erlang.

It will use QRCodeEx elixir library to build a QR code of ejabberd's URI and return it as the server vCard photo.

First add the dependency to `mix.exs`:

```
--- a/mix.exs
+++ b/mix.exs
@@ -46,7 +46,7 @@ defmodule Ejabberd.MixProject do
                      :p1_utils, :stringprep, :yconf],
     included_applications: [:mnesia, :os_mon,
                             :cache_tab, :eimp, :mqtree, :p1_acme,
-                            :p1_oauth2, :pkix, :xmpp]
+                            :p1_oauth2, :pkix, :xmpp, :qrcode_ex]
        ++ cond_apps()]
   end

@@ -113,6 +113,7 @@ defmodule Ejabberd.MixProject do
        {:p1_oauth2, "~> 0.6"},
        {:p1_utils, "~> 1.0"},
        {:pkix, "~> 1.0"},
+       {:qrcode_ex, "~> 0.1.1"},
        {:stringprep, ">= 1.0.26"},
        {:xmpp, "~> 1.5"},
        {:yconf, "~> 1.0"}]
```

Then call QRCodeEx.encode/2, QRCodeEx.png/2, and provide the result as the photo in the server vcard:

```
--- a/src/mod_vcard.erl
+++ b/src/mod_vcard.erl
@@ -206,9 +206,13 @@ process_local_iq(#iq{type = set, lang = Lang} = IQ) ->
     xmpp:make_error(IQ, xmpp:err_not_allowed(Txt, Lang));
 process_local_iq(#iq{type = get, to = To, lang = Lang} = IQ) ->
     ServerHost = ejabberd_router:host_of_route(To#jid.lserver),
+    PhotoEncoded = 'Elixir.QRCodeEx':encode(ejabberd_config:get_uri()),
+    PhotoBin = 'Elixir.QRCodeEx':png(PhotoEncoded, [{color, <<17, 120, 0>>}]),
+    PhotoEl = #vcard_photo{type = <<"image/png">>, binval = PhotoBin},
     VCard = case mod_vcard_opt:vcard(ServerHost) of
     undefined ->
         #vcard_temp{fn = <<"ejabberd">>,
+        photo = PhotoEl,
         url = ejabberd_config:get_uri(),
         desc = misc:get_descr(Lang, ?T("Erlang XMPP Server")),
         bday = <<"2002-11-16">>};
```

Write ejabberd module in elixir

If you plan to write an ejabberd module that heavily depends on Elixir dependencies, you may want to write it in elixir from scratch.

The Elixir source code is placed in the ejabberd's lib/ path. Any elixir module placed in `lib/` will be compiled by Mix, installed with all the other erlang modules, and available for you to use.

As you can see, there's a file named mod_presence_demo.ex which defines an ejabberd module written in elixir called `ModPresenceDemo`. To enable `ModPresenceDemo`, add it to `ejabberd.yml` like this:

```
modules:
  'Elixir.ModPresenceDemo': {}
```

Let's write a new ejabberd module in elixir, add it to ejabberd's source code, compile and install it. This example module requires the QRCodeEx Elixir library, and adds a simple web page that generates QR code of any given JID.

1. Copy the mod_qrcode.ex source code to ejabberd's `lib/` path:

```
lib/mod_qrcode.ex
```

2. Recompile and reinstall ejabberd.

3. Enable the module in `ejabberd.yml`:

```
listen:
  -
    port: 5280
    request_handlers:
      /qrcode: 'Elixir.ModQrcode'

modules:
  'Elixir.ModQrcode': {}
```

4. When restarting ejabberd, it will show in the logs:

```
2022-07-06 13:14:35.363081+02:00 [info] Starting ejabberd module Qrcode
```

5. Now the ejabberd internal web server provides QR codes of any given JID. Try visiting an URL like `http://localhost:5280/qrcode/anyusername/somedomain/`

Elixir module in ejabberd-contrib

Using ejabberd-contrib it's possible to install additional ejabberd modules without compiling ejabberd, or requiring ejabberd source code. This is useful if you install ejabberd using binary installers or a container image.

And it's possible to write a custom module and Add your module to an existing ejabberd installation...

Let's write a new ejabberd module in elixir, compile and install in an existing ejabberd deployment without requiring its source code. This example module adds a simple section listing PIDs in the users page in ejabberd WebAdmin.

1. First, create this path

```
$HOME/.ejabberd-modules/sources/mod_webadmin_pid/lib/
```

2. and copy the mod_webadmin_pid.ex source code to:

```
$HOME/.ejabberd-modules/sources/mod_webadmin_pid/lib/mod_webadmin_pid.ex
```

3. Create a specification file in YAML format as `mod_webadmin_pid.spec` (see examples from ejabberd-contrib). So, create the file

```
$HOME/.ejabberd-modules/sources/mod_webadmin_pid/mod_webadmin_pid.spec
```

with this content:

```
summary: "Display PIDs in User page in Web Admin"
```

4. From that point you should see it as available module:

```
ejabberdctl modules_available
mod_webadmin_pid Display PIDs in User page in Web Admin
```

5. Now you can compile and install that module:

```
ejabberdctl module_install mod_webadmin_pid
```

6. Enable the module in `ejabberd.yml`:

```
modules:
  'Elixir.ModWebAdminPid': {}
```

7. When restarting ejabberd, it will show in the logs:

```
2022-07-06 13:14:35.363081+02:00 [info] Starting ejabberd module WebAdminPid
```

8. Finally, go to ejabberd WebAdmin -> Virtual Hosts -> your vhost -> Users -> some online user -> and there will be a new section "PIDs".

Record definition

To use an erlang record defined in ejabberd's header file, use Elixir's Record to extract the fields and define an Elixir record with its usage macros.

For example, add this to the beginning of mod_presence_demo.ex:

```
require Record

Record.defrecord(:presence,
  Record.extract(:presence, from_lib: "xmpp/include/xmpp.hrl"))
```

Later you can use those macros, named like your record, see the examples.

In our example, let's improve the `on_presence` function and use the `presence` macros to get the `to` field:

```
def on_presence(_user, _server, _resource, packet) do
  to_jid = presence(packet, :to)
  to_str = :jid.to_string(to_jid)
  info('Received presence for #{to_str}:~n~p', [packet])
  :none
end
```

```
mod_qrcode.ex
```

Example ejabberd module written in elixir:

mod_qrcode.ex

```elixir
defmodule ModQrcode do
  use Ejabberd.Module

  def start(host, _opts) do
    info('Starting ejabberd module Qrcode')
    :ok
  end

  def stop(host) do
    info('Stopping ejabberd module Qrcode')
    :ok
  end

  def process([username, hostname] = _path, _query) do
    uri = <<"xmpp:", username::binary, "@", hostname::binary>>
    qr = QRCodeEx.svg(QRCodeEx.encode(uri), [{:color, "#3fb0d2"}])
    qxmlel = :fxml_stream.parse_element(qr)
    {200,
     [{<<"Server">>, <<"ejabberd">>},
      {<<"Content-Type">>, <<"image/svg+xml">>}],
      :ejabberd_web.make_xhtml([], [qxmlel])}
  end

  def process(path, _query) do
    info('Received HTTP query with path: ~p', [path])
    {404, [], "Not Found"}
  end

  def depends(_host, _opts) do
    []
  end

  def mod_options(_host) do
    []
  end

  def mod_doc() do
    %{:desc => 'This is just a demonstration.'}
  end

end
```

```
mod_webadmin_pid.ex
```

Example ejabberd module written in elixir:

mod_webadmin_pid.ex

```elixir
defmodule ModWebAdminPid do
  use Ejabberd.Module

  require Record

  Record.defrecord(:xmlel,
    Record.extract(:xmlel, from_lib: "xmpp/include/xmpp.hrl"))

  Record.defrecord(:request,
    Record.extract(:request, from: "include/ejabberd_http.hrl"))

  ##====================================================================
  ## gen_mod callbacks
  ##====================================================================

  def start(host, _opts) do
    info('Starting ejabberd module WebAdminPid')
    :ejabberd_hooks.add(:webadmin_user, host, __MODULE__, :webadmin_user, 60)
    :ejabberd_hooks.add(:webadmin_page_host, host, __MODULE__, :webadmin_page, 60)
    :ok
  end

  def stop(host) do
    info('Stopping ejabberd module WebAdminPid')
    :ejabberd_hooks.delete(:webadmin_user, host, __MODULE__, :webadmin_user, 60)
    :ejabberd_hooks.delete(:webadmin_page_host, host, __MODULE__, :webadmin_page, 60)
    :ok
  end

  def depends(_host, _opts) do
    []
  end
```

```elixir
  def mod_options(_host) do
    []
  end

  def mod_doc() do
    %{:desc => 'This is just a demonstration.'}
  end

  ##====================================================================
  ## Web Admin
  ##====================================================================

  def webadmin_user(acc, user, server, _lang) do
    resources = :ejabberd_sm.get_user_resources(user, server)

    pids_elements = Enum.map(resources,
      fn resource ->
        pid = :ejabberd_sm.get_session_pid(user, server, resource)
        pid_string = :erlang.pid_to_list(pid)
        xmlel(name: "a", attrs: [{"href", "pid/#{pid_string}"}], children: [xmlcdata: pid_string])
      end)

    pids_separated = Enum.intersperse(pids_elements, {:xmlcdata, ", "})

    new_element = xmlel(name: "h3", children: [xmlcdata: "PIDs:"])

    acc ++ [new_element] ++ pids_separated
  end

  def webadmin_page(_acc, host, request(path: ["user", user, "pid", pid])) do
    res = webadmin_pid(user, host, pid)
    {:stop, res}
  end

  def webadmin_page(acc, _host, _request) do
    acc
  end

  def webadmin_pid(user, host, pid_string) do
    us = :jid.to_string(:jid.make(user, host))
    page_title = 'Pid #{pid_string} of #{us}'

    pid = :erlang.list_to_pid(String.to_charlist(pid_string))
    pid_info = Process.info(pid)
    pid_info_string = :io_lib.format("~p", [pid_info])

    [xmlel(name: "h1", children: [xmlcdata: page_title]),
     xmlel(name: "pre", children: [xmlcdata: pid_info_string])]
  end

end
```

The ejabberd Developer Livebook

> **Info**
>
> This page is designed to run interactively using Livebook. Of course, you could simply reproduce the instructions manually yourself. But, if possible, install Livebook in your machine and get the full experience clicking on the button:
>
> **📗 Run in Livebook**

```elixir
filename = "ejabberd.yml"

if File.exists?(filename) do
  Mix.install([
    {:ejabberd, "~> 24.2"},
    {:kino, "~> 0.12.3"}
  ])
else
  url = "https://raw.githubusercontent.com/processone/ejabberd/master/ejabberd.yml.example"

  Mix.install([:req]) &&
    File.write!(
      filename,
      String.replace(Req.get!(url).body, "starttls_required: true", "")
    )

  IO.puts("ejabberd.yml downloaded correctly, click 'Reconnect and setup' to download ejabberd.")
end
```

Setup ejabberd inside livebook

This Livebook will download, compile and install ejabberd:

1. If you want to use a specific `ejabberd.yml` configuration file, copy it to your Livebook folder.
2. On top of this page, click `Setup`.
3. If `ejabberd.yml` is not found, it will be downloaded from ejabberd git repository.
4. Click `Reconnect and setup` to download ejabberd and all its dependencies. It will be compiled and started... it may take a pair of minutes.

Alternatively, if you already have ejabberd installed and running in your system, you can connect livebook to your ejabberd node

Execute some Erlang code

Now that Livebook is connected a running ejabberd node, you can run Erlang and Elixir code from this page directly in your node.

For example, to run some erlang code, put your mouse over the new lines and click on `Evaluate`:

```erlang
ejabberd_admin:registered_vhosts().
```

Let's define the details of an account, we will later register it. You may change those values:

```erlang
Username = <<"user1">>,
Server = <<"localhost">>,
Password = <<"somepass123">>,
{Username, Server, Password}.
```

Now let's execute an Erlang function to register the account:

```erlang
ejabberd_auth:try_register(Username, Server, Password).
```

Let's check the account was registered:

```
ejabberd_auth:get_users(<<"localhost">>).
```

And is the account's password the one we introduced?

```
Password == ejabberd_auth:get_password(Username, Server).
```

Ok, enough for now, let's remove the account:

```
ejabberd_auth:remove_user(Username, Server).
```

Execute some Elixir code

The same code of the previous section can be executed using Elixir:

```
:ejabberd_admin.registered_vhosts()
```

```
username = <<"user1">>
server = <<"localhost">>
password = <<"somepass123">>
{username, server, password}
```

```
:ejabberd_auth.try_register(username, server, password)
```

```
:ejabberd_auth.get_users(server)
```

```
password == :ejabberd_auth.get_password(username, server)
```

```
:ejabberd_auth.remove_user(username, server)
```

Run API commands

Let's run some ejabberd API commands using the ejabberd_ctl frontend (there is is also mod_http_api and ejabberd_xmlrpc).

For example, the status API command:

```
ejabberd_ctl:process(["status"]).
```

How to register an account using ejabberd_ctl to call the API command

```
command = ~c"register"
:ejabberd_ctl.process([command, username, server, password])
```

If you have ejabberd installed in the the system, and the `ejabberdctl` command-line script is available in your PATH, then you could also try to execute with:

```
os:cmd("ejabberdctl status").
```

```
:os.cmd(~c"ejabberdctl status")
```

Draw process structure

Let's view the ejabberd process tree:

```
Kino.Process.render_app_tree(:ejabberd, direction: :left_right)
```

Let's view the erlang processes that handle XMPP client connections. If this graph is empty, login to ejabberd with a client and reevaluate this code:

```
Kino.Process.render_sup_tree(:ejabberd_c2s_sup, direction: :left_right)
```

And some information about the erlang process that handles the XMPP client session:

```
[resource] = :ejabberd_sm.get_user_resources(username, server)
Elixir.Process.info(:ejabberd_sm.get_session_pid(username, server, resource))
```

Connect Livebook to your ejabberd node

By default this livebook downloads, compiles and starts ejabberd by setting up ejabberd sinde livebook. If you already have ejabberd installed and would like to connect this livebook to your existing ejabberd node, follow those steps:

Get erlang node name

To connect Livebook to your running ejabberd node, you must know its erlang node name and its cookie.

The erlang node name is by default `ejabberd@localhost` . You can check this in many places, for example:

- Using `ejabberdctl status` :

```
$ ejabberdctl status
The node ejabberd@localhost is started with status: started
ejabberd 24.2.52 is running in that node
```

- In the `ejabberd.log` file, which contains a line like:

```
2024-03-26 13:18:35.019288+01:00 [info] <0.216.0>@ejabberd_app:start/2:63
ejabberd 24.2.52 is started in the node ejabberd@localhost in 0.91s
```

Setup ejabberd node

A Livebook can only connect to an Erlang node that has Elixir support. So, make sure you install not only Erlang, but also Elixir.

When compiling ejabberd, make sure to enable Elixir support. It should get enabled by default, but you can ensure this: either by `./configure --with-rebar=mix` or by `./configure --enable-elixir` .

Then start ejabberd with IEx shell: `ejabberdctl iexlive`

Get erlang cookie

The erlang cookie is a random string of capital letters required to connect to an existing erlang node.

You can get it in a running node, simply call:

```
:erlang.get_cookie()
:XQFOPGGPSNEZNUWKRZJU
```

Connect this livebook to your ejabberd node

Now that you have ejabberd running, and you know the information required to connect to it:

1. go to Livebook

2. in the left side bar, click the `Runtime settings` icon, or press `sr` keyboard shortcut

3. click the `Configure` button

4. click the `Attached node` button

5. introduce the erlang node name (`ejabberd@localhost`) and cookie (`XQFOPGGPSNEZNUWKRZJU`) of your ejabberd node

6. click the `Connect` button (it may say `Reconnect`)

7. If it connected successfully, it will now show memory consumption of that node

Test the connection

Now that Livebook is connected to your running ejabberd node, you can run Erlang and Elixir code from this page directly in your node.

For example, to run some erlang code, put your mouse over the new lines and click on `Evaluate` :

```
ejabberd_admin:registered_vhosts().
```

The same code can be executed using Elixir:

```
:ejabberd_admin.registered_vhosts()
```

Stop ejabberd

Let' stop ejabberd insie livebook

```
:ejabberd.stop()
```

Internationalization and Localization

The source code of `ejabberd` supports localization: all built-in modules support the `xml:lang` attribute inside IQ queries, and the Web Admin supports the `Accept-Language` HTTP header.

There are two ways to improve the translation of a language:

- Edit the corresponding .po file in ejabberd-po git repository with a gettext-compatible program (Poedit, KBabel, Lokalize, ...). Then submit a Pull Request.
- Using the ejabberd-po Weblate online service.

Once the translators have improved the `po` files, you can run `make translations`. With that command, the translatable strings are extracted from source code to generate the file `ejabberd.pot`. This file is merged with each .po file to produce updated .po files. Finally those .po files are exported to .msg files, that have a format easily readable by `ejabberd`.

ejabberd Modules Development

Introduction

ejabberd is based on a modular architecture that makes it highly customizable and infinitely extensible.

Here is an overview of ejabberd internal architecture:
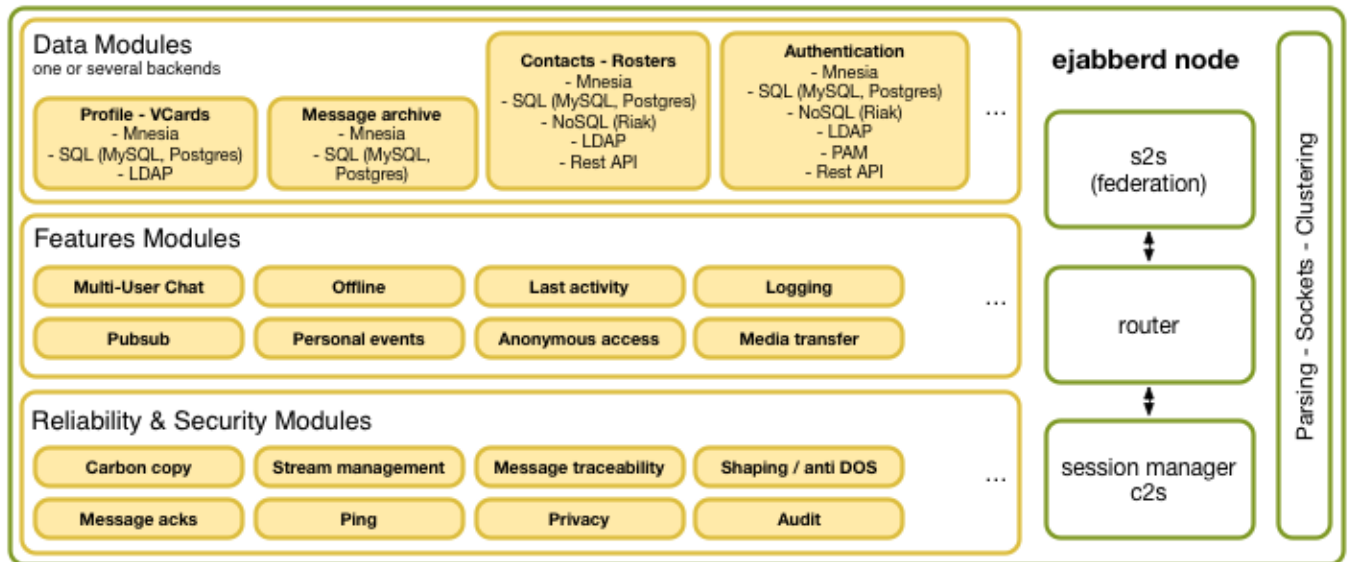
What is a module ?

Outside of a few infrastructure core components most of ejabberd features are developed as modules. Modules are used to extend the features of ejabberd (plugins).

How to write a custom module ?

Ejabberd comes with a lot of modules, but sometimes you may need an unsupported feature from the official sources or maybe you need to write your own custom implementation for your very special needs.

Each modules is written in either Erlang or Elixir. To use them, you typically declare them in ejabberd configuration file. That's also the place where you can configure the module, by passing supported options to overload its default behaviour.

On ejabberd launch, the server will start all the declared modules. You can start (or stop) them manually from Erlang shell as well.

As a convention, module names starts with "mod_", but you can actually call them as you want.

The `gen_mod` behaviour

All ejabberd modules are implementing the `gen_mod` behaviour. It means that a module must provide the following API:

```
start(Host, Opts) -> ok
stop(Host) -> ok
depends(Host, Opts) -> []
mod_options(Host) -> []
```

Parameters are:

- Host = `string()`
- Opts = `[{Name, Value}]`
- Name = Value = `string()`

Host is the name of the virtual host running the module. The `start/2` and `stop/1` functions are called for each virtual host at start and stop time of the server.

`Opts` is a lists of options as defined in the configuration file for the module. They can be retrieved with the `gen_mod:get_opt/3` function.

mod_hello_world

The following code shows the simplest possible module.

mod_hello_world.erl

```erlang
-module(mod_hello_world).

-behaviour(gen_mod).

%% Required by ?INFO_MSG macros
-include("logger.hrl").

%% Required by ?T macro
-include("translate.hrl").

%% gen_mod API callbacks
-export([start/2, stop/1, depends/2, mod_options/1, mod_doc/0]).

start(_Host, _Opts) ->
    ?INFO_MSG("Hello, ejabberd world!", []),
    ok.

stop(_Host) ->
    ?INFO_MSG("Bye bye, ejabberd world!", []),
    ok.

depends(_Host, _Opts) ->
    [].

mod_options(_Host) ->
    [].

mod_doc() ->
    #{desc =>
          ?T("This is an example module.")}.
```

Now you have two ways to compile and install the module: If you compiled ejabberd from source code, you can copy that source code file with all the other ejabberd source code files, so it will be compiled and installed with them. If you installed some compiled ejabberd package, you can create your own module dir, see Add Your Module.

You can enable your new module by adding it in the ejabberd config file. Adding the following snippet in the config file will integrate the module in ejabberd module lifecycle management. It means the module will be started at ejabberd launch and stopped during ejabberd shutdown process:

```yaml
modules:
  ...
  mod_hello_world: {}
```

Or you can start / stop it manually by typing the following commands in an Erlang shell running ejabberd:

- To manually start your module:

    ```erlang
    gen_mod:start_module(<<"localhost">>, mod_hello_world, []).
    ```

- To manually stop your module:

    ```erlang
    gen_mod:stop_module(<<"localhost">>, mod_hello_world).
    ```

When the module is started, either on ejabberd start or manually, you should see the following message in ejabberd log file:

```
19:13:29.717 [info] Hello, ejabberd world!
```

ejabberd-contrib

ejabberd is able to fetch module sources by itself, compile with correct flags and install in a local repository, without any external dependencies. You do not need to know Erlang and have it installed in order to use the contributed modules. The contributed modules repository is ejabberd-contrib on github. It contains most used contributions and also can link to any other external repository. This works with ejabberd modules written in Erlang and will also support new Elixir modules.

Basic usage

As a user, this is how it works:

1. First you need to get/update the list of available modules. This must be done before anything else to let module related features to work correctly. You may want to repeat this command before you need installing a new module or an attended server upgrade.

```
ejabberdctl modules_update_specs
```

2. Then you can list available modules:

```
ejabberdctl modules_available
...
mod_admin_extra Additional ejabberd commands
mod_archive Supports almost all the XEP-0136 version 0.6 except otr
mod_cron Execute scheduled commands
mod_log_chat Logging chat messages in text files
...
```

3. Let's install `mod_cron` from ejabberd-contrib repository:

```
ejabberdctl module_install mod_cron
ok
```

4. You can check anytime what modules are installed:

```
ejabberdctl modules_installed
mod_cron
```

5. An example default configuration is installed, and will be read by ejabberd when it starts. You can find that small configuration in:

```
$HOME/.ejabberd-modules/mod_cron/conf/mod_cron.yml
```

6. And finally, you can remove the module:

```
ejabberdctl module_uninstall mod_cron
ok
```

Add your module

If you install ejabberd using the official ProcessOne installer, it includes everything needed to build ejabberd modules on its own.

1. First, create this path

```
$HOME/.ejabberd-modules/sources/mod_hello_world/src/
```

2. and copy your source code to this location:

```
$HOME/.ejabberd-modules/sources/mod_hello_world/src/mod_hello_world.erl
```

3. Create a specification file in YAML format as mod_hello_world.spec (see examples from ejabberd-contrib). So, create the file

```
$HOME/.ejabberd-modules/sources/mod_hello_world/mod_hello_world.spec
```

with this content:

mod_hello_world.spec

```
summary: "Hello World example module"
```

4. From that point you should see it as available module:

```
ejabberdctl modules_available
mod_hello_world Hello World example module
```

5. Now you can install and uninstall that module like any other, as described in the previous section.

6. If you plan to publish your module, you should check if your module follows the policy and if it compiles correctly:

```
ejabberdctl module_check mod_mysupermodule
ok
```

If all is OK, your're done ! Else, just follow the warning/error messages to fix the issues.

You may consider publishing your module as a tgz/zip archive or git repository, and send your spec file for integration in ejabberd-contrib repository. ejabberd-contrib will only host a copy of your spec file and does not need your code to make it available to all ejabberd users.

Your module with Docker

If you installed ejabberd using the Docker image, these specific instructions allow you to use your module with your Docker image:

1. Create a local directory for the contributed modules:

```
mkdir docker-modules
```

2. Then create the directory structure for your custom module:

```
cd docker-modules

mkdir -p sources/mod_hello_world/
touch sources/mod_hello_world/mod_hello_world.spec

mkdir sources/mod_hello_world/src/
mv mod_hello_world.erl sources/mod_hello_world/src/

mkdir sources/mod_hello_world/conf/
echo -e "modules:\n  mod_hello_world: {}" > sources/mod_hello_world/conf/mod_hello_world.yml

cd ..
```

3. Grant ownership of that directory to the UID that ejabberd will use inside the Docker image:

```
sudo chown 9000 -R docker-modules/
```

4. Start ejabberd in Docker:

```
sudo docker run \
  --name hellotest \
  -d \
  --volume "$(pwd)/docker-modules:/home/ejabberd/.ejabberd-modules/" \
  -p 5222:5222 \
  -p 5280:5280 \
  ejabberd/ecs
```

5. Check the module is available for installing, and then install it:

```
sudo docker exec -it hellotest bin/ejabberdctl modules_available
mod_hello_world []

sudo docker exec -it hellotest bin/ejabberdctl module_install mod_hello_world
```

6. If the module works correctly, you will see the Hello string in the ejabberd logs when it starts:

```
sudo docker exec -it hellotest grep Hello logs/ejabberd.log
2020-10-06 13:40:13.154335+00:00 [info]
  <0.492.0>@mod_hello_world:start/2:15 Hello, ejabberd world!
```

Status

Please note this is provided as a beta version. We want the work in progress to be released early to gather feedback from developers and users.

For now, you need to edit the configuration snippet provided in module's conf directory and copy it into your ejabberd's main configuration. Then you'll need to restart ejabberd or manually start the module.

However, our plan is to keep iterating on the tool and to make our best to make module installation as easy as possible and avoid need to change main configuration: ejabberd should be able to include module configuration snippets on the fly in a near future.

Next steps

From there, you know how to package a module to integrate it inside ejabberd environment. Packaging a module allows you to:

- Integrate in ejabberd overall application life cycle, i.e. with the start and stop mechanism.
- Get data from ejabberd configuration file.

Now, to do useful stuff, you need to integrate with ejabberd data flow. You have two mechanisms available from ejabberd modules:

- Events and Hooks: This is to handle internal ejabberd triggers and subscribe to them to perform actions or provide data.
- IQ Handlers: This is a way to register ejabberd module to handle XMPP Info Queries. This is the XMPP way to provide new services.

MucSub: Multi-User Chat Subscriptions

Motivation

In XMPP, Multi-User Chat rooms design rely on presence. To participate in a MUC room, you need to send a presence to the room. When you get disconnected, you leave the room and stopped being part of the room. User involvement in MUC rooms is not permanent.

This is an issue with the rise of mobile applications. Chatting with friends in a room is a big part of messaging usage on mobile. However, to save battery life, mobile devices will freeze mobile XMPP application after a while when they get to background. It means that the connection is lost and that the session is usually terminated.

Some workaround have been used to try letting user keep on receiving messages from MUC room when the app is in background. The most common one is to keep the session open for a while until a timeout happens. This is the approach promoted on mobile by XEP-0198 - Stream Management. When messages are received and no TCP/IP connection is attached, server usually fallback sending the message to the user using mobile push notification service to warn the user that a message has been received.

This approach has many drawbacks:

1. It is not permanent. The idea of having the session kept into memory for a while is interesting but it is just a long timeout. After that timeout, the session is closed and the user will leave the room. No message will be received anymore.
2. It does not play well with normal server / cluster operations. If you restart the service where the user session is kept, it will disappear. You can dump them to disk and recreate them on start, but it means that if the node crashes, your session will be lost and user will stop receiving messages.
3. It does not change the fundamental nature of MUC chat room. They are still presence-based. It means that if you need to restart the MUC service, or if it crashes, presence are lost. For connected clients, they are expected to join the MUC room again. However, for mobile clients, it cannot happens until user reopens the app. Moreover, it means that on new session start, user client is expected to join all the MUC rooms they want to keep track of on connect.

This specification tries to solve those issues by keeping most of the logic of the MUC room intact. There is attempt to rewrite XMPP Multi-User chat rooms by splitting presence from ability to receive and send messages (XEP-0369: Mediated Information eXchange (MIX)). However, the features covered by the MUC protocol are quite comprehensive and the MIX protocol is not yet ready to cover all the MUC use cases yet. The goal is to produce an intermediate state that is compliant with MUC and leverage most of the MUC features, while adding the most basic feature possible to implement the MUC/Sub extension.

This specifications tries to merge ideas to produce a MUC extension that make MUC rooms mobile clients friendly.

To play well with mobile, MUC room need to support the following features:

- Add the ability to send and receive messages to a room without having to send presence to the room. More generally allow other type of interaction with the room (like configuration changes for example or kick and ban). We will leverage existing publish and subscribe approach.
- Add the ability to resync the client for missed messages on reconnect. We will leverage Message Archive Management service for MUC.
- Finally, ensure that a server can implement push notification service to ensure alerting of offline users when MUC messages are received.

The goal is to learn from real life working implementation to help feeding MIX with feedback from the field, without having to reinvent a complete new protocol.

General principle

The core idea is to expose MUC rooms as PubSub nodes and to introduce the concept of MUC rooms subscribers.

A user affiliated to a MUC room should be able to subscribe to MUC node events and have them routed to their JID, even if they are not a participant in the room. It means that a user can receive messages without having to send presence to the room. In that sense, "joining the room" in XEP-0045 becomes more "Being available in the MUC room".

Discovering support

Discovering support on MUC service

You can check if MUC/Sub feature is available on MUC service by sending Disco Info IQ:

```
<iq from='hag66@shakespeare.example/pda'
    to='muc.shakespeare.example'
    type='get'
    id='ik3vs715'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

MUC service will show a feature of type 'urn:xmpp:mucsub:0' to the response if the feature is supported and enabled:

```
<iq from="muc.shakespeare.example"
    to="hag66@shakespeare.example/pda"
    type="result"
    id="ik3vs715">
  <query xmlns="http://jabber.org/protocol/disco#info">
    <identity category="conference"
              type="text"
              name="Chatrooms" />
    ...
    <feature var="urn:xmpp:mucsub:0" />
    ...
  </query>
</iq>
```

Discovering support on a specific MUC

A user can discover support for MUC/Sub feature on a MUC room as follow:

```
<iq from='hag66@shakespeare.example/pda'
    to='coven@muc.shakespeare.example'
    type='get'
    id='ik3vs715'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

A conference MUST add 'urn:xmpp:mucsub:0' to the response if the feature is supported and enabled:

```
<iq from='coven@muc.shakespeare.example'
    to='hag66@shakespeare.example/pda'
    type='result'
    id='ik3vs715'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <identity category='conference'
              name='A Dark Cave'
              type='text' />
    <feature var='http://jabber.org/protocol/muc' />
    ...
    <feature var='urn:xmpp:mucsub:0' />
    ...
  </query>
</iq>
```

Option MUC room support for subscriptions

Even if MUC room supports MUC/Sub feature, it MAY be explicitly enabled or disabled thanks to a new configuration option:

• Allow subscription: Users can subscribe to MUC/Sub events.

Subscriber role

Until a subscriber is not joined a conference (see Joining a MUC Room), a subscriber role MUST be 'none'. When a subscriber is joined a conference its role is changed according to XEP-0045 rules, that is, it becomes either 'visitor', 'participant' or 'moderator'.

Subscribing to MUC/Sub events

User can subscribe to the following events, by subscribing to specific nodes:

- urn:xmpp:mucsub:nodes:presence
- urn:xmpp:mucsub:nodes:messages
- urn:xmpp:mucsub:nodes:affiliations
- urn:xmpp:mucsub:nodes:subscribers
- urn:xmpp:mucsub:nodes:config
- urn:xmpp:mucsub:nodes:subject
- urn:xmpp:mucsub:nodes:system

Example: User Subscribes to MUC/Sub events

```
<iq from='hag66@shakespeare.example'
    to='coven@muc.shakespeare.example'
    type='set'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <subscribe xmlns='urn:xmpp:mucsub:0'
           nick='mynick'
           password='roompassword'>
    <event node='urn:xmpp:mucsub:nodes:messages' />
    <event node='urn:xmpp:mucsub:nodes:affiliations' />
    <event node='urn:xmpp:mucsub:nodes:subject' />
    <event node='urn:xmpp:mucsub:nodes:config' />
  </subscribe>
</iq>
```

If user is allowed to subscribe, server replies with success. The password attribute can be provided when subscribing to a password-protected room.

Example: Server replies with success

```
<iq from='coven@muc.shakespeare.example'
    to='hag66@shakespeare.example'
    type='result'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <subscribe xmlns='urn:xmpp:mucsub:0'>
    <event node='urn:xmpp:mucsub:nodes:messages' />
    <event node='urn:xmpp:mucsub:nodes:affiliations' />
    <event node='urn:xmpp:mucsub:nodes:subject' />
    <event node='urn:xmpp:mucsub:nodes:config' />
  </subscribe>
</iq>
```

Subscription is associated with a nick. It will implicitly register the nick. Server should otherwise make sure that subscription match the user registered nickname in that room. In order to change the nick and/or subscription nodes, the same request MUST be sent with a different nick or nodes information.

Example: User changes subscription data

```
<iq from='hag66@shakespeare.example'
    to='coven@muc.shakespeare.example'
    type='set'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <subscribe xmlns='urn:xmpp:mucsub:0'
           nick='newnick'>
    <event node='urn:xmpp:mucsub:nodes:messages' />
    <event node='urn:xmpp:mucsub:nodes:presence' />
  </subscribe>
</iq>
```

A room moderator can subscribe another user to MUC Room events by providing the user JID as an attribute in the `<subscribe/>` element.

Example: Room moderator subscribes another user

```
<iq from='king@shakespeare.example'
    to='coven@muc.shakespeare.example'
    type='set'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <subscribe xmlns='urn:xmpp:mucsub:0'
             jid='hag66@shakespeare.example'
             nick='mynick'
             password='roompassword'>
    <event node='urn:xmpp:mucsub:nodes:messages' />
    <event node='urn:xmpp:mucsub:nodes:affiliations' />
    <event node='urn:xmpp:mucsub:nodes:subject' />
    <event node='urn:xmpp:mucsub:nodes:config' />
  </subscribe>
</iq>
```

Unsubscribing from a MUC Room

At any time a user can unsubscribe from MUC Room events.

Example: User unsubscribes from a MUC Room

```
<iq from='hag66@shakespeare.example'
    to='coven@muc.shakespeare.example'
    type='set'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <unsubscribe xmlns='urn:xmpp:mucsub:0' />
</iq>
```

Example: A MUC Room responds to unsubscribe request

```
<iq from='coven@muc.shakespeare.example'
    to='hag66@shakespeare.example'
    type='result'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7' />
```

A room moderator can unsubscribe another room user from MUC Room events by providing the user JID as an attribute in the `<unsubscribe/>` element.

Example: Room moderator unsubscribes another room user

```
<iq from='king@shakespeare.example'
    to='coven@muc.shakespeare.example'
    type='set'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <unsubscribe xmlns='urn:xmpp:mucsub:0'
               jid='hag66@shakespeare.example'/>
</iq>
```

Subscriber actions

If not stated otherwise in this document, a subscriber MUST perform any actions in the conference as described in XEP-0045. For example, it MUST send messages to all occupants according to 7.4 Sending a Message to All Occupants, it MUST configure a conference according to 10.2 Subsequent Room Configuration and so on.

Here are a few examples:

Sending a message

Sending a message is like sending a standard groupchat message in MUC room:

```
<message from="hag66@shakespeare.example"
         to="coven@muc.shakespeare.example"
         type="groupchat">
  <body>Test</body>
</message>
```

No need to join it after you connect. As a subscriber, you can send messages at any time.

Joining a MUC Room

If a user wants to be present in the room, they just have to join the room as defined in XEP-0045.

A subscriber MAY decide to join a conference (in the XEP-0045 sense). In this case a conference MUST behave as described in XEP-0045 7.2 Entering a Room. A conference MUST process events as described under XEP-0045 7.1 Order of Events except it MUST not send room history. When a subscriber is joined, a conference MUST stop sending subscription events and MUST switch to a regular groupchat protocol (as described in XEP-0045) until a subscriber leaves.

Receiving events

Here is as an example message received by a subscriber when a message is posted to a MUC room when subscriber is subscribed to node urn:xmpp:mucsub:nodes:messages:

```
<message from="coven@muc.shakespeare.example"
         to="hag66@shakespeare.example/pda">
  <event xmlns="http://jabber.org/protocol/pubsub#event">
    <items node="urn:xmpp:mucsub:nodes:messages">
      <item id="18277869892147515942">
        <message from="coven@muc.shakespeare.example/secondwitch"
                 to="hag66@shakespeare.example/pda"
                 type="groupchat"
                 xmlns="jabber:client">
          <archived xmlns="urn:xmpp:mam:tmp"
                    by="muc.shakespeare.example"
                    id="1467896732929849" />
          <stanza-id xmlns="urn:xmpp:sid:0"
                     by="muc.shakespeare.example"
                     id="1467896732929849" />
          <body>Hello from the MUC room !</body>
        </message>
      </item>
    </items>
  </event>
</message>
```

Presence changes in the MUC room are received wrapped in the same way by subscribers which subscribed to node urn:xmpp:mucsub:nodes:presence:

```
<message from="coven@muc.shakespeare.example"
         to="hag66@shakespeare.example/pda">
  <event xmlns="http://jabber.org/protocol/pubsub#event">
    <items node="urn:xmpp:mucsub:nodes:presences">
      <item id="8170705750417052518">
        <presence xmlns="jabber:client"
                  from="coven@muc.shakespeare.example/secondwitch"
                  type="unavailable"
                  to="hag66@shakespeare.example/pda">
          <x xmlns="http://jabber.org/protocol/muc#user">
            <item affiliation="none"
                  role="none" />
          </x>
        </presence>
      </item>
    </items>
  </event>
</message>
```

If subscriber is subscribed to node urn:xmpp:mucsub:nodes:subscribers, message will ne sent for every mucsub subscription change. When a user becomes a subscriber:

```
<message from="coven@muc.shakespeare.example"
         to="hag66@shakespeare.example/pda">
  <event xmlns="http://jabber.org/protocol/pubsub#event">
    <items node="urn:xmpp:mucsub:nodes:subscribers">
      <item id="17895981155977588737">
        <subscribe xmlns="urn:xmpp:mucsub:0"
                   jid="bob@server.com"
                   nick="bob"/>
      </item>
    </items>
  </event>
</message>
```

When a user lost its subscription:

```
<message from="coven@muc.shakespeare.example"
         to="hag66@shakespeare.example/pda">
    <event xmlns="http://jabber.org/protocol/pubsub#event">
      <items node="urn:xmpp:mucsub:nodes:subscribers">
        <item id="10776102417321261057">
          <unsubscribe xmlns="urn:xmpp:mucsub:0"
                       jid="bob@server.com"
                       nick="bob"/>
        </item>
      </items>
    </event>
</message>
```

Note: Sometimes jid in subscribe/unsubscribe event may be missing if room is set to anonymous and user is not moderator.

Getting List of subscribed rooms

A user can query the MUC service to get their list of subscriptions.

Example: User asks for subscriptions list

```
<iq from='hag66@shakespeare.example'
    to='muc.shakespeare.example'
    type='get'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <subscriptions xmlns='urn:xmpp:mucsub:0' />
</iq>
```

Example: Server replies with subscriptions list

```
<iq from='muc.shakespeare.example'
    to='hag66@shakespeare.example'
    type='result'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <subscriptions xmlns='urn:xmpp:mucsub:0'>
    <subscription nick='mynick'
                  jid='coven@muc.shakespeare.example'>
      <event node='urn:xmpp:mucsub:nodes:messages'/>
      <event node='urn:xmpp:mucsub:nodes:affiliations'/>
      <event node='urn:xmpp:mucsub:nodes:subject'/>
      <event node='urn:xmpp:mucsub:nodes:config'/>
    </subscription>
    <subscription nick='MyNick'
                  jid='chat@muc.shakespeare.example'>
      <event node='urn:xmpp:mucsub:nodes:messages'/>
    </subscription>
  </subscriptions>
</iq>
```

Getting list of subscribers of a room

A subscriber or room moderator can get the list of subscribers by sending `<subscriptions/>` request directly to the room JID.

Example: Asks for subscribers list

```
<iq from='hag66@shakespeare.example'
    to='coven@muc.shakespeare.example'
    type='get'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <subscriptions xmlns='urn:xmpp:mucsub:0' />
</iq>
```

Example: Server replies with subscribers list

```
<iq from='coven@muc.shakespeare.example'
    to='hag66@shakespeare.example'
    type='result'
    id='E6E10350-76CF-40C6-B91B-1EA08C332FC7'>
  <subscriptions xmlns='urn:xmpp:mucsub:0'>
    <subscription nick='Juliet'
                  jid='juliet@shakespeare.example'>
      <event node='urn:xmpp:mucsub:nodes:messages'/>
      <event node='urn:xmpp:mucsub:nodes:affiliations'/>
    </subscription>
    <subscription nick='Romeo'
                  jid='romeo@shakespeare.example'>
      <event node='urn:xmpp:mucsub:nodes:messages'/>
```

```
        </subscription>
    </subscriptions>
</iq>
```

Compliance with existing MUC clients

MUC/Sub approach is compliant with existing MUC service and MUC clients. MUC clients compliant with XEP-0045 will receive messages posted by subscribers. They may not see the user's presence, but it should not be an issue for most clients. Most clients already support receiving messages from users that are not currently in the MUC room through history retrieval.

This approach should also help most clients to support better integration with third-party services posting to MUC room through API (as)

However, a server could choose to send presence on behalf of subscribers when a user joins the room (in the XEP-0045 sense) so that the subscriber will be shown in MUC roster of legacy clients.

Synchronization of MUC messages: Leveraging MAM support

To be friendly with mobile, the MAM service should allow a user to connect and easily resync their history for all MUC subscriptions. For details about MAM, see XEP-0313 Message Archive Management and your software's documentation, for instance ejabberd's mod_mam.

Thanks to ability to get the list of all the existing subscription, client can get a starting point to interact with MAM service to resync history and get the messages that were missed while the user was offline.

If you subscribe to MucSub, MAM will add the message to your own user JID on new messages. You will simply be able to query them using your own JID from the standard MAM service.

It means, you can get all new MUC message in subscribed room thanks to MucSub, with a single query. For example, if you ask for all messages sent since a specific date, the result will contain both normal chat and MucSub messages.

You would only need to query MUC for MAM for rooms for which you do not use MucSub as with MucSub you will be "delivered" each message (in that case, each message is added your MAM archive).

Push support compliance

Subscriptions are compliant with push mechanism. It is supported out of the box when using ProcessOne p1:push implementation (deployed on ejabberd SaaS for example).

More generally, it is straightforward to handle them through ejabberd developer API to implement custom mechanisms.

Subscriptions are delivered to online users. If the user has no active session, the server can choose to broadcast to the user through a push notification.

When a session is opened, if the server detects that the user has not been recently active, or for any other reason, the server can still forward the message to a push notification service to warn the user that new messages are available in a MUC room.

ejabberd Test Suites

ejabberd comes with a comprehensive test suite to cover various part of the software platform.

XMPP end-to-end protocol test suite

Running ejabberd test suite

This test suite is a set of end-2-end integration tests that act like XMPP clients connecting with the server and is testing ejabberd at the protocol level. It also contains tests for the various backends that ejabberd supports.

The test suite is modular and can be run in parts (to focus on a group of features) or run for a specific backend.

The `CT_BACKENDS` environment variable specifies which backend tests to run. Current `CT_BACKENDS` values:

- `extauth`
- `ldap`
- `mnesia`
- `mssql`
- `mysql`
- `odbc`
- `pgsql`
- `redis`
- `sqlite`

Note: You must build ejabberd with proper backend support for the tests to work. If the tests fail and you aren't sure why, check the `configure` build options to make sure ejabberd is compiled with adequate backend support.

Note: these tests are e2e tests that operate a full ejabberd instance. So the ports that ejabberd needs must be available for testing. So you can't run an ejabberd instance at the same time you test.

Other options you can use to limit the tests that will be run is to pass a list of `groups` to test. Some `groups` examples:

- `no_db` : Runs subgroups `generic` and `test_proxy65` .
- `component`
- `extauth`
- `ldap`
- `mnesia`
- `mssql`
- `mysql`
- `pgsql`
- `redis`
- `s2s`
- `sqlite`

Usually, it is enough to just limit tests with `CT_BACKENDS` and let the test suite decide which relevant tests to run. Sometimes you may want to only focus on a specific backend, skipping the generic `no_db` tests.

Some example commands for running the XMPP end-to-end test suite using rebar and rebar3 ct:

```
CT_BACKENDS=mnesia rebar  ct suites=ejabberd
CT_BACKENDS=mnesia rebar  ct suites=ejabberd groups=mnesia
CT_BACKENDS=mnesia rebar  ct suites=ejabberd groups=generic
```

```
CT_BACKENDS=mnesia rebar3 ct --suite=test/ejabberd_SUITE --group=offline_flex,offline_send_all
CT_BACKENDS=redis  rebar3 ct --suite=test/ejabberd_SUITE --group=offline_flex,offline_send_all
```

If you have every backend configured, you can run all the tests with:

```
make test
```

Test suite conventions

The records used in test suite are autogenerated and come from `tools/xmpp_codec.hrl`. This is handy to match packets/results against expected values.

Dependency tests

ejabberd depends on a lot of dependent modules. The dependencies can be tested independently by checking them out and running their test suites directly.

Build test status

We run tests for ejabberd and dependencies automatically via Github Actions. We have a Dashboard available on Github to check the overall test status for all projects: ProcessOne Github Dashboard

Developing ejabberd with VSCode

🏷 added in 23.01

The ejabberd git repository includes basic configuration and a few scripts to get started with ejabberd development using Visual Studio Code.

There are several Visual Studio Code flavours suitable for ejabberd development:

- Visual Studio Code desktop app – local development with no dependencies
- VSCodium desktop app – local development installing dependencies
- Coder's code-server container image – local or remote development
- GitHub Codespaces service – quick and short remote development

Visual Studio Code

The official Visual Studio Code installers provided by Microsoft can use the official marketplace. That allows to install the Dev Container extension to compile and run ejabberd inside a prepared container, which includes Erlang/OTP and all the required libraries, so you don't need to install them in your machine.

However that installer is licensed under a not-FLOSS license and contains telemetry/tracking.

Once installed: install Git as suggested, clone the ejabberd git repository locally, let it install the Dev Container extension, then let it reopen the path inside the devcontainer.

VSCodium

VSCodium provides Free/Libre Open Source Software Binaries of VSCode. This is a great alternative to the official VSCode installer.

However, it can't use the official marketplace, uses instead the open-vsx.com marketplace, and the Dev Containers extension is not available. This means that you must install the ejabberd dependencies in your system to compile and debug ejabberd.

Once installed: open your local ejabberd git clone. It's highly recommended to go the EXTENSIONS tab and install the Erlang LS extension.

Coder's code-server

An easy, zero-cost, way to use VSCode in a web browser is through the ejabberd's code-server container image. This image is based in the Debian docker image and includes Coder's code-server, Erlang/OTP, Elixir, and all the required libraries.

Download and start the container, and provide as volume the path of your local ejabberd git clone:

```
docker run \
    --name coder \
    -it \
    -p 5208:5208 \
    -v $(pwd)/ejabberd:/workspaces/ejabberd \
    ghcr.io/processone/code-server
```

Now open in your web browser: `http://0.0.0.0:5208/`

The next time it can be started with `docker start -i coder`

GitHub Codespaces

The ejabberd git repository contains default configuration to use it in the GitHub Codespaces service.

This can be used remotely over a web browser, no need to install anything. Notice this is a service that can be used for free several hours each month, and later requires a subscription.

To start using it:

1. Go to https://github.com/codespaces
2. Click "New codespace"
3. Select ejabberd repository, desired branch, click "Create codespace"

Basic Usage

Once you have VSCode running and ejabberd git repository opened, open some erlang file, so Erlang LS extension gets started, and now you can go to RUN and run ejabberd for the first time. The first time it will take some time to compile, be patient.

Now you can:

- In RUN click `▷ Relive` to compile and start ejabberd
- In EXPLORER open any source code, and add a breakpoint
- In TERMINAL you can call: `ejabberdctl register admin localhost somepass`
- In PORTS you can view the addresses you can use to connect to the running ejabberd

The ejabberd configuration file is in `_build/relive/conf/ejabberd.yml`.

You can connect to ejabberd using a XMPP client using HTTPS BOSH or WS on port 5443. Webadmin is on port 5280, if it complains 404, add `admin/` to the URL.

Getting Started with XMPPFramework

Introduction

XMPP development on smartphone has always been challenging given the constraints on mobile platform.

This area will help you understand the challenges and help you get started with XMPP development on Apple iOS platform.

The main library to support XMPP on iOS is XMPPFramework.

XMPPFramework

XMPPFramework is a large framework relying on several dependencies. The easiest way to get started is to use Cocoapods to integrate XMPPFramework in your own project. It will take care of adding all dependencies and perform all the required configuration steps.

Here are the steps needed to get started:

1. Create a new iOS project in Xcode, if you do not have one.
2. If you do not yet have a `Podfile`, create it if `pod init` command from the project root directory.
3. Edit your `Podfile` to use XMPPFramework as a target. It may looks like:

```
platform :ios, '6.0'
use_frameworks!

target 'projectname' do
    pod 'XMPPFramework'
end
```

1. Run `pod install` command. It should download, install and configure three pods.
2. Open your XCode project with the newly created workspace file instead of the project file. This is required by Cocoapods so that you can use the installed Pods.
3. At this stage, you should be able to build your project successfully with the XMPP framework setup.

API

ejabberd Rest API

Introduction

ejabberd comes with a powerful API serving two goals:

1. Manage the XMPP service and integrate the platform with back-end platforms and script tools.

2. Allow users to perform tasks via the client, allowing simple basic clients that do not need to use XMPP protocol. This can be handy, for example to send a message from your smartwatch, or show the number of offline messages.

   The system is powerful and very versatile and you can configure it very finely, but it can be quite daunting to set up.

   This section is written to demystify ejabberd API configuration and help you integrate ejabberd with your other back-ends or script through an HTTP / HTTPS ReST API.

Understanding ejabberd "commands"

ejabberd operations are organised around the concept of commands. ejabberd standard modules already provide many commands, but the mechanism is generic and any module can provide its own set of commands. This exposition of commands for third-party modules make it very powerful.

All commands can be exposed through interfaces. Available interfaces are:

- ejabberdctl command-line tool,
- mod_http_api for ReST calls using JSON data,
- ejabberd_xmlrpc for XML-RPC calls,
- to some extend, XMPP protocol itself through discovery and adhoc commands, using mod_configure.

The ejabberd-contrib Github repository provides other interfaces that can be installed to execute ejabberd commands in different ways: `mod_rest` (HTTP POST service), `mod_shcommands` (ejabberd WebAdmin page).

Any module in ejabberd can add its own command(s) through ejabberd Erlang/Elixir API, making the whole system totally extensible. A third-party module can expose its own command(s) and feel like a real part of the system. A module that exposes commands allows server admins to expose it the way they want.

ejabberd commands are universal, extensible and widely available through various configurable entrypoints.

Note: The XML-RPC API still works but is deprecated in favor of the ReST API. You should migrate to ReST if you are using it.

The role of ejabberd API

As we have seen, ejabberd API role is to provide and control access to ejabberd commands over HTTP/HTTPS.

Thus, ejabberd API primary goal is to grant access to some or all ejabberd "commands".

An admin ejabberd ReST API requires:

- At least one admin user, if you plan to check credentials for command access (You can alternatively rely on originating IP addresses).

- HTTP/HTTPS handlers configured to expose the desired commands.

- The selection of authentication mechanisms that can be used to access the API. Two mechanisms are available to access the HTTP API:

- Basic authentication. This mechanism is enabled by default.

- OAuth 2.0 token based authentication. It has to be explicitly added to the config file.

Learning the basics

The first resources to read to learn about ejabberd ReST API configuration are the following:

- Simple API configuration

- Using ejabberd client API libraries and tools

The list of available commands is available in the API Reference section. Additionally, you can check at runtime what commands are available in your installed server using ejabberdctl:

```
❯ ejabberdctl
Usage: ejabberdctl [--no-timeout] [--node nodename] [--version api_version] command [arguments]

Available commands in this ejabberd node:
  backup file
            Store internal Mnesia database to binary backup file
  ban_account user host reason
            Ban an account: kick sessions and set random password
  ...

❯ ejabberdctl help
  ...

❯ ejabberdctl help ban_account
  ...
```

Next steps

You can dig deeper into ejabberd ReST API configuration on the following pages:

- API Permissions
- OAuth Support
- Administration API Commands

API Reference

This section describes API commands of ejabberd 24.02. If you are using an old ejabberd release, please refer to the corresponding archived version of this page in the Archive. The commands that changed in this version are marked with ⬤.

abort_delete_old_mam_messages

🏷 added in 22.05

Abort currently running delete old MAM messages operation

Arguments:

- *host* :: string : Name of host where operation should be aborted

Result:

- *status* :: string : Status text

Tags: purge

Module: mod_mam

Examples:

```
POST /api/abort_delete_old_mam_messages
{
  "host": "localhost"
}

HTTP/1.1 200 OK
"Operation aborted"
```

abort_delete_old_messages

🏷 added in 22.05

Abort currently running delete old offline messages operation

Arguments:

- *host* :: string : Name of host where operation should be aborted

Result:

- *status* :: string : Status text

Tags: purge

Examples:

```
POST /api/abort_delete_old_messages
{
  "host": "localhost"
}

HTTP/1.1 200 OK
"Operation aborted"
```

add_rosteritem ⬤

🏷 updated in 24.02

Add an item to a user's roster (supports ODBC)

Arguments:

- *localuser* :: string : User name
- *localhost* :: string : Server name
- *user* :: string : Contact user name
- *host* :: string : Contact server name
- *nick* :: string : Nickname
- *groups* :: [group::string] : Groups
- *subs* :: string : Subscription

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: roster, v1

Module: mod_admin_extra

Examples:

```
POST /api/add_rosteritem
{
  "localuser": "user1",
  "localhost": "myserver.com",
  "user": "user2",
  "host": "myserver.com",
  "nick": "User 2",
  "groups": [
    "Friends",
    "Team 1"
  ],
  "subs": "both"
}

HTTP/1.1 200 OK
""
```

backup

Backup the Mnesia database to a binary file

Arguments:

- *file* :: string : Full path for the destination backup file

Result:

- *res* :: string : Raw result string

Tags: mnesia

Examples:

```
POST /api/backup
{
  "file": "/var/lib/ejabberd/database.backup"
}

HTTP/1.1 200 OK
"Success"
```

ban_account

Ban an account: kick sessions and set random password

Arguments:

- *user* :: string : User name to ban
- *host* :: string : Server name
- *reason* :: string : Reason for banning user

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: accounts

Module: mod_admin_extra

Examples:

```
POST /api/ban_account
{
  "user": "attacker",
  "host": "myserver.com",
  "reason": "Spaming other users"
}

HTTP/1.1 200 OK
""
```

bookmarks_to_pep

Export private XML storage bookmarks to PEP

Arguments:

- *user* :: string : Username
- *host* :: string : Server

Result:

- *res* :: string : Raw result string

Tags: private

Module: mod_private

Examples:

```
POST /api/bookmarks_to_pep
{
  "user": "bob",
  "host": "example.com"
}

HTTP/1.1 200 OK
"Bookmarks exported"
```

change_password

Change the password of an account

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *newpass* :: string : New password for user

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: accounts

Module: mod_admin_extra

Examples:

```
POST /api/change_password
{
  "user": "peter",
  "host": "myserver.com",
  "newpass": "blank"
}

HTTP/1.1 200 OK
""
```

change_room_option

Change an option in a MUC room

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service
- *option* :: string : Option name
- *value* :: string : Value to assign

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/change_room_option
{
  "name": "room1",
  "service": "muc.example.com",
  "option": "members_only",
  "value": "true"
}

HTTP/1.1 200 OK
""
```

check_account

Check if an account exists or not

Arguments:

- *user* :: string : User name to check
- *host* :: string : Server to check

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: accounts

Module: mod_admin_extra

Examples:

```
POST /api/check_account
{
  "user": "peter",
  "host": "myserver.com"
}

HTTP/1.1 200 OK
""
```

check_password

Check if a password is correct

Arguments:

- *user* :: string : User name to check

- *host* :: string : Server to check

- *password* :: string : Password to check

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: accounts

Module: mod_admin_extra

Examples:

```
POST /api/check_password
{
  "user": "peter",
  "host": "myserver.com",
  "password": "secret"
}

HTTP/1.1 200 OK
""
```

check_password_hash

Check if the password hash is correct

Allows hash methods from the Erlang/OTP crypto application.

Arguments:

- *user* :: string : User name to check

- *host* :: string : Server to check

- *passwordhash* :: string : Password's hash value

- *hashmethod* :: string : Name of hash method

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: accounts

Module: mod_admin_extra

Examples:

```
POST /api/check_password_hash
{
  "user": "peter",
  "host": "myserver.com",
  "passwordhash": "5ebe2294ecd0e0f08eab7690d2a6ee69",
  "hashmethod": "md5"
}

HTTP/1.1 200 OK
""
```

clear_cache

Clear database cache on all nodes

Arguments:

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: server

Examples:

```
POST /api/clear_cache
{

}

HTTP/1.1 200 OK
""
```

compile

Recompile and reload Erlang source code file

Arguments:

- *file* :: string : Filename of erlang source file to compile

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: erlang

Module: mod_admin_extra

Examples:

```
POST /api/compile
{
  "file": "/home/me/srcs/ejabberd/mod_example.erl"
}

HTTP/1.1 200 OK
""
```

connected_users

List all established sessions

Arguments:

Result:

- *connected_users* :: [sessions::string] : List of users sessions

Tags: session

Examples:

```
POST /api/connected_users
{

}

HTTP/1.1 200 OK
[
  "user1@example.com",
  "user2@example.com"
]
```

connected_users_info

List all established sessions and their information

Arguments:

Result:

- *connected_users_info* :: [{jid::string, connection::string, ip::string, port::integer, priority::integer, node::string, uptime::integer, status::string, resource::string, statustext::string}]

Tags: session

Module: mod_admin_extra

Examples:

```
POST /api/connected_users_info
{

}

HTTP/1.1 200 OK
[
  {
    "jid": "user1@myserver.com/tka",
    "connection": "c2s",
    "ip": "127.0.0.1",
    "port": 42656,
    "priority": 8,
    "node": "ejabberd@localhost",
    "uptime": 231,
    "status": "dnd",
    "resource": "tka",
    "statustext": ""
  }
]
```

connected_users_number

Get the number of established sessions

Arguments:

Result:

- *num_sessions* :: integer

Tags: session, statistics

Examples:

```
POST /api/connected_users_number
{

}
```

```
HTTP/1.1 200 OK
2
```

connected_users_vhost

Get the list of established sessions in a vhost

Arguments:

- *host* :: string : Server name

Result:

- *connected_users_vhost* :: [sessions::string]

Tags: session

Module: mod_admin_extra

Examples:

```
POST /api/connected_users_vhost
{
  "host": "myexample.com"
}

HTTP/1.1 200 OK
[
  "user1@myserver.com/tka",
  "user2@localhost/tka"
]
```

convert_to_scram

Convert the passwords of users to SCRAM

Arguments:

- *host* :: string : Vhost which users' passwords will be scrammed

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: sql

Examples:

```
POST /api/convert_to_scram
{
  "host": "example.com"
}

HTTP/1.1 200 OK
""
```

convert_to_yaml

Convert the input file from Erlang to YAML format

Arguments:

- *in* :: string : Full path to the original configuration file
- *out* :: string : And full path to final file

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: config

Examples:

```
POST /api/convert_to_yaml
{
  "in": "/etc/ejabberd/ejabberd.cfg",
  "out": "/etc/ejabberd/ejabberd.yml"
}

HTTP/1.1 200 OK
""
```

create_room

Create a MUC room name@service in host

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service
- *host* :: string : Server host

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/create_room
{
  "name": "room1",
  "service": "muc.example.com",
  "host": "example.com"
}

HTTP/1.1 200 OK
""
```

create_room_with_opts

Create a MUC room name@service in host with given options

The syntax of `affiliations` is: `Type:JID,Type:JID` . The syntax of `subscribers` is: `JID:Nick:Node:Node2:Node3,JID:Nick:Node` .

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service
- *host* :: string : Server host
- *options* :: [{name::string, value::string}] : List of options

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc_room, muc_sub

Module: mod_muc_admin

Examples:

```
POST /api/create_room_with_opts
{
  "name": "room1",
  "service": "muc.example.com",
  "host": "localhost",
  "options": [
    {
      "name": "members_only",
      "value": "true"
    },
    {
      "name": "affiliations",
      "value": "owner:bob@example.com,member:peter@example.com"
    },
    {
      "name": "subscribers",
      "value": "bob@example.com:Bob:messages:subject,anne@example.com:Anne:messages"
    }
  ]
}

HTTP/1.1 200 OK
""
```

create_rooms_file

Create the rooms indicated in file

Provide one room JID per line. Rooms will be created after restart.

Arguments:

• *file* :: string : Path to the text file with one room JID per line

Result:

• *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/create_rooms_file
{
  "file": "/home/ejabberd/rooms.txt"
}

HTTP/1.1 200 OK
""
```

delete_expired_messages

Delete expired offline messages from database

Arguments:

Result:

• *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: purge

Examples:

```
POST /api/delete_expired_messages
{
```

```
    }

HTTP/1.1 200 OK
""
```

delete_expired_pubsub_items

🏷 added in 21.12

Delete expired PubSub items

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: purge

Module: mod_pubsub

Examples:

```
POST /api/delete_expired_pubsub_items
{
}

HTTP/1.1 200 OK
""
```

delete_mnesia

Delete elements in Mnesia database for a given vhost

Arguments:

- *host* :: string : Vhost which content will be deleted in Mnesia database

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: mnesia

Examples:

```
POST /api/delete_mnesia
{
  "host": "example.com"
}

HTTP/1.1 200 OK
""
```

delete_old_mam_messages

Delete MAM messages older than DAYS

Valid message TYPEs: `chat` , `groupchat` , `all` .

Arguments:

- *type* :: string : Type of messages to delete (`chat` , `groupchat` , `all`)
- *days* :: integer : Days to keep messages

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: purge

Module: mod_mam

Examples:

```
POST /api/delete_old_mam_messages
{
  "type": "all",
  "days": 31
}

HTTP/1.1 200 OK
""
```

delete_old_mam_messages_batch

🏷 added in 22.05

Delete MAM messages older than DAYS

Valid message TYPEs: `chat` , `groupchat` , `all` .

Arguments:

- *host* :: string : Name of host where messages should be deleted
- *type* :: string : Type of messages to delete (`chat` , `groupchat` , `all`)
- *days* :: integer : Days to keep messages
- *batch_size* :: integer : Number of messages to delete per batch
- *rate* :: integer : Desired rate of messages to delete per minute

Result:

- *res* :: string : Raw result string

Tags: purge

Module: mod_mam

Examples:

```
POST /api/delete_old_mam_messages_batch
{
  "host": "localhost",
  "type": "all",
  "days": 31,
  "batch_size": 1000,
  "rate": 10000
}

HTTP/1.1 200 OK
"Removal of 5000 messages in progress"
```

delete_old_mam_messages_status

🏷 added in 22.05

Status of delete old MAM messages operation

Arguments:

- *host* :: string : Name of host where messages should be deleted

Result:

- *status* :: string : Status test

Tags: purge

Module: mod_mam

Examples:

```
POST /api/delete_old_mam_messages_status
{
  "host": "localhost"
}

HTTP/1.1 200 OK
"Operation in progress, delete 5000 messages"
```

delete_old_messages

Delete offline messages older than DAYS

Arguments:

- *days* :: integer : Number of days

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: purge

Examples:

```
POST /api/delete_old_messages
{
  "days": 31
}

HTTP/1.1 200 OK
""
```

delete_old_messages_batch

🏷 added in 22.05

Delete offline messages older than DAYS

Arguments:

- *host* :: string : Name of host where messages should be deleted
- *days* :: integer : Days to keep messages
- *batch_size* :: integer : Number of messages to delete per batch
- *rate* :: integer : Desired rate of messages to delete per minute

Result:

- *res* :: string : Raw result string

Tags: purge

Examples:

```
POST /api/delete_old_messages_batch
{
  "host": "localhost",
  "days": 31,
```

```
    "batch_size": 1000,
    "rate": 10000
}

HTTP/1.1 200 OK
"Removal of 5000 messages in progress"
```

delete_old_messages_status

🏷 added in 22.05

Status of delete old offline messages operation

Arguments:

- *host* :: string : Name of host where messages should be deleted

Result:

- *status* :: string : Status test

Tags: purge

Examples:

```
POST /api/delete_old_messages_status
{
  "host": "localhost"
}

HTTP/1.1 200 OK
"Operation in progress, delete 5000 messages"
```

delete_old_pubsub_items

🏷 added in 21.12

Keep only NUMBER of PubSub items per node

Arguments:

- *number* :: integer : Number of items to keep per node

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: purge

Module: mod_pubsub

Examples:

```
POST /api/delete_old_pubsub_items
{
  "number": 1000
}

HTTP/1.1 200 OK
""
```

delete_old_push_sessions

Remove push sessions older than DAYS

Arguments:

- *days* :: integer

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: purge

Module: mod_push

Examples:

```
POST /api/delete_old_push_sessions
{
  "days": 1
}

HTTP/1.1 200 OK
""
```

delete_old_users

Delete users that didn't log in last days, or that never logged

To protect admin accounts, configure this for example:

```
access_rules:
  protect_old_users:
    - allow: admin
    - deny: all
```

Arguments:

- *days* :: integer : Last login age in days of accounts that should be removed

Result:

- *res* :: string : Raw result string

Tags: accounts, purge

Module: mod_admin_extra

Examples:

```
POST /api/delete_old_users
{
  "days": 30
}

HTTP/1.1 200 OK
"Deleted 2 users: ["oldman@myserver.com", "test@myserver.com"]"
```

delete_old_users_vhost

Delete users that didn't log in last days in vhost, or that never logged

To protect admin accounts, configure this for example:

```
access_rules:
  delete_old_users:
    - deny: admin
    - allow: all
```

Arguments:

- *host* :: string : Server name

- *days* :: integer : Last login age in days of accounts that should be removed

Result:

- *res* :: string : Raw result string

Tags: accounts, purge

Module: mod_admin_extra

Examples:

```
POST /api/delete_old_users_vhost
{
  "host": "myserver.com",
  "days": 30
}

HTTP/1.1 200 OK
"Deleted 2 users: ["oldman@myserver.com", "test@myserver.com"]"
```

delete_rosteritem

Delete an item from a user's roster (supports ODBC)

Arguments:

- *localuser* :: string : User name

- *localhost* :: string : Server name

- *user* :: string : Contact user name

- *host* :: string : Contact server name

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: roster

Module: mod_admin_extra

Examples:

```
POST /api/delete_rosteritem
{
  "localuser": "user1",
  "localhost": "myserver.com",
  "user": "user2",
  "host": "myserver.com"
}

HTTP/1.1 200 OK
""
```

destroy_room

Destroy a MUC room

Arguments:

- *name* :: string : Room name

- *service* :: string : MUC service

Result:

• *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/destroy_room
{
  "name": "room1",
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
""
```

destroy_rooms_file

Destroy the rooms indicated in file

Provide one room JID per line.

Arguments:

• *file* :: string : Path to the text file with one room JID per line

Result:

• *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/destroy_rooms_file
{
  "file": "/home/ejabberd/rooms.txt"
}

HTTP/1.1 200 OK
""
```

dump

Dump the Mnesia database to a text file

Arguments:

• *file* :: string : Full path for the text file

Result:

• *res* :: string : Raw result string

Tags: mnesia

Examples:

```
POST /api/dump
{
  "file": "/var/lib/ejabberd/database.txt"
}

HTTP/1.1 200 OK
"Success"
```

dump_config

Dump configuration in YAML format as seen by ejabberd

Arguments:

- *out* :: string : Full path to output file

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: config

Examples:

```
POST /api/dump_config
{
  "out": "/tmp/ejabberd.yml"
}

HTTP/1.1 200 OK
""
```

dump_table

Dump a Mnesia table to a text file

Arguments:

- *file* :: string : Full path for the text file
- *table* :: string : Table name

Result:

- *res* :: string : Raw result string

Tags: mnesia

Examples:

```
POST /api/dump_table
{
  "file": "/var/lib/ejabberd/table-muc-registered.txt",
  "table": "muc_registered"
}

HTTP/1.1 200 OK
"Success"
```

export2sql

Export virtual host information from Mnesia tables to SQL file

Configure the modules to use SQL, then call this command. After correctly exported the database of a vhost, you may want to delete from mnesia with the delete_mnesia API.

Arguments:

- *host* :: string : Vhost
- *file* :: string : Full path to the destination SQL file

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: mnesia

Examples:

```
POST /api/export2sql
{
  "host": "example.com",
  "file": "/var/lib/ejabberd/example.com.sql"
}

HTTP/1.1 200 OK
""
```

export_piefxis

Export data of all users in the server to PIEFXIS files (XEP-0227)

Arguments:

- *dir* :: string : Full path to a directory

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: mnesia

Examples:

```
POST /api/export_piefxis
{
  "dir": "/var/lib/ejabberd/"
}

HTTP/1.1 200 OK
""
```

export_piefxis_host

Export data of users in a host to PIEFXIS files (XEP-0227)

Arguments:

- *dir* :: string : Full path to a directory
- *host* :: string : Vhost to export

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: mnesia

Examples:

```
POST /api/export_piefxis_host
{
  "dir": "/var/lib/ejabberd/",
  "host": "example.com"
}

HTTP/1.1 200 OK
""
```

gc

🏷 added in 20.01

Force full garbage collection

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: server

Examples:

```
POST /api/gc
{

}

HTTP/1.1 200 OK
""
```

gen_html_doc_for_commands

Generates html documentation for ejabberd_commands

Arguments:

- *file* :: string : Path to file where generated documentation should be stored
- *regexp* :: string : Regexp matching names of commands or modules that will be included inside generated document
- *examples* :: string : Comma separated list of languages (chosen from `java`, `perl`, `xmlrpc`, `json`) that will have example invocation include in markdown document

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: documentation

Examples:

```
POST /api/gen_html_doc_for_commands
{
  "file": "/home/me/docs/api.html",
  "regexp": "mod_admin",
  "examples": "java,json"
}

HTTP/1.1 200 OK
""
```

gen_markdown_doc_for_commands

Generates markdown documentation for ejabberd_commands

Arguments:

- *file* :: string : Path to file where generated documentation should be stored
- *regexp* :: string : Regexp matching names of commands or modules that will be included inside generated document, or `runtime` to get commands registered at runtime
- *examples* :: string : Comma separated list of languages (chosen from `java`, `perl`, `xmlrpc`, `json`) that will have example invocation include in markdown document

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: documentation

Examples:

```
POST /api/gen_markdown_doc_for_commands
{
  "file": "/home/me/docs/api.html",
  "regexp": "mod_admin",
  "examples": "java,json"
}

HTTP/1.1 200 OK
""
```

gen_markdown_doc_for_tags

🏷 added in 21.12

Generates markdown documentation for ejabberd_commands

Arguments:

- *file* :: string : Path to file where generated documentation should be stored

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: documentation

Examples:

```
POST /api/gen_markdown_doc_for_tags
{
  "file": "/home/me/docs/tags.md"
}

HTTP/1.1 200 OK
""
```

get_cookie

Get the Erlang cookie of this node

Arguments:

Result:

- *cookie* :: string : Erlang cookie used for authentication by ejabberd

Tags: erlang

Module: mod_admin_extra

Examples:

```
POST /api/get_cookie
{

}

HTTP/1.1 200 OK
"MWTAVMODFELNLSMYXPPD"
```

get_last

Get last activity information

Timestamp is UTC and XEP-0082 format, for example: `2017-02-23T22:25:28.063062Z    ONLINE`

Arguments:

- *user* :: string : User name
- *host* :: string : Server name

Result:

- *last_activity* :: {timestamp::string, status::string} : Last activity timestamp and status

Tags: last

Module: mod_admin_extra

Examples:

```
POST /api/get_last
{
  "user": "user1",
  "host": "myserver.com"
}

HTTP/1.1 200 OK
{
  "timestamp": "2017-06-30T14:32:16.060684Z",
  "status": "ONLINE"
}
```

get_loglevel

Get the current loglevel

Arguments:

Result:

- *levelatom* :: string : Tuple with the log level number, its keyword and description

Tags: logs

Examples:

```
POST /api/get_loglevel
{

}

HTTP/1.1 200 OK
"warning"
```

get_offline_count

Get the number of unread offline messages

Arguments:

- *user* :: string
- *host* :: string

Result:

- *value* :: integer : Number

Tags: offline

Module: mod_admin_extra

Examples:

```
POST /api/get_offline_count
{
  "user": "aaaaa",
  "host": "bbbbb"
}

HTTP/1.1 200 OK
5
```

get_presence

Retrieve the resource with highest priority, and its presence (show and status message) for a given user.

The `jid` value contains the user JID with resource.

The `show` value contains the user presence flag. It can take limited values:

- `available`
- `chat` (Free for chat)
- `away`
- `dnd` (Do not disturb)
- `xa` (Not available, extended away)
- `unavailable` (Not connected)

`status` is a free text defined by the user client.

Arguments:

- *user* :: string : User name
- *host* :: string : Server name

Result:

- *presence* :: {jid::string, show::string, status::string}

Tags: session

Module: mod_admin_extra

Examples:

```
POST /api/get_presence
{
  "user": "peter",
  "host": "myexample.com"
}

HTTP/1.1 200 OK
{
  "jid": "user1@myserver.com/tka",
  "show": "dnd",
  "status": "Busy"
}
```

get_room_affiliation

Get affiliation of a user in MUC room

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service
- *jid* :: string : User JID

Result:

- *affiliation* :: string : Affiliation of the user

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/get_room_affiliation
{
  "name": "room1",
  "service": "muc.example.com",
  "jid": "user1@example.com"
}

HTTP/1.1 200 OK
"member"
```

get_room_affiliations

Get the list of affiliations of a MUC room

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service

Result:

- *affiliations* :: [{username::string, domain::string, affiliation::string, reason::string}] : The list of affiliations with username, domain, affiliation and reason

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/get_room_affiliations
{
  "name": "room1",
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
[
  {
    "username": "user1",
    "domain": "example.com",
    "affiliation": "member",
    "reason": "member"
  }
]
```

get_room_history

🏷 added in 23.04

Get history of messages stored inside MUC room state

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service

Result:

- *history* :: [{timestamp::string, message::string}]

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/get_room_history
{
  "name": "room1",
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
[
  {
    "timestamp": "aaaaa",
    "message": "bbbbb"
  },
  {
    "timestamp": "ccccc",
    "message": "ddddd"
  }
]
```

get_room_occupants

Get the list of occupants of a MUC room

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service

Result:

- *occupants* :: [{jid::string, nick::string, role::string}] : The list of occupants with JID, nick and affiliation

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/get_room_occupants
{
  "name": "room1",
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
[
  {
    "jid": "user1@example.com/psi",
    "nick": "User 1",
    "role": "owner"
  }
]
```

get_room_occupants_number

Get the number of occupants of a MUC room

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service

Result:

- *occupants* :: integer : Number of room occupants

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/get_room_occupants_number
{
  "name": "room1",
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
7
```

get_room_options

Get options from a MUC room

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service

Result:

- *options* :: [{name::string, value::string}] : List of room options tuples with name and value

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/get_room_options
{
  "name": "room1",
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
[
  {
    "name": "members_only",
    "value": "true"
  }
]
```

get_roster

🏷️ improved in 23.10

Get list of contacts in a local user roster

subscription can be: none , from , to , both .

pending can be: in , out , none .

Arguments:

- *user* :: string
- *host* :: string

Result:

- *contacts* :: [{jid::string, nick::string, subscription::string, pending::string, groups::[group::string]}]

Tags: roster

Module: mod_admin_extra

Examples:

```
POST /api/get_roster
{
  "user": "aaaaa",
  "host": "bbbbb"
}

HTTP/1.1 200 OK
[
  {
    "jid": "aaaaa",
    "nick": "bbbbb",
    "subscription": "ccccc",
    "pending": "ddddd",
    "groups": [
      "eeeee",
      "fffff"
    ]
  },
  {
    "jid": "ggggg",
    "nick": "hhhhh",
    "subscription": "iiiii",
    "pending": "jjjjj",
    "groups": [
      "kkkkk",
      "lllll"
    ]
  }
]
```

get_subscribers

List subscribers of a MUC conference

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service

Result:

- *subscribers* :: [jid::string] : The list of users that are subscribed to that room

Tags: muc_room, muc_sub

Module: mod_muc_admin

Examples:

```
POST /api/get_subscribers
{
  "name": "room1",
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
[
  "user2@example.com",
  "user3@example.com"
]
```

get_user_rooms

Get the list of rooms where this user is occupant

Arguments:

- *user* :: string : Username
- *host* :: string : Server host

Result:

- *rooms* :: [room::string]

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/get_user_rooms
{
  "user": "tom",
  "host": "example.com"
}

HTTP/1.1 200 OK
[
  "room1@muc.example.com",
  "room2@muc.example.com"
]
```

get_user_subscriptions

🏷 added in 21.04

Get the list of rooms where this user is subscribed

Arguments:

- *user* :: string : Username
- *host* :: string : Server host

Result:

- *rooms* :: [{roomjid::string, usernick::string, nodes::[node::string]}]

Tags: muc, muc_sub

Module: mod_muc_admin

Examples:

```
POST /api/get_user_subscriptions
{
  "user": "tom",
  "host": "example.com"
}

HTTP/1.1 200 OK
[
  {
    "roomjid": "room1@muc.example.com",
    "usernick": "Tommy",
    "nodes": [
      "mucsub:config"
    ]
  }
]
```

get_vcard

Get content from a vCard field

Some vcard field names in `get`/`set_vcard` are:

- FN - Full Name
- NICKNAME - Nickname
- BDAY - Birthday
- TITLE - Work: Position
- ROLE - Work: Role

For a full list of vCard fields check XEP-0054: vcard-temp

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *name* :: string : Field name

Result:

- *content* :: string : Field content

Tags: vcard

Module: mod_admin_extra

Examples:

```
POST /api/get_vcard
{
  "user": "user1",
  "host": "myserver.com",
  "name": "NICKNAME"
}

HTTP/1.1 200 OK
"User 1"
```

get_vcard2

Get content from a vCard subfield

Some vcard field names and subnames in `get`/`set_vcard2` are:

- N FAMILY - Family name
- N GIVEN - Given name
- N MIDDLE - Middle name
- ADR CTRY - Address: Country
- ADR LOCALITY - Address: City
- TEL HOME - Telephone: Home
- TEL CELL - Telephone: Cellphone
- TEL WORK - Telephone: Work
- TEL VOICE - Telephone: Voice
- EMAIL USERID - E-Mail Address
- ORG ORGNAME - Work: Company
- ORG ORGUNIT - Work: Department

For a full list of vCard fields check XEP-0054: vcard-temp

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *name* :: string : Field name
- *subname* :: string : Subfield name

Result:

- *content* :: string : Field content

Tags: vcard

Module: mod_admin_extra

Examples:

```
POST /api/get_vcard2
{
  "user": "user1",
  "host": "myserver.com",
  "name": "N",
  "subname": "FAMILY"
}

HTTP/1.1 200 OK
"Schubert"
```

get_vcard2_multi

Get multiple contents from a vCard field

Some vcard field names and subnames in `get` / `set_vcard2` are:

- N FAMILY - Family name
- N GIVEN - Given name
- N MIDDLE - Middle name
- ADR CTRY - Address: Country
- ADR LOCALITY - Address: City
- TEL HOME - Telephone: Home
- TEL CELL - Telephone: Cellphone
- TEL WORK - Telephone: Work
- TEL VOICE - Telephone: Voice
- EMAIL USERID - E-Mail Address
- ORG ORGNAME - Work: Company
- ORG ORGUNIT - Work: Department

For a full list of vCard fields check XEP-0054: vcard-temp

Arguments:

- *user* :: string
- *host* :: string
- *name* :: string
- *subname* :: string

Result:

- *contents* :: [value::string]

Tags: vcard

Module: mod_admin_extra

Examples:

```
POST /api/get_vcard2_multi
{
  "user": "aaaaa",
  "host": "bbbbb",
  "name": "ccccc",
  "subname": "ddddd"
}

HTTP/1.1 200 OK
[
  "aaaaa",
  "bbbbb"
]
```

halt

🏷 added in 23.10

Halt ejabberd abruptly with status code 1

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: server

Examples:

```
POST /api/halt
{

}

HTTP/1.1 200 OK
""
```

help

Get list of commands, or help of a command (only ejabberdctl)

This command is exclusive for the ejabberdctl command-line script, don't attempt to execute it using any other API frontend.

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: ejabberdctl

Examples:

```
POST /api/help
{

}

HTTP/1.1 200 OK
""
```

import_dir

Import users data from jabberd14 spool dir

Arguments:

- *file* :: string : Full path to the jabberd14 spool directory

Result:

- *res* :: string : Raw result string

Tags: mnesia

Examples:

```
POST /api/import_dir
{
  "file": "/var/lib/ejabberd/jabberd14/"
}

HTTP/1.1 200 OK
"Success"
```

import_file

Import user data from jabberd14 spool file

Arguments:

- *file* :: string : Full path to the jabberd14 spool file

Result:

- *res* :: string : Raw result string

Tags: mnesia

Examples:

```
POST /api/import_file
{
  "file": "/var/lib/ejabberd/jabberd14.spool"
}

HTTP/1.1 200 OK
"Success"
```

import_piefxis

Import users data from a PIEFXIS file (XEP-0227)

Arguments:

- *file* :: string : Full path to the PIEFXIS file

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: mnesia

Examples:

```
POST /api/import_piefxis
{
  "file": "/var/lib/ejabberd/example.com.xml"
}
```

```
HTTP/1.1 200 OK
""
```

import_prosody

Import data from Prosody

Note: this requires ejabberd to be compiled with `./configure --enable-lua` (which installs the `luerl` library).

Arguments:

- *dir* :: string : Full path to the Prosody data directory

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: mnesia, sql

Examples:

```
POST /api/import_prosody
{
  "dir": "/var/lib/prosody/datadump/"
}

HTTP/1.1 200 OK
""
```

incoming_s2s_number

Number of incoming s2s connections on the node

Arguments:

Result:

- *s2s_incoming* :: integer

Tags: statistics, s2s

Examples:

```
POST /api/incoming_s2s_number
{

}

HTTP/1.1 200 OK
1
```

install_fallback

Install Mnesia database from a binary backup file

The binary backup file is installed as fallback: it will be used to restore the database at the next ejabberd start. This means that, after running this command, you have to restart ejabberd. This command requires less memory than restore API.

Arguments:

- *file* :: string : Full path to the fallback file

Result:

- *res* :: string : Raw result string

Tags: mnesia

Examples:

```
POST /api/install_fallback
{
  "file": "/var/lib/ejabberd/database.fallback"
}

HTTP/1.1 200 OK
"Success"
```

join_cluster

Join this node into the cluster handled by Node

This command works only with ejabberdctl, not mod_http_api or other code that runs inside the same ejabberd node that will be joined.

Arguments:

- *node* :: string : Nodename of the node to join

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: cluster

Examples:

```
POST /api/join_cluster
{
  "node": "ejabberd1@machine7"
}

HTTP/1.1 200 OK
""
```

kick_session

Kick a user session

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *resource* :: string : User's resource
- *reason* :: string : Reason for closing session

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: session

Module: mod_admin_extra

Examples:

```
POST /api/kick_session
{
  "user": "peter",
  "host": "myserver.com",
  "resource": "Psi",
  "reason": "Stuck connection"
}
```

```
HTTP/1.1 200 OK
""
```

kick_user

Disconnect user's active sessions

Arguments:

- *user* :: string : User name
- *host* :: string : Server name

Result:

- *num_resources* :: integer : Number of resources that were kicked

Tags: session

Examples:

```
POST /api/kick_user
{
  "user": "user1",
  "host": "example.com"
}

HTTP/1.1 200 OK
3
```

leave_cluster

Remove and shutdown Node from the running cluster

This command can be run from any running node of the cluster, even the node to be removed. In the removed node, this command works only when using ejabberdctl, not mod_http_api or other code that runs inside the same ejabberd node that will leave.

Arguments:

- *node* :: string : Nodename of the node to kick from the cluster

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: cluster

Examples:

```
POST /api/leave_cluster
{
  "node": "ejabberd1@machine8"
}

HTTP/1.1 200 OK
""
```

list_certificates

Lists all ACME certificates

Arguments:

Result:

- *certificates* :: [{domain::string, file::string, used::string}]

Tags: acme

Examples:

```
POST /api/list_certificates
{

}

HTTP/1.1 200 OK
[
  {
    "domain": "aaaaa",
    "file": "bbbbb",
    "used": "ccccc"
  },
  {
    "domain": "ddddd",
    "file": "eeeee",
    "used": "fffff"
  }
]
```

list_cluster

List nodes that are part of the cluster handled by Node

Arguments:

Result:

- *nodes* :: [node::string]

Tags: cluster

Examples:

```
POST /api/list_cluster
{

}

HTTP/1.1 200 OK
[
  "ejabberd1@machine7",
  "ejabberd1@machine8"
]
```

load

Restore Mnesia database from a text dump file

Restore immediately. This is not recommended for big databases, as it will consume much time, memory and processor. In that case it's preferable to use backup API and install_fallback API.

Arguments:

- *file* :: string : Full path to the text file

Result:

- *res* :: string : Raw result string

Tags: mnesia

Examples:

```
POST /api/load
{
  "file": "/var/lib/ejabberd/database.txt"
}

HTTP/1.1 200 OK
"Success"
```

man

🏷 added in 20.01

Generate Unix manpage for current ejabberd version

Arguments:

Result:

- *res* :: string : Raw result string

Tags: documentation

Examples:

```
POST /api/man
{

}

HTTP/1.1 200 OK
"Success"
```

mnesia_change_nodename

Change the erlang node name in a backup file

Arguments:

- *oldnodename* :: string : Name of the old erlang node
- *newnodename* :: string : Name of the new node
- *oldbackup* :: string : Path to old backup file
- *newbackup* :: string : Path to the new backup file

Result:

- *res* :: string : Raw result string

Tags: mnesia

Examples:

```
POST /api/mnesia_change_nodename
{
  "oldnodename": "ejabberd@machine1",
  "newnodename": "ejabberd@machine2",
  "oldbackup": "/var/lib/ejabberd/old.backup",
  "newbackup": "/var/lib/ejabberd/new.backup"
}

HTTP/1.1 200 OK
"Success"
```

mnesia_info

Dump info on global Mnesia state

Arguments:

Result:

- *res* :: string

Tags: mnesia

Examples:

```
POST /api/mnesia_info
{
}

HTTP/1.1 200 OK
"aaaaa"
```

mnesia_info_ctl ●

🏷 renamed in 24.02

Show information of Mnesia system (only ejabberdctl)

This command is exclusive for the ejabberdctl command-line script, don't attempt to execute it using any other API frontend.

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: ejabberdctl, mnesia

Examples:

```
POST /api/mnesia_info_ctl
{
}

HTTP/1.1 200 OK
""
```

mnesia_table_info

Dump info on Mnesia table state

Arguments:

- *table* :: string : Mnesia table name

Result:

- *res* :: string

Tags: mnesia

Examples:

```
POST /api/mnesia_table_info
{
  "table": "roster"
}

HTTP/1.1 200 OK
"aaaaa"
```

module_check

Check the contributed module repository compliance

Arguments:

- *module* :: string : Module name

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: modules

Examples:

```
POST /api/module_check
{
  "module": "mod_rest"
}

HTTP/1.1 200 OK
""
```

module_install

Compile, install and start an available contributed module

Arguments:

- *module* :: string : Module name

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: modules

Examples:

```
POST /api/module_install
{
  "module": "mod_rest"
}

HTTP/1.1 200 OK
""
```

module_uninstall

Uninstall a contributed module

Arguments:

- *module* :: string : Module name

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: modules

Examples:

```
POST /api/module_uninstall
{
  "module": "mod_rest"
}
```

```
HTTP/1.1 200 OK
""
```

module_upgrade

Upgrade the running code of an installed module

In practice, this uninstalls and installs the module

Arguments:

- *module* :: string : Module name

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: modules

Examples:

```
POST /api/module_upgrade
{
  "module": "mod_rest"
}

HTTP/1.1 200 OK
""
```

modules_available

List the contributed modules available to install

Arguments:

Result:

- *modules* :: [{name::string, summary::string}] : List of tuples with module name and description

Tags: modules

Examples:

```
POST /api/modules_available
{

}

HTTP/1.1 200 OK
{
  "mod_cron": "Execute scheduled commands",
  "mod_rest": "ReST frontend"
}
```

modules_installed

List the contributed modules already installed

Arguments:

Result:

- *modules* :: [{name::string, summary::string}] : List of tuples with module name and description

Tags: modules

Examples:

```
POST /api/modules_installed
{

}

HTTP/1.1 200 OK
{
  "mod_cron": "Execute scheduled commands",
  "mod_rest": "ReST frontend"
}
```

modules_update_specs

Update the module source code from Git

A connection to Internet is required

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: modules

Examples:

```
POST /api/modules_update_specs
{

}

HTTP/1.1 200 OK
""
```

muc_online_rooms

List existing rooms

Ask for a specific host, or `global` to use all vhosts.

Arguments:

- *service* :: string : MUC service, or `global` for all

Result:

- *rooms* :: [room::string] : List of rooms

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/muc_online_rooms
{
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
[
  "room1@muc.example.com",
  "room2@muc.example.com"
]
```

muc_online_rooms_by_regex

List existing rooms filtered by regexp

Ask for a specific host, or `global` to use all vhosts.

Arguments:

- *service* :: string : MUC service, or `global` for all
- *regex* :: string : Regex pattern for room name

Result:

- *rooms* :: [{jid::string, public::string, participants::integer}] : List of rooms with summary

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/muc_online_rooms_by_regex
{
  "service": "muc.example.com",
  "regex": "^prefix"
}

HTTP/1.1 200 OK
[
  {
    "jid": "room1@muc.example.com",
    "public": "true",
    "participants": 10
  },
  {
    "jid": "room2@muc.example.com",
    "public": "false",
    "participants": 10
  }
]
```

muc_register_nick

Register a nick to a User JID in a MUC service

Arguments:

- *nick* :: string : Nick
- *jid* :: string : User JID
- *service* :: string : Service

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/muc_register_nick
{
  "nick": "Tim",
  "jid": "tim@example.org",
  "service": "muc.example.org"
}

HTTP/1.1 200 OK
""
```

muc_unregister_nick

Unregister the nick registered by that account in the MUC service

Arguments:

- *jid* :: string : User JID
- *service* :: string : MUC service

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/muc_unregister_nick
{
  "jid": "tim@example.org",
  "service": "muc.example.org"
}

HTTP/1.1 200 OK
""
```

num_resources

Get the number of resources of a user

Arguments:

- *user* :: string : User name
- *host* :: string : Server name

Result:

- *resources* :: integer : Number of active resources for a user

Tags: session

Module: mod_admin_extra

Examples:

```
POST /api/num_resources
{
  "user": "peter",
  "host": "myserver.com"
}

HTTP/1.1 200 OK
5
```

oauth_add_client_implicit

Add OAuth client_id with implicit grant type

Arguments:

- *client_id* :: string
- *client_name* :: string
- *redirect_uri* :: string

Result:

- *res* :: string : Raw result string

Tags: oauth

Examples:

```
POST /api/oauth_add_client_implicit
{
  "client_id": "aaaaa",
  "client_name": "bbbbb",
  "redirect_uri": "ccccc"
}

HTTP/1.1 200 OK
"Success"
```

oauth_add_client_password

Add OAuth client_id with password grant type

Arguments:

- *client_id* :: string
- *client_name* :: string
- *secret* :: string

Result:

- *res* :: string : Raw result string

Tags: oauth

Examples:

```
POST /api/oauth_add_client_password
{
  "client_id": "aaaaa",
  "client_name": "bbbbb",
  "secret": "ccccc"
}

HTTP/1.1 200 OK
"Success"
```

oauth_issue_token ●

updated in 24.02

Issue an OAuth optionredir token for the given jid

Arguments:

- *jid* :: string : Jid for which issue token
- *ttl* :: integer : Time to live of generated token in seconds
- *scopes* :: [scope::string] : List of scopes to allow

Result:

- *result* :: {token::string, scopes::[scope::string], expires_in::string}

Tags: oauth, v1

Examples:

```
POST /api/oauth_issue_token
{
  "jid": "user@server.com",
  "ttl": 3600,
  "scopes": [
    "connected_users_number",
    "muc_online_rooms"
  ]
}

HTTP/1.1 200 OK
{
  "token": "aaaaa",
  "scopes": [
    "bbbbb",
    "ccccc"
  ],
  "expires_in": "ddddd"
}
```

oauth_list_tokens

List OAuth tokens, user, scope, and seconds to expire (only Mnesia)

List OAuth tokens, their user and scope, and how many seconds remain until expirity

Arguments:

Result:

- *tokens* :: [{token::string, user::string, scope::string, expires_in::string}]

Tags: oauth

Examples:

```
POST /api/oauth_list_tokens
{

}

HTTP/1.1 200 OK
[
  {
    "token": "aaaaa",
    "user": "bbbbb",
    "scope": "ccccc",
    "expires_in": "ddddd"
  },
  {
    "token": "eeeee",
    "user": "fffff",
    "scope": "ggggg",
    "expires_in": "hhhhh"
  }
]
```

oauth_remove_client

Remove OAuth client_id

Arguments:

- *client_id* :: string

Result:

- *res* :: string : Raw result string

Tags: oauth

Examples:

```
POST /api/oauth_remove_client
{
```

```
    "client_id": "aaaaa"
}

HTTP/1.1 200 OK
"Success"
```

oauth_revoke_token

🏷 changed in 22.05

Revoke authorization for an OAuth token

Arguments:

- *token* :: string

Result:

- *res* :: string : Raw result string

Tags: oauth

Examples:

```
POST /api/oauth_revoke_token
{
  "token": "aaaaa"
}

HTTP/1.1 200 OK
"Success"
```

outgoing_s2s_number

Number of outgoing s2s connections on the node

Arguments:

Result:

- *s2s_outgoing* :: integer

Tags: statistics, s2s

Examples:

```
POST /api/outgoing_s2s_number
{

}

HTTP/1.1 200 OK
1
```

print_sql_schema 🔴

🏷 added in 24.02

Print SQL schema for the given RDBMS (only ejabberdctl)

This command is exclusive for the ejabberdctl command-line script, don't attempt to execute it using any other API frontend.

Arguments:

- *db_type* :: string : Database type: pgsql | mysql | sqlite
- *db_version* :: string : Your database version: 16.1, 8.2.0...
- *new_schema* :: string : Use new schema: 0, false, 1 or true

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: ejabberdctl, sql

Examples:

```
POST /api/print_sql_schema
{
  "db_type": "pgsql",
  "db_version": "16.1",
  "new_schema": "true"
}

HTTP/1.1 200 OK
""
```

privacy_set

Send a IQ set privacy stanza for a local account

Arguments:

- *user* :: string : Username
- *host* :: string : Server name
- *xmlquery* :: string : Query XML element

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: stanza

Module: mod_admin_extra

Examples:

```
POST /api/privacy_set
{
  "user": "user1",
  "host": "myserver.com",
  "xmlquery": "<query xmlns='jabber:iq:privacy'>..."
}

HTTP/1.1 200 OK
""
```

private_get

Get some information from a user private storage

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *element* :: string : Element name
- *ns* :: string : Namespace

Result:

- *res* :: string

Tags: private

Module: mod_admin_extra

Examples:

```
POST /api/private_get
{
  "user": "user1",
  "host": "myserver.com",
  "element": "storage",
  "ns": "storage:rosternotes"
}

HTTP/1.1 200 OK
"aaaaa"
```

private_set

Set to the user private storage

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *element* :: string : XML storage element

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: private

Module: mod_admin_extra

Examples:

```
POST /api/private_set
{
  "user": "user1",
  "host": "myserver.com",
  "element": "<storage xmlns='storage:rosternotes'/>"
}

HTTP/1.1 200 OK
""
```

process_rosteritems

List/delete rosteritems that match filter

Explanation of each argument:

- `action` : what to do with each rosteritem that matches all the filtering options
- `subs` : subscription type
- `asks` : pending subscription
- `users` : the JIDs of the local user
- `contacts` : the JIDs of the contact in the roster

Mnesia backend:

Allowed values in the arguments:

- `action` = `list` | `delete`
- `subs` = `any` | SUB[:SUB]*
- `asks` = `any` | ASK[:ASK]*
- `users` = `any` | JID[:JID]*
- `contacts` = `any` | JID[:JID]*

where

- SUB = `none` | `from` | `to` | `both`
- ASK = `none` | `out` | `in`
- JID = characters valid in a JID, and can use the globs: `*`, `?`, `!` and `[...]`

This example will list roster items with subscription `none`, `from` or `to` that have any ask property, of local users which JID is in the virtual host `example.org` and that the contact JID is either a bare server name (without user part) or that has a user part and the server part contains the word `icq`: `list none:from:to any *@example.org *:*@*icq*`

SQL backend:

Allowed values in the arguments:

- `action` = `list` | `delete`
- `subs` = `any` | SUB
- `asks` = `any` | ASK
- `users` = JID
- `contacts` = JID

where

- SUB = `none` | `from` | `to` | `both`
- ASK = `none` | `out` | `in`
- JID = characters valid in a JID, and can use the globs: `_` and `%`

This example will list roster items with subscription `to` that have any ask property, of local users which JID is in the virtual host `example.org` and that the contact JID's server part contains the word `icq`: `list to any %@example.org %@%icq%`

Arguments:

- *action* :: string
- *subs* :: string
- *asks* :: string
- *users* :: string
- *contacts* :: string

Result:

- *response* :: [{user::string, contact::string}]

Tags: roster

Module: mod_admin_extra

Examples:

```
POST /api/process_rosteritems
{
  "action": "aaaaa",
  "subs": "bbbbb",
```

```
    "asks": "ccccc",
    "users": "ddddd",
    "contacts": "eeeee"
}

HTTP/1.1 200 OK
[
    {
      "user": "aaaaa",
      "contact": "bbbbb"
    },
    {
      "user": "ccccc",
      "contact": "ddddd"
    }
]
```

push_alltoall

Add all the users to all the users of Host in Group

Arguments:

- *host* :: string : Server name
- *group* :: string : Group name

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: roster

Module: mod_admin_extra

Examples:

```
POST /api/push_alltoall
{
  "host": "myserver.com",
  "group": "Everybody"
}

HTTP/1.1 200 OK
""
```

push_roster

Push template roster from file to a user

The text file must contain an erlang term: a list of tuples with username, servername, group and nick. For example: `[{"user1",` `"localhost", "Workers", "User 1"},` `{"user2", "localhost", "Workers", "User 2"}].`

If there are problems parsing UTF8 character encoding, provide the corresponding string with the `<<"STRING"/utf8>>` syntax, for example: `[{"user2", "localhost", "Workers", <<"User 2"/utf8>>}].`

Arguments:

- *file* :: string : File path
- *user* :: string : User name
- *host* :: string : Server name

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: roster

Module: mod_admin_extra

Examples:

```
POST /api/push_roster
{
  "file": "/home/ejabberd/roster.txt",
  "user": "user1",
  "host": "localhost"
}

HTTP/1.1 200 OK
""
```

push_roster_all

Push template roster from file to all those users

The text file must contain an erlang term: a list of tuples with username, servername, group and nick. Example: `[{"user1",`
`"localhost", "Workers", "User 1"},`
`{"user2", "localhost", "Workers", "User 2"}].`

Arguments:

- *file* :: string : File path

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: roster

Module: mod_admin_extra

Examples:

```
POST /api/push_roster_all
{
  "file": "/home/ejabberd/roster.txt"
}

HTTP/1.1 200 OK
""
```

register

Register a user

Arguments:

- *user* :: string : Username
- *host* :: string : Local vhost served by ejabberd
- *password* :: string : Password

Result:

- *res* :: string : Raw result string

Tags: accounts

Examples:

```
POST /api/register
{
  "user": "bob",
  "host": "example.com",
  "password": "SomEPass44"
}
```

```
HTTP/1.1 200 OK
"Success"
```

registered_users

List all registered users in HOST

Arguments:

- *host* :: string : Local vhost

Result:

- *users* :: [username::string] : List of registered accounts usernames

Tags: accounts

Examples:

```
POST /api/registered_users
{
  "host": "example.com"
}

HTTP/1.1 200 OK
[
  "user1",
  "user2"
]
```

registered_vhosts

List all registered vhosts in SERVER

Arguments:

Result:

- *vhosts* :: [vhost::string] : List of available vhosts

Tags: server

Examples:

```
POST /api/registered_vhosts
{

}

HTTP/1.1 200 OK
[
  "example.com",
  "anon.example.com"
]
```

reload_config

Reload config file in memory

Arguments:

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: config

Examples:

```
POST /api/reload_config
{

}

HTTP/1.1 200 OK
""
```

remove_mam_for_user

Remove mam archive for user

Arguments:

- *user* :: string : Username

- *host* :: string : Server

Result:

- *res* :: string : Raw result string

Tags: mam

Module: mod_mam

Examples:

```
POST /api/remove_mam_for_user
{
  "user": "bob",
  "host": "example.com"
}

HTTP/1.1 200 OK
"MAM archive removed"
```

remove_mam_for_user_with_peer

Remove mam archive for user with peer

Arguments:

- *user* :: string : Username

- *host* :: string : Server

- *with* :: string : Peer

Result:

- *res* :: string : Raw result string

Tags: mam

Module: mod_mam

Examples:

```
POST /api/remove_mam_for_user_with_peer
{
  "user": "bob",
  "host": "example.com",
  "with": "anne@example.com"
}

HTTP/1.1 200 OK
"MAM archive removed"
```

reopen_log

Reopen maybe the log files after being renamed

Has no effect on ejabberd main log files, only on log files generated by some modules. This can be useful when an external tool is used for log rotation. See Log Files.

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: logs

Examples:

```
POST /api/reopen_log
{

}

HTTP/1.1 200 OK
""
```

request_certificate

Requests certificates for all or some domains

Domains can be `all`, or a list of domains separared with comma characters

Arguments:

- *domains* :: string : Domains for which to acquire a certificate

Result:

- *res* :: string : Raw result string

Tags: acme

Examples:

```
POST /api/request_certificate
{
  "domains": "example.com,domain.tld,conference.domain.tld"
}

HTTP/1.1 200 OK
"Success"
```

resource_num

Resource string of a session number

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *num* :: integer : ID of resource to return

Result:

- *resource* :: string : Name of user resource

Tags: session

Module: mod_admin_extra

Examples:

```
POST /api/resource_num
{
  "user": "peter",
  "host": "myserver.com",
  "num": 2
}

HTTP/1.1 200 OK
"Psi"
```

restart

Restart ejabberd gracefully

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: server

Examples:

```
POST /api/restart
{

}

HTTP/1.1 200 OK
""
```

restart_module

Stop an ejabberd module, reload code and start

Arguments:

- *host* :: string : Server name
- *module* :: string : Module to restart

Result:

- *res* :: integer : Returns integer code:
- `0` : code reloaded, module restarted
- `1` : error: module not loaded
- `2` : code not reloaded, but module restarted

Tags: erlang

Module: mod_admin_extra

Examples:

```
POST /api/restart_module
{
  "host": "myserver.com",
  "module": "mod_admin_extra"
}

HTTP/1.1 200 OK
0
```

restore

Restore the Mnesia database from a binary backup file

This restores immediately from a binary backup file the internal Mnesia database. This will consume a lot of memory if you have a large database, you may prefer install_fallback API.

Arguments:

- *file* :: string : Full path to the backup file

Result:

- *res* :: string : Raw result string

Tags: mnesia

Examples:

```
POST /api/restore
{
  "file": "/var/lib/ejabberd/database.backup"
}

HTTP/1.1 200 OK
"Success"
```

revoke_certificate

Revokes the selected ACME certificate

Arguments:

- *file* :: string : Filename of the certificate

Result:

- *res* :: string : Raw result string

Tags: acme

Examples:

```
POST /api/revoke_certificate
{
  "file": "aaaaa"
}

HTTP/1.1 200 OK
"Success"
```

rooms_empty_destroy

Destroy the rooms that have no messages in archive

The MUC service argument can be `global` to get all hosts.

Arguments:

- *service* :: string : MUC service, or `global` for all

Result:

- *rooms* :: [room::string] : List of empty rooms that have been destroyed

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/rooms_empty_destroy
{
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
[
  "room1@muc.example.com",
  "room2@muc.example.com"
]
```

rooms_empty_list

List the rooms that have no messages in archive

The MUC service argument can be `global` to get all hosts.

Arguments:

- *service* :: string : MUC service, or `global` for all

Result:

- *rooms* :: [room::string] : List of empty rooms

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/rooms_empty_list
{
  "service": "muc.example.com"
}

HTTP/1.1 200 OK
[
  "room1@muc.example.com",
  "room2@muc.example.com"
]
```

rooms_unused_destroy

Destroy the rooms that are unused for many days in the service

The room recent history is used, so it's recommended to wait a few days after service start before running this. The MUC service argument can be `global` to get all hosts.

Arguments:

- *service* :: string : MUC service, or `global` for all
- *days* :: integer : Number of days

Result:

- *rooms* :: [room::string] : List of unused rooms that has been destroyed

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/rooms_unused_destroy
{
  "service": "muc.example.com",
  "days": 31
}

HTTP/1.1 200 OK
[
  "room1@muc.example.com",
  "room2@muc.example.com"
]
```

rooms_unused_list

List the rooms that are unused for many days in the service

The room recent history is used, so it's recommended to wait a few days after service start before running this. The MUC service argument can be `global` to get all hosts.

Arguments:

- *service* :: string : MUC service, or `global` for all
- *days* :: integer : Number of days

Result:

- *rooms* :: [room::string] : List of unused rooms

Tags: muc

Module: mod_muc_admin

Examples:

```
POST /api/rooms_unused_list
{
  "service": "muc.example.com",
  "days": 31
}

HTTP/1.1 200 OK
[
  "room1@muc.example.com",
  "room2@muc.example.com"
]
```

rotate_log

Rotate maybe log file of some module

Has no effect on ejabberd main log files, only on log files generated by some modules.

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: logs

Examples:

```
POST /api/rotate_log
{

}

HTTP/1.1 200 OK
""
```

send_direct_invitation ●

🏷 updated in 24.02

Send a direct invitation to several destinations

Since ejabberd 20.12, this command is asynchronous: the API call may return before the server has send all the invitations.

`password` and `message` can be set to `none`.

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service
- *password* :: string : Password, or `none`
- *reason* :: string : Reason text, or `none`
- *users* :: [jid::string] : List of users JIDs

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc_room, v1

Module: mod_muc_admin

Examples:

```
POST /api/send_direct_invitation
{
  "name": "room1",
  "service": "muc.example.com",
  "password": "",
  "reason": "Check this out!",
  "users": [
    "user2@localhost",
    "user3@example.com"
  ]
}

HTTP/1.1 200 OK
""
```

send_message

Send a message to a local or remote bare of full JID

When sending a groupchat message to a MUC room, `from` must be the full JID of a room occupant, or the bare JID of a MUC service admin, or the bare JID of a MUC/Sub subscribed user.

Arguments:

- *type* :: string : Message type: `normal`, `chat`, `headline`, `groupchat`
- *from* :: string : Sender JID
- *to* :: string : Receiver JID
- *subject* :: string : Subject, or empty string
- *body* :: string : Body

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: stanza

Module: mod_admin_extra

Examples:

```
POST /api/send_message
{
  "type": "headline",
  "from": "admin@localhost",
  "to": "user1@localhost",
  "subject": "Restart",
  "body": "In 5 minutes"
}

HTTP/1.1 200 OK
""
```

send_stanza

Send a stanza; provide From JID and valid To JID

Arguments:

- *from* :: string : Sender JID
- *to* :: string : Destination JID
- *stanza* :: string : Stanza

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: stanza

Module: mod_admin_extra

Examples:

```
POST /api/send_stanza
{
  "from": "admin@localhost",
  "to": "user1@localhost",
  "stanza": "<message><ext attr='value'/></message>"
}

HTTP/1.1 200 OK
""
```

send_stanza_c2s

Send a stanza from an existing C2S session

`user` @ `host` / `resource` must be an existing C2S session. As an alternative, use send_stanza API instead.

Arguments:

- *user* :: string : Username
- *host* :: string : Server name
- *resource* :: string : Resource
- *stanza* :: string : Stanza

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: stanza

Module: mod_admin_extra

Examples:

```
POST /api/send_stanza_c2s
{
  "user": "admin",
  "host": "myserver.com",
  "resource": "bot",
  "stanza": "<message to='user1@localhost'><ext attr='value'/></message>"
}

HTTP/1.1 200 OK
""
```

set_last

Set last activity information

Timestamp is the seconds since `1970-01-01 00:00:00 UTC`. For example value see `date +%s`

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *timestamp* :: integer : Number of seconds since epoch
- *status* :: string : Status message

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: last

Module: mod_admin_extra

Examples:

```
POST /api/set_last
{
  "user": "user1",
  "host": "myserver.com",
  "timestamp": 1500045311,
  "status": "GoSleeping"
}

HTTP/1.1 200 OK
""
```

set_loglevel

Set the loglevel

Possible loglevels: `none`, `emergency`, `alert`, `critical`, `error`, `warning`, `notice`, `info`, `debug`.

Arguments:

- *loglevel* :: string : Desired logging level

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: logs

Examples:

```
POST /api/set_loglevel
{
  "loglevel": "debug"
```

```
    }

    HTTP/1.1 200 OK
    ""
```

set_master

Set master node of the clustered Mnesia tables

If `nodename` is set to `self`, then this node will be set as its own master.

Arguments:

- *nodename* :: string : Name of the erlang node that will be considered master of this node

Result:

- *res* :: string : Raw result string

Tags: cluster

Examples:

```
POST /api/set_master
{
  "nodename": "ejabberd@machine7"
}

HTTP/1.1 200 OK
"Success"
```

set_nickname

Set nickname in a user's vCard

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *nickname* :: string : Nickname

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: vcard

Module: mod_admin_extra

Examples:

```
POST /api/set_nickname
{
  "user": "user1",
  "host": "myserver.com",
  "nickname": "User 1"
}

HTTP/1.1 200 OK
""
```

set_presence 🔴

🏷 updated in 24.02

Set presence of a session

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *resource* :: string : Resource
- *type* :: string : Type: `available`, `error`, `probe` ...
- *show* :: string : Show: `away`, `chat`, `dnd`, `xa`.
- *status* :: string : Status text
- *priority* :: integer : Priority, provide this value as an integer

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: session, v1

Module: mod_admin_extra

Examples:

```
POST /api/set_presence
{
  "user": "user1",
  "host": "myserver.com",
  "resource": "tka1",
  "type": "available",
  "show": "away",
  "status": "BB",
  "priority": 7
}

HTTP/1.1 200 OK
""
```

set_room_affiliation

Change an affiliation in a MUC room

Arguments:

- *name* :: string : Room name
- *service* :: string : MUC service
- *jid* :: string : User JID
- *affiliation* :: string : Affiliation to set

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc_room

Module: mod_muc_admin

Examples:

```
POST /api/set_room_affiliation
{
  "name": "room1",
  "service": "muc.example.com",
  "jid": "user2@example.com",
  "affiliation": "member"
}

HTTP/1.1 200 OK
""
```

set_vcard

Set content in a vCard field

Some vcard field names in `get`/`set_vcard` are:

- FN - Full Name
- NICKNAME - Nickname
- BDAY - Birthday
- TITLE - Work: Position
- ROLE - Work: Role

For a full list of vCard fields check XEP-0054: vcard-temp

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *name* :: string : Field name
- *content* :: string : Value

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: vcard

Module: mod_admin_extra

Examples:

```
POST /api/set_vcard
{
  "user": "user1",
  "host": "myserver.com",
  "name": "URL",
  "content": "www.example.com"
}

HTTP/1.1 200 OK
""
```

set_vcard2

Set content in a vCard subfield

Some vcard field names and subnames in `get` / `set_vcard2` are:

- N FAMILY - Family name
- N GIVEN - Given name
- N MIDDLE - Middle name
- ADR CTRY - Address: Country
- ADR LOCALITY - Address: City
- TEL HOME - Telephone: Home
- TEL CELL - Telephone: Cellphone
- TEL WORK - Telephone: Work
- TEL VOICE - Telephone: Voice
- EMAIL USERID - E-Mail Address
- ORG ORGNAME - Work: Company
- ORG ORGUNIT - Work: Department

For a full list of vCard fields check XEP-0054: vcard-temp

Arguments:

- *user* :: string : User name
- *host* :: string : Server name
- *name* :: string : Field name
- *subname* :: string : Subfield name
- *content* :: string : Value

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: vcard

Module: mod_admin_extra

Examples:

```
POST /api/set_vcard2
{
  "user": "user1",
  "host": "myserver.com",
  "name": "TEL",
  "subname": "NUMBER",
  "content": "123456"
}

HTTP/1.1 200 OK
""
```

set_vcard2_multi

Set multiple contents in a vCard subfield

Some vcard field names and subnames in `get`/`set_vcard2` are:

- N FAMILY - Family name
- N GIVEN - Given name
- N MIDDLE - Middle name
- ADR CTRY - Address: Country
- ADR LOCALITY - Address: City
- TEL HOME - Telephone: Home
- TEL CELL - Telephone: Cellphone
- TEL WORK - Telephone: Work
- TEL VOICE - Telephone: Voice
- EMAIL USERID - E-Mail Address
- ORG ORGNAME - Work: Company
- ORG ORGUNIT - Work: Department

For a full list of vCard fields check XEP-0054: vcard-temp

Arguments:

- *user* :: string
- *host* :: string
- *name* :: string
- *subname* :: string
- *contents* :: [value::string]

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: vcard

Module: mod_admin_extra

Examples:

```
POST /api/set_vcard2_multi
{
  "user": "aaaaa",
  "host": "bbbbb",
  "name": "ccccc",
  "subname": "ddddd",
  "contents": [
    "eeeee",
    "fffff"
  ]
}

HTTP/1.1 200 OK
""
```

srg_create ●

🏷 updated in 24.02

Create a Shared Roster Group

Arguments:

- *group* :: string : Group identifier
- *host* :: string : Group server name
- *label* :: string : Group name
- *description* :: string : Group description
- *display* :: [group::string] : List of groups to display

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: shared_roster_group, v1

Module: mod_admin_extra

Examples:

```
POST /api/srg_create
{
  "group": "group3",
  "host": "myserver.com",
  "label": "Group3",
  "description": "Third group",
  "display": [
    "group1",
    "group2"
  ]
}

HTTP/1.1 200 OK
""
```

srg_delete

Delete a Shared Roster Group

Arguments:

- *group* :: string : Group identifier
- *host* :: string : Group server name

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: shared_roster_group

Module: mod_admin_extra

Examples:

```
POST /api/srg_delete
{
  "group": "group3",
  "host": "myserver.com"
}

HTTP/1.1 200 OK
""
```

srg_get_info

Get info of a Shared Roster Group

Arguments:

- *group* :: string : Group identifier
- *host* :: string : Group server name

Result:

- *informations* :: [{key::string, value::string}] : List of group information, as key and value

Tags: shared_roster_group

Module: mod_admin_extra

Examples:

```
POST /api/srg_get_info
{
  "group": "group3",
  "host": "myserver.com"
}

HTTP/1.1 200 OK
[
  {
    "key": "name",
    "value": "Group 3"
  },
  {
    "key": "displayed_groups",
    "value": "group1"
  }
]
```

srg_get_members

Get members of a Shared Roster Group

Arguments:

- *group* :: string : Group identifier
- *host* :: string : Group server name

Result:

- *members* :: [member::string] : List of group identifiers

Tags: shared_roster_group

Module: mod_admin_extra

Examples:

```
POST /api/srg_get_members
{
  "group": "group3",
  "host": "myserver.com"
}

HTTP/1.1 200 OK
[
  "user1@localhost",
  "user2@localhost"
]
```

srg_list

List the Shared Roster Groups in Host

Arguments:

- *host* :: string : Server name

Result:

- *groups* :: [id::string] : List of group identifiers

Tags: shared_roster_group

Module: mod_admin_extra

Examples:

```
POST /api/srg_list
{
  "host": "myserver.com"
}

HTTP/1.1 200 OK
[
  "group1",
  "group2"
]
```

srg_user_add

Add the JID user@host to the Shared Roster Group

Arguments:

- *user* :: string : Username
- *host* :: string : User server name
- *group* :: string : Group identifier
- *grouphost* :: string : Group server name

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: shared_roster_group

Module: mod_admin_extra

Examples:

```
POST /api/srg_user_add
{
  "user": "user1",
  "host": "myserver.com",
  "group": "group3",
  "grouphost": "myserver.com"
}

HTTP/1.1 200 OK
""
```

srg_user_del

Delete this JID user@host from the Shared Roster Group

Arguments:

- *user* :: string : Username
- *host* :: string : User server name
- *group* :: string : Group identifier
- *grouphost* :: string : Group server name

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: shared_roster_group

Module: mod_admin_extra

Examples:

```
POST /api/srg_user_del
{
  "user": "user1",
  "host": "myserver.com",
  "group": "group3",
  "grouphost": "myserver.com"
}

HTTP/1.1 200 OK
""
```

stats

Get some statistical value for the whole ejabberd server

Allowed statistics `name` are: `registeredusers`, `onlineusers`, `onlineusersnode`, `uptimeseconds`, `processes`.

Arguments:

- *name* :: string : Statistic name

Result:

- *stat* :: integer : Integer statistic value

Tags: statistics

Module: mod_admin_extra

Examples:

```
POST /api/stats
{
  "name": "registeredusers"
}

HTTP/1.1 200 OK
6
```

stats_host

Get some statistical value for this host

Allowed statistics `name` are: `registeredusers`, `onlineusers`.

Arguments:

- *name* :: string : Statistic name
- *host* :: string : Server JID

Result:

- *stat* :: integer : Integer statistic value

Tags: statistics

Module: mod_admin_extra

Examples:

```
POST /api/stats_host
{
  "name": "registeredusers",
  "host": "example.com"
}

HTTP/1.1 200 OK
6
```

status

Get status of the ejabberd server

Arguments:

Result:

- *res* :: string : Raw result string

Tags: server

Examples:

```
POST /api/status
{

}

HTTP/1.1 200 OK
"The node ejabberd@localhost is started with status: startedejabberd X.X is running in that node"
```

status_list

List of logged users with this status

Arguments:

- *status* :: string : Status type to check

Result:

- *users* :: [{user::string, host::string, resource::string, priority::integer, status::string}]

Tags: session

Module: mod_admin_extra

Examples:

```
POST /api/status_list
{
  "status": "dnd"
}

HTTP/1.1 200 OK
[
  {
    "user": "peter",
    "host": "myserver.com",
    "resource": "tka",
    "priority": 6,
    "status": "Busy"
```

```
      }
    ]
```

status_list_host

List of users logged in host with their statuses

Arguments:

- *host* :: string : Server name
- *status* :: string : Status type to check

Result:

- *users* :: [{user::string, host::string, resource::string, priority::integer, status::string}]

Tags: session

Module: mod_admin_extra

Examples:

```
POST /api/status_list_host
{
  "host": "myserver.com",
  "status": "dnd"
}

HTTP/1.1 200 OK
[
  {
    "user": "peter",
    "host": "myserver.com",
    "resource": "tka",
    "priority": 6,
    "status": "Busy"
  }
]
```

status_num

Number of logged users with this status

Arguments:

- *status* :: string : Status type to check

Result:

- *users* :: integer : Number of connected sessions with given status type

Tags: session, statistics

Module: mod_admin_extra

Examples:

```
POST /api/status_num
{
  "status": "dnd"
}

HTTP/1.1 200 OK
23
```

status_num_host

Number of logged users with this status in host

Arguments:

- *host* :: string : Server name
- *status* :: string : Status type to check

Result:

- *users* :: integer : Number of connected sessions with given status type

Tags: session, statistics

Module: mod_admin_extra

Examples:

```
POST /api/status_num_host
{
  "host": "myserver.com",
  "status": "dnd"
}

HTTP/1.1 200 OK
23
```

stop

Stop ejabberd gracefully

Arguments:

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: server

Examples:

```
POST /api/stop
{

}

HTTP/1.1 200 OK
""
```

stop_kindly

Inform users and rooms, wait, and stop the server

Provide the delay in seconds, and the announcement quoted, for example: `ejabberdctl stop_kindly 60 \"The server will stop in one minute.\"`

Arguments:

- *delay* :: integer : Seconds to wait
- *announcement* :: string : Announcement to send, with quotes

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: server

Examples:

```
POST /api/stop_kindly
{
  "delay": 60,
  "announcement": "Server will stop now."
}

HTTP/1.1 200 OK
""
```

stop_s2s_connections

Stop all s2s outgoing and incoming connections

Arguments:

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: s2s

Examples:

```
POST /api/stop_s2s_connections
{

}

HTTP/1.1 200 OK
""
```

subscribe_room 🔴

🏷 updated in 24.02

Subscribe to a MUC conference

Arguments:

- *user* :: string : User JID
- *nick* :: string : a user's nick
- *room* :: string : the room to subscribe
- *nodes* :: [node::string] : list of nodes

Result:

- *nodes* :: [node::string] : The list of nodes that has subscribed

Tags: muc_room, muc_sub, v1

Module: mod_muc_admin

Examples:

```
POST /api/subscribe_room
{
  "user": "tom@localhost",
  "nick": "Tom",
  "room": "room1@conference.localhost",
  "nodes": [
    "urn:xmpp:mucsub:nodes:messages",
    "urn:xmpp:mucsub:nodes:affiliations"
  ]
}

HTTP/1.1 200 OK
[
  "urn:xmpp:mucsub:nodes:messages",
  "urn:xmpp:mucsub:nodes:affiliations"
]
```

subscribe_room_many ●

🏷 updated in 24.02

Subscribe several users to a MUC conference

This command accepts up to 50 users at once (this is configurable with the mod_muc_admin option `subscribe_room_many_max_users`)

Arguments:

- *users* :: [{jid::string, nick::string}] : Users JIDs and nicks
- *room* :: string : the room to subscribe
- *nodes* :: [node::string] : nodes separated by commas: `,`

Result:

- *res* :: integer : Status code (`0` on success, `1` otherwise)

Tags: muc_room, muc_sub, v1

Module: mod_muc_admin

Examples:

```
POST /api/subscribe_room_many
{
  "users": [
    {
      "jid": "tom@localhost",
      "nick": "Tom"
    },
    {
      "jid": "jerry@localhost",
      "nick": "Jerry"
    }
  ],
  "room": "room1@conference.localhost",
  "nodes": [
    "urn:xmpp:mucsub:nodes:messages",
    "urn:xmpp:mucsub:nodes:affiliations"
  ]
}

HTTP/1.1 200 OK
""
```

unban_ip

Remove banned IP addresses from the fail2ban table

Accepts an IP address with a network mask. Returns the number of unbanned addresses, or a negative integer if there were any error.

Arguments:

- *address* :: string : IP address, optionally with network mask.

Result:

- *unbanned* :: integer : Amount of unbanned entries, or negative in case of error.

Tags: accounts

Module: mod_fail2ban

Examples:

```
POST /api/unban_ip
{
  "address": "::FFFF:127.0.0.1/128"
}

HTTP/1.1 200 OK
3
```

unregister

Unregister a user

This deletes the authentication and all the data associated to the account (roster, vcard...).

Arguments:

- *user* :: string : Username

- *host* :: string : Local vhost served by ejabberd

Result:

- *res* :: string : Raw result string

Tags: accounts

Examples:

```
POST /api/unregister
{
  "user": "bob",
  "host": "example.com"
}

HTTP/1.1 200 OK
"Success"
```

unsubscribe_room

Unsubscribe from a MUC conference

Arguments:

- *user* :: string : User JID

- *room* :: string : the room to subscribe

Result:

- *res* :: integer : Status code (0 on success, 1 otherwise)

Tags: muc_room, muc_sub

Module: mod_muc_admin

Examples:

```
POST /api/unsubscribe_room
{
  "user": "tom@localhost",
  "room": "room1@conference.localhost"
}

HTTP/1.1 200 OK
""
```

update

Update the given module

To update all the possible modules, use `all` .

Arguments:

• *module* :: string

Result:

• *res* :: string : Raw result string

Tags: server

Examples:

```
POST /api/update
{
  "module": "mod_vcard"
}

HTTP/1.1 200 OK
"Success"
```

update_list

List modified modules that can be updated

Arguments:

Result:

• *modules* :: [module::string]

Tags: server

Examples:

```
POST /api/update_list
{

}

HTTP/1.1 200 OK
[
  "mod_configure",
  "mod_vcard"
]
```

user_resources

List user's connected resources

Arguments:

• *user* :: string : User name

• *host* :: string : Server name

Result:

• *resources* :: [resource::string]

Tags: session

Examples:

```
POST /api/user_resources
{
  "user": "user1",
  "host": "example.com"
}
```

```
HTTP/1.1 200 OK
[
  "tka1",
  "Gajim",
  "mobile-app"
]
```

user_sessions_info

Get information about all sessions of a user

Arguments:

- *user* :: string : User name
- *host* :: string : Server name

Result:

- *sessions_info* :: [{connection::string, ip::string, port::integer, priority::integer, node::string, uptime::integer, status::string, resource::string, statustext::string}]

Tags: session

Module: mod_admin_extra

Examples:

```
POST /api/user_sessions_info
{
  "user": "peter",
  "host": "myserver.com"
}

HTTP/1.1 200 OK
[
  {
    "connection": "c2s",
    "ip": "127.0.0.1",
    "port": 42656,
    "priority": 8,
    "node": "ejabberd@localhost",
    "uptime": 231,
    "status": "dnd",
    "resource": "tka",
    "statustext": ""
  }
]
```

API Tags

This section enumerates the API tags of ejabberd 24.02. If you are using an old ejabberd release, please refer to the corresponding archived version of this page in the Archive.

accounts

- ban_account
- change_password
- check_account
- check_password
- check_password_hash
- delete_old_users
- delete_old_users_vhost
- register
- registered_users
- unban_ip
- unregister

acme

- list_certificates
- request_certificate
- revoke_certificate

cluster

- join_cluster
- leave_cluster
- list_cluster
- set_master

config

- convert_to_yaml
- dump_config
- reload_config

documentation

- gen_html_doc_for_commands
- gen_markdown_doc_for_commands
- gen_markdown_doc_for_tags
- man

ejabberdctl

- help
- mnesia_info_ctl
- print_sql_schema

erlang

- compile
- get_cookie
- restart_module

last

- get_last
- set_last

logs

- get_loglevel
- reopen_log
- rotate_log
- set_loglevel

mam

- remove_mam_for_user
- remove_mam_for_user_with_peer

mnesia

- backup
- delete_mnesia
- dump
- dump_table
- export2sql
- export_piefxis
- export_piefxis_host
- import_dir
- import_file
- import_piefxis
- import_prosody
- install_fallback
- load

- mnesia_change_nodename
- mnesia_info
- mnesia_info_ctl
- mnesia_table_info
- restore

modules

- module_check
- module_install
- module_uninstall
- module_upgrade
- modules_available
- modules_installed
- modules_update_specs

muc

- create_rooms_file
- destroy_rooms_file
- get_user_rooms
- get_user_subscriptions
- muc_online_rooms
- muc_online_rooms_by_regex
- muc_register_nick
- muc_unregister_nick
- rooms_empty_destroy
- rooms_empty_list
- rooms_unused_destroy
- rooms_unused_list

muc_room

- change_room_option
- create_room
- create_room_with_opts
- destroy_room
- get_room_affiliation
- get_room_affiliations
- get_room_history
- get_room_occupants
- get_room_occupants_number
- get_room_options

- get_subscribers
- send_direct_invitation
- set_room_affiliation
- subscribe_room
- subscribe_room_many
- unsubscribe_room

muc_sub

- create_room_with_opts
- get_subscribers
- get_user_subscriptions
- subscribe_room
- subscribe_room_many
- unsubscribe_room

oauth

- oauth_add_client_implicit
- oauth_add_client_password
- oauth_issue_token
- oauth_list_tokens
- oauth_remove_client
- oauth_revoke_token

offline

- get_offline_count

private

- bookmarks_to_pep
- private_get
- private_set

purge

- abort_delete_old_mam_messages
- abort_delete_old_messages
- delete_expired_messages
- delete_expired_pubsub_items
- delete_old_mam_messages
- delete_old_mam_messages_batch
- delete_old_mam_messages_status

- delete_old_messages
- delete_old_messages_batch
- delete_old_messages_status
- delete_old_pubsub_items
- delete_old_push_sessions
- delete_old_users
- delete_old_users_vhost

roster

- add_rosteritem
- delete_rosteritem
- get_roster
- process_rosteritems
- push_alltoall
- push_roster
- push_roster_all

s2s

- incoming_s2s_number
- outgoing_s2s_number
- stop_s2s_connections

server

- clear_cache
- gc
- halt
- registered_vhosts
- restart
- status
- stop
- stop_kindly
- update
- update_list

session

- connected_users
- connected_users_info
- connected_users_number
- connected_users_vhost
- get_presence

- kick_session
- kick_user
- num_resources
- resource_num
- set_presence
- status_list
- status_list_host
- status_num
- status_num_host
- user_resources
- user_sessions_info

shared_roster_group

- srg_create
- srg_delete
- srg_get_info
- srg_get_members
- srg_list
- srg_user_add
- srg_user_del

sql

- convert_to_scram
- import_prosody
- print_sql_schema

stanza

- privacy_set
- send_message
- send_stanza
- send_stanza_c2s

statistics

- connected_users_number
- incoming_s2s_number
- outgoing_s2s_number
- stats
- stats_host
- status_num

- status_num_host

v1

- add_rosteritem
- oauth_issue_token
- send_direct_invitation
- set_presence
- srg_create
- subscribe_room
- subscribe_room_many

vcard

- get_vcard
- get_vcard2
- get_vcard2_multi
- set_nickname
- set_vcard
- set_vcard2
- set_vcard2_multi

Simple ejabberd Rest API Configuration

Restrict to Local network

If you are planning to use ejabberd API for admin purpose, it is often enough to configure it to be available local commands. Access is thus generally limited by IP addresses, either restricted to localhost only, or restricted to one of your platform back-end.

1. Make sure an ejabberd_http listener is using mod_http_api on a given root URL and on a desired port:

```
listen:
  -
    port: 5281
    module: ejabberd_http
    ip: 127.0.0.1
    request_handlers:
      /api: mod_http_api
```

The `ip` option ensures it listens only on the local interface (127.0.0.1) instead of listening on all interface (0.0.0.0).

2. By defining `api_permissions`, you can then allow HTTP request from a specific IP to trigger API commands execution without user credentials:

```
api_permissions:
  "API used from localhost allows all calls":
    who:
      ip: 127.0.0.1/8
    what:
      - "*"
      - "!stop"
      - "!start"
```

Note: stop and start commands are disabled in that example as they are usually restricted to ejabberdctl command-line tool. They are consider too sensitive to be exposed through API.

3. Now you can query the API, for example:

```
curl '127.0.0.1:5281/api/registered_users?host=localhost'

["user2","user8"]
```

Encryption

If you already defined certificates and your connection is not on a local network, you may want to use encryption.

1. Setup encryption like this:

```
listen:
  -
    port: 5281
    module: ejabberd_http
    tls: true
    request_handlers:
      /api: mod_http_api
```

2. Now you can query using HTTPS:

```
curl 'https://127.0.0.1:5281/api/registered_users?host=localhost'

["user2","user8"]
```

3. If you are using a self-signed certificate, you can bypass the corresponding error message:

```
curl --insecure 'https://127.0.0.1:5281/api/registered_users?host=localhost'

["user2","user8"]
```

Basic Authentication

Quite probably you will want to require authentication to execute API queries, either using basic auth or OAuth.

1. Assuming you have the simple listener:

```
listen:
  -
    port: 5281
    module: ejabberd_http
    ip: 127.0.0.1
    request_handlers:
      /api: mod_http_api
```

2. Define an ACL with the account that you will use to authenticate:

```
acl:
  apicommands:
    user: john@localhost
```

3. Allow only that ACL to use the API:

```
api_permissions:
  "some playing":
    from:
      - ejabberd_ctl
      - mod_http_api
    who:
      acl: apicommands
    what: "*"
```

4. If that account does not yet exist, register it:

```
ejabberdctl register john localhost somePass
```

5. Now, when sending an API query, provide the authentication for that account:

```
curl --basic --user john@localhost:somePass \
     '127.0.0.1:5281/api/registered_users?host=localhost'

["user2","user8","john"]
```

6. Example Python code:

```
import requests

url = "http://localhost:5281/api/registered_users"
data = {
    "host": "localhost"
}

res = requests.post(url, json=data, auth=("john@localhost", "somePass"))

print(res.text)
```

OAuth Authentication

Before using OAuth to interact with ejabberd API, you need to configure OAuth support in ejabberd.

Here are example entries to check / change in your ejabberd configuration file:

1. Add a request handler for OAuth:

```
listen:
  -
    # Using a separate port for oauth and API to make it easy to protect it
    # differently than BOSH and WebSocket HTTP interface.
    port: 5281
    # oauth and API only listen on localhost interface for security reason
    # You can set ip to 0.0.0.0 to open it widely, but be careful!
    ip: 127.0.0.1
    module: ejabberd_http
    request_handlers:
      /api: mod_http_api
      /oauth: ejabberd_oauth
```

2. Set the oauth_access top-level option to allow token creation:

```
oauth_access: all
```

3. Define an ACL with the account that you will use to authenticate:

```
acl:
  apicommands:
    user: john@localhost
```

4. You can then configure the OAuth commands you want to expose and who can use them:

```
api_permissions:
  "admin access":
    who:
      oauth:
        scope: "ejabberd:admin"
        scope: "registered_users"
        access:
          allow:
            acl: apicommands
    what: "*"
```

5. If that account does not yet exist, register it:

```
ejabberdctl register john localhost somePass
```

6. Request an authorization token. A quick way is using ejabberdctl:

```
ejabberdctl oauth_issue_token user123@localhost 3600 ejabberd:admin
erHymcBiT2r0QsuOpDjIrsEvnOS4grkj   [<<"ejabberd:admin">>]   3600 seconds
```

7. Now, when sending an API query, provide the authentication for that account:

```
curl -H "Authorization: Bearer erHymcBiT2r0QsuOpDjIrsEvnOS4grkj" \
     '127.0.0.1:5281/api/registered_users?host=localhost'

["user2","user8","john"]
```

Or quite simply:

```
curl --oauth2-bearer erHymcBiT2r0QsuOpDjIrsEvnOS4grkj \
     '127.0.0.1:5281/api/registered_users?host=localhost'

["user2","user8","john"]
```

API Permissions

🏷 added in 16.12

This page describes ejabberd's flexible permission mechanism.

Access to all available endpoints are configured using `api_permissions` option.

It allows to define multiple groups, each one with separate list of filters on who and what are allowed by rules specified inside it.

Basic rule looks like this:

```
api_permissions:
  "admin access":
    who:
      - admin
    what:
      - "*"
      - "!stop"
    from:
      - ejabberd_ctl
      - mod_http_api
```

It tells that group named `Admin access` allows all users that are accepted by ACL rule `admin` to execute all commands except command `stop`, using the command-line tool `ejabberdctl` or sending a ReST query handled by `mod_http_api`.

Each group has associated name (that is just used in log messages), `who` section for rules that authentication details must match, `what` section for specifying list of command, and `from` with list of modules that API was delivered to.

Rules inside `who` section

There are 3 types of rules that can be placed in `who` section:

- **acl:** *Name | ACLDefinition*
  or the short version:
  Name | ACLRule
  This accepts a command when the authentication provided matches rules of `Name` Access Control List (or inline rules from `ACLDefinition` or `ACLRule`)

- **access:** *Name | AccessDefinition*
  This allows execution if the Access Rule `Name` or inline `AccessDefinition` returns `allowed` for command's authentication details

- **oauth:** *ListOfRules*
  This rule (and only this) will match for commands that were executed with OAuth authentication. Inside ListOfRules you can use any rule described above (`acl: Name`, `AClName`, `access: Name`) and additionally you must include `scope: ListOfScopeNames` with OAuth scope names that must match scope used to generate OAuth token used in command authentication.

`who` allows the command to be executed if at least one rule matches.

If you want to require several rules to match at this same time, use `access` (see examples below).

Missing `who` rule is equivalent to `who: none` which will stop group from accepting any command.

Examples of `who` rules

This accepts user `admin@server.com` or commands originating from localhost:

```
who:
  user: admin@server.com
  ip: 127.0.0.1/8
```

This only allows execution of a command if it's invoked by user `admin@server.com` and comes from localhost address. If one of those restrictions isn't satisfied, execution will fail:

```
who:
  access:
    allow:
      user: admin@server.com
      ip: 127.0.0.1/8
```

Those rules match for users from `muc_admin` ACL both using regular authentication and OAuth:

```
who:
  access:
    allow:
      acl: muc_admin
  oauth:
    scope: "ejabberd:admin"
    access:
      allow:
        acl: muc_admin
```

Rules in `what` section

Rules in `what` section are constructed from `"strings"` literals. You can use:

- "*command_name*" of an existing [API command](#)
- *command_name* is same as before, but no need to provide `"`
- **"*"** is a wildcard rule to match all commands
- **[tag:** *tagname* **]** allows all commands that have been declared with tag `tagname`. You can consult the list of tags and their commands with: `ejabberdctl help tags`

Additionally each rule can be prepended with `!` character to change it into negative assertion rule. Command names that would match what is after `!` character will be removed from list of allowed commands.

Missing `what` rule is equivalent to `what: "!*"` which will stop group from accepting any command.

Example of `what` rules

This allows execution of all commands except command `stop`:

```
what:
  - "*"
  - "!stop"
```

This allows execution of `status` and commands with tag `session` (like `num_resources` or `status_list`):

```
what:
  - status
  - "[tag:account]"
```

This matches no command:

```
what:
  - start
  - "!*"
```

Rules in `from` section

This section allows to specify list of allowed module names that expose API to outside world. Currently those modules are `ejabberd_xmlrpc`, `mod_http_api` and `ejabberd_ctl`.

If `from` section is missing from group then all endpoints are accepted, if it's specified endpoint must be listed inside it to be allowed to execute.

Examples

Those rules allow execution of any command invoked by `ejabberdctl` shell command, or all command except `start` and `stop` for users in ACL admin, with regular authentication or `ejabberd:admin` scoped OAuth tokens.

```
api_permissions:
  "console commands":
    from:
      - ejabberd_ctl
    who: all
    what: "*"
  "admin access":
    who:
      access:
        allow:
          - acl: admin
      oauth:
        scope: "ejabberd:admin"
        access:
          allow:
            - acl: admin
    what:
      - "*"
      - "!stop"
      - "!start"
```

OAuth Support

added in 15.09

Introduction

ejabberd includes a full support OAuth 2.0 deep inside the ejabberd stack.

This OAuth integration makes ejabberd:

- an ideal project to develop XMPP applications with Web in mind, as it exposes ejabberd features as ReST or XML-RPC HTTP based API endpoints. **OAuth makes ejabberd the ideal XMPP server to integrate in a larger Web / HTTP ecosystem**.

- a more **secure tool that can leverage the use of oAuth token to authenticate, hiding your real password from the client itself**. As your password is never shared with client directly with our X-OAUTH2 authentication mechanism, user have less risks of having their primary password leaked.

- a tool that can be used at the core of larger platforms as oauth token can be used by users and admins to **delegate rights to subcomponents / subservices**.

- a tool that is friendly to other online services as users can delegate rights to others SaaS platform they are using. This will be possible to let services access your message archive, show your offline message count or with future commands send message to users and chatrooms on your behalf. This is done in a granular way, with a scope limited to a specific function. And the delegation rights for a specific app / third party can always be revoked at any time as this is usually the case with OAuth services.

You can read more on OAuth from OAuth website.

Configuration

Authentication method

An X-OAUTH2 SASL authentication mechanism is enabled by default in ejabberd.

However, if the `ejabberd_oauth` HTTP request handler is not enabled, there is no way to generate token from outside ejabberd. In this case, you may want to disable X-OAUTH2 with the disable_sasl_mechanisms top-level option in `ejabberd.yml` file, either at global or at virtual host level:

```
disable_sasl_mechanisms: ["X-OAUTH2"]
```

ejabberd listeners

To enable OAuth support in ejabberd, you need to edit your `ejabberd.yml` file to add the following snippets.

You first need to expose more HTTP endpoint in ejabberd_http modules:

- `ejabberd_oauth` is the request handler that will allow generating token for third-parties (clients, services). It is usually exposed on "/oauth" endpoint. This handler is mandatory to support OAuth.

- mod_http_api request handler enables ReST API endpoint to perform delegated actions on ejabberd using an HTTP JSON API. This handler is usually exposed on "/api" endpoint. It is optional.

- ejabberd_xmlrpc listener can be set on a separate port to query commands using the XML-RPC protocol.

Here is a example of the `listen` section in ejabberd configuration file, focusing on HTTP handlers:

```
listen:
  -
    port: 4560
    module: ejabberd_http
    request_handlers:
      ## Handle ejabberd commands using XML-RPC
```

```
      /: ejabberd_xmlrpc
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /websocket: ejabberd_http_ws
      /log: mod_log_http
      # OAuth support:
      /oauth: ejabberd_oauth
      # ReST API:
      /api: mod_http_api
```

Module configuration

Some commands are implemented by ejabberd internals and are always available, but other commands are implemented by optional modules. If the documentation of a command you want to use mentions a module, make sure you have enabled that module in `ejabberd.yml` . For example the add_rosteritem command is implemented in the mod_admin_extra module.

By the way, ejabberd implements several commands to manage OAuth, check the oauth tag documentation.

OAuth specific parameters

OAuth is configured using those top-level options:

- oauth_access
- oauth_cache_life_time
- oauth_cache_missed
- oauth_cache_rest_failure_life_time
- oauth_cache_size
- oauth_client_id_check
- oauth_db_type
- oauth_expire
- oauth_use_cache

A basic setup is to allow all accounts to create tokens, and tokens expire after an hour:

```
oauth_access: all
oauth_expire: 3600
```

authorization_token

An easy way to generate a token is using the oauth_issue_token command with the ejabberdctl shell script:

```
ejabberdctl oauth_issue_token user1@localhost 3600 ejabberd:admin

r9KFladBTYJS71OggKCifo0GJwyT7oY4 [<<"ejabberd:admin">>]  3600 seconds
```

The users can generate tokens themselves by visiting `/oauth/authorization_token` in a webview in your application or in a web browser. For example, URL can be:

```
`http://example.net:5280/oauth/authorization_token?response_type=token&client_id=Client1&redirect_uri=http://client.uri&scope=get_roster+sasl_auth`
```

Note: To use the `get_roster` scope, enable `mod_admin_extra` , because the get_roster API is defined in that module. Otherwise, the command is unknown and you will get an `invalid_scope` error. See Module configuration for details.

Parameters are described in OAuth 2.0 specification:

- `response_type` : Should be `token` .
- `client_id` : This is the name of the application that is asking for Oauth token.
- `scope` : This is the scope of the rights being delegated to the application. It will limit the feature the application can perform and thus ensure the user is not giving away more right than expected by the application. As a developer, you should always limit the scope to what you actually need.
- `redirect_uri` : After token is generated, token is passed to the application using the redirect URI. It can obviously work for web applications, but also for mobile applications, using a redirect URI that the mobile application have registered: Proper code for handling the token will thus be executed directly in the mobile application.
- `state` : State parameter is optional and use by client to pass information that will be passed as well as state parameter in the redirect URI.

Directing the user to this URL will present an authentication form summarizing what is the app requiring the token and the scope / rights that are going to be granted.

The user can then put their login and password to confirm that they accept granting delegating rights and confirm the token generation. If the provided credentials are valid, the browser or webview will redirect the user to the redirect_uri, to actually let ejabberd pass the token to the app that requested it. It can be either a Web app or `a mobile / desktop application.

redirect_uri

The `redirect_uri` originally passed in the authorization_token request will be called on successful validation of user credentials, with added parameters.

For example, redirect URI called by ejabberd can be:

```
http://client.uri/?
access_token=RHIT8DoudzOctdzBhYL9bYvXz28xQ4Oj&token_type=bearer&expires_in=3600&scope=user_get_roster+sasl_auth&state=
```

Parameters are described in OAuth specification:

- `access_token` : This is the actual token that the client application can use for OAuth authentication.
- `token_type` : ejabberd supports `bearer` token type.
- `expires_in` : This is the validity duration of the token, in seconds. When the token expires, a new authorization token will need to be generated an approved by the user.
- `scope` : Confirms the granted scope to the requesting application. Several scopes can be passed, separated by '+'.
- `state` : If a state parameter was passed by requesting application in authorization_token URL, it will be passed back to the application as a parameter of the `redirect_uri` to help with the client workflow.

Scopes

- `sasl_auth` : This scope is use to generate a token that can login over XMPP using SASL X-OAUTH2 mechanism.
- `ejabberd:admin`
- `ejabberd:user`
- Scopes for each existing API command. For example, there is a scope `registered_users` because there is a command called registered_users. Ensure you enable the module that defines the command that you want to use, see Module configuration for details.

X-OAuth2 Authentication

You can connect to ejabberd using an X-OAUTH2 token that is valid in the scope `sasl_auth` . You can use an OAuth token as generated in the previous steps instead of a password when connecting to ejabberd servers support OAuth SASL mechanism.

When enabled, X-OAUTH2 SASL mechanism is advertised in server stream features:

```
<stream:features>
  <c xmlns="http://jabber.org/protocol/caps" node="http://www.process-one.net/en/ejabberd/" ver="nM19M+JK0ZBMXK7iJAvKnmDuQus=" hash="sha-1"/>
  <register xmlns="http://jabber.org/features/iq-register"/>
  <mechanisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
    <mechanism>PLAIN</mechanism>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>X-OAUTH2</mechanism>
    <mechanism>SCRAM-SHA-1</mechanism>
  </mechanisms>
</stream:features>
```

Authentication with X-OAUTH2 is done by modifying the SASL auth element as follow:

```
<auth mechanism='X-OAUTH2'
      xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  base64("\0" + user_name + "\0" + oauth_token)
</auth>
```

The content in the auth element should be the base64 encoding of a string containing a null byte, followed by the user name, another null byte and the string representation of the user's OAuth token. This is similar to how to authenticate with a password using the PLAIN mechanism, except the token is added instead of the user's password.

The response is standard for SASL XMPP authentication. For example, on success, server will reply with:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

ReST Example

It is possible to use OAuth to authenticate a client when attempting to perform a ReST or XML-RPC query.

Configuring

First of all check all the required options are setup (listener, OAuth, API and ACL):

```
listen:
  -
    port: 5280
    ip: "::"
    module: ejabberd_http
    request_handlers:
      /api: mod_http_api
      /oauth: ejabberd_oauth

oauth_expire: 3600
oauth_access: all

api_permissions:
  "admin access":
    who:
      oauth:
        scope: "ejabberd:admin"
        access:
          allow:
            - acl: loopback
            - acl: admin
    what:
      - "*"
      - "!stop"
      - "!start"

acl:
  admin:
    user:
      - user1@localhost

modules:
  mod_admin_extra: {}
  mod_roster: {}
```

Register the account with admin rights, and another one used for the queries:

```
ejabberdctl register user1 localhost asd
ejabberdctl register user2 localhost asd
ejabberdctl add_rosteritem user2 localhost tom localhost Tom "" none
```

Obtain bearer token

Obtain a bearer token as explained in authorization_token, either using `ejabberdctl` :

```
ejabberdctl oauth_issue_token user1@localhost 3600 ejabberd:admin
r9KFladBTYJS71OggKCifo0GJwyT7oY4         [<<"ejabberd:admin">>]  3600 seconds
```

Or using a web browser:

- visit the URL `http://localhost:5280/oauth/authorization_token?response_type=token&scope=ejabberd:admin`
- User (jid): `user1@localhost`
- Password: `asd`
- and click `Accept`

This redirects to a new URL which contains the access_token, for example:

```
http://localhost:5280/oauth/authorization_token?
access_token=r9KFladBTYJS71OggKCifo0GJwyT7oY4&token_type=bearer&expires_in=31536000&scope=ejabberd:admin&state=
```

Passing credentials

When using ReST, the client authorization is done by using a bearer token (no need to pass the user and host parameters). For that, include an Authorization HTTP header like:

```
Authorization: Bearer r9KFladBTYJS71OggKCifo0GJwyT7oY4
```

Query examples

Let's use `curl` to get the list of registered_users with a HTTP GET query:

```
curl -X GET \
     -H "Authorization: Bearer r9KFladBTYJS71OggKCifo0GJwyT7oY4" \
     http://localhost:5280/api/registered_users?host=localhost

["user1","user2"]
```

Or provide the bearer token with this option:

```
curl -X GET \
     --oauth2-bearer r9KFladBTYJS71OggKCifo0GJwyT7oY4 \
     http://localhost:5280/api/registered_users?host=localhost
```

With a command like get_roster you can get your own roster, or act as an admin to get any user roster.

The HTTP endpoint does not take any parameter, so we can just do an HTTP POST with empty JSON structure list (see `-d` option).

In this example let's use a HTTP POST query:

```
curl -v -X POST \
     --oauth2-bearer r9KFladBTYJS71OggKCifo0GJwyT7oY4 \
     http://localhost:5280/api/get_roster \
     -d '{"user": "user2", "server": "localhost"}'

[{"jid":"tom@localhost","nick":"Tom","subscription":"none","ask":"none","group":""}]
```

XML-RPC Example

For XML-RPC, credentials must be passed as XML-RPC parameters, including token but also user and host parameters. This is for legacy reason, but will likely change in a future version, making user and host implicit, thanks to bearer token.

Here is an (Erlang) XML-RPC example on how to get your own roster:

```
xmlrpc:call({127, 0, 0, 1}, 4560, "/",
  {call, get_roster, [
```

```
      {struct, [{user, "peter"},
                {server, "example.com"},
                {token, "0n6LaEjyAOxVDyZChzZfoKMYxc8uUk6L"}]}]},
   false, 60000, "Host: localhost\r\n", []).
```

This will lead to sending this XML-RPC payload to server:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>get_roster</methodName>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>server</name>
            <value>
              <string>example.com</string>
            </value>
          </member>
          <member>
            <name>user</name>
            <value>
              <string>peter</string>
            </value>
          </member>
          <member>
            <name>token</name>
            <value>
              <string>0n6LaEjyAOxVDyZChzZfoKMYxc8uUk6L</string>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

To get roster of other user using admin authorization, this erlang XML-RPC code can be used:

```
xmlrpc:call({127, 0, 0, 1}, 4560, "/",
  {call, get_roster, [
    {struct, [{user, "admin"},
              {server, "example.com"},
              {token, "0n6LaEjyAOxVDyZChzZfoKMYxc8uUk6L"}
              {admin, true}]},
    {struct, [{user, "peter"},
              {server, "example.com"}]}]}]},
   false, 60000, "Host: localhost\r\n", []).
```

This is an equivalent Python 2 script:

```python
import xmlrpclib

server_url = 'http://127.0.0.1:4560'
server = xmlrpclib.ServerProxy(server_url)

LOGIN = {'user': 'admin',
         'server': 'example.com',
         'token': '0n6LaEjyAOxVDyZChzZfoKMYxc8uUk6L',
         'admin': True}

def calling(command, data):
    fn = getattr(server, command)
    return fn(LOGIN, data)

print calling('get_roster', {'user':'peter', 'server':'example.com'})
```

And this is an equivalent Python 3 script:

```python
from xmlrpc import client

server_url = 'http://127.0.0.1:4560'
server = client.ServerProxy(server_url)

LOGIN = {'user': 'admin',
         'server': 'example.com',
         'token': '0n6LaEjyAOxVDyZChzZfoKMYxc8uUk6L',
         'admin': True}

def calling(command, data):
    fn = getattr(server, command)
    return fn(LOGIN, data)

result = calling('get_roster', {'user':'peter', 'server':'example.com'})
print(result)
```

Those calls would send this XML to server:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<methodCall>
  <methodName>get_roster</methodName>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>admin</name>
            <value>
              <boolean>1</boolean>
            </value>
          </member>
          <member>
            <name>server</name>
            <value>
              <string>example.com</string>
            </value>
          </member>
          <member>
            <name>user</name>
            <value>
              <string>admin</string>
            </value>
          </member>
          <member>
            <name>token</name>
            <value>
              <string>0n6LaEjyAOxVDyZChzZfoKMYxc8uUk6L</string>
            </value>
          </member>
        </struct>
      </value>
    </param>
    <param>
      <value>
        <struct>
          <member>
            <name>user</name>
            <value>
              <string>peter</string>
            </value>
          </member>
          <member>
            <name>server</name>
            <value>
              <string>example.com</string>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

ejabberd commands

By defining command using api available through `ejabberd_commands` module, it's possible to add operations that would be available to users through `ejabberdctl` command, XML-RPC socket or JSON based REST service.

Each command needs to provide information about required arguments and produced result by filling `#ejabberd_commands` record and registering it in dispatcher by calling `ejabberd_commands:register_commands([ListOfEjabberdCommandsRecords])`.

Structure of `#ejabberd_commands` record

Writing ejabberd commands supporting OAuth

If you have existing commands that you want to make OAuth compliant, you can make them OAuth compliant very easily.

An ejabberd command is defined by an `#ejabberd_commands` Erlang record. The record requires a few fields:

- **name**: This is an atom defining the name of the command.
- **tags**: This is a list of atoms used to group the command into consistent group of commands. This is mostly used to group commands in `ejabberdctl` command-line tool. Existing categories are:
- `session`: For commands related to user XMPP sessions.
- `roster`: Commands related to contact list management.
- **desc**: Description of the command for online help.
- **module** and **function**: Module and function to call to execute the command logic.
- **args**: Argument of the command. An argument is defined by a tuple of atoms of the form `{argument_name, data_type}`. `data_type` can be one of:
- binary
- **result**: defines what the command will return.
- **policy**: Is an optional field, containing an atom that define restriction policy of the command. It can be on of: `open`, `admin`, `user`, `restricted`. Default is `restricted`, meaning the command can be used from ejabberdctl command-line tool.
- **version**: API version number where this command is available (see *API versioning* documentation for details).

To define a command that can be used by server user over ReST or XML-RPC API, you just have to define it with policy `user`. Then, you have to make sure that the function will take a user binary and a host binary as first parameter of the function. They do not have to be put in the `args` list in `#ejabberd_commands` record as the `user policy implicitly expect them.

That's all you need to have commands that can be used in a variety of ways.

Here is a example way to register commands when

```
start(_Host, _Opts) ->
    ejabberd_commands:register_commands(commands()).

stop(_Host) ->
    ejabberd_commands:unregister_commands(commands()).

%%%
%%% Register commands
%%%

commands() ->
    [#ejabberd_commands{name = user_get_roster,
                        tags = [roster],
                        desc = "Retrieve the roster",
                        longdesc =
                            "Returns a list of the contacts in a "
                            "user roster.\n\nAlso returns the state "
                            "of the contact subscription. Subscription "
                            "can be either  \"none\", \"from\", \"to\", "
                            "\"both\". Pending can be \"in\", \"out\" "
                            "or \"none\".",
                        module = ?MODULE, function = get_roster,
                        args = [],
```

```
            policy = user,
            result =
                {contacts,
                 {list,
                  {contact,
                   {tuple,
                    [{jid, string},
                     {groups, {list, {group, string}}},
                     {nick, string}, {subscription, string},
                     {pending, string}]}}}}
     ].
```

API Versioning

🏷 added in 24.02

Introduction

It is possible to support different versions of the ejabberd API. Versioning is used to ensure compatibility with third party backend that uses the API.

When a command is modified (either its declaration or its definition, breaking compatibility), those modifications can be done in a new version of the API, keeping the old command still available in the previous API version. An API version is an integer (sub-versions are not supported).

If the API client does not specify the API version, ejabberd uses by default the most recent available API version.

Alternatively, the API client can specify an API version, and ejabberd will use that one to process the query, or the most recent to the one specified. For example: if a command is defined in API versions 0, 2, 3, 7, and 9, and a client declares to support up to API version 5, then ejabberd uses the command API version 3, which is the most recent available for the one supported by the client.

API versioning is supported by `mod_http_api` ReST interface and `ejabberdctl` command line script. However `ejabberd_xmlrpc` doesn't support API versioning, and consequently it can only use the latest API version.

Command Definition

If a command is modified, a new `#ejabberd_commands` record should be defined with a `version` attribute set to the API version (an integer) where this command version is available. There is no need to add a new `#ejabberd_commands` record for commands that are not modified in a given API version, immediate inferior version is used.

By default, all commands are in API version 0, and latest API is used if no version is specified when calling `ejabberd_commands` directly without specifying a version.

API Documentation

The command documentation indicates the api version as a tag: `v1`, `v2` ... Commands not versioned do not have such a tag: they are version 0.

The ejabberd Docs: API Tags page lists the most recent API versions, and what commands are included.

To know exactly what is the format expected for a command in a specific API version, use `ejabberdctl` specifying what API version you want to consult and the command name, for example:

```
ejabberdctl --version 0 help get_roster
```

mod_http_api

The server administrator can set the default version when configuring `request_handlers`, by including a `vN` in its path, where `N` is an integer corresponding to the version.

In any case, the API client can specify a version when sending the request, by appending `vN` to the request path.

For example, when configured like:

```
listen:
  -
    request_handlers:
      /api/v0: mod_http_api
      /v1/api: mod_http_api
      /api: mod_http_api
```

See what API version will be used depending on the URL:

- `api/command` use the latest available version

- `api/command/v0` use version 0

- `api/command/v1` use version 1

- `v1/api/command` use version 1

- `v1/api/command/v0` use version 0

- `api/v0/command` use version 0

- `api/v0/command/v1` use version 1

In this example, the server administrator configured the default API version to 0:

```
listen:
  -
    request_handlers:
      /api/v0: mod_http_api
```

The client doesn't specify any version, so 0 is used:

```
$ curl -k -X POST -H "Content-type: application/json" \
  -d '{}' "http://localhost:5280/api/v0/get_loglevel"
{"levelatom":"info"}
```

This time the client requests the API version 2:

```
$ curl -k -X POST -H "Content-type: application/json" \
  -d '{}' "http://localhost:5280/api/v0/get_loglevel/v2"
"info"
```

ejabberdctl

By default the latest version of a given command is used. To use a command in a specific API version, use the `--version` switch, followed by the version number, and then the command name.

Example:

```
ejabberdctl --version 2 set_loglevel 4
```

Use the most recent API version:

```
$ ejabberdctl get_roster admin localhost
jan@localhost jan   none    subscribe       group1,group2
tom@localhost tom   none    subscribe       group3
```

Use version 0:

```
$ ejabberdctl --version 0 get_roster admin localhost
jan@localhost jan   none    subscribe       group1;group2
tom@localhost tom   none    subscribe       group3
```

Archive

ChangeLog

Version 24.02

CORE:

- Added Matrix gateway in `mod_matrix_gw`
- Support SASL2 and Bind2
- Support tls-server-end-point channel binding and sasl2 codec
- Support tls-exporter channel binding
- Support XEP-0474: SASL SCRAM Downgrade Protection
- Fix presenting features and returning results of inline bind2 elements
- `disable_sasl_scram_downgrade_protection` : New option to disable XEP-0474
- `negotiation_timeout` : Increase default value from 30s to 2m
- mod_carboncopy: Teach how to interact with bind2 inline requests

OTHER:

- ejabberdctl: Fix startup problem when having set `EJABBERD_OPTS` and logger options
- ejabberdctl: Set EJABBERD_OPTS back to `""` , and use previous flags as example
- eldap: Change logic for `eldap tls_verify=soft` and `false`
- eldap: Don't set `fail_if_no_peer_cert` for eldap ssl client connections
- Ignore hints when checking for chat states
- mod_mam: Support XEP-0424 Message Retraction
- mod_mam: Fix XEP-0425: Message Moderation with SQL storage
- mod_ping: Support XEP-0198 pings when stream management is enabled
- mod_pubsub: Normalize pubsub `max_items` node options on read
- mod_pubsub: PEP nodetree: Fix reversed logic in node fixup function
- mod_pubsub: Only care about PEP bookmarks options when creating node from scratch

SQL:

- MySQL: Support `sha256_password` auth plugin
- ejabberd_sql_schema: Use the first unique index as a primary key
- Update SQL schema files for MAM's XEP-0424
- New option `sql_flags` : right now only useful to enable `mysql_alternative_upsert`

INSTALLERS AND CONTAINER:

- Container: Add ability to ignore failures in execution of `CTL_ON_*` commands
- Container: Update to Erlang/OTP 26.2, Elixir 1.16.1 and Alpine 3.19
- Container: Update this custom ejabberdctl to match the main one
- make-binaries: Bump OpenSSL 3.2.1, Erlang/OTP 26.2.2, Elixir 1.16.1
- make-binaries: Bump many dependency versions

COMMANDS API:

- `print_sql_schema` : New command available in ejabberdctl command-line script

- ejabberdctl: Rework temporary node name generation

- ejabberdctl: Print argument description, examples and note in help

- ejabberdctl: Document exclusive ejabberdctl commands like all the others

- Commands: Add a new `muc_sub` tag to all the relevant commands

- Commands: Improve syntax of many commands documentation

- Commands: Use list arguments in many commands that used separators

- Commands: `set_presence` : switch priority argument from string to integer

- ejabberd_commands: Add the command API version as a tag `vX`

- ejabberd_ctl: Add support for list and tuple arguments

- ejabberd_xmlrpc: Fix support for restuple error response

- mod_http_api: When no specific API version is requested, use the latest

COMPILATION WITH REBAR3/ELIXIR/MIX:

- Fix compilation with Erlang/OTP 27: don't use the reserved word 'maybe'

- configure: Fix explanation of `--enable-group` option (#4135)

- Add observer and runtime_tools in releases when `--enable-tools`

- Update "make translations" to reduce build requirements

- Use Luerl 1.0 for Erlang 20, 1.1.1 for 21-26, and temporary fork for 27

- Makefile: Add `install-rel` and `uninstall-rel`

- Makefile: Rename `make rel` to `make prod`

- Makefile: Update `make edoc` to use ExDoc, requires mix

- Makefile: No need to use `escript` to run rebar|rebar3|mix

- configure: If `--with-rebar=rebar3` but rebar3 not system-installed, use local one

- configure: Use Mix or Rebar3 by default instead of Rebar2 to compile ejabberd

- ejabberdctl: Detect problem running iex or etop and show explanation

- Rebar3: Include Elixir files when making a release

- Rebar3: Workaround to fix protocol consolidation

- Rebar3: Add support to compile Elixir dependencies

- Rebar3: Compile explicitly our Elixir files when `--enable-elixir`

- Rebar3: Provide proper path to `iex`

- Rebar/Rebar3: Update binaries to work with Erlang/OTP 24-27

- Rebar/Rebar3: Remove Elixir as a rebar dependency

- Rebar3/Mix: If `dev` profile/environment, enable tools automatically

- Elixir: Fix compiling ejabberd as a dependency (#4128)

- Elixir: Fix ejabberdctl start/live when installed

- Elixir: Fix: `FORMATTER ERROR: bad return value` (#4087)

- Elixir: Fix: Couldn't find file `Elixir Hex API`

- Mix: Enable stun by default when `vars.config` not found

- Mix: New option `vars_config_path` to set path to `vars.config` (#4128)

- Mix: Fix ejabberdctl iexlive problem locating iex in an OTP release

Version 23.10

COMPILATION:

- Erlang/OTP: Raise the requirement to Erlang/OTP 20.0 as a minimum
- CI: Update tests to Erlang/OTP 26 and recent Elixir
- Move Xref and Dialyzer options from workflows to `rebar.config`
- Add sections to `rebar.config` to organize its content
- Dialyzer dirty workarounds because `re:mp()` is not an exported type
- When installing module already configured, keep config as example
- Elixir 1.15 removed support for `--app`
- Elixir: Improve support to stop external modules written in Elixir
- Elixir: Update syntax of function calls as recommended by Elixir compiler
- Elixir: When building OTP release with mix, keep `ERLANG_NODE=ejabberd@localhost`
- `ejabberdctl`: Pass `ERLANG_OPTS` when calling `erl` to parse the `INET_DIST_INTERFACE` ([#4066](#))

COMMANDS:

- `create_room_with_opts`: Fix typo and move examples to `args_example` ([#4080](#))
- `etop`: Let `ejabberdctl etop` work in a release (if `observer` application is available)
- `get_roster`: Command now returns groups in a list instead of newlines ([#4088](#))
- `halt`: New command to halt ejabberd abruptly with an error status code
- `ejabberdctl`: Fix calling ejabberdctl command with wrong number of arguments with Erlang 26
- `ejabberdctl`: Improve printing lists in results
- `ejabberdctl`: Support `policy=user` in the help and return proper arguments
- `ejabberdctl`: Document how to stop a debug shell: control+g

CONTAINER:

- Dockerfile: Add missing dependency for mssql databases
- Dockerfile: Reorder stages and steps for consistency
- Dockerfile: Use Alpine as base for `METHOD=package`
- Dockerfile: Rename packages to improve compatibility
- Dockerfile: Provide specific OTP and elixir vsn for direct compilation
- Halt ejabberd if a command in `CTL_ON_` fails during ejabberd startup

CORE:

- `auth_external_user_exists_check`: New option ([#3377](#))
- `gen_mod`: Extend `gen_mod` API to simplify hooks and IQ handlers registration
- `gen_mod`: Add shorter forms for `gen_mod` hook/`iq_handler` API
- `gen_mod`: Update modules to the new `gen_mod` API
- `install_contrib_modules`: New option to define contrib modules to install automatically
- `unix_socket`: New listener option, useful when setting unix socket files ([#4059](#))
- `ejabberd_systemd`: Add a few debug messages
- `ejabberd_systemd`: Avoid using `gen_server` timeout ([#4054](#))([#4058](#))
- `ejabberd_listener`: Increase default listen queue backlog value to 128, which is the default value on both Linux and FreeBSD ([#4025](#))
- OAuth: Handle `badpass` error message

- When sending message on behalf of user, trigger `user_send_packet` ([#3990](#))
- Web Admin: In roster page move the `AddJID` textbox to top ([#4067](#))
- Web Admin: Show a warning when visiting webadmin with non-privileged account ([#4089](#))

DOCS:

- Example configuration: clarify 5223 tls options; specify s2s shaper
- Make sure that `policy=user` commands have `host` instead of `server` arg in docs
- Improve syntax of many command descriptions for the Docs site
- Move example Perl extauth script from ejabberd git to Docs site
- Remove obsolete example files, and add link in Docs to the archived copies

INSTALLERS (`MAKE-BINARIES`):

- Bump Erlang/OTP version to 26.1.1, and other dependencies
- Remove outdated workaround
- Don't build Linux-PAM examples
- Fix check for current Expat version
- Apply minor simplifications
- Don't duplicate config entries
- Don't hard-code musl version
- Omit unnecessary glibc setting
- Set kernel version for all builds
- Let curl fail on HTTP errors

MODULES:

- `mod_muc_log`: Add trailing backslash to URLs shown in disco info
- `mod_muc_occupantid`: New module with support for XEP-0421 Occupant Id ([#3397](#))
- `mod_muc_rtbl`: Better error handling in ([#4050](#))
- `mod_private`: Add support for XEP-0402 PEP Native Bookmarks
- `mod_privilege`: Don't fail to edit roster ([#3942](#))
- `mod_pubsub`: Fix usage of `plugins` option, which produced `default_node_config` ignore ([#4070](#))
- `mod_pubsub`: Add `pubsub_delete_item` hook
- `mod_pubsub`: Report support of `config-node-max` in pep
- `mod_pubsub`: Relay pubsub iq queries to muc members without using bare jid ([#4093](#))
- `mod_pubsub`: Allow pubsub node owner to overwrite items published by other persons
- `mod_push_keepalive`: Delay `wake_on_start`
- `mod_push_keepalive`: Don't let hook crash
- `mod_push`: Add `notify_on` option
- `mod_push`: Set `last-message-sender` to bare JID
- `mod_register_web`: Make redirect to page that end with `/` ([#3177](#))
- `mod_shared_roster_ldap`: Don't crash in `get_member_jid` on empty output ([#3614](#))

MUC:

- Add support to register nick in a room ([#3455](#))
- Convert `allow_private_message` MUC room option to `allowpm` ([#3736](#))
- Update xmpp version to send `roomconfig_changesubject` in disco#info ([#4085](#))

- Fix crash when loading room from DB older than ffa07c6, 23.04

- Fix support to retract a MUC room message

- Don't always store messages passed through `muc_filter_message` (#4083)

- Pass also MUC room retract messages over the `muc_filter_message` (#3397)

- Pass MUC room private messages over the `muc_filter_message` too (#3397)

- Store the subject author JID, and run `muc_filter_message` when sending subject (#3397)

- Remove existing role information for users that are kicked from room (#4035)

- Expand rule "mucsub subscribers are members in members only rooms" to more places

SQL:

- Add ability to force alternative upsert implementation in mysql

- Properly parse mysql version even if it doesn't have type tag

- Use prepared statement with mysql

- Add alternate version of mysql upsert

- `ejabberd_auth_sql` : Reset scram fields when setting plain password

- `mod_privacy_sql` : Fix return values from `calculate_diff`

- `mod_privacy_sql` : Optimize `set_list`

- `mod_privacy_sql` : Use more efficient way to calculate changes in `set_privacy_list`

Version 23.04

GENERAL:

- New `s2s_out_bounce_packet` hook

- Re-allow anonymous connection for connection without client certificates (#3985)

- Stop `ejabberd_system_monitor` before stopping node

- `captcha_url` option now accepts `auto` value, and it's the default

- `mod_mam` : Add support for XEP-0425: Message Moderation

- `mod_mam_sql` : Fix problem with results of mam queries using rsm with max and before

- `mod_muc_rtbl` : New module for Real-Time Block List for MUC rooms (#4017)

- `mod_roster` : Set roster name from XEP-0172, or the stored one (#1611)

- `mod_roster` : Preliminary support to store extra elements in subscription request (#840)

- `mod_pubsub` : Pubsub xdata fields `max_item/item_expira/children_max` use `max` not `infinity`

- `mod_vcard_xupdate` : Invalidate `vcard_xupdate` cache on all nodes when vcard is updated

ADMIN:

- `ext_mod` : Improve support for loading `*.so` files from `ext_mod` dependencies

- Improve output in `gen_html_doc_for_commands` command

- Fix ejabberdctl output formatting (#3979)

- Log HTTP handler exceptions

MUC:

- New command `get_room_history`

- Persist `none` role for outcasts

- Try to populate room history from mam when unhibernating

- Make `mod_muc_room:set_opts` process persistent flag first

- Allow passing affiliations and subscribers to `create_room_with_opts` command
- Store state in db in `mod_muc:create_room()`
- Make subscribers members by default

SQL SCHEMAS:

- Fix a long standing bug in new schema migration
- `update_sql` command: Many improvements in new schema migration
- `update_sql` command: Add support to migrate MySQL too
- Change PostgreSQL SERIAL to BIGSERIAL columns
- Fix minor SQL schema inconsistencies
- Remove unnecessary indexes
- New SQL schema migrate fix

MS SQL:

- MS SQL schema fixes
- Add `new` schema for MS SQL
- Add MS SQL support for new schema migration
- Minor MS SQL improvements
- Fix MS SQL error caused by `ORDER BY` in subquery

SQL TESTS:

- Add support for running tests on MS SQL
- Add ability to run tests on upgraded DB
- Un-deprecate `ejabberd_config:set_option/2`
- Use python3 to run `extauth.py` for tests
- Correct README for creating test docker MS SQL DB
- Fix TSQLlint warnings in MSSQL test script

TESTING:

- Fix Shellcheck warnings in shell scripts
- Fix Remark-lint warnings
- Fix Prospector and Pylint warnings in test `extauth.py`
- Stop testing ejabberd with Erlang/OTP 19.3, as Github Actions no longer supports ubuntu-18.04
- Test only with oldest OTP supported (20.0), newest stable (25.3) and bleeding edge (26.0-rc2)
- Upload Common Test logs as artifact in case of failure

ECS CONTAINER IMAGE:

- Update Alpine to 3.17 to get Erlang/OTP 25 and Elixir 1.14
- Add `tini` as runtime init
- Set `ERLANG_NODE` fixed to `ejabberd@localhost`
- Upload images as artifacts to Github Actions
- Publish tag images automatically to ghcr.io

EJABBERD CONTAINER IMAGE:

- Update Alpine to 3.17 to get Erlang/OTP 25 and Elixir 1.14
- Add `METHOD` to build container using packages ([#3983](#3983))
- Add `tini` as runtime init

- Detect runtime dependencies automatically

- Remove unused Mix stuff: ejabberd script and static COOKIE

- Copy captcha scripts to `/opt/ejabberd-*/lib` like the installers

- Expose only `HOME` volume, it contains all the required subdirs

- ejabberdctl: Don't use `.../releases/COOKIE`, it's no longer included

INSTALLERS:

- make-binaries: Bump versions, e.g. erlang/otp to 25.3

- make-binaries: Fix building with erlang/otp v25.x

- make-packages: Fix for installers workflow, which didn't find lynx

Version 23.01

GENERAL:

- Add `misc:uri_parse/2` to allow declaring default ports for protocols

- CAPTCHA: Add support to define module instead of path to script

- Clustering: Handle `mnesia_system_event mnesia_up` when other node joins this ([#3842](#3842))

- ConverseJS: Don't set i18n option because Converse enforces it instead of browser lang ([#3951](#3951))

- ConverseJS: Try to redirect access to files `mod_conversejs` to CDN when there is no local copies

- ext_mod: compile C files and install them in ejabberd's `priv`

- ext_mod: Support to get module status from Elixir modules

- make-binaries: reduce log output

- make-binaries: Bump zlib version to 1.2.13

- MUC: Don't store mucsub presence events in offline storage

- MUC: `hibernation_time` is not an option worth storing in room state ([#3946](#3946))

- Multicast: Jid format when `multicastc` was cached ([#3950](#3950))

- mysql: Pass `ssl` options to mysql driver

- pgsql: Do not set `standard_conforming_strings` to `off` ([#3944](#3944))

- OAuth: Accept `jid` as a HTTP URL query argument

- OAuth: Handle when client is not identified

- PubSub: Expose the `pubsub#type` field in `disco#info` query to the node ([#3914](#3914))

- Translations: Update German translation

ADMIN:

- `api_permissions`: Fix option crash when doesn't have `who:` section

- `log_modules_fully`: New option to list modules that will log everything

- `outgoing_s2s_families`: Changed option's default to IPv6, and fall back to IPv4

- Fix bash completion when using Relive or other install methods

- Fix portability issue with some shells ([#3970](#3970))

- Allow admin command to subscribe new users to `members_only` rooms

- Use alternative `split/2` function that works with Erlang/OTP as old as 19.3

- Silent warning in OTP24 about not specified `cacerts` in SQL connections

- Fix compilation warnings with Elixir 1.14

DOAP:

- Support extended `-protocol` erlang attribute

- Add extended RFCs and XEP details to some protocol attributes

- `tools/generate-doap.sh` : New script to generate DOAP file, add `make doap` (#3915)

- `ejabberd.doap` : New DOAP file describing ejabberd supported protocols

MQTT:

- Add MQTT bridge module

- Add support for certificate authentication in MQTT bridge

- Implement reload in MQTT bridge

- Add support for websockets to MQTT bridge

- Recognize ws5/wss5 urls in MQTT bridge

- `mqtt_publish` : New hook for MQTT publish event

- `mqtt_(un)subscribe` : New hooks for MQTT subscribe & unsubscribe events

VSCODE:

- Improve `.devcontainer` to use use devcontainer image and `.vscode`

- Add `.vscode` files to instruct VSCode how to run ejabberd

- Add Erlang LS default configuration

- Add Elvis default configuration

Version 22.10

CORE:

- Add `log_burst_limit_*` options (#3865)

- Support `ERL_DIST_PORT` option to work without epmd

- Auth JWT: Catch all errors from `jose_jwt:verify` and log debugging details (#3890)

- CAPTCHA: Support `@VERSION@` and `@SEMVER@` in `captcha_cmd` option (#3835)

- HTTP: Fix unix socket support (#3894)

- HTTP: Handle invalid values in `X-Forwarded-For` header more graceful y

- Listeners: Let module take over socket

- Listeners: Don't register listeners that failed to start in config reload

- `mod_admin_extra` : Handle empty roster group names

- `mod_conversejs` : Fix crash when mod_register not enabled (#3824)

- `mod_host_meta` : Complain at start if listener is not encrypted

- `mod_ping` : Fix regression on `stop_ping` in clustering context (#3817)

- `mod_pubsub` : Don't crash on command failures

- `mod_shared_roster` : Fix cache invalidation

- `mod_shared_roster_ldap` : Update roster_get hook to use `#roster_item{}`

- `prosody2ejabberd` : Fix parsing of scram password from prosody

MIX:

- Fix MIX's filter_nodes

- Return user jid on join

- `mod_mix_pam` : Add new MIX namespaces to disco features

- `mod_mix_pam` : Add handling of IQs with newer MIX namespaces

- `mod_mix_pam` : Do roster pushes on join/leave

- `mod_mix_pam` : Parse sub elements of the mix join remote result

- `mod_mix_pam` : Provide MIX channels as roster entries via hook

- `mod_mix_pam` : Display joined channels on webadmin page

- `mod_mix_pam` : Adapt to renaming of `participant-id` from mix_roster_channel record

- `mod_roster` : Change hook type from `#roster{}` to `#roster_item{}`

- `mod_roster` : Respect MIX `<annotate/>` setting

- `mod_roster` : Adapt to change of mix_annotate type to boolean in roster_query

- `mod_shared_roster` : Fix wrong hook type `#roster{}` (now `#roster_item{}`)

MUC:

- Store role, and use it when joining a moderated room ([#3330](#))

- Don't persist `none` role ([#3330](#))

- Allow MUC service admins to bypass max_user_conferences limitation

- Show allow_query_users room option in disco info ([#3830](#))

- mod_muc_room: Don't set affiliation to `none` if it's already `none` in `process_item_change/3`

- Fix mucsub unsubscribe notification payload to have muc_unsubcribe in it

- Allow muc_{un}subscribe hooks to modify sent packets

- Pass room state to muc_{un}subscribed hook

- The archive_msg export fun requires MUC Service for room archives

- Export `mod_muc_admin:get_room_pid/2`

- Export function for getting room diagnostics

SQL:

- Handle errors reported from begin/commit inside transaction

- Make connection close errors bubble up from inside sql transaction

- Make first sql reconnect wait shorter time

- React to sql driver process exit earlier

- Skip connection exit message when we triggered reconnection

- Add syntax_tools to applications, required when using ejabberd_sql_pt ([#3869](#))

- Fix mam delete_old_messages_batch for sql backend

- Use `INSERT ... ON DUPLICATE KEY UPDATE` for upsert on mysql

- Update mysql library

- Catch mysql connection being close earlier

BUILD:

- `make all` : Generate start scripts here, not in `make install` ([#3821](#))

- `make clean` : Improve this and "distclean"

- `make deps` : Ensure deps configuration is ran when getting deps ([#3823](#))

- `make help` : Update with recent changes

- `make install` : Don't leak DESTDIR in files copied by 'make install'

- `make options` : Fix error reporting on OTP24+

- `make update` : configure also in this case, similarly to `make deps`

- Add definition to detect OTP older than 25, used by ejabberd_auth_http

- Configure eimp with mix to detect image convert properly (#3823)
- Remove unused macro definitions detected by rebar3_hank
- Remove unused header files which content is already in xmpp library

CONTAINER:

- Get ejabberd-contrib sources to include them
- Copy `.ejabberd-modules` directory if available
- Do not clone repo inside container build
- Use `make deps`, which performs additional steps (#3823)
- Support `ERL_DIST_PORT` option to work without epmd
- Copy `ejabberd-docker-install.bat` from docker-ejabberd git and rename it
- Set a less frequent healthcheck to reduce CPU usage (#3826)
- Fix build instructions, add more podman examples

INSTALLERS:

- make-binaries: Include CAPTCHA script with release
- make-binaries: Edit rebar.config more carefully
- make-binaries: Fix linking of EIMP dependencies
- make-binaries: Fix GitHub release version checks
- make-binaries: Adjust Mnesia spool directory path
- make-binaries: Bump Erlang/OTP version to 24.3.4.5
- make-binaries: Bump Expat and libpng versions
- make-packages: Include systemd unit with RPM
- make-packages: Fix permissions on RPM systems
- make-installers: Support non-root installation
- make-installers: Override code on upgrade
- make-installers: Apply cosmetic changes

EXTERNAL MODULES:

- ext_mod: Support managing remote nodes in the cluster
- ext_mod: Handle correctly when COMMIT.json not found
- Don't bother with COMMIT.json user-friendly feature in automated user case
- Handle not found COMMIT.json, for example in GH Actions
- Add WebAdmin page for managing external modules

WORKFLOWS ACTIONS:

- Update workflows to Erlang 25
- Update workflows: Ubuntu 18 is deprecated and 22 is added
- CI: Remove syntax_tools from applications, as fast_xml fails Dialyzer
- Runtime: Add Xref options to be as strict as CI

Version 22.05

CORE

- C2S: Don't expect that socket will be available in `c2s_terminated` hook
- Event handling process hook tracing

- Guard against `erlang:system_info(logical_processors)` not always returning a number
- `domain_balancing` : Allow for specifying `type` only, without specifying `component_number`

MQTT

- Add TLS certificate authentication for MQTT connections
- Fix login when generating client id, keep connection record (#3593)
- Pass property name as expected in mqtt_codec (fixes login using MQTT 5)
- Support MQTT subscriptions spread over the cluster (#3750)

MUC

- Attach meta field with real jid to mucsub subscription events
- Handle user removal
- Stop empty MUC rooms 30 seconds after creation
- `default_room_options` : Update options configurable
- `subscribe_room_many_max_users` : New option in `mod_muc_admin`

MOD_CONVERSEJS

- Improved options to support `@HOST@` and `auto` values
- Set `auth` and `register` options based on ejabberd configuration
- `conversejs_options` : New option
- `conversejs_resources` : New option

PUBSUB

- `mod_pubsub` : Allow for limiting `item_expire` value
- `mod_pubsub` : Unsubscribe JID on whitelist removal
- `node_pep` : Add config-node and multi-items features (#3714)

SQL

- Improve compatibility with various db engine versions
- Sync old-to-new schema script with reality (#3790)
- Slight improvement in MSSQL testing support, but not yet complete

OTHER MODULES

- `auth_jwt` : Checking if an user is active in SM for a JWT authenticated user (#3795)
- `mod_configure` : Implement Get List of Registered/Online Users from XEP-0133
- `mod_host_meta` : New module to serve host-meta files, see XEP-0156
- `mod_mam` : Store all mucsub notifications not only message notifications
- `mod_ping` : Delete ping timer if resource is gone after the ping has been sent
- `mod_ping` : Don't send ping if resource is gone
- `mod_push` : Fix notifications for pending sessions (XEP-0198)
- `mod_push` : Keep push session ID on session resume
- `mod_shared_roster` : Adjust special group cache size
- `mod_shared_roster` : Normalize JID on unset_presence (#3752)
- `mod_stun_disco` : Fix parsing of IPv6 listeners

DEPENDENCIES

- autoconf: Supported from 2.59 to the new 2.71

- fast_tls: Update to 1.1.14 to support OpenSSL 3

- jiffy: Update to 1.1.1 to support Erlang/OTP 25.0-rc1

- luerl: Update to 1.0.0, now available in hex.pm

- lager: This dependency is used only when Erlang is older than 22

- rebar2: Updated binary to work from Erlang/OTP 22 to 25

- rebar3: Updated binary to work from Erlang/OTP 22 to 25

- `make update` : Fix when used with rebar 3.18

COMPILE

- `mix release` : Copy `include/` files for ejabberd, deps and otp, in `mix.exs`

- `rebar3 release` : Fix ERTS path in `ejabberdctl`

- `configure.ac` : Set default ejabberd version number when not using git

- `mix.exs` : Move some dependencies as optional

- `mix.exs` : No need to use Distillery, Elixir has built-in support for OTP releases (#3788)

- `tools/make-binaries` : New script for building Linux binaries

- `tools/make-installers` : New script for building command line installers

START

- New `make relive` similar to `ejabberdctl live` without installing

- `ejabberdctl` : Fix some warnings detected by ShellCheck

- `ejabberdctl` : Mention in the help: `etop` , `ping` and `started` / `stopped`

- `make rel` : Switch to paths: `conf/` , `database/` , `logs/`

- `mix.exs` : Add `-boot` and `-boot_var` in `ejabberdctl` instead of adding `vm.args`

- `tools/captcha.sh` : Fix some warnings detected by ShellCheck

COMMANDS

- Accept more types of ejabberdctl commands arguments as JSON-encoded

- `delete_old_mam_messages_batch` : New command with rate limit

- `delete_old_messages_batch` : New command with rate limit

- `get_room_occupants_number` : Don't request the whole MUC room state (#3684, #1964)

- `get_vcard` : Add support for MUC room vCard

- `oauth_revoke_token` : Add support to work with all backends

- `room_unused_*` : Optimize commands in SQL by reusing `created_at`

- `rooms_unused_...` : Let `get_all_rooms` handle `global` argument (#3726)

- `stop|restart` : Terminate ejabberd_sm before everything else to ensure sessions closing (#3641)

- `subscribe_room_many` : New command

TRANSLATIONS

- Updated Catalan

- Updated French

- Updated German

- Updated Portuguese

- Updated Portuguese (Brazil)

- Updated Spanish

WORKFLOWS

- CI: Publish CT logs and Cover on failure to an external GH Pages repo

- CI: Test shell scripts using ShellCheck (#3738)

- Container: New workflow to build and publish containers

- Installers: Add job to create draft release

- Installers: New workflow to build binary packages

- Runtime: New workflow to test compilation, rel, starting and ejabberdctl

Version 21.12

COMMANDS

- `create_room_with_opts` : Fixed when using SQL storage

- `change_room_option` : Add missing fields from config inside `mod_muc_admin:change_options`

- piefxis: Fixed arguments of all commands

MODULES

- mod_caps: Don't forget caps on XEP-0198 resumption

- mod_conversejs: New module to serve a simple page for Converse.js

- mod_http_upload_quota: Avoid `max_days` race

- mod_muc: Support MUC hats (XEP-0317, conversejs/prosody compatible)

- mod_muc: Optimize MucSub processing

- mod_muc: Fix exception in mucsub {un}subscription events multicast handler

- mod_multicast: Improve and optimize multicast routing code

- mod_offline: Allow storing non-composing x:events in offline

- mod_ping: Send ping from server, not bare user JID

- mod_push: Fix handling of MUC/Sub messages

- mod_register: New allow_modules option to restrict registration modules

- mod_register_web: Handle unknown host gracefully

- mod_register_web: Use mod_register configured restrictions

PUBSUB

- Add `delete_expired_pubsub_items` command

- Add `delete_old_pubsub_items` command

- Optimize publishing on large nodes (SQL)

- Support unlimited number of items

- Support `max_items=max` node configuration

- Bump default value for `max_items` limit from 10 to 1000

- Use configured `max_items` by default

- node_flat: Avoid catch-all clauses for RSM

- node_flat_sql: Avoid catch-all clauses for RSM

SQL

- Use `INSERT ... ON CONFLICT` in SQL_UPSERT for PostgreSQL >= 9.5

- mod_mam export: assign MUC entries to the MUC service

- MySQL: Fix typo when creating index

- PgSQL: Add SASL auth support, PostgreSQL 14
- PgSQL: Add missing SQL migration for table `push_session`
- PgSQL: Fix `vcard_search` definition in pgsql new schema

OTHER

- `captcha-ng.sh`: "sort -R" command not POSIX, added "shuf" and "cat" as fallback
- Make s2s connection table cleanup more robust
- Update export/import of scram password to XEP-0227 1.1
- Update Jose to 1.11.1 (the last in hex.pm correctly versioned)

Version 21.07

COMPILATION

- Add rebar3 3.15.2 binary
- Add support for mix to: `./configure --enable-rebar=mix`
- Improved `make rel` to work with rebar3 and mix
- Add `make dev` to build a development release with rebar3 or mix
- Hex: Add `sql/` and `vars.config` to Hex package files
- Hex: Update mix applications list to fix error `p1_utils is listed as both...`
- There are so many targets in Makefile... add `make help`
- Fix extauth.py failure in test suite with Python 3
- Added experimental support for GitHub Codespaces
- Switch test service from TravisCI to GitHub Actions

COMMANDS:

- Display extended error message in ejabberdctl
- Remove SMP option from ejabberdctl.cfg, `-smp` was removed in OTP 21
- `create_room`: After creating room, store in DB if it's persistent
- `help`: Major changes in its usage and output
- `srg_create`: Update to use `label` parameter instead of `name`

MODULES:

- ejabberd_listener: New `send_timeout` option
- mod_mix: Improvements to update to 0.14.1
- mod_muc_room: Don't leak owner JIDs
- mod_multicast: Routing for more MUC packets
- mod_multicast: Correctly strip only other bcc addresses
- mod_mqtt: Allow shared roster group placeholder in mqtt topic
- mod_pubsub: Several fixes when using PubSub with RSM
- mod_push: Handle MUC/Sub events correctly
- mod_shared_roster: Delete cache after performing change to be sure that in cache will be up to date data
- mod_shared_roster: Improve database and caching
- mod_shared_roster: Reconfigure cache when options change
- mod_vcard: Fix invalid_encoding error when using extended plane characters in vcard
- mod_vcard: Update econf:vcard() to generate correct vcard_temp record

- WebAdmin: New simple pages to view mnesia tables information and content
- WebSocket: Fix typos

SQL:

- MySQL Backend Patch for scram-sha512
- SQLite: When exporting for SQLite, use its specific escape options
- SQLite: Minor fixes for new_sql_schema support
- mod_privacy: Cast as boolean when exporting privacy_list_data to PostgreSQL
- mod_mqtt: Add mqtt_pub table definition for MSSQL
- mod_shared_roster: Add missing indexes to `sr_group` tables in all SQL databases

Version 21.04

API COMMANDS:

- `add_rosteritem/...` : Add argument guards to roster commands
- `get_user_subscriptions` : New command for MUC/Sub
- `remove_mam_for_user_with_peer` : Fix when removing room archive
- `send_message` : Fix bug introduced in ejabberd 21.01
- `set_vcard` : Return modules errors

BUILD AND SETUP:

- Allow ejabberd to be compatible as a dependency for an Erlang project using rebar3
- CAPTCHA: New question/answer-based CAPTCHA script
- `--enable-lua` : new configure option for luerl instead of --enable-tools
- Remove support for HiPE, it was experimental and Erlang/OTP 24 removes it
- Update `sql_query` record to handle the Erlang/OTP 24 compiler reports
- Updated dependencies to fix Dialyzer warnings

MISCELLANEOUS:

- CAPTCHA: Update `FORM_TYPE` from captcha to register
- LDAP: fix eldap certificate verification
- MySQL: Fix for "specified key was too long"
- Translations: updated the Esperanto, Greek, and Japanese translations
- Websocket: Fix PONG responses

MODULES:

- `mod_block_strangers` : If stanza is type error, allow it passing
- `mod_caps` : Don't request roster when not needed
- `mod_caps` : Skip reading roster in one more case
- `mod_mam` : Remove `queryid` from MAM fin element
- `mod_mqtt` : When deregistering XMPP account, close its MQTT sessions
- `mod_muc` : Take in account subscriber's affiliation when checking access to moderated room
- `mod_muc` : Use monitors to track online and hard-killed rooms
- `mod_muc` : When occupant is banned, remove his subscriptions too
- `mod_privacy` : Make fetching roster lazy
- `mod_pubsub` : Don't fail on PEP unsubscribe

- `mod_pubsub` : Fix `gen_pubsub_node:get_state` return value

- `mod_vcard` : Obtain and provide photo type in vCard LDAP

Version 21.01

- `log_rotate_size` option: Fix handling of 'infinity' value

- `mod_time` : Fix invalid timezone

- Auth JWT: New `check_decoded_jwt` hook runs the default JWT verifier

- MUC: Allow non-occupant non-subscribed service admin send private MUC message

- MUC: New `max_password` and `max_captcha_whitelist` options

- OAuth: New `oauth_cache_rest_failure_life_time` option

- PEP: Skip reading pep nodes that we know won't be requested due to caps

- SQL: Add sql script to migrate mysql from old schema to new

- SQL: Don't use REPLACE for upsert when there are "-" fields.

- Shared Rosters LDAP: Add multi-domain support (and flexibility)

- Sqlite3: Fix dependency version

- Stun: Block loopback addresses by default

- Several documentation fixes and clarifications

COMMANDS:

- `decide_room` : Use better fallback value for room activity time when skipping room

- `delete_old_message` : Fix when using sqlite spool table

- `module_install` : Make ext_mod compile module with debug_info flags

- `room_unused_*` : Don't fetch subscribers list

- `send_message` : Don't include empty in messages

- `set_room_affiliation` : Validate affiliations

RUNNING:

- Docker: New `Dockerfile` and `devcontainer.json`

- New `ejabberdctl foreground-quiet`

- Systemd: Allow for listening on privileged ports

- Systemd: Integrate nicely with systemd

TRANSLATIONS:

- Moved gettext PO files to a new `ejabberd-po` repository

- Improved several translations: Catalan, Chinese, German, Greek, Indonesian, Norwegian, Portuguese (Brazil), Spanish.

Version 20.12

- Add support for `SCRAM-SHA-{256,512}-{PLUS}` authentication

- Don't use same value in cache for user don't exist and wrong password

- `outgoing_s2s_ipv*_address` : New options to set ipv4/ipv6 outbound s2s out interface

- s2s_send_packet: this hook now filters outgoing s2s stanzas

- start_room: new hook runs when a room process is started

- check_decoded_jwt: new hook to check decoded JWT after success authentication

ADMIN

- Docker: Fix DB initialization

- New sql_odbc_driver option: choose the mssql ODBC driver

- Rebar3: Fully supported. Enable with `./configure --with-rebar=/path/to/rebar3`

- systemd: start ejabberd in foreground

MODULES:

- MAM: Make sure that jid used as base in mam xml_compress is bare

- MAM: Support for MAM Flipped Pages

- MUC: Always show MucSub subscribers nicks

- MUC: Don't forget not-persistent rooms in load_permanent_rooms

- MUC Admin: Better error reporting

- MUC Admin: Fix commands with hibernated rooms

- MUC Admin: Many improvements in rooms_unused_list/destroy

- MUC Admin: create_room_with_opts Store options only if room starts

- Pubsub: Remove 'dag' node plugin documentation

- Push: Fix API call return type on error

- Push: Support cache config changes on reload

- Register: Allow for account-removal-only setup again

- Roster: Make roster subscriptions work better with invalid roster state in db

- Vcard: Fix vCard search by User when using Mnesia

- WebAdmin: Allow vhost admins to view WebAdmin menus

- WebAdmin: Don't do double utf-8 conversion on translated strings

- WebAdmin: Mark dangerous buttons with CSS

- WebSocket: Make websocket send put back pressure on c2s process

Version 20.07

CHANGES IN THIS VERSION

- Add support for using unix sockets in listeners.

- Make this version compatible with erlang R23

- Make room permissions checks more strict for subscribers

- Fix problem with muc rooms crashing when using muc logger with some locales

- Limit stat calls that logger module issues

- Don't throw errors when using user_regexp acl rule and having non-matching host

- Fix problem with leaving old data when updating shared rosters

- Fix edge case that caused failure of resuming old sessions with stream management.

- Fix crash when room that was started with logging enabled was later changed to logging disabled

- Increase default shaper limits (this should help with delays for clients that are using jingle)

- Fix couple compatibility problems which prevented working on erlang R19

- Fix sending presence unavailable when session terminates for clients that only send directed presences (helps with sometimes not leaving muc rooms on disconnect).

- Prevent supervisor errors for sockets that were closed before they were passed to handler modules

- Make stun module work better with ipv6 addresses

Version 20.03

CHANGES IN THIS VERSION

- Add support of ssl connection when connection to mysql database (configured with `sql_ssl: true` option)
- Experimental support for cockroachdb when configured with postgres connector
- Add cache and optimize queries issued by `mod_shared_roster`, this should greatly improve performance of this module when used with `sql` backend
- Fix problem with accessing webadmin
- Make webadmin work even when url is missing trailing slash
- When compiling external modules with ext_mod, use flags that were detected during compilation of ejabberd
- Make config changed to ldap options be updated when issued `reload_config` command
- Fix `room_empty_destory` command
- Fix reporting errors in `send_stanza` command when xml passed to it couldn't be passed correctly

Version 20.02

CHANGES IN THIS VERSION

- Fix problems when trying to use string format with unicode values directly in xmpp nodes
- Add missing oauth_client table declaration in lite.new.sql
- Improve compatibility with CocroachDB
- Fix importing of piefxis files that did use scram passwords
- Fix importing of piefxis files that had multiple includes in them
- Update jiffy dependency
- Allow storage of emojis when using mssql database (Thanks to Christoph Scholz)
- Make ejabberd_auth_http be able to use auth_opts
- Make custom_headers options in http modules correctly override built-in values
- Fix return value of reload_config and dump_config commands

Version 20.01

NEW FEATURES

- Implement OAUTH authentication in mqtt
- Make logging infrastructure use new logger introduced in Erlang (requires OTP22)
- New configuration parser/validator
- Initial work on being able to use CockroachDB as database backend
- Add gc command
- Add option to disable using prepared statements on Postgresql
- Implement routine for converting password to SCRAM format for all backends not only SQL
- Add infrastructure for having module documentation directly in individual module source code
- Generate man page automatically
- Implement copy feature in mod_carboncopy

FIXES

- Make webadmin work with configurable paths
- Fix handling of result in xmlrpc module

- Make webadmin work even when accessed through not declared domain

- Better error reporting in xmlrpc

- Limit amount of results returned by disco queries to pubsub nodes

- Improve validation of configured JWT keys

- Fix race condition in Redis/SQL startup

- Fix loading order of third party modules

- Fix reloading of ACL rules

- Make account removal requests properly route response

- Improve handling of malformed inputs in send_message command

- Omit push notification if storing message in offline storage failed

- Fix crash in stream management when timeout was not set

Version 19.09

ADMIN

- The minimum required Erlang/OTP version is now 19.3

- Fix API call using OAuth (#2982)

- Rename MUC command arguments from Host to Service (#2976)

WEBADMIN

- Don't treat 'Host' header as a virtual XMPP host (#2989)

- Fix some links to Guide in WebAdmin and add new ones (#3003)

- Use select fields to input host in WebAdmin Backup (#3000)

- Check account auth provided in WebAdmin is a local host (#3000)

ACME

- Improve ACME implementation

- Fix IDA support in ACME requests

- Fix unicode formatting in ACME module

- Log an error message on IDNA failure

- Support IDN hostnames in ACME requests

- Don't attempt to create ACME directory on ejabberd startup

- Don't allow requesting certificates for localhost or IP-like domains

- Don't auto request certificate for localhost and IP-like domains

- Add listener for ACME challenge in example config

AUTHENTICATION

- JWT-only authentication for some users (#3012)

MUC

- Apply default role after revoking admin affiliation (#3023)

- Custom exit message is not broadcast (#3004)

- Revert "Affiliations other than admin and owner cannot invite to members_only rooms" (#2987)

- When join new room with password, set pass and password_protected (#2668)

- Improve rooms_* commands to accept 'global' as MUC service argument (#2976)

- Rename MUC command arguments from Host to Service (#2976)

SQL

- Fix transactions for Microsoft SQL Server (#2978)
- Spawn SQL connections on demand only

MISC

- Add support for XEP-0328: JID Prep
- Added gsfonts for captcha
- Log Mnesia table type on creation
- Replicate Mnesia 'bosh' table when nodes are joined
- Fix certificate selection for s2s (#3015)
- Provide meaningful error when adding non-local users to shared roster (#3000)
- Websocket: don't treat 'Host' header as a virtual XMPP host (#2989)
- Fix sm ack related c2s error (#2984)
- Don't hide the reason why c2s connection has failed
- Unicode support
- Correctly handle unicode in log messages
- Fix unicode processing in ejabberd.yml

Version 19.08

ADMINISTRATION

- Improve ejabberd halting procedure
- Process unexpected erlang messages uniformly: logging a warning
- mod_configure: Remove modules management

CONFIGURATION

- Use new configuration validator
- ejabberd_http: Use correct virtual host when consulting trusted_proxies
- Fix Elixir modules detection in the configuration file
- Make option 'validate_stream' global
- Allow multiple definitions of host_config and append_host_config
- Introduce option 'captcha_url'
- mod_stream_mgmt: Allow flexible timeout format
- mod_mqtt: Allow flexible timeout format in session_expiry option

MISC

- Fix SQL connections leakage
- New authentication method using JWT tokens
- extauth: Add 'certauth' command
- Improve SQL pool logic
- Add and improve type specs
- Improve extraction of translated strings
- Improve error handling/reporting when loading language translations
- Improve hooks validator and fix bugs related to hooks registration
- Gracefully close inbound s2s connections

- mod_mqtt: Fix usage of TLS

- mod_offline: Make count_offline_messages cache work when using mam for storage

- mod_privacy: Don't attempt to query 'undefined' active list

- mod_privacy: Fix race condition

MUC

- Add code for hibernating inactive muc_room processes

- Improve handling of unexpected iq in mod_muc_room

- Attach mod_muc_room processes to a supervisor

- Restore room when receiving message or generic iq for not started room

- Distribute routing of MUC messages across all CPU cores

PUBSUB

- Fix pending nodes retrieval for SQL backend

- Check access_model when publishing PEP

- Remove deprecated pubsub plugins

- Expose access_model and publish_model in pubsub#metadata

Version 19.05

ADMIN

- The minimum required Erlang/OTP version is now 19.1

- Provide a suggestion when unknown command, module, option or request handler is detected

- Deprecate some listening options: captcha, register, web_admin, http_bind and xmlrpc

- Add commands to get Mnesia info: mnesia_info and mnesia_table_info

- Fix Register command to respect mod_register's Access option

- Fixes in Prosody import: privacy and rooms

- Remove TLS options from the example config

- Improve request_handlers validator

- Fix syntax in example Elixir config file

AUTH

- Correctly support cache tags in ejabberd_auth

- Don't process failed EXTERNAL authentication by mod_fail2ban

- Don't call to mod_register when it's not loaded

- Make anonymous auth don't {de}register user when there are other resources

DEVELOPER

- Rename listening callback from start/2 to start/3

- New hook called when room gets destroyed: room_destroyed

- New hooks for tracking mucsub subscriptions changes: muc_subscribed, muc_unsubscribed

- Make static hooks analyzer working again

MUC

- Service admins are allowed to recreate room even if archive is nonempty

- New option user_mucsub_from_muc_archive

- Avoid late arrival of get_disco_item response

- Handle get_subscribed_rooms call from mod_muc_room pid
- Fix room state cleanup from db on change of persistent option change
- Make get_subscribed_rooms work even for non-persistant rooms
- Allow non-moderator subscribers to get list of room subscribers

OFFLINE

- New option bounce_groupchat: make it not bounce mucsub/groupchat messages
- New option use_mam_for_storage: fetch data from mam instead of spool table
- When applying limit of max msgs in spool check only spool size
- Do not store mucsub wrapped messages with no-store hint in offline storage
- Always store ActivityMarker messages
- Don't issue count/message fetch queries for offline from mam when not needed
- Properly handle infinity as max number of message in mam offline storage
- Sort messages by stanza_id when using mam storage in mod_offline
- Return correct value from count_offline_messages with mam storage option
- Make mod_offline put msg ignored by mam in spool when mam storage is on

SQL:

- Add SQL schemas for MQTT tables
- Report better errors on SQL terms decode failure
- Fix PostgreSQL compatibility in mod_offline_sql:remove_old_messages
- Fix handling of list arguments on pgsql
- Preliminary support for SQL in process_rosteritems command

TESTS

- Add tests for user mucsub mam from muc mam
- Add tests for offline with mam storage
- Add tests for offline use_mam_for_storage
- Initial Docker environment to run ejabberd test suite
- Test offline:use_mam_for_storage, mam:user_mucsub_from_muc_archive used together

WEBSOCKET

- Add WebSockets support to mod_mqtt
- Return "Bad request" error when origin in websocket connection doesn't match
- Fix RFC6454 violation on websocket connection when validating Origin header
- Origin header validation on websocket connection

OTHER MODULES

- mod_adhoc: Use xml:lang from stanza when it's missing in element
- mod_announce: Add 'sessionid' attribute when required
- mod_bosh: Don't put duplicate polling attribute in bosh payload
- mod_http_api: Improve argument error messages and log messages
- mod_http_upload: Feed whole image to eimp:identify/1
- mod_http_upload: Log nicer warning on unknown host
- mod_http_upload: Case-insensitive host comparison
- mod_mqtt: Support other socket modules

- mod_push: Check for payload in encrypted messages

Version 19.02

ADMIN

- Fix in configure.ac the Erlang/OTP version: from 17.5 to 19.0
- reload_config command: Fix crash when sql_pool_size option is used
- reload_config command: Fix crash when SQL is not configured
- rooms_empty_destroy command: Several fixes to behave more conservative
- Fix serverhost->host parameter name for muc_(un)register_nick API

CONFIGURATION

- Allow specifying tag for listener for api_permission purposes
- Change default ciphers to intermediate
- Define default ciphers/protocol_option in example config
- Don't crash on malformed 'modules' section
- mod_mam: New option clear_archive_on_room_destroy to prevent archive removal on room destroy
- mod_mam: New option access_preferences to restrict who can modify the MAM preferences
- mod_muc: New option access_mam to restrict who can modify that room option
- mod_offline: New option store_groupchat to allow storing group chat messages

CORE

- Add MQTT protocol support
- Fix (un)setting of priority
- Use OTP application startup infrastructure for starting dependencies
- Improve starting order of several dependencies

MAM

- mod_mam_mnesia/sql: Improve check for empty archive
- disallow room creation if archive not empty and clear_archive_on_room_destroy is false
- allow check if archive is empty for or user or room
- Additional checks for database failures

MUC

- Make sure that room_destroyed is called even when some code throws in terminate
- Update muc room state after adding extra access field to it
- MUC/Sub: Send mucsub subscriber notification events with from set to room jid

SHARED ROSTER

- Don't perform roster push for non-local contacts
- Handle versioning result when shared roster group has remote account
- Fix SQL queries

MISCELANEA

- CAPTCHA: Add no-store hint to CAPTCHA challenge stanzas
- HTTP: Reject http_api request with malformed Authentication header
- mod_carboncopy: Don't lose carbons on presence change or session resumption

- mod_mix: Fix submission-id and channel resource

- mod_ping: Fix ping IQ reply/timeout processing (17.x regression)

- mod_private: Hardcode item ID for PEP bookmarks

- mod_push: Improve notification error handling

- PIEFXIS: Fix user export when password is scrammed

- Prosody: Improve import of roster items, rooms and attributes

- Translations: fixed "make translations"

- WebAdmin: Fix support to restart module with new options

Version 18.12

- MAM data store compression

- Proxy protocol support

- MUC Self-Ping optimization (XEP-0410)

- Bookmarks conversion (XEP-0411)

Roadmap

ejabberd Roadmap

In the Works

- **Rework ejabberd Docs**

  The ejabberd Docs site needs some reorganization, as some sections has grown quite a lot and would better be reorganized.

Planned

- **Set as default automatic SQL schema update**

  Automatic SQL schema update was included as a beta-testing feature in ejabberd 23.10 (see the feature announcement), and it will become the default.

- **Remove support for Rebar2**

  ejabberd includes many tweaks to support rebar3 and rebar2. By removing support for rebar2, we could simplify rebar.config and other files a lot. But more importantly, dependencies would not need to be updated just because other dependencies are updated: Rebar2 requires exact version numbers to be provided, Rebar3 doesn't require that, and neither does Mix.

Released

2024

- 24.02
- Matrix gateway
- RFC 9266 Channel Bindings for TLS 1.3
- XEP-0386: Bind 2
- XEP-0388: Extensible SASL Profile (SASL2)
- XEP-0424: Message Retraction
- XEP-0440: SASL Channel-Binding Type Capability
- XEP-0474: SASL SCRAM Downgrade Protection
- XEP-0480: SASL Upgrade Tasks
- Automatic SQL schema creation and update
- Commands API versioning
- Support Elixir 1.16 and Erlang/OTP 27.0-rc1

2023

- 23.10
- Support for XEP-0402: PEP Native Bookmarks
- Support for XEP-0421: Occupant Id
- MySQL Performance enhancements

- 23.04
- `mod_mam` support for XEP-0425: Message Moderation
- New `mod_muc_rtbl`: Real-Time Block List for MUC rooms
- Binaries use Erlang/OTP 25.3, and changes in containers
- 23.01
- New `mod_mqtt_bridge`: MQTT bridge

2022

- 22.10
- Improved MIX support
- Improved SQL reconnection Mechanism
- Better burst traffix handling
- 22.05
- Improved MQTT, MUC and ConverseJS integration
- New installers and container image
- Support for Erlang/OTP 25

2021

- 21.12
- New `mod_conversejs`: built-in ConverseJS web client
- Support for MUC Hats extension
- PubSub, MucSub and Multicast improvements
- 21.07
- Improved database and caching for shared rosters
- Broader multicast support for MUC
- Improved rebar3 and Elixir support
- 21.04
- Full support for Erlang/OTP 24 and rebar3
- New API commands
- New CAPTCHA script
- 21.01
- Systemd watchdog support
- STUN improvements

2020

- 20.12
- Extended SCRAM-SHA support
- Microsoft ODBC Driver support
- 20.07
- Support for Unix Domain Sockets
- Erlang/OTP 23 compatibility

- 20.04
- New `mod_stun_disco` : support XEP-0215 including Audio/Video calls
- Improved MS SQL support
- 20.03
- SSL connection to MySQL
- Improved performance of `mod_shared_roster`
- 20.02
- Improved compatibility with CockroachDB
- Emoji storage in MSSQL
- 20.01
- OAuth support for ejabberd's MQTT
- New OTP 22 event logger
- New config parser & validator

2019

- 19.09
- Significant improvements in ACME support: ACME v2
- Erlang/OTP 19.3 is required
- 19.08
- New JWT (JSON Web Token) authentication
- New configuration validator, yconf
- Improved MUC scalability
- Removed Riak support
- 19.05
- MQTT over WebSocket
- Improved MucSub
- Erlang/OTP 19.1 is required
- 19.02
- MQTT Support
- MIX improvements

2018

- 18.12
- XML Compression in message archive storage
- PROXY protocol support versions 1 and 2
- MUC Self-Ping server optimisation (XEP-0410)
- Bookmarks Conversion (XEP-0411)
- 18.09
- Default configuration file simplification
- Improved logging
- OpenSSL 1.1.1 support
- Modular ejabberd core

- 18.06
- Stop ejabberd initialization on invalid/unknown options
- Support SASL PLAIN
- Drop support of mod_irc
- 18.04
- 18.03
- New SQL schemas with server_host
- 18.01

2017

- 17.12
- SNI (Server Name Indication) for inbound connections
- Rewrite ejabberd system monitor
- Support PubSub v1.14 and OMEMO
- 17.11
- ACME Support
- Introduce 'certfiles' global option
- PubSub improved, and SQL storage
- 17.09
- New mod_avatar
- SRV for XMPP over TLS
- 17.08
- XEP-0357: Push Notifications
- Modular cluster with cluster_backend
- 17.07
- 17.06
- New Caching system
- Extended Riak support
- Certificate manager
- 17.04
- 17.03
- Modular code
- Dynamic configuration reload
- mod_blockstrangers for spam protection
- S2S dialback
- 17.01
- PostgreSQL SSL support

2016

- 16.12
- New BOSH module
- New Commands API permissions framework
- XMPP packet handling using dedicated `xmpp` erlang library
- New ejaberd/mix Docker container
- 16.09
- Support for Elixir configuration file
- XEP-0355 Namespace Delegation
- XEP-0356 Privileged Entity
- 16.08
- New MUC/Sub
- Improved Elixir support
- Major clean-up and improvement on OAuth ReST API
- 16.06
- New ACL (Access Control List) infrastructure
- 16.04
- 16.03
- Experimental support for MIX (Mediated Information eXchange)
- Erlang/OTP 17.5 required
- 16.02
- XEP-0013 Flexible Offline Message Retrieval
- Improved Message Archive Management (MAM)
- Published ejabberd on hex.pm
- Faster and more memory efficient XML parsing and TLS encryption.
- Stream compression after SASL
- Migration script from Prosody
- 16.01

2015

- 15.11
- Improved `join_cluster` and `leave_cluster`
- 15.10
- New mod_http_upload with support for XEP-0363 HTTP File Upload
- Added support for Grapherl
- 15.09
- OAuth 2.0 delegation framework
- Preliminary OAuth and HTTP based ejabberd API
- X-AUTH2 authentication mechanism
- 15.07

- 15.06
- New mod_mam with XEP-0313 Message Archive Management
- Configuration checking on launch
- Added Windows 7/8 installers, RPM and DEB packages
- Document protocol support and version inside each module
- 15.04
- Added mod_admin_extra and mod_muc_admin
- Added XEP-0033 Extended Stanza Addressing
- Support to embed ejabberd in an Elixir app
- Erlang/OTP R16B03-1 is required
- 15.03
- Added support for WebSocket
- Customizable session backends
- SCRAM support for SQL authentication backend
- Documentation was converted from LaTeX to Markdown and published in docs.ejabberd.im/
- 15.02
- Added Elixir support
- New command to reload configuration withour restart
- Bug tracker moves from JIRA to GitHub Issues
- Revamped ejabberd website, new logo, new development process and bugtracking migrated from JIRA to GitHub

2014

- 14.12
- New mod_client_state with XEP-0352: Client State Indication
- New mod_fail2ban
- 14.07
- SIP Outbound (RFC 5626)
- 14.05
- RFC-3261 SIP proxy/registrar
- RFC-5766 TURN: Traversal Using Relays around NAT
- XEP-0198 Stream Management
- XEP-0321 Remote Roster Management
- Several improvements regarding encryption
- New Bash completion script for ejabberdctl

2013

- 13.12
- New OpenSSL ciphers option in c2s, s2s and s2s_out
- ejabberd_xmlrpc included

- 13.10
- ejabberd configuration file in YAML format
- Log files are created using Lager
- Rebar2 is used to manage dependencies
- Erlang/OTP R15 is required
- 13.03-beta1 (announcement)
- Binarize and indent code
- New versioning scheme

2012

- 2.1.11
- Added ODBC support for several modules

2011

- 2.1.10
- 2.1.9
- New SASL SCRAM-SHA-1 authentication mechanism

2010

- 2.1.6
- mod_register: New captcha_protected option to require CAPTCHA
- Support PostgreSQL 9.0
- October: the source code repository and the bug tracker were finally moved to GitHub
- 2.1.5
- 2.1.4
- Full support for XEP-0115 Entity Capabilities v1.5
- 2.1.2

2009

- 2.1.1
- 2.1.0
- LDAPS support
- STUN server
- New XEPs supported: XMPP Ping, Roster Versioning, Import/Export Format
- Erlang/OTP R13 is supported
- 2.0.5 (announcement)
- 2.0.4 (announcement)
- 2.0.3 (announcement)

2008

- 2.0.2 (announcement)
- 2.0.1 (announcement)
- 2.0.0 (announcement)
- New front-end and back-end cluster architecture
- Complete rewrite of the PubSub module
- New Proxy65 file transfer proxy
- BOSH support
- Many more improvements

2007

- 1.1.4
- 1.1.3

2006

- 1.1.2 (announcement)
- LDAP improvements
- Microsoft SQL supported
- New Windows installer
- 1.1.1 (announcement)
- Erlang/OTP R9C-2 required
- 1.1.0 (announcement)
- JEP-0050: Ad-Hoc Commands
- JEP-0138: Stream Compression
- JEP-0175: SASL anonymous
- Native MySQL support
- MUC improvement: Chatroom logging

2005

- 1.0.0 (announcement)
- S2S encryption: STARTTLS + SASL_EXTERNAL and STARTTLS + Dialback
- Different certificates can be defined for each virtual host.
- Support for vCard storage in ODBC
- New tool to convert Mnesia to ODBC
- Native PostgreSQL support
- 0.9.8 (announcement)
- Enhanced virtual hosting
- Enhanced PubSub
- 0.9.1 (announcement)

- 0.9 (announcement)

- Added support for virtual hosts

- New mod_shared_roster

- Added PostgreSQL support

- February: source code moved from SVN to Git, and the bug tracker from Bugzilla to JIRA

- Beginning of 2005, source code moved from JabberStudio CVS to ProcessOne SVN

2004

- October: website moved from JabberStudio to ejabberd.jabber.ru, and the bug tracker to Jabber.ru's Bugzilla

- 0.7.5

- Support for STARTTLS with C2S connections

- Support for authentification via external script

- Added module which implement JUD and vCard services using LDAP

- Improvements in web-based administration interface (user creation/removal, roster and offline queue management)

- Support for message expiration (JEP-0023)

- Support for HTTPS in web interface

- 0.7

- Support for LDAP authentification

- Support for HTTP Polling

- Support for web-based administration interface

- Added command-line administration utility "ejabberdctl"

- Support for history management in MUC rooms

2003

- 16th November, 0.5

- First release

- January, initial documentation in LaTeX: Ejabberd Installation and Operation Guide

2002

- 18th November, first commit to CVS

- 16th November, first erlang modules written