ejabberd Docs

ejabberd Community Server

Copyright © 2008 - 2024 ProcessOne

Table of contents

Overview 5

- Getting started 👋 5
- Features 8
- Frequently Asked Questions 10
- ejabberd Use Cases 12
- GNU GENERAL PUBLIC LICENSE 15
- Security Policy 19
- Readme 20

Install 22

- Installation 22
- Install ejabberd using a Container Image 23
- `ejabberd` Container Image 24
- `ecs` Container Image 33
- Binary Installers 40
- Operating System Packages 42
- Install ejabberd from Source Code 43
- Install ejabberd on macOS 51
- Installing ejabberd development environment on OSX 52
- Next Steps 55

Configure 57

- Configuring ejabberd 57
- File format 58
- Basic Configuration 60
- Authentication 70
- Database Configuration 75
- LDAP Configuration 80
- Listen Modules 90
- Listen Options 98
- Top-Level Options 105
- Modules Options 133

Advanced 0

- Advanced ejabberd Administration 0
- Architecture 0
- Clustering 0
- Managing an ejabberd server 0

Copyright © 2008 - 2024 ProcessOne

- Add More Modules 0
- Securing ejabberd 0
- Troubleshooting ejabberd 0
- Upgrade Procedure for ejabberd 0
- ejabberd and XMPP tutorials 0
- Getting started with MIX 0
- MQTT Support 0
- Setting vCards / Avatars for MUC rooms 0
- Using ejabberd with MySQL 0

Development 0

- ejabberd for Developers 0
- ejabberd Developer Guide 0
- PubSub overview 0
- Roster versioning 0
- ejabberd Stanza Routing 0
- ejabberd SQL Database Schema 0
- External authentication 0
- Main contribution repository 0
- ejabberd API libraries 0
- Old / obsolete contributions 0
- Contributing to ejabberd 0
- Contributor Covenant Code of Conduct 0
- Contributors 0
- Understanding ejabberd and its dependencies 0
- ejabberd Docs Source Code 0
- ejabberd for Elixir Developers 0
- The ejabberd Developer Livebook 0
- Internationalization and Localization 0
- ejabberd Modules Development 0
- MucSub: Multi-User Chat Subscriptions 0
- ejabberd Test Suites 0
- Developing ejabberd with VSCode 0
- Getting Started with XMPPFramework 0

API 0

- ejabberd Rest API 0
- API Reference 0
- API Tags 0
- Simple ejabberd Rest API Configuration 0

Copyright © 2008 - 2024 ProcessOne

API Permissions 0
OAuth Support 0
ejabberd commands 0
API Versioning 0
Archive 0
ChangeLog 0
Roadmap 0
ejabberd Roadmap 0

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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

Copyright © 2008 - 2024 ProcessOne

- 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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

Security Policy

Supported Versions

We recommend that all users always use the latest version of ejabberd.

To ensure the best experience and security, upgrade to the latest version available on this repo.

Reporting a Vulnerability

Private Reporting

Preferred Method: Use GitHub's private vulnerability reporting system by clicking the "Report a Vulnerability" button in the Security tab of this repository. This ensures your report is securely transmitted and tracked.

Alternative: If you cannot use the GitHub system, send an email to `contact@process-one.net` with the following details:

- A clear description of the vulnerability.
- Steps to reproduce the issue.
- Any potential impact or exploitation scenarios.

Response Time

We aim to acknowledge receipt of your report within 72 hours. You can expect regular updates on the status of your report.

Resolution

If the vulnerability is confirmed, we will work on a patch or mitigation strategy.
We will notify you once the issue is resolved and coordinate a public disclosure if needed.

Acknowledgements

We value and appreciate the contributions of security researchers and community members.
If you wish, we are happy to acknowledge your efforts publicly by listing your name (or alias) below in this document.
Please let us know if you would like to be recognized when reporting the vulnerability.

Public Discussion

For general inquiries or discussions about the project's security, feel free to chat with us here:

- XMPP room: `ejabberd@conference.process-one.net`
- GitHub Discussions

However, please note that if the issue is **critical** or potentially exploitable, **do not share it publicly**. Instead, we strongly recommend you contact the maintainers directly via the private reporting methods outlined above to ensure a secure and timely response.

Thank you for helping us improve the security of ejabberd!

Copyright © 2008 - 2024 ProcessOne

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 Page or GitHub Releases for releases.

GitHub Actions for master branch (`run` / `deb` / `rpm` for `x64` and `arm64`)

Docker Containers:

`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

More info can be found in the `Installation` part of ejabberd Docs.

Documentation

Please check the ejabberd Docs website.

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

Once ejabberd is installed, try:

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.

Security

For information on how to report security vulnerabilities, please refer to the SECURITY.md file. It contains guidelines on how to report vulnerabilities privately and securely, ensuring that any issues are addressed in a timely and confidential manner.

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)
- ejabberd translations under **MIT License**.

Copyright © 2008 - 2024 ProcessOne

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

Copyright © 2008 - 2024 ProcessOne

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
```


Copyright © 2008 - 2024 ProcessOne

`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:

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

Stop the running container:

Restart the stopped ejabberd container:

Start with Erlang console attached

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

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:

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.

Copyright © 2008 - 2024 ProcessOne

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
```

Macros in environment

ejabberd reads `EJABBERD_MACRO_*` environment variables and uses them to define the `*` macros, overwriting the corresponding macro definition if it was set in the configuration file. This is supported since ejabberd 24.12.

For example, if you configure this in `ejabberd.yml` :

```
acl:
  admin:
    user: ADMINJID
```


Copyright © 2008 - 2024 ProcessOne

now you can define the admin account JID using an environment variable:

```
    environment:
      - EJABBERD_MACRO_ADMINJID=admin@localhost
```

Check the full example for other example.

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 \
    .
```


Copyright © 2008 - 2024 ProcessOne

Podman build

To build the image using Podman, please 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.

If using Docker, write this `docker-compose.yml` file and start it with `docker-compose up`:

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

If using Podman, write this `minimal.yml` file and start it with `podman kube play minimal.yml`:

```
apiVersion: v1

kind: Pod

metadata:
  name: ejabberd

spec:
  containers:

  - name: ejabberd
    image: ghcr.io/processone/ejabberd
```


Copyright © 2008 - 2024 ProcessOne

```
    ports:
    - containerPort: 5222
      hostPort: 5222
    - containerPort: 5269
      hostPort: 5269
    - containerPort: 5280
      hostPort: 5280
    - containerPort: 5443
      hostPort: 5443
```

Customized Example

This example shows the usage of several customizations: it uses a local configuration file, defines a configuration macro using an environment variable, 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
```

Use a macro in `ejabberd.yml` to set the served vhost, with `localhost` as default value:

```
define_macro:
  XMPPHOST: localhost

hosts:
  - XMPPHOST
```

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

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

If using Docker, write this `docker-compose.yml` file and start it with `docker-compose up`:

```
version: '3.7'

services:

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

If using Podman, write this `custom.yml` file and start it with `podman kube play custom.yml`:

```
apiVersion: v1

kind: Pod

metadata:
  name: ejabberd

spec:
  containers:

  - name: ejabberd
    image: ghcr.io/processone/ejabberd
    env:
    - name: CTL_ON_CREATE
      value: register admin example.com asd
    - name: CTL_ON_START
      value: registered_users example.com ;
             status
    ports:
    - containerPort: 5222
      hostPort: 5222
    - containerPort: 5269
```


Copyright © 2008 - 2024 ProcessOne

```
      hostPort: 5269
    - containerPort: 5280
      hostPort: 5280
    - containerPort: 5443
      hostPort: 5443
    volumeMounts:
    - mountPath: /opt/ejabberd/conf/ejabberd.yml
      name: config
      readOnly: true
    - mountPath: /opt/ejabberd/database
      name: db

  volumes:
  - name: config
    hostPath:
      path: ./ejabberd.yml
      type: File
  - name: db
    hostPath:
      path: ./database
      type: DirectoryOrCreate
```

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.

If using Docker, write this `docker-compose.yml` file and start it with `docker-compose up`:

```
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
    environment:
      - ERLANG_NODE_ARG=ejabberd@replica
      - ERLANG_COOKIE=dummycookie123
      - CTL_ON_CREATE=join_cluster ejabberd@main
      - CTL_ON_START=registered_users localhost ;
                     status
    ports:
      - "5222:5222"
      - "5269:5269"
      - "5280:5280"
      - "5443:5443"
```

If using Podman, write this `cluster.yml` file and start it with `podman kube play cluster.yml`:

```
apiVersion: v1

kind: Pod

metadata:
  name: cluster

spec:
  containers:

  - name: first
    image: ghcr.io/processone/ejabberd
    env:
    - name: ERLANG_NODE_ARG
      value: main@cluster
    - name: ERLANG_COOKIE
      value: dummycookie123
```


Copyright © 2008 - 2024 ProcessOne

```
    - name: CTL_ON_CREATE
      value: register admin localhost asd
    - name: CTL_ON_START
      value: stats registeredusers ;
             status
    - name: EJABBERD_MACRO_PORT_C2S
      value: 6222
    - name: EJABBERD_MACRO_PORT_C2S_TLS
      value: 6223
    - name: EJABBERD_MACRO_PORT_S2S
      value: 6269
    - name: EJABBERD_MACRO_PORT_HTTP_TLS
      value: 6443
    - name: EJABBERD_MACRO_PORT_HTTP
      value: 6280
    - name: EJABBERD_MACRO_PORT_MQTT
      value: 6883
    - name: EJABBERD_MACRO_PORT_PROXY65
      value: 6777
    volumeMounts:
    - mountPath: /opt/ejabberd/conf/ejabberd.yml
      name: config
      readOnly: true

  - name: second
    image: ghcr.io/processone/ejabberd
    env:
    - name: ERLANG_NODE_ARG
      value: replica@cluster
    - name: ERLANG_COOKIE
      value: dummycookie123
    - name: CTL_ON_CREATE
      value: join_cluster main@cluster ;
             started ;
             list_cluster
    - name: CTL_ON_START
      value: stats registeredusers ;
             check_password admin localhost asd ;
             status
    ports:
    - containerPort: 5222
      hostPort: 5222
    - containerPort: 5280
      hostPort: 5280
    volumeMounts:
    - mountPath: /opt/ejabberd/conf/ejabberd.yml
      name: config
      readOnly: true

  volumes:
  - name: config
    hostPath:
      path: ./conf/ejabberd.yml
      type: File
```

Your configuration file should use those macros to allow each ejabberd node use different listening port numbers:

```
diff --git a/ejabberd.yml.example b/ejabberd.yml.example
index 39e423a64..6e875b48f 100644
--- a/ejabberd.yml.example
+++ b/ejabberd.yml.example
@@ -24,9 +24,19 @@ loglevel: info
 #  - /etc/letsencrypt/live/domain.tld/fullchain.pem
 #  - /etc/letsencrypt/live/domain.tld/privkey.pem

+define_macro:
+  PORT_C2S: 5222
+  PORT_C2S_TLS: 5223
+  PORT_S2S: 5269
+  PORT_HTTP_TLS: 5443
+  PORT_HTTP: 5280
+  PORT_STUN: 5478
+  PORT_MQTT: 1883
+  PORT_PROXY65: 7777
+
 listen:
   -
-    port: 5222
+    port: PORT_C2S
     ip: "::"
     module: ejabberd_c2s
     max_stanza_size: 262144
@@ -34,7 +44,7 @@ listen:
     access: c2s
     starttls_required: true
   -
-    port: 5223
+    port: PORT_C2S_TLS
     ip: "::"
     module: ejabberd_c2s
     max_stanza_size: 262144
@@ -42,13 +52,13 @@ listen:
```


Copyright © 2008 - 2024 ProcessOne

```
     access: c2s
     tls: true
   -
-    port: 5269
+    port: PORT_S2S
     ip: "::"
     module: ejabberd_s2s_in
     max_stanza_size: 524288
     shaper: s2s_shaper
   -
-    port: 5443
+    port: PORT_HTTP_TLS
     ip: "::"
     module: ejabberd_http
     tls: true
@@ -60,14 +70,14 @@ listen:
       /upload: mod_http_upload
       /ws: ejabberd_http_ws
   -
-    port: 5280
+    port: PORT_HTTP
     ip: "::"
     module: ejabberd_http
     request_handlers:
       /admin: ejabberd_web_admin
       /.well-known/acme-challenge: ejabberd_acme
   -
-    port: 5478
+    port: PORT_STUN
     ip: "::"
     transport: udp
     module: ejabberd_stun
@@ -77,7 +87,7 @@ listen:
     ## The server's public IPv6 address:
     # turn_ipv6_address: "2001:db8::3"
   -
-    port: 1883
+    port: PORT_MQTT
     ip: "::"
     module: mod_mqtt
     backlog: 1000
@@ -207,6 +217,7 @@ modules:
   mod_proxy65:
     access: local
     max_connections: 5
+    port: PORT_PROXY65
   mod_pubsub:
     access_createnode: pubsub_createnode
     plugins:
```


Copyright © 2008 - 2024 ProcessOne

`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:

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:

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

Start with Erlang console attached

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

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:

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:

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 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 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
```


Copyright © 2008 - 2024 ProcessOne

```
      - ::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 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
```


Copyright © 2008 - 2024 ProcessOne

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 ejabberdctl started
docker exec -it $OLDCONTAINER ejabberdctl register user1 localhost somepass
docker exec -it $OLDCONTAINER ejabberdctl registered_users localhost
```

Methods to know the Erlang node name:

```
ls database/ | grep ejabberd@
docker exec -it $OLDCONTAINER 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
```


Copyright © 2008 - 2024 ProcessOne

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 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 ejabberdctl mnesia_change_nodename $OLDNODE $NEWNODE $OLDFILE $NEWFILE
```

6. Install the backup file as a fallback:

```
docker exec -it $NEWCONTAINER 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 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 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.

Copyright © 2008 - 2024 ProcessOne

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
```


Copyright © 2008 - 2024 ProcessOne

```
    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
```


Copyright © 2008 - 2024 ProcessOne

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/`.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

Install ejabberd from Source Code

The canonical distribution form 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 26.2, 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 package from ProcessOne Downloads or GitHub Releases
- Latest development code from ejabberd Git repository using the commands:

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

Compile

The generic instructions to compile ejabberd are:

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

Let's view them in detail.

Copyright © 2008 - 2024 ProcessOne

`./configure`

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

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

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):

```
./configure --enable-user=ejabberd --enable-mysql
```

If you get `Error loading module rebar3`, please consult how to use rebar with old Erlang.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

`make`

This gets the erlang depencies and compiles everything, among 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 task list:

```
make help
```

Note: The required erlang dependencies are downloaded from Internet. Or you can copy `$HOME/.hex/` package cache from another machine.

Install

There are several ways to install and run ejabberd after it's compiled from source code:

- system install
- system install a release
- building a production release
- building a development release
- don't install at all, just start with `make relive`

System Install

To install ejabberd in the destination directories, run:

```
make install
```

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

Copyright © 2008 - 2024 ProcessOne

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

System Install Release

added in 24.02

This builds a production release, and then performs a system install of that release, obtaining a result similar to the one mentioned in the previous section.

Simply run:

```
make install-rel
```

The benefits of `install-rel` over `install`:

- this uses OTP release code from rebar/rebar3/mix, and consequently requires less code in our `Makefile.in` file
- `uninstall-rel` correctly deletes all the library files
- the `*.beam` files are smaller as debug information is stripped

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.

Copyright © 2008 - 2024 ProcessOne

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.

Erlang Configuration

Usually ejabberd is started using the `ejabberdctl` script, which takes care to setup the Erlang runtime system options following your `ejabberdctl.cfg` configuration file.

Copyright © 2008 - 2024 ProcessOne

However, there are other methods to start ejabberd, and they use specific methods to configure those options:

- When using rebar3 `make relive`:
  system is set in `rebar.config`, apps in `rel/relive.config`
- When using mix `make relive`:
  system is set in `Makefile`, apps in `config/runtime.exs`
- When using rebar3 `make dev && ejabberd`:
  system is set in `rel/vm.args`, apps in `rel/sys.config`
- When using mix `make dev|prod && ejabberd`:
  system is set in `rel/vm.args.eex` and `rel/env.sh.eex`, apps in `config/runtime.exs`

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

man

ejabberd includes a man page which documents the toplevel and modules options, the same information that is published in the Top-Level Options and Modules Options sections.

The man file can be read locally with:

```
man -l man/ejabberd.yml.5
```

rebar with old Erlang

The ejabberd source code package includes `rebar` and `rebar3` binaries that work with Erlang/OTP 24.0 up to 27.

To compile ejabberd using rebar/rebar3 and Erlang 20.0 up to 23.3, you can install it from your operating system, or compile yourself from the rebar source code, or download the old binary from ejabberd 21.12:

```
wget https://github.com/processone/ejabberd/raw/21.12/rebar
wget https://github.com/processone/ejabberd/raw/21.12/rebar3
```


Copyright © 2008 - 2024 ProcessOne

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
```


Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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
```


Copyright © 2008 - 2024 ProcessOne

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:

Copyright © 2008 - 2024 ProcessOne

```
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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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/ on binary installers
- https://localhost:5443/admin/ on binary installers
- https://localhost:5280/admin/ on Debian package

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

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).

Copyright © 2008 - 2024 ProcessOne

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
```


Copyright © 2008 - 2024 ProcessOne

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: ""
```


Copyright © 2008 - 2024 ProcessOne

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:

Copyright © 2008 - 2024 ProcessOne

`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
```


Copyright © 2008 - 2024 ProcessOne

```
    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
```


Copyright © 2008 - 2024 ProcessOne

```
|       └── 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
```

`ACLName` is any name you may want to give, and later can use it in other options to reference that ACL. The following `ACLName` are pre-defined:

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

Copyright © 2008 - 2024 ProcessOne

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

Copyright © 2008 - 2024 ProcessOne

- **`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: peter@example.org
    - user:
        yozhik@example.org
    - user: {susan: example.org}
    - user:
        jan: 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}
    - shared_group:
        secteam: 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\\."
```


Copyright © 2008 - 2024 ProcessOne

- `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.

Access Rules

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

```
access_rules:
  AccessName:
    - allow|deny: ACLName|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: ACLName` that matches when the ACL named `ACLName` matches). If no ACL name or definition is defined, the ACL `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:
```


Copyright © 2008 - 2024 ProcessOne

```
    - 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: ACLName|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: ACLName|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
```


Copyright © 2008 - 2024 ProcessOne

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
```


Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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

Copyright © 2008 - 2024 ProcessOne

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.

If compiling ejabberd from source code, PAM support is disabled by default, so you have to enable PAM support when configuring the ejabberd compilation: `./configure --enable-pam`

Options:

- pam_service
- pam_userinfotype

Example:

```
auth_method: [pam]
pam_service: ejabberd
```


Copyright © 2008 - 2024 ProcessOne

Though it is quite easy to set up PAM support in ejabberd, there are several problems that you may need to solve:

- 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.
- ejabberd binary installers include `epam` pointing to module paths that may not work in your system. If authentication doesn't work correctly, check if syslog (example: `journalctl -t epam -f`) reports errors like `PAM unable to dlopen(/home/runner/... No such file or directory`. In that case, create a PAM configuration file (example: `/etc/pam.d/ejabberd`) and provide the real path to that file in your machine:

```
#%PAM-1.0
auth        sufficient  /usr/lib/x86_64-linux-gnu/security/pam_unix.so audit
account     sufficient  /usr/lib/x86_64-linux-gnu/security/pam_unix.so audit
```

- `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 (in Debian it would be `/etc/pam.d/ejabberd`). 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]
```


Copyright © 2008 - 2024 ProcessOne

```
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.

Copyright © 2008 - 2024 ProcessOne

Database Configuration

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.

Supported storages

The following databases are supported by ejabberd:

- `Mnesia`. Used by default, nothing to setup to start using it
- `MySQL`. Check the tutorial Using ejabberd with MySQL
- `PostgreSQL`
- `MS SQL Server/SQL Azure`. Check the Microsoft SQL Server section
- `SQLite`
- Any `ODBC` compatible database
- `Redis` (only for transient data). Check the Redis section
- `LDAP` is documented in the LDAP section

Virtual Hosting

If you define several host names in the `ejabberd.yml` configuration file, probably you 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 the host_config top-level option for each virtual host).

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 your configuration by setting the default database with the `default_db` top-level option:

- it sets the default authentication method when the `auth_method` top-level option is not configured
- it defines the database to use in ejabberd modules that support the `db_type` option, when that option is not configured.

Database Schema

updated in 24.06

Copyright © 2008 - 2024 ProcessOne

The update_sql_schema top-level option allows ejabberd to create and update the tables automatically in the SQL database when using MySQL, PostgreSQL or SQLite. That option was added in ejabberd 23.10, and enabled by default in 24.06. If you can use that feature:

1. Create the database in your SQL server
2. Create an account in the SQL server and grant it rights in the database
3. Configure in ejabberd the SQL Options that allow it to connect
4. Start ejabberd ...
5. and it will take care to create the tables (or update them if they exist from a previous ejabberd version)

If that option is disabled, or you are using a different SQL database, or an older ejabberd release, then you must create the tables in the database manually before starting ejabberd. The SQL database schema files are available:

- If installing ejabberd from sources, sql files are in the installation directory. By default: `/usr/local/lib/ejabberd/priv/sql`
- If installing ejabberd from Process-One installer, sql files are in the ejabberd's installation path under `<base>/lib/ejabberd*/priv/sql`

See ejabberd SQL Database Schema for details on database schemas.

Default and *New* Schemas

If using MySQL, PostgreSQL, Microsoft SQL or SQLite, you can choose between two database schemas:

- 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
```

When creating the tables, if ejabberd can use the update_sql_schema top-level option as explained in the Database Schema section, it will take care to create the tables with the correct schema.

On the other hand, if you are creating the tables manually, 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
```


Copyright © 2008 - 2024 ProcessOne

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, see section SQL with SSL connection
- 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, see section Database Schema
- new_sql_schema, see section Default and New Schemas

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 with SSL Connection

The sql_ssl top-level option allows SSL encrypted connections to MySQL, PostgreSQL, and Microsoft SQL servers.

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 Authentication

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

Copyright © 2008 - 2024 ProcessOne

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

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.

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.

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.

Copyright © 2008 - 2024 ProcessOne

The available top-level options are:

- redis_server
- redis_port
- redis_password
- redis_db
- redis_connect_timeout

Example configuration:

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


Copyright © 2008 - 2024 ProcessOne

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_backups
- ldap_encrypt
- ldap_tls_verify
- ldap_tls_certfile
- 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: "**********"
```

When there are several LDAP servers available as backup, set one in `ldap_servers` and the others in `ldap_backups`. At server start, ejabberd connects to all the servers listed in `ldap_servers`. If a connection is lost, ejabberd connects to the next server in `ldap_backups`. If the connection is lost, the next server in the list is connected, and this repeats infinitely with all the servers in `ldap_servers` and `ldap_backups` until one is successfully connected:

```
ldap_servers:
  - ldap1.example.org
```


Copyright © 2008 - 2024 ProcessOne

```
ldap_backups:
  - ldap2.example.org
  - ldap3.example.org
```

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
```


Copyright © 2008 - 2024 ProcessOne

```
    ## 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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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:

A. 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`.

B. 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
```


Copyright © 2008 - 2024 ProcessOne

```
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
```

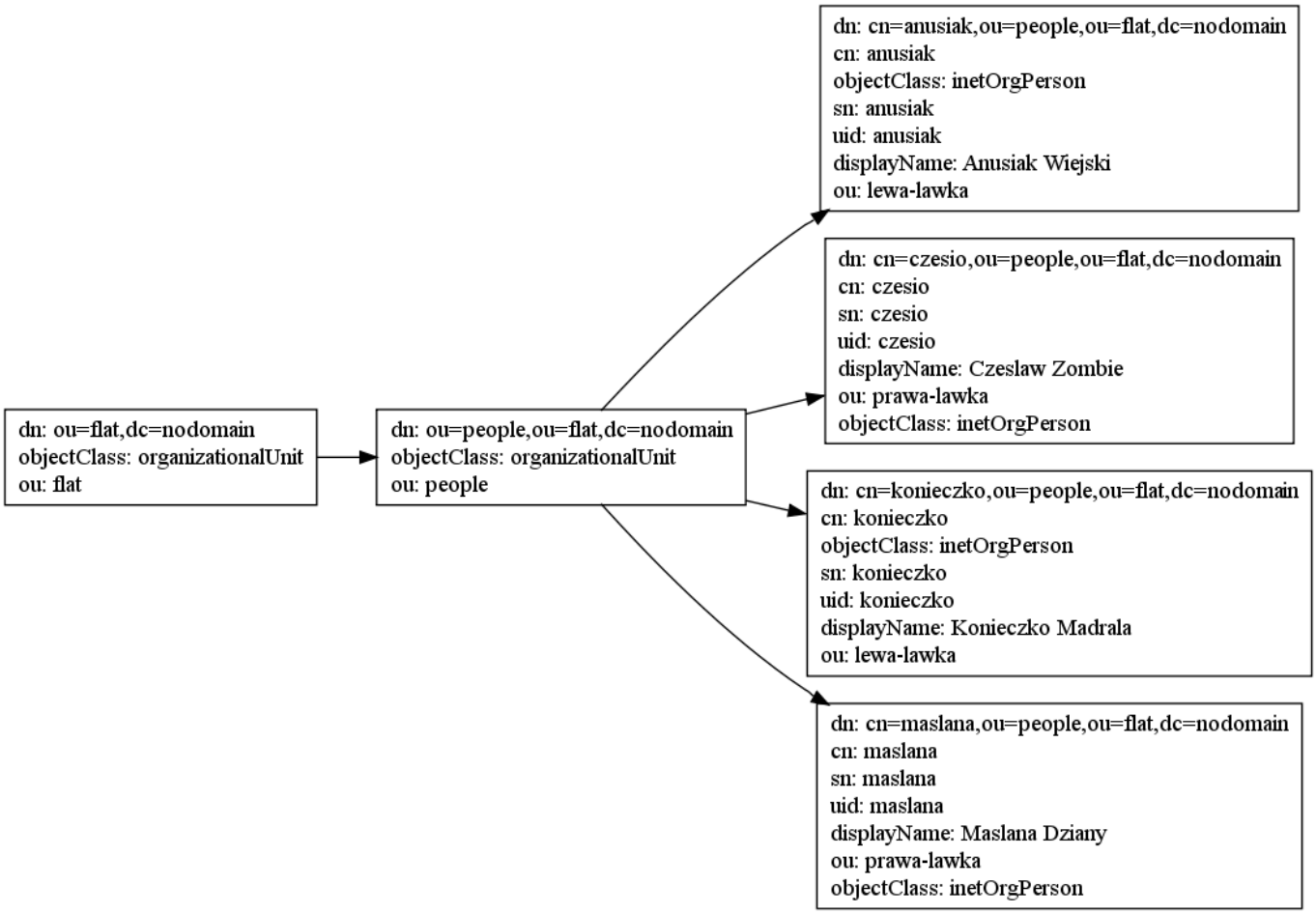
Configuration examples

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:

Copyright © 2008 - 2024 ProcessOne



Such layout has a few downsides, including:

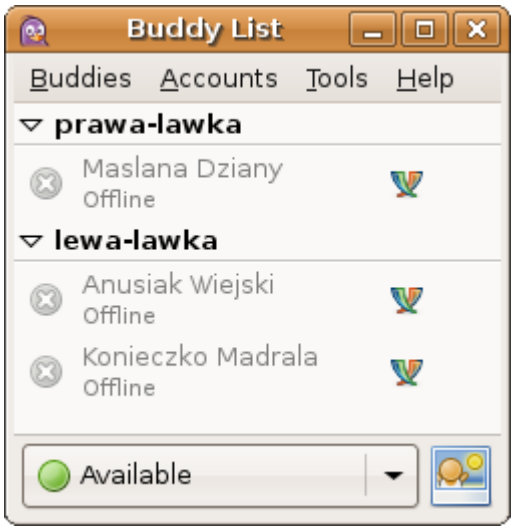
- information duplication – the group name is repeated in every member object
- difficult group management – information about group members is not centralized, but distributed between member objects
- inefficiency – the list of unique group names has to be computed by iterating over all users

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
```

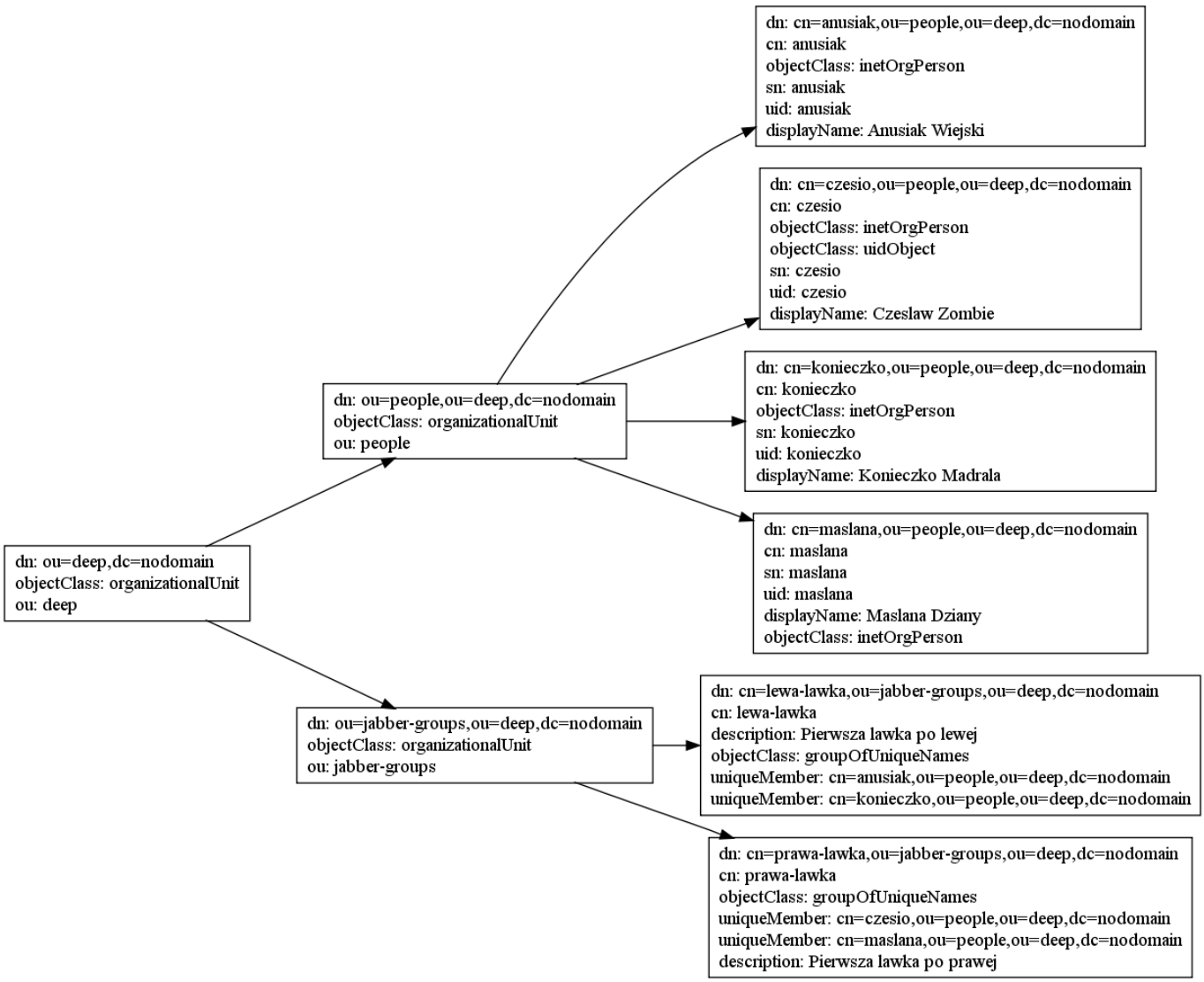
…to be provided with a roster upon connecting as user `czesio`, as shown in this figure:

Copyright © 2008 - 2024 ProcessOne



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.



Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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
```


Copyright © 2008 - 2024 ProcessOne

Listen Modules

Please note

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 Option

The listen top-level 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 of listen options:

- **port** and **module** mandatory options
- **ip**, **transport**, **unix_socket** additional options
- other options for listening module, enumerated 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, allow_unencrypted_sasl2, 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.

Copyright © 2008 - 2024 ProcessOne

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):

Copyright © 2008 - 2024 ProcessOne

```
listen:
  -
    port: 5478
    transport: udp
    module: ejabberd_stun
  -
    port: 5478
    module: ejabberd_stun
  -
    port: 5349
    module: ejabberd_stun
    tls: true
    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: 5478
    transport: udp
    use_turn: true
    turn_ipv4_address: 10.20.30.1
    module: ejabberd_stun
```

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.
```


Copyright © 2008 - 2024 ProcessOne

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:
```


Copyright © 2008 - 2024 ProcessOne

```
      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.

Just for reference, there's also the old `ejabberd_xmlrpc documentation` with example clients in other languages.

Copyright © 2008 - 2024 ProcessOne

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 5478 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: 5478
    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]
```


Copyright © 2008 - 2024 ProcessOne

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:
```


Copyright © 2008 - 2024 ProcessOne

```
      /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>
```


Copyright © 2008 - 2024 ProcessOne

Listen Options

> **Please note**
>
> 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`.

allow_unencrypted_sasl2

true | *false*

As per `XEP-0388`, ejabberd rejects SASL2 negotiations over non-TLS connections by default. Setting this option to `true` allows SASL2 over plaintext connections, which may be useful in case TLS is terminated by some proxy in front of ejabberd.

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*

Copyright © 2008 - 2024 ProcessOne

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

Copyright © 2008 - 2024 ProcessOne

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).

ip

string()

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`

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.

module

ModuleName

Mandatory option to define what listening module will serve the incoming connections to the port.

Copyright © 2008 - 2024 ProcessOne

password

Secret

Specify the password to verify an external component that connects to the port.

port

pos_integer() | string()

improved in 20.07

This mandatory option defines which port to listen for incoming connections: it can be a Jabber/XMPP standard port or any other valid port number between `1` and `65535` to listen on TCP or UDP socket,

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`.

Tip

improved in 25.xx

If it's a relative path, then it's created in the mnesia spool directory. For example, if set to `"unix:dir/file.socket"`, then the socket file is created in `/opt/ejabberd/database/dir/file.socket`, or whatever path the Mnesia database is stored in your installation.

File permissions can be set using the unix_socket option.

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

Copyright © 2008 - 2024 ProcessOne

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

Copyright © 2008 - 2024 ProcessOne

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.

transport

tcp|udp

Defines the transport protocol. Default is `tcp`.

unix_socket

{mode|owner|group: Value}

added in 23.10

Set the mode, owner and group of the unix domain socket defined in the port option.

The owner and group must be specified as integers, not as names.

Example:

```
listen:
  -
    port: "unix:sockets/ctl_over_http.socket"
    unix_socket:
      mode: '0600'
      owner: 117
      group: 135
```


Copyright © 2008 - 2024 ProcessOne

```
    module: ejabberd_http
    request_handlers:
      /ctl: ejabberd_ctl
    tag: "ctl_over_http"
```

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.

Copyright © 2008 - 2024 ProcessOne

Top-Level Options

Please note

This section describes top level options of ejabberd 24.12. 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: ACLName|ACLDefinition}}`

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}}`

This 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

Copyright © 2008 - 2024 ProcessOne

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`

Copyright © 2008 - 2024 ProcessOne

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`.

Copyright © 2008 - 2024 ProcessOne

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}`

Add a few specific options to a certain virtual host.

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, ...]`

Copyright © 2008 - 2024 ProcessOne

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:

Copyright © 2008 - 2024 ProcessOne

```
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

Copyright © 2008 - 2024 ProcessOne

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. The keyword *@VERSION@* is replaced with ejabberd version number in `XX.YY` format. The keyword `@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`.

Copyright © 2008 - 2024 ProcessOne

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 database to store persistent data in 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:

Copyright © 2008 - 2024 ProcessOne

```
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**: `Value`
  How to deliver stanzas to connected components. The default value is `random`. Possible values:

  - 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

  - destination
  an instance is chosen by the full JID of the packet's `to` attribute

  - random
  an instance is chosen at random

  - source
  by the full JID of the packet's `from` attribute

  Example:

```
domain_balancing:
  component.domain.tld:
    type: destination
    component_number: 5
  transport.example.org:
    type: bare_source
```


Copyright © 2008 - 2024 ProcessOne

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 in external auth, 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 auth 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.

Copyright © 2008 - 2024 ProcessOne

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, ...]`

List of one or more host names (or domains) that ejabberd will serve. This is a **mandatory** option.

include_config_file

[Filename, ...] | {Filename: Options}

Read and include additional file 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`

Copyright © 2008 - 2024 ProcessOne

This ACL rule defines accounts that can use only the JWT 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 JWT key. The default value is `undefined`.

language

`Language`

Define 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 (see LDAP connection). When no servers listed in ldap_servers option are reachable, ejabberd connects to these backup servers. The default is an empty list, i.e. no backup servers specified. Please notice that ejabberd only connects to the next server when the existing connection is lost; it doesn't detect when a previously-attempted server becomes available again.

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

Copyright © 2008 - 2024 ProcessOne

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 (see LDAP connection). ejabberd connects immediately to all of them, and reconnects infinitely if connection is lost. 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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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 10 Mb expressed in bytes: `10485760`.

loglevel

`none | emergency | alert | critical | error | warning | notice | info | debug`

Verbosity of ejabberd logging. 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}`

Set all the modules configuration options.

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.

Copyright © 2008 - 2024 ProcessOne

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 the new SQL schema. All schemas are located at https://github.com/processone/ejabberd/tree/24.12/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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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`.

Copyright © 2008 - 2024 ProcessOne

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`.

Copyright © 2008 - 2024 ProcessOne

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 🟤

`Host | IP Address | Unix Socket Path`

improved in 24.12

A hostname, IP address or unix domain socket file of the Redis server. Setup the path to unix domain socket like: `"unix:/path/to/socket"`. The default value 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.

Copyright © 2008 - 2024 ProcessOne

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`

Copyright © 2008 - 2024 ProcessOne

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`

Copyright © 2008 - 2024 ProcessOne

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: ACLName|ACLDefinition}}`

This option defines shaper rules 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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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 | IP Address | ODBC Connection String | Unix Socket Path`

Copyright © 2008 - 2024 ProcessOne

improved in 24.06

The hostname or IP address of the SQL server. For sql_type `mssql` or `odbc` this can also be an ODBC connection string. When sql_type is `mysql` or `pgsql`, this can be the path to a unix domain socket expressed like: `"unix:/path/to/socket"`.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`.

Copyright © 2008 - 2024 ProcessOne

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`

updated in 24.06

Allow ejabberd to update SQL schema in MySQL, PostgreSQL and SQLite databases. This option was added in ejabberd 23.10, and enabled by default since 24.06. The default value is `true`.

update_sql_schema_timeout

`timeout()`

added in 24.07

Time allocated to SQL schema update queries. The default value is set to 5 minutes.

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

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

Modules Options

Please note

This section describes modules options of ejabberd 24.12. 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 API: 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.

push_roster API (and push_roster_all API): 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 API: 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 push_roster API:

```
[{<<"bob">>, <<"example.org">>, <<"workers">>, <<"Bob">>},
{<<"mart">>, <<"example.org">>, <<"workers">>, <<"Mart">>},
{<<"Rich">>, <<"example.org">>, <<"bosses">>, <<"Rich">>}].
```


Copyright © 2008 - 2024 ProcessOne

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 API 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).

Copyright © 2008 - 2024 ProcessOne

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_auth_fast 🟤

added in 24.12

The module adds support for XEP-0484: Fast Authentication Streamlining Tokens that allows users to authenticate using self-managed tokens.

Available options:

- **db_type**: `mnesia`
  Same as top-level default_db option, but applied to this module only.
- **token_lifetime**: `timeout()`
  Time that tokens will be kept, measured from it's creation time. Default value set to 30 days
- **token_refresh_age**: `timeout()`
  This time determines age of token, that qualifies for automatic refresh. Default value set to 1 day

Example:

```
modules:
  mod_auth_fast:
    token_lifetime: 14days
```

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.

Copyright © 2008 - 2024 ProcessOne

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,

Copyright © 2008 - 2024 ProcessOne

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).

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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 are enabled in at least one `request_handlers`.

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 `@HOST@` keyword 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
```


Copyright © 2008 - 2024 ProcessOne

```
    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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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`

Copyright © 2008 - 2024 ProcessOne

Available options:

- **default_version* 🟤 **: integer() | string()*
  added in 24.12 What API version to use when none is specified in the URL path. If setting an ejabberd version, it will use the latest API version that was available in that ejabberd version. For example, setting `"24.06"` in this option implies `2`. The default value is the latest version.

Example:

```
listen:
  -
    port: 5280
    module: ejabberd_http
    request_handlers:
      /api: mod_http_api

modules:
  mod_http_api:
    default_version: 2
```

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:

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

Available options:

Copyright © 2008 - 2024 ProcessOne

- **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: Implementation. 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 this module, 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. The default value is `"HTTP File Upload"`. Please note this will only be displayed by some XMPP clients.
- **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.

Copyright © 2008 - 2024 ProcessOne

- **thumbnail**: `true | false`
  This option specifies whether ejabberd should create thumbnails of uploaded images. If a thumbnail is created, a <thumbnail/> element that contains the download <uri/> 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 also option `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:

Copyright © 2008 - 2024 ProcessOne

Notice 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.

Copyright © 2008 - 2024 ProcessOne

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. Erlang/OTP 25 or higher is required to use this module.

Copyright © 2008 - 2024 ProcessOne

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 `true`, 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 `false`, 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: "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
    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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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: {Key: Value}}`
  Declaration of data to share for each ServerUrl. Server URLs can use schemas: `mqtt`, `mqtts` (mqtt with tls), `mqtt5`, `mqtt5s` (both to trigger v5 protocol), `ws`, `wss`, `ws5`, `wss5`. Keys must be:
- **authentication**: `{AuthKey: AuthValue}`
  List of authentication information, where AuthKey can be: `username` and `password` fields, or `certfile` pointing to client certificate. Certificate authentication can be used only with mqtts, mqtt5s, wss, wss5.
- **publish**: `{LocalTopic: RemoteTopic}`
  Either publish or subscribe must be set, or both.
- **subscribe**: `{RemoteTopic: LocalTopic}`
  Either publish or subscribe must be set, or both.

Example:

```
modules:
  mod_mqtt_bridge:
    replication_user: "mqtt@xmpp.server.com"
    servers:
      "mqtt://server.com":
        authentication:
          certfile: "/etc/ejabberd/mqtt_server.pem"
        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
```

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.

Copyright © 2008 - 2024 ProcessOne

Available options:

Copyright © 2008 - 2024 ProcessOne

- **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.

Copyright © 2008 - 2024 ProcessOne

- **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:

Copyright © 2008 - 2024 ProcessOne

- **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`.

Copyright © 2008 - 2024 ProcessOne

- **persistent**: `true | false`
  The room persists even if the last participant leaves. The default value is `false`.
- **presence_broadcast**: `[Role]`
  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`.

Copyright © 2008 - 2024 ProcessOne

- **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`.

Copyright © 2008 - 2024 ProcessOne

- **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 API. 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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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:

Copyright © 2008 - 2024 ProcessOne

```
# 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.

The delete_expired_messages API allows to delete expired messages, and delete_old_messages API deletes older ones.

Copyright © 2008 - 2024 ProcessOne

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. By enabling the option, this module uses the `archive` table from mod_mam 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
  ...
```


Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

Note

Nowadays modern XMPP clients rely on XEP-0191: Blocking Command which is implemented by mod_blocking. 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 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

improved in 24.10

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.

Copyright © 2008 - 2024 ProcessOne

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:

- **iq**: `{Namespace: Options}`
  This option defines namespaces and their IQ permissions. By default no permissions are given. The `Options` are:
- **both**: `AccessName`
  Allows sending IQ stanzas of type `get` and `set`. The default value is `none`.
- **get**: `AccessName`
  Allows sending IQ stanzas of type `get`. The default value is `none`.
- **set**: `AccessName`
  Allows sending IQ stanzas of type `set`. The default value is `none`.
- **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:
    iq:
      http://jabber.org/protocol/pubsub:
        get: all
    roster:
      get: all
    presence:
```


Copyright © 2008 - 2024 ProcessOne

```
      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.

Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

Available options:

Copyright © 2008 - 2024 ProcessOne

- **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:

Copyright © 2008 - 2024 ProcessOne

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
```


Copyright © 2008 - 2024 ProcessOne

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`.

Copyright © 2008 - 2024 ProcessOne

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. If `AccessName` is `none`, then registering new accounts using In-Band Registration is disabled and the corresponding stream feature is not announced to clients.
- **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`.

  Example:

```
modules:
  mod_register:
    welcome_message:
      subject: "Welcome!"
      body: |-
        Hi!
        Welcome to this XMPP server
```


Copyright © 2008 - 2024 ProcessOne

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.

Copyright © 2008 - 2024 ProcessOne

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_bidi

added in 24.10

The module adds support for XEP-0288: Bidirectional Server-to-Server Connections that allows using single s2s connection to communicate in both directions.

The module has no options.

Example:

```
modules:
  mod_s2s_bidi: {}
```

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).

Copyright © 2008 - 2024 ProcessOne