

processone/ejabberd

- ☐

Overview

Overview

- Features
- FAQ
- Use Cases
- License
- Security
- Glossary
- Quickstart

- ☒

Install

Install

- ☐ Containers Containers

- Start ejabberd
 - daemon
 - with Erlang console
 - with your data
 - Next steps
 - Register admin account
 - Check ejabberd log
 - Inspect container files
 - Open debug console
 - CAPTCHA
 - Advanced
 - Ports
 - Volumes
 - Commands on start
 - Macros in environment
 - ejabberd-contrib
 - ejabberdapi
 - Clustering
 - Change Mnesia Node Name
 - Build Container Image
 - Build ejabberd
 - Build ecs
 - Composer Examples
 - Minimal Example
 - Customized Example
 - Clustering Example
 - Images Comparison
- Binary Installers
- Operating System Package
- Compile Source Code
- Homebrew
- Mac OSX
- Next Steps

- ☐

Configure

Configure

- File Format
- Basic Configuration
- Authentication
- Databases
- LDAP
- Listen Modules
- Listen Options
- Top-Level Options
- Modules Options

- ☐

Advanced

Advanced

- Architecture
- Clustering
- Managing
- Modules / Contrib
- Security
- Troubleshooting

- Upgrade
- Tutorials
- MIX tutorial
- MQTT tutorial
- MUC Hats
- MUC vCards
- MySQL tutorial
- ☐
 - Development
 - Development
 - Developer Guide
 - Pubsub Dev
 - Simplified Roster Versioning
 - Stanza Routing
 - SQL Schema
 - Contributions
 - Contributing
 - Contributor Covenant
 - Contributors
 - Dependencies
 - Docs
 - Elixir Dev
 - Livebook
 - Localization
 - Modules Development
 - MUC/Sub Extension
 - Testing
 - VSCode
 - XMPPFramework (iOS)
- ☐
 - API
 - API
 - API Reference
 - API Tags
 - Simple Configuration
 - Permissions
 - OAuth Support
 - Commands
 - Versioning
- ☐
 - Archive
 - Archive
 - ☐
 - 25.07
 - 25.07
 - API Reference
 - API Tags
 - Listen Modules
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 25.07
 - ☐
 - 25.04

- 25.04
 - API Reference
 - API Tags
 - Listen Modules
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 25.04
- ☐
- 25.03
- 25.03
 - API Reference
 - API Tags
 - Listen Modules
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 25.03
- ☐
- 24.12
- 24.12
 - API Reference
 - API Tags
 - Listen Modules
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 24.12
- ☐
- 24.10
- 24.10
 - API Reference
 - API Tags
 - Listen Modules
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 24.10
- ☐
- 24.07
- 24.07
 - API Reference
 - API Tags
 - Listen Modules
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 24.07
- ☐
- 24.06
- 24.06
 - API Reference
 - API Tags
 - Listen Modules

- Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 24.06
- ☐
- 24.02
- 24.02
 - API Reference
 - API Tags
 - Listen Modules
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 24.02
- ☐
- 23.10
- 23.10
 - API Reference
 - API Tags
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 23.10
- ☐
- 23.04
- 23.04
 - API Reference
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 23.04
- ☐
- 23.01
- 23.01
 - API Reference
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 23.01
- ☐
- 22.10
- 22.10
 - API Reference
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 22.10
- ☐
- 22.05
- 22.05

- API Reference
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 22.05
- ☐
 - 21.12
 - 21.12
 - API Reference
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 21.12
- ☐
 - 21.07
 - 21.07
 - API Reference
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 21.07
- ☐
 - 21.04
 - 21.04
 - API Reference
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
 - Upgrade to ejabberd 21.04
- ☐
 - 21.01
 - 21.01
 - API Reference
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
- ☐
 - 20.12
 - 20.12
 - API Reference
 - Listen Option
 - Listen Options
 - Modules Options
 - Top-Level Options
- ☐
 - 20.07
 - 20.07
 - API Reference
 - Listen

- Listen Options
 - Modules Options
 - Top-Level Options
- ☐
- 20.04
- 20.04
 - API Reference
 - Listen Modules
 - Listen Options
 - Modules Options
 - Top-Level Options
- ☐
- 20.03
- 20.03
- ☐
- 20.02
- 20.02
- ☐
- 20.01
- 20.01
 - Upgrade to ejabberd 20.01
- ☐ Older releases
- Older releases
 - Upgrade to ejabberd 19.08
 - Upgrade to ejabberd 19.05
 - Upgrade to ejabberd 19.02
 - Upgrade to ejabberd 18.12
 - Upgrade to ejabberd 18.09
 - Upgrade to ejabberd 18.06
 - Upgrade to ejabberd 18.04
 - Upgrade to ejabberd 18.03
 - Upgrade to ejabberd 18.01
 - Upgrade to ejabberd 17.11
 - Upgrade to ejabberd 17.09
 - Upgrade to ejabberd 17.06
 - Upgrade to ejabberd 17.03
 - Upgrade to ejabberd 16.08
 - Upgrade to ejabberd 16.06
 - Upgrade to ejabberd 16.04
 - Upgrade to ejabberd 16.03
 - Upgrade to ejabberd 16.02
 - Upgrade from 2.1.1x to 16.02
- Configure 20.03
- ChangeLog
- ☐
- Roadmap
- Roadmap
- Start ejabberd
 - daemon
 - with Erlang console
 - with your data
- Next steps
 - Register admin account
 - Check ejabberd log



- Inspect container files
- Open debug console
- CAPTCHA
- Advanced
 - Ports
 - Volumes
 - Commands on start
 - Macros in environment
 - ejabberd-contrib
 - ejabberdapi
 - Clustering
 - Change Mnesia Node Name
- Build Container Image
 - Build ejabberd
 - Build ecs
- Composer Examples
 - Minimal Example
 - Customized Example
 - Clustering Example
- Images Comparison



ejabberd Container Images¶

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 page documents those container images (images comparison):

-  ejabberd
published in ghcr.io/processone/ejabberd, built using ejabberd repository, both for stable ejabberd releases and the master branch, in x64 and arm64 architectures.
-  ecs
published in docker.io/ejabberd/ecs, built using docker-ejabberd/ecs repository for ejabberd stable releases in x64 architectures.

For Microsoft Windows, see Docker Desktop for Windows 10, and Docker Toolbox for Windows 7.

For Kubernetes Helm, see [help-ejabberd](#).

Start ejabberd¶

daemon¶

Start ejabberd in a new container:

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

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

Restart the stopped ejabberd container:

```
docker restart ejabberd
```

Stop the running container:


```
docker stop ejabberd
```

Remove the ejabberd container:

```
docker rm ejabberd
```

with Erlang console¶

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

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

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

with your data¶

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

```
mkdir conf && cp ejabberd.yml.example conf/ejabberd.yml
```

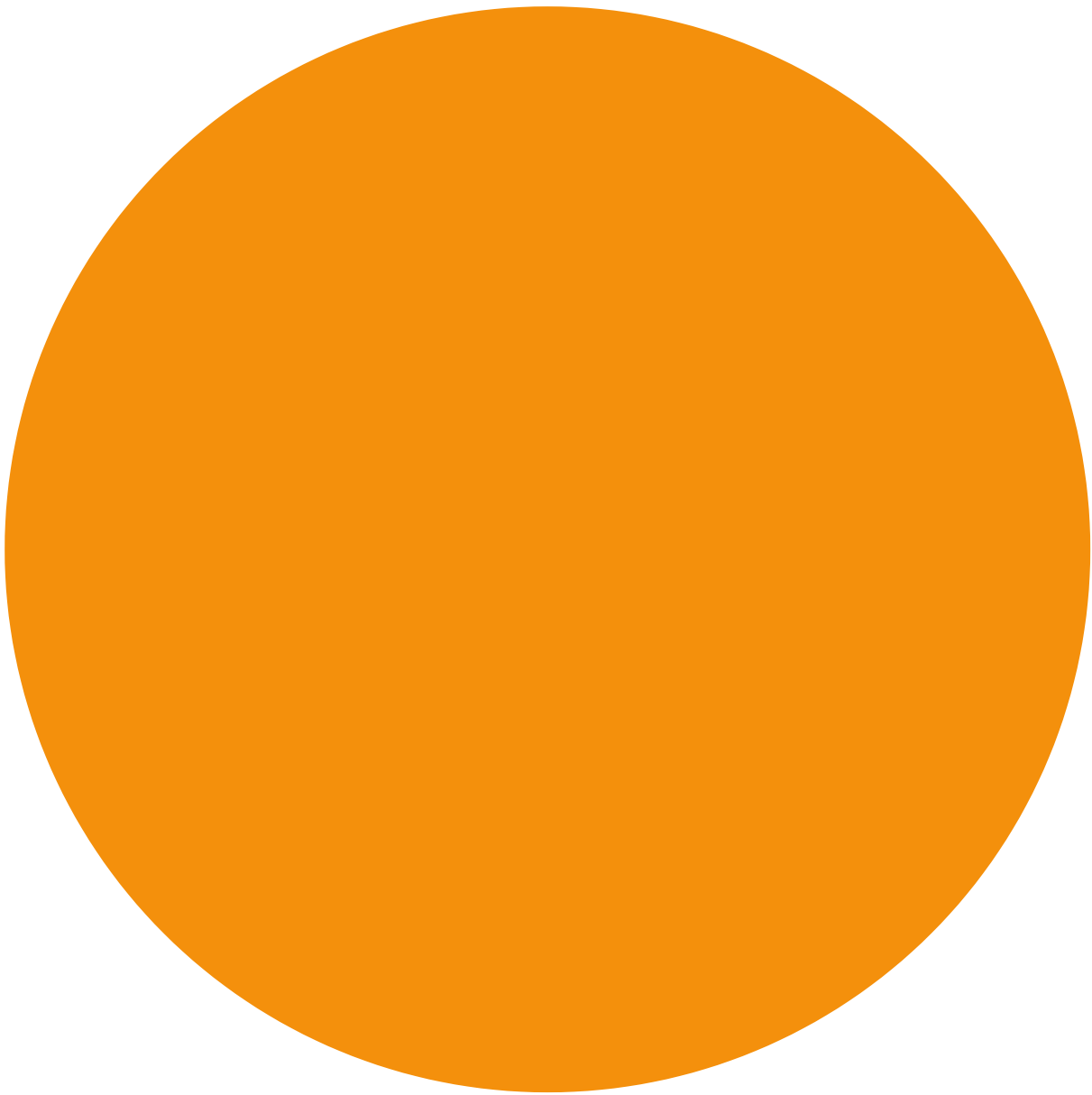
```
mkdir database && chown ejabberd database
```

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

Notice that ejabberd runs in the container with an account named `ejabberd` with UID 9000 and group `ejabberd` with GID 9000, and the volumes you mount must grant proper rights to that account.

Next steps¶

Register admin account¶



¶

If you set the `REGISTER_ADMIN_PASSWORD` environment variable, an account is automatically registered with that password, and admin privileges are granted to it. The account created depends on what variables you have set:

- `EJABBERD_MACRO_ADMIN=juliet@example.org -> juliet@example.org`
- `EJABBERD_MACRO_HOST=example.org -> admin@example.org`
- None of those variables are set -> `admin@localhost`

The account registration is shown in the container log:

```
> ejabberdctl register admin example.org somePassw0rd
User admin@example.org successfully registered
```

Alternatively, you can register the account manually yourself and edit `conf/ejabberd.yml` and add the ACL as explained in [ejabberd Docs: Administration 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 it, for example:

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

Check ejabberd log¶

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 container files¶

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

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

Open 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: "$HOME/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.

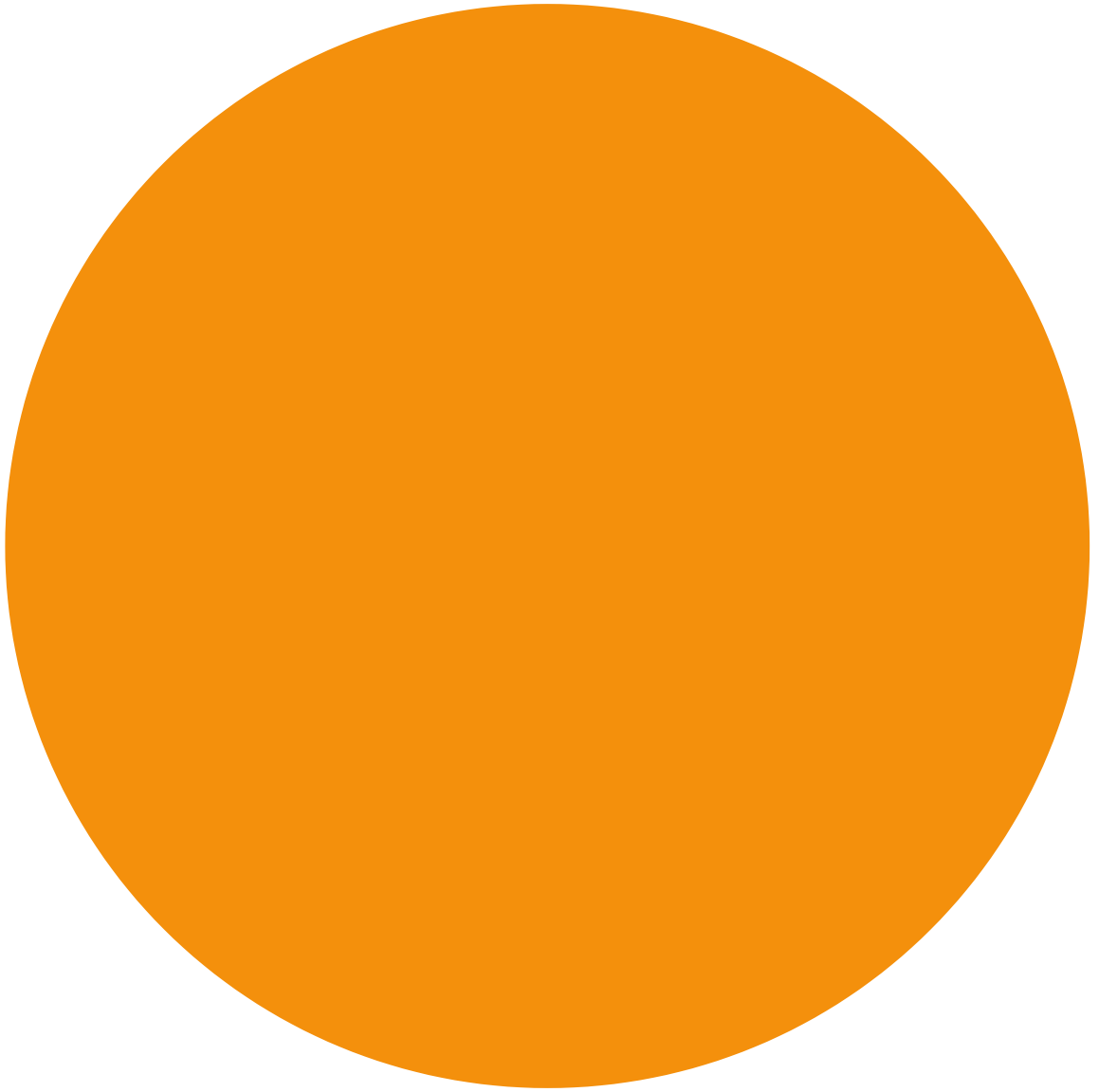
Advanced¶

Ports¶

The container image exposes several ports (check also [Docs: Firewall Settings](#)):

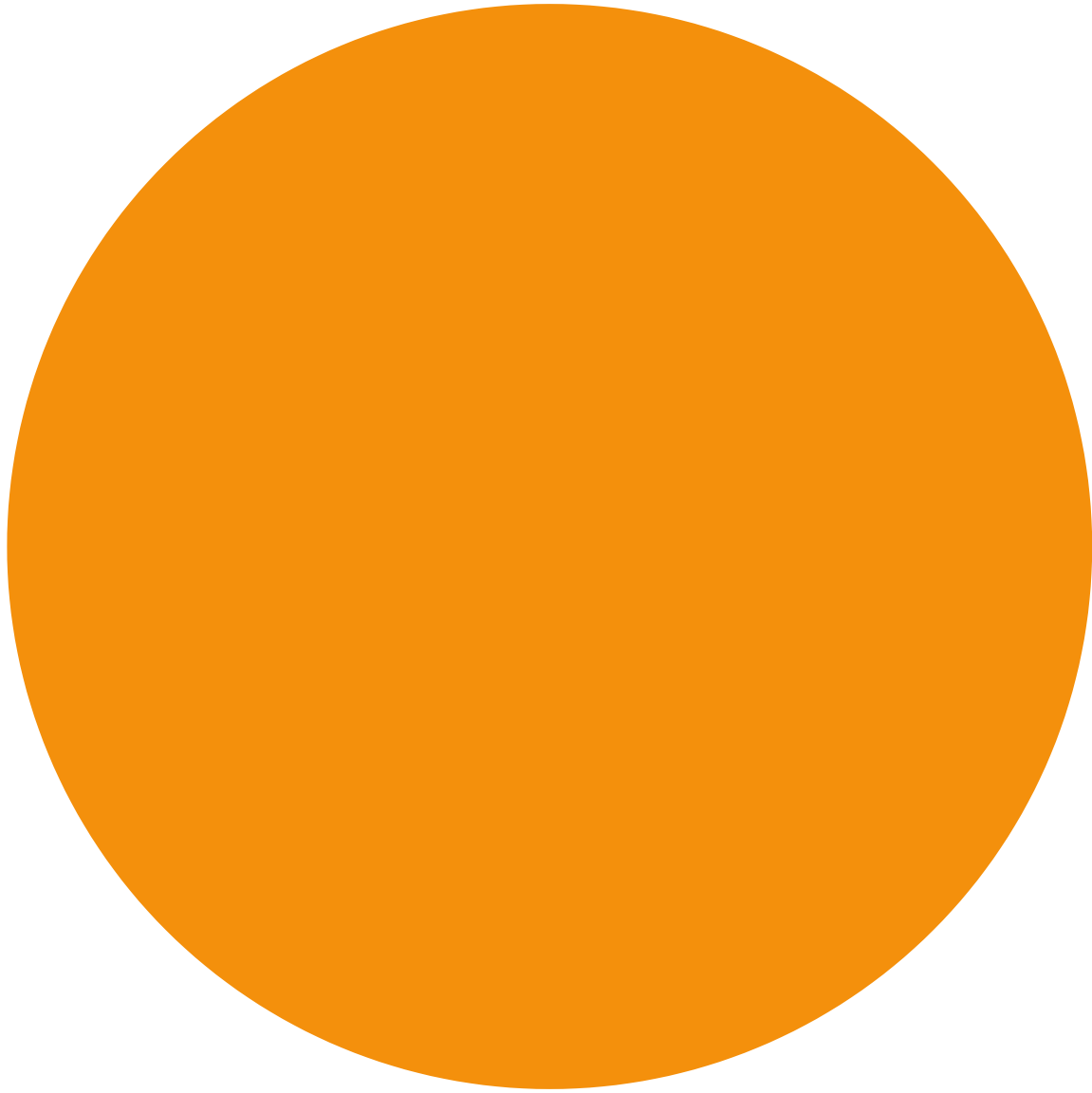
- 5222: The default port for XMPP clients.
- 5269: For XMPP federation. Only needed if you want to communicate with users on other servers.
- 5280: For admin interface (URL is admin/).

- 1880: For admin interface (URL is /, useful for podman-desktop and docker-desktop)



- 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 an account named ejabberd with group ejabberd in the container. Its 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.

If using Docker, try:

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

If using Podman, try:

```
mkdir database  
podman unshare chown 9000:9000 database
```

It's possible to install additional ejabberd modules using volumes, check [this Docs tutorial](#).

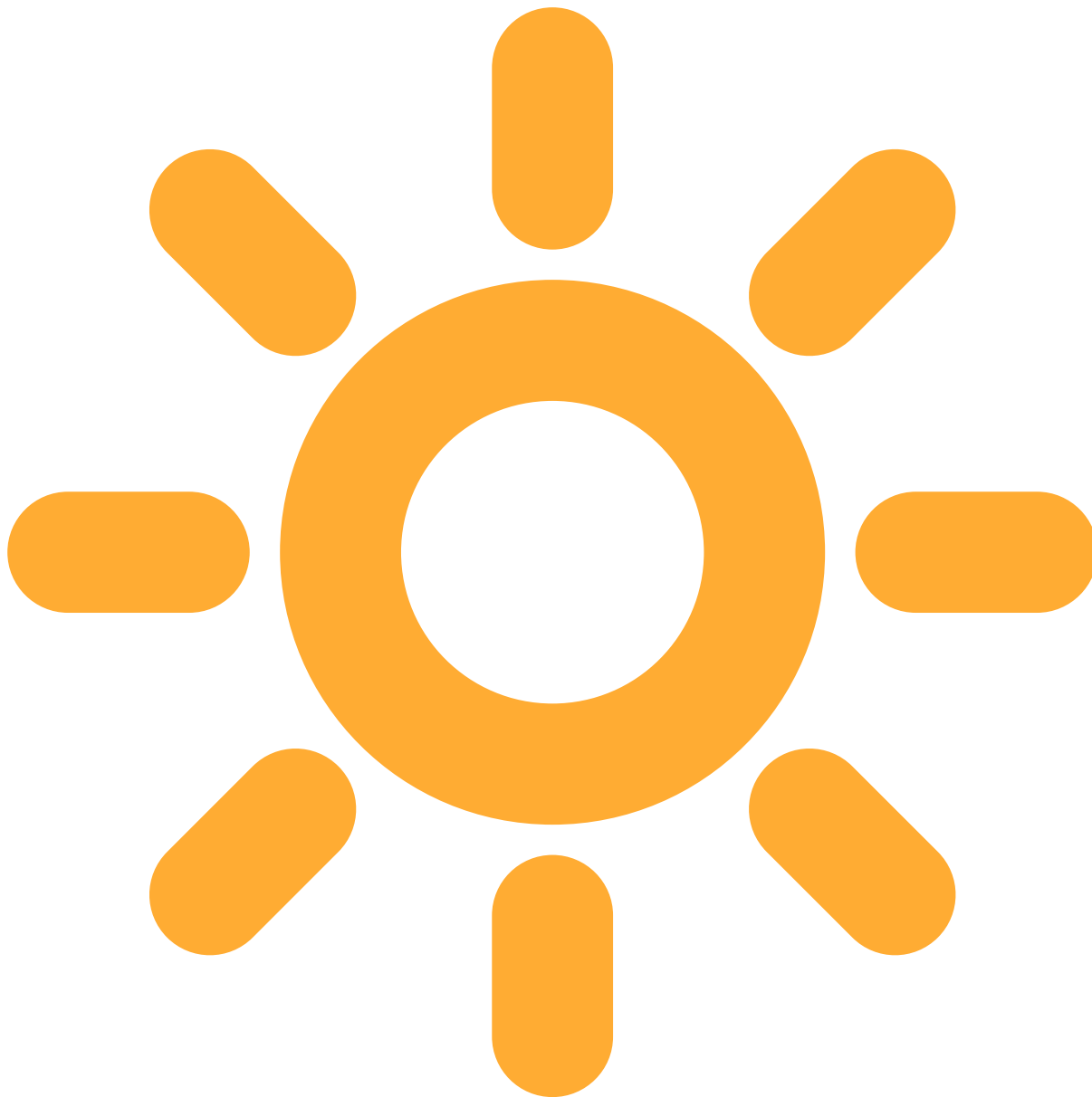
Commands on start¶

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.

If any of those commands returns a failure, the container starting gets aborted. If there is a command with a result that can be ignored, prefix that command with !

This example, registers an admin@localhost account when the container is first created. Everytime the container starts, it shows the list of registered accounts, checks that the admin account exists and password is valid, changes the password of an account if it exists (ignoring any failure), and shows the ejabberd starts (check also the [full example](#)):

```
environment:  
- CTL_ON_CREATE=register admin localhost asd  
- CTL_ON_START=stats registeredusers ;  
                check_password admin localhost asd ;  
                ! change_password bot123 localhost qqq ;  
                status
```



ejabberd reads `EJABBERD_MACRO_*` environment variables and uses them to define the corresponding 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: ADMIN
```

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

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

Check the [full example](#) for other example.

[ejabberd-contrib](#)

This section addresses those topics related to [ejabberd-contrib](#):

- [Download source code](#)

- Install a module
 - Install git for dependencies
 - Install your module
-

Download source code¶

The ejabberd container image includes the ejabberd-contrib git repository source code, but ecs does not, so first download it:

```
$ docker exec ejabberd ejabberdctl modules_update_specs
```

Install a module¶

Compile and install any of the contributed modules, for example:

```
docker exec ejabberd ejabberdctl module_install mod_statsdx
```

Module mod_statsdx has been installed and started.

It's configured in the file:

```
/opt/ejabberd/.ejabberd-modules/mod_statsdx/conf/mod_statsdx.yml
```

Configure the module in that file, or remove it
and configure in your main ejabberd.yml

Install git for dependencies¶

Some modules depend on erlang libraries, but the container images do not include git or mix to download them. Consequently, when you attempt to install such a module, there will be error messages like:

```
docker exec ejabberd ejabberdctl module_install ejabberd_observer_cli
```

I'll download "recon" using git because I can't use Mix to fetch from hex.pm:

```
/bin/sh: mix: not found
```

Fetching dependency observer_cli:

```
/bin/sh: git: not found
```

...

the solution is to install git in the container image:

```
docker exec --user root ejabberd apk add git
```

```
fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/main/x86_64/APKINDEX.tar.gz
```

```
fetch https://dl-cdn.alpinelinux.org/alpine/v3.21/community/x86_64/APKINDEX.tar.gz
```

```
(1/3) Installing pcre2 (10.43-r0)
```

```
(2/3) Installing git (2.47.2-r0)
```

```
(3/3) Installing git-init-template (2.47.2-r0)
```

```
Executing busybox-1.37.0-r12.trigger
```

```
OK: 27 MiB in 42 packages
```

and now you can upgrade the module:

```
docker exec ejabberd ejabberdctl module_upgrade ejabberd_observer_cli
```

I'll download "recon" using git because I can't use Mix to fetch from hex.pm:

```
/bin/sh: mix: not found
```

Fetching dependency observer_cli: Cloning into 'observer_cli'...

Fetching dependency os_stats: Cloning into 'os_stats'...

Fetching dependency recon: Cloning into 'recon'...


```
Inlining: inline_size=24 inline_effort=150
```

```
Old inliner: threshold=0 functions=[{insert,2},{merge,2}]
```

```
Module ejabberd_observer_cli has been installed.
```

```
Now you can configure it in your ejabberd.yml
```

```
I'll download "recon" using git because I can't use Mix to fetch from hex.pm:
```

```
/bin/sh: mix: not found
```

Install your module¶

If you developed an ejabberd module, you can install it in your container image:

1. Create a local directory for ejabberd-modules:

```
mkdir docker-modules
```

2. Then create the directory structure for your custom module:

```
cd docker-modules
```

```
mkdir -p sources/mod_hello_world/
```

```
touch sources/mod_hello_world/mod_hello_world.spec
```

```
mkdir sources/mod_hello_world/src/
```

```
mv mod_hello_world.erl sources/mod_hello_world/src/
```

```
mkdir sources/mod_hello_world/conf/
```

```
echo -e "modules:\n  mod_hello_world: {}" > sources/mod_hello_world/conf/mod_hello_world.yml
```

```
cd ..
```

3. Grant ownership of that directory to the UID that ejabberd will use inside the Docker image:

```
sudo chown 9000 -R docker-modules/
```

4. Start ejabberd in the container:

```
sudo docker run \  
  --name hellotest \  
  -d \  
  --volume "$(pwd)/docker-modules:/home/ejabberd/.ejabberd-modules/" \  
  -p 5222:5222 \  
  -p 5280:5280 \  
  ejabberd/ecs
```

5. Check the module is available for installing, and then install it:

```
sudo docker exec -it hellotest ejabberdctl modules_available  
mod_hello_world []
```

```
sudo docker exec -it hellotest ejabberdctl module_install mod_hello_world
```

6. If the module works correctly, you will see Hello in the ejabberd logs when it starts:

```
sudo docker exec -it hellotest grep Hello logs/ejabberd.log  
2020-10-06 13:40:13.154335+00:00 [info]  
<0.492.0>@mod_hello_world:start/2:15 Hello, ejabberd world!
```

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, the container image includes the `ejabberdapi` executable. Please check the [ejabberd-api homepage](#) for configuration and usage details.

For example, if you configure ejabberd like this:

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

acl:
  loopback:
    ip:
      - 127.0.0.0/8
      - ::1/128
      - ::FFFF:127.0.0.1/128

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

Then you could register new accounts with this query:

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

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
```

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:/opt/ejabberd/database \
  ghcr.io/processone/ejabberd: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=/opt/ejabberd/database/old.backup  
NEWFILE=/opt/ejabberd/database/new.backup
```

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

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

```
docker exec -it $OLDCONTAINER 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:/opt/ejabberd/database \  
  ghcr.io/processone/ejabberd: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 \  
  ghcr.io/processone/ejabberd:latest
```

```
-v $(pwd)/database:/opt/ejabberd/database \
-e ERLANG_NODE_ARG=$OLDNODE \
ghcr.io/processone/ejabberd: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](#).

Build Container Image

The container image includes ejabberd as a standalone OTP release built using Elixir.

Build ejabberd

The ejabberd Erlang/OTP release is configured with:

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

Direct build

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

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

Podman build

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
```

Build ecs

The ejabberd Erlang/OTP release is configured 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 personal/ejabberd .
```

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.

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
    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.

Prepare an ejabberd configuration file:

```
mkdir conf && cp ejabberd.yml.example conf/ejabberd.yml
```

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

- Docker:

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

- Podman:

```
mkdir database && podman unshare 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_HOST=example.com
- EJABBERD_MACRO_ADMIN=admin@example.com
- REGISTER_ADMIN_PASSWORD=somePassw0rd
- CTL_ON_START=registered_users example.com ;
status

```
    ports:
```

- "5222:5222"
- "5269:5269"
- "5280:5280"
- "5443:5443"

```
    volumes:
```

- ./conf/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: EJABBERD_MACRO_HOST
  value: example.com
- name: EJABBERD_MACRO_ADMIN
  value: admin@example.com
- name: REGISTER_ADMIN_PASSWORD
  value: somePassw0rd
- name: CTL_ON_START
  value: registered_users example.com ;
      status
ports:
- containerPort: 5222
  hostPort: 5222
- containerPort: 5269
  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: ./conf/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: 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: 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
  - 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

```

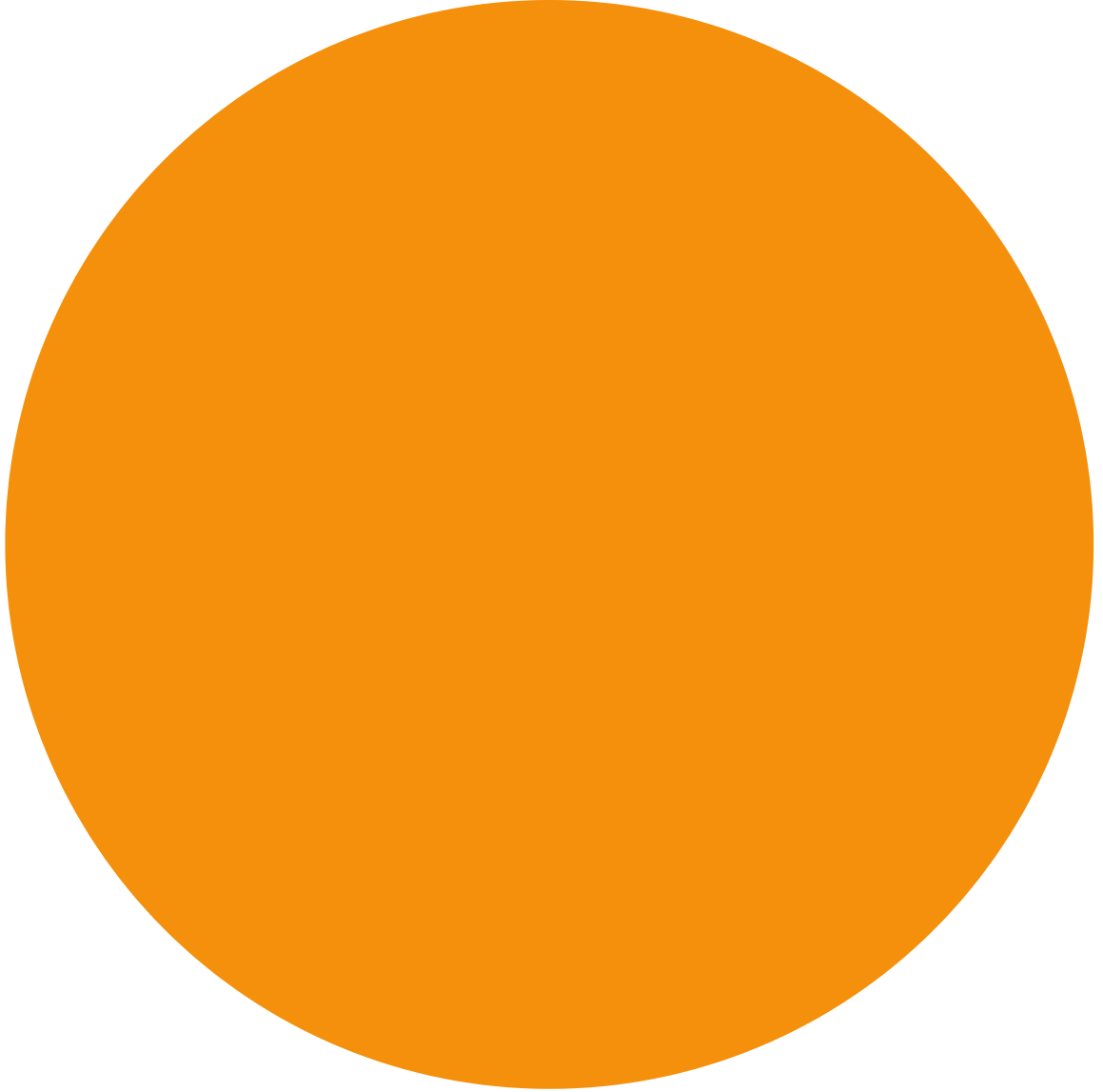
Images Comparison¶

Let's summarize the differences between both container images. Legend:



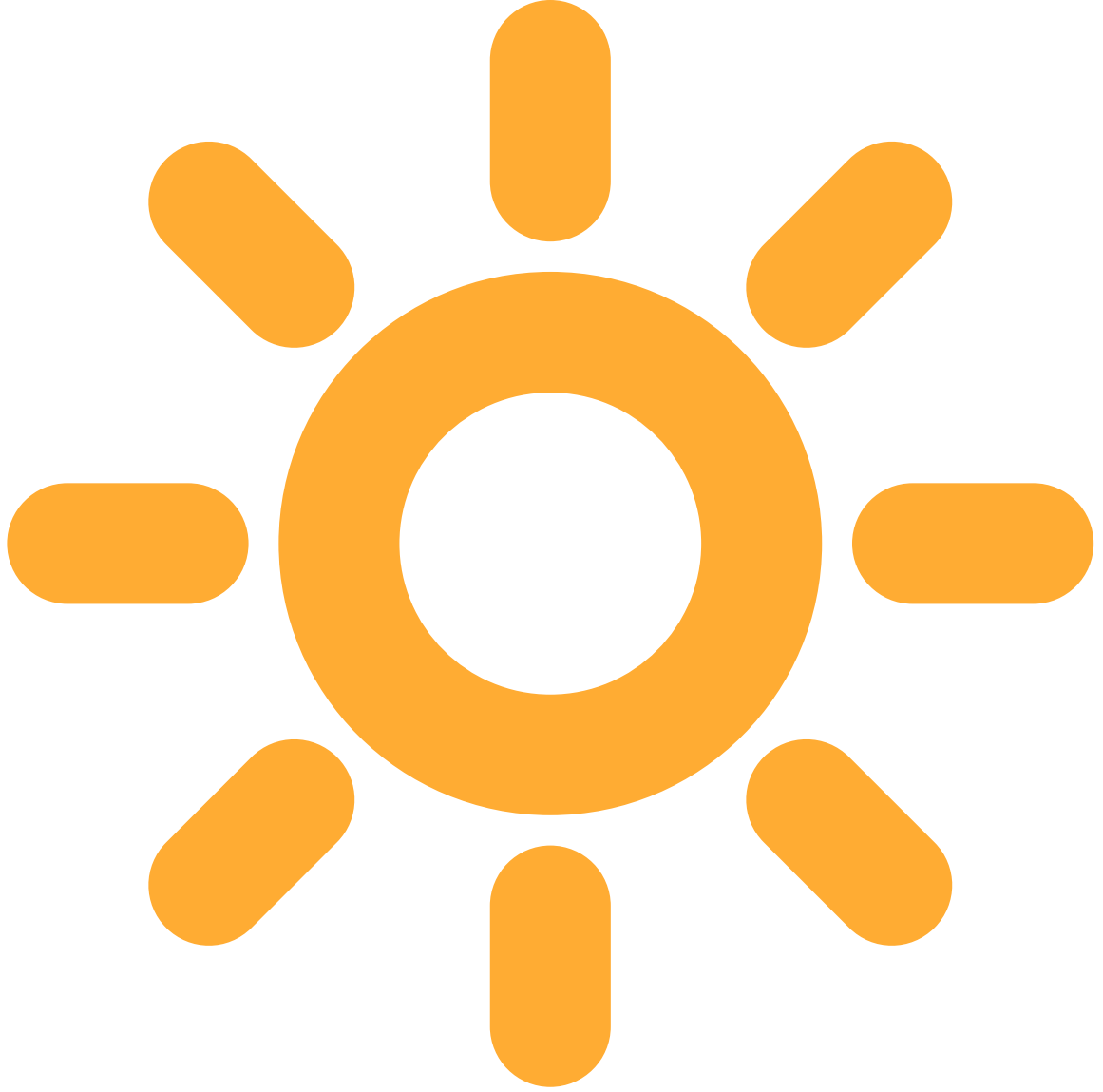
•

is the recommended alternative



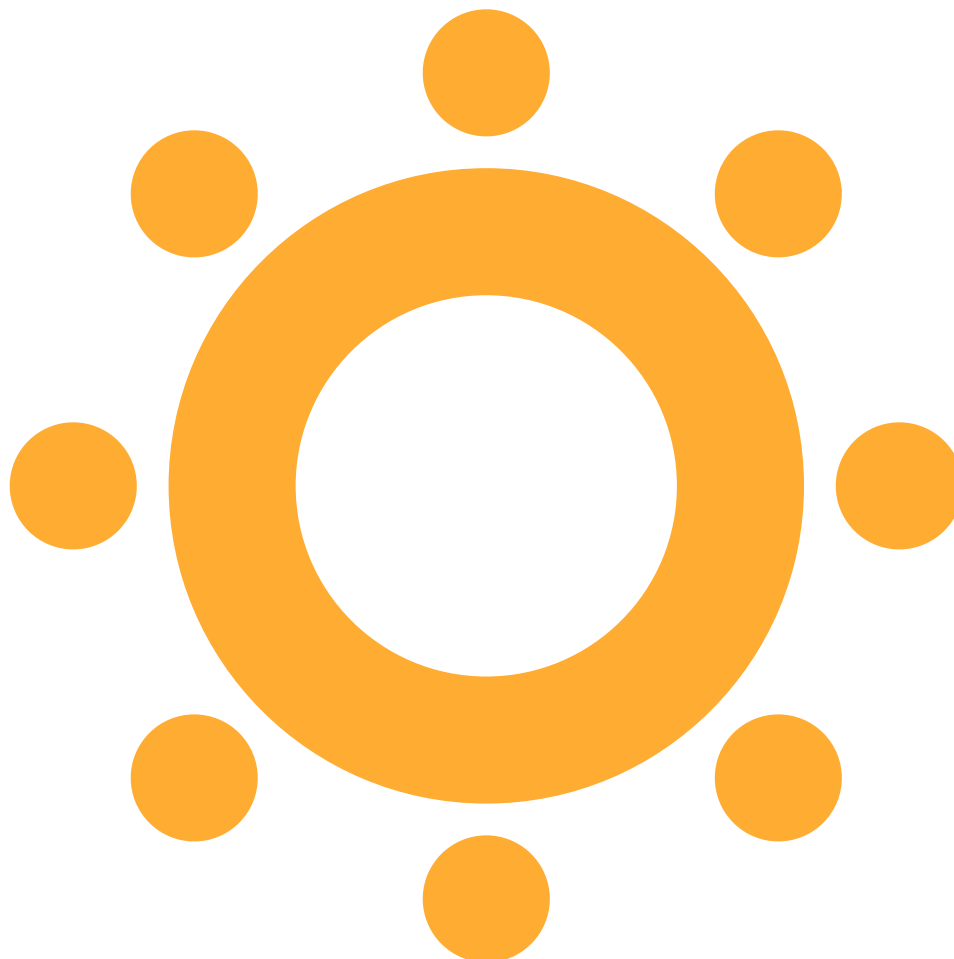
-

added in the latest release (ejabberd 25.03)





•

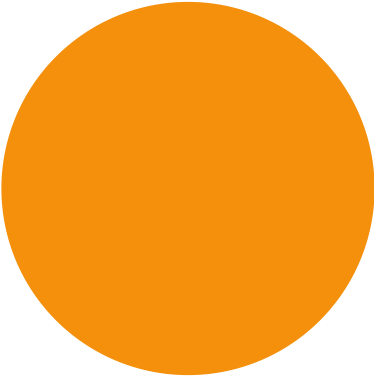
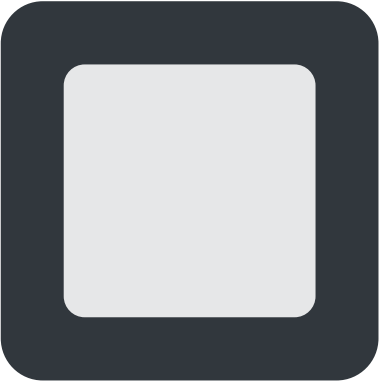
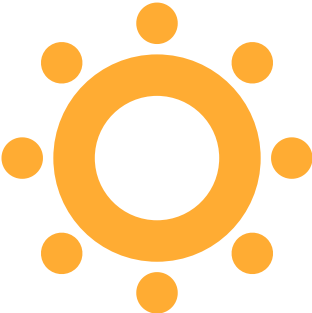
added in the previous release (ejabberd 24.12)





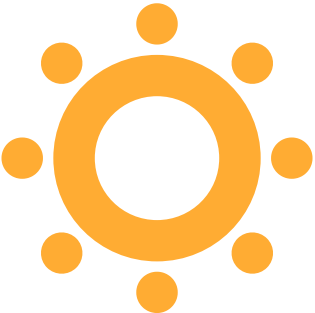
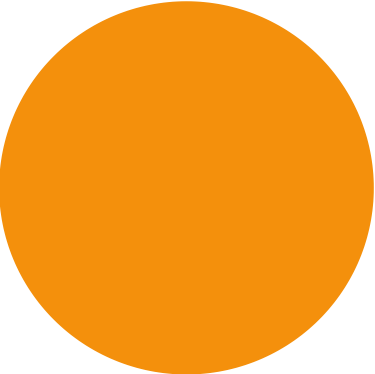
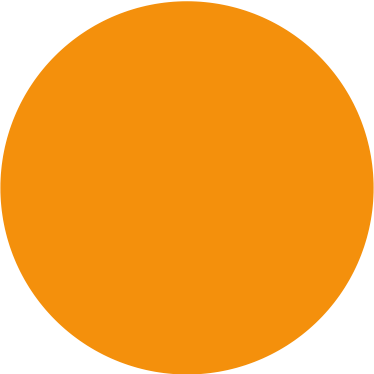



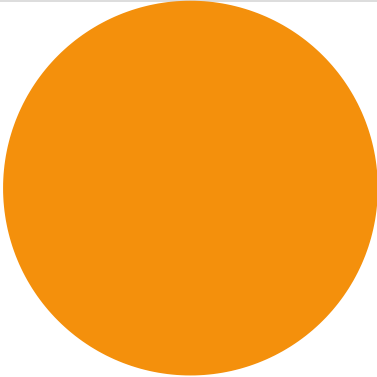

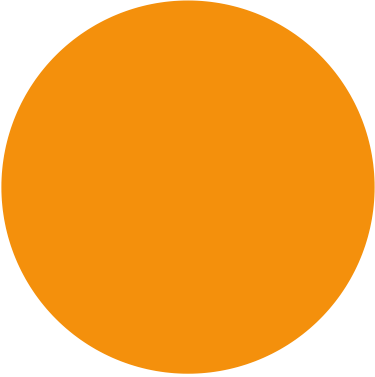

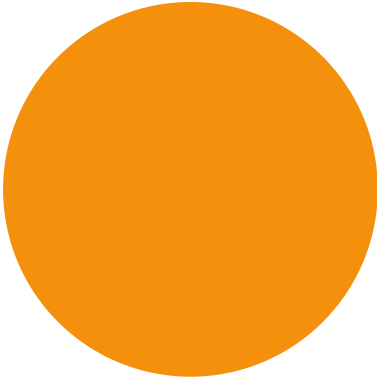

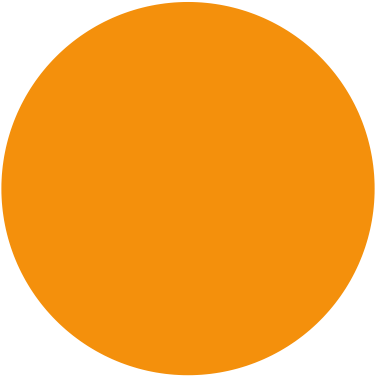
- added in the pre-previous release (ejabberd 24.10)



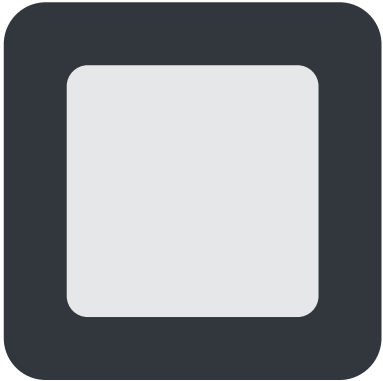
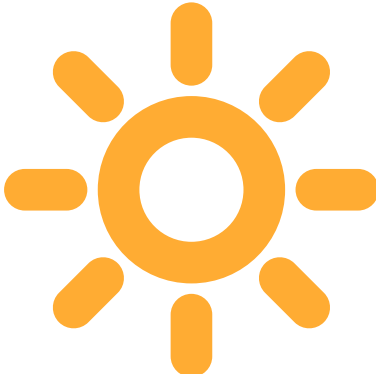
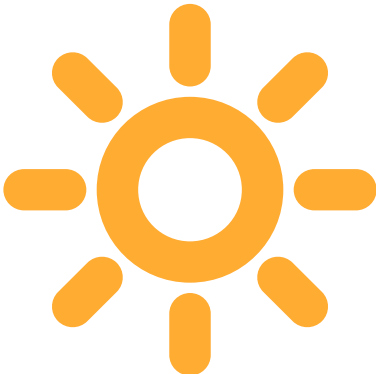
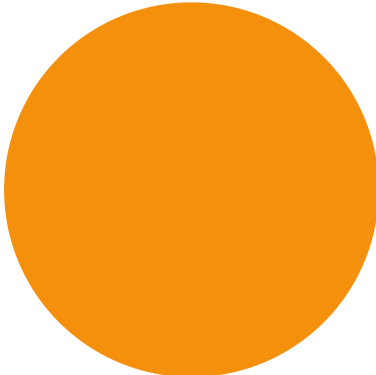
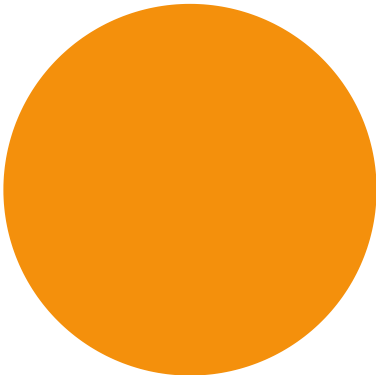
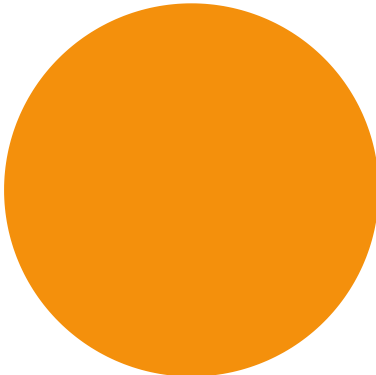
	 ejabberd	 ecs
Source code	ejabberd/.github/container	docker-ejabberd/ecs
Generated by	container.yml	tests.yml
Built for	stable releases master branch	stable releases master branch zip
Architectures	linux/amd64 linux/arm64	linux/amd64
Software	Erlang/OTP 27.3.4.1-alpine Elixir 1.18.4	Alpine 3.19 Erlang/OTP 26.2 Elixir 1.15.7
Published in	ghcr.io/processone/ejabberd	docker.io/ejabberd/ecs ghcr.io/processone/ecs

	 ejabberd	 ecs
 Additional content		
ejabberd-contrib	included	not included
ejabberdapi	included 	included
 Ports		
1880 for WebAdmin	yes 	yes 

	ejabberd	ecs
5210 for ERL_DIST_PORT	supported	supported 
 Paths		
\$HOME	/opt/ejabberd/	/home/ejabberd/
User data	<div>\$HOME </div> <div>/home/ejabberd/ </div>	<div>\$HOME /opt/ejabberd/ </div> <div></div>

	 ejabberd	 ecs
ejabberdctl	<div>ejabberdctl</div> <div></div> <div>bin/ejabberdctl</div>	<div>bin/ejabberdctl</div> <div>ejabberdctl</div> <div></div> <div></div>
captcha.sh	<div>\$HOME/bin/captcha.sh</div> <div></div>	<div>\$HOME/bin/captcha.sh</div> <div></div>
.sql files	<div>\$HOME/sql/.sql</div> <div></div>	<div>\$HOME/database/*.sql</div> <div>\$HOME/sql/*.sql</div>

	ejabberd	ecs
	<div> \$HOME/database/*.sql</div>	<div> </div>
Mnesia spool files	<div>\$HOME/database/  \$HOME/database/NODENAME/ </div>	<div>\$HOME/database/NODENAME/ \$HOME/database/  </div>

	 ejabberd	 ecs
 Variables		
EJABBERD_MACRO_*	supported 	supported 
Macros used in ejabberd.yml	yes 	yes 
EJABBERD_MACRO_ADMIN	Grant admin rights  (default admin@localhost)	Hardcoded admin@localhost

	 ejabberd	 ecs
REGISTER_ADMIN_PASSWORD	Register admin account 	unsupported
CTL_OVER_HTTP	enabled 	unsupported