Jitsi Meet with Authentication

https://www.youtube.com/embed/lnYNWcPmDRo

Jitsi Meet with Authentication

JitsiMeet is an amazing, powerful, open source, self hosted vide, audio, and collaboration server. It has the option for Chat, shared screens, and so much more, all from your modern web browser.

Before the Pandemic, JitsiMeet was as simple as typing in a URL (e.g. https://meet.jit.si) adding a slash '/', and then naming a room. Share that link with anyone and everyone, and they can all join without issue.

When the pandemic happened,and more people began turning to open source, self hosted projects to help their businesses, classrooms, and social interactions continue furing lock-down periods and heightened remote work / learning, it became clear that JitsiMeet needed some authentication around it.

Today, I'll be going through how to setup two of those forms of authentication. The first is the 'internal' authentication method where the user information is stored as part of your jitsi install. This is great for a single user, or maybe even a few users, but much beyond that, and setting up the user accounts will quickly become cumbersome at best.

For larger organizations, we have more options. We'll be looking at the LDAP option, and I'll be showing how I used FreeIPA to setup the LDAP authentication with JitsiMeet as well.

What You'll Need

- A server, vm, or container to host the JitsiMeet software.
- Docker-CE and Docker Compose installed on that server.
- A domain / sub-domain name that you can add A-Record DNS entries for.
- (Optional) A Reverse Proxy
- (Optional, but required for the second part) An LDAP server and access to the appropriate settings of that server (OpenLDAP, FreeIPA, etc).
- About 30 minutes of your time

Install Docker-CE and Docker Compose

If you're on Windows, then go get the official docker stuff for Windows. I don't use Windows, so couldn't begin to tell you how to do it.

If you're on Linux (Smart move by the way), then you can likely install Docker and Docker compose with a one line script.

```
curl https://get.docker.com | sh
```

Enter your superuser password when prompted, and let the script do its work.

Add your user to the 'docker' group, and you'll be able to run `docker` and `docker-compose` commands without using `sudo` every time.

```
sudo usermod -aG docker <your-username>
```

Replace `<your-username>` with your actual username on the system.

Now log out and back in, and you'll have access to the `docker` and `docker compose` commands.

Get the Jitsi Docker Compose Files

First, we'll create a folder structure for our docker based application(s). I do this so it's very easy to zip up the parent 'docker' folder, and then back it up on a backup server else where in case of disaster.

```
mkdir -p docker/jitsi
```

Now, we'll move into the new jitsi folder:

```
cd docker/jitsi
```

Next, we'll download teh necessary files to this location with the command:

```
wget $(curl -s https://api.github.com/repos/jitsi/docker-jitsi-meet/releases/latest | grep 'zip' | cut -d\" -f4)
```

This will download the latest releae version of the files for us in a .zip file.

Let's install the 'unzip' tool, and decompress this file.

Ubuntu / Debian

```
sudo apt install unzip
```

RedHat / CentOS / Fedora

`sudo dnf install unzip`

Now we can unzip the file we downloaded with:

`unzip ./stable-<version number>`

Replace `<version number>` with the actual file name and number you downloaded.

After it's unzipped, you'll have a new folder with a really long name. Mine is:

`jitsi-docker-jitsi-meet-aae3756`

Let's change that by renaming it:

`mv jitsi-docker-jitsi-meet-aae3756 jitsi-meet`

This will rename the folder to 'jitsi-meet'. Now we can move into the folder, and start editing the files we need in order for the system to function the way we want it to.

`cd jitsi-meet`

Copy Docker Compose and Environment Files

We'll make copies of two of the files to work from. First, let's copy the 'env.example' file to a new hidden file called '.env'. This is the file docker compose uses by default for configurations.

`cp env.example .env`

Next, let's copy the `docker-compose.yml` file to a new file called 'compose.yaml' (this is the newer naming convention used by Docker Compose). Then we'll rename the original file by adding a '.bak' extension to the end, so Docker Compose will default to our 'compose.yaml' file.

`cp docker-compose.yml compose.yaml`

'mv docker-compose.yml docker-compose.yml.bak'

Edit our .env file

Now we need to make some changes to the example .env file we are starting with. This is the place we want to put any configuration values whenever possible.

Setting Up Internal Authentication

Open the 'env' file to make changes. Note that any line beginning with a '#' hashtag / pound sign / number sign is a comment, and is ignored by Docker when the containers are started.

```
nano .env
```

Move down to the line that says `CONFIG=~/.jitsi-meet-cfg`, and change this by removing the '~' tilda, and replace it with a period '.'. It should look like this now:

```
CONFIG=./.jitsi-meet-cfg
```

This change puts our configuration volume inside of our parent `docker/jitsi/jitsi-meet` folder with the rest of our files.

Next, we'll move down to the HTTP_PORT and HTTPS_PORT entries. If you have nothing using the default ports of 8000 and 8443, then feel free to leave these as is. If, however, you have other applications already using these ports, then you should change to a port that is not in use on the system already. Just remember what you change them to, as we'll need them when setting up our Reverse PRoxy. I left mine as-is.

Next, let's set our timezone properly. for the `TZ=UTC` setting, remove `UTC` and enter your timezone. I'm in Central US Time, so I entere `TZ=America/Chicago`.

Now for one of the big one. The `PUBLIC_URL` value is very important. This tells the JisiMeet system where to expect a request to come to for it's pages. So, if I want users to reach my JitsiMeet site at `https://meet.itsawesome.com`, I need to enter that for the `PUBLIC_URL`.

```
PUBLIC_URL=https://meet.itsawesome.com
```

For the `JVB_ADVERTISE_IPS` value, we want to enter the IPv4 address(es) where the JitsiMeet will be reached. I entered two addresses. First my private LAN address, then my public IP address.

```
JVB_ADVERTISE_IPS=192.168.21.14,209.41.5.158
```

You don't have to enter more than 1 address if you are setting up your server facing the internet directly through it's Public IP.

At this point, let's quickly save so we don't lose any chagnes. Use CTRL + O to save, then press Enter to confirm.

Continuing down, we need to move way down to the entry for `Authentication configuration`.

In this section we'll be initially setting up the 'Internal' authentication method.

First, uncomment the line that says `#ENABLE_AUTH=1`, which means just remove the `#` at the beginning of that line.

Next, you need to decide if you'll have users who are "guests" (users who don't have any authentication credentials for this server), who may need / want to join a meeting. If you do, then we need to uncomment the line that says `ENABLE_GUESTS=1` (again, just remove the `#` from the beginning of the line.)

If you don't want to allow guests, then only users who can authenticate will be able to join meetings.

Users who are guests, cannot start a meeting, they can only get prepared, then will have to wait until a user who can authenticate starts the meeting to allow the guests in.

Finally, we need to set the `AUTH_TYPE`. This is an important one, as it determines what other informaiton you may need to enter in order to allow users to authenticate for meetings. We'll start with 'interna'.

```
AUTH_TYPE=internal
```

Again, let's save our changes. CTRL + O to save, then press Enter to confirm.

We'll scroll down some more to get to the `Security` section. Here you'll be asked to enter multiple long strong passwords in order to better protect your site. Never fear! The good folks at JitsiMeet mad e simple script to help us do this easily.

Let's exit our `.env` file with CTRL + X (if you are prompted to save, then press 'y', and then press Enter).

Now, let's run the script by entering:

```
./gen-passwords.sh
```

After running, you should be returned to a simple command prompt.

We'll jump back into our `.env` file with:

```
nano .env
```

And we can move down quickly to the `Security` section by typing CTRL + W (search), then enter '# Security', and press Enter.

If you look at the six password request fields now, they should each have a different long, strong password filled in, randomized by the script. Yay!

Finally, let's uncomment the `RESTART_POLICY=unless-stopped`, as this set's this as the restart for each of the containers.

Save one last time with CTRL + O, then press Enter to confirm, and exit the nano editor with CTRL + X.

At this point, there should be no changes required in our `compose.yaml` file. Let's pull the images for our setup with:

```
docker compose pull
```

If everything pulls down with no errors, then we are ready to start our server up. We can do that with two concatenated commands:

```
docker compose up -d && docker compose logs -f
```

The first part tells Docker Compose to start the server containers and run them detached in the background (-d). The second part tells Docker Compose to display the live log output as the contares are starting up, and allows us to follow (-f) the logs. When you're done looking at the logs, you can stop following them with CTRL + C.

Now, our application is running, but remember, we set a domain / url for it to be accessed from. Let's go setup our DNS A-Record and / or Reverse Proxy.

Setup DNS and Reverse Proxies

DNS A Records allow you to point a domain / subdomain that you own at a specific IP address. In this case, we likely want a public IP address.

I own the domain `opensourceisawesome.com`. I setup an A-Record in my DNS settings on my domain registrars page that points `meet.opensourceisawesome.com` to the public IP address of my server host. My application is not running directly on the host, however, and is instead running in a virtual machine (called docker). So I need to "proxy" the traffic from my host through to that virtual machine.

I can setup my reverse proxy (I use NGinX Proxy Manager) to listen for requests for `meet.opensourceisawesome.com`, and when it receives such a request, have it forward that traffic to my internal private IP and the Port number I setup. In my case I set the following settings in NGinX Proxy Manager.

```
Details Tab
-----------------
Domain Name: meet.opensourceisawesome.com
Sheme: https
Forward Hostname / IP: 192.168.42.188
Port: 8443 <-- the HTTPS port
Block Common Exploits: enabled
Websockets Support: enabled
```

```
SSL Tab
---------------
SSL Certificate: Request a new certificate
Force SSL: enabled
HTTP/2 Support: enabled
HSTS Enabled: enabled
HSTS Subdomains: enabled

Email: entered@yourdomain.com
Accept LetsEncrypt TOS: Enabled
```

Once I set those settings, I clicked save, and waited for about 20 seconds. LetsEncrypt checked that my server could be reached, and then issued a valid SSL certificate for my site.

Now, we can open our favorite modern web browse, and go to our JitsiMeet URL to make sure it loads.

I went to `https://meet.opensourceisawesome.com` and it loaded just fine.

If, however, I try to create a room, and join it, when I finish allowin my camera and mic, and enter my name, then I'll be stopped, and told I need to login as a moderator, or wait for a moderator to join before I can join the room.

Setup a User for Internal Auth

Here is where the `internal` authentication method becomes cumbersome. We have to manually create users and passwords for our Jitsi instance through the command line, inside the docker container for Prosody.

If you only have 1 or 2 users, no big deal, but if you get much beyond 4 users, it will start to be a pain, and if you are trying to keep up with people being hired, and leaving a company, it will be a nightmare in a hurry. But, let's set one up, and you can decide if you want to keep it this way.

Let's go back to our command line, and enter the command:

`docker compose ps`

This command will list out the running containers from the `compose.yaml` file in this folder.

We will see several heading, the first of which is `NAME`. We need to find the name of the `prosody` container.

In mine it's `jitsi-meet-jitsi-prosody-1`.

So, let's get into this container with:

`docker exec -it jitsi-meet-prosody-1 /bin/bash`

You should see a different prompt come up in your command line, which indicates you are inside of a container, and it's likely the container id.

From here we can now enter the JitsiMeet command to add a user.

`prosodyctl --config /config/prosody.cfg.lua register <username> meet.jitsi <users-assigned-password>`

Replace `<username>` and `<users-assigned-passwrod>` with the actual username and password you want for the user. You'll need to repeat this for each user you want to have access to start meetings on the system. If you chose to not allow guests, then you'll need to do this to create a user for every person who needs to join a meeting.

Now you can exit the container with `exit`.

Head back to your browser, refresh the page, and now try to join the meeting. Click the 'Login' button when prompted, and enter your username and password. If all went according to plan, you'll be put into your active live meeting room.

Not ideal for large groups, but if it's just you, then it's probably just fine.

LDAP using FreeIPA

If you're looking for a more flexible way to control who can and can't login to your JitsiMeet instance, then you have LDAP as a potential solution. I setup FreeIPA a few videos back. One of the things it provides is the ability to have LDAP authentication.

We can put that to use here to make it easier to manager users, and hopefully provide them more of an SSO type login as needed.

If you havne't setup FreeIPA, and are intereseted in it, here's a link to my Video on it.

Assuming you have a LDAP server setup (preferrably FreeIPA as that's what I'm using), then let's move forward with getting JitsiMeet to authenticate a user with it.

First, we'll stop our containers with the command:

`docker compose down`

Once stopped, we need to remove the configuration mapped volume we setup in the .env folder earlier, so we can get out LDAP configuration setup instead.

`sudo rm -rf .jitsi-meet-cfg`

Now, we'll edit our .env with:

`nano .env`

Use CTRL + W, and enter the term "# Authentication" then press Enter to jump down to the Authnetication section.

Here, we just need to change the `AUTH_TYPE` from `internal` to `ldap`. After the change it should look like:

`AUTH_TYPE=ldap`

Save this change with CTRL + O, then press Enter to confirm.

We'll scroll down until we get to the "LDAP Authentication" section. Here we'll need some information about our FreeIPA server setup. You need to have the following things:

1. Your FreeIPA fully qualified domain name (even it it's only a local domain), or your FreeIPA IPv4 Address.
2. Your FreeIPA Base DN. You can get this from the FreeIPA server in the command line by looking at the 'default.conf' file in `/etc/ipa`.

`cat /etc/ipa/default.conf`

Find the value for `basedn` in that file.

Back in our `.env` file, let's uncomment the field for `LDAP_URL` and enter the URL to your FreeIPA server. In my case, my server is local only, and it's FQDN is `freeipa.fixitdelrio.local`, so my line looks like:

`LDAP_URL=ldap://freeipa.fixitdelrio.local`

Next, uncomment the line for `LDAP_BASE`. This is where you'll need the 'basedn' information from your server.

My `default.conf` file shows my 'basedn' as:

`basedn = dc=fixitdelrio,dc=local`

We also need to add a `CN` component to the `LDAP_BASE` for 'accounts'.

We need to structure the `LDAP_BASE` in the following way, using your own values for the basedn fields of course.

`LDAP_BASE=CN=accounts,DC=fixitdelrio,DC=local`

Next, move down to the line that starts with `#LDAP_FILTER=(sAM...`, and uncomment this line. Remove the contents inside the parentheses, and replace them with `uid=%u`. The proper line should look like:

`LDAP_FILTER=(uid=%u)`

Uncomment the line for `LDAP_VERSION=3`, and finally uncomment the line for `LDAP_USE_TLS=1`.

The whole block should look like this:

```
# LDAP authentication (for more information see the Cyrus SASL saslauthd.conf man page)
#

# LDAP url for connection
LDAP_URL=ldap://freeipa.fixitdelrio.local

# LDAP base DN. Can be empty
LDAP_BASE=CN=accounts,DC=fixitdelrio,DC=local

# LDAP user DN. Do not specify this parameter for the anonymous bind
#LDAP_BINDDN=UID=admin,OU=users,DC=example,DC=com

# LDAP user password. Do not specify this parameter for the anonymous bind
#LDAP_BINDPW=some-password

# LDAP filter. Tokens example:
# %1-9 - if the input key is user@mail.domain.com, then %1 is com, %2 is domain and %3 i>
# %s - %s is replaced by the complete service string
# %r - %r is replaced by the complete realm string
LDAP_FILTER=(uid=%u)

# LDAP authentication method
#LDAP_AUTH_METHOD=bind

# LDAP version
LDAP_VERSION=3

# LDAP TLS using
LDAP_USE_TLS=1

# List of SSL/TLS ciphers to allow
```

```
#LDAP_TLS_CIPHERS=SECURE256:SECURE128:!AES-128-CBC:!ARCFOUR-128:!CAMELLIA-128-CBC:!3DES->

# Require and verify server certificate
#LDAP_TLS_CHECK_PEER=1

# Path to CA cert file. Used when server certificate verify is enabled
#LDAP_TLS_CACERT_FILE=

# Path to CA certs directory. Used when server certificate verify is enabled
#LDAP_TLS_CACERT_DIR=

# Wether to use starttls, implies LDAPv3 and requires ldap:// instead of ldaps://
#LDAP_START_TLS=1
```

Save your changes with CTRL + O, then press Enter to confirm, and exit the nano editor with CTRL + X.

Make sure your Prosody container can find your FreeIPA Server

One of the things I ran into, is with my FreeIPA server being only on the LAN / VPN, and me not setting up DNS to deal with it (:-s), is that the Prosody Container didn't have a way to route to my FreeIPA server, and thus logging in failed. So, let's fix that now.

We can modify our `compose.yaml` file to allow the Porsody container to know where our server is. Normally you would add an entry in your `/etc/hosts` file, but in this case the great people at the Docker and Docker Compose projects thought of this, and gave us an easy way to fis it before we bring up our containers.

`nano compose.yaml`

Now, scroll down to the section for the `prosody:` service. Mine shows `# XMPP server` right above it.

Move your cursor down to the `volumes` line, and hit Enter to make a new blank line above it. Make sure to fix any spacing issues, as yaml is space-dependent.

Now insert a new entry lined up directly above the 'v' in `volumes:`. Here we want to add `extra_hosts:`, then go to the next line, and indent 2 more spaces beyond the 'e' in `extra_hosts:`. Here we'll add a line for our FreeIPA server hostname, and another for the FreeIPA fqdn. Mine looks like:

```
        extra_hosts:
          - "freeipa:192.168.10.17"
          - "freeipa.fixitdelrio.local:192.168.10.17"
```

The whole section now looks like this:

```
prosody:
        image: jitsi/prosody:${JITSI_IMAGE_VERSION:-stable-10133-1}
        restart: ${RESTART_POLICY:-unless-stopped}
        container_name: jitsi-prosody
        expose:
            - '${XMPP_PORT:-5222}'
            - '${PROSODY_S2S_PORT:-5269}'
            - '5347'
            - '${PROSODY_HTTP_PORT:-5280}'
        labels:
            service: "jitsi-prosody"
        extra_hosts:
          - "freeipa:192.168.10.17"
          - "freeipa.fixitdelrio.local:192.168.10.17"
        volumes:
            - ${CONFIG}/prosody/config:/config:Z
            - ${CONFIG}/prosody/prosody-plugins-custom:/prosody-plugins-custom:Z
```

Along with everything before and after that part, of course.

This `extra_hosts:` section allows us to add entries for the container to know about, even if it isn't using the host server's networking directly. It's like adding to the `/etc/hosts` file inside the container essentially.

Save your changes with CTRL + O, then press Enter, and exit the nano editor with CTRL + X.

Now, we can restart our Jitsi Containers, and check our work.

`docker compose up -d && docker compose logs -f`

Again, as long as you don't get any serious errors, or containers constantly exiting and trying again, you should be set after 30 seconds or so. You can quit following the logs with CTRL + C.

Open your browser back up, and refresh the page at the very least. If possible clear the cache just to make sure the updates are fresh and ready.

Create a room on your Jitsi server, allow your mic and / or camera, enter you name, and click the Join button. You should again be prompted to 'Login', so click that 'Login' button, and when prompted enter your LDAP / FreeIPA username and password, then proceed.

If we've set everything up properly, you should be placed into your live meeting room.

This is a lot, and believe me, I know it. It took me about a week of tinkering, researching, and trying and failing before I got this to work. Be patient, think through the process, and be persistent.

I hope this helps you, and if it does make sure to share it with others.

Revision #1
Created 6 May 2025 15:28:21 by Brian McGonagill
Updated 6 May 2025 15:30:05 by Brian McGonagill