Our Chat !

An easy-to-install Enterprise LLM chat system using LibreChat with AWS Bedrock and LDAP/AD authentication.

Why is this needed? Can't users just access AWS Bedrock directly? They might; however, in most organizations, the AWS Console is reserved for power users, as it typically takes some time until you are familiar with it (to put it mildly).

LibreChat, on the other hand, takes zero on-boarding time; users simply login with their enterprise credentials and use the system just like ChatGPT or Claude.ai. Another reason to like LibreChat is its superior user interface. It has gained much popularity and is often trending on GitHub.

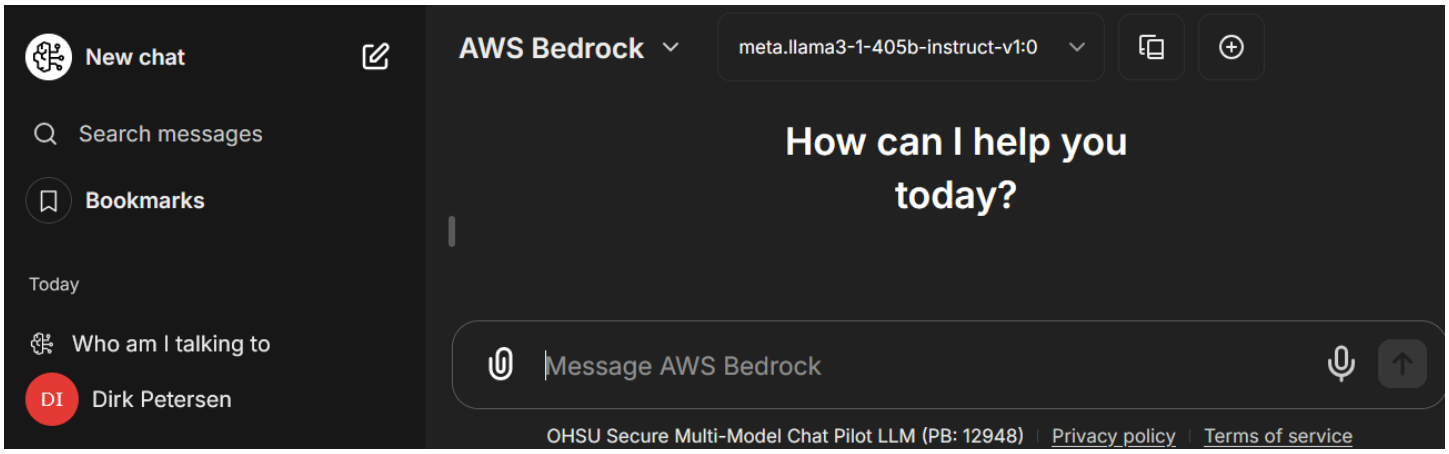


Figure 1: image

Other big benefits:

- No per-user cost. Even a few hundred ChatGPT users can cost an organization hundreds of thousands of dollars per year. With LibreChat, you can serve tens of thousands of users and only pay the API cost of what they actually use.
- No configuration needed to prevent the cloud LLM from learning with your data. All API use in Bedrock, as well as OpenAI, is excluded from learning usage.
- Presumably better security: OpenAI, Anthropic, and other startups have significantly smaller cybersecurity operations than the cloud Hyperscalers, such as AWS, Azure, & Google.
- Unified user interface: In addition to Bedrock, you can use OpenAI or Google API or even on-prem use cases, all within the same interface.

Table of Contents:

Prerequisites

- Get a RHEL virtual server (this process was tested with RHEL 9.4) with at least 8GB RAM and 50GB free disk space, of which about half should be under /home. As this server might process sensitive data, ask for all security software (log forwarder, Antivirus/malware, intrusion prevention) to be preinstalled.
- This machine must be able to communicate with the `ldaps port 636` of your enterprise LDAP server (for example, Active Directory).
- An LDAP/AD security group that contains the users who are allowed to use the chat system. For now, we call this group `our-chat-users`.
- An SSL certificate, unless you use Let's Encrypt.
- AWS credentials (AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY) for an AWS service account (perhaps called librechat or ochat) that has no permissions except for the AmazonBedrockFullAccess policy attached to it.
- You don't require root access if your sysadmins can run the `prepare-server.sh` script for you, but they should allow you to switch to the ochat user, e.g., `sudo su - ochat`.

Prepare Server

Run the prepare-server.sh script as root user to install docker and prepare the `ochat` user account. You can also start it as a normal user if you have requested the correct sudo config.

```
curl https://raw.githubusercontent.com/dirkpetersen/our-chat/refs/heads/main/prepare-server.sh?token=$(date +%
```

Now switch to the ochat user with `sudo su - ochat` and continue with Prepare install.

Prepare install

After switching to the ochat user (`sudo su - ochat`), please clone the our-chat repository from GitHub and copy the .env.ochat, librechat.yml and nginx.conf files to the root of the home directory of the ochat user:

```
cd ~
git clone https://github.com/dirkpetersen/our-chat/
cp ~/our-chat/.env.ochat ~/.env
cp ~/our-chat/librechat.yaml ~/librechat.yaml
cp ~/our-chat/nginx.conf ~/nginx.conf
```

AWS connectivity

Export the AWS credentials and region of the Bedrock service account (the one that has only the AmazonBedrockFullAccess policy attached) to environment variables:

```
export AWS_ACCESS_KEY_ID=abcdefghijklmnopqrstuvwxyz
export AWS_SECRET_ACCESS_KEY=abcdefghijklmnopqrstuvwxyz123456
export AWS_DEFAULT_REGION=us-west-2
```

Then test the connectivity to AWS Bedrock by running:

```
~/our-chat/tests/bedrock-test.py
```

You should see that your credentials have been written to the correct places under ~/.aws and Bedrock shows a list of available models and responds to a "Hello World" prompt:

```
Written to /home/librechat/.aws/credentials: AWS credentials for [default]
Written to /home/librechat/.aws/config: AWS region configuration for [default]

List of available models on AWS Bedrock:

amazon.titan-tg1-large
amazon.titan-embed-g1-text-02
.
.
mistral.mistral-large-2407-v1:0

Response to 'Hello, world': Hello! How can I assist you today? Feel free to ask me anything or let me know if
```

If you don't get that response or the script shows an error, go back to your AWS Administrator for troubleshooting your AWS credentials or permissions before you continue.

You can find more details about AWS in the AWS budget section below.

SSL certificates

Here we cover the standard case of you receiving SSL certs from your enterprise team. In many cases your team will send you a PKCS12 archive which comes as a *.pfx file along with an import password. After entering the import password, you need to set a new PEM pass phrase. At the end of this process you should have `~/our-chat.pem` and `~/our-chat.pw` at the root of the ochat home directory. (Please note: If you **leave a space at the beginning** of the echo "your-pem-passphrase" line, the pass phrase will not end up in your bash history).

```
openssl pkcs12 -in cert-from-IT.pfx -out ~/our-chat.pem

Enter Import Password:
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

 echo "your-pem-passphrase" > ~/our-chat.pw
chmod 600 ~/our-chat.*
```

LibreChat Configuration

You will likely need to edit each of these config files at some point, but for now you should only edit the `~/.env` file to enable LDAP authentication and to update a few security tokens. If you are in a Disaster Recovery / Business Continuity sitution, please do not edit the 3 files below but restore them from your secure location.

.env

Please find these settings in ~/.env:

```
vi ~/.env

LDAP_URL=ldaps://ldap.domain.edu:636
LDAP_USER_SEARCH_BASE=OU=User Accounts,dc=domain,dc=edu
LDAP_BIND_DN=CN=myserviceaccount,OU=Service Accounts,OU=User Accounts,DC=domain,DC=edu
LDAP_BIND_CREDENTIALS="ad-password"
LDAP_LOGIN_USES_USERNAME=true
LDAP_SEARCH_FILTER=(&(sAMAccountName={{username}})(memberOf=CN=our-chat-users,OU=Groups,DC=domain,DC=edu))
LDAP_FULL_NAME=displayName
```

These settings should be pretty self-explanatory. For example, `LDAP_LOGIN_USES_USERNAME` means that you can log in with your username instead of typing your entire email address. The LDAP_SEARCH_FILTER is a bit of a workaround that we use to restrict LibreChat to members of an AD/LDAP security group. The filter was not intended for authorization, and if a new user is not a member of that group, they will get a 401 error (AuthN) instead of 403 (AuthZ). This can be a bit confusing. On some LDAP systems, the LDAP_BIND_DN can be the email address (aka service principal) of the service account, e.g., `myserviceaccount@domain.edu`, but it is safest to use the DN (distinguished name).

After you have configured all these 7 settings, please use the LDAP test script `~/our-chat/tests/ldap-test.py <testuser>` to verify these settings. To check the LDAP_SEARCH_FILTER, you have to pass the username of a test user who is member of that security group (e.g. our-chat-users)

```
~/our-chat/tests/ldap-test.py peterdir

/ldap-test.py
Successfully read environment variables: ['LDAP_URL', 'LDAP_USER_SEARCH_BASE', 'LDAP_BIND_DN', 'LDAP_BIND_CREDI

Connected as: CN=myserviceaccount,OU=Service Accounts,OU=User Accounts,DC=domain,DC=edu

Evaluating LDAP_SEARCH_FILTER with testuser peterdir:

Evaluating search filter: (&(sAMAccountName=peterdir)(memberOf=CN=our-chat-users,OU=Groups,DC=domain,DC=edu))
Found 1 matching entries:
DN: CN=peterdir,OU=User Accounts,DC=ohsum01,DC=ohsu,DC=edu
Attributes:
  displayName: Dirk Petersen
  memberOf: ['CN=unix_ochat,OU=Groups,DC=domain,DC=edu', ......
```

And as a final step we want to setup unique tokens for

CREDS_KEY, CREDS_IV, JWT_SECRET, JWT_REFRESH_SECRET and MEILI_MASTER_KEY

Go to https://www.librechat.ai/toolkit/creds_generator, generate keys and replace the existing tokens for these 5 keys in ~/.env

```
vi ~/.env
```

You can likely skip the next 2 config files for now, but might want to change them later

librechat.yml (optional)

For example, use `vi ~/librechat.yml && cp ~/librechat.yml ~/LibreChat/librechat.yml` to change the terms of service, modify the site footer and change a few advanced bedrock settings, for example allowed AWS regions.

nginx.conf (optional)

The only change `vi ~/nginx.conf && cp ~/nginx.conf ~/LibreChat/client/nginx.conf` likely requires, is setting the filenames for the SSL certificates for https, if you choose a different cerificate name than our-chat.pem / our-chat.pw . You can set addional nginx headers to further enhance security.

```
   ssl_certificate /etc/librechat/ssl/our-chat.pem;
   ssl_certificate_key /etc/librechat/ssl/our-chat.pem;
   ssl_password_file /etc/librechat/ssl/our-chat.pw;
```

Install LibreChat

When all the prep work is done correctly, you should be able to run `install-librechat.sh` and have a running system in a few seconds:

```
~/our-chat/install-librechat.sh

Cloning into 'LibreChat'...
```

```
remote: Enumerating objects: 33792, done.
remote: Counting objects: 100% (5880/5880), done.
remote: Compressing objects: 100% (1247/1247), done.
remote: Total 33792 (delta 5103), reused 4786 (delta 4622), pack-reused 27912 (from 1)
Receiving objects: 100% (33792/33792), 43.72 MiB | 44.24 MiB/s, done.
Resolving deltas: 100% (24182/24182), done.
Copying /home/ochat/LibreChat/deploy-compose.yml to /home/ochat/LibreChat/deploy-compose-ourchat.yml
~/.awsrc has been added to .bashrc
Copying /home/ochat/.env to /home/ochat/LibreChat/.env and expanding env vars
Copying /home/ochat/librechat.yaml to /home/ochat/LibreChat/librechat.yaml
Copying /home/ochat/nginx.conf to /home/ochat/LibreChat/client/nginx.conf
Generating DH parameters, 2048 bit long safe prime
.
.
[+] Running 8/8
  Network librechat_default       Created
  Volume "librechat_pgdata2"      Created
  Container chat-mongodb          Started
  Container chat-meilisearch      Started
  Container librechat-vectordb-1  Started
  Container librechat-rag_api-1   Started
  Container LibreChat-API         Started
  Container LibreChat-NGINX       Started

no crontab for ochat
Cron job added to run /home/ochat/purge_old_messages.py daily at 2:22 AM

stopping: docker compose -f /home/ochat/LibreChat/deploy-compose-ourchat.yml down
starting: docker compose -f /home/ochat/LibreChat/deploy-compose-ourchat.yml up -d
```

Now try to access your chat system, e.g. `https://ourchat.domain.edu`. If you encounter issues, please see the troubleshooting section below.

Budgeting

Since LibreChat does not support cost control at this time, we need to rely on AWS. The service `AWS Budgets` is a good start, but we quickly realize that this product is for alerting only and does not actually stop services from being used if there is a budget overrun. AWS will point out that there is another service called `AWS Budget Actions`, but it seems those are mainly triggering `AWS Lambda` scripts. They also lack the flexibility we need, which raises the question: why not just run an hourly Lambda script to check our spend?

Let's assume our monthly budget is $1000 (and not a penny more). If we disable the service after $950 has been spent, this may happen on the first day of the month, and then nobody will be able to use the system for the rest of the month. We could also set the budget to $32 per day, and with a maximum of 31 days in a month, we would never exceed $992 / month. But that would mean people cannot really do big things, and we want to support innovation. We do not want someone to be constrained by the $32 per day if only $100 has been spent this month.

So let's see if we can accumulate the budget. For example, if the system has not been used for 3 days, we would set the budget of the 4th day to $128 (`$32*4`). If someone came and used that entire amount on the 4th day, we would start fresh on the 5th day, and the budget would be $32 per day again. In an extreme case, assume the system has not been used for the first 30 days in a month. Then someone could come along and use $960 (`$32*30`) on the 31st day, and we would still be within budget.

How should this be implemented? Please see https://github.com/dirkpetersen/our-chat/issues/1 for further discussion.

API usage

LibreChat does not support API access (there is the RAG API feature, but that is a bit of a misnomer, as it actually requires the web ui). Instead, LibreChat is an aggregation point for many different APIs for users. Currently, LibreChat does not support complex cost control and all users of one server will access the same AWS account and there is only one single line item on the invoice and we cannot who was responsible for the cost. API users can automate their workflows which can lead to runaway costs. This means that API users should get their own AWS account with their own budget control and access the Bedrock API direcly. ~/our-chat/tests/bedrock-test.py may service as a resonable initial example that can list LLMs and ask a question.

Purging of old chats

Especially in healthcare environments we want to make sure that sensitive data does not reside on systems any longer than necessary. As of October 2024 LibreChat does not have the ability to purge data, however OurChat by default has a purge script activated, that deletes any messages and files older than X days. (60 days by default)

Updating Librechat

Update Librechat top the latest version and check the console output:

```
cd ~/LibreChat
docker compose -f ./deploy-compose-ourchat.yml down
git pull
docker compose -f ./deploy-compose-ourchat.yml pull
docker compose -f ./deploy-compose-ourchat.yml up
```

if everything works, we start LibreChat in Daemon mode:

```
docker compose -f ~/LibreChat/deploy-compose-ourchat.yml down
docker compose -f ~/LibreChat/deploy-compose-ourchat.yml up -d
```

should an upgrade fail, please follow these steps (Note: The chat history and uploaded files of all users will be lost)

```
docker compose -f ~/LibreChat/deploy-compose-ourchat.yml down
mv ~/LibreChat ~/LibreChat.fail.1
~/our-chat/install-librechat.sh
```

Disaster Recovery / Business Continuity / Emergency Operation

As the system only stores data temporarily, disaster recovery and business continuity procedures are limited to backing up 5 configuraton files / certificates in a secure place:

```
~/.env
~/librechat.yaml
~/nginx.conf
~/our-chat.pem
~/our-chat.pw
```

The install procedures at https://github.com/dirkpetersen/our-chat also serve as disaster recovery process as they are enabling you to restore service in less than 30 minutes, once you have the prerequisites met. (You may decide to have another VM already running to reduce the provisioning time).

While following the install procedures do not edit any of the 5 files but restore them from your secure backup location to `~/` (the root of the home direcotry of the service account) and copy them to the right locations in your LibreChat installation:

```
cp ~/.env ~/LibreChat/
cp ~/librechat.yaml ~/LibreChat/
cp ~/nginx.conf ~/LibreChat/client/
cp ~/our-chat.pem ~/LibreChat/client/ssl/
cp ~/our-chat.pw ~/LibreChat/client/ssl/
```

On-premises use cases

If you have use cases that are not permitted to use cloud services (e.g. patent issues), you can use on premises GPUs or even connect to your HPC cluster and use those resources all within the same LibreChat interface. Configure these custom endpoints in .env and in librechat.yaml

Longer term vision

In the future we may extend our chat eco-system, and add apps with addional capabilities but at this time LibreChat seems to meet our needs.

Troubleshooting

Get debug output

If something is not working, the first step is to enable debugging and bringing up the docker containers in non-daemon mode so that they are printing all logs to the console. Setup debug output, open ~/.env in editor

```
vi ~/.env && cp ~/.env ~/LibreChat/
```

and set `DEBUG_LOGGING=true`

Open up a second terminal, shutdown the containers and bring them up again, this time (`up without a -d`)

```
docker compose -f ~/LibreChat/deploy-compose-ourchat.yml down
docker compose -f ~/LibreChat/deploy-compose-ourchat.yml up
```

Check for error messages. When you are done, execute the `down` and then the `up -d`. command

cannot create docker group

In some cases your docker installation may not create a `docker` group in /etc/group. This is often because your IT department may have created a global docker group in their IAM system in order to manage their systems centrally. In that case the `groupadd docker` command will fail. There are 2 workaounds for this:

1. You can try editing `/etc/group` directly and add a line `docker:x:986:ochat` to it. The downside of this approach is, that you then have 2 different docker groups with different gidNumbers. While it should work without issues, it can be confusing, expecially if you are troubleshooting
2. You can create a new group (for example `ldocker`, e.g. for local docker), add the ochat user to it and ask docker to use it instead of the `docker` group. This requires these steps:

- edit /etc/docker/daemon.json

```
sudo echo -e '{\n  "group": "ldocker"\n}' >> /etc/docker/daemon.json
```

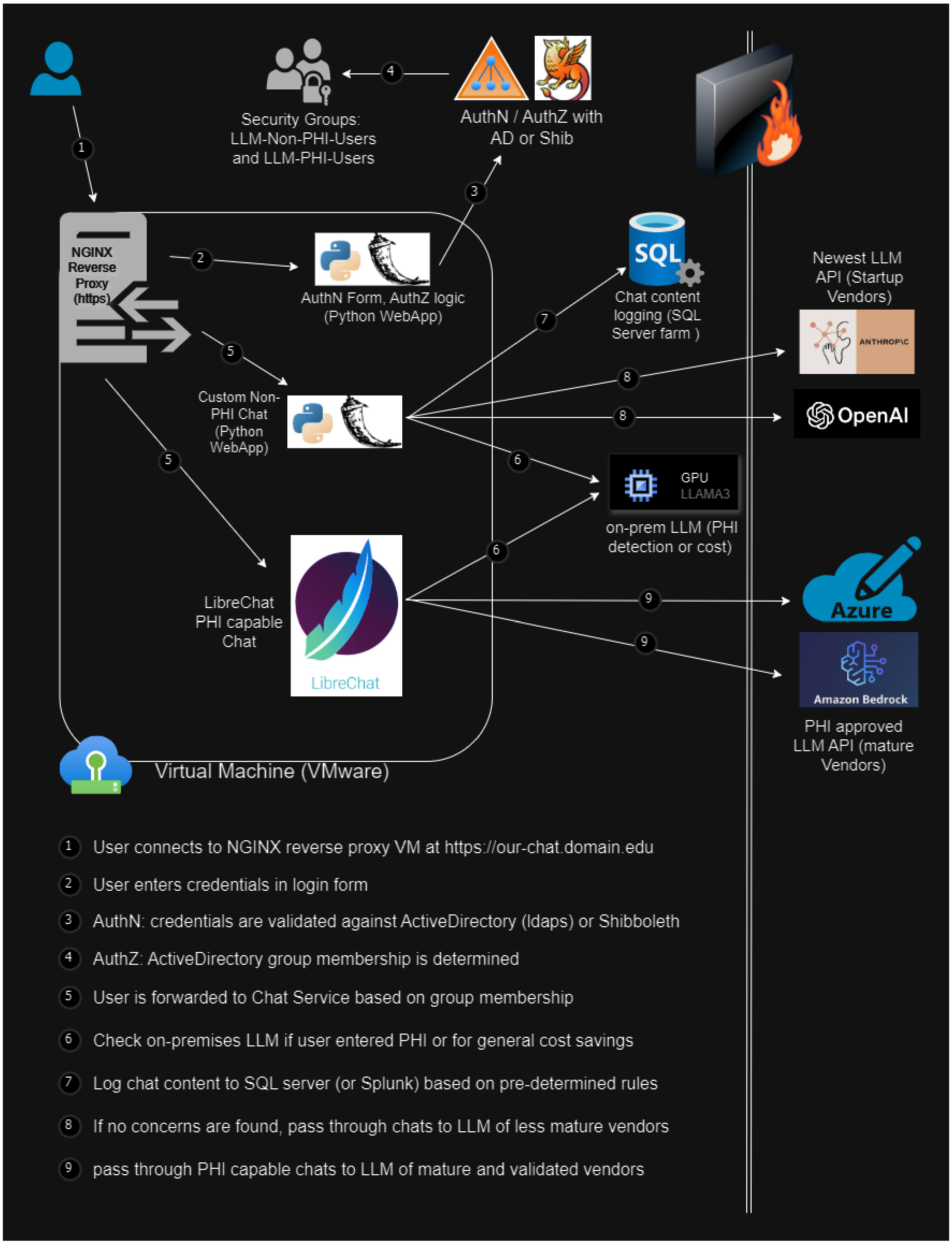
- run `systemctl edit docker.socket` and set the SocketGroup=ldocker

```
### Editing /etc/systemd/system/docker.socket.d/override.conf
### Anything between here and the comment below will become the new contents of the file

[Socket]
SocketGroup=ldocker

### Lines below this comment will be discarded
```

- restart the docker socket (not the docker daemon) and check that the socket has the correct group ownership


Security Groups: LLM-Non-PHI-Users and LLM-PHI-Users
AuthN / AuthZ with AD or Shib
NGINX Reverse Proxy (https)
AuthN Form, AuthZ logic (Python WebApp)
SQL
Chat content logging (SQL Server farm)
Newest LLM API (Startup Vendors)
ANTHROP\C
OpenAI
Custom Non-PHI Chat (Python WebApp)
GPU LLAMA3
on-prem LLM (PHI detection or cost)
LibreChat PHI capable Chat
LibreChat
Azure
Amazon Bedrock
PHI approved LLM API (mature Vendors)
Virtual Machine (VMware)
1
2
3
4
5
6
7
8
9

1. User connects to NGINX reverse proxy VM at https://our-chat.domain.edu
2. User enters credentials in login form
3. AuthN: credentials are validated against ActiveDirectory (ldaps) or Shibboleth
4. AuthZ: ActiveDirectory group membership is determined
5. User is forwarded to Chat Service based on group membership
6. Check on-premises LLM if user entered PHI or for general cost savings
7. Log chat content to SQL server (or Splunk) based on pre-determined rules
8. If no concerns are found, pass through chats to LLM of less mature vendors
9. pass through PHI capable chats to LLM of mature and validated vendors

Figure 2: our-chat-dark

```
systemctl restart docker.socket

ls -l /var/run/docker.sock
srw-rw----. 1 root ldocker 0 Oct  4 21:23 /var/run/docker.sock
```

If this is not working, please remove the docker packages using `dnf` and reinstall docker

I don't have root permissions

If your IT infrastructure team cannot give you `root` access to the virtual server you requested, you may still be able to get access to management features via sudo. Ask your sysadmin to run `visudo` and paste in the config below. Change yourusername to your actual user name:

```
yourusername (ALL) NOPASSWD: /usr/bin/dnf, /usr/bin/systemctl, /usr/bin/loginctl enable-linger *, /usr/bin/doc
yourusername (ALL) !/usr/bin/su -
yourusername (ALL) !/usr/bin/su - root
```

optionally you could also get these, for example if you need another NGINX reverse proxy

```
yourusername (ALL) NOPASSWD: /usr/bin/rpm, /usr/bin/vi /etc/nginx/nginx.conf, /usr/bin/vi /etc/nginx/sites-ava
```

Cleaning up docker

```
### cleanup as root
sudo systemctl stop docker
sudo rm -rf /var/lib/docker/*
sudo rm -rf /var/lib/containerd/*
sudo systemctl start docker
```