nginx-ldap-auth-service

Release 2.0.5

Caltech IMSS ADS

Jul 20, 2023

OVERVIEW

CHAPTER

ONE

INSTALLATION

Requirements

Python 3.x >= 3.8

To install the latest released version:

```
$ pip install nginx-ldap-auth-service
```

1.1 From Source

You can install `nginx-ldap-auth-service` from source just as you would install any other Python package:

```
$ pip install git+https://github.com/caltechads/nginx-ldap-auth-service.git
```

This will allow you to keep up to date with development on GitHub:

```
$ pip install -U git+https://github.com/caltechads/nginx-ldap-auth-service.git
```

1.2 From Docker Hub

You can also run `nginx-ldap-auth-service` from Docker Hub:

```
$ docker pull caltechads/nginx-ldap-auth-service:latest
$ docker run \
    -d \
    -p 8888:8888 \
    -e LDAP_URI=ldap://ldap.example.com \
    -e LDAP_BASEDN=dc=example,dc=com \
    -e LDAP_BINDDN=cn=admin,dc=example,dc=com \
    -e LDAP_PASSWORD=secret \
    caltechads/nginx-ldap-auth-service
```

Or use `docker-compose`. Create a `docker-compose.yml` file with the following contents:

```
version: '3.7'
services:
  nginx_ldap_auth_service:
    image: caltechads/nginx-ldap-auth-service:latest
```

(continues on next page)

(continued from previous page)

```
    hostname: nginx-ldap-auth-service
    container_name: nginx-ldap-auth-service
    ports:
      - 8888:8888
    environment:
      - LDAP_URI=ldap://ldap.example.com
      - LDAP_BASEDN=dc=example,dc=com \
      - LDAP_BINDDN=cn=admin,dc=example,dc=com
      - LDAP_PASSWORD=secret
```

Then run:

```
$ docker-compose up -d
```

CHAPTER

TWO

RUNNING NGINX_LDAP_AUTH_SERVICE

You can run `nginx_ldap_auth_service` as daemon running alongside your nginx process on your web server, or as a Docker sidecar container.

2.1 nginx-ldap-auth command line

After installing `nginx_ldap_auth_service` you will have access to the command line script `nginx-ldap-auth`.

Basic usage:

```
$ nginx-ldap-auth start [OPTIONS]
```

Positional and keyword arguments can also be passed, but it is recommended to load configuration from environment variables or with the `--env-file` option rather than the command line.

2.1.1 Arguments

- `-env-file FILE` - Specify an environment file to use to configure `nginx-ldap-auth-service`. This is the recommended way to configure `nginx-ldap-auth-service`. Note that you can't configure any of the below options with an environment file; those environment variables if used must be set in the shell environment.
- `-h BIND, --host=BIND` - Specify an IP address to which to bind. Defaults to the value of the `HOST` environment variable or `0.0.0.0`
- `-p PORT, --port=PORT` - Specify an port to which to bind. Defaults to the value of the `PORT` environment variable or `8888`
- `-w WORKERS, --workers=WORKERS` - Number of worker processes. Defaults to the value of the `WORKERS` environment variable, or 1 if neither is set.
- `--keyfile=KEYFILE` - Specify a keyfile to use for SSL. Defaults to the value of the `SSL_KEYFILE` environment variable, or `/certs/server.key`. `/certs/server.key`.
- `--certfile=CERTFILE` - Specify a certfile to use for SSL. Defaults to the value of the `SSL_CERTFILE` environment variable, or `/certs/server.crt`.

2.2 Deployments

2.2.1 Docker sidecar container

The preferred way to run `nginx_ldap_auth_service` is as a Docker sidecar container. This allows you to run `nginx_ldap_auth_service` alongside your nginx container, and have nginx talk to it when it needs to perform authentication or authorization.

Here is an example `docker-compose.yml` file that runs `nginx` and `nginx_ldap_auth_service`:

```
version: '3'

services:
  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - "8443:443"
    volumes:
      - ./etc/nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./etc/nginx/certs:/certs
    depends_on:
      - nginx_ldap_auth_service
    links:
      - nginx_ldap_auth_service

  nginx_ldap_auth_service:
    image: caltechads/nginx-ldap-auth-service:latest
    hostname: auth-service
    container_name: nginx-ldap-auth-service
    ports:
      - "8888:8888"
    environment:
      - LDAP_URI=ldap://ldap.example.com
      - LDAP_BASEDN=dc=example,dc=com
      - LDAP_BINDDN=cn=readonly,dc=example,dc=com
      - LDAP_PASSWORD=readonly
      ...
```

Kubernetes/AWS Elastic Container Service deployment details are left as an exercise for the reader.

2.2.2 As a daemon

`nginx-ldap-auth-service` runs only in the foreground and it writes its logs to stdout, so if you want to run it as a daemon you will need to use a process manager like `supervisord` or `systemd` that can put it in the background and capture its output.

Here is an example of running it with `supervisord`. First make the log folder:

```
$ mkdir -p /var/log/nginx-ldap-auth-service
$ chown $supervisor_user /var/log/nginx-ldap-auth-service
```

Then configure `supervisord` to run `nginx-ldap-auth-service` as a daemon. Below we've configured it to read its configuration from an environment file. See *nginx-ldap-auth command line* and *Environment*) for details about the

environment variables that can be set in the environment file.

```
[program:nginx-ldap-auth-service]
command=/path/to/nginx-ldap-auth --env-file /path/to/env-file
directory=/tmp
childlogdir=/var/log/nginx-ldap-auth-service
stdout_logfile=/var/log/nginx-ldap-auth-service/stdout.log
stdout_logfile_maxbytes=1MB
redirect_stderr=true
user=nobody
autostart=true
autorestart=true
redirect_stderr=true
```

CHAPTER

THREE

CONFIGURATION OVERVIEW

Important: This page deals with configuring `nginx-ldap-auth-service`. For configuring `nginx` to use `nginx-ldap-auth-service`, see *Configuring nginx*.

`nginx-ldap-auth-service` reads configuration from three places, in decreasing order of precedence:

1. Command line options for `nginx-ldap-auth start`
2. headers set in the location blocks of the `nginx` config file
3. the environment

Not all configuration options are available in all places.

Note: To print your resolved configuration when using the command line, you can run the following command:

```
$ nginx-ldap-auth settings
```

3.1 Command Line

If an option is specified on the command line, it overrides all other values that may have been specified in the app specific environment variables. configuration file. Not all `nginx-ldap-auth-service` settings are available to be set from the command line. To see the full list of command line settings you can do the usual:

```
$ nginx-ldap-auth start --help
```

3.2 nginx Header Configuration

If an option is specified in the `nginx` configuration file, it overrides the associated setting in `nginx-ldap-auth-service`.

You can set the following headers in your nginx configuration to configure `nginx-ldap-auth-service` on a per `nginx` server basis. You might do this if you have multiple `nginx` servers all using the same `nginx-ldap-auth-service` instance, but want to configure them differently.

Note: You can only set the following headers in the `location` blocks that proxy to `nginx-ldap-auth-service`. If you set them in the `server` block, they will be ignored.

X-Auth-Realm

The title for the login form. This goes in the `location` block for the `/auth` location. Defaults to the value of *nginx_ldap_auth.settings.Settings.auth_realm* for the `nginx-ldap-auth-service` instance.

Example:

```
location /auth {
    proxy_pass http://nginx-ldap-auth-service:8888/auth;
    proxy_set_header X-Auth-Realm "My Login Form";
}
```

X-Cookie-Name

The name of the session cookie. This goes in the `location` block for the `/auth` and `/check-auth` locations. Defaults to the value of *nginx_ldap_auth.settings.Settings.cookie_name* for the `nginx-ldap-auth-service` instance.

Changing the cookie name with `X-Cookie-Name` implies some other `nginx` configuration changes also, so all the highlighted lines below are things you need to change if you change the cookie name.

Example:

```
location /auth {
    proxy_pass http://nginx-ldap-auth-service:8888/auth;
    proxy_set_header X-Cookie-Name "mycookie";

    # other lines omitted for brevity
}

location /check-auth {
    proxy_pass http://nginx-ldap-auth-service:8888/check;

    # Cache our auth responses for 10 minutes so that we're not
    # hitting the auth service on every request.
    proxy_cache auth_cache;
    proxy_cache_valid 200 10m;

    # other lines omitted for brevity

    proxy_set_header X-Cookie-Name "mycookie";
    proxy_set_header Cookie mycookie=$cookie_mycookie;
    proxy_cache_key "$http_authorization$cookie_mycookie";
}
```

If you're not doing any caching, you can ignore the cache related lines above.

X-Cookie-Domain

The domain for the session cookie. This goes in the `location` block for the `/auth` and `/check-auth` locations. Defaults to the value of *nginx_ldap_auth.settings.Settings.cookie_domain* for the `nginx-ldap-auth-service` instance.

Example:

```
location /auth {
    proxy_pass http://nginx-ldap-auth-service:8888/auth;
    proxy_set_header X-Cookie-Domain ".example.com";

    # other lines omitted for brevity
}

location /check-auth {
    proxy_pass http://nginx-ldap-auth-service:8888/check;

    # other lines omitted for brevity

    proxy_set_header X-Cookie-Domain ".example.com";
}
```

3.3 Environment

You can either export the appropriate variables directly into your shell environment, or you can use an environment file and specify it with the `--env-file` option to `nginx-ldap-auth start`.

The following environment variables are available to configure `nginx-ldap-auth-service`:

Important: You must set at least these variables to localize to your organization:

- *LDAP_URI*
- *LDAP_BINDDN*
- *LDAP_PASSWORD*,
- *LDAP_BASEDN*
- *SECRET_KEY*.

You should also look at these variables to see whether their defaults work for you:

- *LDAP_USERNAME_ATTRIBUTE*
- *LDAP_FULL_NAME_ATTRIBUTE*
- *LDAP_GET_USER_FILTER*
- *LDAP_AUTHORIZATION_FILTER*
- *AUTH_REALM*
- *SESSION_MAX_AGE*

3.3.1 Web Server

These settings configure the web server that `nginx-ldap-auth-service` runs, `uvicorn`.

HOSTNAME

The hostname to listen on. Defaults to `0.0.0.0`.

PORT

The port to listen on. Defaults to `8888`.

SSL_KEYFILE

The path to the SSL key file. Defaults to `/certs/server.key`.

SSL_CERTFILE

The path to the SSL certificate file. Defaults to `/certs/server.crt`.

WORKERS

The number of worker processes to spawn. Defaults to `1`.

DEBUG

Set to `1` or `True` to enable debug mode. Defaults to `False`.

3.3.2 Login form and sessions

These settings configure the login form and session handling.

AUTH_REALM

The title for the login form. Defaults to `Restricted`.

COOKIE_NAME

The name of the cookie to use for the session. Defaults to `nginxauth`.

COOKIE_DOMAIN

The domain for the cookie to use for the session. Defaults to no domain.

SESSION_MAX_AGE

How many seconds a session should last after first login. Defaults to `0`, no expiry. If *USE_ROLLING_SESSIONS* is `True`, this value is used to reset the session lifetime on every request.

USE_ROLLING_SESSIONS

If `True`, session lifetime will be reset to *SESSION_MAX_AGE* on every request. Defaults to `False`.

SECRET_KEY

Required The secret key to use for the session. Defaults to `SESSION_SECRET`.

SESSION_BACKEND

The session backend to use. Defaults to `memory`. Valid options are `memory` and `redis`. If you choose `redis`, you must also set *REDIS_URL*.

REDIS_URL

The DSN to the Redis server. See *nginx_ldap_auth.settings.Settings.redis_url* for details on the format of the DSN.

Defaults to `None`

REDIS_PREFIX

The prefix to use for Redis keys. Defaults to `nginx_ldap_auth`.

3.3.3 LDAP

These settings configure the LDAP server to use for authentication.

LDAP_URI

Required. The URL to the LDAP server. Defaults to `ldap://localhost`.

LDAP_BINDDN

Required. The DN to use to bind to the LDAP server for doing our user and authorization searches.

LDAP_PASSWORD

Required. The password to use to with *LDAP_BINDDN* to bind to the LDAP server for doing our user and authorization searches.

LDAP_STARTTLS

Set to `1` or `True` to enable STARTTLS on our LDAP connections. Defaults to `False`.

LDAP_DISABLE_REFERRALS

Set to `1` or `True` to disable LDAP referrals. Defaults to `False`.

LDAP_BASEDN

Required The base DN to use for our LDAP searches.

LDAP_USERNAME_ATTRIBUTE

The LDAP attribute to use for the username. Defaults to `uid`.

LDAP_FULL_NAME_ATTRIBUTE

The LDAP attribute to use for the full name. Defaults to `cn`.

LDAP_GET_USER_FILTER

The LDAP search filter to use when searching for users. Defaults to `{username_attribute}={username}`, where `{username_attribute}` is the value of *LDAP_USERNAME_ATTRIBUTE* and `{username}` is the username provided by the user. See *nginx_ldap_auth.settings.Settings.ldap_get_user_filter* for more details.

The filter will within the base DN given by *LDAP_BASEDN* and with scope of `SUBTREE`.

LDAP_AUTHORIZATION_FILTER

The LDAP search filter to use when determining if a user is authorized to login. for authorizations. Defaults to no filter, meaning all users are authorized if they exist in LDAP. See *nginx_ldap_auth.settings.Settings.ldap_authorization_filter* for more details.

The filter will within the base DN given by *LDAP_BASEDN* and with scope of `SUBTREE`.

LDAP_TIMEOUT

The maximum number of seconds to wait when acquiring a connection to the LDAP server. Defaults to `15`.

LDAP_MIN_POOL_SIZE

The minimum number of connections to keep in the LDAP connection pool. Defaults to `1`.

LDAP_MAX_POOL_SIZE

The maximum number of connections to keep in the LDAP connection pool. Defaults to `30`.

LDAP_POOL_CONNECTION_LIFETIME_SECONDS

The maximum number of seconds to keep a connection in the LDAP connection pool. Defaults to `20`.

CHAPTER

FOUR

CONFIGURING NGINX

This page describes how to configure nginx to use `nginx-ldap-auth-service` to password protect your site using LDAP.

4.1 ngx_http_auth_request_module

`nginx-ldap-auth-service` requires your `nginx` to have the `ngx_http_auth_request_module` to do its work. To see if your version of nginx has that installed, do `nginx -V` and look for `--with-http_auth_request_module`:

```
$ nginx -V
nginx version: nginx/1.23.4
built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
built with OpenSSL 1.1.1n  15 Mar 2022
TLS SNI support enabled
configure arguments: --prefix=/etc/nginx --sbin-path=/usr/sbin/nginx --modules-path=/usr/
↪lib/nginx/modules --conf-path=/etc/nginx/nginx.conf --error-log-path=/var/log/nginx/
↪error.log --http-log-path=/var/log/nginx/access.log --pid-path=/var/run/nginx.pid --
↪lock-path=/var/run/nginx.lock --http-client-body-temp-path=/var/cache/nginx/client_
↪temp --http-proxy-temp-path=/var/cache/nginx/proxy_temp --http-fastcgi-temp-path=/var/
↪cache/nginx/fastcgi_temp --http-uwsgi-temp-path=/var/cache/nginx/uwsgi_temp --http-
↪scgi-temp-path=/var/cache/nginx/scgi_temp --user=nginx --group=nginx --with-compat --
↪with-file-aio --with-threads --with-http_addition_module --with-http_auth_request_
↪module --with-http_dav_module --with-http_flv_module --with-http_gunzip_module --with-
↪http_gzip_static_module --with-http_mp4_module --with-http_random_index_module --with-
↪http_realip_module --with-http_secure_link_module --with-http_slice_module --with-http_
↪ssl_module --with-http_stub_status_module --with-http_sub_module --with-http_v2_module␣
↪--with-mail --with-mail_ssl_module --with-stream --with-stream_realip_module --with-
↪stream_ssl_module --with-stream_ssl_preread_module --with-cc-opt='-g -O2 -ffile-prefix-
↪map=/data/builder/debuild/nginx-1.23.4/debian/debuild-base/nginx-1.23.4=. -fstack-
↪protector-strong -Wformat -Werror=format-security -Wp,-D_FORTIFY_SOURCE=2 -fPIC' --
↪with-ld-opt='-Wl,-z,relro -Wl,-z,now -Wl,--as-needed -pie'
```

4.2 nginx.conf

There four bits to this configuration:

1. Configuring your site's `location` block to use `auth_request` and to redirect any unauthenticated requests to the `nginx-ldap-auth-service` login page.
2. Configuring a `location` for `nginx-ldap-auth-service` to use to authenticate and logout users.
3. Configuring the `location` that `auth_request` will use to see if a user is authenticated.
4. (optional) Configuring a cache for the `auth_request` location so that we don't have to hit the auth service on every request.

Below is a minimal example configuration for a site that uses LDAP to authenticate users that want to access the site whose root page is `/`.

Things to note:

- We serve all the login related views in an `server` block that is HTTPS only. This is because we don't want to send the user's password over the wire in plain text.
- In the `proxy_pass` lines below, we're naming the server that hosts the auth service `nginx_ldap_auth_service` on port 8888. Change this to whatever hostname and port the service answers on in your architecture.
- The login and logout related views are served by `nginx-ldap-auth-service` and always use the paths `/auth/login` and `/auth/logout`, and those paths are hard-coded into the login form; you can't change them. The `/auth` location handles the proxying of those paths to `nginx-ldap-auth-service`.
- See *nginx Header Configuration* for information on how to configure `nginx-ldap-auth-service` behavior using custom headers.

```
user nginx;
worker_processes auto;

error_log  /dev/stderr info;
pid /tmp/nginx.pid;

events {
  worker_connections 1024;
}

http {
  proxy_cache_path /tmp/nginx-cache keys_zone=auth_cache:10m;
  include /etc/nginx/mime.types;
  default_type application/octet-stream;

  server {
    listen 443 ssl http2;

    ssl_certificate /certs/localhost.crt;
    ssl_certificate_key /certs/localhost.key;

    location / {
        auth_request /check-auth;
        root   /usr/share/nginx/html;
        index  index.html index.htm;
```

(continues on next page)

(continued from previous page)

```
        # If the auth service returns a 401, redirect to the login page.
        error_page 401 =200 /auth/login?service=$request_uri;
    }

    location /auth {
        proxy_pass https://nginx_ldap_auth_service:8888/auth;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    location /check-auth {
        internal;
        proxy_pass https://nginx_ldap_auth_service:8888/check;

        # Ensure that we don't pass the user's headers or request body to
        # the auth service.
        proxy_pass_request_headers off;
        proxy_pass_request_body off;
        proxy_set_header Content-Length "";

        # We use the same auth service for managing the login and logout and
        # checking auth.  The SessionMiddleware, which is used for all requests,
        # will always be trying to set cookies even on our /check path.  Thus we
        # need to ignore the Set-Cookie header so that nginx will cache the
        # response.  Otherwise, it will think this is a dynamic page that
        # shouldn't be cached.
        proxy_ignore_headers "Set-Cookie";
        proxy_hide_header "Set-Cookie";

        # Cache our auth responses for 10 minutes so that we're not
        # hitting the auth service on every request.
        proxy_cache auth_cache;
        proxy_cache_valid 200 10m;

        proxy_set_header Cookie nginxauth=$cookie_nginxauth;
        proxy_cache_key "$http_authorization$cookie_nginxauth";
    }
  }
}
```

CHAPTER

FIVE

CONTRIBUTING

5.1 Instructions for contributors

In order to make a clone of the Github repo:

```
$ git clone https://github.com/caltechads/nginx-ldap-auth-service.git
```

Workflow is pretty straightforward:

1. Make sure you are reading the latest version of this document.
2. Setup your machine with the required development environment
3. Checkout a new branch, named for yourself and a summary of what you're trying to accomplish.
4. Make a change
5. Make sure all tests passed
6. Commit changes to your branch
7. Merge your changes into master and push.

5.2 Preconditions for working on nginx-ldap-auth-service

You'll need Python 3.11.3 installed. We recommend using `pyenv` to manage your Python installations. You'll also need `pip` and `wheel` installed.

```
$ cd nginx-ldap-auth-service
$ pyenv virtualenv 3.11.3 nginx-ldap-auth-service
$ pip install --upgrade pip wheel
$ pyenv local nginx-ldap-auth-service
```

After that please install libraries required for development:

```
$ pip install -r requirements.dev.txt
```

5.3 Precondiions for running the docker-compose stack in development

Since `nginx-ldap-auth-service` authenticates against an LDAP or Active Directory service, you will need to provide one. The LDAP/AD server you use needs these features:

- It must support `STARTTLS`
- It must support `LDAPv3`
- It must support `SIMPLE` bind
- It must have an account that with sufficient privileges to bind to the LDAP/AD server with a password and search for users.

5.4 Prepare the docker environment

Now copy in the Docker environment file to the appropriate place on your dev box:

```
$ cp etc/environment.txt .env
```

Edit `.env` replace these with settings appropriate for your LDAP/AD server:

- `__LDAP_URI__`
- `__LDAP_BINDDN__`
- `__LDAP_BASEDN__`
- `__LDAP_PASSWORD__`

5.5 Build the Docker image

```
$ make build
```

5.6 Run the stack

```
$ make dev
```

This will bring up the full dev stack:

- `nginx`
- `nginx-ldap-auth-service`

If you want to bring up a redis instance for session storage, you can do that by uncommenting the `redis` service in `docker-compose.yml` and adding these two settings to the `environment` section of the `nginx_ldap_auth_service` service:

```
- SESSION_BACKEND=redis
- REDIS_URL=redis://redis:6379/0
```

5.7 Use your dev environment

You should how be able to browse to https://localhost:8443/ and be redirected to the login page.

CHAPTER

SIX

VIEWS

async nginx_ldap_auth.app.main.**login**(*request: Request*, *service: str = '/'*)

If the user is already logged in, redirect to the URI named by the `service`, query paremeter, defaulting to `/` if that is not present. Otherwise, render the login page.

If the header `X-Auth-Realm` is set, use that as the title for the login page. Otherwise use *nginx_ldap_auth.settings.Settings.auth_realm*.

Parameters
request – The request object

Keyword Arguments
service – redirect the user to this URL after successful login

async nginx_ldap_auth.app.main.**login_handler**(*request: Request*)

Process our user's login request. If authentication is successful, redirect to the value of the `service` hidden input field on our form.

If authentication fails, display the login form again.

If the header `X-Auth-Realm` is set, use that as the title for the login page. Otherwise use *nginx_ldap_auth.settings.Settings.auth_realm*.

Parameters
request – The request object

async nginx_ldap_auth.app.main.**logout**(*request: Request*)

Log the user out and redirect to the login page.

Parameters
request – The request object

async nginx_ldap_auth.app.main.**check_auth**(*request: Request*, *response: Response*)

Ensure the user is still authorized. If the user is authorized, return 200 OK, otherwise return 401 Unauthorized.

The user is authorized if the cookie exists, the session the cookie refers to exists, and the `username` key in the settings is set.

Additionally, the user must still exist in LDAP, and if *nginx_ldap_auth.settings.Settings.ldap_authorization_filter* is not `None`, the user must also match the filter.

Parameters

- **request** – The request object
- **response** – The response object

CHAPTER

SEVEN

MODELS

class nginx_ldap_auth.app.models.**UserManager**

model

The model class for users

alias of *User*

settings

The application settings

pool: *TimeLimitedAIOConnectionPool* | None

The LDAP connection pool

client() → LDAPClient

Return a new LDAP client instance.

If *nginx_ldap_auth.settings.Settings.ldap_starttls* is True, the client will be configured to use TLS.

async **create_pool**() → None

Create the LDAP connection pool and save it as *pool*.

async **authenticate**(*username: str*, *password: str*) → bool

Authenticate a user against the LDAP server.

Parameters

- **username** – the username to authenticate
- **password** – the password to authenticate with

Raises

LDAPError – if an error occurs while communicating with the LDAP server

Returns

True if the user is authenticated, False otherwise

async **exists**(*username: str*) → bool

Return True if the user exists in the LDAP directory, False otherwise.

Parameters

username – the username to check

Raises

- **LDAPError** – if an error occurred while communicating with the LDAP server

- **AuthenticationError** – if the LDAP server rejects the credentials of *nginx_ldap_auth.settings.Settings.ldap_binddn* and *nginx_ldap_auth.settings.Settings.ldap_password*

Returns
`True` if the user exists in the LDAP directory, `False` otherwise

async is_authorized(*username: str*) → bool

Test whether the user is authorized to log in. This is done by performing an LDAP search using the filter specified in *nginx_ldap_auth.settings.Settings.ldap_authorization_filter*. If that setting is `None`, the user is considered authorized.

Parameters
username – the username to check

Raises

- **LDAPError** – if an error occurred while communicating with the LDAP server
- **AuthenticationError** – if the LDAP server rejects the credentials of *nginx_ldap_auth.settings.Settings.ldap_binddn* and *nginx_ldap_auth.settings.Settings.ldap_password*

Returns
`True` if the user is authorized to log in, `False` otherwise.

async get(*username: str*) → *User* | None

Get a user from the LDAP directory, and return it as a *User*. When getting the user, we will use the LDAP search filter specified in *nginx_ldap_auth.settings.Settings.ldap_get_user_filter*.

Parameters
username – the username for which to get user information

Raises

- **LDAPError** – if an error occurred while communicating with the LDAP server
- **AuthenticationError** – if the LDAP server rejects the credentials of *nginx_ldap_auth.settings.Settings.ldap_binddn* and *nginx_ldap_auth.settings.Settings.ldap_password*

Returns
The user information as a *User* instance, or `None` if the user is not returned by the LDAP search filter

async cleanup() → None

Close the LDAP connection pool.

class nginx_ldap_auth.app.models.**User**(*, *uid: str*, *full_name: str*)

uid: str

The username of the user.

full_name: str

The full name of the user. We really only use this for logging.

async authenticate(*password: str*) → bool

Authenticate this user against the LDAP server.

Parameters
password – the password to authenticate with

Returns

`True` if the user is authenticated, `False` otherwise

CHAPTER

EIGHT

LDAP

class nginx_ldap_auth.ldap.**TimeLimitedAIOLDAPConnection**(*client: LDAPClient*, *expires: int = 20*, *loop=None*)

A time-limited LDAP connection. This allows us to have a connection pool that will close connections after a certain amount of time.

Parameters
client – The LDAP client.

Keyword Arguments

- **expires** – The number of seconds after which the connection will expire.
- **loop** – The asyncio event loop.

property is_expired: bool

abandon()

Abandon ongoing operation associated with the given message id.

add(*entry: LDAPEntry*, *timeout: float | None = None*) → Any

Add new LDAPEntry to the LDAP server.

close()

Close connection with the LDAP Server.

closed

Connection is closed

async delete(*dname*, *timeout=None*, *recursive=False*)

Delete an LDAPEntry with the given distinguished name.

fileno()

Get the socket descriptor that belongs to the connection.

async get_result(*msg_id*, *timeout=None*)

Poll the status of the operation associated with the given message id from LDAP server.

is_async

Asynchronous connection

modify_password(*user: str | LDAPDN | None = None*, *new_password: str | None = None*, *old_password: str | None = None*, *timeout: float | None = None*) → Any

Modify password for the user.

open(*timeout=None*)

Open connection with the LDAP Server.

paged_search(*base: str | LDAPDN | None = None*, *scope: LDAPSearchScope | int | None = None*, *filter_exp: str | None = None*, *attrlist: List[str] | None = None*, *timeout: float | None = None*, *sizelimit: int = 0*, *attrsonly: bool = False*, *sort_order: List[str] | None = None*, *page_size: int = 1*) → Any

search(*base: str | LDAPDN | None = None*, *scope: LDAPSearchScope | int | None = None*, *filter_exp: str | None = None*, *attrlist: List[str] | None = None*, *timeout: float | None = None*, *sizelimit: int = 0*, *attrsonly: bool = False*, *sort_order: List[str] | None = None*) → Any

Search for LDAP entries.

virtual_list_search(*base: str | LDAPDN | None = None*, *scope: LDAPSearchScope | int | None = None*, *filter_exp: str | None = None*, *attrlist: List[str] | None = None*, *timeout: float | None = None*, *sizelimit: int = 0*, *attrsonly: bool = False*, *sort_order: List[str] | None = None*, *offset: int = 1*, *before_count: int = 0*, *after_count: int = 0*, *est_list_count: int = 0*, *attrvalue: str | None = None*) → Any

whoami(*timeout: float | None = None*) → Any

LDAPv3 Who Am I operation.

class nginx_ldap_auth.ldap.**TimeLimitedAIOConnectionPool**(*settings: Settings*, *client: LDAPClient*, *minconn: int = 1*, *maxconn: int = 10*, *loop=None*, ***kwargs: Any*)

A pool of time-limited LDAP connections. This allows us to have relatively fresh connections to our LDAP server while not having to create a new connection for every request.

Parameters

- **settings** – The application settings.
- **client** – The LDAP client.

Keyword Arguments

- **minconn** – The minimum number of connections to keep in the pool.
- **maxconn** – The maximum number of connections to keep in the pool.
- **loop** – The asyncio event loop.

async get() → AIOLDAPConnection

Get a connection from the pool. If a connection has expired, close it and create a new connection, then return the new connection.

Raises

- **ClosedPool** – The pool has not been initialized.
- **EmptyPool** – There are no connections in the pool.

Returns

A connection from the pool.

async close() → None

Close the pool and all of its managed connections.

property closed: bool

Read-only property that will be True when the connection pool has been closed.

property empty: bool

Read-only property that will be True when the connection pool has no free connection to use.

property idle_connection: int

the number of idle connection.

property max_connection: int

The maximal number of connections that the pool can have.

async open() → None

Open the connection pool by initialising the minimal number of connections.

async put(*conn: AIOLDAPConnection*) → None

Put back a connection to the connection pool. The caller is allowed to close the connection (if, for instance, it is in an error state), in which case it's not returned to the pool and a subsequent get will grow the pool if needed.

Parameters
conn (*LDAPConnection*) – the connection managed by the pool.

Raises

- **ClosedPool** – when the method is called on a closed pool.
- **PoolError** – when tying to put back an object that's not managed by this pool.

property shared_connection: int

The number of shared connections.

spawn(**args: Any*, ***kwargs: Any*) → AsyncGenerator[AIOLDAPConnection, None]

Context manager method that acquires a connection from the pool and returns it on exit. It also opens the pool if it hasn't been opened before.

Params *args
the positional arguments passed to `bonsai.pool.ConnectionPool.get`.

Params **kwargs
the keyword arguments passed to `bonsai.pool.ConnectionPool.get`.

CHAPTER

NINE

MIDDLEWARE

class nginx_ldap_auth.app.middleware.**SessionMiddleware**(*app: Callable[[MutableMapping[str, Any], Callable[[], Awaitable[MutableMapping[str, Any]]], Callable[[MutableMapping[str, Any]], Awaitable[None]]], Awaitable[None]]*, *store: SessionStore*, *lifetime: int | timedelta = 0*, *rolling: bool = False*, *cookie_name: str = 'session'*, *cookie_same_site: str = 'lax'*, *cookie_https_only: bool = True*, *cookie_domain: str | None = None*, *cookie_path: str | None = None*, *serializer: Serializer | None = None*)

Bases: `SessionMiddleware`

Override the `starsession.SessionMiddleware` to allow us to set the cookie name and domain via the `X-Cookie-Name` and `X-Cookie-Domain` headers, respectively. If those headers are not present, the values from the constructor are used.

We need this so that we can set the cookie name and domain dynamically based on the request. This is necessary because we may have multiple nginx severs that use a single `nginx_ldap_auth` server for authentication.

Note: Unfortunately, the :py:meth:`__call__` method is monolithic in the superclass, so we have to re-implement it here in is entirety to do what we want to do.

COOKIE_NAME_HEADER: Final[str] = 'X-Cookie-Name'

COOKIE_DOMAIN_HEADER: Final[str] = 'X-Cookie-Domain'

CHAPTER

TEN

SETTINGS

class nginx_ldap_auth.settings.**Settings**(*_case_sensitive: bool | None = None*, *_env_prefix: str | None = None*, *_env_file: DotenvType | None = PosixPath('.')*, *_env_file_encoding: str | None = None*, *_env_nested_delimiter: str | None = None*, *_secrets_dir: str | Path | None = None*, ***, *debug: bool = False*, *loglevel: Literal['NOTSET', 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL'] = 'INFO'*, *log_type: Literal['json', 'text'] = 'text'*, *auth_realm: str = 'Restricted'*, *cookie_name: str = 'nginxauth'*, *cookie_domain: str | None = None*, *secret_key: str*, *session_max_age: int = 0*, *use_rolling_session: bool = False*, *session_backend: Literal['redis', 'memory'] = 'memory'*, *redis_url: Url[Url] | None = None*, *redis_prefix: str = 'nginx_ldap_auth.'*, *ldap_uri: str*, *ldap_binddn: str*, *ldap_password: str*, *ldap_starttls: bool = True*, *ldap_disable_referrals: bool = False*, *ldap_basedn: str*, *ldap_username_attribute: str = 'uid'*, *ldap_full_name_attribute: str = 'cn'*, *ldap_get_user_filter: str = '{username_attribute}={username}'*, *ldap_authorization_filter: str | None = None*, *ldap_timeout: int = 15*, *ldap_min_pool_size: int = 1*, *ldap_max_pool_size: int = 30*, *ldap_pool_connection_lifetime_seconds: int = 20*, *sentry_url: str | None = None*)

debug: `bool`

FastAPI debug mode

loglevel: `Literal['NOTSET', 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL']`

Default log level. Choose from any of the standard Python log levels.

log_type: `Literal['json', 'text']`

What format should we log in? Valid values are `json` and `text`

auth_realm: `str`

Use this as the title for the login form, to give a hint to the user as to what they're logging into

cookie_name: `str`

The name of the cookie to set when a user authenticates

cookie_domain: `str | None`

The domain to use for our session cookie, if any.

secret_key: `str`

The secret key to use for session cookies

session_max_age: int

The maximum age of a session cookie in seconds

use_rolling_session: bool

Reset the session lifetime to *session_max_age* every time the user accesses the protected site

session_backend: Literal['redis', 'memory']

either `redis` or `memory`

Type
Session type

redis_url: Url[Url] | None

If using the Redis session backend, the DSN on which to connect to Redis.

A fully specified Redis DSN looks like this:

```
redis://[username][:password]@host:port/db
```

- The username is only necessary if you are using role-based access controls on your Redis server. Otherwise the password is sufficient if you have a server password for your Redis server.
- If you don't specify a database, `0` is used.
- If you don't specify a password, no password is used.
- If you don't specify a port, `6379` is used.

redis_prefix: str

If using the Redis session backend, the prefix to use for session keys

ldap_uri: str

The URI via which to connect to LDAP

ldap_binddn: str

The DN as which to bind to LDAP

ldap_password: str

The password to use when binding to LDAP when doing our searches

ldap_starttls: bool

Whether to use TLS when connecting to LDAP

ldap_disable_referrals: bool

Whether to disable LDAP referrals

ldap_basedn: str

The base DN under which to perform searches

ldap_username_attribute: str

The LDAP attribute to use as the username when searching for a user

ldap_full_name_attribute: str

The LDAP attribute to use as the full name when getting search results

ldap_get_user_filter: str

The LDAP search filter to use when searching for a user. This should be a valid LDAP search filter. The search will be a SUBTREE search with the base DN of *ldap_basedn*.

You may use these replacement fields in the filter:

- {username_attribute}: the value of *Settings.ldap_username_attribute*
- {username_full_name_attribute}: the value of *Settings.ldap_full_name_attribute*

Use {username} in the search filter as the placeholder for the username supplied by the user from the login form.

ldap_authorization_filter: str | None

The LDAP search filter to use to determine whether a user is authorized. This should a valid LDAP search filter. If this is None, all users who can successfully authenticate will be authorized. If this is not None, the search with this filter must return at least one result for the user to be authorized.

You may use these replacement fields in the filter:

- {username_attribute}: the value of *ldap_username_attribute*
- {username_full_name_attribute}: the value of *ldap_full_name_attribute*

Use {username} in the search filter as the placeholder for the username supplied by the user from the login form.

ldap_timeout: int

Number of seconds to wait for an LDAP connection to be established

ldap_min_pool_size: int

Min number of LDAP connections to keep in the pool

ldap_max_pool_size: int

Max number of LDAP connections to keep in the pool

ldap_pool_connection_lifetime_seconds: int

Recycle LDAP connections after this many seconds

sentry_url: str | None

The sentry DSN to use for error reporting. If this is None, no error reporting will be done.

model_config: ClassVar[SettingsConfigDict] = {'arbitrary_types_allowed': True, 'case_sensitive': False, 'env_file': None, 'env_file_encoding': None, 'env_nested_delimiter': None, 'env_prefix': '', 'extra': 'forbid', 'protected_namespaces': ('model_', 'settings_'), 'secrets_dir': None, 'validate_default': True}

redis_url_required_if_session_type_is_redis()

If we've configured the session backend to be redis, *redis_url* is required.

Raises

ValidationError – redis_url is required if session_backend is redis

classmethod construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

copy(***, *include: AbstractSetIntStr | MappingIntStrAny | None = None*, *exclude: AbstractSetIntStr | MappingIntStrAny | None = None*, *update: Dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

This method is now deprecated; use *model_copy* instead. If you need *include* or *exclude*, use:

```
`py data = self.model_dump(include=include, exclude=exclude, round_trip=True)
data = {**data, **(update or {})} copied = self.model_validate(data) `
```

Parameters

- **include** – Optional set or mapping specifying which fields to include in the copied model.

- **exclude** – Optional set or mapping specifying which fields to exclude in the copied model.
- **update** – Optional dictionary of field-value pairs to override field values in the copied model.
- **deep** – If True, the values of fields that are Pydantic models will be deep copied.

Returns
A copy of the model with included, excluded and updated fields as specified.

dict(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*) → Dict[str, Any]

classmethod from_orm(*obj: Any*) → Model

json(*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *encoder: Callable[[Any], Any] | None = PydanticUndefined*, *models_as_dict: bool = PydanticUndefined*, ***dumps_kwargs: Any*) → str

property model_computed_fields: dict[str, ComputedFieldInfo]

Get the computed fields of this model instance.

Returns
A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

classmethod model_construct(*_fields_set: set[str] | None = None*, ***values: Any*) → Model

Creates a new instance of the *Model* class with validated data.

Creates a new model setting *__dict__* and *__pydantic_fields_set__* from trusted or pre-validated data. Default values are respected, but no other validation is performed. Behaves as if *Config.extra = 'allow'* was set since it adds all passed values

Parameters

- **_fields_set** – The set of field names accepted for the Model instance.
- **values** – Trusted or pre-validated data dictionary.

Returns
A new instance of the *Model* class with validated data.

model_copy(*, *update: dict[str, Any] | None = None*, *deep: bool = False*) → Model

Returns a copy of the model.

Parameters

- **update** – Values to change/add in the new model. Note: the data is not validated before creating the new model. You should trust this data.
- **deep** – Set to *True* to make a deep copy of the model.

Returns
New model instance.

model_dump(*, *mode: Literal['json', 'python'] | str = 'python'*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → dict[str, Any]

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump

Generate a dictionary representation of the model, optionally specifying which fields to include or exclude.

Parameters

- **mode** – The mode in which *to_python* should run. If mode is 'json', the dictionary will only contain JSON serializable types. If mode is 'python', the dictionary may contain any Python objects.
- **include** – A list of fields to include in the output.
- **exclude** – A list of fields to exclude from the output.
- **by_alias** – Whether to use the field's alias in the dictionary key if defined.
- **exclude_unset** – Whether to exclude fields that are unset or None from the output.
- **exclude_defaults** – Whether to exclude fields that are set to their default value from the output.
- **exclude_none** – Whether to exclude fields that have a value of *None* from the output.
- **round_trip** – Whether to enable serialization and deserialization round-trip support.
- **warnings** – Whether to log warnings when invalid fields are encountered.

Returns
A dictionary representation of the model.

model_dump_json(*, *indent: int | None = None*, *include: IncEx = None*, *exclude: IncEx = None*, *by_alias: bool = False*, *exclude_unset: bool = False*, *exclude_defaults: bool = False*, *exclude_none: bool = False*, *round_trip: bool = False*, *warnings: bool = True*) → str

Usage docs: https://docs.pydantic.dev/dev-v2/usage/serialization/#modelmodel_dump_json

Generates a JSON representation of the model using Pydantic's *to_json* method.

Parameters

- **indent** – Indentation to use in the JSON output. If None is passed, the output will be compact.
- **include** – Field(s) to include in the JSON output. Can take either a string or set of strings.
- **exclude** – Field(s) to exclude from the JSON output. Can take either a string or set of strings.
- **by_alias** – Whether to serialize using field aliases.
- **exclude_unset** – Whether to exclude fields that have not been explicitly set.
- **exclude_defaults** – Whether to exclude fields that have the default value.
- **exclude_none** – Whether to exclude fields that have a value of *None*.
- **round_trip** – Whether to use serialization/deserialization between JSON and class instance.
- **warnings** – Whether to show any warnings that occurred during serialization.

Returns
A JSON string representation of the model.

property model_extra: dict[str, Any] | None

Get extra fields set during validation.

Returns
A dictionary of extra fields, or *None* if *config.extra* is not set to *"allow"*.

```
model_fields: ClassVar[dict[str, FieldInfo]] = {'auth_realm':
FieldInfo(annotation=str, required=False, default='Restricted'), 'cookie_domain':
FieldInfo(annotation=Union[str, NoneType], required=False), 'cookie_name':
FieldInfo(annotation=str, required=False, default='nginxauth'), 'debug':
FieldInfo(annotation=bool, required=False, default=False),
'ldap_authorization_filter': FieldInfo(annotation=Union[str, NoneType],
required=False), 'ldap_basedn': FieldInfo(annotation=str, required=True),
'ldap_binddn': FieldInfo(annotation=str, required=True), 'ldap_disable_referrals':
FieldInfo(annotation=bool, required=False, default=False),
'ldap_full_name_attribute': FieldInfo(annotation=str, required=False,
default='cn'), 'ldap_get_user_filter': FieldInfo(annotation=str, required=False,
default='{username_attribute}={username}'), 'ldap_max_pool_size':
FieldInfo(annotation=int, required=False, default=30), 'ldap_min_pool_size':
FieldInfo(annotation=int, required=False, default=1), 'ldap_password':
FieldInfo(annotation=str, required=True), 'ldap_pool_connection_lifetime_seconds':
FieldInfo(annotation=int, required=False, default=20), 'ldap_starttls':
FieldInfo(annotation=bool, required=False, default=True), 'ldap_timeout':
FieldInfo(annotation=int, required=False, default=15), 'ldap_uri':
FieldInfo(annotation=str, required=True), 'ldap_username_attribute':
FieldInfo(annotation=str, required=False, default='uid'), 'log_type':
FieldInfo(annotation=Literal['json', 'text'], required=False, default='text'),
'loglevel': FieldInfo(annotation=Literal['NOTSET', 'DEBUG', 'INFO', 'WARN',
'ERROR', 'CRITICAL'], required=False, default='INFO'), 'redis_prefix':
FieldInfo(annotation=str, required=False, default='nginx_ldap_auth.'), 'redis_url':
FieldInfo(annotation=Union[Annotated[pydantic_core._pydantic_core.Url,
UrlConstraints(max_length=None, allowed_schemes=['redis', 'rediss'],
host_required=None, default_host='localhost', default_port=6379,
default_path='/0')], NoneType], required=False), 'secret_key':
FieldInfo(annotation=str, required=True), 'sentry_url':
FieldInfo(annotation=Union[str, NoneType], required=False), 'session_backend':
FieldInfo(annotation=Literal['redis', 'memory'], required=False, default='memory'),
'session_max_age': FieldInfo(annotation=int, required=False, default=0),
'use_rolling_session': FieldInfo(annotation=bool, required=False, default=False)}
```

property model_fields_set: set[str]

Returns the set of fields that have been set on this model instance.

Returns

A set of strings representing the fields that have been set,
i.e. that were not filled from defaults.

classmethod model_json_schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, *schema_generator: type[GenerateJsonSchema] = <class 'pydantic.json_schema.GenerateJsonSchema'>*, *mode: JsonSchemaMode = 'validation'*) → dict[str, Any]

Generates a JSON schema for a model class.

To override the logic used to generate the JSON schema, you can create a subclass of *GenerateJsonSchema* with your desired modifications, then override this method on a custom base class and set the default value of *schema_generator* to be your subclass.

Parameters

- **by_alias** – Whether to use attribute aliases or not.
- **ref_template** – The reference template.

- **schema_generator** – The JSON schema generator.
- **mode** – The mode in which to generate the schema.

Returns
The JSON schema for the given model class.

classmethod model_parametrized_name(*params: tuple[type[Any], ...]*) → str

Compute the class name for parametrizations of generic classes.

This method can be overridden to achieve a custom naming scheme for generic BaseModels.

Parameters
params – Tuple of types of the class. Given a generic class *Model* with 2 type variables and a concrete model *Model[str, int]*, the value *(str, int)* would be passed to *params*.

Returns
String representing the new class where *params* are passed to *cls* as type variables.

Raises
TypeError – Raised when trying to generate concrete names for non-generic models.

model_post_init(*_BaseModel__context: Any*) → None

Override this method to perform additional initialization after *__init__* and *model_construct*. This is useful if you want to do some validation that requires the entire model to be initialized.

classmethod model_rebuild(**, force: bool = False, raise_errors: bool = True, _parent_namespace_depth: int = 2, _types_namespace: dict[str, Any] | None = None*) → bool | None

Try to rebuild the pydantic-core schema for the model.

This may be necessary when one of the annotations is a ForwardRef which could not be resolved during the initial attempt to build the schema, and automatic rebuilding fails.

Parameters

- **force** – Whether to force the rebuilding of the model schema, defaults to *False*.
- **raise_errors** – Whether to raise errors, defaults to *True*.
- **_parent_namespace_depth** – The depth level of the parent namespace, defaults to 2.
- **_types_namespace** – The types namespace, defaults to *None*.

Returns
Returns *None* if the schema is already "complete" and rebuilding was not required. If rebuilding _was_ required, returns *True* if rebuilding was successful, otherwise *False*.

classmethod model_validate(*obj: Any, *, strict: bool | None = None, from_attributes: bool | None = None, context: dict[str, Any] | None = None*) → Model

Validate a pydantic model instance.

Parameters

- **obj** – The object to validate.
- **strict** – Whether to raise an exception on invalid fields.
- **from_attributes** – Whether to extract data from object attributes.
- **context** – Additional context to pass to the validator.

Raises
ValidationError – If the object could not be validated.

Returns
The validated model instance.

classmethod model_validate_json(*json_data: str | bytes | bytearray*, ***, *strict: bool | None = None*, *context: dict[str, Any] | None = None*) → Model

Validate the given JSON data against the Pydantic model.

Parameters

- **json_data** – The JSON data to validate.
- **strict** – Whether to enforce types strictly.
- **context** – Extra variables to pass to the validator.

Returns
The validated Pydantic model.

Raises
ValueError – If *json_data* is not a JSON string.

classmethod parse_file(*path: str | Path*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → Model

classmethod parse_obj(*obj: Any*) → Model

classmethod parse_raw(*b: str | bytes*, ***, *content_type: str | None = None*, *encoding: str = 'utf8'*, *proto: _deprecated_parse.Protocol | None = None*, *allow_pickle: bool = False*) → Model

classmethod schema(*by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*) → Dict[str, Any]

classmethod schema_json(***, *by_alias: bool = True*, *ref_template: str = '#/$defs/{model}'*, ***dumps_kwargs: Any*) → str

classmethod settings_customise_sources(*settings_cls: type[BaseSettings]*, *init_settings: PydanticBaseSettingsSource*, *env_settings: PydanticBaseSettingsSource*, *dotenv_settings: PydanticBaseSettingsSource*, *file_secret_settings: PydanticBaseSettingsSource*) → tuple[PydanticBaseSettingsSource, ...]

Define the sources and their order for loading the settings values.

Parameters

- **settings_cls** – The Settings class.
- **init_settings** – The *InitSettingsSource* instance.
- **env_settings** – The *EnvSettingsSource* instance.
- **dotenv_settings** – The *DotEnvSettingsSource* instance.
- **file_secret_settings** – The *SecretsSettingsSource* instance.

Returns
A tuple containing the sources and their order for loading the settings values.

classmethod update_forward_refs(***localns: Any*) → None

classmethod validate(*value: Any*) → Model

nginx-ldap-auth-service provides a daemon (nginx-ldap-auth) that communicates with an LDAP or Active Directory server to authenticate users with their username and password, as well as a login form for actually allowing users to authenticate. You can use this in combination with the nginx module ngx_http_auth_request_module to provide authentication for your nginx server.

CHAPTER

ELEVEN

FEATURES

11.1 User authentication

- Built for use with the ngx_http_auth_request_module
- Provides its own login form and authentication backend
- Users login once via the login form, creating a login session that will be used for all subsequent requests to determine that the user is logged in.
- Session data can be either in memory or Redis for high availability and session persistence though server restarts.
- The same `nginx_ldap_auth_service` server can be used by multiple nginx servers. This allows you to use a single login form for multiple sites (single signon like), or you can configure each nginx server to use different session cookies so that login sessions are not shared between sites.

11.2 User authorization

- Users can be authorized to access resources based on an LDAP search filter you supply.

11.3 Other features

- Implemented in FastAPI for speed and connection management.
- Available a Docker image that can be used as a sidecar container with nginx.

PYTHON MODULE INDEX

n

INDEX

N

O

P

R

S

T

U

V

W