

# [How-To] Install NetBox on Ubuntu 24 Server

## Purpose

The purpose of this how to is to show the process for installing and configuring NetBox on ubuntu 24.

## Prerequisites

List of prerequisites:

- Sudo user
- Ubuntu 24 LTS VM

## Instructions

### Step 1: Postgresql Installation

First, update the system:

```
sudo apt update
```

Then, install postgresql server:

```
sudo apt install -y postgresql
```

Once installed, run this to verify its installed and check its version:

```
psql -V
```

At a minimum, we need to create a database for NetBox and assign it a username and password for authentication. Start by invoking the PostgreSQL shell as the system Postgres user.

```
sudo -u postgres psql
```

Within the shell, enter the following commands to create the database and user (role), substituting your own value for the password:

```
CREATE DATABASE netbox;  
CREATE USER netbox WITH PASSWORD 'J5brHrAXFLQSif0K';  
ALTER DATABASE netbox OWNER TO netbox;  
-- the next two commands are needed on PostgreSQL 15 and later  
\connect netbox;  
GRANT CREATE ON SCHEMA public TO netbox;
```

Use a strong password

**Do not use the password from the example.** Choose a strong, random password to ensure secure database authentication for your NetBox installation.

Once complete, enter `\q` to exit the PostgreSQL shell.

You can verify that authentication works by executing the `psql` command and passing the configured username and password. (Replace `localhost` with your database server if using a remote database.)

```
$ psql --username netbox --password --host localhost netbox  
Password for user netbox:  
psql (12.5 (Ubuntu 12.5-0ubuntu0.20.04.1))  
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)  
Type "help" for help.  
  
netbox=> \conninfo  
You are connected to database "netbox" as user "netbox" on host "localhost" (address "127.0.0.1") at port  
"5432".  
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits: 256, compression: off)  
netbox=> \q
```

If successful, you will enter a `netbox` prompt. Type `\conninfo` to confirm your connection, or type `\q` to exit.

## Step 2: Redis Configuration

```
sudo apt install -y redis-server
```

Before continuing, verify that your installed version of Redis is at least v4.0:

```
redis-server -v
```

You may wish to modify the Redis configuration at `/etc/redis.conf` or `/etc/redis/redis.conf`, however in most cases the default configuration is sufficient.

Use the `redis-cli` utility to ensure the Redis service is functional:

```
redis-cli ping
```

If successful, you should receive a `PONG` response from the server.

## Step 3: NetBox Installation

This section of the documentation discusses installing and configuring the NetBox application itself.

### Install System Packages

Begin by installing all system packages required by NetBox and its dependencies.

Python 3.10 or later required

NetBox supports Python 3.10, 3.11, and 3.12.

```
sudo apt install -y python3 python3-pip python3-venv python3-dev build-essential libxml2-dev libxslt1-dev libffi-dev libpq-dev libssl-dev zlib1g-dev
```

Before continuing, check that your installed Python version is at least 3.10:

```
python3 -V
```

## Download NetBox

This documentation provides two options for installing NetBox: from a downloadable archive, or from the git repository. Installing from a package (option A below) requires manually fetching and extracting the archive for every future update, whereas installation via git (option B) allows for seamless upgrades by re-pulling the `master` branch.

### Option A: Download a Release Archive

Download the [latest stable release](#) from GitHub as a tarball or ZIP archive and extract it to your desired path. In this example, we'll use `/opt/netbox` as the NetBox root.

```
sudo wget https://github.com/netbox-community/netbox/archive/refs/tags/vX.Y.Z.tar.gz
sudo tar -xzf vX.Y.Z.tar.gz -C /opt
```

```
sudo ln -s /opt/netbox-X.Y.Z/ /opt/netbox
```

It is recommended to install NetBox in a directory named for its version number. For example, NetBox v3.0.0 would be installed into `/opt/netbox-3.0.0`, and a symlink from `/opt/netbox/` would point to this location. (You can verify this configuration with the command `ls -l /opt | grep netbox`.) This allows for future releases to be installed in parallel without interrupting the current installation. When changing to the new release, only the symlink needs to be updated.

## Option B: Clone the Git Repository

Create the base directory for the NetBox installation. For this guide, we'll use `/opt/netbox`.

```
sudo mkdir -p /opt/netbox/  
cd /opt/netbox/
```

If `git` is not already installed, install it:

```
sudo apt install -y git
```

Next, clone the **master** branch of the NetBox GitHub repository into the current directory. (This branch always holds the current stable release.)

```
sudo git clone -b master --depth 1 https://github.com/netbox-community/netbox.git .
```

The `git clone` command above utilizes a "shallow clone" to retrieve only the most recent commit. If you need to download the entire history, omit the `--depth 1` argument.

The `git clone` command should generate output similar to the following:

```
Cloning into '.'...  
remote: Enumerating objects: 996, done.  
remote: Counting objects: 100% (996/996), done.  
remote: Compressing objects: 100% (935/935), done.  
remote: Total 996 (delta 148), reused 386 (delta 34), pack-reused 0  
Receiving objects: 100% (996/996), 4.26 MiB | 9.81 MiB/s, done.  
Resolving deltas: 100% (148/148), done.
```

Installation via git also allows you to easily try out different versions of NetBox. To check out a [specific NetBox release](#), use the `git checkout` command with the desired release tag. For

example, `git checkout v3.0.8`.

## Create the NetBox System User

Create a system user account named `netbox`. We'll configure the WSGI and HTTP services to run under this account. We'll also assign this user ownership of the media directory. This ensures that NetBox will be able to save uploaded files.

```
sudo adduser --system --group netbox
sudo chown --recursive netbox /opt/netbox/netbox/media/
sudo chown --recursive netbox /opt/netbox/netbox/reports/
sudo chown --recursive netbox /opt/netbox/netbox/scripts/
```

## Configuration

Move into the NetBox configuration directory and make a copy of `configuration_example.py` named `configuration.py`. This file will hold all of your local configuration parameters.

```
cd /opt/netbox/netbox/netbox/
sudo cp configuration_example.py configuration.py
```

Open `configuration.py` with your preferred editor to begin configuring NetBox. NetBox offers [many configuration parameters](#), but only the following four are required for new installations:

- `ALLOWED_HOSTS`
- `DATABASE`
- `REDIS`
- `SECRET_KEY`

### ALLOWED\_HOSTS

This is a list of the valid hostnames and IP addresses by which this server can be reached. You must specify at least one name or IP address. (Note that this does not restrict the locations from which NetBox may be accessed: It is merely for [HTTP host header validation](#).)

```
ALLOWED_HOSTS = ['netbox.example.com', '192.0.2.123']
```

If you are not yet sure what the domain name and/or IP address of the NetBox installation will be, you can set this to a wildcard (asterisk) to allow all host values:

```
ALLOWED_HOSTS = ['*']
```

### DATABASE

This parameter holds the database configuration details. You must define the username and password used when you configured PostgreSQL. If the service is running on a remote host, update the `HOST` and `PORT` parameters accordingly. See the [configuration documentation](#) for more detail on individual parameters.

```
DATABASE = {
    'NAME': 'netbox',          # Database name
    'USER': 'netbox',          # PostgreSQL username
    'PASSWORD': 'J5brHrAXFLQSif0K', # PostgreSQL password
    'HOST': 'localhost',       # Database server
    'PORT': '',                # Database port (leave blank for default)
    'CONN_MAX_AGE': 300,       # Max database connection age (seconds)
}
```

## REDIS

Redis is a in-memory key-value store used by NetBox for caching and background task queuing. Redis typically requires minimal configuration; the values below should suffice for most installations. See the [configuration documentation](#) for more detail on individual parameters.

Note that NetBox requires the specification of two separate Redis databases: `tasks` and `caching`. These may both be provided by the same Redis service, however each should have a unique numeric database ID.

```
REDIS = {
    'tasks': {
        'HOST': 'localhost',    # Redis server
        'PORT': 6379,           # Redis port
        'PASSWORD': '',         # Redis password (optional)
        'DATABASE': 0,          # Database ID
        'SSL': False,           # Use SSL (optional)
    },
    'caching': {
        'HOST': 'localhost',
        'PORT': 6379,
        'PASSWORD': '',
        'DATABASE': 1,          # Unique ID for second database
        'SSL': False,
    }
}
```

## SECRET\_KEY

This parameter must be assigned a randomly-generated key employed as a salt for hashing and related cryptographic functions. (Note, however, that it is *never* directly used in the encryption of secret data.) This key must be unique to this installation and is recommended to be at least 50 characters long. It should not be shared outside the local system.

A simple Python script named `generate_secret_key.py` is provided in the parent directory to assist in generating a suitable key:

```
python3 ../generate_secret_key.py
```

SECRET\_KEY values must match

In the case of a highly available installation with multiple web servers, `SECRET_KEY` must be identical among all servers in order to maintain a persistent user session state.

When you have finished modifying the configuration, remember to save the file.

## Optional Requirements

All Python packages required by NetBox are listed in `requirements.txt` and will be installed automatically. NetBox also supports some optional packages. If desired, these packages must be listed in `local_requirements.txt` within the NetBox root directory.

## Remote File Storage

By default, NetBox will use the local filesystem to store uploaded files. To use a remote filesystem, install the `django-storages` library and configure your [desired storage backend](#) in `configuration.py`.

```
sudo sh -c "echo 'django-storages' >> /opt/netbox/local_requirements.txt"
```

## Remote Data Sources

NetBox supports integration with several remote data sources via configurable backends. Each of these requires the installation of one or more additional libraries.

- Amazon S3: `boto3`
- Git: `dulwich`

For example, to enable the Amazon S3 backend, add `boto3` to your local requirements file:

```
sudo sh -c "echo 'boto3' >> /opt/netbox/local_requirements.txt"
```

These packages were previously required in NetBox v3.5 but now are optional.

## Sentry Integration

NetBox may be configured to send error reports to [Sentry](#) for analysis. This integration requires installation of the `sentry-sdk` Python library.

```
sudo sh -c "echo 'sentry-sdk' >> /opt/netbox/local_requirements.txt"
```

Sentry integration was previously included by default in NetBox v3.6 but is now optional.

## Run the Upgrade Script

Once NetBox has been configured, we're ready to proceed with the actual installation. We'll run the packaged upgrade script (`upgrade.sh`) to perform the following actions:

- Create a Python virtual environment
- Installs all required Python packages
- Run database schema migrations
- Builds the documentation locally (for offline use)
- Aggregate static resource files on disk

If you still have a Python virtual environment active from a previous installation step, disable it now by running the `deactivate` command. This will avoid errors on systems where `sudo` has been configured to preserve the user's current environment.

```
sudo /opt/netbox/upgrade.sh
```

Note that **Python 3.10 or later is required** for NetBox v4.0 and later releases. If the default Python installation on your server is set to a lesser version, pass the path to the supported installation as an environment variable named `PYTHON`. (Note that the environment variable must be passed *after* the `sudo` command.)

```
sudo PYTHON=/usr/bin/python3.10 /opt/netbox/upgrade.sh
```

Upon completion, the upgrade script may warn that no existing virtual environment was detected. As this is a new installation, this warning can be safely ignored.

## Create a Super User

NetBox does not come with any predefined user accounts. You'll need to create a super user (administrative account) to be able to log into NetBox. First, enter the Python virtual environment



created by the upgrade script:

```
source /opt/netbox/venv/bin/activate
```

Once the virtual environment has been activated, you should notice the string `(venv)` prepended to your console prompt.

Next, we'll create a superuser account using the `createsuperuser` Django management command (via `manage.py`). Specifying an email address for the user is not required, but be sure to use a very strong password.

```
cd /opt/netbox/netbox
python3 manage.py createsuperuser
```

## Schedule the Housekeeping Task

NetBox includes a `housekeeping` management command that handles some recurring cleanup tasks, such as clearing out old sessions and expired change records. Although this command may be run manually, it is recommended to configure a scheduled job using the system's `cron` daemon or a similar utility.

A shell script which invokes this command is included at `contrib/netbox-housekeeping.sh`. It can be copied to or linked from your system's daily cron task directory, or included within the crontab directly. (If installing NetBox into a nonstandard path, be sure to update the system paths within this script first.)

```
sudo ln -s /opt/netbox/contrib/netbox-housekeeping.sh /etc/cron.daily/netbox-housekeeping
```

## Test the Application

At this point, we should be able to run NetBox's development server for testing. We can check by starting a development instance locally.

Check that the Python virtual environment is still active before attempting to run the server.

```
python3 manage.py runserver 0.0.0.0:8000 --insecure
```

If successful, you should see output similar to the following:

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
August 30, 2021 - 18:02:23
```

```
Django version 3.2.6, using settings 'netbox.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Next, connect to the name or IP of the server (as defined in `ALLOWED_HOSTS`) on port 8000; for example, <http://127.0.0.1:8000/>. You should be greeted with the NetBox home page. Try logging in using the username and password specified when creating a superuser.

By default RHEL based distros will likely block your testing attempts with `firewalld`. The development server port can be opened with `firewall-cmd` (add `--permanent` if you want the rule to survive server restarts):

```
firewall-cmd --zone=public --add-port=8000/tcp
```

The development server is for development and testing purposes only. It is neither performant nor secure enough for production use. **Do not use it in production.**

If the test service does not run, or you cannot reach the NetBox home page, something has gone wrong. Do not proceed with the rest of this guide until the installation has been corrected.

Type `Ctrl+c` to stop the development server.

## Step 4: Gunicorn Configuration

This page provides instructions for setting up the [gunicorn](#) WSGI server. If you plan to use [uWSGI](#) instead, go [here](#).

NetBox runs as a [WSGI application](#) behind an HTTP server. This documentation shows how to install and configure [gunicorn](#) (which is automatically installed with NetBox) for this role, however other WSGI servers are available and should work similarly well.

## Configuration

NetBox ships with a default configuration file for gunicorn. To use it, copy `/opt/netbox/contrib/gunicorn.py` to `/opt/netbox/gunicorn.py`. (We make a copy of this file rather than pointing to it directly to ensure that any local changes to it do not get overwritten during a future NetBox upgrade.)

```
sudo cp /opt/netbox/contrib/gunicorn.py /opt/netbox/gunicorn.py
```

While the provided configuration should suffice for most initial installations, you may wish to edit this file to change the bound IP address and/or port number, or to make performance-related adjustments. See [the Gunicorn documentation](#) for the available configuration parameters.

## systemd Setup

We'll use systemd to control both gunicorn and NetBox's background worker process. First, copy `contrib/netbox.service` and `contrib/netbox-rq.service` to the `/etc/systemd/system/` directory and reload the systemd daemon.

## Check user & group assignment

The stock service configuration files packaged with NetBox assume that the service will run with the `netbox` user and group names. If these differ on your installation, be sure to update the service files accordingly.

```
sudo cp -v /opt/netbox/contrib/*.service /etc/systemd/system/  
sudo systemctl daemon-reload
```

Then, start the `netbox` and `netbox-rq` services and enable them to initiate at boot time:

```
sudo systemctl enable --now netbox netbox-rq
```

You can use the command `systemctl status netbox` to verify that the WSGI service is running:

```
systemctl status netbox.service
```

You should see output similar to the following:

```
● netbox.service - NetBox WSGI Service  
   Loaded: loaded (/etc/systemd/system/netbox.service; enabled; vendor preset: enabled)  
   Active: active (running) since Mon 2021-08-30 04:02:36 UTC; 14h ago  
     Docs: https://docs.netbox.dev/  
  Main PID: 1140492 (gunicorn)  
    Tasks: 19 (limit: 4683)  
   Memory: 666.2M  
    CGroup: /system.slice/netbox.service  
           └─1140492 /opt/netbox/venv/bin/python3 /opt/netbox/venv/bin/gunicorn --pid /va>  
           └─1140513 /opt/netbox/venv/bin/python3 /opt/netbox/venv/bin/gunicorn --pid /va>  
           └─1140514 /opt/netbox/venv/bin/python3 /opt/netbox/venv/bin/gunicorn --pid /va>  
...
```

If the NetBox service fails to start, issue the command `journalctl -eu netbox` to check for log messages that may indicate the problem.

Once you've verified that the WSGI workers are up and running, move on to HTTP server setup.

There is a bug in the current stable release of gunicorn (v21.2.0) where automatic restarts of the worker processes can result in 502 errors under heavy load. (See [gunicorn bug #3038](#) for more detail.) Users who encounter this issue may opt to downgrade to an earlier, unaffected release of gunicorn ( `pip install gunicorn==20.1.0` ). Note, however, that this earlier release does not officially support Python 3.11.

## Step 5: Web Server Configuration

This documentation provides example configurations for both [nginx](#) and [Apache](#), though any HTTP server which supports WSGI should be compatible.

For the sake of brevity, only Ubuntu 20.04 instructions are provided here. These tasks are not unique to NetBox and should carry over to other distributions with minimal changes. Please consult your distribution's documentation for assistance if needed.

### Obtain an SSL Certificate

To enable HTTPS access to NetBox, you'll need a valid SSL certificate. You can purchase one from a trusted commercial provider, obtain one for free from [Let's Encrypt](#), or generate your own (although self-signed certificates are generally untrusted). Both the public certificate and private key files need to be installed on your NetBox server in a location that is readable by the `netbox` user.

The command below can be used to generate a self-signed certificate for testing purposes, however it is strongly recommended to use a certificate from a trusted authority in production. Two files will be created: the public certificate (`netbox.crt`) and the private key (`netbox.key`). The certificate is published to the world, whereas the private key must be kept secret at all times.

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \  
-keyout /etc/ssl/private/netbox.key \  
-out /etc/ssl/certs/netbox.crt
```

The above command will prompt you for additional details of the certificate; all of these are optional.

### HTTP Server Installation

## Option A: nginx

Begin by installing nginx:

```
sudo apt install -y nginx
```

Once nginx is installed, copy the nginx configuration file provided by NetBox to `/etc/nginx/sites-available/netbox`. Be sure to replace `netbox.example.com` with the domain name or IP address of your installation. (This should match the value configured for `ALLOWED_HOSTS` in `configuration.py`.)

```
sudo cp /opt/netbox/contrib/nginx.conf /etc/nginx/sites-available/netbox
```

### gunicorn vs. uWSGI

The reference nginx configuration file assumes that gunicorn is in use. If using uWSGI instead, you'll need to remove the gunicorn-specific configuration (lines beginning with `proxy_pass` and `proxy_set_header`) and uncomment the uWSGI section below them before proceeding.

Then, delete `/etc/nginx/sites-enabled/default` and create a symlink in the `sites-enabled` directory to the configuration file you just created.

```
sudo rm /etc/nginx/sites-enabled/default
sudo ln -s /etc/nginx/sites-available/netbox /etc/nginx/sites-enabled/netbox
```

Finally, restart the `nginx` service to use the new configuration.

```
sudo systemctl restart nginx
```

## Option B: Apache

Begin by installing Apache:

```
sudo apt install -y apache2
```

Next, copy the default configuration file to `/etc/apache2/sites-available/`. Be sure to modify the `ServerName` parameter appropriately.

```
sudo cp /opt/netbox/contrib/apache.conf /etc/apache2/sites-available/netbox.conf
```

Finally, ensure that the required Apache modules are enabled, enable the `netbox` site, and reload Apache:

```
sudo a2enmod ssl proxy proxy_http headers rewrite
sudo a2ensite netbox
```

```
sudo systemctl restart apache2
```

## Confirm Connectivity

At this point, you should be able to connect to the HTTPS service at the server name or IP address you provided.

Please keep in mind that the configurations provided here are bare minimums required to get NetBox up and running. You may want to make adjustments to better suit your production environment.

Certain components of NetBox (such as the display of rack elevation diagrams) rely on the use of embedded objects. Ensure that your HTTP server configuration does not override the `X-Frame-Options` response header set by NetBox.

## Troubleshooting

If you are unable to connect to the HTTP server, check that:

- Nginx/Apache is running and configured to listen on the correct port.
- Access is not being blocked by a firewall somewhere along the path. (Try connecting locally from the server itself.)

If you are able to connect but receive a 502 (bad gateway) error, check the following:

- The WSGI worker processes (gunicorn) are running (`systemctl status netbox` should show a status of "active (running)")
- Nginx/Apache is configured to connect to the port on which gunicorn is listening (default is 8001).
- SELinux is not preventing the reverse proxy connection. You may need to allow HTTP network connections with the command `setsebool -P httpd_can_network_connect 1`

## Step 6: LDAP Configuration

This guide explains how to implement LDAP authentication using an external server. User authentication will fall back to built-in Django users in the event of a failure.

### Install System Requirements

```
sudo apt install -y libldap2-dev libsasl2-dev libssl-dev
```

### Install django-auth-ldap

Activate the Python virtual environment and install the `django-auth-ldap` package using pip:

```
source /opt/netbox/venv/bin/activate
pip3 install django-auth-ldap
```

Once installed, add the package to `local_requirements.txt` to ensure it is re-installed during future rebuilds of the virtual environment:

```
sudo sh -c "echo 'django-auth-ldap' >> /opt/netbox/local_requirements.txt"
```

## Configuration

First, enable the LDAP authentication backend in `configuration.py`. (Be sure to overwrite this definition if it is already set to `RemoteUserBackend`.)

```
REMOTE_AUTH_BACKEND = 'netbox.authentication.LDAPBackend'
```

Next, create a file in the same directory as `configuration.py` (typically `/opt/netbox/netbox/netbox/`) named `ldap_config.py`. Define all of the parameters required below in `ldap_config.py`. Complete documentation of all `django-auth-ldap` configuration options is included in the project's [official documentation](#).

## General Server Configuration

When using Active Directory you may need to specify a port on `AUTH_LDAP_SERVER_URI` to authenticate users from all domains in the forest. Use `3269` for secure, or `3268` for non-secure access to the GC (Global Catalog).

```
import ldap

# Server URI
AUTH_LDAP_SERVER_URI = "ldaps://ad.example.com"

# The following may be needed if you are binding to Active Directory.
AUTH_LDAP_CONNECTION_OPTIONS = {
    ldap.OPT_REFERRALS: 0
}

# Set the DN and password for the NetBox service account.
AUTH_LDAP_BIND_DN = "CN=NETBOXSA, OU=Service Accounts, DC=example, DC=com"
AUTH_LDAP_BIND_PASSWORD = "demo"

# Include this setting if you want to ignore certificate errors. This might be needed to accept a self-signed cert.
```

```
# Note that this is a NetBox-specific setting which sets:
#   ldap.set_option(ldap.OPT_X_TLS_REQUIRE_CERT, ldap.OPT_X_TLS_NEVER)
LDAP_IGNORE_CERT_ERRORS = True

# Include this setting if you want to validate the LDAP server certificates against a CA certificate directory on
your server
# Note that this is a NetBox-specific setting which sets:
#   ldap.set_option(ldap.OPT_X_TLS_CACERTDIR, LDAP_CA_CERT_DIR)
LDAP_CA_CERT_DIR = '/etc/ssl/certs'

# Include this setting if you want to validate the LDAP server certificates against your own CA.
# Note that this is a NetBox-specific setting which sets:
#   ldap.set_option(ldap.OPT_X_TLS_CACERTFILE, LDAP_CA_CERT_FILE)
LDAP_CA_CERT_FILE = '/path/to/example-CA.crt'
```

STARTTLS can be configured by setting `AUTH_LDAP_START_TLS = True` and using the `ldap://` URI scheme.

## User Authentication

When using Windows Server 2012+, `AUTH_LDAP_USER_DN_TEMPLATE` should be set to None.

```
from django_auth_ldap.config import LDAPSearch

# This search matches users with the sAMAccountName equal to the provided username. This is required if the
user's
# username is not in their DN (Active Directory).
AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=Users,dc=example,dc=com",
                                   ldap.SCOPE_SUBTREE,
                                   "(sAMAccountName=%(user)s)")

# If a user's DN is producible from their username, we don't need to search.
AUTH_LDAP_USER_DN_TEMPLATE = "uid=%(user)s,ou=users,dc=example,dc=com"

# You can map user attributes to Django attributes as so.
AUTH_LDAP_USER_ATTR_MAP = {
    "first_name": "givenName",
    "last_name": "sn",
    "email": "mail"
}
```



# User Groups for Permissions

When using Microsoft Active Directory, support for nested groups can be activated by using `NestedGroupOfNamesType()` instead of `GroupOfNamesType()` for `AUTH_LDAP_GROUP_TYPE`. You will also need to modify the import line to use `NestedGroupOfNamesType` instead of `GroupOfNamesType`.

```
from django_auth_ldap.config import LDAPSearch, GroupOfNamesType

# This search ought to return all groups to which the user belongs. django_auth_ldap uses this to determine
# group
# hierarchy.
AUTH_LDAP_GROUP_SEARCH = LDAPSearch("dc=example,dc=com", ldap.SCOPE_SUBTREE,
                                     "(objectClass=group)")
AUTH_LDAP_GROUP_TYPE = GroupOfNamesType()

# Define a group required to login.
AUTH_LDAP_REQUIRE_GROUP = "CN=NETBOX_USERS,DC=example,DC=com"

# Mirror LDAP group assignments.
AUTH_LDAP_MIRROR_GROUPS = True

# Define special user types using groups. Exercise great caution when assigning superuser status.
AUTH_LDAP_USER_FLAGS_BY_GROUP = {
    "is_active": "cn=active,ou=groups,dc=example,dc=com",
    "is_staff": "cn=staff,ou=groups,dc=example,dc=com",
    "is_superuser": "cn=superuser,ou=groups,dc=example,dc=com"
}

# For more granular permissions, we can map LDAP groups to Django groups.
AUTH_LDAP_FIND_GROUP_PERMS = True

# Cache groups for one hour to reduce LDAP traffic
AUTH_LDAP_CACHE_TIMEOUT = 3600
```

- `is_active` - All users must be mapped to at least this group to enable authentication. Without this, users cannot log in.
- `is_staff` - Users mapped to this group are enabled for access to the administration tools; this is the equivalent of checking the "staff status" box on a manually created user. This doesn't grant any specific permissions.

- `is_superuser` - Users mapped to this group will be granted superuser status. Superusers are implicitly granted all permissions.

Authentication will fail if the groups (the distinguished names) do not exist in the LDAP directory.

## Authenticating with Active Directory

Integrating Active Directory for authentication can be a bit challenging as it may require handling different login formats. This solution will allow users to log in either using their full User Principal Name (UPN) or their username alone, by filtering the DN according to either the `sAMAccountName` or the `userPrincipalName`. The following configuration options will allow your users to enter their usernames in the format `username` or `username@domain.tld`.

Just as before, the configuration options are defined in the file `ldap_config.py`. First, modify the `AUTH_LDAP_USER_SEARCH` option to match the following:

```
AUTH_LDAP_USER_SEARCH = LDAPSearch(  
    "ou=Users,dc=example,dc=com",  
    ldap.SCOPE_SUBTREE,  
    "(|(userPrincipalName=%(user)s)(sAMAccountName=%(user)s))"  
)
```

In addition, `AUTH_LDAP_USER_DN_TEMPLATE` should be set to `None` as described in the previous sections. Next, modify `AUTH_LDAP_USER_ATTR_MAP` to match the following:

```
AUTH_LDAP_USER_ATTR_MAP = {  
    "username": "sAMAccountName",  
    "email": "mail",  
    "first_name": "givenName",  
    "last_name": "sn",  
}
```

Finally, we need to add one more configuration option, `AUTH_LDAP_USER_QUERY_FIELD`. The following should be added to your LDAP configuration file:

```
AUTH_LDAP_USER_QUERY_FIELD = "username"
```

With these configuration options, your users will be able to log in either with or without the UPN suffix.

## Example Configuration

This configuration is intended to serve as a template, but may need to be modified in accordance with your environment.

```
import ldap
from django_auth_ldap.config import LDAPSearch, NestedGroupOfNamesType

# Server URI
AUTH_LDAP_SERVER_URI = "ldaps://ad.example.com:3269"

# The following may be needed if you are binding to Active Directory.
AUTH_LDAP_CONNECTION_OPTIONS = {
    ldap.OPT_REFERRALS: 0
}

# Set the DN and password for the NetBox service account.
AUTH_LDAP_BIND_DN = "CN=NETBOXSA,OU=Service Accounts,DC=example,DC=com"
AUTH_LDAP_BIND_PASSWORD = "demo"

# Include this setting if you want to ignore certificate errors. This might be needed to accept a self-signed cert.
# Note that this is a NetBox-specific setting which sets:
#     ldap.set_option(ldap.OPT_X_TLS_REQUIRE_CERT, ldap.OPT_X_TLS_NEVER)
LDAP_IGNORE_CERT_ERRORS = False

# Include this setting if you want to validate the LDAP server certificates against a CA certificate directory on
# your server
# Note that this is a NetBox-specific setting which sets:
#     ldap.set_option(ldap.OPT_X_TLS_CACERTDIR, LDAP_CA_CERT_DIR)
LDAP_CA_CERT_DIR = '/etc/ssl/certs'

# Include this setting if you want to validate the LDAP server certificates against your own CA.
# Note that this is a NetBox-specific setting which sets:
#     ldap.set_option(ldap.OPT_X_TLS_CACERTFILE, LDAP_CA_CERT_FILE)
LDAP_CA_CERT_FILE = '/path/to/example-CA.crt'

# This search matches users with the sAMAccountName equal to the provided username. This is required if the
# user's
# username is not in their DN (Active Directory).
AUTH_LDAP_USER_SEARCH = LDAPSearch(
    "ou=Users,dc=example,dc=com",
    ldap.SCOPE_SUBTREE,
```

```

"(|(userPrincipalName=%(user)s)(sAMAccountName=%(user)s))"
)

# If a user's DN is producible from their username, we don't need to search.
AUTH_LDAP_USER_DN_TEMPLATE = None

# You can map user attributes to Django attributes as so.
AUTH_LDAP_USER_ATTR_MAP = {
    "username": "sAMAccountName",
    "email": "mail",
    "first_name": "givenName",
    "last_name": "sn",
}

AUTH_LDAP_USER_QUERY_FIELD = "username"

# This search ought to return all groups to which the user belongs. django_auth_ldap uses this to determine
group
# hierarchy.
AUTH_LDAP_GROUP_SEARCH = LDAPSearch(
    "dc=example,dc=com",
    ldap.SCOPE_SUBTREE,
    "(objectClass=group)"
)
AUTH_LDAP_GROUP_TYPE = NestedGroupOfNamesType()

# Define a group required to login.
AUTH_LDAP_REQUIRE_GROUP = "CN=NETBOX_USERS,DC=example,DC=com"

# Mirror LDAP group assignments.
AUTH_LDAP_MIRROR_GROUPS = True

# Define special user types using groups. Exercise great caution when assigning superuser status.
AUTH_LDAP_USER_FLAGS_BY_GROUP = {
    "is_active": "cn=active,ou=groups,dc=example,dc=com",
    "is_staff": "cn=staff,ou=groups,dc=example,dc=com",
    "is_superuser": "cn=superuser,ou=groups,dc=example,dc=com"
}

# For more granular permissions, we can map LDAP groups to Django groups.

```

```
AUTH_LDAP_FIND_GROUP_PERMS = True

# Cache groups for one hour to reduce LDAP traffic
AUTH_LDAP_CACHE_TIMEOUT = 3600
AUTH_LDAP_ALWAYS_UPDATE_USER = True
```

If you have issues with django-ldap-auth not being installed, leave the venv and install again with the break param and then re-compile the app.

## Troubleshooting LDAP

`systemctl restart netbox` restarts the NetBox service, and initiates any changes made to `ldap_config.py`. If there are syntax errors present, the NetBox process will not spawn an instance, and errors should be logged to `/var/log/messages`.

For troubleshooting LDAP user/group queries, add or merge the following [logging](#) configuration to `configuration.py`:

```
LOGGING = {
    'version': 1,
    'disable_existing_loggers': False,
    'handlers': {
        'netbox_auth_log': {
            'level': 'DEBUG',
            'class': 'logging.handlers.RotatingFileHandler',
            'filename': '/opt/netbox/local/logs/django-ldap-debug.log',
            'maxBytes': 1024 * 500,
            'backupCount': 5,
        },
    },
    'loggers': {
        'django_auth_ldap': {
            'handlers': ['netbox_auth_log'],
            'level': 'DEBUG',
        },
    },
}
```

Ensure the file and path specified in logfile exist and are writable and executable by the application service account. Restart the netbox service and attempt to log into the site to trigger log entries to this file.

With this, you should have a fully functioning NetBox installation integrated with active directory.

<https://netboxlabs.com/docs/netbox/en/stable/installation/>

---

Revision #3

Created 22 December 2024 19:27:17 by Mike Leffring

Updated 25 December 2024 05:07:11 by Mike Leffring