Nextcloud

Aplicativo de compartilhamento e armazenamento.

- Repositorios de instalações Nextcloud
  - Nextcloud repositório oficial - comunidade
  - Deploy Nextcloud with docker compose - (Medium site)
- Instalação Nextcloud Docker
  - Nextcloud Docker oficial instalação

Repositorios de instalações Nextcloud

Repositorios de instalações Nextcloud

Nextcloud repositório oficial - comunidade

Link: https://github.com/nextcloud/docker git clone https://github.com/nextcloud/docker.git

Versão 29 extraída em Maio/2024.

What is Nextcloud?

GitHub CI build status badge update.sh build status badge amd64 build status badge arm32v5 build status badge arm32v6 build status badge arm32v7 build status badge arm64v8 build status badge i386 build status badge mips64le build status badge ppc64le build status badge s390x build status badge

A safe home for all your data. Access & share your files, calendars, contacts, mail & more from any device, on your terms.

logo

⚠⚠⚠ This image is maintained by community volunteers and designed for expert use. For quick and easy deployment that supports the full set of Nextcloud Hub features, use the Nextcloud All-in-One docker container maintained by Nextcloud GmbH.

How to use this image

This image is designed to be used in a micro-service environment. There are two versions of the image you can choose from.

The `apache` tag contains a full Nextcloud installation including an apache web server. It is designed to be easy to use and gets you running pretty fast. This is also the default for the `latest` tag and version tags that are not further specified.

The second option is a `fpm` container. It is based on the php-fpm image and runs a fastCGI-Process that serves your Nextcloud page. To use this image it must be combined with any webserver that can proxy the http requests to the FastCGI-port of the container.

Try in PWD

Using the apache image

The apache image contains a webserver and exposes port 80. To start the container type:

```
$ docker run -d -p 8080:80 nextcloud
```

Now you can access Nextcloud at http://localhost:8080/ from your host system.

Using the fpm image

To use the fpm image, you need an additional web server, such as nginx, that can proxy http-request to the fpm-port of the container. For fpm connection this container exposes port 9000. In most cases, you might want to use another container or your host as proxy. If you use your host you can address your Nextcloud container directly on port 9000. If you use another container, make sure that you add them to the same docker network (via `docker run --network <NAME> ...` or a `docker-compose` file). In both cases you don't want to map the fpm port to your host.

```
$ docker run -d nextcloud:fpm
```

As the fastCGI-Process is not capable of serving static files (style sheets, images, ...), the webserver needs access to these files. This can be achieved with the `volumes-from` option. You can find more information in the docker-compose section.

Using an external database

By default, this container uses SQLite for data storage but the Nextcloud setup wizard (appears on first run) allows connecting to an existing MySQL/MariaDB or PostgreSQL database. You can also link a database container, e. g. `--link my-mysql:mysql`, and then use `mysql` as the database host on setup. More info is in the docker-compose section.

Persistent data

The Nextcloud installation and all data beyond what lives in the database (file uploads, etc.) are stored in the unnamed docker volume volume `/var/www/html`. The docker daemon will store that data within the docker directory `/var/lib/docker/volumes/...`. That means your data is saved even if the container crashes, is stopped or deleted.

A named Docker volume or a mounted host directory should be used for upgrades and backups. To achieve this, you need one volume for your database container and one for Nextcloud.

Nextcloud:

- `/var/www/html/` folder where all Nextcloud data lives

```
$ docker run -d \
-v nextcloud:/var/www/html \
nextcloud
```

Database:

- `/var/lib/mysql` MySQL / MariaDB Data
- `/var/lib/postgresql/data` PostgreSQL Data

```
$ docker run -d \
-v db:/var/lib/mysql \
mariadb:10.6
```

Additional volumes

If you want to get fine grained access to your individual files, you can mount additional volumes for data, config, your theme and custom apps. The `data`, `config` files are stored in respective subfolders inside `/var/www/html/`. The apps are split into core `apps` (which are shipped with Nextcloud and you don't need to take care of) and a `custom_apps` folder. If you use a custom theme it would go into the `themes` subfolder.

Overview of the folders that can be mounted as volumes:

- `/var/www/html` Main folder, needed for updating
- `/var/www/html/custom_apps` installed / modified apps
- `/var/www/html/config` local configuration
- `/var/www/html/data` the actual data of your Nextcloud
- `/var/www/html/themes/<YOUR_CUSTOM_THEME>` theming/branding

If you want to use named volumes for all of these, it would look like this:

```
$ docker run -d \
-v nextcloud:/var/www/html \
-v apps:/var/www/html/custom_apps \
-v config:/var/www/html/config \
-v data:/var/www/html/data \
-v theme:/var/www/html/themes/<YOUR_CUSTOM_THEME> \
nextcloud
```

Custom volumes

If mounting additional volumes under `/var/www/html`, you should consider:

- Confirming that upgrade.exclude contains the files and folders that should persist during installation and upgrades; or
- Mounting storage volumes to locations outside of `/var/www/html`.

Warning

You should note that data inside the main folder (`/var/www/html`) will be overridden/removed during installation and upgrades, unless listed in upgrade.exclude. The additional volumes officially supported are already in that list, but custom volumes will need to be added by you. We suggest mounting custom storage volumes outside of `/var/www/html` and if possible read-only so that making this adjustment is unnecessary. If you must do so, however, you may build a custom image with a modified `/upgrade.exclude` file that incorporates your custom volume(s).

Using the Nextcloud command-line interface

To use the Nextcloud command-line interface (aka. `occ` command):

```
$ docker exec --user www-data CONTAINER_ID php occ
```

or for docker-compose:

```
$ docker-compose exec --user www-data app php occ
```

Auto configuration via environment variables

The Nextcloud image supports auto configuration via environment variables. You can preconfigure everything that is asked on the install page on first run. To enable auto configuration, set your database connection via the following environment variables. You must specify all of the environment variables for a given database or the database environment variables defaults to SQLITE. ONLY use one database type!

SQLite:

- `SQLITE_DATABASE` Name of the database using sqlite

MYSQL/MariaDB:

- `MYSQL_DATABASE` Name of the database using mysql / mariadb.
- `MYSQL_USER` Username for the database using mysql / mariadb.
- `MYSQL_PASSWORD` Password for the database user using mysql / mariadb.
- `MYSQL_HOST` Hostname of the database server using mysql / mariadb.

PostgreSQL:

- `POSTGRES_DB` Name of the database using postgres.
- `POSTGRES_USER` Username for the database using postgres.
- `POSTGRES_PASSWORD` Password for the database user using postgres.
- `POSTGRES_HOST` Hostname of the database server using postgres.

As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to the previously listed environment variables, causing the initialization script to load the values for those variables from files present in the container. See Docker secrets section below.

If you set any group of values (i.e. all of `MYSQL_DATABASE`, `MYSQL_USER`, `MYSQL_PASSWORD`, `MYSQL_HOST`), they will not be asked in the install page on first run. With a complete configuration by using all variables for your database type, you can additionally configure your Nextcloud instance by setting admin user and password (only works if you set both):

- `NEXTCLOUD_ADMIN_USER` Name of the Nextcloud admin user.
- `NEXTCLOUD_ADMIN_PASSWORD` Password for the Nextcloud admin user.

If you want, you can set the data directory, otherwise default value will be used.

- `NEXTCLOUD_DATA_DIR` (default: `/var/www/html/data`) Configures the data directory where nextcloud stores all files from the users.

One or more trusted domains can be set through environment variable, too. They will be added to the configuration after install.

- `NEXTCLOUD_TRUSTED_DOMAINS` (not set by default) Optional space-separated list of domains

The install and update script is only triggered when a default command is used (`apache-foreground` or `php-fpm`). If you use a custom command you have to enable the install / update with

- `NEXTCLOUD_UPDATE` (default: `0`)

You might want to make sure the htaccess is up to date after each container update. Especially on multiple swarm nodes as any discrepancy will make your server unusable.

- `NEXTCLOUD_INIT_HTACCESS` (not set by default) Set it to true to enable run `occ maintenance:update:htaccess` after container initialization.

If you want to use Redis you have to create a separate Redis container in your setup / in your docker-compose file. To inform Nextcloud about the Redis container, pass in the following parameters:

- `REDIS_HOST` (not set by default) Name of Redis container
- `REDIS_HOST_PORT` (default: `6379`) Optional port for Redis, only use for external Redis servers that run on non-standard ports.
- `REDIS_HOST_PASSWORD` (not set by default) Redis password

The use of Redis is recommended to prevent file locking problems. See the examples for further instructions.

To use an external SMTP server, you have to provide the connection details. To configure Nextcloud to use SMTP add:

- `SMTP_HOST` (not set by default): The hostname of the SMTP server.
- `SMTP_SECURE` (empty by default): Set to `ssl` to use SSL, or `tls` to use STARTTLS.
- `SMTP_PORT` (default: `465` for SSL and `25` for non-secure connections): Optional port for the SMTP connection. Use `587` for an alternative port for STARTTLS.
- `SMTP_AUTHTYPE` (default: `LOGIN`): The method used for authentication. Use `PLAIN` if no authentication is required.
- `SMTP_NAME` (empty by default): The username for the authentication.
- `SMTP_PASSWORD` (empty by default): The password for the authentication.
- `MAIL_FROM_ADDRESS` (not set by default): Set the local-part for the 'from' field in the emails sent by Nextcloud.
- `MAIL_DOMAIN` (not set by default): Set a different domain for the emails than the domain where Nextcloud is installed.

At least `SMTP_HOST`, `MAIL_FROM_ADDRESS` and `MAIL_DOMAIN` must be set for the configurations to be applied.

Check the Nextcloud documentation for other values to configure SMTP.

To use an external S3 compatible object store as primary storage, set the following variables:

- `OBJECTSTORE_S3_BUCKET`: The name of the bucket that Nextcloud should store the data in
- `OBJECTSTORE_S3_REGION`: The region that the S3 bucket resides in
- `OBJECTSTORE_S3_HOST`: The hostname of the object storage server
- `OBJECTSTORE_S3_PORT`: The port that the object storage server is being served over
- `OBJECTSTORE_S3_KEY`: AWS style access key
- `OBJECTSTORE_S3_SECRET`: AWS style secret access key
- `OBJECTSTORE_S3_STORAGE_CLASS`: The storage class to use when adding objects to the bucket
- `OBJECTSTORE_S3_SSL` (default: `true`): Whether or not SSL/TLS should be used to communicate with object storage server
- `OBJECTSTORE_S3_USEPATH_STYLE` (default: `false`): Not required for AWS S3
- `OBJECTSTORE_S3_LEGACYAUTH` (default: `false`): Not required for AWS S3
- `OBJECTSTORE_S3_OBJECT_PREFIX` (default: `urn:oid:`): Prefix to prepend to the fileid
- `OBJECTSTORE_S3_AUTOCREATE` (default: `true`): Create the container if it does not exist
- `OBJECTSTORE_S3_SSE_C_KEY` (not set by default): Base64 encoded key with a maximum length of 32 bytes for server side encryption (SSE-C)

Check the Nextcloud documentation for more information.

To use an external OpenStack Swift object store as primary storage, set the following variables:

- `OBJECTSTORE_SWIFT_URL`: The Swift identity (Keystone) endpoint
- `OBJECTSTORE_SWIFT_AUTOCREATE` (default: `false`): Whether or not Nextcloud should automatically create the Swift container
- `OBJECTSTORE_SWIFT_USER_NAME`: Swift username
- `OBJECTSTORE_SWIFT_USER_PASSWORD`: Swift user password
- `OBJECTSTORE_SWIFT_USER_DOMAIN` (default: `Default`): Swift user domain
- `OBJECTSTORE_SWIFT_PROJECT_NAME`: OpenStack project name
- `OBJECTSTORE_SWIFT_PROJECT_DOMAIN` (default: `Default`): OpenStack project domain
- `OBJECTSTORE_SWIFT_SERVICE_NAME` (default: `swift`): Swift service name
- `OBJECTSTORE_SWIFT_REGION`: Swift endpoint region
- `OBJECTSTORE_SWIFT_CONTAINER_NAME`: Swift container (bucket) that Nextcloud should store the data in

Check the Nextcloud documentation for more information.

To customize other PHP limits you can simply change the following variables:

- `PHP_MEMORY_LIMIT` (default `512M`) This sets the maximum amount of memory in bytes that a script is allowed to allocate. This is meant to help prevent poorly written scripts from eating up all available memory but it can prevent normal operation if set too tight.

- `PHP_UPLOAD_LIMIT` (default `512M`) This sets the upload limit (`post_max_size` and `upload_max_filesize`) for big files. Note that you may have to change other limits depending on your client, webserver or operating system. Check the Nextcloud documentation for more information.

To customize Apache max file upload limit you can change the following variable:

- `APACHE_BODY_LIMIT` (default `1073741824` [1GiB]) This restricts the total size of the HTTP request body sent from the client. It specifies the number of *bytes* that are allowed in a request body. A value of **0** means **unlimited**. Check the Nextcloud documentation for more information.

Auto configuration via hook folders

There are 5 hooks

- `pre-installation` Executed before the Nextcloud is installed/initiated
- `post-installation` Executed after the Nextcloud is installed/initiated
- `pre-upgrade` Executed before the Nextcloud is upgraded
- `post-upgrade` Executed after the Nextcloud is upgraded
- `before-starting` Executed before the Nextcloud starts

To use the hooks triggered by the `entrypoint` script, either

- Added your script(s) to the individual of the hook folder(s), which are located at the path `/docker-entrypoint-hooks.d` in the container
- Use volume(s) if you want to use script from the host system inside the container, see example.

Note: Only the script(s) located in a hook folder (not sub-folders), ending with `.sh` and marked as executable, will be executed.

Example: Mount using volumes

```
...
  app:
    image: nextcloud:stable

    volumes:
      - ./app-hooks/pre-installation:/docker-entrypoint-hooks.d/pre-installation
      - ./app-hooks/post-installation:/docker-entrypoint-hooks.d/post-installation
      - ./app-hooks/pre-upgrade:/docker-entrypoint-hooks.d/pre-upgrade
      - ./app-hooks/post-upgrade:/docker-entrypoint-hooks.d/post-upgrade
      - ./app-hooks/before-starting:/docker-entrypoint-hooks.d/before-starting
...
```

Using the apache image behind a reverse proxy and auto configure server host and protocol

The apache image will replace the remote addr (IP address visible to Nextcloud) with the IP address from `X-Real-IP` if the request is coming from a proxy in `10.0.0.0/8`, `172.16.0.0/12` or `192.168.0.0/16` by default. If you want Nextcloud to pick up the server host (`HTTP_X_FORWARDED_HOST`), protocol (`HTTP_X_FORWARDED_PROTO`) and client IP (`HTTP_X_FORWARDED_FOR`) from a trusted proxy, then disable rewrite IP and add the reverse proxy's IP address to `TRUSTED_PROXIES`.

- `APACHE_DISABLE_REWRITE_IP` (not set by default): Set to 1 to disable rewrite IP.
- `TRUSTED_PROXIES` (empty by default): A space-separated list of trusted proxies. CIDR notation is supported for IPv4.

If the `TRUSTED_PROXIES` approach does not work for you, try using fixed values for overwrite parameters.

- `OVERWRITEHOST` (empty by default): Set the hostname of the proxy. Can also specify a port.
- `OVERWRITEPROTOCOL` (empty by default): Set the protocol of the proxy, http or https.
- `OVERWRITECLIURL` (empty by default): Set the cli url of the proxy (e.g. https://mydnsname.example.com)
- `OVERWRITEWEBROOT` (empty by default): Set the absolute path of the proxy.
- `OVERWRITECONDADDR` (empty by default): Regex to overwrite the values dependent on the remote address.

Check the Nexcloud documentation for more details.

Keep in mind that once set, removing these environment variables won't remove these values from the configuration file, due to how Nextcloud merges configuration files together.

Running this image with docker-compose

The easiest way to get a fully featured and functional setup is using a `docker-compose` file. There are too many different possibilities to setup your system, so here are only some examples of what you have to look for.

At first, make sure you have chosen the right base image (fpm or apache) and added features you wanted (see below). In every case, you would want to add a database container and docker volumes to get easy access to your persistent data. When you want to have your server reachable from the internet, adding HTTPS-encryption is mandatory! See below for more information.

Base version - apache

This version will use the apache image and add a mariaDB container. The volumes are set to keep your data persistent. This setup provides **no ssl encryption** and is intended to run behind a proxy.

Make sure to pass in values for `MYSQL_ROOT_PASSWORD` and `MYSQL_PASSWORD` variables before you run this setup.

```
version: '2'

volumes:
  nextcloud:
  db:

services:
  db:
    image: mariadb:10.6
    restart: always
    command: --transaction-isolation=READ-COMMITTED --log-bin=binlog --binlog-format=ROW
    volumes:
      - db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_PASSWORD=
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud

  app:
    image: nextcloud
    restart: always
    ports:
      - 8080:80
    links:
      - db
    volumes:
      - nextcloud:/var/www/html
    environment:
      - MYSQL_PASSWORD=
      - MYSQL_DATABASE=nextcloud
```

```
      - MYSQL_USER=nextcloud
      - MYSQL_HOST=db
```

Then run `docker-compose up -d`, now you can access Nextcloud at http://localhost:8080/ from your host system.

Base version - FPM

When using the FPM image, you need another container that acts as web server on port 80 and proxies the requests to the Nextcloud container. In this example a simple nginx container is combined with the Nextcloud-fpm image and a MariaDB database container. The data is stored in docker volumes. The nginx container also needs access to static files from your Nextcloud installation. It gets access to all the volumes mounted to Nextcloud via the `volumes_from` option. The configuration for nginx is stored in the configuration file `nginx.conf`, that is mounted into the container. An example can be found in the examples section here.

As this setup does **not include encryption**, it should be run behind a proxy.

Make sure to pass in values for `MYSQL_ROOT_PASSWORD` and `MYSQL_PASSWORD` variables before you run this setup.

```
version: '2'

volumes:
  nextcloud:
  db:

services:
  db:
    image: mariadb:10.6
    restart: always
    command: --transaction-isolation=READ-COMMITTED --log-bin=binlog --binlog-format=ROW
    volumes:
      - db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_PASSWORD=
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud

  app:
    image: nextcloud:fpm
    restart: always
    links:
      - db
    volumes:
      - nextcloud:/var/www/html
```

```
    environment:
      - MYSQL_PASSWORD=
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud
      - MYSQL_HOST=db

  web:
    image: nginx
    restart: always
    ports:
      - 8080:80
    links:
      - app
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    volumes_from:
      - app
```

Then run `docker-compose up -d`, now you can access Nextcloud at http://localhost:8080/ from your host system.

Docker Secrets

As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to the previously listed environment variables, causing the initialization script to load the values for those variables from files present in the container. In particular, this can be used to load passwords from Docker secrets stored in `/run/secrets/<secret_name>` files. For example:

```
version: '3.2'

services:
  db:
    image: postgres
    restart: always
    volumes:
      - db:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB_FILE=/run/secrets/postgres_db
      - POSTGRES_USER_FILE=/run/secrets/postgres_user
      - POSTGRES_PASSWORD_FILE=/run/secrets/postgres_password
    secrets:
      - postgres_db
      - postgres_password
      - postgres_user

  app:
    image: nextcloud
    restart: always
```

```
    ports:
      - 8080:80
    volumes:
      - nextcloud:/var/www/html
    environment:
      - POSTGRES_HOST=db
      - POSTGRES_DB_FILE=/run/secrets/postgres_db
      - POSTGRES_USER_FILE=/run/secrets/postgres_user
      - POSTGRES_PASSWORD_FILE=/run/secrets/postgres_password
      - NEXTCLOUD_ADMIN_PASSWORD_FILE=/run/secrets/nextcloud_admin_password
      - NEXTCLOUD_ADMIN_USER_FILE=/run/secrets/nextcloud_admin_user
    depends_on:
      - db
    secrets:
      - nextcloud_admin_password
      - nextcloud_admin_user
      - postgres_db
      - postgres_password
      - postgres_user

volumes:
  db:
  nextcloud:

secrets:
  nextcloud_admin_password:
    file: ./nextcloud_admin_password.txt # put admin password in this file
  nextcloud_admin_user:
    file: ./nextcloud_admin_user.txt # put admin username in this file
  postgres_db:
    file: ./postgres_db.txt # put postgresql db name in this file
  postgres_password:
    file: ./postgres_password.txt # put postgresql password in this file
  postgres_user:
    file: ./postgres_user.txt # put postgresql username in this file
```

Currently, this is only supported for `NEXTCLOUD_ADMIN_PASSWORD`, `NEXTCLOUD_ADMIN_USER`, `MYSQL_DATABASE`, `MYSQL_PASSWORD`, `MYSQL_USER`, `POSTGRES_DB`, `POSTGRES_PASSWORD`, `POSTGRES_USER`, `REDIS_HOST_PASSWORD`, `SMTP_PASSWORD`, `OBJECTSTORE_S3_KEY`, and `OBJECTSTORE_S3_SECRET`.

If you set any group of values (i.e. all of `MYSQL_DATABASE_FILE`, `MYSQL_USER_FILE`, `MYSQL_PASSWORD_FILE`, `MYSQL_HOST`), the script will not use the corresponding group of environment variables (`MYSQL_DATABASE`, `MYSQL_USER`, `MYSQL_PASSWORD`, `MYSQL_HOST`).

Make your Nextcloud available from the internet

Until here, your Nextcloud is just available from your docker host. If you want your Nextcloud available from the internet adding SSL encryption is mandatory.

HTTPS - SSL encryption

There are many different possibilities to introduce encryption depending on your setup.

We recommend using a reverse proxy in front of your Nextcloud installation. Your Nextcloud will only be reachable through the proxy, which encrypts all traffic to the clients. You can mount your manually generated certificates to the proxy or use a fully automated solution which generates and renews the certificates for you.

In our examples section we have an example for a fully automated setup using a reverse proxy, a container for Let's Encrypt certificate handling, database and Nextcloud. It uses the popular nginx-proxy and docker-letsencrypt-nginx-proxy-companion containers. Please check the according documentations before using this setup.

First use

When you first access your Nextcloud, the setup wizard will appear and ask you to choose an administrator account username, password and the database connection. For the database use `db` as host and `nextcloud` as table and user name. Also enter the password you chose in your `docker-compose.yml` file.

Update to a newer version

Updating the Nextcloud container is done by pulling the new image, throwing away the old container and starting the new one.

It is only possible to upgrade one major version at a time. For example, if you want to upgrade from version 14 to 16, you will have to upgrade from version 14 to 15, then from 15 to 16.

Since all data is stored in volumes, nothing gets lost. The startup script will check for the version in your volume and the installed docker version. If it finds a mismatch, it automatically starts the upgrade process. Don't forget to add all the volumes to your new container, so it works as expected.

```
$ docker pull nextcloud
$ docker stop <your_nextcloud_container>
$ docker rm <your_nextcloud_container>
$ docker run <OPTIONS> -d nextcloud
```

Beware that you have to run the same command with the options that you used to initially start your Nextcloud. That includes volumes, port mapping.

When using docker-compose your compose file takes care of your configuration, so you just have to run:

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

Adding Features

A lot of people want to use additional functionality inside their Nextcloud installation. If the image does not include the packages you need, you can easily build your own image on top of it. Start your derived image with the `FROM` statement and add whatever you like.

```
FROM nextcloud:apache

RUN ...
```

The examples folder gives a few examples on how to add certain functionalities, like including the cron job, smb-support or imap-authentication.

If you use your own Dockerfile, you need to configure your docker-compose file accordingly. Switch out the `image` option with `build`. You have to specify the path to your Dockerfile. (in the example it's in the same directory next to the docker-compose file)

```
  app:
    build: .
    restart: always
```

```
    links:
      - db
    volumes:
      - data:/var/www/html/data
      - config:/var/www/html/config
      - apps:/var/www/html/apps
```

If you intend to use another command to run the image, make sure that you set `NEXTCLOUD_UPDATE=1` in your Dockerfile. Otherwise the installation and update will not work.

```
FROM nextcloud:apache

...

ENV NEXTCLOUD_UPDATE=1

CMD ["/usr/bin/supervisord"]
```

Updating your own derived image is also very simple. When a new version of the Nextcloud image is available run:

```
docker build -t your-name --pull .
docker run -d your-name
```

or for docker-compose:

```
docker-compose build --pull
docker-compose up -d
```

The `--pull` option tells docker to look for new versions of the base image. Then the build instructions inside your `Dockerfile` are run on top of the new image.

Migrating an existing installation

You're already using Nextcloud and want to switch to docker? Great! Here are some things to look out for:

1. Define your whole Nextcloud infrastructure in a `docker-compose` file and run it with `docker-compose up -d` to get the base installation, volumes and database. Work from there.
2. Restore your database from a mysqldump (nextcloud_db_1 is the name of your db container)

- To import from a MySQL dump use the following commands

```
docker cp ./database.dmp nextcloud_db_1:/dmp
docker-compose exec db sh -c "mysql --user USER --password PASSWORD nextcloud < /dmp"
docker-compose exec db rm /dmp
```

- To import from a PostgreSQL dump use to following commands

```
docker cp ./database.dmp nextcloud_db_1:/dmp
docker-compose exec db sh -c "psql -U USER --set ON_ERROR_STOP=on nextcloud < /dmp"
docker-compose exec db rm /dmp
```

3. Edit your config.php
   1. Set database connection
      - In case of MySQL database

      ```
      'dbhost' => 'db:3306',
      ```

      - In case of PostgreSQL database

      ```
      'dbhost' => 'db:5432',
      ```

   2. Make sure you have no configuration for the `apps_paths`. Delete lines like these

      ```
      'apps_paths' => array (
        ),
      ),
      ```

   3. Make sure to have the `apps` directory non writable and the `custom_apps` directory writable

      ```
      'apps_paths' => array (
        0 => 
        ),
        1 => 
        ),
      ),
      ```

   4. Make sure your data directory is set to /var/www/html/data

      ```
      'datadirectory' => '/var/www/html/data',
      ```

4. Copy your data (nextcloud_app_1 is the name of your Nextcloud container):

```
docker cp ./data/ nextcloud_app_1:/var/www/html/
docker-compose exec app chown -R www-data:www-data /var/www/html/data
docker cp ./theming/ nextcloud_app_1:/var/www/html/
docker-compose exec app chown -R www-data:www-data /var/www/html/theming
docker cp ./config/config.php nextcloud_app_1:/var/www/html/config
docker-compose exec app chown -R www-data:www-data /var/www/html/config
```

If you want to preserve the metadata of your files like timestamps, copy the data directly on the host to the named volume using plain `cp` like this:

```
cp --preserve --recursive ./data/ /path/to/nextcloudVolume/data
```

5. Copy only the custom apps you use (or simply redownload them from the web interface):

```
docker cp ./custom_apps/ nextcloud_data:/var/www/html/
docker-compose exec app chown -R www-data:www-data /var/www/html/custom_apps
```

Help (Questions / Issues)

If you have any questions or problems while using the image, please ask for assistance on the Help Forum first (https://help.nextcloud.com).

Also, most Nextcloud Server matters are covered in the Nextcloud Admin Manual which is routinely updated.

If you believe you've found a bug (or have an enhancement idea) in the image itself, please search for already reported bugs and enhancement ideas. If there is an existing open issue, you can either add to the discussion there or upvote to indicate you're impacted by (or interested in) the same issue. If you believe you've found a new bug, please create a new Issue so that others can try to reproduce it and remediation can be tracked.

Thanks for helping to make the Nextcloud community maintained micro-services image better!

Deploy Nextcloud with docker compose - (Medium site)

Link: https://chrisgrime.medium.com/deploy-nextcloud-with-docker-compose-935a76a5eb78
8 Setembro 2023.

In the spirit of DIY, experimenting, and learning, I set up a Nextcloud server to replace Google Drive and One Drive. Over the years my Nextcloud instance has become home to many of my backed up files, contacts, calendar, notes, and my server even has an office suite thanks to Collabora.

Nextcloud can be a pretty awesome, open source alternative to many services.

For better or for worse, I decided to deploy Nextcloud on my home server using a Docker container. If you’re looking to get experience in Docker or trying to find out how to set up a Docker Compose file to deploy Nextcloud you’re in the right spot.

The docker-compose.yml

```
---
version: '3'

services:
  nextcloud:
    image: nextcloud
    container_name: nextcloud
    restart: unless-stopped
    networks:
      - cloud
    depends_on:
      - nextclouddb
      - redis
    ports:
      - 8081:80
    volumes:
      - ./html:/var/www/html
      - ./custom_apps:/var/www/html/custom_apps
      - ./config:/var/www/html/config
      - ./data:/var/www/html/data
```

```
  environment:
    - PUID=1000
    - PGID=1000
    - TZ=America/Los_Angeles
    - MYSQL_DATABASE=nextcloud
    - MYSQL_USER=nextcloud
    - MYSQL_PASSWORD=dbpassword
    - MYSQL_HOST=nextclouddb
    - REDIS_HOST=redis

nextclouddb:
  image: mariadb
  container_name: nextcloud-db
  restart: unless-stopped
  command: --transaction-isolation=READ-COMMITTED --binlog-format=ROW
  networks:
    - cloud
  volumes:
    - ./nextclouddb:/var/lib/mysql
  environment:
    - PUID=1000
    - PGID=1000
    - TZ=America/Los_Angeles
    - MYSQL_RANDOM_ROOT_PASSWORD=true
    - MYSQL_PASSWORD=dbpassword
    - MYSQL_DATABASE=nextcloud
    - MYSQL_USER=nextcloud

collabora:
  image: collabora/code
  container_name: collabora
  restart: unless-stopped
  networks:
    - cloud
  environment:
    - PUID=1000
    - PGID=1000
    - TZ=America/Los_Angeles
    - password=password
    - username=nextcloud
    - domain=example.com
    - extra_params=--o:ssl.enable=true
  ports:
    - 9980:9980

redis:
  image: redis:alpine
  container_name: redis
  volumes:
    - ./redis:/data
  networks:
    - cloud
```

```
  nginx-proxy:
    image: 'jc21/nginx-proxy-manager:latest'
    container_name: nginx-proxy
    environment:
      - PUID=1000
      - PGID=1000
      - TZ=America/Los_Angeles
    restart: unless-stopped
    ports:
      - '80:80'
      - '81:81'
      - '443:443'
    volumes:
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt

networks:
  cloud:
    name: cloud
    driver: bridge
```

Alright. Lets break this down.

This Docker Compose file will deploy 5 containers. They are:

1. Nextcloud
2. MySQL database required for Nextcloud
3. Collabora — A awesome open source office suite similar to google docs. Collabora Office also has mobile apps.
4. Redis — Memory Caching. If you're going to rely on next cloud for important files, I highly recommend setting up Redis.
5. Nginx Proxy Manager — A reverse proxy manager to handle incoming requests to the server.

Docker Compose

A little explanation for what each line does.

Nextcloud:

```
nextcloud:
    image: nextcloud # The image that will be used. The official nextcloud docker
    container_name: nextcloud # Just the name of the container. Help you identify it
    restart: unless-stopped # If something happens like the container crashes then we want the container to start u
    networks: # Link all the containers through the network "cloud"
      - cloud
    depends_on: # Wait for the database and redis containers before starting nextcloud
      - nextclouddb
```

```
    - redis
  ports: # If you have multiple web service on your server you need to change the port. I am directing nextcloud f
    - 8081:80
  volumes: # These are important. This will map a file directory inside the container to a directory on your actual
    - ./html:/var/www/html # Map the /var/www/html directory in the container to the html folder in the same folde
    - ./custom_apps:/var/www/html/custom_apps # These volumes allow us to easily interact with the files in the c
    - ./config:/var/www/html/config
    - ./data:/var/www/html/data
  environment:
    - PUID=1000 # The user ids. Most likely both should be 1000. Incorrectly setting these will led to file permissio
    - PGID=1000 # Set these to whatever your user is.
    - TZ=America/Los_Angeles # Set this to your timezone
    - MYSQL_DATABASE=nextcloud # This is the database information we will set up in the next section
    - MYSQL_USER=nextcloud
    - MYSQL_PASSWORD=dbpassword
    - MYSQL_HOST=nextclouddb
    - REDIS_HOST=redis # The redis container to use
```

Nextcloud Database:

```
nextclouddb:
  image: mariadb # offical mariadb image
  container_name: nextcloud-db
  restart: unless-stopped
  command: --transaction-isolation=READ-COMMITTED --binlog-format=ROW # I honestly cant remember. If you k
  networks:
    - cloud
  volumes:
    - ./nextclouddb:/var/lib/mysql
  environment:
    - PUID=1000 # Should be the same as the other containers
    - PGID=1000
    - TZ=America/Los_Angeles
    - MYSQL_RANDOM_ROOT_PASSWORD=true
    - MYSQL_PASSWORD=dbpassword # Same information that was entered in the nextcloud portion
    - MYSQL_DATABASE=nextcloud
    - MYSQL_USER=nextcloud
```

Collabora (Optional, but really cool):

```
collabora:
  image: collabora/code:latest
  container_name: collabora
  restart: unless-stopped
  networks:
    - cloud
  environment:
    - PUID=1000
    - PGID=1000
    - TZ=America/Los_Angeles # Should be the same as the others
    - password=password
    - username=nextcloud
```

```
    - domain=example.com # domain your nextcloud is on
    - extra_params=--o:ssl.enable=true # Use if have ssl. You should.
  ports:
    - 9980:9980
```

Redis (Optional, but seriously, just add it. You will run into file lock issues eventually and Redis will prevent it):

```
redis:
  image: redis:alpine
  container_name: redis
  volumes:
    - ./redis:/data
  networks:
    - cloud
```

Nginx Proxy Manager:

```
nginx-proxy:
  image: 'jc21/nginx-proxy-manager:latest'
  container_name: nginx-proxy
  environment:
    - PUID=1000
    - PGID=1000
    - TZ=America/Los_Angeles
  restart: unless-stopped
  ports:
    - '80:80'
    - '81:81'
    - '443:443'
  volumes:
    - ./data:/data
    - ./letsencrypt:/etc/letsencrypt
```

Nginx Proxy manager is an awesome program that will direct incoming traffic to the server towards Nextcloud. Add a new host.

On the details panel enter your domain name.

Scheme = http, Forward Hostname = whatever your machines local IP address is, (for example 192.168.1.35) and forward port = 80.

Cache Assets, Block Common Exploits, Websockets support all should be on.

Next on the Custom Locations tab we will enable caldav and carddav to allow remote access to your calendars and contacts.

Location 1:

1. location = /.well-known/caldav

2. scheme = html
3. Forward Hostname = <local IP>/ remote.php/dav
4. Forward Port 80

Location 2:

1. location = /.well-known/carddav
2. scheme = html
3. Forward Hostname = <local IP>/ remote.php/dav
4. Forward Port 80

Additional Configuration

Now if you are setting up Nextcloud to work with your custom domain you will need to open the config.php file and change trusted domains to whatever your domain is.

If you are trying to access your Nextcloud from your network it could be useful to add your Nextcloud's local IP address.

```
'trusted_domains' =>
 array (
 0 => 'example.com',
 1 => '192.168.1.12:8081',
 ),
 'overwritehost' => 'example.com',
 'overwriteprotocol' => 'https',
```

Since Niginx Proxy Manager is set up, the following needs to be added to the config.php file:

```
'default_phone_region' => 'US',
 'trustedproxies' =>
 array (
 0 => 'NginxProxyManager',
 1 => '192.168.0.145',
 ),
```

To solve some of the warnings you need to do the following:

```
 'default_phone_region' => 'US',
```

To set up mail alerts you will need to add the following to the config file. The values will need to be obtained from your email provider.

```
 'mail_from_address' => 'user', # insert your emails user
 'mail_smtpmode' => 'smtp',
```

```
'mail_sendmailmode' => 'smtp',
'mail_domain' => 'example.com', # Your email domain
'mail_smtphost' => 'smtp.example.com',
'mail_smtpport' => '465',
'mail_smtpauth' => 1,
'mail_smtpsecure' => 'ssl',
'mail_smtpname' => 'user@example.com',
'mail_smtppassword' => 'secretpassword',
```

Run the container

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

Congratulations! Nextcloud is setup using docker containers and docker compose! Let me know if you have any questions.

Potential Issues

- If you run into a 502 Gateway Error try clearing the cookies in your browser for the domain your server is hosted. It works most of the time for me.
- Make sure to update the docker images on a regular basis. Nextcloud in a Docker cannot whole number skip versions. For example if your version is 24 and the newest version is 26, **DO NOT** update straight to 26. I learned this the hard way. Update first to 25. So run “docker-compose pull” somewhat regularly.

Instalação Nextcloud Docker

Procedimentos de instalação Nextcloud Docker

Nextcloud Docker oficial instalação

Link: https://github.com/nextcloud/docker

18/05/2025

What is Nextcloud?

A safe home for all your data. Access & share your files, calendars, contacts, mail & more from any device, on your terms.

logo

⚠⚠⚠ This image is maintained by community volunteers and designed for expert use. For quick and easy deployment that supports the full set of Nextcloud Hub features, use the Nextcloud All-in-One docker container maintained by Nextcloud GmbH.

How to use this image

This image is designed to be used in a micro-service environment. There are two versions of the image you can choose from.

The `apache` tag contains a full Nextcloud installation including an apache web server. It is designed to be easy to use and gets you running pretty fast. This is also the default for the `latest` tag and version tags that are not further specified.

The second option is a `fpm` container. It is based on the php-fpm image and runs a fastCGI-Process that serves your Nextcloud page. To use this image it must be combined with any webserver that can proxy the http requests to the FastCGI-port of the container.

Try in PWD

Getting help

Most Nextcloud Server administrative matters are covered in the official Nextcloud Admin Manual or other official Nextcloud documentation (which are all routinely updated).

Discourse Users Discourse Posts

If you have any problems or usage questions while using the image, please ask for assistance on the Nextcloud Community Help Forum rather than reporting them as "bugs" (unless they are bugs of course). This helps the maintainers (who are volunteers) remain focused on making the image better (rather than responding solely to one-on-one support issues). (Tip: Some of the maintainers are also regular responders to help requests on the community help forum.)

For the image specifically, we provide some simple deployment examples as well as some more extensive deployment examples. In addition, the community help forum has a "how-to" section with further examples of other peoples' container based Nextcloud stacks.

Below you'll find the main documentation for using this image.

Using the apache image

The apache image contains a webserver and exposes port 80. To start the container type:

```
$ docker run -d -p 8080:80 nextcloud
```

Now you can access Nextcloud at http://localhost:8080/ from your host system.

WARNING: This example is only suitable for limited testing purposes. Please read on to understand how the image handles storing your data and other aspects you need to consider to establish a full Nextcloud stack.

Using the fpm image

To use the fpm image, you need an additional web server, such as nginx, that can proxy http-request to the fpm-port of the container. For fpm connection this container exposes port 9000. In most cases, you might want to use another container or your host as proxy. If you use your host you can address your Nextcloud container directly on port 9000. If you use another container, make sure that you add them to the same docker network (via `docker run --network <NAME> ...` or a `docker compose` file). In both cases you don't want to map the fpm port to your host.

```
$ docker run -d nextcloud:fpm
```

As the fastCGI-Process is not capable of serving static files (style sheets, images, ...), the webserver needs access to these files. This can be achieved with the `volumes-from` option. You can find more information in the docker compose section.

Using an external database

By default, this container uses SQLite for data storage but the Nextcloud setup wizard (appears on first run) allows connecting to an existing MySQL/MariaDB or PostgreSQL database. You can also link a database container, e. g. `--link my-mysql:mysql`, and then use `mysql` as the database host on setup. More info is in the docker compose section.

Persistent data

The Nextcloud installation and all data beyond what lives in the database (file uploads, etc.) are stored in the unnamed docker volume volume `/var/www/html`. The docker daemon will store that data within the docker directory `/var/lib/docker/volumes/...`. That means your data is saved even if the container crashes, is stopped or deleted.

A named Docker volume or a mounted host directory should be used for upgrades and backups. To achieve this, you need one volume for your database container and one for Nextcloud.

Nextcloud:

- `/var/www/html/` folder where all Nextcloud data lives

```
$ docker run -d \
-v nextcloud:/var/www/html \
nextcloud
```

Database:

- `/var/lib/mysql` MySQL / MariaDB Data
- `/var/lib/postgresql/data` PostgreSQL Data

```
$ docker run -d \
-v db:/var/lib/mysql \
mariadb:lts
```

Additional volumes

If you want to get fine grained access to your individual files, you can mount additional volumes for data, config, your theme and custom apps. The `data`, `config` files are stored in respective subfolders inside `/var/www/html/`. The apps are split into core `apps` (which are shipped with Nextcloud and you don't need to take care of) and a `custom_apps` folder. If you use a custom theme it would go into the `themes` subfolder.

Overview of the folders that can be mounted as volumes:

- `/var/www/html` Main folder, needed for updating
- `/var/www/html/custom_apps` installed / modified apps
- `/var/www/html/config` local configuration

- `/var/www/html/data` the actual data of your Nextcloud
- `/var/www/html/themes/<YOUR_CUSTOM_THEME>` theming/branding

If you want to use named volumes for all of these, it would look like this:

```
$ docker run -d \
-v nextcloud:/var/www/html \
-v custom_apps:/var/www/html/custom_apps \
-v config:/var/www/html/config \
-v data:/var/www/html/data \
-v theme:/var/www/html/themes/<YOUR_CUSTOM_THEME> \
nextcloud
```

If you'd prefer to use bind mounts instead of named volumes, for instance, when working with different device or network mounts for user data files and configuration:

```
$ docker run -d \
-v /path/on/host/to/folder/nextcloud:/var/www/html \
-v /path/on/host/to/folder/custom_apps:/var/www/html/custom_apps \
-v /path/on/host/to/folder/config:/var/www/html/config \
-v /path/on/host/to/folder/data:/var/www/html/data \
-v /path/on/host/to/folder/theme:/var/www/html/themes/<YOUR_CUSTOM_THEME> \
nextcloud
```

Here’s the same example using Docker's more detailed `--mount`. Note that with `-v` or `--volume`, the specified folders are created automatically if they don't exist. However, when using `--mount` for bind mounts, the directories must already exist on the host, or Docker will return an error.

```
$ docker run -d \
--mount type=bind,source=/path/on/host/to/folder/nextcloud,target=/var/www/html \
--mount type=bind,source=/path/on/host/to/folder/custom_apps,target=/var/www/html/custom_apps \
--mount type=bind,source=/path/on/host/to/folder/config,target=/var/www/html/config \
--mount type=bind,source=/path/on/host/to/folder/data,target=/var/www/html/data \
--mount type=bind,source=/path/on/host/to/folder/theme,target=/var/www/html/themes/<YOUR_CUSTOM_THEME>
nextcloud
```

The examples above use figurative directory `/path/on/host/to/folder/` for bind mounts. Please modify the paths by using either a relative or absolute path.

NOTE: Do not confuse the `apps` and `custom_apps` folders. These folders contain different sets of apps, and mixing them will result in a broken installation. The former contains "shipped" apps, which come with Nextcloud Server. The latter contains apps you install from the App Store.

Custom volumes

If mounting additional volumes under `/var/www/html`, you should consider:

- Confirming that upgrade.exclude contains the files and folders that should persist during installation and upgrades; or
- Mounting storage volumes to locations outside of `/var/www/html`.

Data inside the main folder (`/var/www/html`) will be overridden/removed during installation and upgrades, unless listed in upgrade.exclude. The additional volumes officially supported are already in that list, but custom volumes will need to be added by you. We suggest mounting custom storage volumes outside of `/var/www/html` and if possible read-only so that making this adjustment is unnecessary. If you must do so, however, you may build a custom image with a modified `/upgrade.exclude` file that incorporates your custom volume(s).

Running as an arbitrary user / file permissions / changing the default container user

The default user within a container is root (uid = 0). By default, processes inside the container will expect to have root privileges. Network services will drop privileges and use `www-data` to serve requests.

Depending on your volumes configuration, this can lead to permission issues. You can address this by running the container with a different default user. When changing the default user, the image will no longer assume it has root privileges and will run all processes under the specified uid. To accomplish this, use the `--user` / `user` option in your container environment.

See:

- https://docs.docker.com/engine/containers/run/#user
- https://github.com/docker-library/docs/tree/master/php#running-as-an-arbitrary-user
- https://docs.podman.io/en/stable/markdown/podman-run.1.html#user-u-user-group

Accessing the Nextcloud command-line interface (`occ`)

To use the Nextcloud command-line interface (aka. `occ` command):

```
$ docker exec -it --user www-data CONTAINER_ID php occ
```

or for docker compose:

```
$ docker compose exec --user www-data app php occ
```

or even shorter:

```
$ docker compose exec -u33 app ./occ
```

Note: substitute `82` for `33` if using the Alpine-based images.

Viewing the Nextcloud configuration (`config.php`)

The image takes advantage of Nextcloud's Multiple config.php support to inject auto configuration environment variables and set image specific config values.

This means that merely viewing your `config.php` will not give you an accurate view of your running config. Instead, you should use Nextcloud's `occ config:list system` command to get get a complete view of your merged configuration. This has the added benefit of automatically omitting sensitive values such as passwords and secrets from the output by default (e.g. useful for shared publicly or assisting others when troubleshooting or reporting a bug).

```
$ docker compose exec -u33 app ./occ config:list system
```

The `--private` flag can also be specified, in order to output all configuration values including passwords and secrets.

Auto configuration via environment variables

The Nextcloud image supports auto configuration of the Nextcloud Server installation via environment variables. You can preconfigure everything that would otherwise be prompted for by the Nextcloud Installation Wizard (as well as a few other key parameters relevant to initial installation).

Database parameters

To enable auto configuration, define your database connection via the following environment variables. If you set any group of values (i.e. all of `MYSQL_DATABASE`, `MYSQL_USER`, `MYSQL_PASSWORD`, `MYSQL_HOST`), they will not be requested via the Installation Wizard on first run.

You must specify all of the environment variables for a given database or the database environment variables defaults to SQLITE. ONLY use one database type!

SQLite:

- `SQLITE_DATABASE` Name of the database using sqlite

MYSQL/MariaDB:

- `MYSQL_DATABASE` Name of the database using mysql / mariadb.
- `MYSQL_USER` Username for the database using mysql / mariadb.
- `MYSQL_PASSWORD` Password for the database user using mysql / mariadb.
- `MYSQL_HOST` Hostname of the database server using mysql / mariadb.

PostgreSQL:

- `POSTGRES_DB` Name of the database using postgres.
- `POSTGRES_USER` Username for the database using postgres.
- `POSTGRES_PASSWORD` Password for the database user using postgres.
- `POSTGRES_HOST` Hostname of the database server using postgres.

As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to the previously listed environment variables, causing the initialization script to load the values for those variables from files present in the container. See Docker secrets section below for details.

Initial admin account

If you specify all the variables for your database type (above), you can also auto configure your initial admin user and password (only works if you set both):

- `NEXTCLOUD_ADMIN_USER` Name of the Nextcloud admin user.
- `NEXTCLOUD_ADMIN_PASSWORD` Password for the Nextcloud admin user.

Specifying a complete database and admin credential config set at initial deployment will result in a fully automated installation (i.e. bypassing the web-based Nextcloud Installation Wizard).

Addition parameters may also be set at installation time and are described below.

Custom Data directory (`datadirectory`)

If you don't want to use the default data directory (`datadirectory`) location, you can set a custom one:

- `NEXTCLOUD_DATA_DIR` (default: `/var/www/html/data`) Configures the data directory where nextcloud stores all files from the users.

Trusted domains (`trusted_domains`)

One or more trusted domains can be set through environment variable, too. They will be added to the configuration after install.

- `NEXTCLOUD_TRUSTED_DOMAINS` (not set by default) Optional space-separated list of domains

Image specific

The install and update script is only triggered when a default command is used (`apache-foreground` or `php-fpm`). If you use a custom command you have to enable the install / update with

- `NEXTCLOUD_UPDATE` (default: `0`)

You might want to make sure the htaccess is up to date after each container update. Especially on multiple swarm nodes as any discrepancy will make your server unusable.

- `NEXTCLOUD_INIT_HTACCESS` (not set by default) Set it to true to enable run `occ maintenance:update:htaccess` after container initialization.

Redis Memory Caching

To use Redis for memory caching as well as PHP session storage, specify the following values and also add a Redis container to your stack. See the examples for further instructions.

- `REDIS_HOST` (not set by default) Name of Redis container
- `REDIS_HOST_PORT` (default: `6379`) Optional port for Redis, only use for external Redis servers that run on non-standard ports.
- `REDIS_HOST_USER` (not set by default) Optional username for Redis, only use for external Redis servers that require a user.
- `REDIS_HOST_PASSWORD` (not set by default) Redis password

Check the Nextcloud documentation for more information.

E-mail (SMTP) Configuration

To use an external SMTP server, you have to provide the connection details. Note that if you configure these values via Docker, you should **not** use the Nexcloud Web UI to configure external SMTP server parameters. Conversely, if you prefer to use the Web UI, do **not** set these variables here (because these variables will override whatever you attempt to set in the Web UI for these parameters). To configure Nextcloud to use SMTP add:

- `SMTP_HOST` (not set by default): The hostname of the SMTP server.
- `SMTP_SECURE` (empty by default): Set to `ssl` to use SSL, or `tls` to use STARTTLS.
- `SMTP_PORT` (default: `465` for SSL and `25` for non-secure connections): Optional port for the SMTP connection. Use `587` for an alternative port for STARTTLS.
- `SMTP_AUTHTYPE` (default: `LOGIN`): The method used for authentication. Use `PLAIN` if no authentication is required.
- `SMTP_NAME` (empty by default): The username for the authentication.
- `SMTP_PASSWORD` (empty by default): The password for the authentication.
- `MAIL_FROM_ADDRESS` (not set by default): Set the local-part for the 'from' field in the emails sent by Nextcloud.
- `MAIL_DOMAIN` (not set by default): Set a different domain for the emails than the domain where Nextcloud is installed.

At a minimum, the `SMTP_HOST`, `MAIL_FROM_ADDRESS` and `MAIL_DOMAIN` parameters must be defined.

Check the Nextcloud documentation for other values to configure SMTP.

Object Storage (Primary Storage)

By default, Nextcloud stores all files in `/var/www/html/data/` (or whatever custom data directory you've configured). Nextcloud also allows the use of object storages (like OpenStack Swift or any Amazon S3-compatible implementation) as *Primary Storage*. This semi-replaces the default storage of files in the data directory. Note: This data directory might still be used for compatibility reasons and still needs to exist. Check the Nextcloud documentation for more information.

To use an external S3 compatible object store as primary storage, set the following variables:

- `OBJECTSTORE_S3_BUCKET`: The name of the bucket that Nextcloud should store the data in
- `OBJECTSTORE_S3_REGION`: The region that the S3 bucket resides in
- `OBJECTSTORE_S3_HOST`: The hostname of the object storage server
- `OBJECTSTORE_S3_PORT`: The port that the object storage server is being served over
- `OBJECTSTORE_S3_KEY`: AWS style access key
- `OBJECTSTORE_S3_SECRET`: AWS style secret access key
- `OBJECTSTORE_S3_STORAGE_CLASS`: The storage class to use when adding objects to the bucket
- `OBJECTSTORE_S3_SSL` (default: `true`): Whether or not SSL/TLS should be used to communicate with object storage server
- `OBJECTSTORE_S3_USEPATH_STYLE` (default: `false`): Not required for AWS S3
- `OBJECTSTORE_S3_LEGACYAUTH` (default: `false`): Not required for AWS S3
- `OBJECTSTORE_S3_OBJECT_PREFIX` (default: `urn:oid:`): Prefix to prepend to the fileid
- `OBJECTSTORE_S3_AUTOCREATE` (default: `true`): Create the container if it does not exist
- `OBJECTSTORE_S3_SSE_C_KEY` (not set by default): Base64 encoded key with a maximum length of 32 bytes for server side encryption (SSE-C)

Check the Nextcloud documentation for more information.

To use an external OpenStack Swift object store as primary storage, set the following variables:

- `OBJECTSTORE_SWIFT_URL`: The Swift identity (Keystone) endpoint
- `OBJECTSTORE_SWIFT_AUTOCREATE` (default: `false`): Whether or not Nextcloud should automatically create the Swift container
- `OBJECTSTORE_SWIFT_USER_NAME`: Swift username
- `OBJECTSTORE_SWIFT_USER_PASSWORD`: Swift user password
- `OBJECTSTORE_SWIFT_USER_DOMAIN` (default: `Default`): Swift user domain
- `OBJECTSTORE_SWIFT_PROJECT_NAME`: OpenStack project name
- `OBJECTSTORE_SWIFT_PROJECT_DOMAIN` (default: `Default`): OpenStack project domain
- `OBJECTSTORE_SWIFT_SERVICE_NAME` (default: `swift`): Swift service name
- `OBJECTSTORE_SWIFT_REGION`: Swift endpoint region
- `OBJECTSTORE_SWIFT_CONTAINER_NAME`: Swift container (bucket) that Nextcloud should store the data in

Check the Nextcloud documentation for more information.

PHP Configuration

To customize PHP limits you can change the following variables:

- `PHP_MEMORY_LIMIT` (default `512M`) This sets the maximum amount of memory in bytes that a script is allowed to allocate. This is meant to help prevent poorly written scripts from eating up all available memory but it can prevent normal operation if set too tight.
- `PHP_UPLOAD_LIMIT` (default `512M`) This sets the upload limit (`post_max_size` and `upload_max_filesize`) for big files. Note that you may have to change other limits depending on your client, webserver or operating system. Check the Nextcloud documentation for more information.
- `PHP_OPCACHE_MEMORY_CONSUMPTION` (default `128`) This sets the `opcache.memory_consumption` value. It's the size of the shared memory storage used by OPcache, in megabytes.

Apache Configuration

To customize the Apache max file upload limit you can change the following variable:

- `APACHE_BODY_LIMIT` (default `1073741824` [1GiB]) This restricts the total size of the HTTP request body sent from the client. It specifies the number of *bytes* that are allowed in a request body. A value of **0** means **unlimited**.

Check the Nextcloud documentation for more information.

Using the image behind a reverse proxy and specifying the server host and protocol

By default, the apache image will replace the remote addr (IP address visible to Nextcloud) with the IP address from `X-Real-IP` if the request is coming from a reverse proxy in `10.0.0.0/8`, `172.16.0.0/12` or `192.168.0.0/16`. If you want Nextcloud to pick up the server host (`HTTP_X_FORWARDED_HOST`), protocol (`HTTP_X_FORWARDED_PROTO`) and client IP (`HTTP_X_FORWARDED_FOR`) from a trusted proxy, then disable rewrite IP and add the reverse proxy's IP address to `TRUSTED_PROXIES`.

- `APACHE_DISABLE_REWRITE_IP` (not set by default): Set to 1 to disable rewrite IP.
- `TRUSTED_PROXIES` (empty by default): A space-separated list of trusted proxies. CIDR notation is supported for IPv4.

If the `TRUSTED_PROXIES` approach does not work for you, try using fixed values for overwrite parameters.

- `OVERWRITEHOST` (empty by default): Set the hostname of the proxy. Can also specify a port.

- `OVERWRITEPROTOCOL` (empty by default): Set the protocol of the proxy, http or https.
- `OVERWRITECLIURL` (empty by default): Set the cli url of the proxy (e.g. https://mydnsname.example.com)
- `OVERWRITEWEBROOT` (empty by default): Set the absolute path of the proxy.
- `OVERWRITECONDADDR` (empty by default): Regex to overwrite the values dependent on the remote address.
- `FORWARDED_FOR_HEADERS` (empty by default): HTTP headers with the original client IP address

Check the Nexcloud documentation for more details.

Keep in mind that once set at install time, removing these environment variables later won't remove them from your `config/config.php`, due to how Nextcloud generates and merges the initial configuration at installation time. They can still, however, be removed manually from your `config/config.php`.

Handling `Warning: /var/www/html/config/$cfgFile differs from the latest version of this image at /usr/src/nextcloud/config/$cfgFile` (aka: Auto configuration and Nextcloud updates)

The image comes with special config files for Nextcloud that set parameters specific to containerized usage (e.g. `upgrade-disable-web.config.php`) or enable auto configuration via environment variables (e.g. `reverse-proxy.config.php`). Not keeping these files up-to-date when this warning appears may cause certain auto configuration environment variables to be ignored or the image to not work as documented or expected.

During a fresh Nextcloud installation, the latest version (from the image) of these files are copied into `/var/www/html/config` so that they are stored within your container's persistent storage and picked up by Nextcloud alongside your local configuration.

The copied files, however, are **not** automatically overwritten whenever you update your environment with a newer Nextcloud image. This is to prevent local changes in `/var/www/html/config` from being unexpectedly overwritten. This may lead to your image-specific configuration files becoming outdated and image functionality not matching that which is documented.

Within each image, the latest version of these config files are located in `/usr/src/nextcloud/config`.

A warning will be generated in the container log output when outdated image-specific configuration files are detected at startup in a running container. When you see this warning, you should manually compare (or copy) the files from `/usr/src/nextcloud/config` to `/var/www/html/config`. A command to copy these configs would e.g. be:

```
docker exec <container-name> sh -c "cp /usr/src/nextcloud/config/*.php /var/www/html/config"
```

As long as you have not modified any of the provided config files in `/var/www/html/config` (other than `config.php`) or only added new ones with names that do not conflict with the image specific ones, copying the new ones into place should be safe (but check the source path `/usr/src/nextcloud/config` for any newly named config files to avoid new overlaps just in case).

Auto configuration via hook folders

There are 5 hooks

- `pre-installation` Executed before the Nextcloud is installed/initiated
- `post-installation` Executed after the Nextcloud is installed/initiated
- `pre-upgrade` Executed before the Nextcloud is upgraded
- `post-upgrade` Executed after the Nextcloud is upgraded
- `before-starting` Executed before the Nextcloud starts

To use the hooks triggered by the `entrypoint` script, either

- Added your script(s) to the individual of the hook folder(s), which are located at the path `/docker-entrypoint-hooks.d` in the container
- Use volume(s) if you want to use script from the host system inside the container, see example.

Note: Only the script(s) located in a hook folder (not sub-folders), ending with `.sh` and marked as executable, will be executed.

Example: Mount using volumes

```
...
  app:
    image: nextcloud:stable

    volumes:
      - ./app-hooks/pre-installation:/docker-entrypoint-hooks.d/pre-installation
      - ./app-hooks/post-installation:/docker-entrypoint-hooks.d/post-installation
      - ./app-hooks/pre-upgrade:/docker-entrypoint-hooks.d/pre-upgrade
      - ./app-hooks/post-upgrade:/docker-entrypoint-hooks.d/post-upgrade
      - ./app-hooks/before-starting:/docker-entrypoint-hooks.d/before-starting
...
```

Running this image with `docker compose`

The easiest way to get a fully featured and functional setup is using a `compose.yaml` file. There are too many different possibilities to setup your system, so here are only some examples of what you have to look for.

At first, make sure you have chosen the right base image (fpm or apache) and added features you wanted (see below). In every case, you would want to add a database container and docker volumes to get easy access to your persistent data. When you want to have your server reachable from the internet, adding HTTPS-encryption is mandatory! See below for more information.

Base version - apache

This version will use the apache variant and add a MariaDB container. The volumes are set to keep your data persistent. This setup provides **no TLS encryption** and is intended to run behind a proxy.

Make sure to pass in values for `MYSQL_ROOT_PASSWORD` and `MYSQL_PASSWORD` variables before you run this setup.

```
services:
  # Note: MariaDB is external service. You can find more information about the configuration here:
  # https://hub.docker.com/_/mariadb
  db:
    # Note: Check the recommend version here: https://docs.nextcloud.com/server/latest/admin_manual/installatior
    image: mariadb:lts
    restart: always
    command: --transaction-isolation=READ-COMMITTED
    volumes:
      - db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_PASSWORD=
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud

  # Note: Redis is an external service. You can find more information about the configuration here:
  # https://hub.docker.com/_/redis
  redis:
    image: redis:alpine
    restart: always

  app:
    image: nextcloud
    restart: always
    ports:
      - 8080:80
    depends_on:
      - redis
      - db
    volumes:
      - nextcloud:/var/www/html
    environment:
      - MYSQL_PASSWORD=
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud
      - MYSQL_HOST=db
```

```
volumes:
  nextcloud:
  db:
```

Then run `docker compose up -d`, now you can access Nextcloud at http://localhost:8080/ from your host system.

Base version - FPM

When using the FPM image, you need another container that acts as web server on port 80 and proxies the requests to the Nextcloud container. In this example a simple nginx container is combined with the Nextcloud-fpm image and a MariaDB database container. The data is stored in docker volumes. The nginx container also needs access to static files from your Nextcloud installation. It gets access to all the volumes mounted to Nextcloud via the `volumes` option. The configuration for nginx is stored in the configuration file `nginx.conf`, that is mounted into the container. An example can be found in the examples section here.

This setup provides **no TLS encryption** and is intended to run behind a reverse proxy.

Make sure to pass in values for `MYSQL_ROOT_PASSWORD` and `MYSQL_PASSWORD` variables before you run this setup.

```
services:
  # Note: MariaDB is an external service. You can find more information about the configuration here:
  # https://hub.docker.com/_/mariadb
  db:
    # Note: Check the recommend version here: https://docs.nextcloud.com/server/latest/admin_manual/installatior
    image: mariadb:lts
    restart: always
    command: --transaction-isolation=READ-COMMITTED
    volumes:
      - db:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_PASSWORD=
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud

  # Note: Redis is an external service. You can find more information about the configuration here:
  # https://hub.docker.com/_/redis
  redis:
    image: redis:alpine
    restart: always

  app:
    image: nextcloud:fpm
    restart: always
    depends_on:
```

```
      - redis
      - db
    volumes:
      - nextcloud:/var/www/html
    environment:
      - MYSQL_PASSWORD=
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud
      - MYSQL_HOST=db

  # Note: Nginx is an external service. You can find more information about the configuration here:
  # https://hub.docker.com/_/nginx/
  web:
    image: nginx:alpine-slim
    restart: always
    ports:
      - 8080:80
    depends_on:
      - app
    volumes:
      # https://docs.nextcloud.com/server/latest/admin_manual/installation/nginx.html
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
    volumes_from:
      - app

volumes:
  nextcloud:
  db:
```

Then run `docker compose up -d`, now you can access Nextcloud at http://localhost:8080/ from your host system.

Docker Secrets

As an alternative to passing sensitive information via environment variables, `_FILE` may be appended to some the previously listed environment variables, causing the initialization script to load the values for those variables from files present in the container. In particular, this can be used to load passwords from Docker secrets stored in `/run/secrets/<secret_name>` files.

Currently, this is only supported for `NEXTCLOUD_ADMIN_PASSWORD`, `NEXTCLOUD_ADMIN_USER`, `MYSQL_DATABASE`, `MYSQL_PASSWORD`, `MYSQL_USER`, `POSTGRES_DB`, `POSTGRES_PASSWORD`, `POSTGRES_USER`, `REDIS_HOST_PASSWORD`, `SMTP_PASSWORD`, `OBJECTSTORE_S3_KEY`, and `OBJECTSTORE_S3_SECRET`.

If you set any group of `_FILE` based values (i.e. all of `MYSQL_DATABASE_FILE`, `MYSQL_USER_FILE`, `MYSQL_PASSWORD_FILE`), their non-`_FILE` counterparts will be ignored (`MYSQL_DATABASE`, `MYSQL_USER`, `MYSQL_PASSWORD`).

Any files containing secrets must be readable by the UID the container is running Nextcloud as (i.e. `www-data` / `33`).

Example:

```
services:
  # Note: PostgreSQL is external service. You can find more information about the configuration here:
  # https://hub.docker.com/_/postgres
  db:
    # Note: Check the recommend version here: https://docs.nextcloud.com/server/latest/admin_manual/installatior
    image: postgres:alpine
    restart: always
    volumes:
      - db:/var/lib/postgresql/data
    environment:
      - POSTGRES_DB_FILE=/run/secrets/postgres_db
      - POSTGRES_USER_FILE=/run/secrets/postgres_user
      - POSTGRES_PASSWORD_FILE=/run/secrets/postgres_password
    secrets:
      - postgres_db
      - postgres_password
      - postgres_user
  # Note: Redis is an external service. You can find more information about the configuration here:
  # https://hub.docker.com/_/redis
  redis:
    image: redis:alpine
    restart: always

  app:
    image: nextcloud
    restart: always
    ports:
      - 8080:80
    volumes:
      - nextcloud:/var/www/html
    environment:
      - POSTGRES_HOST=db
      - POSTGRES_DB_FILE=/run/secrets/postgres_db
      - POSTGRES_USER_FILE=/run/secrets/postgres_user
      - POSTGRES_PASSWORD_FILE=/run/secrets/postgres_password
      - NEXTCLOUD_ADMIN_PASSWORD_FILE=/run/secrets/nextcloud_admin_password
      - NEXTCLOUD_ADMIN_USER_FILE=/run/secrets/nextcloud_admin_user
    depends_on:
      - redis
      - db
    secrets:
      - nextcloud_admin_password
      - nextcloud_admin_user
      - postgres_db
      - postgres_password
      - postgres_user

volumes:
```

```
  db:
  nextcloud:

secrets:
  nextcloud_admin_password:
    file: ./nextcloud_admin_password.txt # put admin password in this file
  nextcloud_admin_user:
    file: ./nextcloud_admin_user.txt # put admin username in this file
  postgres_db:
    file: ./postgres_db.txt # put postgresql db name in this file
  postgres_password:
    file: ./postgres_password.txt # put postgresql password in this file
  postgres_user:
    file: ./postgres_user.txt # put postgresql username in this file
```

Make your Nextcloud available from the internet

Until here, your Nextcloud is just available from your docker host. If you want your Nextcloud available from the internet adding SSL encryption is mandatory.

HTTPS - SSL encryption

There are many different possibilities to introduce encryption depending on your setup.

We recommend using a reverse proxy in front of your Nextcloud installation. Your Nextcloud will only be reachable through the proxy, which encrypts all traffic to the clients. You can mount your manually generated certificates to the proxy or use a fully automated solution which generates and renews the certificates for you.

In our examples section we have an example for a fully automated setup using a reverse proxy, a container for Let's Encrypt certificate handling, database and Nextcloud. It uses the popular nginx-proxy and acme-companion containers. Please check the according documentations before using this setup.

First use

When you first access your Nextcloud, the setup wizard will appear and ask you to choose an administrator account username, password and the database connection (unless of course you've provided all the necessary auto-config config values ahead of time).

For the database use `db` as host and `nextcloud` as table and user name. Also enter the password you chose in your `compose.yaml` file.

Update to a newer version

Updating the Nextcloud container is done by pulling the new image, throwing away the old container and starting the new one.

It is only possible to upgrade one major version at a time. For example, if you want to upgrade from version 14 to 16, you will have to upgrade from version 14 to 15, then from 15 to 16.

Since all data is stored in volumes, nothing gets lost. The startup script will check for the version in your volume and the installed docker version. If it finds a mismatch, it automatically starts the upgrade process. Don't forget to add all the volumes to your new container, so it works as expected.

```
$ docker pull nextcloud
$ docker stop <your_nextcloud_container>
$ docker rm <your_nextcloud_container>
$ docker run <OPTIONS> -d nextcloud
```

Beware that you have to run the same command with the options that you used to initially start your Nextcloud. That includes volumes, port mapping.

When using docker compose your compose file takes care of your configuration, so you just have to run:

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

Adding Features

A lot of people want to use additional functionality inside their Nextcloud installation. If the image does not include the packages you need, you can easily build your own image on top of it. Start your derived image with the `FROM` statement and add whatever you like.

```
FROM nextcloud:apache

RUN ...
```

The examples folder gives a few examples on how to add certain functionalities, like including the cron job, smb-support or imap-authentication.

If you use your own Dockerfile, you need to configure your docker compose file accordingly. Switch out the `image` option with `build`. You have to specify the path to your Dockerfile. (in the example it's in the same directory next to the `compose.yaml` file)

```
app:
  build: .
  restart: always
```

```
    depends_on:
      - db
    volumes:
      - data:/var/www/html/data
      - config:/var/www/html/config
      - apps:/var/www/html/apps
```

If you intend to use another command to run the image, make sure that you set `NEXTCLOUD_UPDATE=1` in your Dockerfile. Otherwise the installation and update will not work.

```
FROM nextcloud:apache

...

ENV NEXTCLOUD_UPDATE=1

CMD ["/usr/bin/supervisord"]
```

Updating your own derived image is also very simple. When a new version of the Nextcloud image is available run:

```
docker build -t your-name --pull .
docker run -d your-name
```

or for docker compose:

```
docker compose build --pull
docker compose up -d
```

The `--pull` option tells docker to look for new versions of the base image. Then the build instructions inside your `Dockerfile` are run on top of the new image.

Migrating an existing installation

You're already using Nextcloud and want to switch to docker? Great! Here are some things to look out for:

1. Define your whole Nextcloud infrastructure in a `compose.yaml` file and run it with `docker compose up -d` to get the base installation, volumes and database. Work from there.
2. Restore your database from a mysqldump (db is the name of your database container / service name)
   - To import from a MySQL dump use the following commands

     ```
     docker compose cp ./database.dmp db:/dmp
     docker compose exec db sh -c "mysql --user USER --password PASSWORD nextcloud < /dmp"
     docker compose exec db rm /dmp
     ```

   - To import from a PostgreSQL dump use to following commands

```
docker compose cp ./database.dmp db:/dmp
docker compose exec db sh -c "psql -U USER --set ON_ERROR_STOP=on nextcloud < /dmp"
docker compose exec db rm /dmp
```

3. Edit your config.php
   1. Set database connection
      - In case of MySQL database

```
'dbhost' => 'db:3306',
```

      - In case of PostgreSQL database

```
'dbhost' => 'db:5432',
```

   2. Make sure you have no configuration for the `apps_paths`. Delete lines like these

```
'apps_paths' => array (
  ),
),
```

   3. Make sure to have the `apps` directory non writable and the `custom_apps` directory writable

```
'apps_paths' => array (
  0 => 
  ),
  1 => 
  ),
),
```

   4. Make sure your data directory is set to /var/www/html/data

```
'datadirectory' => '/var/www/html/data',
```

4. Copy your data (`app` is the name of your Nextcloud container / service name):

```
docker compose cp ./data/ app:/var/www/html/
docker compose exec app chown -R www-data:www-data /var/www/html/data
docker compose cp ./theming/ app:/var/www/html/
docker compose exec app chown -R www-data:www-data /var/www/html/theming
docker compose cp ./config/config.php app:/var/www/html/config
docker compose exec app chown -R www-data:www-data /var/www/html/config
```

If you want to preserve the metadata of your files like timestamps, copy the data directly

on the host to the named volume using plain `cp` like this:

```
cp --preserve --recursive ./data/ /path/to/nextcloudVolume/data
```

5. Copy only the custom apps you use (or simply redownload them from the web interface):

```
docker compose cp ./custom_apps/ app:/var/www/html/
docker compose exec app chown -R www-data:www-data /var/www/html/custom_apps
```

Migrating from a non-Alpine image to an Alpine image

If you already use one of our non-Alpine images, but want to switch to an Alpine-based image, you may experience permissions problems with your existing volumes. This is because the Alpine images uses a different user ID for `www-data`. So, you must change the ownership of the `/var/www/html` (or `$NEXTCLOUD_DATA_DIR`) folder to be compatible with Alpine:

```
docker exec container-name chown -R www-data:root /var/www/html
```

After changing the permissions, restart the container and the permission errors should disappear.

Reporting bugs or suggesting enhancements

If you believe you've found a bug in the image itself (or have an enhancement idea specific to the image), please search for already reported bugs and enhancement ideas.

If there is a relevant existing open issue, you can either add to the discussion there or upvote it to indicate you're impacted by (or interested in) the same issue.

If you believe you've found a new bug, please create a new Issue so that others can try to reproduce it and remediation can be tracked.

GitHub Issues or Pull Requests GitHub Issues or Pull Requests by label
GitHub Issues or Pull Requests by label GitHub Issues or Pull Requests by label

If you have any problems or usage questions while using the image, please ask for assistance on the Nextcloud Community Help Forum rather than reporting them as "bugs" (unless they really are bugs of course). This helps the maintainers (who are volunteers) remain focused on making the image better (rather than responding solely to one-on-one support issues). (Tip: Some of the maintainers are also regular responders to help requests on the Nextcloud Community Help Forum.)

Discourse Users Discourse Posts

Most Nextcloud Server matters are covered in the official Nextcloud Admin Manual or the other official Nextcloud documentation (which are routinely updated).