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!

Revision #1
Created 12 May 2024 12:17:17 by Administrador
Updated 4 July 2024 18:33:49 by Administrador