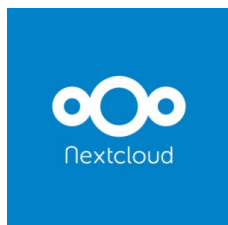




# Nextcloud auf Ubuntu Server 20.04 LTS mit nginx, MariaDB, PHP, Let's Encrypt, Redis und Fail2ban

11. Mai 2020 Jan [Home-Server](#), [182](#)

In diesem Artikel wird die Installation und Konfiguration von **Nextcloud auf Ubuntu Server 20.04 LTS** („Focal Fossa“) mit **nginx**, **MariaDB**, **PHP** und **Let's Encrypt** beschrieben. Ebenfalls kommen **Redis**, **Fail2ban** und **ufw** zur Verbesserung der Sicherheit und Performance zum Einsatz.

Zum Thema Nextcloud auf Ubuntu Server gab es in diesem Blog bereits zahlreiche Artikel. Die letzte Anleitung [Nextcloud auf Ubuntu Server 18.04 LTS mit nginx, MariaDB, PHP, Let's Encrypt, Redis und Fail2ban](#) hat bereits ein ähnliches Setup beschrieben, basierte jedoch auf der Ubuntu-Version 18.04 LTS. Nach gut zwei Jahren ist nun eine neue LTS-Version von Ubuntu erschienen, so dass diese Artikel-Serie mit der neusten Version der Distribution

fortgeführt wird.

Daneben bietet dieses Tutorial einen ganz neuen Ansatz: In vorherigen Artikeln zu diesem Thema wurde Nextcloud immer in einem Unterverzeichnis des Webservers installiert. Hier stieg allerdings die Nachfrage nach einem Setup, bei dem Nextcloud direkt im Root-Verzeichnis einer Domain installiert werden kann. Als positiver Nebeneffekt wird damit auch die Webserver-Konfiguration ein wenig einfacher. Daher wird die eigene Cloud in diesem Artikel direkt im Web-Root der Domain installiert und nicht mehr in einem Unterverzeichnis.

## Update-Historie (letztes Update: 22.06.2020)

### Inhalt [\[hide\]](#)

#### 1 Motivation, Voraussetzungen und Konzept

- 1.1 Ziele
- 1.2 Zeitaufwand und Programm-Versionen
- 1.3 Änderungen gegenüber älteren Nextcloud-Artikeln
- 1.4 Voraussetzungen
  - 1.4.1 Betriebssystem und Hardware
  - 1.4.2 Zugriff Per SSH
  - 1.4.3 Domain und DynDNS/feste IP
  - 1.4.4 Port-Forwarding
  - 1.4.5 Internet-Anschluss
  - 1.4.6 Root-Rechte
- 1.5 Konzept
  - 1.5.1 LEMP-Stack
  - 1.5.2 Virtuelle Hosts und Konfigurations-Dateien von nginx
  - 1.5.3 Hinweis bei abweichender Verzeichnisstruktur bei nginx

#### 2 Installation und Konfiguration benötigter Programme

- 2.1 Update des Betriebssystems
- 2.2 Programm-Installation
  - 2.2.1 Exkurs: Ubuntu-Paketquellen und Hersteller-Paketquellen
  - 2.2.2 nginx
  - 2.2.3 MariaDB
  - 2.2.4 PHP
  - 2.2.5 Let's Encrypt/acme.sh
- 2.3 Konfiguration der Programme
  - 2.3.1 Konfiguration nginx
  - 2.3.2 Konfiguration MariaDB
  - 2.3.3 Konfiguration PHP

#### 3 Generierung der Zertifikate für HTTPS

- 3.1 Vorbereiten der Verzeichnisstruktur
- 3.2 Anlegen des ersten virtuellen Hosts (HTTP-Gateway)
- 3.3 Überprüfung des Webservers vor der Zertifikats-Generierung
- 3.4 Generierung der Zertifikate
- 3.5 Erneuerung der Zertifikate
- 3.6 Diffie-Hellman-Parameter

#### 4 Webserver für Nextcloud vorbereiten

- 4.1 SSL-Konfiguration
- 4.2 Header-Konfiguration
- 4.3 Virtueller Host für Nextcloud

## IT-Services





Cloud &amp; IT Services

## Direktlinks

[searx-Instanz von decatec.de](#)[Windows Server Advanced Power Management](#)[MagicPacket](#)[Portable WebDAV Library](#)[Objektive und Kameras für die Infrarot-Fotografie](#)[Update-Historie](#)

## Beliebt

 [Nextcloud auf Ubuntu Server 20.04 LTS mit nginx, MariaDB, PHP, Let's Encrypt, Redis und Fail2ban](#)  
1.2k views | 34 comments

 [Nextcloud Talk mit eigenem Signaling-Server \(High Performance Backend\)](#)  
555 views | 14 comments

[Linux: Einfach E-Mails versenden mit msmtpp](#)  
323 views | 8 comments

[Nextcloud auf Ubuntu Server 18.04 LTS mit nginx, MariaDB, PHP, Let's Encrypt, Redis und Fail2ban](#)  
310 views | 5 comments

[Jitsi Meet: Videokonferenz-System unter Ubuntu Server mit nginx](#)  
230 views | 7 comments

[Nextcloud: Updates richtig durchführen](#)  
204 views | 2 comments

[Nextcloud: Backups erstellen und wiederherstellen – manuell oder per Skript](#)  
196 views | 2 comments

[Nextcloud: Direkter Zugriff auf Dateien über das Dateisystem](#)  
195 views | 0 comments

[Nextcloud – Tipps & Tricks für Admins \(und Benutzer\)](#)  
184 views | 0 comments

[Nextcloud Zwei-Faktor-Authentifizierung mit Nitrokey FIDO2](#)  
159 views | 4 comments

## 5 Installation Nextcloud

### 5.1 Download

### 5.2 Anlegen des Datenverzeichnisses

### 5.3 Datenbank für Nextcloud anlegen

### 5.4 Nextcloud-Setup

### 5.5 Warnungen im Admin-Bereich

### 5.6 Optimieren der Nextcloud-Konfiguration

### 5.7 Installation/Einrichtung Redis

#### 5.7.1 Installation und Konfiguration Redis

#### 5.7.2 Einbinden von Redis in Nextcloud

### 5.8 Cronjob für Nextcloud einrichten

### 5.9 Weitere Konfiguration Nextcloud

## 6 Optimierung der Server-Umgebung für Nextcloud

### 6.1 ufw

### 6.2 Fail2ban

#### 6.2.1 Konfigurations-Dateien von Fail2ban

#### 6.2.2 Deaktivieren des Nextcloud Brute-Force-Schnutzes

#### 6.2.3 Installation und Konfiguration Fail2ban

#### 6.2.4 Optional: E-Mail-Versand durch Fail2ban

#### 6.2.5 Test der Funktionalität

## 7 Überprüfung der Sicherheit

### 7.1 Qualys SSL Labs

### 7.2 Mozilla Observatory

### 7.3 Nextcloud Security Scan

## 8 FAQ

## 9 Troubleshooting

## 10 Nextcloud: Ein sicherer Ort für all deine Daten

## 11 Weiterführende Artikel

## 12 Links

## Schlagwörter

.NET Framework acme.sh Apps Backup

Bildbearbeitung C# Cloud Docker EBV

Fail2ban Fotografie **Home-**

**Server** HTTPS Let's

Encrypt Lightroom **Linux**

MagicPacket MariaDB **Nextcloud**

Nextcloud App **nginx** Office ownCloud

PHP Programmierung Reverse Proxy

Security Sicherheit **Software** SSL

**Tipps & Tricks** TLS Tutorial

**Ubuntu** Ubuntu Server

Virtualisierung Visual Studio Wake On

LAN Wake On WAN WebDAV Webserver

**Windows** Windows 10 Windows

Phone **Windows** Server

**Advanced** Power

**Management**

## Kategorien

Allgemein (3)

Fotografie (9)

Infrarot (2)

Home-Server (45)

IT (12)

Linux (7)

Programmierung (11)

Raspberry Pi (4)

Software (27)

MagicPacket (5)

Windows Server Advanced Power  
Management (23)

## Meta

Anmelden

Feed der Einträge

Kommentare-Feed

WordPress.org

## Motivation, Voraussetzungen und Konzept

Dieser Artikel wird sicherlich wieder lang und umfangreich werden. Wie immer möchte ich hier **kein** „Copy & Paste Tutorial“ bereit stellen, in dem einfach nur eine Reihe von (Kommandozeilen-)Befehlen aufgeführt wird. Das hauptsächliche Ziel des Artikels ist es daher wieder, **Wissen zu vermitteln**: Dem Leser soll ein solides Grundlagenwissen vermittelt werden, so dass er sich zu jeder Zeit bewusst ist, warum die Dinge wie hier im Artikel skizziert umgesetzt wurden. Das Wissen um Zusammenhänge und Hintergründe ist ebenfalls bei auftretenden Problemen sehr wichtig. Denn wer eine eigene Cloud betreibt (und hier evtl. noch eigene Anforderungen umsetzen möchte), der wird froh sein, wenn auftretende Probleme eigenständig analysiert und gelöst werden können.

## Ziele

Mit diesem Tutorial werden konkret folgende Ziele verfolgt:

- Installation der eigenen **Nextcloud** auf **Ubuntu Server 20.04 LTS** mit **nginx**, **MariaDB** und **PHP** (LEMP-Stack).
- **Erhöhte Sicherheit** der Cloud (PHP-Konfiguration, SSL, Nextcloud-Konfiguration laut [Nextcloud Administration Manual](#)).
- **Verschlüsselte Verbindung** zur eigenen Cloud mittels **HTTPS**. Dazu kommt ein Zertifikat von **Let's Encrypt** zum Einsatz (ECDSA- und RSA-Zertifikate im Hybrid-Betrieb mit TLSv1.3).
- Nextcloud soll in einem **Root-Verzeichnis** des Webservers laufen und direkt über die URL <https://nextcloud.meinedomain.de> erreichbar sein.
- In der Admin-Oberfläche von Nextcloud sollen **keine Warnungen** angezeigt werden.
- Das Datenverzeichnis von Nextcloud soll **außerhalb** des www-Verzeichnisses liegen.
- Verbesserung der **Performance** durch Verwendung von **Redis** für das Transactional File Locking.
- Absicherung gegen Brute-Force-Attacken mit **Fail2ban**.
- Einrichtung der Firewall **ufw**.

## Zeitaufwand und Programm-Versionen

**Zeitaufwand:** ca. 3 Stunden

### Eingesetzte Programm-Versionen:

- Ubuntu Server 20.04 LTS („Focal Fossa“)
- Nextcloud 18.0.3
- nginx 1.17
- MariaDB 10.4
- PHP 7.4

- Redis 5.0.7
- Fail2ban 0.11.1
- OpenSSL 1.1.1f

## Änderungen gegenüber älteren Nextcloud-Artikeln

In diesem Blog gab es bereits einige Artikel zur Installation von Nextcloud auf Ubuntu Server. Im aktuellen Tutorial gibt es nun allerdings einige Änderungen, die das gesamte Setup betreffen – natürlich abgesehen, dass eine neue Version von Ubuntu Server zum Einsatz kommt.

Folgende Punkte wurden nun im Vergleich zu den vorherigen Nextcloud-Artikeln geändert:

- **Nextcloud wird nun nicht mehr in einem Unterverzeichnis, sondern direkt im Root-Verzeichnis von nginx installiert:**

Dies macht die komplette Webserver-Konfiguration wesentlich einfacher, da kein Gateway-Host mehr benötigt wird, der alle Requests entgegen nehmen und weiterleiten muss. Dies verhindert allerdings nicht die Installation weiterer Webanwendungen. Diese benötigen im hier gezeigten Setup jedoch jeweils eigene (Sub-)Domains.

- **Verzicht auf die Konfiguration von open\_basedir.**

Mittels `open_basedir` kann in einer PHP-Umgebung der Zugriff von PHP auf bestimmte Verzeichnisse beschränkt werden. In der Praxis sorgte diese Einschränkung doch immer wieder für Fehler/Probleme. Da dieses Thema in der heutigen Zeit nicht mehr so relevant für die Sicherheit ist, wird die Konfiguration von `open_basedir` nun ausgelassen.

- **Die Generierung der HTTPS-Zertifikate setzt nun konsequent auf acme.sh mit TLSv1.2 und TLSv1.3 im Hybrid-Betrieb (RSA- und ECDSA-Zertifikate parallel).**

Die älteren Artikel hatten dieses Konzept nicht so konsequent umgesetzt, so dass man die Informationen zu dem Thema ggf. aus weiteren Artikeln zusammen suchen musste. Das Vorgehen dazu wird nun hier direkt im Artikel ausführlich behandelt.

Neben diesen offensichtlichen Änderungen sind aber ebenfalls noch viele kleine Verbesserungen in diesen Artikel eingeflossen.

## Voraussetzungen

### Betriebssystem und Hardware

Als Betriebssystem kommt Ubuntu Server 20.04 LTS („Focal Fossa“) zum Einsatz. Generell ist hier der Einsatz einer LTS-Version ([Long Term Support](#)) zu empfehlen, da durch den verlängerten Support-Zeitraum (in diesem Fall fünf Jahre – bis April 2025) ein solches System über eine lange Zeit betrieben werden kann, ohne dass ein Distributions-Update zwingend erforderlich ist.

Das Betriebssystem sollte bereits installiert sein. Eine genaue Anleitung der Betriebssystem-Installation würde sicherlich den Rahmen des Artikels sprengen.

Alle gezeigten Schritte können allerdings auch mit anderen Linux-Distributionen (z.B. Debian) umgesetzt werden. Evtl. sind hier dann kleinere Anpassungen an einigen wenigen Schritten notwendig.

Als Hardware braucht man prinzipiell nur eine Maschine, auf der Linux läuft. Wenn die Anforderungen nicht besonders hoch sind, kann das Tutorial auch auf einem Kleinstrechner wie dem [Raspberry Pi \(Affiliate-Link\)](#) umgesetzt werden. Wird mehr Leistung benötigt, bietet sich z.B. auch eine [Intel NUC \(Affiliate Link\)](#) an. Die Cloud muss allerdings auch nicht zwingend in den eigenen vier Wänden betrieben werden: Wer einen eigenen Root-Server gemietet hat, kann sich ebenfalls an diesem Artikel orientieren.

### Zugriff Per SSH

Ein Server läuft normalerweise „headless“, d.h. ohne angeschlossenen Monitor, Maus und Tastatur. Der Zugriff findet dann in der Regel über [SSH](#) statt. Daher sollte man im Rahmen des Ubuntu-Setups gleich einen SSH-Server mit installieren.

### Domain und DynDNS/feste IP

Die eigene Cloud soll später ja auch aus dem Internet aus erreichbar sein. Daher ist zunächst eine Domain erforderlich. Im Rahmen des Artikels nutze ich die beispielhafte (Sub-)Domain [nextcloud.meinedomain.de](#).

Diese Domain muss zunächst auf den eigenen Internet-Anschluss verweisen. Wer einen Anschluss mit fester IP hat (meistens Business-Anschlüsse), tut sich hier sehr leicht. In diesem Fall reicht es aus, den [A-Record](#) (IPv4) und [AAAA-Record](#) (IPv6) von der Domain auf die IPs des Anschlusses zu setzen. Dies wird normalerweise in den Einstellungen beim Domain-Provider erledigt.

Wenn keine feste IP-Adresse vorhanden ist – was bei den meisten privaten Internet-Anschlüssen der Fall ist – muss man auf [DynDNS](#) setzen. Dies sorgt dafür, dass die (öffentliche) IP des Servers/Routers immer auf eine DynDNS-Domain gemappt wird. Dies ist notwendig, da die meisten Provider nach der 24-stündigen Zwangstrennung dem Kunden einen neue IP-Adresse zuweisen.

Ebenfalls benötigt man schon für die Erzeugung der SSL-Zertifikate zwingend eine Domain, da Zertifikate nicht auf IP-Adressen ausgestellt werden können.

Hierzu benötigt man einen sog. DynDNS-Dienst. Hier gibt es zahlreiche kostenlose Anbieter. Ein paar empfehlenswerte Dienste wären dabei:

- [DDNS Service](#)
- [deSEC](#)
- [dynv6](#)

Trotzdem kann es – gerade bei kostenlosen Diensten – immer mal wieder zu Problemen kommen. Eine meist bessere Alternative ist daher der Einsatz eines bezahlten Angebots, welches dann meistens zuverlässiger ist. Dazu bieten viele Webhoster auch DynDNS-Dienste an. Empfehlen kann ich hier den Hoster [All-Inkl.com \(Affiliate Link\)](#): Hier ist ein DynDNS-Dienst bereits um Webhosting-Paket „Privat Plus“ enthalten.

Im Router muss dann diese DynDNS-Domain noch konfiguriert werden, damit sich dieser korrekt am DynDNS-Dienst anmeldet. Erst nach diesem Schritt ist das eigene Netzwerk per DynDNS-Domain erreichbar. Das Vorgehen zum konfigurieren vom DynDNS hängt vom verwendeten Router-Modell, aber auch vom DynDNS-Dienst ab. Eine Informationsquelle sind hier die Hersteller-Seiten des Routers (z.B. [AVM](#)), aber auch die Websites der jeweiligen DynDNS-Dienste (z.B. [ddnss.de](#)) bieten meistens Hinweise zu verschiedenen Router-Modellen.

## Port-Forwarding

Damit der (lokale) Webserver später auch aus dem Internet erreichbar ist, muss ein sog. [Port-Forwarding](#) im Router für die Ports 80 (HTTP) und 443 (HTTPS) für die IP-Adresse des Servers (in diesem Beispiel: 192.168.178.60) konfiguriert werden. Da sich das Vorgehen dazu wieder von Router zu Router unterscheidet, kann hier ebenfalls keine konkrete Anleitung erfolgen. Oftmals beschreiben die Hersteller das genaue Vorgehen auf ihren Internet-Seiten. Beispielsweise findet man die Vorgehensweise für die Einrichtung von Port-Forwardings für eine FritzBox auf den [Hilfeseiten vom AVM](#).

## Internet-Anschluss

Der Internet-Anschluss muss auch gewisse Voraussetzungen erfüllen. Hierbei darf es sich **nicht** um einen [Dual-Stack Lite \(DS-Lite\)](#) Anschluss handeln. Dies ist eine Technik, bei der der Endkunde keine „echte“ IPv4 Adresse bekommt, sondern nur eine IPv6-Adresse. Verbindungen die über IPv4 laufen, werden dann über IPv6 getunnelt. Leider setzen durch die gegebene Knappheit an IPv4-Adressen immer mehr Provider auf DS-Lite.

Mit einem DS-Lite-Anschluss ist der Betrieb eines selbstgehosteten Webdienstes leider nicht direkt möglich.

**Daher wird zum Hosten einer eigenen Nextcloud zwingend ein „echter“ IPv4-Anschluss benötigt. Mit einem DS-Lite-Anschluss ist es nicht möglich, mit der hier gezeigten Vorgehensweise eine eigene Cloud aufzusetzen.**

Wer doch nur einen DS-Lite-Anschluss hat, kann evtl. eine Zusatz-Option für einen echten IPv4-Anschluss beim Provider dazu buchen. Falls der Provider eine solche Option nicht direkt anbietet, kann es oftmals auch helfen, beim Provider anzurufen und nach einer solchen Option zu fragen, die dann evtl. manuell gebucht werden kann.

Falls dies nicht möglich ist, kann man nur einen Umweg über einen sog. [Portmapper](#) gehen. Diese Lösung ist allerdings sehr aufwendig und verursacht zusätzliche Kosten.

## Root-Rechte

Für die Installation und die Konfiguration der benötigten Programmen sind unter Linux meist Root-Rechte erforderlich. Unter Ubuntu kann jeder Befehl durch das Voranstellen von „sudo“ mit Root-Rechten ausgeführt werden. Damit dieses „sudo“ nicht immer mit angegeben werden muss, ist es für die Installation empfehlenswert, sich am Anfang der Installation/Konfiguration einmalig Root-Rechte mit dem Befehl `sudo -s` zu verschaffen.

## Konzept

Der folgende Abschnitt erklärt das Konzept, welches diesem Tutorial zugrunde liegt.

### LEMP-Stack

Wer sich im Bereich Webhosting ein wenig auskennt, hat sicher schon mal etwas vom sog. LAMP-Stack gehört. Darunter versteht man eine Reihe an Programmen, die oftmals zusammen eingesetzt werden, wenn es um das Hosten dynamischer Webseiten geht: **Linux** (Betriebssystem), **Apache** (Webserver), **MySQL** (Datenbank) und **PHP** (Skriptsprache). Setzt man die Anfangsbuchstaben dieser Programme zusammen, versteht man, warum dies als [LAMP-Stack](#) bezeichnet wird.

Ich setze im Rahmen des Artikels jedoch auf einen sog. [LEMP-Stack](#): Dies eine eine weitere Variante dieses Software-Pakets, bei der zunächst ebenfalls Linux als Grundlage zum Einsatz kommt. Allerdings nutzt man nun keinen Apache, sondern [nginx](#) als Webserver. Ebenso setze ich auf [MariaDB](#) als Datenbanksystem (statt MySQL). Was gleich bleibt ist der Einsatz von PHP. Die Bezeichnung LEMP-Stack ergibt sich wieder aus dem Anfangsbuchstaben der Programme (das „E“ kommt von der Aussprache von nginx: „Engine-X“).

Warum nutze ich nun abweichend vom Standard einen anderen Webserver und ein anderes Datenbanksystem? Dies bringt meiner Meinung nach folgende Vorteile:

- Der Webserver [nginx](#) ist im Allgemeinen etwas ressourcenschonender als Apache. Letzterer erstellt pro (Client-)Verbindung neue [Threads](#) bzw. [Prozesse](#), über die die einzelnen Webrequest abgearbeitet werden. Die

Erzeugung neuer Threads/Prozesse ist allerdings teuer, da vergleichsweise viel Rechenleistung benötigt wird. nginx dagegen arbeitet mit einem sog. Thread-Pool: Hier werden beim Start des Webservers bereits mehrere Threads angelegt. Diese arbeiten dann die Webrequests ab, können aber im weiteren Verlauf wieder verwendet werden. Dies benötigt im Allgemeinen nicht so viele Ressourcen. Darüber hinaus finde ich die Konfiguration von nginx einfacher als die von Apache. Dies ist allerdings meine persönliche Meinung und mit Sicherheit Geschmackssache.

- Das Datenbanksystem **MariaDB** ging aus einem **Fork** von MySQL hervor und ist zu diesem **binärkompatibel**. MariaDB ist daher ein sog. Drop-In-Replacement für MySQL (sozusagen ein 1:1 Ersatz). Deshalb spielt es für eine Anwendung keine Rolle, ob das dahinter liegende Datenbanksystem nun MySQL oder MariaDB ist. Ebenfalls sind alle Tools und Programme, die für MySQL entwickelt wurde auch automatisch mit MariaDB kompatibel. Für unser Vorhaben spielt es daher keine große Rolle, ob nun MySQL oder MariaDB zum Einsatz kommt. Im Gegensatz zu MySQL steht hinter MariaDB jedoch kein großes Unternehmen (bei MySQL ist dies Oracle), sondern es handelt sich dabei um „echte“ Open-Source-Software. Das ist auch der Grund, warum mittlerweile viele Linux-Distributionen MariaDB als Standard-Datenbanksystem einsetzen. Da mir dieses System zukunftsicherer erscheint, nutze ich daher im Rahmen des Artikels MariaDB als Datenbanksystem.

## Virtuelle Hosts und Konfigurations-Dateien von nginx

Bevor es losgeht, noch ein paar Anmerkungen zu den Konfigurationsdateien bei nginx. Der Webserver verwendet sog. **virtuelle Hosts** (vHosts). Dies sind zunächst einmal reine Textdateien, die die Konfiguration für genau eine Website oder Anwendung beschreibt. Folgende Dateien/Verzeichnisse sind dabei wichtig:

- */etc/nginx/nginx.conf*: Dies ist die globale Konfiguration von nginx. Hier werden alle globalen Einstellungen definiert, die für alle Websites gelten sollen.
- */etc/nginx/conf.d*: In diesem Verzeichnis erwartet nginx die virtuellen Hosts und lädt diese beim Start. vHosts müssen in diesem Verzeichnis liegen und mit der Dateiendung \*.conf gespeichert werden. Andere Dateien, die nicht mit .conf enden, werden beim Start des Webservers ignoriert.
- Andere Verzeichnisse/Includes: Neben dieses fest definierten Verzeichnissen können aber auch andere Konfigurations-Dateien zum Einsatz kommen. Diese können irgendwo gespeichert sein und in jeder nginx-Konfiguration/vHost eingebunden werden. Ich nutze für solche „Snippets“ immer das Verzeichnis */etc/nginx/snippets*. Was es damit auf sich hat und wie die Dateien inkludiert werden, wird im weiteren Verlauf des Artikels deutlich.

## Hinweis bei abweichender Verzeichnisstruktur bei nginx

Es kann vorkommen, dass je nach verwendeter nginx-Version die Verzeichnisstruktur etwas anders ist. Nach der Installation sollte daher schnell überprüft werden, wo der default-vHost zu finden ist. Wenn dieses das oben genannten Verzeichnis ist (*/etc/nginx/conf.d*), kann dieser hier Abschnitt übersprungen werden.

Wenn der default-vHost allerdings nicht in in diesem Verzeichnis zu finden ist, dann liegt dieser meist unter */etc/nginx/sites-enabled*. In diesem Fall kommt eine alternative Verzeichnisstruktur bei nginx zum Einsatz. Hier werden die virtuellen Hosts dann folgendermaßen verarbeitet:

- */etc/nginx/sites-available*: In diesem Verzeichnis sind die verfügbaren virtuellen Hosts enthalten, d.h. vHosts, die beim Start von nginx **nicht** automatisch geladen werden. Die Dateiendung für vHosts ist hier egal und man lässt diese für gewöhnlich einfach weg.
- */etc/nginx/sites-enabled*: Hier sind die virtuellen Hosts gespeichert, die der Webserver beim Starten lädt, die „aktiven vHosts“.
- Um einen virtuellen Host zu aktivieren, muss sich dieser also im Verzeichnis *sites-enabled* befinden. Um nun nicht zwei Dateien verwalten zu müssen (unter *sites-available* und *sites-enabled*), kann einfach eine **symbolische Verknüpfung** für einen virtuellen Host angelegt werden. Ein Beispiel: Wenn ein (deaktivierter) vHost */etc/nginx/sites-available/meinedomain.de* aktiviert werden soll, dann wird mit folgendem Befehl eine entsprechende Verknüpfung angelegt:

```
Shell
1 ln -s /etc/nginx/sites-available/meinedomain.de /etc/nginx/sites-enabled/
```

**Wenn diese alternative Verzeichnisstruktur bei nginx zum Einsatz kommt, dann ist im weiteren Verlauf des Artikels darauf zu achten, die Dateien der virtuellen Hosts an der richtigen Stelle zu platzieren.**

## Installation und Konfiguration benötigter Programme

Genug der Vorrede, nun geht es an die Praxis!

Ubuntu Server 20.04 LTS ist bereits installiert? Gut.

Das System sollte dabei eine statische IP im LAN zugeteilt bekommen haben. Ich nutze im Folgenden die beispielhafte IP **192.168.178.60**.

## Update des Betriebssystems

Zunächst bringen wir das System auf den neusten Stand und starten den Server anschließend einmal neu:

	Shell
1	<code>apt update &amp;&amp; apt upgrade -V &amp;&amp; apt dist-upgrade &amp;&amp; apt autoremove</code>
2	<code>reboot now</code>

## Programm-Installation

Bevor Nextcloud installiert werden kann, müssen zunächst einmal alle Programme installiert werden, die zum Betrieb der Cloud notwendig sind.

### Exkurs: Ubuntu-Paketquellen und Hersteller-Paketquellen

Die meisten benötigten Programme sind bereits in den offiziellen [Ubuntu-Paketquellen](#) enthalten. Diese könnte man nun ohne Umwege direkt über `apt` installieren. Jedoch legt die Distribution Wert auf die Stabilität der Programme, die in den offiziellen Paketquellen enthalten sind. Leider sind diese Programm-Versionen oftmals ziemlich alt und erhalten keine größeren Updates mehr. Daher altert ein solches System recht schnell und man kann keine Features der Programme nutzen, die in neueren Versionen hinzugefügt werden.

Die Hersteller der Programme bieten darüber hinaus auch meistens eigene Paketquellen (Repositories) an. Die hier enthaltenen Versionen sind meistens wesentlich aktueller und werden auch zukünftig mit (Feature-)Updates versorgt. Oftmals unterscheiden die Hersteller dabei nochmals zwischen zwei (oder sogar mehr) [Entwicklungs-Banches](#): Meist gibt es einen **Stable-Branch**, dessen Programm-Versionen möglichst stabil gehalten werden. Daher sind auch hier meistens keine größeren Updates mehr zu erwarten und neue Features werden erst nach z.T. langer Zeit in diesen Branch mit aufgenommen. Daneben gibt es meist noch einen **Mainline-Branch**, der zwar auch als stabil betrachtet werden kann, aber zeitnah mit Updates und neuen Features versorgt wird.

Daher muss man nun für sich selbst entscheiden, welche Programm-Versionen/Repositories zum Einsatz kommen:

- Wenn **Stabilität** das absolut Wichtigste ist, dann sollten die offiziellen **Ubuntu-Paketquellen** verwendet werden.
- Wenn man eine LTS-Version von Ubuntu gewählt hat, bleibt man oftmals sehr lange auf dieser einen Betriebssystem-Version. Hier ist **Stabilität** ein wichtiger Faktor. Wenn darüber hinaus trotzdem einigermaßen **aktuelle Software** eingesetzt werden soll, dann ist der **Stable-Branch** der Hersteller zu empfehlen.
- Wer von **neuen Features** aktueller Programm-Versionen profitieren und die Programme auf einem möglichst aktuellen Stand halten möchte, sollte den **Mainline-Branch** der Software-Hersteller nutzen.

Meiner Erfahrung nach sind die Mainline-Banches der Hersteller allerdings ausreichend stabil, so dass auch der Einsatz in produktiven Umgebungen empfohlen werden kann. **Daher nutze ich in diesem Artikel meist die Mainline-Banches aus den Paketquellen der Software-Hersteller.**

Wer auf andere Versionen/Branches setzt, muss evtl. die im Artikel gezeigten Schritte geringfügig anpassen, da z.B. einige Programm-Features nicht verfügbar sind.

**Hinweis für Raspberry Pi Benutzer:** Oftmals verzichten die Software-Hersteller darauf, in ihren Paketquellen Programm-Versionen für die [ARM-Architektur](#) bereit zu stellen. In diesem Fall würde ein Hinzufügen der Hersteller-Repositories zu einem Fehler bei der Programm-Installation führen. In diesem Fall muss man auf die Paketquellen von Raspbian/Debian zurückgreifen und kann keine Hersteller-Repositories einbinden.

## nginx

Zuerst wird der Webserver installiert. Da ich hier das Mainline-Repository von nginx verwende, muss diese Paketquelle zunächst auf dem System eingebunden werden.

Als erstes fügen wir den Key des Repositories hinzu, da es ansonsten später bei der Installation zu einer Fehlermeldung kommt:

	Shell
1	<code>wget -O - http://nginx.org/keys/nginx_signing.key   apt-key add -</code>

Nun kann die Paketquelle selbst hinzugefügt werden. Dazu legen wir eine entsprechende Datei an:

	Shell
1	<code>nano /etc/apt/sources.list.d/nginx.list</code>

Diese wird mit folgendem Inhalt gefüllt:

	Shell
1	<code># Nginx (Mainline)</code>
2	<code>deb [arch=amd64] http://nginx.org/packages/mainline/ubuntu/ focal nginx</code>
3	<code>deb-src [arch=amd64] http://nginx.org/packages/mainline/ubuntu/ focal nginx</code>

Die Installation des Webservers erfolgt nun über diesen Befehl:

	Shell
1	<code>apt update &amp;&amp; apt install nginx</code>

Direkt nach der Installation kann man die Funktionsweise des Webservers im Browser testen (einfach durch Aufruf der IP des Servers):



## MariaDB

Weiter geht es mit dem Datenbanksystem. Wie schon bei nginx pflegt der Hersteller hier seine eigenen Repositories. Hier gibt es die beiden Branches „Stable“ und „Development“ – hier ist auf jeden Fall der „Stable“ Release vorzuziehen. Mehr Informationen zu den Releases von MariaDB gibt es in der [MariaDB-Knowledgebase](#).

Wie schon bei nginx, muss erst einmal der Repository-Key auf dem System bekannt gemacht werden:

```
Shell
1 sudo apt-key adv --fetch-keys 'https://mariadb.org/mariadb_release_signing_key.asc'
```

Die MariaDB-Paketquellen werden kann wieder in einer einzelnen Datei hinzugefügt:

```
Shell
1 nano /etc/apt/sources.list.d/MariaDB.list
```

```
Shell
1 # MariaDB 10.4 repository list
2 # http://downloads.mariadb.org/mariadb/repositories/
3 deb [arch=amd64] http://mirror.netcologne.de/mariadb/repo/10.4/ubuntu focal main
4 deb-src http://mirror.netcologne.de/mariadb/repo/10.4/ubuntu focal main
```

Die Installation von MariaDB wird dann mit folgendem Befehl durchgeführt:

```
Shell
1 apt update && apt install mariadb-server
```

## PHP

Als nächstes werden die benötigten PHP-Pakete installiert:

```
Shell
1 apt-get install php-fpm php-gd php-mysql php-curl php-xml php-zip php-intl php-mbstring php-bz2
```

Werden später weitere PHP-Pakete benötigt (z.B. für einige Nextcloud Apps), so können diese zu einem späteren Zeitpunkt nachinstalliert werden.

## Let's Encrypt/acme.sh

Zur Generierung der Zertifikate für HTTPS ist noch ein Let's Encrypt Client notwendig. Ich empfehle hier den Client [acme.sh](#), da es sich hierbei um ein reines Bash-Skript handelt. Auf diese Weise ist man nicht abhängig von einer „echten“ Programm-Installation, welche zu einem Problem werden kann, wenn die Version in den Paketquellen zu alt ist.

acme.sh sollte dabei **nicht mit Root-Rechten** (also auch ohne *sudo*) ausgeführt werden. Daher wird extra für acme.sh ein spezieller User angelegt:

```
Shell
1 adduser letsencrypt
```

Für diesen User muss kein Passwort vergeben werden (einfach leer lassen). Ebenso können die weiteren Angaben zum User (Name, E-Mail, etc.) weggelassen werden.

Dieser User wird dann noch der Gruppe *www-data* hinzugefügt:

```
Shell
1 usermod -a -G www-data letsencrypt
```

Der User *letsencrypt* braucht anschließend noch die Rechte, nginx ohne Eingabe eines Root-Passworts neu zu laden. Dies wird mit folgendem Befehl konfiguriert:

```
Shell
1 visudo
```

Ganz am Ende der Datei wird nun folgende Zeile hinzugefügt:

```
Shell
1 letsencrypt ALL=NOPASSWD: /bin/systemctl reload nginx.service
```

Nun kann auch schon acme.sh installiert werden. Dazu wechseln wir zunächst zum User *letsencrypt*:

```
Shell
1 su - letsencrypt
```



Die Installation wird dann mit folgendem Befehl gestartet:

```
Shell
1 curl https://get.acme.sh | sh
```

Nach einem kurzen Augenblick ist die Installation abgeschlossen und wir wechseln wieder zum normalen User:

```
Shell
1 exit
```

## Konfiguration der Programme

Die Installation aller benötigten Programme ist nun abgeschlossen. Nun müssen diese allerdings noch konfiguriert werden, bevor die Nextcloud-Installation beginnen kann.

### Konfiguration nginx

Hier bedarf es ein paar kleineren Anpassungen an der globalen nginx-Konfiguration:

```
Shell
1 nano /etc/nginx/nginx.conf
```

Hier sollten folgende Punkte angepasst/kontrolliert werden:

- *user*: Gibt den Benutzer an, unter dem der Webserver läuft. Dies sollte immer *www-data* sein.
- *worker\_processes*: Die Anzahl der Threads, die nginx dazu verwendet, Client-Requests abzuarbeiten. Die optimale Einstellung ist hier *auto*, da nginx dann pro CPU-Kern einen Thread verwendet.
- *server\_tokens*: Mit der Angabe *off* sorgt man dafür, dass nginx keine Versions-Informationen preisgibt (z.B. auf Fehlerseiten). Wenn diese Variable nicht vorhanden ist, muss man diese im HTTP-Block der Datei einfügen:  
*server\_tokens off;*

Nun deaktivieren wir gleich noch die Default-Seite von nginx, da dies sowieso nur eine Art Demo-Seite ist:

```
Shell
1 mv /etc/nginx/conf.d/default.conf /etc/nginx/conf.d/default.conf_disabled
2 service nginx restart
```

Am Schluss wird nginx noch neu gestartet, damit die Änderungen übernommen werden.

### Konfiguration MariaDB

Bei der Datenbank muss nicht viel konfiguriert werden, allerdings sollte die Installation auf maximale Sicherheit getrimmt werden:

```
Shell
1 mysql_secure_installation
```

An dieser Stelle kann gleich ein Root-Passwort für MariaDB gesetzt werden. Alle weiteren Fragen des Assistenten sollten mit „Ja“ (y) beantwortet werden.

### Konfiguration PHP

Bei PHP muss einiges mehr konfiguriert werden, da sich die Einstellungen über mehrere Dateien verteilen.

PHP wird bei nginx über **FPM (FastCGI Process Manager)** betrieben. Dies ist eine performante Möglichkeit der Kommunikation zwischen PHP und dem Webserver. FPM definiert einen sog. Thread-Pool, der die Anfragen abarbeitet (ähnlich wie schon bei nginx). Die Konfiguration ist dazu in folgender Datei zu finden:

```
Shell
1 nano /etc/php/7.4/fpm/pool.d/www.conf
```

Folgende Anpassungen sollten hier durchgeführt/kontrolliert werden:

- *user/group*: Benutzer unter dem PHP ausgeführt wird. Dies ist wieder der Webserver-User *www-data*:

```
Shell
1 user = www-data
2 group = www-data
```

- *listen*: Die Kommunikation zwischen dem Webserver und PHP findet über einen sog. **Socket** ab. Der Pfad des Sockets wird hier angegeben:

```
Shell
1 listen = /run/php/php7.4-fpm.sock
```

- *Umgebungs-Variablen*: Umgebungs-Variablen werden von PHP in der Standard-Einstellung nicht veräußert. Für den Betrieb von Nextcloud ist dies jedoch erforderlich. Daher suchen wir nach *Pass environment variables like LD\_LIBRARY\_PATH. ALL \$VARIABLES are taken from the current environment* (Shortcut für die Suche in nano:



STRG + W). Alle Einträge, die mit `env` beginnen sind hier auskommentiert. Durch das Entfernen des Semikolons am Zeilenanfang wird die Weitergabe der Umgebungs-Variablen aktiviert:

```
Shell
1 env[HOSTNAME] = $HOSTNAME
2 env[PATH] = /usr/local/bin:/usr/bin:/bin
3 env[TMP] = /tmp
4 env[TMPDIR] = /tmp
5 env[TEMP] = /tmp
```

Neben der Pool-Konfiguration gibt es noch zwei weitere Dateien, die allgemeine Einstellungen zu PHP beinhalten. Die erste ist die `php.ini` für FPM:

```
Shell
1 nano /etc/php/7.4/fpm/php.ini
```

- `cgi.fix_pathinfo`: Sorgt für eine sichere Interpretation von Pfadangaben:

```
Shell
1 cgi.fix_pathinfo = 0
```

- `memory_limit`: Dieser Wert gibt an, wie viel Speicher ein PHP-Skript nutzen darf. Nextcloud benötigt mindestens 512 MB als Memory Limit:

```
Shell
1 memory_limit = 512M
```

- `OPcache`: **PHP OPcache** erhöht bessere Performance durch das Ablegen vorkompilierten Bytecodes in den Arbeitsspeicher. Diese Einträge dazu sollten in der `php.ini` bereits vorhanden sein (allerdings auskommentiert). Eine Suche in der Datei sollte einiges an Tipparbeit sparen. Folgende Werte sind hier anzugeben:

```
Shell
1 opcache.enable = 1
2 opcache.enable_cli = 1
3 opcache.memory_consumption = 128
4 opcache.interned_strings_buffer = 8
5 opcache.max_accelerated_files = 10000
6 opcache.revalidate_freq = 1
7 opcache.save_comments = 1
```

Es gibt noch eine weitere Datei mit dem Namen `php.ini`. Diese enthält Einstellungen für **PHP-CLI** (PHP-Aufrufe direkt über die Kommandozeile):

```
Shell
1 nano /etc/php/7.4/cli/php.ini
```

- `cgi.fix_pathinfo`: Wie oben beschrieben:

```
Shell
1 cgi.fix_pathinfo = 0
```

Nach einem Neustart von PHP werden alle Änderungen übernommen

```
Shell
1 service php7.4-fpm restart
```

## Generierung der Zertifikate für HTTPS

Nachdem alle Vorbereitungen getroffen worden sind, kann es nun an die Generierung der HTTPS-Zertifikate gehen.

### Vorbereiten der Verzeichnisstruktur

Zunächst legen wir ein paar Verzeichnisse an und weisen diesen die entsprechenden Rechte zu:

```
Shell
1 mkdir -p /var/www/letsencrypt/.well-known/acme-challenge
2 chown -R www-data:www-data /var/www/letsencrypt
3 chmod -R 775 /var/www/letsencrypt
4 mkdir -p /etc/letsencrypt/nextcloud.meinedomain.de/rsa
5 mkdir -p /etc/letsencrypt/nextcloud.meinedomain.de/ecc
6 chown -R www-data:www-data /etc/letsencrypt
7 chmod -R 775 /etc/letsencrypt
```

Das erste Verzeichnis (`/var/www/letsencrypt/.well-known/acme-challenge`) wird später vom Webserver bereit gestellt. Darüber kann Let's Encrypt die Domain verifizieren.

Unter `/etc/letsencrypt` werden die eigentlichen Zertifikate gespeichert. Ich lege hier immer ein Unterverzeichnis pro Domain an. Darunter dann getrennte Verzeichnisse für RSA- und ECDSA-Zertifikate. Damit bleibt es auch übersichtlich, wenn später mehrere Domains durch den Webserver bereit gestellt werden sollen.

Wichtig ist hier noch die Zuweisung der Rechte (`chmod -R 775`), damit der User für Let's Encrypt auch die erforderlichen Schreibrechte hat (dieser ist Mitglied der Gruppe `www-data`).

## Anlegen des ersten virtuellen Hosts (HTTP-Gateway)

Als nächsten braucht man nun den virtuellen Host, der für die Zertifikats-Generierung benötigt wird. Ich nenne diesen Host hier einfach „HTTP-Gateway“, da dieser die gesamte Kommunikation über HTTP (Port 80) behandelt. Auch wenn dieser zunächst einmal für nur eine Domain gilt, kann dieser später ebenfalls erweitert werden, wenn weitere Domains gehostet werden sollen.

```
Shell
1 nano /etc/nginx/conf.d/HttpGateway.conf
```

Folgender Inhalt wird hier eingefügt:

```
Shell
1 upstream php-handler {
2     server unix:/run/php/php7.4-fpm.sock;
3 }
4
5 server {
6     listen 80 default_server;
7     listen [::]:80 default_server;
8     server_name nextcloud.meinedomain.de 192.168.178.60;
9
10    root /var/www;
11
12    location ^~ /.well-known/acme-challenge {
13        default_type text/plain;
14        root /var/www/letsencrypt;
15    }
16
17    location / {
18        return 301 https://$host$request_uri;
19    }
20 }
```

Anzumerken sind hier noch folgende Dinge:

- Ganz oben wird ein sog. Upstream für PHP definiert. Durch die Anlage im HTTP-Gateway ist dieser immer verfügbar (und wird später für Nextcloud benötigt).
- Weiter unten sieht man, wie sämtlicher Traffic, der nichts mit der Generierung der HTTPS-Zertifikate zu tun hat, an HTTPS weitergeleitet wird. Dadurch findet immer eine „Zwangs-Umleitung“ auf HTTPS statt, damit sämtliche Verbindungen stets verschlüsselt sind. Nur für die Zertifikats-Generierung über Let's Encrypt muss der Traffic unverschlüsselt über Port 80 (HTTP) erfolgen.

Damit die Änderungen übernommen werden, muss der Webserver noch neu gestartet werden.

```
Shell
1 service nginx restart
```

## Überprüfung des Webserver vor der Zertifikats-Generierung

Bevor man sich nun an der Generierung der Zertifikate macht, sollte die korrekte Konfiguration des Webservers (und der DynDNS-Domain/des Port-Forwardings) erfolgen. Hintergrund ist eine Einschränkung von Let's Encrypt: Wenn die Generierung fehlschlägt (z.B. weil der Webserver einfach nicht über die DynDNS-Domain erreichbar ist), ist man meistens gewillt, den Befehl zum Generieren der Zertifikate mehrfach hintereinander auszuprobieren. Irgendwann stößt man hier an ein „Rate Limit“, nachdem Let's Encrypt die Zertifikats-Generierung für 7 Tage sperrt. In diesem Fall muss man mit der Installation der Cloud für eine Woche warten und dies sorgt meist für Frust.

Dazu legen wir eine einfache Text-Datei an der Stelle ab, an der Let's Encrypt die Verifizierung der Domain vornimmt:

```
Shell
1 echo "Test" >> /var/www/letsencrypt/.well-known/acme-challenge/test.txt
```

Nun erfolgt ein einfacher Aufruf dieser Datei im Browser: <http://nextcloud.meinedomain.de/.well-known/acme-challenge/test.txt>. Wichtig ist hier die Angabe von HTTP (nicht HTTPS). Hier sollte dann der Inhalt der zuvor angelegten Datei angezeigt werden.

Test des Webservers vor der Generierung der Zertifikate

Am Schluss kann die Text-Datei wieder gelöscht werden:

```
Shell
1 rm /var/www/letsencrypt/.well-known/acme-challenge/test.txt
```

**Wichtig:** Die Generierung der Zertifikate sollte erst nach einem erfolgreichen Test durchgeführt werden.

Wenn der Test nicht erfolgreich ist (und die Datei nicht angezeigt wird), sollte ein Blick in das Log von nginx erfolgen (*/var/log/nginx/error.log*). Hier sollte ersichtlich werden, an welcher Stelle der die Datei *test.txt* (erfolglos) sucht. Wenn hier keine passenden Inhalte im Log zu finden sind, liegt das Problem vermutlich an der DynDNS-Domain bzw. dem Port-Forwarding, so dass der Request gar nicht am Webserver ankommt.

## Generierung der Zertifikate

Nun können endlich die Zertifikate erzeugt werden. Dazu wechseln wir zunächst wieder auf den User *letsencrypt*:

```
Shell
1 su - letsencrypt
```

Nun werden zunächst die RSA-Zertifikate erzeugt:

```
Shell
1 acme.sh --issue -d nextcloud.meinedomain.de --keylength 4096 -w /var/www/letsencrypt --key-file
```

Wenn dies erfolgreich war, können noch die ECDSA-Zertifikate generiert werden:

```
Shell
1 acme.sh --issue -d nextcloud.meinedomain.de --keylength ec-384 -w /var/www/letsencrypt --key-fi
```

Im Anschluss findet man die Zertifikat-Dateien im Verzeichnis */etc/letsencrypt/nextcloud.meinedoamin.de/rsa* bzw. */etc/letsencrypt/nextcloud.meinedoamin.de/ecc*:

- **cert.pem:** Das öffentliche Zertifikat in Reinform
- **ca.pem:** Öffentliches Zertifikat aus der sog. Keychain
- **fullchain.pem:** Entspricht cert.pem + chain.pem
- **key.pem:** Privates Zertifikat (diese Datei sollte man daher **niemals** weitergeben)

Im Anschluss wechseln wir wieder auf den normalen User:

```
Shell
1 exit
```

## Erneuerung der Zertifikate

Die Zertifikate von Let's Encrypt haben eine Gültigkeit von 90 Tagen und müssen vor Ablauf dieses Zeitraums erneuert werden.

Um diese Erneuerung muss man sich allerdings nicht mehr kümmern, da im Rahmen der Zertifikats-Generierung für den User *letsencrypt* ein Cronjob eingerichtet wurde, der in regelmäßigen Abständen prüft, ob die Zertifikate bald ablaufen und diese ggf. automatisch erneuert.

## Diffie-Hellman-Parameter

Die Sicherheit des Kommunikation mittels HTTPS kann noch durch den Einsatz sog. **Diffie-Hellman-Parameter** weiter erhöht werden. Mittels dieser Parameter kann einfach gesagt der Schlüsselaustausch beim Verbindungsaufbau weiter abgesichert werden. Da die Generierung eines dafür benötigten Schlüssels recht einfach ist, sollte dieser Schritt auf jeden Fall durchgeführt werden.

**Achtung:** Auf schwächerer Hardware – z.B. bei einem älteren Raspberry Pi – kann die Generierung des Schlüssels sehr lange dauern (bis zu einigen Stunden). Wer nicht so lange warten möchte, der kann auch einen Schlüssel mit „nur“ 2048 Bit errechnen lassen (die Zahl des zweiten Befehls gibt hierbei die Länge des Schlüssels in Bit an).

```
Shell
1 mkdir -p /etc/nginx/dhparams
2 openssl dhparam -out /etc/nginx/dhparams/dhparams.pem 4096
```

## Webserver für Nextcloud vorbereiten

Nachdem wir nun gültige Zertifikate haben, kann der Webserver für die Installation von Nextcloud vorbereitet werden.

## SSL-Konfiguration

Die allgemeine Konfiguration für SSL lagere ich immer in eine spezielle Datei aus. Diese kann dann später einfach in einen vHost eingebunden werden. Durch diese Modularität bleibt man bei der Konfiguration der virtuellen Hosts flexibler. Beispielsweise kann diese allgemeine SSL-Konfiguration später auch in anderen vHosts eingebunden werden, wenn später mehrere Webanwendungen gehostet werden sollen.

Dazu legen wir zunächst ein Verzeichnis für solche Snippets an:

```
Shell
```

```
1 mkdir -p /etc/nginx/snippets
```

Nun wird eine Datei für die allgemeine SSL-Konfiguration angelegt:

```
Shell
1 nano /etc/nginx/snippets/ssl.conf
```

Der Inhalt dieser Datei sieht folgendermaßen aus:

```
Shell
1 #
2 # Configure SSL
3 #
4
5 # Diffie-Hellman parameter for DHE ciphersuites, recommended 4096 bits
6 ssl_dhparam /etc/nginx/dhparams/dhparams.pem;
7
8 # Not using TLSv1 will break:
9 # Android <= 4.4.40 IE <= 10 IE mobile <=10
10 # Removing TLSv1.1 breaks nothing else!
11 ssl_protocols TLSv1.2 TLSv1.3;
12
13 # SSL ciphers: RSA + ECDSA
14 # Two certificate types (ECDSA, RSA) are needed.
15 ssl_ciphers 'TLS-CHACHA20-POLY1305-SHA256:TLS-AES-256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305
16
17 # Use multiple curves.
18 ssl_ecdh_curve secp521r1:secp384r1;
19
20 # Server should determine the ciphers, not the client
21 ssl_prefer_server_ciphers on;
22
23 # SSL session handling
24 ssl_session_timeout 1d;
25 ssl_session_cache shared:SSL:50m;
26 ssl_session_tickets off;
27
28 # SSL stapling has to be done seperately, because it will not work with self signed certs
29 # OCSP Stapling fetch OCSP records from URL in ssl_certificate and cache them
30 ssl_stapling on;
31 ssl_stapling_verify on;
32
33 # DNS resolver
34 resolver 192.168.178.1;
```

Wichtig ist hier die Angabe des richtigen „Resolvers“: Dies ist der lokale DNS-Server, mittels dessen der Webserver Domain-Namen auflösen kann. In den meisten Fällen ist dies einfach die IP des Routers (in diesem Beispiel 192.168.178.1), der einen DNS-Server im lokalen Netzwerk bereit stellt. Alternativ kann hier ein beliebiger DNS-Server angegeben werden. Hier sollte man am besten nicht auf DNS-Server von Google oder Cloudflare setzen, sondern besser auf einen Server, der nicht protokolliert und nicht zensiert. Eine Liste mit empfehlenswerten Servern gibt es z.B. [hier](#).

## Header-Konfiguration

Bei jeder Webanwendung sollten vom Webserver gewisse Header gesetzt werden. Diese lagern wir wieder in eine spezielle Datei aus, damit diese später wieder in virtuelle Hosts eingebunden werden kann.

```
Shell
1 nano /etc/nginx/snippets/headers.conf
```

Diese Datei wird mit folgendem Inhalt gefüllt:

```
Shell
1 #
2 # Add headers to serve security related headers
3 #
4 # HSTS (ngx_http_headers_module is required)
5 add_header Strict-Transport-Security "max-age=63072000; includeSubdomains; preload;" always;
6 add_header X-Content-Type-Options "nosniff" always;
7 add_header X-XSS-Protection "1; mode=block" always;
8 add_header X-Robots-Tag none always;
9 add_header X-Download-Options noopen always;
10 add_header X-Permitted-Cross-Domain-Policies none always;
11 add_header Referrer-Policy no-referrer always;
12 add_header X-Frame-Options "SAMEORIGIN" always;
13
14 # Remove X-Powered-By, which is an information leak
15 fastcgi_hide_header X-Powered-By;
```

## Virtueller Host für Nextcloud

Nun setzen wir alle Puzzleteile zusammen und erstellen den vHost für Nextcloud selbst. Diesen benenne ich hier einfach

nach dem verwendeten Domain-Namen:

```
Shell
1 nano /etc/nginx/conf.d/nextcloud.meinedomain.de.conf
```

Der Inhalt ist diesmal etwas umfangreicher:

```
Shell
1 server {
2     listen 443 ssl http2;
3     listen [::]:443 ssl http2;
4     server_name nextcloud.meinedomain.de;
5
6     # SSL configuration
7     # RSA certificates
8     ssl_certificate /etc/letsencrypt/nextcloud.meinedomain.de/rsa/fullchain.pem;
9     ssl_certificate_key /etc/letsencrypt/nextcloud.meinedomain.de/rsa/key.pem;
10    # ECC certificates
11    ssl_certificate /etc/letsencrypt/nextcloud.meinedomain.de/ecc/fullchain.pem;
12    ssl_certificate_key /etc/letsencrypt/nextcloud.meinedomain.de/ecc/key.pem;
13
14    # This should be ca.pem (certificate with the additional intermediate certificate)
15    # See here: https://certbot.eff.org/docs/using.html
16    # ECC
17    ssl_trusted_certificate /etc/letsencrypt/nextcloud.meinedomain.de/ecc/ca.pem;
18
19    # Include SSL configuration
20    include /etc/nginx/snippets/ssl.conf;
21
22    # Include headers
23    include /etc/nginx/snippets/headers.conf;
24
25    #
26    # Nextcloud configuration
27    #
28
29    # Path to the root of your installation
30    root /var/www/nextcloud/;
31
32    location = /robots.txt {
33        allow all;
34        log_not_found off;
35        access_log off;
36    }
37
38    # The following 2 rules are only needed for the user_webfinger app. Uncomment it if you'r
39    #rewrite ^/.well-known/host-meta /public.php?service=host-meta last;
40    #rewrite ^/.well-known/host-meta.json /public.php?service=host-meta-json last;
41
42    # Well-known URL for CardDAV
43    location = /.well-known/carddav {
44        return 301 $scheme://$host/remote.php/dav;
45    }
46
47    # Well-known URL for CalDAV
48    location = /.well-known/caldav {
49        return 301 $scheme://$host/remote.php/dav;
50    }
51
52    # Well-known URL for Webfinger
53    # Regardless of this rule, you'll get a warning in the admin UI when the social app is no
54    location = /.well-known/webfinger {
55        return 301 $scheme://$host/public.php?service=webfinger;
56    }
57
58    # set max upload size
59    client_max_body_size 10G;
60    fastcgi_buffers 64 4K;
61
62    # Enable gzip but do not remove ETag headers
63    gzip on;
64    gzip_vary on;
65    gzip_comp_level 4;
66    gzip_min_length 256;
67    gzip_proxied expired no-cache no-store private no_last_modified no_etag auth;
68    gzip_types application/atom+xml application/javascript application/json application/ld+js
69
70    # Uncomment if your server is build with the ngx_pagespeed module This module is currently
71    #pagespeed off;
72
73    location / {
74        rewrite ^ /index.php;
75    }
76
```

```

77     location ~ ^/(?:(?:build|tests|config|lib|3rdparty|templates|data)/ {
78         deny all;
79     }
80
81     location ~ ^/(?!(?:\.|autotest|occ|issue|indie|db_|console) {
82         deny all;
83     }
84
85     location ~ ^/(?!(?:index|remote|public|cron|core/ajax/update|status|ocs/v[12]|updater\./
86         fastcgi_split_path_info ^(.+?\.php)(\/.*|)$;
87         set $path_info $fastcgi_path_info;
88         try_files $fastcgi_script_name =404;
89         include fastcgi_params;
90         fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
91         fastcgi_param PATH_INFO $fastcgi_path_info;
92         fastcgi_param HTTPS on;
93         #Avoid sending the security headers twice
94         fastcgi_param modHeadersAvailable true;
95         fastcgi_param front_controller_active true;
96         fastcgi_pass php-handler;
97         fastcgi_intercept_errors on;
98         fastcgi_request_buffering off;
99         fastcgi_read_timeout 600;
100        fastcgi_send_timeout 600;
101        fastcgi_connect_timeout 600;
102
103        fastcgi_param PHP_VALUE "upload_max_filesize = 10G
104        post_max_size = 10G
105        max_execution_time = 3600
106        output_buffering = off";
107    }
108
109    location ~ ^/(?!(?:updater|ocs-provider|ocm-provider)(?:$|\/) {
110        try_files $uri/ =404;
111        index index.php;
112    }
113
114    # Adding the cache control header for js and css files
115    # Make sure it is BELOW the PHP block
116    location ~ \.(?:css|js|woff2?|svg|gif)$ {
117        try_files $uri /index.php$request_uri;
118        add_header Cache-Control "public, max-age=15778463";
119        # Add headers to serve security related headers (It is intended to
120        # have those duplicated to the ones above)
121        # Before enabling Strict-Transport-Security headers please read into
122        # this topic first.
123        add_header Strict-Transport-Security "max-age=63072000; includeSubdomains; preload;"
124
125        # WARNING: Only add the preload option once you read about
126        # the consequences in https://hstspreload.org/. This option
127        # will add the domain to a hardcoded list that is shipped
128        # in all major browsers and getting removed from this list
129        # could take several months
130        add_header X-Content-Type-Options nosniff always;
131        add_header X-XSS-Protection "1; mode=block" always;
132        add_header X-Robots-Tag none always;
133        add_header X-Download-Options noopen always;
134        add_header X-Permitted-Cross-Domain-Policies none always;
135        add_header Referrer-Policy no-referrer always;
136
137        # Optional: Don't log access to assets
138        access_log off;
139    }
140
141    location ~ \.(?:png|html|ttf|ico|jpg|jpeg)$ {
142        try_files $uri /index.php$request_uri;
143        # Optional: Don't log access to other assets
144        access_log off;
145    }
146 }

```

Diese Konfiguration ist an die vorgeschlagene nginx-Konfiguration im [Nextcloud Administration Manual](#) angelehnt. Zu beachten ist hier folgendes:

- Es wird nur ein *server* für HTTPS (nicht HTTP) definiert. Dies reicht aus, da die Kommunikation mit der Nextcloud ausschließlich verschlüsselt (über HTTPS) ablaufen soll. Der zuvor angelegte HTTP-Gateway nimmt alle Verbindungen über HTTP entgegen und leitet diese an den HTTPS-vHost weiter.
- Die Let's Encrypt Zertifikate werden hier in doppelter Ausführung (einmal RSA, einmal ECDSA) angegeben. Beim Verbindungsaufbau wird dann das ECDSA-Zertifikat verwendet, wenn der Client dies unterstützt. Wenn nicht, findet ein „Fallback“ auf das RSA-Zertifikat statt.
- Die bereits angelegten Dateien für die allgemeine SSL-Konfiguration und die Header werden hier einfach per *include* eingebunden.

Bitte auch die Kommentare in dieser Datei beachten und ggf. an die eigenen Bedürfnisse anpassen.

Nach diesen Änderung muss der Webserver natürlich noch neu gestartet werden:

```
Shell
1 service nginx restart
```

## Installation Nextcloud

Endlich sind nun alle Vorbereitungen getroffen worden, damit kann es an die Installation von Nextcloud gehen.

## Download

Zunächst laden wir die aktuellste Version von Nextcloud herunter und entpacken diese in das `www`-Verzeichnis des Webservers:

```
Shell
1 wget https://download.nextcloud.com/server/releases/latest.tar.bz2
2 tar -xjf latest.tar.bz2 -C /var/www
3 rm latest.tar.bz2
```

Im Anschluss müssen noch die Berechtigungen für den Webserver-User gesetzt werden:

```
Shell
1 chown -R www-data:www-data /var/www/nextcloud
```

## Anlegen des Datenverzeichnisses

Das Datenverzeichnis von Nextcloud sollte aus Sicherheitsgründen nicht innerhalb des `www`-Verzeichnisses liegen. Daher legen wir ein eigenes Verzeichnis dafür an und setzen hier ebenfalls die entsprechenden Berechtigungen:

```
Shell
1 mkdir -p /var/nextcloud_data
2 chown -R www-data:www-data /var/nextcloud_data
```

## Datenbank für Nextcloud anlegen

Vor dem Nextcloud-Setup muss noch eine Datenbank für Nextcloud angelegt werden. Dazu melden wir uns einfach an den MySQL-Kommandozeile an:

```
Shell
1 mysql -u root -p
```

Nach der Eingabe des Root-Passworts für MariaDB wird zunächst ein spezieller Datenbank-User für Nextcloud angelegt. Anschließend wird eine Datenbank angelegt und dem Nextcloud-Datenbank-User die entsprechenden Rechte zugewiesen. Die Angabe `localhost` sorgt dafür, dass der Zugriff auf die Datenbank nur auf dem lokalen System erfolgen kann. Ein Remote-Zugriff über das Netzwerk (auf diese Datenbank) ist damit aus Sicherheitsgründen nicht möglich. Die Befehle auf der MySQL-Kommandozeile müssen mit einem Semikolon abgeschlossen werden:

```
MySQL
1 CREATE USER nextcloud_db_user@localhost IDENTIFIED BY 'MeInPasSw0rT';
2 CREATE DATABASE nextcloud_db CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci;
3 GRANT ALL PRIVILEGES on nextcloud_db.* to nextcloud_db_user@localhost;
4 FLUSH privileges;
5 exit;
```

Hier sollte natürlich ein eigenes Passwort für den Nextcloud-DB-User gewählt werden. Beim Anlegen der Datenbank ist auch die Angabe von `CHARACTER SET utf8mb4 COLLATE utf8mb4_general_ci` wichtig, damit die Nextcloud-Datenbank mit 4-Byte-Support erstellt wird.

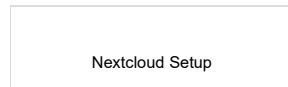
## Nextcloud-Setup

Nun kann das Nextcloud-Setup gestartet werden. Dazu wird einfach die URL der Cloud im Browser aufgerufen (<https://nextcloud.meinedomain.de>). Hier werden nun folgende Informationen angegeben:

- **Benutzer/Passwort (Administrator):** Im Rahmen des Setups wird ein erster Benutzer angelegt, der automatisch Administrator-Rechte für die Cloud besitzt. Der Benutzername ist dabei frei wählbar. Achten sollte man hier auf ein starkes Passwort, da die Cloud ja später öffentlich im Internet erreichbar ist.
- **Datenverzeichnis:** Ebenso wird hier nun der Pfad des Datenverzeichnisses angegeben. Standardmäßig will das Setup diesen Datenpfad innerhalb des `www`-Verzeichnisses erstellen. Allerdings wird aus Gründen der Sicherheit empfohlen, dass das Datenverzeichnis außerhalb des Verzeichnisses `/var/www` liegen sollte. Daher sollte man hier genau darauf achten, dass hier das richtige Verzeichnis angegeben wird, welches wir zuvor für das Datenverzeichnis von Nextcloud erstellt hatten (`/var/nextcloud_data`).
- **Datenbank-Verbindung:** Hier sind die Zugangsdaten für die soeben angelegte Datenbank anzugeben.



- **Empfohlene Apps:** Ganz unten wird noch eine Option angeboten, dass empfohlene Apps gleich im Rahmen des Nextcloud-Setups aktiviert werden („Install recommend apps“). Dabei handelt es sich dann um die Apps Calender, Contacts, Talk, Mail und OnlyOffice. Da für diese Apps aber z.T. weitere Voraussetzungen erfüllt werden müssen, empfehle ich, diese Option im Nextcloud-Setup zu deaktivieren und diese Apps dann bei Bedarf zu einem späteren Zeitpunkt zu installieren.

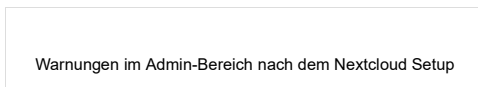


Nach einem Klick auf *Installation abschließen* wird das Setup ausgeführt. Nach einem kurzen Augenblick wird man dann zur eigenen Cloud weitergeleitet.

## Warnungen im Admin-Bereich

Nach der Installation sollte man gleich einen Blick in den Admin-Bereich der Cloud werfen (oben rechts auf das Symbol für den Benutzer klicken und dann *Einstellungen* wählen). Weil der erste angelegte Benutzer Admin-Rechte hat, findet man nun auf der linken Seite unter *Übersicht* eine Zusammenfassung über die Cloud.

Hier werden nach einer frischen Installation mehrere Warnungen angezeigt:



Die erste Warnung bemängelt das Fehlen eines Memory-Caches für PHP. Dies ist dabei nicht als Fehler zu verstehen, sondern eher als Hinweis, dass die Performance der Cloud durch einen Memory-Cache verbessert werden kann. Dazu muss einfach nur eine Zeile in der Nextcloud-Konfiguration geändert werden:

```
Shell
1 nano /var/www/nextcloud/config/config.php
```

Hier wird einfach am Ende folgende Zeile ergänzt:

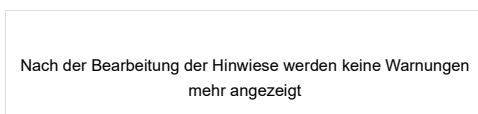
```
PHP
1 'memcache.local' => '\OC\Memcache\APCu',
```

Die beiden anderen Meldungen beziehen sich auf die Datenbank (fehlende Indizes und fehlende Konvertierung in „big int“ für einige Tabellen). Um diese Warnungen zu beseitigen müssen zwei OCC-Befehle (Kommandozeilen-Interface für Nextcloud) ausgeführt werden:

```
Shell
1 cd /var/www/nextcloud
2 sudo -u www-data php occ db:add-missing-indices
3 sudo -u www-data php occ db:convert-filecache-bigint
```

Beim zweiten Befehl wird zwar ein Hinweis angezeigt, dass die Konvertierung sehr lange dauern kann, dies ist bei einer komplett neuen Nextcloud-Installation nicht gegeben.

Wenn die Admin-Übersicht in der Nextcloud nun neu geladen wird, sollten alle Warnungen verschwunden sein.



**Wenn mehr Warnungen im Admin-Bereich angezeigt werden:** Wenn hier immer noch Warnungen zu sehen sein sollten, dann bitte nochmals alle Schritte dieses Tutorials überprüfen, ob nicht ein kleines Detail ausgelassen wurde (besonders die PHP-Konfiguration). Im [Nextcloud Administration Manual](#) findet man auch eine Übersicht aller Warnungen, die hier im Admin-Bereich angezeigt werden könnten und entsprechende Lösungsansätze.

## Optimieren der Nextcloud-Konfiguration

Die Konfiguration von Nextcloud sollte über die Datei *config.php* noch weiter optimiert werden:

```
Shell
1 nano /var/www/nextcloud/config/config.php
```

Hier sollten nun folgende Anweisungen eingefügt/kontrolliert werden:

- **HTTPS als Standard definierten:** Um alle Verbindungen zur Nextcloud strikt per HTTPS zu behandeln, wird die Verbindung nur über *https://* zu forcieren, sollte hier folgende Variable gesetzt werden:

```
PHP
1 'overwriteprotocol' => 'https',
```

- **Zeitzone für Log-Einträge:** Die richtige Zeitzone (z.B. für die Zeitangaben der Log-Einträge) wird durch diese

Variable konfiguriert:

```

PHP
1 'logtimezone' => 'Europe/Berlin',

```

## Installation/Einrichtung Redis

Eine weitere Optimierung, die nun gleich durchgeführt werden sollte, ist die Einrichtung von Redis für Nextcloud. Die Cloud nutzt sog. **Transactional File Locking**, um Sperren bei parallelem Zugriff auf Dateien zu realisieren. Dateien, die gerade im Zugriff sind, werden dadurch „gelockt“, damit es bei parallelen Zugriffen nicht zu Datenverlusten kommen kann. Ansonsten würde hier immer die letzte Änderung „gewinnen“ und alle parallel durchgeführten Änderungen überschreiben.

Diese Sperren werden standardmäßig in der Datenbank von Nextcloud verwaltet. Um hier Last von der Datenbank zu nehmen und die generelle Performance zu erhöhen, kann hier auch **Redis** genutzt werden. Diese **In-Memory-Datenbank** ist genau für solche Aufgaben optimiert, wodurch sich die Performance-Verbesserung ergibt.

Hier sollte man allerdings nicht zu viel erwarten: In kleineren Cloud-Instanzen mit nur wenigen Benutzern wird man hier kaum messbare Änderungen erzielen. Trotzdem empfehle ich den generellen Einsatz von Redis, da die Einrichtung auch nicht besonders aufwändig ist.

### Installation und Konfiguration Redis

Zunächst wird Redis und das dazugehörige PHP-Modul installiert:

```

Shell
1 apt update && apt install redis-server php-redis

```

Nun erfolgt die Konfiguration der In-Memory-Datenbank:

```

Shell
1 nano /etc/redis/redis.conf

```

Analog zur Konfiguration von PHP sollte auch Redis über einen Socket angesprochen werden. Dazu müssen in dieser Konfigurations-Datei folgende Einstellungen gesetzt werden.

```

Shell
1 port 0
2 unixsocket /var/run/redis/redis-server.sock
3 unixsocketperm 770

```

Hiermit wird folgendes konfiguriert:

- **port**: Redis lauscht standardmäßig auf dem Port 6379. Wie gesagt wollen wir aber einen Socket nutzen, daher wird das Lauschen auf einem Port deaktiviert.
- **unixsocket**: Hiermit wird der eigentliche Socket definiert.
- **unixsocketperm**: Berechtigungen für diesen Socket.

Nun muss der Redis-Benutzer noch der Gruppe **www-data** hinzugefügt werden:

```

Shell
1 usermod -a -G redis www-data

```

Damit die Änderungen übernommen werden, muss Redis noch neu gestartet werden:

```

Shell
1 service redis-server restart

```

### Einbinden von Redis in Nextcloud

Damit die eigene Cloud nun auch Redis nutzt, muss dieses noch in der Nextcloud-Konfiguration angepasst werden:

```

Shell
1 nano /var/www/nextcloud/config/config.php

```

Am Ende der Datei wird nun folgende Konfiguration hinzugefügt:

```

PHP
1 'filelocking.enabled' => 'true',
2 'memcache.locking' => '\OC\Memcache\Redis',
3 'redis' => array(
4     'host' => '/var/run/redis/redis-server.sock',
5     'port' => 0,
6     'timeout' => 0.0,
7     ),

```

Anschließend nutzt Nextcloud nun das Transactional File Locking.

### Cronjob für Nextcloud einrichten

Nextcloud benötigt für den einwandfreien Betrieb die regelmäßige Ausführung von Hintergrundaufgaben (z.B. Bereinigung verwaister Datenbank-Einträge, etc.). Diese Aufgaben werden direkt nach der Installation per **AJAX** ausgeführt: Immer, wenn eine Seite der Cloud aufgerufen wird, wird beim Laden einer Seite geprüft, ob Hintergrundaufgaben anstehen und diese ggf. ausgeführt. Diese Methode hat jedoch zwei Nachteile: Es wird erstens immer ein angemeldeter Benutzer benötigt, denn nur durch diesen können die Hintergrundaufgaben beim Laden der Seite angestoßen werden. Wenn die Cloud nun lange ohne Benutzeranmeldung läuft, dann werden u.U. über einen längeren Zeitraum keine Hintergrund-Jobs ausgeführt. Zum anderen ist die Lösung über AJAX wenig performant.

Daher sollte die Hintergrundaufgaben besser per **Cron** ausgeführt werden. Hier prüft ein Hintergrund-Dienst in regelmäßigen Abständen, ob Hintergrundaufgaben ausgeführt werden sollten. Besonders auf schwächerer Hardware ist diese Methode deutlich effizienter.

Dazu wird einfach ein Cronjob unter dem Webserver-Benutzer angelegt:

```
Shell
1 crontab -u www-data -e
```

Am Ende der Datei wird anschließend folgende Zeile eingefügt:

```
Shell
1 */5 * * * * php -f /var/www/nextcloud/cron.php > /dev/null 2>&1
```

Nach dem Speichern der Datei wird dieser Cronjob alle 5 Minuten automatisch ausgeführt. Theoretisch könnte hier auch eine andere Zeitspanne genutzt werden, allerdings ist die allgemeine Empfehlung 5 Minuten (siehe [Nextcloud Admin-Manual](#)).

Nach fünf Minuten kann nun in der Admin-Oberfläche (unter den *Grundeinstellungen*) der Cloud geprüft werden, ob der Cronjob ausgeführt wurde: Die entsprechende Option sollte nun auf *Cron* stehen. Ebenso sollte nach jeder Ausführung des Cronjobs unter *Letzte Aufgabe ausgeführt* folgendes stehen: *Gerade eben*.

Nextcloud: Ausführung von Hintergrund-Aufgaben per Cron

## Weitere Konfiguration Nextcloud

Damit ist die grundsätzliche Konfiguration von Nextcloud soweit abgeschlossen. Nun ist es an der Zeit, die Cloud an die eigenen Bedürfnisse anzupassen.

Nextcloud kann nun durch eine große Auswahl an Apps erweitert werden: siehe [Nextcloud App Store](#).

Generell ist bei der erweiterten Konfiguration von Nextcloud ein Blick ins [Nextcloud Administration Manual](#) empfehlenswert.

## Optimierung der Server-Umgebung für Nextcloud

Die eigene Nextcloud ist nun soweit fertig eingerichtet und kann bereits genutzt werden. Dennoch kann die Server-Umgebung noch weiter optimiert werden.

### ufw

ufw (**un**complicated **f**irewall) ist eine Software-Firewall, die auf einer Ubuntu-Installation standardmäßig bereits installiert ist. Im Heim-Bereich übernimmt normalerweise der Router schon die Aufgaben einer Firewall, daher ist die Einrichtung von hier ufw optional. Auf einem Root-Server ist aber in der Regel keine Firewall vor den Server geschaltet, daher sollte die Einrichtung von ufw hier in jedem Fall erfolgen.

Falls ufw noch nicht installiert sein sollte, kann man dies mit folgendem Befehl nachholen:

```
Shell
1 apt update && apt install ufw
```

Die Firewall soll sämtlichen eingehenden Traffic blockieren, mit einigen Ausnahmen:

- SSH (Port 22): Hier soll der Zugriff nur aus dem Heimnetzwerk erlaubt sein.
- HTTP/HTTPS (Ports 80/443): Da der Zugriff aus dem Web erforderlich ist, muss hier auch eine Ausnahme hinzugefügt werden.

Mit folgenden Befehlen werden diese Regeln umgesetzt:

```
Shell
1 ufw default deny
2 ufw allow proto tcp from 192.168.178.0/24 to any port 22
3 ufw allow 80
4 ufw allow 443
5 ufw enable
```

Nach dem Aktivieren der Firewall können die Regeln nochmal überprüft werden:

	Shell
1	<code>ufw status</code>

**Wichtig:** Wenn auf dem Server weitere Anwendungen installiert sind, die andere Ports nutzen oder z.B. ein abweichender Port für SSH genutzt wird, ist dies natürlich bei der Erstellung der Firewall-Regeln zu beachten. Man sollte also immer im Hinterkopf behalten, dass eine Firewall auf dem System aktiv ist und das diese ggf. angepasst werden muss.

## Fail2ban

Nextcloud wird standardmäßig mit einem **Brute-Force-Schutz** ausgeliefert. Dieser sorgt bei fehlgeschlagenen Logins für eine Verzögerung von Login-Versuchen aus dem betroffenen **Subnetz**. Bei **Brute-Force-Attacken** führt dies zu maximal zu einer Verzögerung um 30 Sekunden zwischen Login-Versuchen. Allerdings werden keine IPs gebannt. Daher ist für eine Absicherung der Cloud der Einsatz von **Fail2ban** sinnvoll. Gegenüber dem Brute-Force-Schutz der Nextcloud bietet Fail2ban folgende Vorteile:

- Mit Fail2ban können **IPs automatisch gebannt** werden. Nach einem solchen Ban kann die betroffene IP nicht mehr auf die Nextcloud-Instanz zugreifen, es wird lediglich eine Fehlermeldung des Browsers angezeigt.
- Fail2ban arbeitet **IP-basiert**: Es wird nur die betroffene IP blockiert, von der aus zu viele fehlgeschlagene Login-Versuche unternommen wurden. Andere Nutzer mit abweichenden IP-Adressen aus dem gleichen Netzwerk (Subnet) werden dabei nicht gebannt.
- Das Einsatzgebiet von Fail2ban ist sehr weit gefächert: Man kann hier nicht nur die Nextcloud-Installation, sondern auch weitere **Anwendungen auf dem Server absichern** (z.B. Webserver und SSH).

## Konfigurations-Dateien von Fail2ban

Vor der Installation noch ein paar Worte zu den Konfigurations-Dateien von Fail2ban: Das Programm unterscheidet zwischen zwei Arten von Konfigurations-Dateien: `*.conf` und `*.local`. Die `conf`-Dateien kommen bei der Installation mit und sind die „Standard-Einstellungen“. Diese Dateien sollten aber nie direkt bearbeitet werden, da diese bei einem Update von Fail2ban überschrieben werden können. Daher nutzt man für individuelle Änderungen Dateien mit gleichem Namen, aber mit der Dateiendung `.local`. Diese `local`-Dateien überschreiben dann die Standard-Einstellungen der `conf`-Dateien. Auf diese Weise geht eine individuelle Konfiguration bei einem Update von Fail2ban nicht verloren.

## Deaktivieren des Nextcloud Brute-Force-Schnutzes

Wenn Fail2ban zum Einsatz kommt, dann der integrierte Brute-Force-Schutz von Nextcloud deaktiviert werden. Dazu bearbeiten wir die Konfigurations-Datei von Nextcloud:

	Shell
1	<code>nano /var/www/nextcloud/config/config.php</code>

Der Schutz wird durch die Eingabe folgender Zeile deaktiviert:

	Shell
1	<code>'auth.bruteforce.protection.enabled' =&gt; false,</code>

Hier sollte auch gleich noch kontrolliert werden, ob Nextcloud die richtige Zeitzone für die Einträge im Log nutzt, da es für Fail2ban essentiell ist, dass hier korrekte Zeiten geloggt werden:

	Shell
1	<code>'logtimezone' =&gt; 'Europe/Berlin',</code>

## Installation und Konfiguration Fail2ban

Nun kann Fail2ban installiert werden:

	Shell
1	<code>apt update &amp;&amp; apt install fail2ban</code>

Anschließend muss ein sog. „Filter“ für Nextcloud definiert werden:

	Shell
1	<code>nano /etc/fail2ban/filter.d/nextcloud.local</code>

In dieser Filter-Datei wird nun ein regulärer Ausdruck hinterlegt. Dieser definiert das Format, welches einen fehlgeschlagenen Login-Versuch im Nextcloud-Log identifiziert:

	Shell
1	<code>[Definition]</code>
2	<code>failregex=^{"reqId":".*","remoteAddr":".*","app":"core","message":"Login failed: '.*' \\\(Remote</code>
3	<code>^{"reqId":".*","level":2,"time":".*","remoteAddr":".*","user":".*","app":".*","meth</code>
4	<code>^{"reqId":".*","level":2,"time":".*","remoteAddr":".*","user":".*","app":".*","meth</code>

Damit dieser Filter zum Einsatz kommt, wird dieser Fail2ban noch bekannt gemacht:

	Shell

```
1 nano /etc/fail2ban/jail.local
```

Hier fügen wir folgenden Inhalt ein:

	Shell
1	[DEFAULT]
2	maxretry=3
3	bantime=1800
4	
5	[nextcloud]
6	enabled=true
7	port=80,443
8	protocol=tcp
9	filter=nextcloud
10	logpath=/var/nextcloud_data/nextcloud.log
11	
12	[nginx-http-auth]
13	enabled = true

Der Teil unter `[DEFAULT]` gilt dabei für alle Regel-Sets (Jails). Hier werden die allgemeinen Einstellungen für Bans hinterlegt:

- `maxretry`: Anzahl der Fehlversuche, bevor Fail2ban eine IP sperrt.
- `bantime`: Zeitraum (in Sekunden), für den eine IP gesperrt werden soll. Durch die Eingabe von `-1` werden betroffene IPs dauerhaft gebannt.

Unter `[nextcloud]` wird dann neues Jail für Nextcloud definiert:

- `enabled`: Aktivierung dieser Regel. Falls das Jail später (temporär) deaktiviert werden soll, kann dies einfach durch diese Option erfolgen.
- `port`: Port, hier 80 (HTTP) und 443 (HTTPS).
- `protocol`: Das verwendete Protokoll, in unserem Fall TCP.
- `filter`: Name des Filters aus gleichnamiger Datei unter `/etc/fail2ban/filter.d`. Der Filter für Nextcloud wurde ja schon im vorherigen Schritt angelegt.
- `logpath`: Log-Datei, die Fail2ban für dieses Jail beobachten soll. Auf diese Datei wird der oben definierte Filter angewendet.
- Hier könnten auch noch Werte für `maxretry` und `bantime` hinterlegt werden, die dann sie Standard-Werte überschreiben und nur für das Nextcloud-Jail gelten.

Wir aktivieren hier unter `[nginx-http-auth]` auch gleich noch die Überwachung der Logs des Webservers. Dieses Jail ist bereits in den Standard-Jails (`/etc/fail2ban/jail.conf`) definiert und wird an dieser Stelle lediglich „scharf geschaltet“. Dies sorgt für eine Überwachung bei anderen Webanwendungen, bei den eine [HTTP-Basic-Authentifizierung](#) zum Einsatz kommt. Die Ban-Optionen (`maxretry` und `bantime`) werden hier nicht angegeben, daher gelten die Standard-Werte, die in der Sektion `[DEFAULT]` angegeben wurden.

Nun fehlt nur noch ein Neustart des Dienstes:

	Shell
1	service fail2ban restart

### Optional: E-Mail-Versand durch Fail2ban

Fail2ban arbeitet nun „lautlos“ im Hintergrund. Nur ein Administrator kann den aktuellen Status von Fail2ban über die Kommandozeile prüfen.

Sinnvoll ist nun noch eine Erweiterung, damit Fail2ban im Falle eines Bans eine E-Mail an den Server-Administrator verschickt. Die Voraussetzung ist dabei, dass das Linux-System bereits Mails versenden kann. Dies kann ohne großen Aufwand über [msmtp](#) realisiert werden. Die Installation und Einrichtung des Programms wurde bereits im Artikel [Linux: Einfach E-Mails senden mit msmtp](#) ausführlich erklärt.

Bei Fail2ban ist bereits der Versand von E-Mails vorgesehen, daher reichen hier kleinere Anpassungen, wenn msmtp bereits installiert wurde:

	Shell
1	nano /etc/fail2ban/jail.local

Am Anfang werden nun in der Default-Sektion folgende Zeilen eingefügt (hier markiert):

	Shell
1	[DEFAULT]
2	maxretry=3
3	bantime=1800
4	
5	# Destination email address used solely for the interpolations in
6	# jail.{conf,local,d/*} configuration files.
7	destemail = meineemail@provider.de
8	

```

9 # Sender email address used solely for some actions
10 sender = absenderadresse@provider.de
11
12 # E-mail action. Since 0.8.1 Fail2Ban uses sendmail MTA for the
13 # mailing. Change mta configuration parameter to mail if you want to
14 # revert to conventional 'mail'.
15 mta = mail
16 action = %(action_mwl)s

```

Folgende Optionen werden damit gesetzt:

- *destemail* ist die Mail-Adresse, an die Benachrichtigungen verschickt werden.
- *sender* ist die Adresse, von der die E-Mail gesendet werden (Absender).
- Wichtig ist insbesondere die Zeile *action = %(action\_mwl)s*: Hierdurch werden E-Mails standardmäßig versendet.

Nun würde man bei jeder Aktion von Fail2ban eine E-Mail erhalten, also auch wenn der Dienst z.B. neu gestartet wurde. Wenn Fail2ban ausschließlich beim Bannen einer IP eine Mail versenden soll, müssen noch ein paar Änderungen vorgenommen werden. Die Konfiguration dazu wird wieder über local-Dateien im Verzeichnis */etc/fail2ban/action.d* realisiert. Folgende Dateien müssen dafür angelegt werden:

- *mail-buffered.local*
- *mail.local*
- *mail-whois-lines.local*
- *mail-whois.local*
- *sendmail-buffered.local*
- *sendmail-common.local*

Der Inhalt sieht dabei immer gleich aus:

```

Shell
1 [Definition]
2
3 # Option:  actionstart
4 # Notes.:  command executed once at the start of Fail2Ban.
5 # Values:  CMD
6 #
7 actionstart =
8
9 # Option:  actionstop
10 # Notes.: command executed once at the end of Fail2Ban
11 # Values:  CMD
12 #
13 actionstop =

```

Am Ende noch Fail2ban neu starten:

```

Shell
1 service fail2ban restart

```

## Test der Funktionalität

Am Schluss sollte noch die Funktionalität von Fail2ban getestet werden. Dies ist insbesondere sinnvoll, damit man auch mal das Entbannen einer IP durchgeführt hat – es könnte ja mal sein, dass man sich selbst „aussperrt“.

Dazu meldet man sich einfach drei mal mit einem nicht vorhandenen Benutzer an der eigenen Cloud an. Beim vierten Versuch sollte wie Webseite dann gar nicht mehr aufrufbar sein. Zeitgleich sollte man eine E-Mail erhalten, die über den Ban informiert. Auf der Kommandozeile kann man sich diesen Ban nun anzeigen lassen:

```

Shell
1 fail2ban-client status nextcloud

```

Hier sollte nun die gebannte IP aufgeführt werden. Um den Ban aufzuheben, geben wir nun folgenden Befehl ein:

```

Shell
1 fail2ban-client set nextcloud unbanip 45.135.54.12

```

Anschließend sollte man wieder wie gewohnt auf Nextcloud zugreifen können.

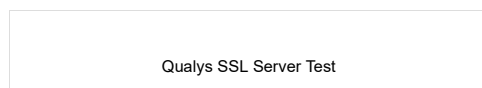
## Überprüfung der Sicherheit

Die Sicherheit der eigenen Cloud war ja von Anfang an ein wichtiges Ziel dieses Artikels. Nach der Einrichtung von Nextcloud gibt es einige Tools, mit denen dies nochmal kontrolliert werden kann.

### Qualys SSL Labs

Mit dem [SSL Server Test von Qualys SSL Labs](#) kann die HTTPS-Verschlüsselung auf einfache Weise überprüft werden.

Getestet wird hier sowohl die Einbindung der Let's Encrypt Zertifikate, als auch die SSL-Einstellungen des Webservers. Wenn alles richtig konfiguriert wurde, sollte hier eine Bewertung von A+ angezeigt werden:



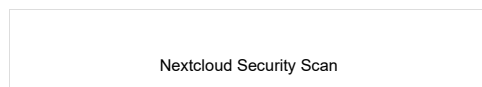
## Mozilla Observatory

Ebenfalls ein guter Test ist [Mozilla Observatory](#). Hier sollte ebenso ein Rating von A+ erzielt werden können.



## Nextcloud Security Scan

Ein weiterer Test ist der [Nextcloud Security Scan](#). Dieses Tool prüft die öffentlich verfügbaren Informationen der Nextcloud-Instanz und kann eventuelle Schwachstellen aufzeigen. Auch hier sollte ein Rating von A+ angezeigt werden:



## FAQ

An dieser Stelle sollen die häufigsten Fragen zu diesem Artikel beantwortet werden.

### Ich habe bereits eine Nextcloud-Installation unter Ubuntu Server 18.04 LTS am laufen. Soll ich diese nun möglichst schnell auf Ubuntu Server 20.04 LTS updaten?

Ubuntu 18.04 LTS wird noch bis April 2023 unterstützt. Daher ist beim Update auf Ubuntu Server 20.04 LTS keine Eile geboten. Im Gegenteil: Wenn das System ansonsten keine Probleme bereitet, würde ich mit dem Update noch etwas abwarten – zumindest bis zur Version 20.04.1.

Bei komplett neuen Installationen macht es dagegen durchaus Sinn, direkt auf Ubuntu Server 20.04 LTS aufzusetzen.

### Der Artikel zeigt, wie man Nextcloud direkt im Root-Verzeichnis des Webservers installiert. Ich möchte die Nextcloud allerdings in einem Unterverzeichnis des Webservers laufen lassen. Was muss ich dazu beachten?

Durch die vereinfachte Webserver-Konfiguration empfehle ich mittlerweile die Installation der eigenen Cloud direkt im Web-Root (z.B. <https://nextcloud.meinedomain.de>). Wer Nextcloud dennoch in einem Unterverzeichnis des Webservers laufen lassen will (z.B. <https://meinedomain.de/nextcloud>), der kann sich dazu am vorherigen Artikel [Nextcloud auf Ubuntu Server 18.04 LTS mit nginx, MariaDB, PHP, Let's Encrypt, Redis und Fail2ban](#) orientieren.

### Ich möchte eine weitere Webanwendung auf dem gleichen Server installieren. Wie muss ich dazu vorgehen?

Angenommen, man möchte neben Nextcloud ([nextcloud.meinedomain.de](https://nextcloud.meinedomain.de)) noch eine WordPress-Instanz unter ([blog.meinedomain.de](https://blog.meinedomain.de)) aufsetzen. In diesem Fall muss zunächst einmal sicher gestellt sein, dass die zweite Domain auch auf die eigene IP-Adresse verweist. Hier reichen wieder A- bzw. AAAA-Records bei der zweiten Domain aus, wenn es sich um eine fest IP handelt. Falls nicht, muss ein sog. [CNAME-Eintrag](#) für die Domain [blog.meinedomain.de](https://blog.meinedomain.de) erstellt werden, der auf die DynDNS-Adresse verweist (in diesem Fall [nextcloud.meinedomain.de](https://nextcloud.meinedomain.de)). Dies ist notwendig, weil die meisten Router nicht die gleichzeitige Verwendung mehrerer DynDNS-Konfigurationen erlauben.

Anschließend muss der HTTP-Gateway erweitert werden, so dass dieser auch auf die neue Domain reagiert:

```
Shell
1 nano /etc/nginx/conf.d/HttpGateway.conf
```

Hier fügen wir die neue Domain unter `server_name` ein:

```
Shell
1 server {
2     listen 80 default_server;
3     listen [::]:80 default_server;
4     server_name nextcloud.meinedomain.de blog.meinedomain.de 192.168.178.60;
5
6     #...
7 }
```

Nach einem Neustart können nun für die zweite Domain Zertifikate bei Let's Encrypt beantragt werden. Dies geschieht analog wie im Artikel für die Domain [nextcloud.meinedomain.de](https://nextcloud.meinedomain.de) gezeigt.

Nun kann ein weiterer virtueller Host für die zweite Domain erstellt werden:

```
Shell
1 nano /etc/nginx/conf.d/blog.meinedomain.de.conf
```



Hier werden dann die soeben ausgestellten Zertifikate eingebunden. Dank der Modularisierung mit den „Snippets“ für die allgemeine SSL-Konfiguration und den Headern, können diese Bausteine wiederverwendet werden:

```

1 server {
2     listen 443 ssl http2;
3     listen [::]:443 ssl http2;
4     server_name blog.meinedomain.de;
5
6     # SSL configuration
7     # RSA certificates
8     ssl_certificate /etc/letsencrypt/blog.meinedomain.de/rsa/fullchain.pem;
9     ssl_certificate_key /etc/letsencrypt/blog.meinedomain.de/rsa/key.pem;
10    # ECC certificates
11    ssl_certificate /etc/letsencrypt/blog.meinedomain.de/ecc/fullchain.pem;
12    ssl_certificate_key /etc/letsencrypt/blog.meinedomain.de/ecc/key.pem;
13
14    # This should be ca.pem (certificate with the additional intermediate certificate)
15    # See here: https://certbot.eff.org/docs/using.html
16    # ECC
17    ssl_trusted_certificate /etc/letsencrypt/blog.meinedomain.de/ecc/ca.pem;
18
19    # Include SSL configuration
20    include /etc/nginx/snippets/ssl.conf;
21
22    # Include headers
23    include /etc/nginx/snippets/headers.conf;
24
25    #...
26 }

```

Die weitere Konfiguration des vHosts für *blog.meinedomain.de* ist abhängig von der jeweiligen Webanwendung, die installiert werden soll. Für WordPress findet man z.B. [hier](#) eine Konfiguration, die man als Grundlage verwenden kann.

Auf diese Weise hat man die Konfigurationen zu den einzelnen Webseiten durch den Einsatz von zwei separaten vHosts gut getrennt.

## Troubleshooting

Das hier gezeigte Setup ist durchaus umfangreich. Daher kann es passieren, dass nicht alles auf Anhieb klappt. Für diese Fälle hier noch ein paar Tipps und Tricks für die Fehlersuche:

- **Wurden alle Schritte korrekt ausgeführt?**

Die Installation der eigenen Cloud erfordert viele Schritte, die aufeinander aufbauen. Hier kann es leicht vorkommen, dass man einen kleinen Schritt übersieht, oder es während der Ausführung des Schrittes zu einem Fehler gekommen ist, den man aber nicht bemerkt hat. Auch wenn der Schritt noch so unbedeutend aussehen mag: Die kleinste Abweichung kann dazu führen, dass „das große Ganze“ nicht mehr funktioniert. Als ersten Punkt für die Fehlersuche empfehle ich daher immer nochmal die genaue Kontrolle der einzelnen Schritte.

- **Kontrolle der Logs**

Wenn irgend etwas nicht auf Anhieb klappt mag, hilft oftmals ein Blick in die entsprechenden Log-Dateien:

- nginx (*/var/log/nginx/error.log*): Der Webserver führt hier alle Warnungen und Fehler auf. Dies sollte immer die erste Anlaufstelle sein, wenn sich Nextcloud gar nicht aufrufen lässt bzw. Links nicht richtig funktionieren.
- Nextcloud (*/var/nextcloud\_data/nextcloud.log*): Hier sind Fehler/Warnungen von Nextcloud selbst enthalten. Die Logs können auch direkt über die Admin-Oberfläche von Nextcloud untersucht werden. Die Einträge im Nextcloud-Log sind v.a. dann interessant, wenn die eigene Cloud prinzipiell aufgerufen werden kann (und der Webserver vermutlich korrekt eingerichtet wurde), es aber bei der Benutzung von Nextcloud zu Problemen kommt.

- **Developer-Console im Browser**

Wenn die Kontrolle der Logs zu keinen Erkenntnissen führen, dann es sinnvoll sein, einen Request mit geöffneter Developer-Console im Browser (meist zu öffnen mit F12) auszuführen. Hier wird detailliert aufgelistet, was bei einem Aufruf des Clients passiert. Die Bedienung der Console läuft in jedem Browser etwas anders ab, daher hilft ein Blick in die entsprechende Dokumentation:

- Firefox: [Firefox Developer Tools – Toolbox](#)
- Chrome: [Using the console](#)
- Edge: [Microsoft Edge Dev Tools – Console](#)

Falls ihr also Probleme bei der Einrichtung von Nextcloud haben solltet, **bitte erst einmal diese Punkte überprüfen**. Oftmals findet man dann relativ schnell die Ursache. Wenn alles nicht hilft, dann könnt ihr hier natürlich einen **Kommentar** hinterlassen, vielleicht kann euch schnell weitergeholfen werden.

Falls sich bestimmte Probleme häufen sollte, werde ich den Artikel ggf. anpassen und/oder erweitern.

## Nextcloud: Ein sicherer Ort für all deine Daten

Dies ist der bekannte Slogan von Nextcloud und dieser trifft es auch ziemlich genau: Auch wenn der Aufwand zum Aufsetzen der eigenen Cloud nicht zu unterschätzen ist, ist man nach getaner Arbeit jedoch unabhängig von einem (externen) Cloud-Anbieter. Dadurch bleibt man jederzeit Herr über die eigenen Daten, die sicher und einfach in der eigenen Cloud gespeichert werden können.

Jedoch sollte man sich auch im Klaren darüber sein, dass man nun auch für den Betrieb der Cloud verantwortlich ist. Dies erfordert nach dem Aufsetzen von Nextcloud auch einen gewissen administrativen Aufwand. So ist der Cloud-Admin dafür verantwortlich, dass die Cloud und das Betriebssystem auf dem aktuellen Stand gehalten werden. Dennoch sollte sich der Aufwand hierfür in Grenzen halten.

Ich hoffe, dass dieser Artikel hilfreich war und ich einigen unter euch etwas Zeit (und Nerven) bei der Einrichtung der eigenen Cloud ersparen konnte. Für konstruktive Kritik und Verbesserungsvorschläge bin ich immer offen, daher hinterlasst mir doch gern einen Kommentar.

In diesem Sinne heißt es wie immer: **Viel Spaß mit eurer eigenen Nextcloud!**

## Weiterführende Artikel

- [Nextcloud auf Ubuntu Server 18.04 LTS mit nginx, MariaDB, PHP, Let's Encrypt, Redis und Fail2ban](#) (alter Artikel zur Installation von Nextcloud auf Ubuntu Server)
- [Nextcloud – Tipps & Tricks für Admins \(und Benutzer\)](#)
- [Nextcloud: Online-Office mit Collabora](#)
- [Nextcloud: Online-Office mit ONLYOFFICE \(mit eigener Subdomain\)](#)
- [Nextcloud: Updates richtig durchführen](#)
- [Nextcloud: Backups erstellen und wiederherstellen – manuell oder per Skript](#)
- [Nextcloud auf anderen Rechner umziehen](#)
- [Let's Encrypt Zertifikate mit acme.sh und nginx](#)
- [RSA und ECDSA-Zertifikate mit nginx \(Hybrid-Lösung\)](#)

## Links

- [Offizielle Nextcloud Homepage \(englisch\)](#)
- [Offizielle nginx Homepage \(englisch\)](#)
- [Offizielle MariaDB Homepage \(englisch\)](#)
- [Offizielle PHP Homepage \(englisch\)](#)
- [Offizielle Let's Encrypt Homepage \(englisch\)](#)
- [acme.sh \(GitHub Repository\)](#)
- [Offizielle Redis Homepage \(englisch\)](#)
- [Offizielle Fail2ban Homepage \(englisch\)](#)
- [Nextcloud Administration Manual \(englisch\)](#)
- [Nextcloud Security Scan](#)
- [SSL Server Test \(Qualys SSL Labs\)](#)
- [Mozilla Observatory](#)

## Ähnliche Artikel:

<a href="#">Nextcloud Logo</a>	<a href="#">Nextcloud Logo</a>	<a href="#">ownCloud Logo</a>	<a href="#">ownCloud Logo</a>
<a href="#">Nextcloud auf Ubuntu Server 18.04 LTS mit</a>	<a href="#">Nextcloud auf Ubuntu Server mit nginx,</a>	<a href="#">ownCloud 9 auf Ubuntu Server 16.04</a>	<a href="#">ownCloud auf Ubuntu Server mit nginx,</a>

[acme.sh](#), [Cloud](#), [Fail2ban](#), [Home-Server](#), [Let's Encrypt](#), [Linux](#), [MariaDB](#), [Nextcloud](#), [nginx](#), [PHP](#), [Redis](#), [Reverse Proxy](#), [SSL](#), [TLS](#), [TLSv1.3](#), [Ubuntu](#), [Ubuntu 20.04 LTS](#), [Ubuntu Server](#), [ufw](#)

[Jitsi Meet: Videokonferenz-System unter Ubuntu Server mit nginx](#)

[Webserver-Konfiguration \(nginx\): Mehrere Webanwendungen auf einem Server hosten](#)

## Kommentare: 182

Rob sagt:

31. Juli 2020 um 21:09 Uhr

Hi Jan,

die letzten 2 Kommentare kannst Du löschen...

Irgendwie hadere ich noch mit einer SSL-Zertifikat Erstellung die dann für alle subdomains ein gültiges .pem bereitstellt..

auch der STUN/TURN Server will noch nicht dem Test genügen :-/

[Antworten](#)

Jan sagt:

2. August 2020 um 9:52 Uhr

Hi Rob,

warum haderst du noch mit den Zertifikaten? Eigentlich ja eine praktische Sache.  
Die Kommentare habe ich wie gewünscht gelöscht.

Gruß,  
Jan

[Antworten](#)

Martin sagt:

1. August 2020 um 17:15 Uhr

Hi Jan,

irgendwie sind meine letzten Kommentare mit Deinen Antworten nach dem Seitenumbruch auf Kommentarseite 3 nicht mehr zu sehen. Es ging nochmal darum, ob man eine Letsencrypt-Zertifikatseinrichtung rückgängig machen und nochmal neu durchführen kann.

Das Problem ist ja leider noch immer folgender Fehler im nginx-Log:

```
„2020/08/01 16:46:57 [error] 1499#1499: *8548 open() „/var/www/letsencrypt/.well-known/acme-challenge /xxxxxxxxxxxxx“ failed (2: No such file or directory), client: 66.133.109.36, server: cloud.meinedomain.de, request: „GET /.well-known/acme-challenge/xxxxxxxxxxxxx HTTP/1.1“, host: „meinedomain.de“ “
```

Letsencrypt geht also auf die Second-Level-Domain („host: „meinedomain.de“ “ und nicht, wie erwartet, „host: „cloud.meinedomain.de“ „).

Ich setze mittlerweile schon eine brachliegenden zweiten Server mit einer anderen Domäne nach Deiner Anleitung auf, um zu sehen, ob ich den Fehler erneut bekomme.

Ab morgen geht's aber erstmal 1 Woche in den Urlaub :). Ich freue mich trotzdem über jeden Lösungsvorschlag.  
Vielen Dank!

Grüße  
Martin

[Antworten](#)

Jan sagt:

2. August 2020 um 9:56 Uhr

Hi Martin,

OK, komisch wenn da was verloren gegangen sein sollte. Dann einfach nochmal:  
Du kannst die Zertifikate nochmals beantragen, musst dann nur ein `--force` am Ende des Befehls hinzufügen.  
Wenn du das zu oft für eine Domain machst (weil es z.B. nicht funktioniert beim ersten Mal), dann sperrt Let's Encrypt die Domain für 7 Tage (Rate Limit). Daher lieber nach dem ersten fehlgeschlagenen Versuch aufhören und erst einmal den Fehler wie im Artikel beschrieben suchen.

Gruß,  
Jan

[Antworten](#)

Lars sagt:

13. August 2020 um 10:31 Uhr

Hallo Jan,

erstmal vielen Dank für die hervorragende Anleitung. Toll, dass du sowas zur Verfügung stellst!

Bei mir funktioniert alles, ich kann mit XYZ.duckdns.org von extern auf meine Nextcloud mit SSL-Zertifikat zugreifen. Nun möchte ich aber noch über cloud.meinedomain.de auf die Nextcloud. Dazu habe ich bei meinem Anbieter Hetzner die subdomain cloud per CNAME-Record auf XYZ.duckdns.org umgeleitet. Wenn ich die Nextcloud so aufrufe, also über cloud.meinedomain.de erhalte ich im Browser die Warnung, dass das Zertifikat ungültig ist, da es auf XYZ.duckdns.org ausgestellt wurde. Für cloud.meinedomain.de habe ich bei Hetzner ein eigenes Let's Encrypt Zertifikat erstellt. Irgendwas beisst sich und ich weiss nicht, an welcher Stelle. Muss ich cloud.meinedomain.de noch irgendwo eintragen?

Danke und Grüsse, Lars

[Antworten](#)

[« Zurück](#) [1](#) [2](#) [3](#)

## Schreibe einen Kommentar

Deine E-Mail-Adresse wird nicht veröffentlicht. Erforderliche Felder sind mit \* markiert.

**Kommentar**

**Name \***

**E-Mail \***

**Website**

Ich stimme den [Datenschutzbestimmungen](#) zu. \*

**Kommentar abschicken**

Impressum & Datenschutz

DecaTec

[Mastodon](#) | [Twitter](#) | [Codeberg](#) | [GitHub](#)

Designed by Ugesi. Powered by WordPress.