

Git - Versionskontrollsystem

Git ist ein weit verbreitetes Versionskontrollsystem, das von Entwicklern zum effektiven Management und zur Nachverfolgung von Änderungen in Softwareprojekten, Dokumenten oder anderen Sammlungen von Informationsdateien verwendet wird. Entwickelt von Linus Torvalds im Jahr 2005, ermöglicht Git die Zusammenarbeit mehrerer Entwickler an einem gemeinsamen Projekt, indem es ihnen erlaubt, Änderungen in einem dezentralen Repository zu speichern, zu teilen und zu integrieren

- Installation
 - Gitea Docker installation mit mariadb und SSL
- Bedienung
 - Client einrichten

Installation

Wenn man nicht github gitlab etc. verwenden möchte sondern sein eigenes Git.
hier mehrere Installationsmöglichkeiten.

Gitea Docker installation mit mariadb und SSL

Beschreibung:

Ein Docker Container mit gitea. Der als Datenbank mariaDB benutzt und ein Lets Encrypt SSL Certificate holt oder ein selbstsigniertes SSL Certificate erstellt.

Docker compose mit Lets Encrypt:

Der Domainname sollte bereits auf den Server per IP a oder AAA record linken. Je nachdem ob ipv4 oder ipv6.

Genauso sollte docker compose auch installiert sein.

```
apt install docker.io docker-compose
```

Falls apparmor auf dem system nicht installiert ist

```
# AppArmor installieren, falls es nicht installiert ist  
apt-get install apparmor apparmor-utils
```

Als erstes unser Verzeichnis erstellen in root.
Darin liegen unser config Dateien für den container

```
mkdir -p /root/gitea/mariadb  
mkdir -p /root/gitea/data
```

Die .env Datei

```
# Gitea Einstellungen  
USER_UID=1000  
USER_GID=1000  
#Registrieren link einblenden ja/nein true/false  
GITEA__service__DISABLE_REGISTRATION=true
```

```
DISABLE_REGISTRATION=true

# Datenbank Einstellungen
DB_ROOT_PASS=rootpassword
DB_USER=gitea
DB_PASSWD=gitea
DB_NAME=gitea

# Port Einstellungen
GITEA_HTTP_PORT=3000
GITEA_SSH_PORT=222

# Volume directories
#Pfade mit Slash vorran. Denn die werden dann zusmmanegbaut
BASE_PATH_DIR=/root/gitea
MARIADB_VOLUME_DIR=/mariadb
DATA_VOLUME_DIR=/data
#Das ist ein Teil des certbot Pfades ,da dies eine Variable ist, da kommt noch eine weitere Variable im bestehen
Verlauf daran ${CERTBOT_VOLUME_DIR}-log
CERTBOT_VOLUME_DIR=/certbot

# Domain and email for Let's Encrypt
DOMAIN_NAME=ihredomain.de
LETSENCRYPT_EMAIL=ihre-email@beispiel.de
```

Die Composer Datei

```
version: '3.8'

services:
  server:
    image: gitea/gitea:latest
    environment:
      - USER_UID=${USER_UID}
      - USER_GID=${USER_GID}
      - DB_TYPE=mysql
      - DB_HOST=db:3306
      - DB_NAME=${DB_NAME}
      - DB_USER=${DB_USER}
      - DB_PASSWD=${DB_PASSWD}
```

restart: always

volumes:

- \${BASE_PATH_DIR}\${DATA_VOLUME_DIR}:/data

ports:

- "\${GITEA_HTTP_PORT}:3000"
- "\${GITEA_SSH_PORT}:22"

depends_on:

- db
- webserver

db:

image: mariadb:latest

restart: always

environment:

- MYSQL_ROOT_PASSWORD: \${DB_ROOT_PASS}
- MYSQL_USER: \${DB_USER}
- MYSQL_PASSWORD: \${DB_PASSWD}
- MYSQL_DATABASE: \${DB_NAME}

volumes:

- \${BASE_PATH_DIR}\${MARIADB_VOLUME_DIR}:/var/lib/mysql

certbot:

image: certbot/certbot

volumes:

- \${BASE_PATH_DIR}\${CERTBOT_VOLUME_DIR}-etc:/etc/letsencrypt
- \${BASE_PATH_DIR}\${CERTBOT_VOLUME_DIR}-www:/var/www/certbot
- \${BASE_PATH_DIR}\${CERTBOT_VOLUME_DIR}-log:/var/log/letsencrypt

entrypoint: '/bin/sh -c'

webserver:

image: nginx:alpine

restart: unless-stopped

volumes:

- ./nginx.conf:/etc/nginx/nginx.conf:ro
- \${BASE_PATH_DIR}\${CERTBOT_VOLUME_DIR}-etc:/etc/letsencrypt
- \${BASE_PATH_DIR}\${CERTBOT_VOLUME_DIR}-www:/var/www/certbot
- \${BASE_PATH_DIR}\${CERTBOT_VOLUME_DIR}-log:/var/log/letsencrypt
- ./nginx-log:/var/log/nginx

ports:

```
- "80:80"  
- "443:443"
```

Init script, erstellt die nginx config und holt erstmals das certificate.

```
#!/bin/bash  
  
# Laden der Umgebungsvariablen aus der .env-Datei  
if [ -f .env ]; then  
    export $(grep -v '^#' .env | xargs)  
else  
    echo ".env-Datei nicht gefunden!"  
    exit 1  
fi  
  
# Überprüfen, ob DOMAIN_NAME und LETSENCRYPT_EMAIL gesetzt sind  
if [ -z "$DOMAIN_NAME" ] || [ -z "$LETSENCRYPT_EMAIL" ]; then  
    echo "DOMAIN_NAME und/oder LETSENCRYPT_EMAIL sind in der .env-Datei nicht gesetzt!"  
    exit 1  
fi  
  
# Erstellen der benötigten Verzeichnisse  
mkdir -p /root/gitea/mariadb  
mkdir -p /root/gitea/data  
mkdir -p ./certbot-etc  
mkdir -p ./certbot-www  
mkdir -p ./certbot-log  
mkdir -p ./nginx-log  
  
# Erstellen der Nginx-Konfigurationsdatei für den ersten Lauf (ohne SSL)  
cat << EOF > ./nginx.conf  
user nginx;  
worker_processes auto;  
  
error_log /var/log/nginx/error.log warn;  
pid /var/run/nginx.pid;  
  
events {  
    worker_connections 1024;  
}
```

```
http {
    include    /etc/nginx/mime.types;
    default_type application/octet-stream;

    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile    on;
    #tcp_nopush  on;

    keepalive_timeout 65;

    server {
        listen 80;
        server_name ${DOMAIN_NAME};

        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
            try_files $uri =404;
        }
    }
}
EOF

# Starten des Webserver-Containers (erster Lauf ohne SSL)
echo "Starte Webserver-Container..."
docker-compose down
docker-compose up -d webserver

# Warten, um sicherzustellen, dass Nginx vollständig gestartet ist
echo "Warte auf Webserver..."
sleep 5

# Ausführen des Certbot für den ersten Lauf
echo "Starte Certbot..."
docker-compose run --rm --entrypoint "" certbot certbot certonly --webroot --webroot-path=/var/www/certbot \
```

```
--email ${LETSENCRYPT_EMAIL} --agree-tos --no-eff-email \  
--domain ${DOMAIN_NAME} --rsa-key-size 4096
```

Aktualisieren Sie die Nginx-Konfigurationsdatei, um SSL zu aktivieren

```
cat << EOF > ./nginx.conf
```

```
user nginx;
```

```
worker_processes auto;
```

```
error_log /var/log/nginx/error.log warn;
```

```
pid /var/run/nginx.pid;
```

```
events {
```

```
    worker_connections 1024;
```

```
}
```

```
http {
```

```
    include /etc/nginx/mime.types;
```

```
    default_type application/octet-stream;
```

```
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
```

```
        '$status $body_bytes_sent "$http_referer" '
```

```
        '"$http_user_agent" "$http_x_forwarded_for";
```

```
    access_log /var/log/nginx/access.log main;
```

```
    sendfile on;
```

```
    #tcp_nopush on;
```

```
    keepalive_timeout 65;
```

```
    server {
```

```
        listen 80;
```

```
        server_name ${DOMAIN_NAME};
```

```
        location /.well-known/acme-challenge/ {
```

```
            root /var/www/certbot;
```

```
            try_files $uri =404;
```

```
        }
```

```
        location / {
```

```

        return 301 https://\${host}\$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name ${DOMAIN_NAME};

    ssl_certificate /etc/letsencrypt/live/${DOMAIN_NAME}/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/${DOMAIN_NAME}/privkey.pem;

    location / {
        proxy_pass http://server:${GITEA_HTTP_PORT};
        proxy_set_header Host \$host;
        proxy_set_header X-Real-IP \$remote_addr;
        proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto \$scheme;
        client_max_body_size 100M;
    }
}
}
EOF

# Starten Sie den Webserver-Container neu, um die aktualisierte Konfiguration zu verwenden
echo "Starte Webserver-Container neu..."
docker-compose down
docker-compose up -d webserver

echo "Setup abgeschlossen. Bitte überprüfen und anpassen Sie die nginx.conf entsprechend."

```

Lets Encrypt erneuern

Damit jede Nacht das Zertifikat erneuert wird legen wir eine Systemd Service Datei an.
Dazu

```
nano /etc/systemd/system/certbot-renew.service
```

Inhalt

```
[Unit]
Description=Certbot Renewal
```

```
[Service]
Type=oneshot
ExecStart=/usr/bin/docker-compose -f /root/gitea/docker-compose.yml run --rm --entrypoint "" certbot certbot
renew --quiet
ExecStartPost=/usr/bin/docker-compose -f /root/gitea/docker-compose.yml restart webserver
```

Und nun den passenden Timer dazu

```
nano /etc/systemd/system/certbot-renew.timer
```

Inhalt

```
[Unit]
Description=Timer for Certbot Docker Container Renewal

[Timer]
OnCalendar=daily
Persistent=true

[Install]
WantedBy=timers.target
```

Aktivieren des Timers

```
systemctl enable certbot-renew.timer
```

Starten des Timers

```
systemctl start certbot-renew.timer
```

Überprüfen des Timers

```
systemctl list-timers certbot-renew.timer
```

Starten:

Nun mit docker-compose up -d den kram starten.

URL Aufrufen:

<http://git.yourdomain.com>

Die Einstellungen wurden aus der app.ini übernommen und aus den umgebungsvariablen, wie man sieht

Erstkonfiguration

Wenn du Gitea in einem Docker-Container nutzt, lies bitte die [Dokumentation](#), bevor du irgendwelche Einstellungen veränderst.

Datenbankeinstellungen

Gitea benötigt MySQL, PostgreSQL, MSSQL, SQLite3 oder TiDB (MySQL-Protokoll).

Datenbanktyp *

Host *

Benutzername *

Passwort *

Datenbankname *

Allgemeine Einstellungen

Seitentitel *
Du kannst hier den Namen deines Unternehmens eingeben.

Repository-Verzeichnis *
Remote-Git-Repositories werden in diesem Verzeichnis gespeichert.

Git-LFS-Wurzelpfad
In diesem Verzeichnis werden die Dateien von Git LFS abgespeichert. Leer lassen, um LFS zu deaktivieren.

Ausführen als *
Der Nutzer unter dem Gitea ausgeführt wird. Beachte, dass dieser Nutzer Zugriff auf das Repository-Wurzelverzeichnis haben muss.

Server-Domain *
Domain oder Host-Adresse für den Server.

runterscrollen bis zum Pnkt Email

Dort Email Daten für den versand ausfüllen.

Optionale Einstellungen

▼ E-Mail-Einstellungen

SMTP-Host

SMTP-Port

E-Mail senden als
E-Mail-Adresse, die von Gitea genutzt werden soll. Bitte gib die E-Mail-Adresse im Format „Name“ <email@example.com>“ ein.

SMTP-Benutzername

SMTP-Passwort

E-Mail-Bestätigung benötigt zum Registrieren

E-Mail-Benachrichtigungen aktivieren

► Sonstige Server- und Drittserviceeinstellungen

► Administratoreinstellungen

These configuration options will be written into: /data/gitea/conf/app.ini

[Gitea installieren](#)

Nun noch ein Administrator Konto anlegen, Benutzername und Kennwort vergeben fertig.
Der Name admin ist Intern vergeben und kann nicht genutzt werden.

Administrator-Benutzername

E-Mail-Adresse

Passwort

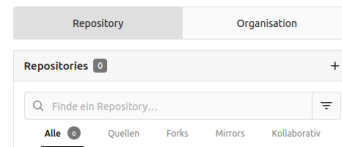
Passwort bestätigen

Dann auf gitea installieren klicken, drücken

These configuration options will be written into: /data/gitea/conf/app.ini

[Gitea installieren](#)

Danach werdet Ihr eingeloggt, fertig



Disable/Enable Registration button nachträglich

Wenn in der der Estprovisionierung true oder fals übergeben wurde wird dies in die app.ini mit übertragen.

Möchte man den Wert in was auch immer später ändern, geht das nur in der app.ini

```
cd ~/gitea/data/gitea/conf/app.ini
```

Nun in ser service Section den Dinst auf true oder fals e stellen, jenach dem was gewünscht ist true disabled den button, false enabled den button

```
...  
[service]  
DISABLE_REGISTRATION = false  
...
```

Bedienung

Client einrichten

Beschreibung:

Um auf das git Repository Zuzugreifen brauchen wir den Namen und den Schlüssel genauso die URL.

Haben wir eine Neue Repo über die Weboberfläche erstellt und die ist noch leer. Dann können wir unser Verzeichnis lokal fürs Pushen vorbereiten. Ist da schon was drin wäre, es ein Pull.

Daten im git config Verzeichnis ändern:

```
git config user.email "meinepersonalemail"  
git config user.name "immohacker"
```

Man kann aber auch einfach im .git Verzeichnis die config Datei editieren und am ende folgendes hinzufügen, dann braucht man keine Befehle.

```
...  
[user]  
  name = <gitusername>  
  email = info@example.com
```

Remote via HTTPS

Verzeichnis vorbereiten für Pull:

```
git init  
git checkout -b main  
git add -A  
git commit -m "first commit"  
git remote add origin https://git.example.com/Strange-Hosting/lb-hetzner-floating-ip.git  
git push -u origin main
```

Danach wird nach dem Git Benutzernamen und Benutzerkennwort gefragt

Ausgabe:

```
estehendes Git-Repository in /home/duffy/strangehosting/loadbalancer/.git/ neuinitialisiert
Zu neuem Branch 'main' gewechselt
[main (Root-Commit) ff1323f] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
error: externes Repository origin existiert bereits.
Username for 'https://git.hacker-net.de':
```

Nach Eingabe wurden die Dateien gepusht:

```
Bestehendes Git-Repository in /home/duffy/strangehosting/loadbalancer/.git/ neuinitialisiert
Zu neuem Branch 'main' gewechselt
[main (Root-Commit) ff1323f] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
error: externes Repository origin existiert bereits.
Username for 'https://git.hacker-net.de': duffyduck
Password for 'https://duffyduck@git.hacker-net.de':
Objekte aufzählen: 3, fertig.
Zähle Objekte: 100% (3/3), fertig.
Schreibe Objekte: 100% (3/3), 210 Bytes | 210.00 KiB/s, fertig.
Gesamt 3 (Delta 0), Wiederverwendet 0 (Delta 0), Pack wiederverwendet 0
remote: . Processing 1 references
remote: Processed 1 references in total
To https://git.hacker-net.de/Strange-Hosting/lb-hetzner-floating-ip.git
 * [new branch]    main -> main
Branch 'main' folgt nun Remote-Branch 'main' von 'origin'.
```

Um die Repo zu aktualisieren (Pushen) werden diese Befehlebenötigt

```
git add -A
git commit -m "first commit"
git push -u origin main
```

Um die Repo zu aktualisieren (Pullen) werden diese Befehlebenötigt

```
git pull
```

Eine Repo Clonen wenn auf dem Computer noch nichts eingerichtet ist.
Neues Verzeichnis erstellen.

```
git clone <Repository-URL>  
git clone https://git.hacker-net.de/Strange-Hosting/lb-hetzner-floating-ip.git
```

Dieser Befehl erstellt ein neues Verzeichnis mit dem Namen des Repositories im aktuellen Arbeitsverzeichnis, initialisiert ein .git Verzeichnis darin, lädt alle Daten für das Repository herunter und checkt eine Arbeitskopie der neuesten Version aus.

Remote VIA SSH

Wenn die Verbindung via SSH aufgebaut werden soll muss erst mal ein Schlüssel auf dem Client erstellt werden:

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

Nun das Ziel angeben:

Zum Beispiel /home/<username>/.ssh/gitea

```
Generating public/private ed25519 key pair.  
Enter file in which to save the key (/home/<username>/.ssh/id_ed25519): /home/<username>/.ssh/gitea
```

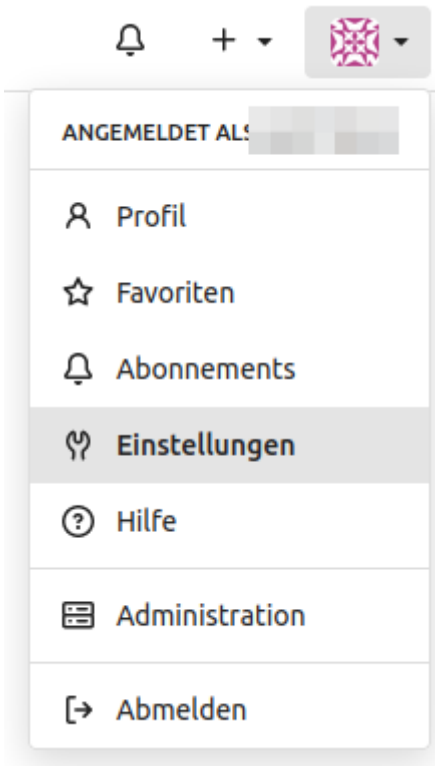
Wenn nach einer gefragt wird, leer lassen. Einfach nur enter / enter

```
Enter passphrase (empty for no passphrase):  
Enter same passphrase again:  
Your identification has been saved in /home/<username>/.ssh/gitea  
Your public key has been saved in /home/<username>/.ssh/gitea.pub  
The key fingerprint is:  
....
```

Den öffentlichen schlüssel anzeigen lassen und im git einfügen.
diesen kopieren

```
cat /home/<username>/.ssh/gitea.pub
```

Dazu gehen wir in Gitea unter Einstellungen



Dann unter SSH-GPG Schlüssel und auf Schlüssel hinzufügen klicken



Schlüsselname eintragen den Key einfügen und auf Schlüssel hinzufügen klicken.

SSH-Schlüssel verwalten Schlüssel hinzufügen

Schlüsselname

mein ssh key vom computer irgendwas

Inhalt

```
startet mit 'ssh-ed25519', 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'sk-ecdsa-sha2-nistp256@openssh.com', oder 'sk-ssh-ed25519@openssh.com'
```

Schlüssel hinzufügen Abbrechen

Diese öffentlichen SSH-Keys sind mit deinem Account verbunden. Der dazugehörigen privaten SSH-Keys geben dir vollen Zugriff auf deine Repositories.
Brauchst du Hilfe? Hier ist GitHubs Anleitung zum [Erzeugen von SSH-Schlüsseln](#) oder zum [Lösen einfacher SSH-Probleme](#).

.ssh/config anpassen

Damit auch der Schlüssel für den Hostnamen genommen wird muss die config angepasst / erstellt werden

```
nano /home/<username>/.ssh/config
```

Dort einfügen / hinzufügen. Die IdentityFile ist der private ssh key

```
Host git.example.com
HostName git.example.com
User <deingitbenutzername>
IdentityFile ~/.ssh/gitea
```

Sollte der SSH Port ein anderer sein, als vom Server. Zum Beispiel weil gitea in einem Container läuft und der port ssh 222 von außen erreichbar ist muss in der ssh conf beim Client noch der Port hinzugefügt werden.

```
Host git.example.com
HostName git.example.com
User <deingitbenutzername>
Port 222
IdentityFile ~/.ssh/gitea
```

Nun die Rechte der Datei anpassen

```
chmod 600 ~/.ssh/config
```

Fertig nun kann sich per ssh verbunden werden.

zum schluss muss nur noch die URL geändert werden wenn vorher https war.

```
git remote remove origin  
git remote add origin git@git.example.com:Strange-Hosting/lb-hetzner-floating-ip.git
```

ansonsten nur

```
git remote add origin git@git.example.com:Strange-Hosting/lb-hetzner-floating-ip.git
```

oder in der `.git/config` datei die URL austauschen wenn vorher schon https war

```
...  
[remote "origin"]  
  url = git@git.example.com:Strange-Hosting/lb-hetzner-floating-ip.git  
...
```