Script-Sammlung

Eine Sammlung verschiedener Scripts über das Basic-Setup hinaus, um dem Admin Dinge zu erleichtern

Backup für die Benutzerdatenbank einrichten

Mit folgendem Backup-Script (bash) können wir die Benutzerdatenbank sichern. Diese können wir außerdem mit `systemd` per Service und Timer automatisieren. Alternativ kann das Bash-Script auch als Cronjob eingebunden werden. Folgendes Script sollte je nach Bedarf angepasst werden (Pfade).

Es ist generell sehr empfehlenswert die Benutzerdatenbank regelmäßig zu sichern. Immer dann, wenn ein Administrator bzw. Manager per Client App neue Accounts hinzufügt, Accounts löscht oder Accounts ändert (z.B. Passwortwechsel), dann werden diese Änderungen in einen transienten Zwischenspeicher (`bffh` Datenbankdatei) gelegt. Dieser sollte mit der lokalen `users.toml` zusammengeführt werden, um alle Nutzer im Fall der Fälle wiederherstellen zu können. Das erfolgt über den `dump-users` Parameter.

> Wird dieser zusammengeführte Dump nicht ausgeführt und crasht der Serverdienst, dann sind alle etwaigen Remote-Änderungen u.U. nicht wiederbringbar.

Backup Script anlegen und konfigurieren

> Wir können theoretisch auch die komplette Datenbank sichern (Benutzerdatenbank + Statusdatenbank). Im Script muss dann `DB_DUMP_CMD` der Parameter `--dump-users` gegen `--dump-db` getauscht werden.

```
vim /opt/fabinfra/scripts/bffh-backup.sh
```

```
#!/bin/bash

# Database dump command
DB_DUMP_CMD="/usr/bin/bffhd -c /etc/bffh/bffh.dhall --dump-users"

# Backup directory
BACKUP_DIR="/etc/bffh/config_backup"

# Number of backups to keep
```

```
NUM_BACKUPS_TO_KEEP=5

# Dry run flag
DRY_RUN=false

# Parse command-line options
while getopts ":n:r" opt; do
  case $opt in
    n)
      NUM_BACKUPS_TO_KEEP="$OPTARG"
      ;;
    r)
      DRY_RUN=true
      ;;
    \?)
      echo "Invalid option: -$OPTARG" >&2
      exit 1
      ;;
    :)
      echo "Option -$OPTARG requires an argument." >&2
      exit 1
      ;;
  esac
done

# Current date and time
CURRENT_DATE=$(date +"%Y%m%d%H%M%S")

# Create a backup file name
BACKUP_FILE="$BACKUP_DIR/db_backup_$CURRENT_DATE.toml"

# Create backup dir, if not existent
if [ ! -d $BACKUP_DIR ]; then
  mkdir -p $BACKUP_DIR
fi

# Execute the database dump command
if [ "$DRY_RUN" = true ]; then
  echo "Dry run mode: Database backup command will not be executed."
  echo "Backup command: $DB_DUMP_CMD $BACKUP_FILE"
```

```
else
    $DB_DUMP_CMD $BACKUP_FILE
fi

# Check if the database dump was successful
if [ $? -eq 0 ]; then
    echo "Database backup completed successfully."

    # Sort backup files by modification time in ascending order
    sorted_backup_files=($(ls -rt "$BACKUP_DIR"))

    # Determine number of backups to delete
    num_backups_to_delete=$((${#sorted_backup_files[@]} - NUM_BACKUPS_TO_KEEP))



    cd $BACKUP_DIR
    # Delete oldest backups if necessary
    if [ $num_backups_to_delete -gt 0 ]; then
        for ((i = 0; i < $num_backups_to_delete; i++)); do
            if [ "$DRY_RUN" = true ]; then
                echo "Dry run mode: Would remove old backup: ${BACKUP_DIR}/${sorted_backup_files[$i]}"
            else
                rm "${sorted_backup_files[$i]}"
                echo "Removed old backup: ${BACKUP_DIR}/${sorted_backup_files[$i]}"
            fi
        done
    fi

else
    echo "Error: Database backup failed."
fi
```

```
chmod +x /opt/fabinfra/scripts/bffh-backup.sh
```

Das Script kann einzeln getestet werden. Es kann mit Parametern gestartet werden:

- n = Anzahl der aufzuhebenden Backups
- r = dry run

```
# Trockenlauf (dry run) - nur testen und nichts löschen
/opt/fabinfra/scripts/bffh-backup.sh -r -n 2

# Backup durchführen und nur die letzten 5 aufheben, alle anderen löschen
/opt/fabinfra/scripts/bffh-backup.sh -n 5
```

Backup-Script mit systemd Timer

Das Script kann als timed Service eingebunden werden, um es so zu automatisieren. Unter Beachtung **obiger Parameter** in `ExecStart` kann folgendes eingebunden werden:

```
vim /opt/fabinfra/scripts/bffh-backup.service
```

```
[Unit]
Description=BFFH Backup Service

[Service]
Type=oneshot
ExecStart=/opt/fabinfra/scripts/bffh-backup.sh -n 10
```

Außerdem als Timer. Dieser muss den gleichen Name haben wie der Service (siehe https://wiki.ubuntuusers.de/systemd/Timer_Units)

```
vim /opt/fabinfra/scripts/bffh-backup.timer
```

```
[Unit]
Description=BFFH Backup Timer

[Timer]
# Run every day at midnight
OnCalendar=daily
Persistent=true

[Install]
WantedBy=timers.target
```

Wir aktivieren und starten das Backup schließlich einmal manuell und prüfen dessen Ausgabe.

Hinweis: Da wir einen Timer für den Service verwenden, müssen wir den Service nicht "enablen". Denn das macht der Timer selbst.

```
sudo ln -sf /opt/fabinfra/scripts/bffh-backup.service /etc/systemd/system/bffh-backup.service
sudo ln -sf /opt/fabinfra/scripts/bffh-backup.timer /etc/systemd/system/bffh-backup.timer
sudo systemctl daemon-reload
sudo systemctl start bffh-backup.service
journal -f -u bffh-backup.service
```

Backup-Script mit cron

Wer lieber auf einen klassischen Cronjob setzten möchte, kann statt dem Service folgendes machen:

```
sudo vim /etc/cron.d/bffh-backup
```

```
#“At 00:00.”
0 0 * * 0 bffh /opt/fabinfra/scripts/bffh-backup.sh -n 10
```

Der Cronjob wird um 00:00 Uhr vom Benutzer `bffh` gestartet. Es gibt keinen Log Output. Dieser lässt sich jedoch leicht ergänzen.

Nutzerdatenbank laden / hashen / prüfen

Nutzerdatenbank laden / hashen

Für das Laden und ggf. Rehashen der Nutzerdatenbank (`users.toml`) kann folgendes Script genutzt werden. Es prüft zunächst, ob die Konfiguration `bffh.dhall` von BFFH valid ist. Wenn nicht, wird nichts unternommen. Im Anschluss werden die Nutzer neu geladen und anschließend in die gleiche Datei **optional** wieder exportiert, um etwaige unverschüsselte Passwörter automatisch zu hashen. BFFH prüft beim Laden die users.toml auf Syntaxfehler. Wem diese Checks nicht ausreichen, der kann sich dem erweiteren Validator-Script widmen,

> Hinweis: Das Script basiert auf einem auf dem System aktiven **systemd** Service namens `bffh.service`

```
mkdir -p /opt/fabinfra/scripts/
vim /opt/fabinfra/scripts/bffh-load-users.sh
```

```
#!/bin/bash
#check config
BFFHD="/usr/bin/bffhd"
DATA="/etc/bffh"
CFG="$DATA/bffh.dhall"
USERS="$DATA/users.toml"

while [[ $# -gt 0 ]]; do
  case $1 in
    -h|--help)
      echo "-h|--help   - show help"
      echo "-r|--rehash - overwrite users.toml with hashed passwords (ensure secure secrets)"
      exit 1
      ;;
    -r|--rehash)
      REHASH="y"
      shift
      ;;
    -*)
```

```
    ;;
  esac
done

echo "use -h|--help to show additional script options!"

$BFFHD --check --config $CFG > /dev/null
if [ $? == 0 ]; then
  #pre-check bffh.dhall
  echo "Config is valid. Loading users ..."
  $BFFHD --verbose --config $CFG --load-users=$USERS

  if [ $? == 0 ]; then
    #then load users.toml
    $BFFHD --verbose --config $CFG --dump-users /tmp/users.toml --force
  else
    echo "Error: Newly given users.toml is invalid!"
    exit 1
  fi

  #if this was successful and service is running, restart it, also do nothing
  if [ $? == 0 ]; then
    if [[ $REHASH == "y" ]]; then #overwrite users if --rehash option is given (not null)
      echo "Rehasing users.toml!"
      cat /tmp/users.toml > $USERS
      rm /tmp/users.toml
    fi
    FAS="bffh.service"
    STATUS="$(systemctl is-active $FAS)"
    if [ "${STATUS}" = "active" ]; then
      echo "restarting $FAS"
      systemctl restart $FAS
    else
      echo -e "\n\n$FAS not active/existing. Not restarting bffh service!"
    fi
  fi

else
  echo "Error: Currently loaded users.toml is invalid!"
  exit 1
```

```
fi
```

```
chmod +x /opt/fabinfra/scripts/bffh-load-users.sh
```

Nutzerdankenbank prüfen (users.toml validator)

Ein Python-Script erlaubt die Auswertung einer `users.toml` Datei mit folgenden Features:

- zählt die Nutzer und deren zugewiesene Rollen
- überprüft auf Duplikate bei Nutzernamen, Passwörtern und Cardkeys
- validiert Passwörter auf Verschlüsselung (Argon2)
- validiert eventuell vergebene Cardkeys (UUID)
- gibt Hinweise bei möglichen Datenbankfehler (z.B. falsche Datentypen)
- erzeugt eine Statistik über die Verwendung von Passwörtern, Cardkeys und Rollen

> Das Script benötigt mindestens Python 3.11. Erst ab dieser Version wird `tomllib` unterstützt!

Tool installieren:

```
cd /opt/fabinfra/tools/
git clone https://github.com/vmario89/fabaccess-users-toml-validator.git
```

Benutzen:

```
python3 /opt/fabinfra/tools/fabaccess-users-toml-validator/validate.py
```

> Über den optionalen Parameter `--db /opt/fabinfra/bffh-data/config/users.toml` kann auch eine andere Benutzerdatenbankdatei an Stelle der Standarddatei `/etc/bffh/users.toml` angegeben werden!

Fehlermeldungen und deren Bedeutung

Invalid initial character for a key part (at line x, column y)

Möglicher enthält der Benutzername Sonderzeichen wie Umlaute und ist nicht in Hochkommas geführt. In diesem Falle bricht das Script ab. `[Ö]` ist kein gültiger Benutzername, `["Ö"]` schon.

FabAccess State Script (Email-Warnung bei Fehlerzustand)

Ein kleines Helferscript, was an ein beliebiges Mail-Postfach eine Email sendet, wenn unser systemd Service `bffh.service` nicht mehr korrekt läuft. Falls der Service nicht installiert wurde, dann wird das Script einen Fehler werfen.

Wir verwenden außerdem im Script das smarte smtp-cli Tool von mludvig: https://github.com/mludvig/smtp-cli/tags. Es kann jedoch auch jeder andere beliebige Mail-Client verwendet werden, um cli-basierte Nachrichten zu versenden. Kleiner Fix: https://github.com/mludvig/smtp-cli/issues/28

```
# smtp-cli installieren
sudo apt install cpanminus
sudo cpanm Net::DNS
sudo apt install libio-socket-ssl-perl libdigest-hmac-perl libterm-readkey-perl libmime-lite-perl libfile-libmagic-perl libio-socket-inet6-perl
cd /opt
sudo wget https://github.com/mludvig/smtp-cli/archive/refs/tags/v3.10.zip
sudo unzip v3.10.zip
sudo rm v3.10.zip
sudo mv /opt/smtp-cli-3.10/smtp-cli /usr/bin/
sudo rm -rf /opt/smtp-cli-3.10/

mkdir -p /opt/fabinfra/scripts/
vim /opt/fabinfra/scripts/bffh-state.sh
```

```
#/!bin/bash
SMTP_SERVER="smtp.fablabchemnitz.de:587"
SMTP_MAILBOX="REDACTED"
SMTP_PW='REDACTED'

FROM="fabaccess.noreply@stadtfabrikanten.org"
TO="webmaster@stadtfabrikanten.org"
SUBJECT="FabAccess BFFH Service failed"

MAILFILE="/tmp/mail.txt"
```

```
systemctl status bffh.service 2>&1 > ${MAILFILE}
if [ $? != 0 ]; then #wenn exit code von systemd unit nicht 0
cat ${MAILFILE}
    /usr/bin/smtp-cli --server ${SMTP_SERVER} --user ${SMTP_MAILBOX} --password ${SMTP_PW} --
from "${FROM}" --to "${TO}" --subject "${SUBJECT}" --body-plain ${MAILFILE}
fi
```

```
chmod +x /opt/fabinfra/scripts/bffh-state.sh
```

Und dann testen wir das Script. Es gibt den Status auf der Kommandozeile aus und sollte parallel eine Email an das Zielpostfach mit gleichem Inhalt senden.

Einbinden in systemd Service

Wir können dieses Script auch direkt in unseren bffh.service einbinden.

Dazu fügen wir folgende Zeile in die Sektion `[Service]` ein:

```
vim /etc/systemd/system/bffh.service
```

```
ExecStopPost=/usr/bin/bash /opt/fabinfra/scripts/bffh-state.sh $EXIT_CODE
```

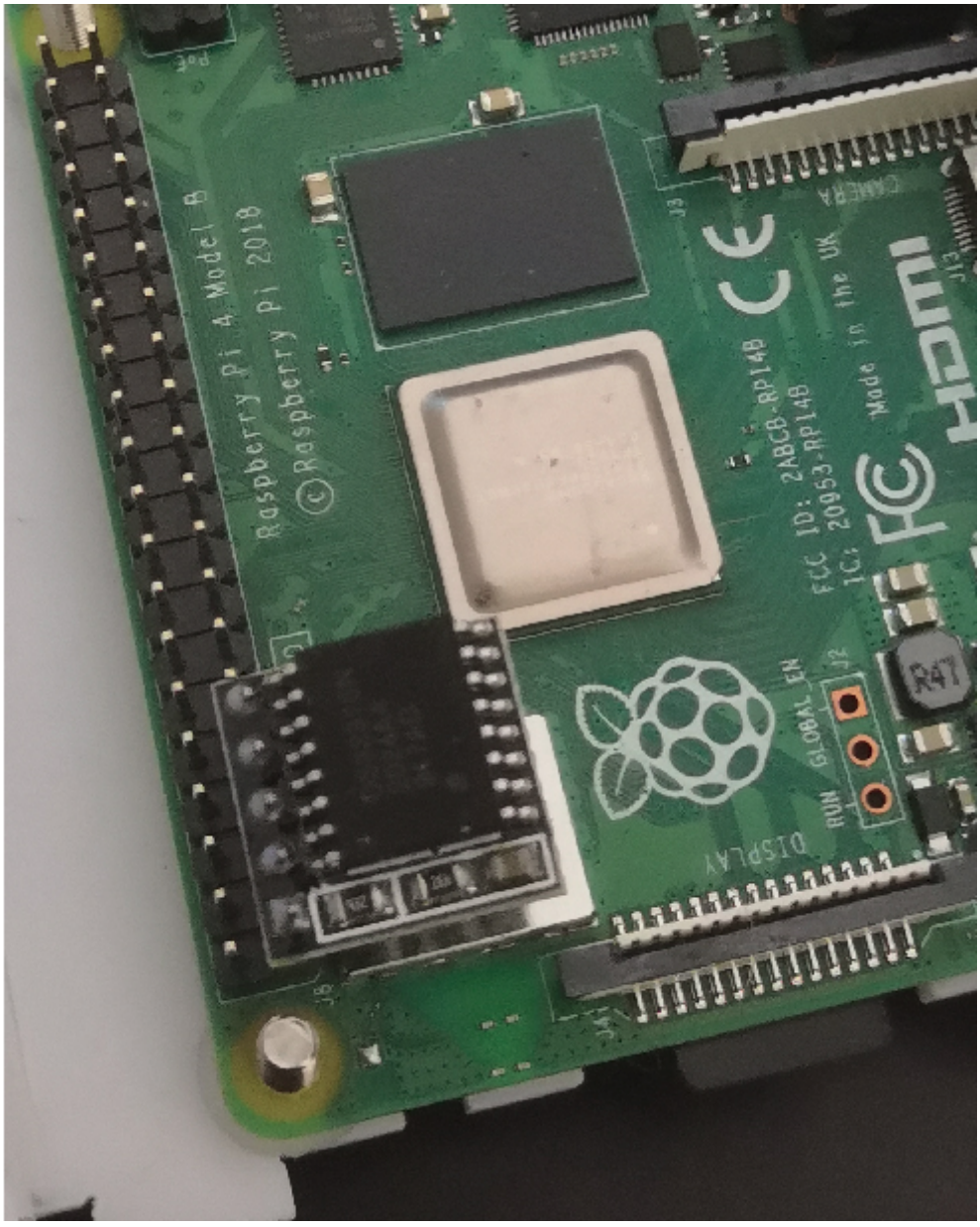
Danach muss der Service neu gestartet werden:

```
sudo systemctl daemon-reload
sudo systemctl restart bffh.service
```

Echtzeituhr (RTC) Modul DS3231 für Raspberry Pi

Wer FabAccess auf einem Raspberry Pi betreiben möchte und eine Echtzeituhr für genaue und unabhängige Zeitstempel wünscht, kann ein RTC Modul installieren und konfigurieren:

Diese Anleitung basiert auf https://learn.adafruit.com/adding-a-real-time-clock-to-raspberry-pi/set-rtc-time



Das von uns verbaute Uhrenmodell ist DS3231

Wir fügen in der Boot-Konfiguration folgende Device Tree Overlays (`dtoverlay`) ein:

```
sudo vim /boot/config.txt
```

```
# Additional overlays and parameters are documented /boot/overlays/README
dtoverlay=i2c-rtc,ds3231
```

```
sudo vim /etc/modules
```

```
i2c-bcm2708
i2c_dev
```

Danach starten wir den Pi neu

```
sudo reboot
```

Nach dem Restart prüfen wir, ob unsere Echtzeituhr verfügbar ist.

```
i2cdetect -y 1 #should be visible at 0x68
```

Wenn ja, dann können wir die standardmäßig installierte "Fake Hardware Clock" deinstallieren:

```
sudo apt remove fake-hwclock
sudo update-rc.d -f fake-hwclock remove
sudo systemctl disable fake-hwclock
```

Noch etwas nachstellen und dann die Uhr in Betrieb nehmen:

```
sudo vim /lib/udev/hwclock-set
```

```
#!/bin/sh
# Reset the System Clock to UTC if the hardware clock from which it was copied by the kernel
was in localtime.
dev=$1
/sbin/hwclock --rtc=$dev --hctosys
```

```
sudo hwclock -r
sudo hwclock --verbose -r
sudo hwclock -w #write the time
sudo hwclock -r #read the time
```