**HIGH AVAILABILITY Umgebung**

IOBROKER AUF PROXMOX v1.40

Inhaltsverzeichnis

[Änderungen 2](#_Toc87677688)

[Vorwort 2](#_Toc87677689)

[Ziel 2](#_Toc87677690)

[Vorrausetzungen & Allgemeine Infos 2](#_Toc87677691)

[Offene Punkte & Ideen für Erweiterungen 3](#_Toc87677692)

[Mögliche zukünftige Erweiterungen 3](#_Toc87677693)

[TODOs 3](#_Toc87677694)

[Bekannte Probleme 3](#_Toc87677695)

[Hardware-Watchdog 3](#_Toc87677696)

[GlusterFS kann nicht mehr gemounted werden 4](#_Toc87677697)

[Installation 4](#_Toc87677698)

[Proxmox 4](#_Toc87677699)

[Container und VMs 4](#_Toc87677700)

[Cluster erstellen 5](#_Toc87677701)

[Optional: Ein Raspberry PI als qdevice 7](#_Toc87677702)

[GlusterFS als Shared Storage 8](#_Toc87677703)

[Container erstellen 15](#_Toc87677704)

[Hardware Watchdog aktivieren 16](#_Toc87677705)

[HA aktivieren 16](#_Toc87677706)

[HA Testen 18](#_Toc87677707)

[Fehlerbehandlung 18](#_Toc87677708)

[Weiterführendes 19](#_Toc87677709)

[Redis 20](#_Toc87677710)

[Vorrausetzungen 20](#_Toc87677711)

[Installation & Konfiguration redis-server 20](#_Toc87677712)

[Installation & Konfiguration redis-sentinel 23](#_Toc87677713)

[Befehlsübersicht 26](#_Toc87677714)

[Weiterführendes 26](#_Toc87677715)

[ioBroker 27](#_Toc87677716)

[Installation & Konfiguration 27](#_Toc87677717)

[GlusterFS Daten bereitstellen 28](#_Toc87677718)

[Smartmeter Anbindung über ser2net & socat 29](#_Toc87677719)

[Archiv 30](#_Toc87677720)

[USBIP 30](#_Toc87677721)

[Installation 30](#_Toc87677722)

[Konfiguration 31](#_Toc87677723)

[Befehle 34](#_Toc87677724)

[Weiterführendes 34](#_Toc87677725)

## Änderungen

|  |  |  |  |
| --- | --- | --- | --- |
| Datum | Version | Autor | Änderungen |
| 01.09.2021 | V1.00 | Thorsten Walk | initialer Upload |
| 12.09.2021 | V1.10 | Thorsten Walk | RPI als qdevice GlusterFS Arbiter Hardware-Watchdog einige Befehls-Beschreibungen ergänzt |
| 03.10.2021 | V1.20 | Thorsten Walk | Bekannte Probleme, USBIP |
| 18.10.2021 | V1.30 | Thorsten Walk | GlusterFS Daten in ioBroker, GlusterFS Performance Tests |
| 13.11.2021 | V1.40 | Thorsten Walk | * Watchdog Test * HA Test * ser2net an Stelle von USBIP |

## Vorwort

Inspiriert von den Diskussionen aus

* <https://forum.iobroker.net/topic/26327/redis-in-iobroker-%C3%BCberblick>
* <https://forum.iobroker.net/topic/45979/iobroker-hochverf%C3%BCgbar>

ist in den letzten Wochen diese Anleitung entstanden. Großen Dank geht an Ingo (@Apollon77) für das Querlesen, Auskunft geben und korrigieren.

## Ziel

Nach dem Durcharbeiten dieser Anleitung steht eine (virtuelle) HA-Umgebung auf Basis von 3 Proxmox-Nodes zur Verfügung. Innerhalb dieser Nodes werden die Daten der Container auf einem Shared-Storage des Typs GlusterFS verfügbar gehalten. ioBroker ist konfiguriert als file(redis)/objects(redis) und hat Zugriff auf 3 Redis-Server-Nodes mit 6 Redis-Server-Prozessen (je LXC einer für die file- und objects-Datenbank). Redis-Sentinel überwacht die Redis-Nodes und stellt die Verfügbarkeit dieser sicher.

Die Anleitung lässt sich für einen produktiven Betrieb auf echter Hardware, z.B. auf ein 3x Intel NUC Setup, adaptieren. Zum Teil wird auch schon direkt Bezug auf „echte“ Hardware genommen (z.B. Kapitel Watchdog).

Zu einem späteren Zeitpunkt wird die virtuelle Ausgangsituation in dieser Anleitung in den Anhang verschoben und der Fokus auf echte Hardware gelegt.

## Vorrausetzungen & Allgemeine Infos

* Entstanden ist die Anleitung mit Proxmox in der Version 6.4 und Debian 10 Buster
* Grundlegendes Verständnis im Umgang mit Linux, Proxmox und ioBroker
* Die Linux Container (LXC) werden aus einem Debian 10 Template erstellt
* Hardware für das Aufsetzen der virtuellen Testumgebung
* Alle Schritte in dieser Anleitung sind als **root** oder mit vorangestelltem **sudo** auszuführen. Danach ist kein Root Zugriff mehr erforderlich und sollte vermieden werden
* In den blauen Boxen stehen die Shell-Befehle, in den grünen die Shell-Ausgaben
* In den Code-Boxen steht in der Regel in der ersten Zeile der Pfad zur gemeinten Datei **# /pfad/…**

## Offene Punkte & Ideen für Erweiterungen

### Mögliche zukünftige Erweiterungen

* Kapitel: Redis
  + Persistenz: AOF (Append-Only File) anstatt RDB
    - Vorteile, Nachteile, Unterschiede
* Kapitel: Backup erstellen
* Kapitel: Netzwerk erstellen
  + Eigener Switch für die Proxmox Nodes
  + Eigenes Netz (VLAN) für HA
    - Wenn Intel NUC, zweiter LAN-Port per USB-Adapter
      * <https://www.amazon.de/gp/product/B087QFQW6F>
* Kapitel: Hardware erstellen
  + Intel NUCs im als Cluster
    - 19“ Blende für 3 Intel NUC (<https://www.amazon.de/dp/B0886JMZW8>)
    - Alternativen zum Intel NUC (<https://www.asrockind.com/en-gb/4X4%20BOX-4800U>)
* Kapitel: USV erstellen
  + NUT-Server
    - <https://forum.iobroker.net/topic/23688/howto-usv-nut-server-auf-sbc-installieren>a
    - <https://wiki.ledhed.net/index.php?title=Raspberry_Pi_NUT_Server>
    - <https://iot-blog.net/2021/04/04/iobroker-usv-der-erste-schritt-zur-hochverfuegbarkeit/>
    - <https://www.facebook.com/groups/440499112958264/posts/1572910643050433>
  + 19“ USV mit geringer Einbautiefe:
    - <https://www.cyberpower.com/de/de/product/sku/or600erm1u>
    - <https://www.cyberpower.com/de/de/product/sku/or1000erm1u>
* Kapitel: Checkmk erstellen

### TODOs

* Umbau der Anleitung vom virtuellen Testsystem zum Intel-NUC Setup
* HomeMatic mit dem HB-RF-ETH & debmatic/Raspberrymatic
  + - <https://github.com/jens-maus/RaspberryMatic/wiki/Experten-Features#hb-rf-eth-anbindung>
* Testfälle um HA sicherzustellen
* Fertigstellung Befehlsübersichten: kurze Erklärung der Befehle
* Kapitel ioBroker
  + Bestehenden ioBroker umstellen (lokale states/objects > redis), inkl. Hinweis auf Multihost Umgebungen: Slaves nicht migrieren!

### Bekannte Probleme

#### Hardware-Watchdog

Nach Verlust der Netzwerk-Verbindung stirbt der Hardware-Watchdog:

Sep 15 16:14:59 pve01 pve-ha-lrm[1157]: watchdog update failed - Broken pipe

Der Proxmox spezifische Dienst watchdog-mux bleibt im Status ‘Failed’ hängen und lässt sich nur durch einen reboot wieder beleben:

service status watchdog-mux

#### GlusterFS kann nicht mehr gemounted werden

Wenn ein Node nach Verlust der Netzwerk-Verbindung wieder erreichbar ist, kann das GlusterFS nicht mehr eingehangen werden. Dabei ist es egal ob von pve01 > pve02, oder pve02 > pve01.

## Installation

### Proxmox

#### Container und VMs

Für die Testumgebung werden 3 Virtuelle Maschinen mit einer Proxmox 6.4 Installation und je einer zusätzlichen HDD für das GlusterFS benutzt. Diese Vorgehensweise sollte genauso auch auf 3 Intel NUCs für einen Produktiven Betrieb umzusetzen sein.

Damit Proxmox Virtuelle Maschinen mit wiederum Proxmox bereitstellen kann, muss **Nested Virtualization** aktiviert werden:

cat /sys/module/kvm\_intel/parameters/nested

N

N bedeutet, das **Nested Virtualization** nicht aktiviert ist.

Das Kernel-Modul **kvm-intel** für Intel CPUs laden:

echo "options kvm-intel nested=Y" > /etc/modprobe.d/kvm-intel.conf  
modprobe -r kvm\_intel  
modprobe kvm\_intel  
reboot

Danach gibt der folgende Befehl ein Y aus:

cat /sys/module/kvm\_intel/parameters/nested

Y

Siehe auch: <https://pve.proxmox.com/wiki/Nested_Virtualization>

Die 3 VMs innerhalb Proxmox erstellen und grundlegend einrichten:

|  |  |  |
| --- | --- | --- |
| ID | Name | IP / WebUI |
| 100 | pve-test-node-1 | 192.168.1.61 / <https://192.168.1.61:8006> |
| 101 | pve-test-node-2 | 192.168.1.62 / <https://192.168.1.62:8006> |
| 102 | pve-test-node-3 | 192.168.1.63 / <https://192.168.1.63:8006> |

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

* Benutzer
* Standard-Software: vim, git, curl, …
* System Updates
* SSH-Zugang

Darauf achten, dass in den Optionen der VMs



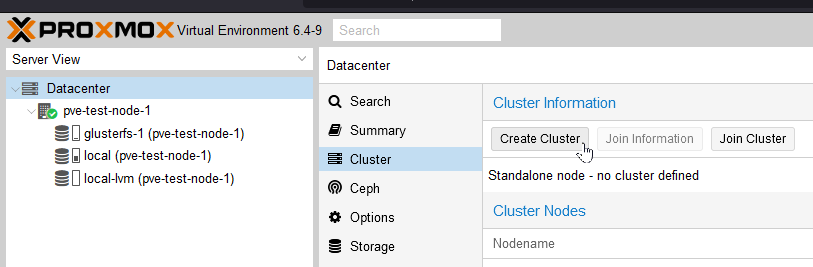
aktiviert ist.

Mit diesen 3 VMs wird im nächsten Kapitel ein Proxmox-Cluster gebildet und darin dann, nach Abschluss des Kapitels „GlusterFS“, die folgenden Linux Container (CT) erstellt:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | IP | OS |
| 100 | pve-test-iobroker | 192.168.1.70 | Debian 10 |
| 101 | pve-test-redis-node-1 | 192.168.1.71 | Debian 10 |
| 102 | pve-test-redis-node-2 | 192.168.1.72 | Debian 10 |
| 103 | pve-test-redis-node-3 | 192.168.1.73 | Debian 10 |

#### Cluster erstellen

Auf pve-test-node-1 über **Datacenter > Cluster > Create Cluster** ein Cluster erstellen:

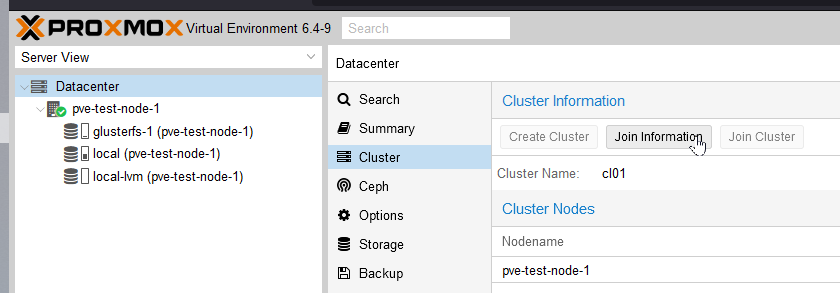


Einen Namen für das Cluster eintragen und mit **Create** bestätigen:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Nach dem anlegen des Clusters über **Join Information** …



Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

… den Schlüssel für das hinzufügen zusätzlicher Nodes kopieren.

Auf pve-test-node-2 und pve-test-node-3 über **Datacenter > Cluster > Join Cluster** dem Cluster CL01 beitreten.

Ein Bild, das Text enthält.

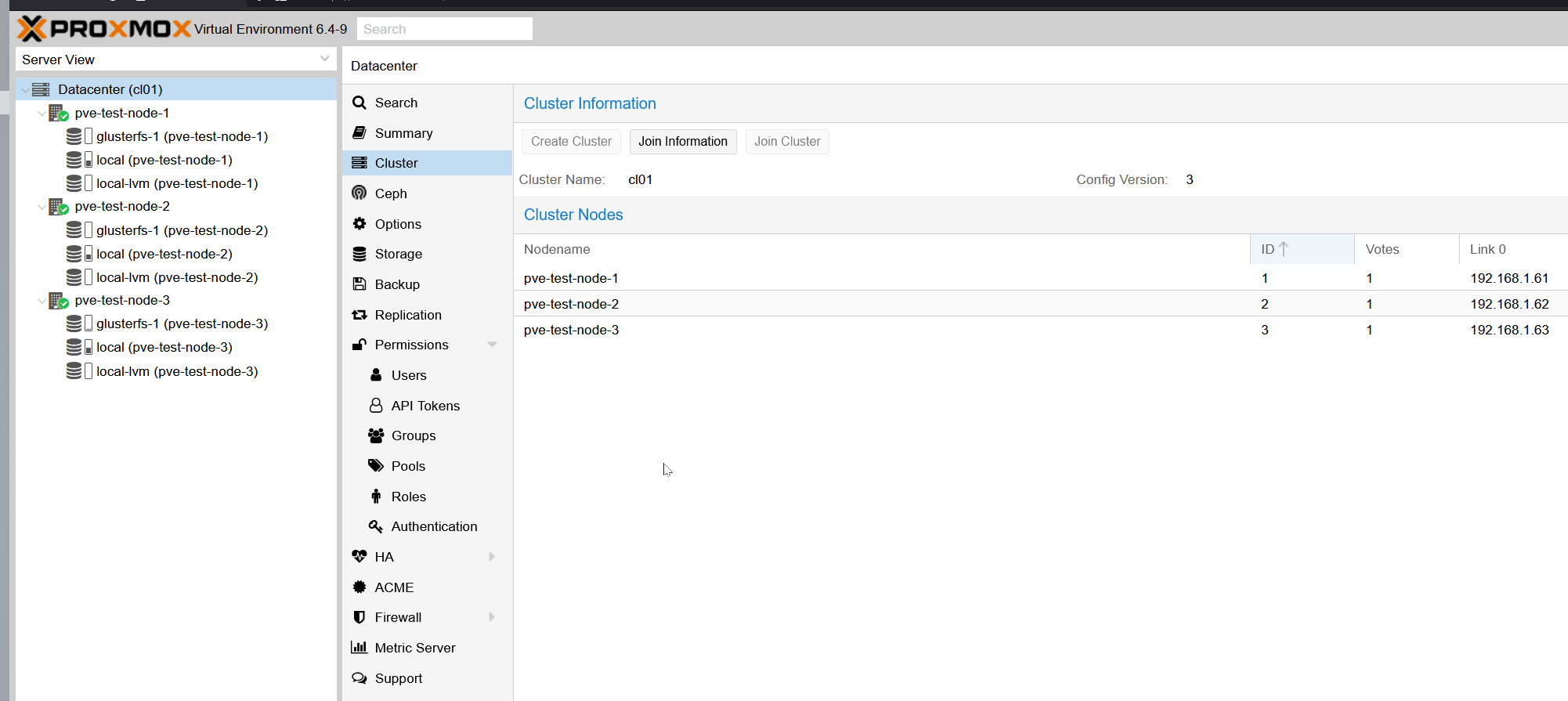
Automatisch generierte Beschreibung

Dazu Schlüssel und Root-Passwort eingeben:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Während dieses Prozesses verlieren die Web-UIs von Node-2 und Node-3 kurz ihre Verbindung. Sobald diese dem Cluster beigetreten sind, können sie wieder normal benutzt werden und zeigen dann jeweils alle Nodes des Clusters an. Damit ist die WebUI des Clusters von allen 3 Nodes erreichbar:



#### Optional: Ein Raspberry PI als qdevice

<https://pve.proxmox.com/pve-docs/chapter-pvecm.html#_corosync_external_vote_support>

Damit das Proxmox-Cluster Beschluss fähig ist (quorated), muss es aus mindestens 3 Nodes bestehen. Als alternative oder auch Zwischenschritt zu einem „echten“ dritten Node kann diese Rolle auch ein Raspberry Pi als „qdevice“ einnehmen. Damit stellt dieser die Beschlussfähigkeit sicher.

**Wichtig: Wird das qdevice eingesetzt, dann auch Teil 8 der GlusterFS Konfiguration beachten.**

##### Auf dem Raspberry PI

Wichtig: Der Raspberry PI muss im selben Subnetz wie die Proxmox-Nodes sein.

Des Weiteren muss der SSH Login als root aktiviert sein. Dies in der /etc/ssh/sshd\_config sicherstellen und den SSH-Dienst neu starten:

PermitRootLogin yes  
systemctl restart sshd.service

Danach wird das Paket für den qdevice Dienst installiert:

apt install corosync-qnetd

##### Auf allen Nodes

Auf den Proxmox-Nodes muss ebenfalls ein Paket für das qdevice installiert werden:

apt install corosync-qdevice -y

##### Auf einem der Nodes

Danach kann auf einem der Proxmox-Nodes mit dem folgenden Befehl das der Raspberry in das Promox-Cluster aufgenommen werden:

pvecm qdevice setup [IP-Raspberry-PI]

Ob das Ganze funktioniert hat, lässt sich mit pvecm status überprüfen

Cluster information  
-------------------  
Name: pve  
Config Version: 3  
Transport: knet  
Secure auth: on

Quorum information  
------------------  
Date: Sun Sep 12 07:38:55 2021  
Quorum provider: corosync\_votequorum  
Nodes: 2  
Node ID: 0x00000001  
Ring ID: 1.274  
Quorate: Yes

Votequorum information  
----------------------  
Expected votes: 3  
Highest expected: 3  
Total votes: 3  
Quorum: 2   
Flags: Quorate Qdevice

Membership information  
----------------------  
 Nodeid Votes Qdevice Name  
0x00000001 1 A,V,NMW 192.168.1.50 (local)  
0x00000002 1 A,V,NMW 192.168.1.51  
**0x00000000 1 Qdevice**

#### GlusterFS als Shared Storage

<https://www.gluster.org>

##### Teil 1 – GlusterFS aktualisieren

**Wichtig: Wenn Proxmox 7 eingesetzt wird, kann dieser Schritt übersprungen werden. Hier wird GlusterFS in der Version 9.2 mit ausgeliefert.**

Mit Proxmox 6.4 wird eine alte Version von GlusterFS ausgeliefert (Version 5.5). Mit den folgenden Schritten kann die GlusterFS auf 8.x angehoben werden.

Quelle: <https://download.gluster.org/pub/gluster/glusterfs/8/LATEST/Debian/>

Siehe auch Kapitel „Weiterführendes“ 🡪 GlusterFS Upgrade Guide v5 auf v8

Achtung: Dies muss auf Node 1,2 und 3 ausgeführt werden!

wget -O - https://download.gluster.org/pub/gluster/glusterfs/8/rsa.pub | apt-key add -  
echo deb [arch=amd64] https://download.gluster.org/pub/gluster/glusterfs/8/LATEST/Debian/buster/amd64/apt buster main > /etc/apt/sources.list.d/gluster.list  
apt update && apt upgrade -y

##### Teil 2 – gstatus installieren

„gstatus is a commandline utility to report the health and other statistics related to a GlusterFS cluster. gstatus consolidates the volume, brick, and peer information of a GlusterFS cluster.

At volume level, gstatus reports detailed infromation on quota usage, snapshots, self-heal and rebalance status“

<https://github.com/gluster/gstatus#install>

curl -LO https://github.com/gluster/gstatus/releases/download/v1.0.6/gstatus  
chmod +x ./gstatusmv ./gstatus /usr/local/bin/gstatus  
gstatus --version

##### Teil 3 – Festplatten für GlusterFS vorbereiten

Achtung: Dies muss auf Node 1,2 und 3 ausgeführt werden!

Prüfen, ob die HDD (in unserem Beispiel /dev/sdb) vorhanden ist:

fdisk -l /dev/sdb

Erwartetes Ergebnis:

Disk /dev/sdb: 32 GiB, 34359738368 bytes, 67108864 sectors  
Disk model: QEMU HARDDISK  
Units: sectors of 1 \* 512 = 512 bytes  
Sector size (logical/physical): 512 bytes / 512 bytes  
I/O size (minimum/optimal): 512 bytes / 512 bytes

**Hinweis: Beim Erstellen der Partition am besten** cgdisk **verwenden, dies stellt sicher, dass das Partition Alignment korrekt gesetzt wird.**

Ein physisches Volume erstellen:

pvcreate /dev/sdb1

Die dazugehörige „Volume Group“ erstellen:

vgcreate vg\_glusterfs /dev/sdb1

Das logical Volume erstellen:

lvcreate --name lv\_glusterfs -l 100%vg vg\_glusterfs

Das Volume mit XFS formatieren:

mkfs.xfs -f -i size=512 -n size=8192 -d su=128K,sw=10 -L GlusterFS /dev/vg\_glusterfs/lv\_glusterfs

Das neue Volume ins System einhängen:

mkdir -p /data/glusterfs  
echo "/dev/mapper/vg\_glusterfs-lv\_glusterfs /data/glusterfs xfs defaults 0 0" >> /etc/fstab  
mount -a

##### Teil 4 – GlusterFS installieren

apt install glusterfs-server glusterfs-client -y

Dafür sorgen das der glusterd auch nach einem Neustart automatisch gestartet wird:

systemctl enable glusterd

Danach sicherstellen, dass der GlusterFS Shared Storage Service nicht aktiviert ist, weil dieser die Mount-Reihenfolge beeinflusst und somit beim Herunterfahren das GlusterFS ggf. unmounted wird, wenn noch VMs / LXCs aktiv sind.

systemctl status glusterfssharedstorage.service

Wenn aktiv, deaktivieren, beenden und den jeweiligen Node neu starten:

systemctl disable glusterfssharedstorage.service  
systemctl stop glusterfssharedstorage.service

Wichtig, am Ende das System neu starten:

reboot

**Grundsätzlich:** Wird irgendwas an der GlusterFS-Konfiguration geändert, sollte das System danach durchgestartet werden.

##### Teil 5 - GlusterFS-Volume erstellen (auf node-1)

Auf pve-test-node-1 Überprüfen, ob die Verbindung zu pve-test-node-2 & pve-test-node-3 hergestellt werden kann:

gluster peer probe 192.168.1.62  
gluster peer probe 192.168.1.63

Erwartetes Ergebnis:

peer probe: success

Ein GlusterFS Volume über Node-1 bis Node-3 erstellen:

gluster volume create glusterfs-1-volume transport tcp replica 3 192.168.1.61:/data/glusterfs 192.168.1.62:/data/glusterfs 192.168.1.63:/data/glusterfs force

Das GlusterFS Volume starten:

gluster volume start glusterfs-1-volume

Spezielle Einstellungen für größere VMs und große Images setzen aktivieren:

gluster volume set glusterfs-1-volume group virt

Diese danach überprüfen:

gluster volume info

features.shard: on

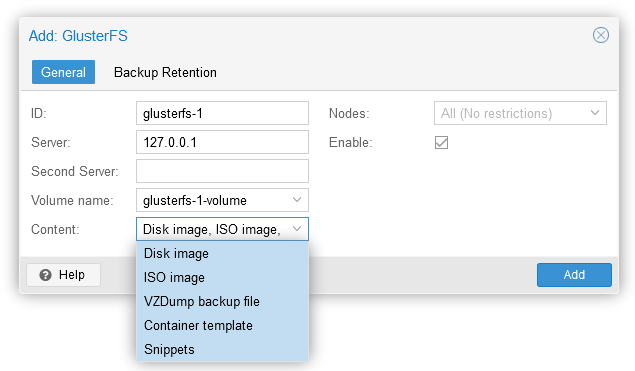
##### Teil 6 – GlusterFS-Volume in Proxmox hinzufügen

Über **Datacenter > Storage > GlusterFS** auf einem der Nodes einen neuen Storage des Typs GlusterFS erzeugen:

Ein Bild, das Text enthält.

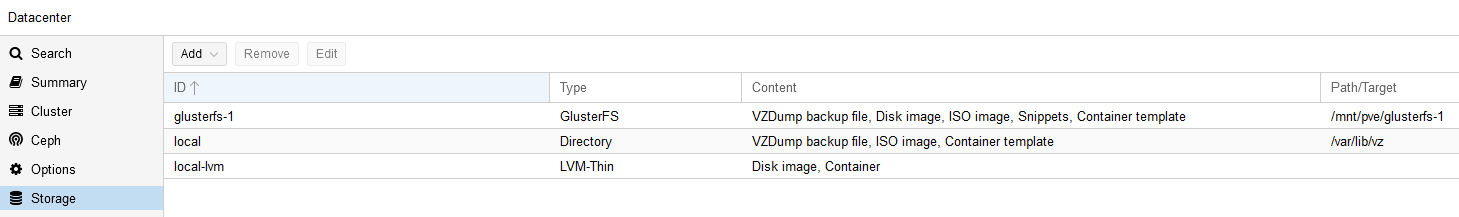
Automatisch generierte Beschreibung

Den Storage mit den folgenden Daten anlegen. Als IP die des localhost (127.0.0.1) nehmen. Somit verbindet sich jeder Proxmox-Node auf sich selbst und den Rest erledigt das GlusterFS-Cluster.



Dies wird, da Proxmox als Cluster konfiguriert ist, automatisch auf die anderen Nodes des Clusters synchronisiert.

Ein erfolgreicher angelegter GlusterFS Storage sieht in der WebUI dann so aus:



##### Teil 7 – Workaround: GlusterFS für LXC bereitstellen

Aktuell können Images der Container (LXCs) im Format RAW nicht auf einem GlusterFS Storage abgelegt werden. Durch den folgenden Workaround wird dies möglich. Einen Nachteil gibt es allerdings: Es können keine Snapshots angelegt werden.

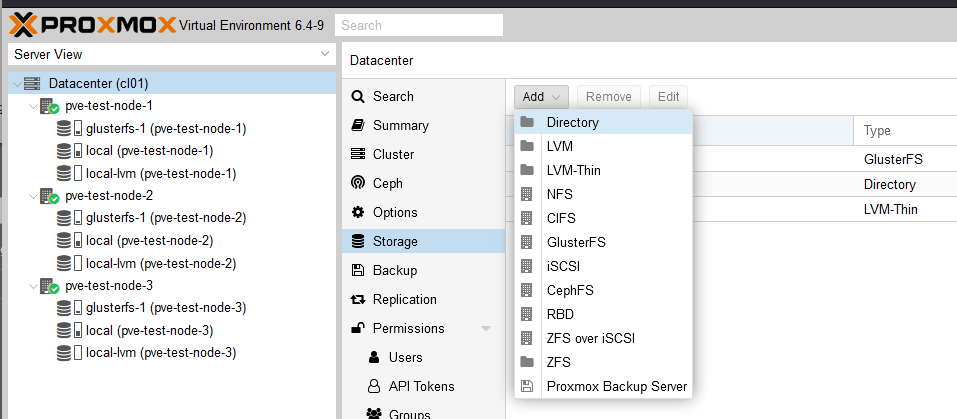
Die **/etc/fstab** um folgenden Eintrag, auf **jedem** der Nodes, erweitern und danach diesen in das System einhängen. Darauf achten, dass der Mount-Pfad (hier im Beispiel */mnt/pve/****glusterfs-1***) der ID / dem Namen entspricht, welcher vorherigen Kapitel „GlusterFS-Volume in Proxmox hinzufügen“ gewählt wurde:

echo "localhost:glusterfs-1-volume /mnt/pve/glusterfs-1 glusterfs defaults,\_netdev 0 0" >> /etc/fstab  
mount -a

Das System gibt dann die folgende Meldung aus, die ist allerdings nur eine Warnung:

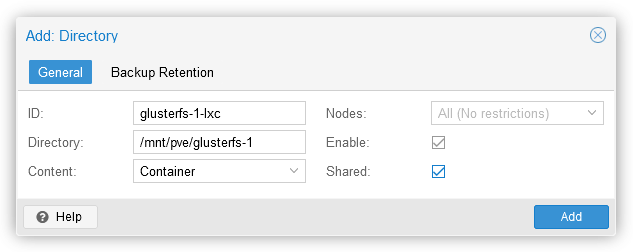
/sbin/mount.glusterfs: according to mtab, GlusterFS is already mounted on /mnt/pve/glusterfs-1

Über **Datacenter > Storage > Add** auf **einem** der Nodes einen neuen Storage vom Typ „Directory“ anlegen:



Die ID des neuen Directorys wird zum Namem des Storages. Unter Directory wird der in der /etc/fstab angebenen Pfad eingetragen.

Das Flag „Shared“ **muss manuell** aktiviert werden!



Beim Erstellen von LXCs kann nun **glusterfs-1-lxc** als Storage ausgewählt werden (siehe auch Kapitel „HA Konfiguration“).

##### Teil 8 - Optional: 2 Bricks, 1 Arbiter

Wenn als alternative oder Zwischenschritt zu einem dritten Node ein Raspberry Pi als qdevice eingesetzt wird (siehe Kapitel „Optional: ein Raspberry Pi als qdevice“), muss auch für das GlusterFS-Cluster ein weiter Node definiert werden. Dafür ist der Arbiter vorgesehen. Dieser kann ebenfalls auf dem Raspberry Pi welcher das qdevice für Proxmox darstellt, installiert werden. Auf diesem werden die Meta-Daten des GlusterFS bereitgehalten. Da dies ein sehr IO lastiger Prozess ist, sollte dafür ein extra USB-Stick verwendet werden, um die System SD-Karte zu schonen.

Details zum Arbiter & GlusterFS unter:

* <https://docs.gluster.org/en/v3/Administrator%20Guide/arbiter-volumes-and-quorum/#how-arbiter-works>
* <https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.4/html/administration_guide/creating_arbitrated_replicated_volumes>

###### Upgrade Raspbian 10 auf 11

Das aktuelle Rasbian Image setzt noch auf Debian 10 auf und sollte manuell auf Debian 11 aktualisiert werden. Denn wie in Teil 1 des GlusterFS Kapitels schon erwähnt, wird bei Debian 10 eine veraltete Version von GlusterFS mitgeliefert. Durch das Upgrade auf Debian 11 wird auch GlusterFS auf die Version 9.2 angehoben.

Der schnellste Weg Rasbian 10 auf 11 zu aktualisieren ist das in diesem Thread verlinkte Bash-Skript: <https://www.raspberrypi.org/forums/viewtopic.php?f=29&t=317888>

**Wichtig: Vor den ausführen von fremden Skripten, diese immer lesen und verstehen! Alternativ die notwendigen Schritte manuell durchführen.**

Nach dem Upgrade unbedingt das System durchstarten.

###### USB-Stick vorbereiten

Als erster Schritt, wird für den GlusterFS Arbiter ein USB-Stick eingerichtet.

Dazu mit

lsblk

prüfen, wie der Name der Partition des USB-Sticks lautet. Hier im Bespiel ist das „sda1“.

Danach wird das Dateisystem erstellt und die Partition eingehängt:

mkfs -t xfs -f -i size=512 -n size=8192 -L GlusterFS /dev/**sda1**

mkdir -p /data/glusterfs

mount -t xfs /dev/**sda1** /data/glusterfs

Um den USB-Stick dauerhaft im System einzuhängen, sucht man sich die UUID der Partition heraus und erweitert die /etc/fstab um folgenden Eintrag:

ls -l /dev/disk/by-uuid/\* | grep **sda1**

umount /data/glusterfs

echo "/dev/disk/by-uuid/[UUID] /data/glusterfs xfs defaults 0 0" >> /etc/fstab

mount -a

###### GlusterFS installieren

Nachdem einhängen des USB-Stick wird GlusterFS installiert und das System neu gestartet:

apt install glusterfs-server glusterfs-client -y

systemctl enable glusterd

reboot

###### Den Arbiter dem GlusterFS-Cluster hinzufügen

Nachdem Neustart wird, der Raspberry Pi dem GlusterFS-Cluster hinzugefügt und das Arbiter Volume erstellt:

gluster peer probe [IP-Raspberry]

gluster volume add-brick glusterfs-1-volume replica 3 arbiter 1 [IP-Raspberry]:/data/glusterfs force

Danach sollte gstatus -ab folgendes Ausgeben:

Cluster:

Status: Healthy GlusterFS: 9.2  
 Nodes: 3/3 Volumes: 1/1

Volumes:

glusterfs-1-volume  
 Replicate Started (UP) - 3/3 Bricks Up - (Arbiter Volume)  
 Capacity: (26.99% used) 126.00 GiB/466.00 GiB (used/total)   
 Bricks:  
 Distribute Group 1:  
 192.168.1.50:/data/glusterfs (Online)  
 192.168.1.51:/data/glusterfs (Online)  
 192.168.1.40:/data/glusterfs (Online)

##### Teil 9 – Performance Tests

<https://godatadriven.com/blog/building-a-raspberry-pi-storage-cluster-to-run-big-data-tools-at-home/>

###### Scrheiben

Maximale Lesegeschwindigkeit von /dev/urandom ermitteln:

dd if=/dev/urandom of=/dev/null count=65536 bs=1024

67108864 bytes (67 MB, 64 MiB) copied, 2.71787 s, 24.7 MB/s

Test mit einer Block-size von einem 1kB

dd if=/dev/urandom of=/mnt/pve/glusterfs-1/perftest.rnd count=65536 bs=1024

67108864 bytes (67 MB, 64 MiB) copied, 9.6823 s, 6.9 MB/s

Test mit einer Block-size von einem 128kB

dd if=/dev/urandom of=/mnt/pve/glusterfs-1/perftest.rnd count=512 bs=131072

67108864 bytes (67 MB, 64 MiB) copied, 2.82918 s, 23.7 MB/s

###### Lesen

Erste Ausführung = kein Cache:

dd if=/mnt/pve/glusterfs-1/perftest.rnd of=/dev/null bs=131072

67108864 bytes (67 MB, 64 MiB) copied, 0.581716 s, 115 MB/s

Zweite Ausführung = Cache aktiv:

dd if=/mnt/pve/glusterfs-1/perftest.rnd of=/dev/null bs=131072

67108864 bytes (67 MB, 64 MiB) copied, 0.0122723 s, 5.5 GB/s

###### Aufräumen

Am Ende die erstellte Datei wieder löschen:

rm /mnt/pve/glusterfs-1/perftest.rnd

##### Befehlsübersicht

|  |  |
| --- | --- |
| Befehl | Beschreibung |
| gstatus | Zeigt den Status des GlusterFS |
| gstatus -a -b | -a = all -b = list bricks |
| gstatus -a -o json | Ausgabe als json. Evtl. interessant für die Visualisierung über ioBroker. |
| gluster volume status glusterfs-1-volume detail | Detaillierte Anzeige des GlusterFS Volumes |
| gluster volume status | Zeigt den Status der Volumes |
| gluster peer status | Zeigt den Status des peerings. |
| gluster volume heal glusterfs-1-volume | Den Heal Prozess manuell anstoßen |
| gluster volume heal glusterfs-1-volume info | Zeigt den Heal Status an |
| gluster volume heal glusterfs-1-volume info split-brain | Zeigt den Heal Status bzgl. einer split-brain Situation an |

#### Container erstellen

Wie im letzten Abschnitt des Kapitels „Container & VM“ erwähnt, werden nun die Container für das weitere Vorgehen erstellt. Beim Erstellen der Container darauf achten, dass diese auf dem GlusterFS-Storage erstellt, werden:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Name | IP | OS |
| 100 | pve-test-iobroker | 192.168.1.70 | Debian 10 |
| 101 | pve-test-redis-node-1 | 192.168.1.71 | Debian 10 |
| 102 | pve-test-redis-node-2 | 192.168.1.72 | Debian 10 |
| 103 | pve-test-redis-node-3 | 192.168.1.73 | Debian 10 |

#### Hardware Watchdog aktivieren

Das Aktivieren des Hardware-Watchdogs hat den Vorteil, das auch wenn die Software (hier Proxmox) nicht mehr reagiert, eine Reaktion über die Hardware ausgeführt werden kann (z.B. einen Neustart des Systems).

Um den Hardware Watchdog auf einem Intel NUC zu aktivieren, muss in /etc/default/pve-ha-manager das entsprechende Modul zum Laden eingetragen werden. Wird eine andere Hardware eingesetzt muss dafür das passende Modul verwendet werden.

WATCHDOG\_MODULE=iTCO\_wdt

Danach das System neu starten und mit cat /var/log/syslog | grep wdt überprüfen, ob das Watchdog-Modul erfolgreich geladen werden konnte:

Sep 6 07:04:05 pve02 watchdog-mux[648]: Loading watchdog module 'iTCO\_wdt'  
Sep 6 07:04:05 pve02 kernel: [ 4.838549] iTCO\_wdt iTCO\_wdt: Found a Intel PCH TCO device (Version=6, TCOBASE=0x0400)  
Sep 6 07:04:05 pve02 kernel: [ 4.839386] iTCO\_wdt iTCO\_wdt: initialized. heartbeat=30 sec (nowayout=0)  
Sep 6 07:04:05 pve02 watchdog-mux[648]: Watchdog driver 'iTCO\_wdt', version 0

Proxmox bringt seinen eigenen Watchdog-Daemon für das Überwachen des HA-Clusters mit, den watchdog-mux.service. Dieser wird automatisch gestartet, wenn HA konfiguriert ist.

Überprüfen lässt sich dies mit

systemctl status watchdog-mux.service

watchdog-mux.service - Proxmox VE watchdog multiplexer  
 Loaded: loaded (/lib/systemd/system/watchdog-mux.service; static)  
 Active: active (running) since Sat 2021-09-11 17:26:02 CEST; 14h ago  
 Main PID: 661 (watchdog-mux)  
 Tasks: 1 (limit: 38297)  
 Memory: 240.0K  
 CPU: 1.819s  
 CGroup: /system.slice/watchdog-mux.service  
 └─661 /usr/sbin/watchdog-mux  
Sep 11 17:26:02 pve01 systemd[1]: Started Proxmox VE watchdog multiplexer.  
Sep 11 17:26:02 pve01 watchdog-mux[661]: Loading watchdog module 'iTCO\_wdt'  
Sep 11 17:26:02 pve01 watchdog-mux[661]: Watchdog driver 'iTCO\_wdt', version 0

Somit ist der Hardware-Watchdog installiert und aktiv.

Der Watchdog kann mit

systemctl stop watchdog-mux.service && echo "1" > /dev/watchdog

getestet werden. Nach einer kurzen Zeit sollte das System einen reboot durchführen.

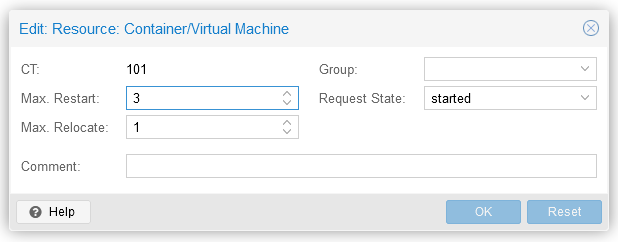
#### HA aktivieren

Auf einem der Proxmox-Nodes über **Datacenter > HA > Add** HA für die Container aktivieren:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Dazu unter **VM** die ID des LXCs oder der VM auswählen und die Max. Anzahl der Start-Versuche auf 3 setzen:



Danach ist HA für diesen Container aktiviert. Der LXC aus dem Beispiel liegt auf Node-1. Geht Node-1 offline, wird der LXC nach einer kurzen Zeit automatisch auf Node-2/-3 verschoben:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Über **Datacenter > HA > Groups** kann die Priorität für die einzelnen Nodes definiert werden. Eine größere Zahl entspricht einer höheren Priorität. Proxmox wird dann versuchen die VM/CT auf den Node mit der höchsten Priorität in einer Ausfallsituation zu migrieren.

Mehr Details dazu findet ihr hier: <https://pve.proxmox.com/wiki/High_Availability#_configuration>

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Über das Flag **nofailback** kann gesteuert werden, ob auf den ausgefallenen Node, falls er wieder Verfügbar ist, der CT zurück migriert wird.

#### HA Testen

Um HA nun zu testen, gibt es unterschiedliche Möglichkeiten:

* Netzwerk Kabel ziehen
* Strom-Kabel ziehen
* Einen Node manuell herunterfahren
* Den Corosync-Dienst beenden: systemctl stop corosync.service

In der Folge sollten nun VMs und LXCs die unter HA laufen auf den noch aktiven Node verschoben werden. Dies lässt sich gut im Syslog auf dem aktiven Node beobachten:

tail -f /var/log/syslog

#### Fehlerbehandlung

##### HA: CT oder VM hängt im Status „deleting“

Auf allen Nodes den HA-Manager stoppen:

systemctl stop pve-ha-crm

Auf einem Node die HA-Konfiguration löschen:

rm /etc/pve/ha/manager\_status

Auf allen Nodes den HA-Manager wieder starten:

systemctl start pve-ha-crm

<https://forum.proxmox.com/threads/cannot-delete-ha-resources-since-pve-5-to-6-update.58151/#post-269561>

##### Firefox: SEC\_ERROR\_REUSED\_ISSUER\_AND\_SERIAL

Nach dem Beitreten zu einem Cluster, kann es beim Benutzen von Firefox dazu kommen, dass man die Web-Oberflächen von Node-2 und Node-3 nicht mehr erreichen kann. Dies liegt an ungültig gewordenen Zertifikaten, welche im Firefox Profil für diese Adressen hinterlegt worden sind. Diese können über

**Einstellungen > Datenschutz & Sicherheit > Zertifikate > Zertifikate anzeigen > Server**

entfernt werden. Danach Firefox beenden und im Ordner des Firefox-Profils die Dateien

cert9.db  
cert\_override.txt

löschen.

Danach Firefox neu Starten und der Zugriff sollte wieder funktionieren.

#### Weiterführendes

##### GlusterFS Release Notes

In Teil 1 der GlusterFS-Anleitung wird dieses auf eine aktuelle Version aktualisiert. Mit den Release-Notes kann bei Bedarf die Version in diesem Schritt angepasst werden. Dazu die entsprechenden URLs aus den Release-Notes kopieren und entsprechend ersetzen.

<https://docs.gluster.org/en/latest/release-notes/>

##### GlusterFS Upgrade Guide v5 > v8

<https://docs.gluster.org/en/latest/Upgrade-Guide/upgrade-to-8/>

##### GlusterFS Volume Optionen

<https://access.redhat.com/documentation/en-us/red_hat_gluster_storage/3.1/html/administration_guide/chap-managing_red_hat_storage_volumes#Configuring_Volume_Options>

##### Proxmox Roadmap

<https://pve.proxmox.com/wiki/Roadmap>

##### Detaillierte Beschreibung Cluster erstellen

<https://www.informaticar.net/how-to-setup-proxmox-cluster-ha/>

##### Zusätzliches Brick hinzufügen

<https://www.cyberciti.biz/faq/howto-add-new-brick-to-existing-glusterfs-replicated-volume>

##### Proxmox-Forum Beitrag Installation GlusterFS

<https://forum.proxmox.com/threads/create-a-proxmox-6-2-cluster-glusterfs-storage.71971/>

##### Proxmox-Forum LXC auf GlusterFS Storage

<https://forum.proxmox.com/threads/container-on-gluster-volume-not-possible.40889/>

### Redis

#### Vorrausetzungen

3 Linux-Container (LXC) für die Redis-Server.

1. pve-test-redis-node-1 (192.168.1.71)
2. pve-test-redis-node-2 (192.168.1.72)
3. pve-test-redis-node-3 (192.168.1.73)

Siehe auch Kapitel „Proxmox - Container und VMs“, letzter Abschnitt.

#### Installation & Konfiguration redis-server

Am Ende dieses Abschnittes existieren 3 Redis Nodes welche sich untereinander replizieren.

##### Auf allen nodes

apt install redis-server -y  
systemctl enable redis-server  
systemctl stop redis-server

Zuerst eine Sicherheitskopie der **redis.conf** anlegen:

cp /etc/redis/redis.conf /etc/redis/redis.conf.org

*Optional: Kommentare aus der* ***redis.conf*** *entfernen:*

grep -o '^[^#]\*' /etc/redis/redis.conf > /etc/redis/redis.conf.no-comments  
cp /etc/redis/redis.conf.no-comments /etc/redis/redis.conf  
rm /etc/redis/redis.conf.no-comments

##### Auf pve-test-redis-node-1

Die **redis.conf** wie folgt anpassen:

bind 0.0.0.0  
protected-mode no

*Alternativ, Anpassen der* ***redis.conf*** *mit sed:*

sed -i "s|bind 127.0.0.1 ::1|bind 0.0.0.0|g" /etc/redis/redis.conf  
sed -i "s|protected-mode yes|protected-mode no|g" /etc/redis/redis.conf

##### Auf pve-test-redis-node-2/3

Die **redis.conf** wie folgt anpassen:

bind 0.0.0.0  
protected-mode noreplicaof 192.168.1.71 6379

*Alternativ, Anpassen der* ***redis.conf*** *mit sed:*

sed -i "s|bind 127.0.0.1 ::1|bind 0.0.0.0|g" /etc/redis/redis.conf  
sed -i "s|protected-mode yes|protected-mode no|g" /etc/redis/redis.conf  
echo "replicaof 192.168.1.71 6379" >> /etc/redis/redis.conf

##### Auf allen nodes

Den Dienst **redis-server** wieder starten:

systemctl start redis-server

##### Überprüfen der Installation

redis-cli info replication

Ausgabe auf pve-test-redis-node-1:

# Replication  
role:master  
connected\_slaves:2  
slave0:ip=192.168.1.72,port=6379,state=online,offset=70,lag=1  
slave1:ip=192.168.1.73,port=6379,state=online,offset=70,lag=1  
master\_replid:e53e73154cf03de4d8879eb2b4a7fdd9ea92bf98  
master\_replid2:0000000000000000000000000000000000000000  
master\_repl\_offset:70  
second\_repl\_offset:-1  
repl\_backlog\_active:1  
repl\_backlog\_size:1048576  
repl\_backlog\_first\_byte\_offset:1  
repl\_backlog\_histlen:7

Ausgabe auf pve-test-redis-node-2 & pve-test-redis-node-3:

# Replication  
role:slave  
master\_host:192.168.1.71  
master\_port:6379  
master\_link\_status:up  
master\_last\_io\_seconds\_ago:3  
master\_sync\_in\_progress:0  
slave\_repl\_offset:196  
slave\_priority:100  
slave\_read\_only:1  
connected\_slaves:0  
master\_replid:e53e73154cf03de4d8879eb2b4a7fdd9ea92bf98  
master\_replid2:0000000000000000000000000000000000000000  
master\_repl\_offset:196  
second\_repl\_offset:-1  
repl\_backlog\_active:1  
repl\_backlog\_size:1048576  
repl\_backlog\_first\_byte\_offset:1  
repl\_backlog\_histlen:196

##### Test der Replikation

Auf dem aktuellen Master-Node, also initial der pve-test-redis-node-1 einen neuen Datensatz erstellen:

redis-cli set db-test test123

Auf einem der anderen Nodes den Wert abfragen:

redis-cli get db-test

“test123”

Auf auf dem aktuellen Master (im Beispiel der pve-test-redis-node-1) kann dieser Datensatz mit

redis-cli del db-test

wieder gelöscht werden.

##### Optional: Getrennte Redis Prozesse für file- und objekt-DB

*“Am einfachsten ist es natürlich, States und Objekte zusammen in einem Redis-Prozess speichern zu lassen. Dies bedeutet allerdings auch dass nur alle Daten zusammen gesichert werden können. Bei der ioBroker File-DB waren States, Objekte und Files getrennt und konnten so selektiv gesichert werden. Auch die Schreiblast ist, wenn alles in einem Redis gespeichert ist, höher, da die Datenbank größer ist. – Apollon77 (*[*https://forum.iobroker.net/topic/26327/redis-in-iobroker-%C3%BCberblick/2*](https://forum.iobroker.net/topic/26327/redis-in-iobroker-%C3%BCberblick/2)*)*

Dazu muss auf **jedem** der 3 Redis-Nodes, ein zusätzlicher Redis-Prozess konfiguriert werden.

Die folgenden Schritte sind dafür erforderlich:

Eine neue Ordner-Struktur für die Daten der zweiten Redis Instanz erstellen:

install -o redis -g redis -d /var/lib/redis2

Die bestehende **redis.conf** kopieren:

cp -p /etc/redis/redis.conf /etc/redis/redis2.conf

Die **redis2.conf** anpassen:

port 6380  
pidfile /var/run/redis/redis-server2.pid  
logfile /var/log/redis/redis-server2.log  
dir /var/lib/redis2

Auf den Slave-Nodes in der **redis2.conf** auf die Zeile

replicaof 192.168.1.71 6380

achten!

*Alternativ, Anpassen der* ***redis2.conf*** *mit sed:*

sed -i "s|port 6379|port 6380|g" /etc/redis/redis2.conf  
sed -i "s|pidfile /var/run/redis/redis-server.pid|pidfile /var/run/redis/redis-server2.pid|g" /etc/redis/redis2.conf  
sed -i "s|logfile /var/log/redis/redis-server.log|logfile /var/log/redis/redis-server2.log|g" /etc/redis/redis2.conf  
sed -i "s|dir /var/lib/redis|dir /var/lib/redis2|g" /etc/redis/redis2.conf  
**# nur auf den slave-nodes**  
sed -i "s|replicaof 192.168.1.71 6379|replicaof 192.168.1.71 6380|g" /etc/redis/redis2.conf

Das systemd-Unit-File für die zweite Instanz erstellen, …

cp /lib/systemd/system/redis-server.service /lib/systemd/system/redis-server2.service

… und anpassen:

ExecStart=/usr/bin/redis-server /etc/redis/redis2.conf  
PIDFile=/run/redis/redis-server2.pid  
ReadWriteDirectories=-/var/lib/redis2  
Alias=redis2.service

*Alternativ, wieder die Anpassung mit sed:*

sed -i "s|ExecStart=/usr/bin/redis-server /etc/redis/redis.conf|ExecStart=/usr/bin/redis-server /etc/redis/redis2.conf|g" /lib/systemd/system/redis-server2.service  
sed -i "s|PIDFile=/run/redis/redis-server.pid|PIDFile=/run/redis/redis-server2.pid|g" /lib/systemd/system/redis-server2.service  
sed -i "s|ReadWriteDirectories=-/var/lib/redis|ReadWriteDirectories=-/var/lib/redis2|g" /lib/systemd/system/redis-server2.service  
sed -i "s|Alias=redis.service|Alias=redis2.service|g" /lib/systemd/system/redis-server2.service

Danach das Service-File aktivieren und das System durchstarten:

systemctl enable redis-server2.service  
reboot

Mit

ps -ef | grep redis

kann überprüft werden, ob der zweite Prozess läuft:

redis 154 1 0 10:22 ? 00:00:00 /usr/bin/redis-server 0.0.0.0:6380  
redis 155 1 0 10:22 ? 00:00:00 /usr/bin/redis-server 0.0.0.0:6379

Alternativ auch mit

systemctl status redis-server  
systemctl status redis-server2

Active: active (running) since …

#### Installation & Konfiguration redis-sentinel

##### Auf allen nodes

apt install redis-sentinel -y  
systemctl enable redis-sentinel  
systemctl stop redis-sentinel

Sicherheitskopie der **sentinel.conf** anlegen:

cp /etc/redis/sentinel.conf /etc/redis/sentinel.conf.org

*Optional: Kommentare aus der* ***sentinel.conf*** *entfernen:*

grep -o '^[^#]\*' /etc/redis/sentinel.conf > /etc/redis/sentinel.conf.no-comments  
cp /etc/redis/sentinel.conf.no-comments /etc/redis/sentinel.conf  
rm /etc/redis/sentinel.conf.no-comments

Die **sentinel.conf** anpassen:

bind 0.0.0.0  
sentinel failover-timeout mymaster 20000  
sentinel monitor mymaster 192.168.1.71 6379 2

*Optional, Anpassung mit sed und echo:*

sed -i "s|bind 127.0.0.1 ::1|bind 0.0.0.0|g" /etc/redis/sentinel.conf  
sed -i "s|sentinel monitor mymaster 127.0.0.1 6379 2|sentinel monitor mymaster 192.168.1.71 6379 2|g" /etc/redis/sentinel.conf  
echo "sentinel failover-timeout mymaster 20000" >> /etc/redis/sentinel.conf

Beim Konfigurieren der **sentinel.conf** sicherstellen, dass die **myid** je Host unterschiedlich ist.

Der failover-timeout von 20 Sekunden (20000ms) sollte so gewählt sein, dass ein normaler Reboot die Failover-Situation nicht auslöst. Bei einem LXC dürften die 20 Sekunden gut passen.

Der Name des Masters lautet in der default Konfiguration: **mymaster**. Dieser wird ggf. bei Tests und Fehlersuche noch benötigt. Siehe auch Kapitel Befehlsübersicht.

Danach den Dienst **redis-sentinel** wieder starten:

systemctl start redis-sentinel

##### Überprüfen der Installation

systemctl status redis-sentinel

redis-cli -p 26379 info sentinel

Ausgabe auf pve-test-redis-node-1:

# Sentinel  
sentinel\_masters:1  
sentinel\_tilt:0  
sentinel\_running\_scripts:0  
sentinel\_scripts\_queue\_length:0  
sentinel\_simulate\_failure\_flags:0  
master0:name=mymaster,status=ok,address=192.168.1.71:6379,slaves=2,**sentinels=3**

Die Anzahl der Sentinels in der letzten Zeile muss der Anzahl der redis-sentinel Instanzen entsprechen – im Beispiel hier, 3 Stück.

###### Failover-Test

Auf pve-test-redis-node-2 oder pve-test-redis-node-3 das redis-sentinel.log aufrufen:

tail -f /var/log/redis/redis-sentinel.log

Dann den aktuellen Redis-Master LXC (pve-test-redis-node-1) stoppen und das Log beobachten. Darin sollte nach rund 20 Sekunden (siehe sentinel.conf failover-timeout) eine Neuwahl eines Masters zu beobachten sein. Alternativ zum Stoppen des LXCs kann auch mittels redis-cli debug sleep 60 der Redis-Prozess pausiert werden.

166:X 13 Jul 2021 19:45:16.005 # +sdown master mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.005 # +sdown sentinel 98ebf650b75f69796b54610aa4fca802d86d555e 192.168.1.71 26379 @ mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.076 # +odown master mymaster 192.168.1.71 6379 #quorum 2/2  
166:X 13 Jul 2021 19:45:16.077 # +new-epoch 1  
166:X 13 Jul 2021 19:45:16.077 # +try-failover master mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.112 # +vote-for-leader 949af74096102af17e79ca75fc0294d78eb0bdae 1  
166:X 13 Jul 2021 19:45:16.184 # 09a2fa6d613acfa3527de0c3ab0fbc2ef1efa2e1 voted for 949af74096102af17e79ca75fc0294d78eb0bdae 1  
166:X 13 Jul 2021 19:45:16.189 # +elected-leader master mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.189 # +failover-state-select-slave master mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.260 # +selected-slave slave 192.168.1.72:6379 192.168.1.72 6379 @ mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.260 \* +failover-state-send-slaveof-noone slave 192.168.1.72:6379 192.168.1.72 6379 @ mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.316 \* +failover-state-wait-promotion slave 192.168.1.72:6379 192.168.1.72 6379 @ mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.482 # +promoted-slave slave 192.168.1.72:6379 192.168.1.72 6379 @ mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.482 # +failover-state-reconf-slaves master mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.555 \* +slave-reconf-sent slave 192.168.1.73:6379 192.168.1.73 6379 @ mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.846 \* +slave-reconf-inprog slave 192.168.1.73:6379 192.168.1.73 6379 @ mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.846 \* +slave-reconf-done slave 192.168.1.73:6379 192.168.1.73 6379 @ mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.921 # +failover-end master mymaster 192.168.1.71 6379  
166:X 13 Jul 2021 19:45:16.921 # +switch-master mymaster 192.168.1.71 6379 192.168.1.72 6379  
166:X 13 Jul 2021 19:45:16.921 \* +slave slave 192.168.1.73:6379 192.168.1.73 6379 @ mymaster 192.168.1.72 6379  
166:X 13 Jul 2021 19:45:16.921 \* +slave slave 192.168.1.71:6379 192.168.1.71 6379 @ mymaster 192.168.1.72 6379  
166:X 13 Jul 2021 19:45:46.997 # +sdown slave 192.168.1.71:6379 192.168.1.71 6379 @ mymaster 192.168.1.72 6379

##### Optional: sentinel für mehrere Redis-Prozesse konfigurieren

Wenn im vorherigen Kapitel mehrere Redis-Prozesse konfiguriert wurden, muss auch der Redis-Sentinel entsprechend angepasst werden. Allerdings braucht es dazu keinen eigenen redis-sentinel Prozess. Es reicht aus, die Konfiguration in der **sentinel.conf, auf jedem Node** zu verdoppeln.

Zuerst Redis-Sentinel Prozess stoppen:

systemctl stop redis-sentinel

Die Konfiguration anpassen:

sentinel monitor objects-master 192.168.1.71 6379 2  
sentinel failover-timeout objects-master 20000  
sentinel config-epoch objects-master 0  
sentinel leader-epoch objects-master 0

sentinel monitor states-master 192.168.1.71 6380 2  
sentinel failover-timeout states-master 20000  
sentinel config-epoch states-master 0  
sentinel leader-epoch states-master 0

Die **sentinel.conf** sollte dann so aussehen:

**Achtung bei Copy+Paste: Die myid muss eindeutig sein und sollte nicht überschrieben werden!**

bind 0.0.0.0  
port 26379  
daemonize yes  
pidfile "/var/run/sentinel/redis-sentinel.pid"  
logfile "/var/log/redis/redis-sentinel.log"  
dir "/var/lib/redis"  
protected-mode no  
  
sentinel myid 9ff532162c4b9f660839aae047c827533b8b1f59  
sentinel deny-scripts-reconfig yes

sentinel monitor objects-master 192.168.1.71 6379 2  
sentinel failover-timeout objects-master 20000  
sentinel config-epoch objects-master 0  
sentinel leader-epoch objects-master 0

sentinel monitor states-master 192.168.1.71 6380 2  
sentinel failover-timeout states-master 20000  
sentinel config-epoch states-master 0  
sentinel leader-epoch states-master 0

Danach den Redis-Sentinel Prozess wieder starten:

systemctl start redis-sentinel

Nach dem Start von redis-sentinel wird die sentinel.conf mit weiteren Daten des Sentinels-Cluster erweitert:

# Generated by CONFIG REWRITE  
…  
sentinel known-replica objects-master 192.168.1.72 6379  
sentinel known-replica objects-master 192.168.1.73 6379  
sentinel current-epoch 0  
…

Die Redis Instanz auf Port 6379 stellt somit die Datenbank für die ioBroker objects bereit, die Redis Instanz auf Port 6380 die Datenbank für die ioBroker states. Die Konfiguration des ioBrokers erfolgt im nächsten Kapitel.

Überprüfen, ob der Sentinel läuft:

systemctl status redis-sentinel

Active: active (running)

redis-cli -p 26379 info sentinel

# Sentinel  
sentinel\_masters:2  
sentinel\_tilt:0  
sentinel\_running\_scripts:0  
sentinel\_scripts\_queue\_length:0  
sentinel\_simulate\_failure\_flags:0  
master0:name=states-master,status=ok,address=192.168.1.71:6380,slaves=2,sentinels=3  
master1:name=objects-master,status=ok,address=192.168.1.71:6379,slaves=2,sentinels=3

#### Befehlsübersicht

Im Standard benutzt redis-cli den Port 6379. Sind, wie optional beschrieben, mehrere Redis-Instanzen je Container aufgesetzt, kann auf die zweite Instanz über den Port 6380 zugegriffen werden. Über den Port 26379 wird der redis-sentinel abgefragt.

|  |  |
| --- | --- |
| Befehl | Beschreibung |
| redis-cli info replication | Informationen des aktuellen Zustandes der Replikation |
| redis-cli -p 6379 debug sleep 60 | Pausiert den redis-Prozess |
| redis-cli -p 26379 info sentinel | Informationen zum redis-sentinel |
| redis-cli -p 26379 sentinel get-master-addr-by-name <master-name> | Zeigt den redis-master von <master-name>  Anstelle von <master-name>: objects-master / states-master bei getrennten Redis-Prozessen |
| redis-cli -p 26379 sentinel sentinels mymaster |  |
| redis-cli -p 26379 sentinel slaves mymaster |  |

#### Weiterführendes

**Redis Dokumentation**

<https://redis.io/topics/quickstart>

**Redis Sentinel Dokumentation**

<https://redis.io/topics/sentinel>

**Überblick Redis und ioBroker**

<https://forum.iobroker.net/topic/26327/redis-in-iobroker-%C3%BCberblick>

**Redis Replica / HA Howto (Achtung: Das weicht zum Teil von diesem hier ab!)**

[https://www.tecmint.com/setup-redis-replication-in-centos-8](https://www.tecmint.com/setup-redis-replication-in-centos-8/)  
[https://www.tecmint.com/setup-redis-high-availability-with-sentinel-in-centos-8](https://www.tecmint.com/setup-redis-high-availability-with-sentinel-in-centos-8/)

### ioBroker

#### Installation & Konfiguration

Den ioBroker mit

curl -sL https://iobroker.net/install.sh | sudo bash

installieren. Damit werden Node JS / npm und ioBroker installiert.

Danach den ioBroker wieder stoppen und das Setup aufrufen:

iobroker stop  
iobroker setup custom

Current configuration:  
- Objects database:  
 - Type: file  
 - Host/Unix Socket: 127.0.0.1  
 - Port: 9001  
- States database:  
 - Type: file  
 - Host/Unix Socket: 127.0.0.1  
 - Port: 9000  
- Data Directory: ../../iobroker-data/

**Wichtig: Es muss der Port des Sentinels angegeben werden: 26379**

Als Host werden die IPs der Redis-Nodes angegeben: 192.168.1.71,192.168.1.72,192.168.1.73 – somit erkennt der ioBroker auch das es sich um eine Sentinel-Konfiguration handelt und frägt den Namen der DBs ab.

Type of objects DB [(f)ile, (r)edis, ...], default [file]: r

When Objects and Files are stored in a Redis database please consider the following:

1. All data will be stored in RAM, make sure to have enough free RAM available!  
2. Make sure to check Redis persistence options to make sure a Redis problem will not cause data loss!  
3. The Redis persistence files can get big, make sure not to use an SD card to store them.

Host / Unix Socket of objects DB(redis), default[127.0.0.1]: 192.168.1.71,192.168.1.72,192.168.1.73  
Port of objects DB(redis), default[26379]: 26379  
Objects Redis Sentinel Master Name [mymaster]: objects-master  
Type of states DB [(f)file, (r)edis, ...], default [redis]: r  
Host / Unix Socket of states DB (redis), default[192.168.1.71,192.168.1.72,192.168.1.73]: <Enter>  
Port of states DB (redis), default[26379]: <Enter>  
States Redis Sentinel Master Name [objects-master]: states-master  
Host name of this machine [pve-test-iobroker]: <Enter>  
Please choose if this is a Master/single host (enter "m") or a Slave host (enter "S") you are about to edit. For Slave hosts the data migration will be skipped. [S/m]: m

Important: Using redis for the Objects database is only supported with js-controller 2.0 or higher!

When your system consists of multiple hosts please make sure to have js-controller 2.0 or higher installed on ALL hosts \*before\* continuing!

Important #2: If you already did the migration on an other host please \*do not\* migrate again! This can destroy your system!

Important #3: The process will migrate all files that were officially uploaded into the ioBroker system. If you have manually copied files into iobroker-data/files/... into own directories then these files will NOT be migrated! Make sure all files are in adapter directories inside the files directory!

Do you want to migrate objects and states from "file/file" to "redis/redis" [y/N]: y

Migrating the objects database will overwrite all objects! Are you sure that this is not a slave host and you want to migrate the data? [y/N]: y

Danach den ioBroker wieder starten:

iobroker start

#### GlusterFS Daten bereitstellen

Die Daten werden über den Webserver nginx für ioBroker bereitgestellt. Dieser wird mit

apt install nginx -y

installiert. Weitere Schritte in der Konfiguration sind nicht erforderlich.

Die GlusterFS Informationen werden mit Hilfe von gstatus in ein json exportiert und über den Webserver nginx unter /var/www/html/gstatus.json bereitgestellt.

Um die zyklische Ausführung kümmert sich systemd mit einem Timer:

**# /etc/systemd/system/gstatus-status-to-json.timer**  
  
[Unit]  
Description=gstatus to json timer

[Timer]  
OnBootSec=1min  
OnUnitInactiveSec=1min  
Persistent=true  
Unit=gstatus-status-to-json.service

[Install]  
WantedBy=timers.target

Der Timer startet jede Minute den folgenden Service:

**# /etc/systemd/system/gstatus-status-to-json.service**  
  
[Unit]  
Description=gstatus to json service

[Service]  
Type=oneshot  
User=root  
ExecStart=/bin/sh -c "/usr/local/bin/gstatus -o json > /var/www/html/gstatus.json"

Den Timer aktivieren:

systemctl daemon-reload  
systemctl enable gstatus-status-to-json.timer  
systemctl start gstatus-status-to-json.timer

Nun werden jede Minute unter /var/www/html/gstatus.json die aktuellen Daten aus gstatus abgelegt.

Im ioBroker ruft das folgende JavaScript, ebenfalls jede Minute, diese Daten ab und speichert sie in einer Objekt-Struktur:

// source: https://forum.iobroker.net/topic/13018/json-werte-in-datenpunkte/8?\_=1631436211662

function iter(name, obj) {  
 for(let i in obj) {  
 if(typeof obj[i] == 'object') iter(name + '.' + i, obj[i]);  
 else {  
 log(name + '.' + i + ': ' + obj[i]);  
 if(existsState(name + '.' + i)) setState(name + '.' + i, obj[i]);  
 else createState(name + '.' + i, obj[i]); // type: "mixed"  
 }  
 }  
}

function SendRequest(gfs\_node){  
 var options = {  
 url: 'http://'+gfs\_node+'/glusterfs.json'  
 };  
 request(options, function (error, response, body){  
 if (!error) iter('0\_userdata.0.GlusterFS', JSON.parse(body));  
 else console.error(error);  
 });  
}

function get\_gfs\_data\_from\_host(){  
 if (getState('ping.0.pve01').val == true) {  
 SendRequest('pve01');  
 } else if (getState('ping.0.pve02').val == true) {  
 SendRequest('pve02');  
 } else if (getState('ping.0.freya').val == true) {  
 SendRequest('freya');  
 } else {  
 console.error('no host from glusterfs-cluster available');  
 }  
}

// run at script start  
get\_gfs\_data\_from\_host

// run every 1min  
schedule('\* \* \* \* \*', get\_gfs\_data\_from\_host);

#### Smartmeter Anbindung über ser2net & socat

Da ich bisher keine funktionale Lösung für den Betrieb von USBIP in einem LXC gefunden habe, musste ich mich nach Alternativen umschauen. Die Nutzung von USBIP war in meinem Setup der einzige Grund, wieso ioBroker in einer VM laufen musste. Zur Auswahl standen 2 Alternativen, das Aufsetzen eines Multihost-Setups (Master im LXC und Slave auf einem RPI) oder die Kombination aus ser2net und socat. Da letztere die Ressourcenschonendere Variante bietet, wurde diese zuerst getestet. Die Installation ist sehr einfach, es benötigt auf dem Server (bei mir ein Raspberry PI an dem 2 IR-Köpfe zum Auslesen der Smartmeter angeschlossen sind) ser2net und auf dem Client (ein LXC mit ioBroker) socat. Beide jeweils installieren, auf dem Server die ser2net.conf anpassen und auf dem Client die systemd Dienste erstellen und starten.

##### Auf dem Server – ser2net

apt install ser2net -y

**# /etc/ser2net.conf**

192.168.1.41,9990:raw:0:/dev/ttyUSB0:9600 8DATABITS NONE 1STOPBIT  
192.168.1.41,9991:raw:0:/dev/ttyUSB1:9600 8DATABITS NONE 1STOPBIT

reboot

##### Auf dem Client (ioBroker) - socat

apt install socat -y

**# /etc/systemd/system/socat-loki-usb0.service**

[Unit]  
Description=SOCAT Listen Service - USB0  
After=network.target iobroker.service  
Wants=iobroker.service

[Service]  
Type=simple  
User=iobroker  
ExecStart=/bin/bash -c '/usr/bin/socat pty,link=/tmp/ttyCOM0 tcp:192.168.1.41:9990'  
Restart=on-failure

[Install]  
WantedBy=multi-user.target

**# /etc/systemd/system/socat-loki-usb1.service**

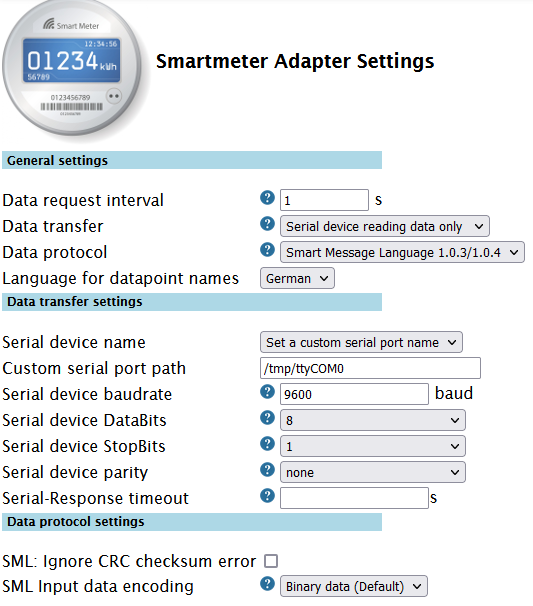
[Unit]  
Description=SOCAT Listen Service – USB1  
After=network.target iobroker.service  
Wants=iobroker.service

[Service]  
Type=simple  
User=iobroker  
ExecStart=/bin/bash -c '/usr/bin/socat pty,link=/tmp/ttyCOM1 tcp:192.168.1.41:9991'  
Restart=on-failure

[Install]  
WantedBy=multi-user.target

systemctl daemon-reload  
systemctl enable socat-loki-usb0.service  
systemctl enable socat-loki-usb1.service  
systemctl start socat-loki-usb0.service  
systemctl start socat-loki-usb1.service

##### Smartmeter Adapter Konfiguration



## Archiv

### USBIP

Offene gebliebenen Punkte aus der TODO-Liste:

* USBIP: Anstelle von Logfile-Eintrag, /dev/ttyUSB\* prüfen  
  [ -e /dev/ttyUSB0 ] | [ -e /dev/ttyUSB1 ] && echo 'y' || 'n'

Um USB-Geräte unabhängig vom Proxmox-Cluster zu betreiben bietet sich USBIP an. Wie der Name schon verrät, werden dadurch die USB-Daten per TCP/IP übertragen. In meinem Setup läuft der USBIP-Server auf einem Raspberry PI in der Nähe der Smartmeter und gibt die Daten an den USBIP-Client, also das Proxmox-Cluster weiter.

#### Installation

Meine Systeme laufen unter Debian, daher erfolgt die Installation direkt über:

apt install usbip hwdata

Unter Ubuntu sollte dies über die linux-tools-generic erfolgen. Siehe auch <https://wiki.ubuntuusers.de/USBIP>.

#### Konfiguration

##### Server (Raspberry PI)

###### Kernel-Modul laden

modprobe usbip-host  
echo 'usbip\_host' >> /etc/modules

###### systemd konfigurieren

Das Unit-File für den USBIP-Daemon erstellen:

**# /lib/systemd/system/usbipd.service**[Unit]  
Description=usbipd  
After=network.target[Service]  
Type=forking  
ExecStart=/usr/sbin/usbipd -D[Install]  
WantedBy=multi-user.target

Danach aktivieren und starten:

systemctl daemon-reload  
systemctl enable usbipd  
systemctl start usbipd

Jetzt wird der Dienst für das Exportieren der USB-Geräte erstellt:

**# /lib/systemd/system/usbipd-bind@.service**[Unit]  
Description=usbipd binding %I  
After=network-online.target usbipd.service  
Wants=network-online.target  
Requires=usbipd.service

[Service]  
Type=simple  
ExecStart=/usr/sbin/usbip --log bind -b %i  
RemainAfterExit=yes  
ExecStop=/usr/sbin/usbip --log unbind -b %i  
Restart=on-failure

[Install]  
WantedBy=multi-user.target

Danach werden auch diese aktiviert und gestartet.

In meinem Beispiel erstelle ich Dienste für den Export der Bus-IDs 1-1.2 und 1-1.3:

systemctl daemon-reload  
systemctl enable usbipd-bind@1-1.2  
systemctl enable usbipd-bind@1-1.3  
systemctl start usbipd-bind@1-1.2  
systemctl start usbipd-bind@1-1.3

###### Sicherstellen der Funktion des Clients

Da der USBIP-Client seine Verbindung zum USBIP-Server verliert, sollte dieser neu gestartet werden, sichere ich die Verbindung mit den folgenden Skripten ab. Der Dienst **usbipd-restart-usbip-client.service** startet auf dem entfernten Host den Service USBIP-Client und die ioBroker Adapter smartmeter.0 und smartmeter.1 nach dem Bootvorgang des USBIP-Servers durch und verbindet somit wieder den Client mit dem Server.

Damit dies funktioniert, muss der Passwortlose SSH-Zugriff von USBIP-Server auf USBIP-Client eingerichtet sein und der Benutzer auf dem entfernten Host den Befehl service über sudo ausführen dürfen:

**# visudo**darkiop ALL=(ALL) NOPASSWD: /usr/sbin/service

Der Dienst wird mit Hilfe von systemd am Ende des Bootvorgangs gestartet.

Dazu wird zuerst neues systemd-Target auf dem USBIP-Server erstellt:

**# /etc/systemd/system/custom.target**[Unit]  
Description=do things after boot   
Requires=multi-user.target  
After=multi-user.target  
AllowIsolate=yes

Und danach das Service-File:

**# /etc/systemd/system/usbipd-restart-usbip-client.service**[Unit]  
Description=restart usbip-client iobroker   
After=multi-user.target

[Service]  
Type=simple  
ExecStart=ssh USER@USBIP-CLIENT bash -c "'sudo service usbip-client restart'"

[Install]  
WantedBy=custom.target

Zum Schluss noch das neue systemd-Target und Service aktivieren:

systemctl daemon-reload

mkdir /etc/systemd/system/custom.target.wants

systemctl enable usbipd-restart-usbip-client.service

systemctl set-default custom.target

reboot

Nun wird am Ende jedes Bootvorgangs der Service **usbipd-restart-usbip-client.service** über das neue Target custom.target aufgerufen und somit über SSH der Service usbip-client.service auf dem USBIP-Client neu gestartet. Dieser wird im nächsten Abschnitt auf dem USBIP-Client eingerichtet.

##### Client (ioBroker VM)

###### Kernel-Modul laden

modprobe vhci-hcd  
echo ' vhci-hcd' >> /etc/modules

###### systemd konfigurieren

**# /lib/systemd/system/usbip-client.service**

[Unit]  
Description=usbip-client  
After=network.target  
Before=iobroker.service

[Service]  
Type=oneshot  
RemainAfterExit=yes  
ExecStart=/bin/sh -c "usbip attach -r 192.168.1.41 -b $(usbip list -r 192.168.1.41 | grep '1-1.2' | cut -d: -f1 | sed -e 's/^[[:space:]]\*//'); usbip attach -r 192.168.1.41 -b $(usbip list -r 192.168.1.41 | grep '1-1.3' | cut -d: -f1 | sed -e 's/^[[:space:]]\*//');iob restart smartmeter.0; iob restart smartmeter.1"  
ExecStop=/bin/sh -c "/usr/sbin/usbip detach --port=$(/usr/sbin/usbip port | grep '<Port in Use>' | sed -E 's/^Port ([0-9][0-9]).\*/\1/');/usr/sbin/usbip detach --port=$(/usr/sbin/usbip port | grep '<Port in Use>' | sed -E 's/^Port ([0-9][0-9]).\*/\1/'); iob stop smartmeter.0; iob stop smartmeter.1"

[Install]  
WantedBy=multi-user.target

###### Überwachen des ioBroker Logfiles

Wird der USBIP-Server / Dienst neu gestartet oder gar das System neu gebootet, verliert der USBIP-Client die Verbindung zum Server. Um einen Reboot abzufangen wurde bereits auf dem Client der Dienst **usbipd-restart-usbip-client.service** installiert um nach einen Reboot den USB-Client durchzustarten. Für den Fall, das der USBIP-Server nicht neu gestartet, aber der USBIP-Client die Verbindung verloren hat, wird ein weiter Dienst auf dem USBIP-Client benötigt.

Der ioBroker Adapter Smartmeter meldet bei Verlust der Verbindung zum USBIP-Server

„No or too long answer from Serial Device after last request“

im Log. Diese Zeile kann man mit dem folgenden Dienst überwachen.

Der Dienst wird mit Hilfe von systemd am Ende des Bootvorgangs gestartet.

Dazu wird zuerst neues systemd-Target auf dem USBIP-Server erstellt:

**# /etc/systemd/system/custom.target**[Unit]  
Description=do things after boot   
Requires=multi-user.target  
After=multi-user.target  
AllowIsolate=yes

Und danach das Service-File:

**# /etc/systemd/system/usbip-monitor-iobroker-log.service**[Unit]  
Description=checks iobroker log and restart usbip server if necessary  
After=multi-user.target  
  
[Service]  
Type=simple  
ExecStart=/opt/iobroker/bin/usbip-monitor-log-and-restart-usbipd.sh  
ExecStop=killall usbip-monitor-log-and-restart-usbipd.sh  
  
[Install]  
WantedBy=custom.target

Zum Abschluss muss noch das Skript erstellt werden, welches das ioBroker überwacht und durch den eben angelegten Dienst gestartet wurde:

**# /opt/iobroker/bin/usbip-monitor-log-and-restart-usbipd.sh**

#!/bin/bash  
tail -fn0 /opt/iobroker/log/iobroker.current.log | \  
while read line ; do  
  echo "$line" | grep "No or too long answer from Serial Device after last request"  
  if [ $? = 0 ]; then  
    /usr/bin/ssh USER@USBIP-SERVER bash -c "'sudo /usr/sbin/reboot -f'"  
  fi  
done

Das Skript löst bei Erkennen des erwähnten Log-Eintrags einen Reboot des USBIP-Servers aus. Am Ende dessen Reboots wird durch den Dienst **usbipd-restart-usbip-client.service** auch der USBIP-Client neu gestartet und somit die Verbindung zwischen beiden wieder hergestellt.

Damit dies funktioniert, muss der Passwortlose SSH-Zugriff von USBIP-Client auf USBIP-Server eingerichtet sein und der Benutzer auf dem entfernten Host den Befehl reboot über sudo ausführen dürfen:

Zum Schluss noch das neue systemd-Target und Service aktivieren:

systemctl daemon-reload

mkdir /etc/systemd/system/custom.target.wants

systemctl enable usbip-monitor-iobroker-log.service

systemctl set-default custom.target

reboot

#### Befehle

##### Server

|  |  |
| --- | --- |
| Befehl | Beschreibung |
| usbipd -D | Startet den USBIP-Daemon (standalone) |
| usbip list -l | Zeigt die lokalen USB-Geräte an |
| usbip bind -b "<USBID>" | Exportiert das angegebene USB-Gerät |
| usbip unbind -b "<USBID>" | Nimmt den Export vom USB-Gerät zurück |

##### Client

|  |  |
| --- | --- |
| Befehl | Beschreibung |
| usbip list -r <IP> | Zeigt die exportierten USB-Geräte von <IP> an |
| usbip attach -r <IP> -b "<USBID>" | Hängt Gerät mit <USBID> von <IP> ein. |
| usbip detach --port<#> | Entfernt Gerät auf Port <#> (z.B. 0,1,…, siehe usbip port) |
| usbip port | Zeigt die importierten USB-Geräte an |

#### Weiterführendes

<https://wiki.ubuntuusers.de/USBIP/>

<https://www.florian-diesch.de/doc/linux/usb-over-ip-mit-usbipd.html>

https://wiki.archlinux.org/title/USB/IP