Self Hosted Jitsi Server - auf Debian 10.x aka. Buster und LattePanda

verwendete Quellen

Folgende Quellen liegen meiner Installationsanleitung zugrunde:

- LatterPanda Home
- Jitsi Home
- https://github.com/jitsi/jitsi-meet/blob/master/doc/quick-install.md
- https://github.com/jitsi/ice4j/blob/master/doc/configuration.md
- https://forum.golem.de/kommentare/opensource/homeoffice-videokonferenzen-auf-eigenen-servern-mit-jitsi-meet/bevor-sich-noch-wer-die-zaehne-ausbeisst/133384,5616554,5616554,read.html#msg-5616554
- https://community.jitsi.org/t/not-working-for-more-than-2-people-in-the-room/18821/18
- https://community.jitsi.org/t/ice4j-config-with-dynamic-ip-address/23102
- https://github.com/jitsi/jitsi-meet/blob/master/doc/manual-install.md
- https://www.kuketz-blog.de/kurzanleitung-jitsi-meet-videokonferenz-per-browser-oder-app/
- https://scheible.it/jitsi-meet-server-installation/
- https://decatec.de/home-server/jitsi-meet-videokonferenz-system-unter-ubuntu-server-mit-nginx/https://decatec.de/home-server/jitsi-meet-videokonferenz-system-unter-ubuntu-server-mit-nginx/]]

Router vorbereiten

Damit der eigene Jitsi Server von jedem außerhalb des eigenen (privaten) Netzwerkes errichbar ist müssen ein paar Ports im Router an den Jitsi Host weitergereicht werden (sog. port forwarding). Wie das beim jeweils eingesetzten Router eingestellt wird verrät die passende Dokumentation oder eine gute Suchmaschine 😉

Port	Protokoll	Bemerkung
80	TCP	http 1*)
443	TCP	https für Nginx Webserver
4443	TCP	alternativ https für VideoBridge
10000	UDP	für VideoBridge

1*) Der Port 80 (HTTP) ist wichtig für den automatischen Bezug eines LetsEncrypt Zertifikates, siehe nachfolgende Installtionsschritte.

Hardware Basis

Zum Betrieb eines 24/7 Videokonferenz-Servers bietet sich
z.B. ein aktueller Raspberry Pi 4 an. Er ist klein,
leistungsstark und verbraucht dabei sehr wenig Energie.
Auf dem Raspberry Pi kommt dafür eine ARM basierte CPU

von Broadcom zum Einsatz. Und da gehen die Probleme leider auch schon los, denn **das Jitsi Projekt stellt keine ARM Programmpakete in ihrem Repository bereit**.

Also ist der Einsatz einer Rechnerplattform auf der Basis einer X86 bzw X86-64 CPU zwingend erforderlich - leider 😟

Als Host verwende ich ein kleines Embedded Board namens LattePanda.
Es zeichnet sich ebenfalls durch seine geringe Größe und einen minimalen Energieverbrauch aus. Zusätzlich hat es 64 GB Flash Speicher on Board, so dass kein externes Speichermedium für das Betriebssystem notwendig sind.

Mein LattePanda in der Version 4 GB RAM & 64 GB FLASH

Debian 10.x aka. Buster auf dem LattePanda

Bootfähigen USB-Stick erstellen:

```
# Debian besorgen:
wget https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/debian-10.3.0-amd64-netinst.iso
# Bootfähigen USB-Stick erstellen:
sudo dd bs=4M if=/path/to/debian-10.3.0-amd64-netinst.iso of=/dev/sdx status=progress oflag=sync
```

LattePanda mit USB-Stick booten und Debian z.B mit grafischem Installer (im Expert-Mode) installieren. Im Anschluss das Basis-System booten und als Benutzer `root` anmelden. Dann die Paketquellen aktualisieren, notwendige Updates und ein paar nützliche Programmen installieren.

Paketquellen aktualisieren:

```
apt-get update
```

System aktualisieren:

```
apt-get upgrade
```

Fehlende Pakete installieren:

```
apt-get -y install mc aptitude sudo gnupg2 acpid iptraf iftop curl
```

Den eigene Benutzer der Gruppe `sudo` hinzufügen:

```
adduser <USERNMAE> sudo
```

Das Login als Root erlauben um weitere Konfigurationen und Installationen via SSH durchführen zu können:

```
mcedit /etc/ssh/sshd_config
# ändere "PermitRootLogin prohibit-password" in "PermitRootLogin yes"
# erlaube Login via Public Keys: "PubkeyAuthentication yes"
<c/ode>

Restart SSH Server:
<code bash>
/etc/init.d/ssh restart
```

Feste IP Adresse einstellen:

```
mcedit /etc/network/interfaces

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enx00e04c004797

#iface enx00e04c004797 inet dhcp

iface enx00e04c004797 inet static
    address 192.168.11.5
    broadcast 192.168.1.255
    netmask 255.255.255.0
    gateway 192.168.11.1
    dns-nameservers 192.168.11.1
```

Jitsi installieren

Zusätzliche Pakete nachinstallieren:

```
apt-get install apt-transport-https nginx
```

PGP Schlüssel zur Überprüfung der signierten Pakete einstallieren:

```
wget -qO - https://download.jitsi.org/jitsi-key.gpg.key | sudo apt-key add -
```

Jitsi Paketquelle hinzufügen:

```
echo "deb https://download.jitsi.org stable/" >>
/etc/apt/sources.list.d/jitsi-stable.list
```

Paketquellen aktualisieren:

```
apt-get update
```

Jitsi installieren, dabei wird ein FQDN benötigt. Dieser ist für einen reibungslosen Betrieb essentiell daher darf hier keine IP-Adresse angegeben werden.

```
apt-get -y install jitsi-meet
# Enter hostname (FQDN): --> <meine.eigene.domain.de>
# Wirklich wichtig!
# Ohne FQDN bekommt man kein LetsEncrypt Zertifikat (siehe unten) und ohne
Zertifikat läuft WebRTC nicht. Video/Audio Stream geht nur mit https!
```

Jitsi konfigurieren

Basiskonfiguration

FQDN in `hosts` Datei eintragen:

```
mcedit /etc/hosts
...
127.0.0.1   localhost <meine.eigene.domain.de>

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

Automatisch ein LetsEncrypt Zertifikat beziehen:

Ohne eine gültiges Zertifikat wird jeder moderne Browser vor dem Verbindungsaufbau eine Feghlermeldung bzgl. eines ungültigen Zertifikates zeigen.
Die Audio- und die Videoübertragung via webRTC sind dann ebenfalls nicht möglich. Self-signed Zertifikate sind lediglich für reine LAN-Varianten (quasi hausintern) ok.

```
/usr/share/jitsi-meet/scripts/install-letsencrypt-cert.sh
```

Jitsi Videobridge tweaks ;-)

Bandbreite sparen durch herabsetzten der standard Auflösung z.B auf 480 Pixel (Kommentarzeichen vor den Zeilen 116 bis 125 entfernen. Ebenfalls die sog. „Third Party Requests“ abschalten.

```
mcedit /etc/jitsi/meet/meet.example.com-config.js
109: resolution: 480,
...
120: ideal: 360,
121: max: 360,
122: min: 180
...
```

```
~330: disableThirdPartyRequests: true,
```

Eine weitere Konfig Datei anpassen um die Übertragung von Statistiken zu unterbinden:

```
mcedit /etc/jitsi/videobridge/sip-communicator.properties
...
org.jitsi.videobridge.ENABLE_STATISTICS=false
```

OUTDATED - DO NOT USE Jitsi Videobridge tweaks

Bis zum SW Stand Ende März waren folgende Änderungen bzw. Anpassungen in der Jitsi Konfig notwendig. Nach aktuellem Stand (08.04.2020) sind diese Änderungen nicht mehr nötig. Ich lasse sie hier aber der Vollständigkeit halber weiterhin stehen.

```
mcedit /etc/jitsi/videobridge/sip-communicator.properties
...
org.jitsi.videobridge.AUTHORIZED_SOURCE_REGEXP=focus@auth.<meine.eigene.doma
in.de>/.*
org.ice4j.ice.harvest.NAT_HARVESTER_LOCAL_ADDRESS=192.168.11.5
org.ice4j.ice.harvest.NAT_HARVESTER_PUBLIC_ADDRESS=<meine.eigene.domain.de>
#org.ice4j.ice.harvest.STUN_MAPPING_HARVESTER_ADDRESSES=stun.l.google.com:19
302 ,stun1.l.google.com:19302 ,stun2.l.google.com:19302
#org.ice4j.ice.harvest.STUN_MAPPING_HARVESTER_ADDRESSES=stun.1und1.de:3478,
stun.einsundeins.de:3478, stun.gmx.de:3478, stun.hosteurope.de:3478
org.ice4j.ice.harvest.STUN_MAPPING_HARVESTER_ADDRESSES=stun.t-online.de:3478
org.jitsi.videobridge.SINGLE_PORT_HARVESTER_PORT=10000
org.jitsi.videobridge.DISABLE_TCP_HARVESTER=true
org.ice4j.ipv6.DISABLED=true
```

Und diese Änderungen. Wobei als STUN-Server jeder beliebige frei verfügbare Server verwendet werden kann.

```
mcedit /etc/jitsi/meet/<meine.eigene.domain.de>-config.js
...
stunServers: [
            { urls: 'stun.t-online.de:3478' }
],
...
```

Dynamische IP vs. statische IP

Beim regulären Start der Jitsi Videobridge (`jitsi-videobridge2`) wird die öffentliche IP-Adresse via STUN ermittelt über die der Jitsi Server erreichbar ist.
Siehe Log unter `tail -n 1 -f /var/log/jitsi/jvb.log`:

```
2020-04-13 16:55:43.788 INFORMATION: [24]
org.ice4j.ice.harvest.StunMappingCandidateHarvester.discover: Discovered
public address <PUBLIC-IP>:55172/udp from STUN server 3.127.113.113:443/udp
```

```
using local address 192.168.11.5:0/udp
2020-04-13 16:55:43.804 INFORMATION: [23]
org.ice4j.ice.harvest.MappingCandidateHarvesters.initialize: Using
org.ice4j.ice.harvest.StunMappingCandidateHarvester, face=/192.168.11.5,
mask=/<PUBLIC-IP>
```

Beim Betrieb einer Jitsi-Instanz an einem (privaten) DSL-Anschluss, z.B. zu Hause, ändert sich jedoch die öffentliche IP bei jedem Verbindungsaufbau zum IPS. Davon bekommt der Jitsi-Server jedoch nichts mit. Die „alte“ IP wird von der Jitsi VideoBridge weiter verwendet und bei jedem Verbindungsaufbau durch einen Clients an diesen propagiert.
Das folgende Skript behebt dieses Problem indem es die aktuelle IP mit der zuletzt bekannten vergleicht und bei einer Abweichung das erneute Einlesen der Jitsi Konfiguration mittels `restart` des Dienstes durchführt.

check_public_ip.sh

```
#!/bin/bash
# Workaround to reload Jitsi VideoBridge configuration after
# unnoticed public IP address change when jitsi is self hosted
# and connected to the internet via private DSL connection.
# This happens approximately every 24 hours and is often forced
# by the ISP.
#
# Author: C. von Thuelen

wan_ip_logfile=/tmp/last_wan_ip.log
wan_ip_changelog=/tmp/wan_ip_change.log

# read last public IP from logfile:
if [ -e $wan_ip_logfile ]; then
   source $wan_ip_logfile
fi

# get actual public IP:
new_wan_ip=`curl -s ifconfig.me/ip`

# at first start "$lastip" is empty
if [ -z $last_wan_ip ]; then
   echo "last_wan_ip=\"$new_wan_ip\"" > $wan_ip_logfile
fi

# compare last and actual public IPs:
if [ "$new_wan_ip" != "$last_wan_ip" ]; then
# if not equal:
   # save new public IP to logfile
   echo "last_wan_ip=\"$new_wan_ip\"" > $wan_ip_logfile
   daytime=`date +%Y%m%d_-_%H:%M`
   echo "Last IP update: $daytime, IP changes from $last_wan_ip to
$new_wan_ip" >> $wan_ip_changelog
   # and reload jitsi videobridge
   /etc/init.d/jitsi-videobridge2 restart
```

```
else
# do nothing ;-)
   :
fi
```

Das neue Skript herunter laden, ausführbar machen und als CRON-Job alle 5 Minuten ausführen lassen - fertig!

```
sudo su
cd /usr/bin
wget -O check_public_ip.sh
"https://www.von-thuelen.de/doku.php/wiki/linux/videotelefonie/jitsi?do=expo
rt_code&codeblock=17" && chmod +x check_public_ip.sh
cd /
touch /var/spool/cron/crontabs/root
/usr/bin/crontab /var/spool/cron/crontabs/root
echo "*/5 * * * * /usr/bin/check_public_ip.sh" >>
/var/spool/cron/crontabs/root
```

Sysctl tweaks ;-)

Die folgenden Änderungen brachten leichte Performence Verbesserungen auf einem älteren Intel ATOM N510 Netbook.

```
cat /etc/sysctl.conf
# append:
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
net.ipv6.conf.ens160.disable_ipv6 = 1

# increase Linux TCP buffer limits
net.core.rmem_max = 2097152
net.core.wmem_max = 2097152

# increase Linux autotuning TCP buffer limits
# min, default, and max number of bytes to use
net.ipv4.tcp_rmem = 4096 87380 2097152
net.ipv4.tcp_wmem = 4096 65536 2097152
```

Jitsi neu starten

```
service nginx restart; service prosody restart; service jicofo restart;
service jitsi-videobridge2 restart
```

Jitsi deinstallieren

```
apt-get -y purge jigasi jitsi-meet jitsi-meet-web-config jitsi-meet-prosody
jitsi-meet-web jicofo jitsi-videobridge
```

Debugging

Chrome Session debugging

Über diesen Weg kann man kontrollieren, ob von Jitsi die richtige öffentliche IP an den Client propagiert wird:

URL: **`chrome://webrtc-internals`** in einem neuen Chrome Tab öffnen
dritter Reiter –> suche nach `setRemoteDescription`
`–> a=candidate:1 1 udp nnnnnnnn <irgendwas_IPV6> 100000 type host generation 0`
`–> a=candidate:2 1 udp nnnnnnnn <interne IP> 100000 type host generation 0`
`–> a=candidate:3 1 udp nnnnnnnn <`**`öffentliche IP`**`> 100000 type srflx raddr <`**`interne IP`**`> rport 10000 generation 0`

WebRTC

https://test.webrtc.org/