How i would install Ubuntu server 20.04 LTS with bridged interfaces?

work in progress

STUFF TO READ:

- https://wiki.ubuntu.com/IPv6
- http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html_single/Linux+IPv6-HOWTO.html

Szenario / User requirements:

This is the network how i like to have it:

- [X] are Hosts/Devices
- {X} are networks/clouds
- ==> are just bidirectional links/cable of some sort.
- Workstations are just any number of LAN connected devices/computers/hosts in your home or SOHO.
- LAN is the local private network. I dont want my ISP to sniff around here how know how many devices i have. I want it to use a private self defined IPv6 address range (or ULA) like **fd00:feed:dead:beef::/64** .
- LINUX SERVER is a plain Linux PC with TWO network interfaces.
- DMZ is the unsecure “transfer network” or “demilitarized zone” between my Linux server and the cable router to the ISP. Its not yet open internet, but “can” contain mostly exposed hosts and services for the internets. The ISP assigned the /64 IPv6 network/prefix **2a02:xxxx:xxxx:xxxx::/64** to me. And since i dont what them to dictate what addresses i am going to use in my networks, i just decided to use the assigned prefix with my own defined node address **2a02:xxxx:xxxx:xxxx:feed:dead:beef:20/64** for my Linux server on the DMZ side. **Note:** the 'dead beef' thing is just a hexadecimal number, not a word! it can be any hex number.
- “ISP router” in my case is a “Vodafone Station” a.k.a. ARRIS cable router in a DS lite (dslite dual stack lite) configuration. In my case ISP provides a shitty artificially crippled router without any real control over anything on your network. No IPv6 dhcp, no control over the “ipv6 firewall” or “port filtering”, nothing. You can disable the routers firewall temporarily for like 24h and then they re-enable it by force without your consent.
- “Internet” is any network beyond the cable router.

```
[workstations]==>{LAN}==>[LINUX SERVER]==>{DMZ}==>[ISP router]==>{Internet}
```

UBUNTU Server base installation:

1. Use UBUNTU Server 20.04 LTS ISO as DVD or USB Stick to boot from. Make sure you have a a single fast and free SSD for the /boot and root filesystem /.
    1. Use whole Disk (SSD), use default filesystem
    2. make sure /boot has no less then 1GB space
2. Use manual (fixed ip) network configuration for servers right from the beginning
3. install updates as soon as possible

```
sudo apt update && sudo apt upgrade -y && sudo reboot
```

4. **FOR VMs ONLY:** If this installation is a virtual machine on some hypervisor don't forget to install the “guest drivers” or “guest tools” matching your virtualisation host. So a VM on a VMWARE host needs **open-vm-tools** to be installed. On a ProxMox/QEMU/KVM host you usually install **qemu-guest-agent**. On Virtualbox it's called **virtualbox-guest-utils**:

```
# Pick one, only one!

apt install -y qemu-guest-agent # for proxmox qemu kvm VMs

# or
apt install -y virtualbox-guest-utils # for VMs on Virtualbox

# or
apt install -y open-vm-tools # i wouldn't use VMWARE. no, really!
```

5. Install generic useful tools early:

```
apt install -y ethtool bridge-utils net-tools vim nmap atop htop iftop
iotop lvm2 tmux screenfetch firefox
```

6. Edit bootloader “grub”, disable graphical bootloader modes, make it human readable:

```
cat << EOF >> /etc/default/grub
GRUB_TIMEOUT_STYLE=menu
GRUB_TERMINAL=console
GRUB_TIMEOUT=3
GRUB_CMDLINE_LINUX_DEFAULT="noquiet nosplash"
EOF
update-grub
```

7. Install screenfetch to every console.

```
echo '* * * * * root echo "\l\n$(screenfetch)" > /etc/issue' >>
/etc/crontab
```

8. Setup preferred console font and most important, the font size, so humans can conveniently read the console without a magnifier.

```
cat << EOF > /etc/default/console-setup
# CONFIGURATION FILE FOR SETUPCON
```

```
# Consult the console-setup(5) manual page.

ACTIVE_CONSOLES="/dev/tty[1-6]"

CHARMAP="UTF-8"

CODESET="guess"
FONTFACE="Terminus"
FONTSIZE="14x28"

VIDEOMODE=
EOF
setupcon
```

9. **Optional:** If you need to change your hostname after installing the base os, this is what i do:

```
# set new fqdn/hostname
hostnamectl set-hostname newFullQualifiedHostname.lan

# update /etc/hosts with that new hostname (full + short)
sed -i "s/\(^127.0.1.1\s\).*/\1$(hostname -f)\t$(hostname -s)/"
/etc/hosts
```

10. Setup networking interfaces and ip adresses. This Server is supposed to work as a router/firewall so it has multiple interfaces. Your interface names may differ. Same with the “transfer net” IP adresses, the net between our servers internet NIC and the ISPs shitty black-box router. **Watch the indentation! Its kinda important with yaml files!**:

```
cat << EOF > /etc/netplan/55-manual-net-config.yaml
network:
  version: 2
  ethernets:
    enp5s8:
      addresses:
      - 192.168.178.20/24
      - 2a02:8071:b581:4920:feed:dead:beef:20/64
      - fe80::feed:dead:beef:20/64
      gateway4: 192.168.178.1
      gateway6: 2a02:8071:b581:4920::1
      ipv6-privacy: true
      accept-ra: false
      nameservers:
        addresses:
        - ::1
        - 127.0.0.1
    enp1s0:
      dhcp4: false
      dhcp6: false
    usb0:
      dhcp6: true
```

```
      dhcp4: true
  bridges:
    br0:
      accept-ra: false
      addresses:
      - 192.168.0.1/24
      - 192.168.0.2/24
      - 192.168.0.3/24
      - fd00:feed:dead:beef::1/64
      - fd00:feed:dead:beef::2/64
      - fd00:feed:dead:beef::3/64
      dhcp4: false
      dhcp6: false
      interfaces:
      - enp1s0
      nameservers:
        addresses:
        - ::1
        - 127.0.0.1
        search:
        - lan
EOF
netplan apply
```

11. Configure SSH daemon, so it uses a non standard port for all interfaces (wan) that are not used directly from lan:

```
cat << EOF >> /etc/ssh/sshd_config
PasswordAuthentication yes
ListenAddress 0.0.0.0:65000
ListenAddress [::]:65000

ListenAddress [::1]:22
ListenAddress 127.0.0.1:22
ListenAddress [fd00:feed:dead:beef::1]:22
ListenAddress 192.168.0.1:22
EOF
```

RADVD server installation and configuration for ipv6:


Seems most IPv6 clients prefer to do SLAAC only. Thus “ignoring” the dhcp6 information. So it may be necessary to provide a radvd (Router ADVerticement Daemon) as well, do satisfy these clients.

1. Install the RADVD Service:

```
aptitude install -y radvd
```

```
systemctl enable radvd
```

2. Create a configuration file and enter the IPv6 information there:

```
cat > /etc/radvd.conf << EOF
#
# radvd configuration
#
# radvd configuration generated by radvdump 2.17
# received by interface enp5s8
#

interface br0
{
    AdvSendAdvert on;
    # Note: {Min,Max}RtrAdvInterval cannot be obtained with radvdump
    AdvManagedFlag on; # use administered (stateful) protocol (dhcp6
server)
    AdvOtherConfigFlag on; # use administered (stateful) protocol
(dhcp6 server)
    #AdvReachableTime 3600000;
    #AdvRetransTimer 0;
    #AdvCurHopLimit 64;
    AdvDefaultLifetime 1800;

    RDNSS fd00:feed:dead:beef::1
    {
        #    AdvRDNSSLifetime 300;
    }; # End of RDNSS definition

    DNSSL lan {
        #list of dnssl specific options
    }; # end of DNS search list

    prefix fd00:feed:dead:beef::/64
    {
        #AdvValidLifetime 9000;
        #AdvPreferredLifetime 73588;
        #AdvOnLink on;
        #AdvAutonomous on;
        #AdvRouterAddr off;
    }; # End of prefix definition


    route ::/0
    {
        AdvRoutePreference medium;
        AdvRouteLifetime 1800;
    }; # End of route definition

}; # End of interface definition
```

```
EOF
```

3. Start the Service:

```
systemctl restart radvd
```

4. If service startup was successful you should start to see IPv6 configurations accepted by your IPv6 clients on your network. check on Linux client with:

```
ip ad show
```

DHCP server installation and configuration for ipv4 and ipv6:

1. Install the dhcp server:

```
aptitude install -y isc-dhcp-server
```

2. ipv4 configuration file:

```
# IPv4 DHCP only
cat << EOF > /etc/dhcp/dhcpd.conf
server-name "freiburg.lan" ;
ddns-update-style interim;
ddns-updates on;
update-static-leases on;
option T150 code 150 = string;
authoritative;
update-optimization off;
update-conflict-detection off;


# option definitions common to all supported networks...
option domain-name "lan";
option domain-search "lan";
option domain-name-servers 192.168.0.1;
ddns-domainname "lan";
default-lease-time 36000;
max-lease-time 86400;

######################

subnet 192.168.0.0 netmask 255.255.255.0 {
  range 192.168.0.50 192.168.0.99;
  option broadcast-address 192.168.0.255;
  option routers 192.168.0.1;
}
```

```
######################
# HOST DEFINITIONS
################/etc/default/isc-dhcp-server######

######################
host hpcplj1525n {
  hardware ethernet 44:44:44:44:44:44;
  fixed-address printer-hpcplj1525n.lan;
}
######################
host tv-panasonic-sz {
  hardware ethernet 00:00:11:11:11:11;
  fixed-address tv-panasonic-sz.lan;
}
######################
host wlan-pl-ap-sz {
  hardware ethernet 00:00:00:00:00:00;
  fixed-address wlan-pl-ap-sz.lan;
}
######################

key rndc-key {
    algorithm hmac-md5;
    secret xxxxxxxxx;
};


zone lan. {
    primary 127.0.0.1;
    key rndc-key ;
}

zone 0.168.192.in-addr.arpa. {
    primary 127.0.0.1;
    key rndc-key ;
}


EOF
```

3. ipv6 configuration file:

```
cat << EOF > /etc/dhcp/dhcpd6.conf
server-name "freiburg.lan" ;
ddns-update-style interim;
#ddns-update-style standard;
ddns-updates on;
update-static-leases on;
#option T150 code 150 = string;
authoritative;
#update-optimization off;
```

```
#update-conflict-detection off;
#default-lease-time 2592000;
#preferred-lifetime 604800;
#option dhcp-renewal-time 3600;
#option dhcp-rebinding-time 7200;
allow leasequery;
option dhcp6.name-servers fd00:feed:dead:beef::1;
option dhcp6.domain-search "lan";
ddns-domainname "lan";

subnet6 fd00:feed:dead:beef::/64 {
    range6 fd00:feed:dead:beef::50 fd00:feed:dead:beef::ff;
}

key rndc-key {
    algorithm hmac-md5;
    secret xxxxxxxxxx;
};


zone lan. {
    primary 127.0.0.1;
    key rndc-key ;
}

EOF
```

4. Bind DHCP Servers to the LAN interfaces only:

```
cat << EOF > /etc/default/isc-dhcp-server
# Defaults for isc-dhcp-server (sourced by /etc/init.d/isc-dhcp-server)

# Path to dhcpd's config file (default: /etc/dhcp/dhcpd.conf).
#DHCPDv4_CONF=/etc/dhcp/dhcpd.conf
#DHCPDv6_CONF=/etc/dhcp/dhcpd6.conf

# Path to dhcpd's PID file (default: /var/run/dhcpd.pid).
#DHCPDv4_PID=/var/run/dhcpd.pid
#DHCPDv6_PID=/var/run/dhcpd6.pid

# Additional options to start dhcpd with.
#   Don't use options -cf or -pf here; use DHCPD_CONF/ DHCPD_PID
instead
#OPTIONS=""

# On what interfaces should the DHCP server (dhcpd) serve DHCP
requests?
#   Separate multiple interfaces with spaces, e.g. "eth0 eth1".
INTERFACESv4="br0"
INTERFACESv6="br0"
```

```
EOF
```

5. Restart DHCP daemons:

```
systemctl restart isc-dhcp-server
systemctl restart isc-dhcp-server6
```

DNS Server installation and configuration:

1. Install bind9 DNS Server:

```
aptitude install -y bind9
```

2. Make sure bind9/named only serves on LAN interfaces/ip addresses. AND configure our default remote DNS servers we would like to use for name resolution:

   /etc/bind/named.conf.options

```
options {
    directory "/var/cache/bind";

    // If there is a firewall between you and nameservers you want
    // to talk to, you may need to fix the firewall to allow
multiple
    // ports to talk.  See http://www.kb.cert.org/vuls/id/800113

    // If your ISP provided one or more IP addresses for stable
    // nameservers, you probably want to use them as forwarders.
    // Uncomment the following block, and insert the addresses
replacing
    // the all-0's placeholder.

    // forwarders {
    //     0.0.0.0;
    // };

    # public DNS Servers
    forwarders {
        2001:4860:4860::8888;
        2001:4860:4860::8844;
        8.8.8.8;
        8.8.4.4;
        1.1.1.1;
        1.0.0.1;
        2606:4700:4700::1111;
        2606:4700:4700::1001;
    };
    forward first;
```

```
    auth-nxdomain no;    # conform to RFC1035

    #
    # make sure to listen/bind to LAN interfaces only!
    # (and localhost of course)
    #
    listen-on-v6 {
        ::1;
        fd00:feed:dead:beef::1;
    };

    listen-on {
        127.0.0.1;
        192.168.0.1;
    };
};
```

3. Configure which DNS zones (domains) we'd like to serve to our clients and which secret key is used for dynamic dns (ddns) updates:

   /etc/bind/named.conf.local

```
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in
your
// organization
include "/etc/bind/zones.rfc1918";


zone "lan" in {
    //type master;
    type primary;
    file "/var/lib/bind/db.lan";
    notify no;
    allow-update { key "rndc-key" ; } ;
};

zone "0.168.192.in-addr.arpa" in {
    //type master;
    type primary;
    file "/var/lib/bind/db.192.168.0.0";
    notify no;
    allow-update { key "rndc-key" ; } ;
};
```

```
//
//  NOTE: ipv6 forward zone is same as for ipv4!
//
//
// ipv6 reverse lookup zone missing

/*
zone "e.f.f.a.f.e.e.b.d.a.e.d.0.0.d.f.ip6.arpa" {
        type master;
    file
"/var/lib/bind/db.e.f.f.a.f.e.e.b.d.a.e.d.0.0.d.f.ip6.arpa";
    allow-update { key "rndc-key" ; } ;
};
*/


zone "freiburg.myhome-server.de" in {
    type master;
    file "/var/lib/bind/db.freiburg.myhome-server.de";
    notify no;
    allow-update { key "rndc-key" ; } ;
};

#zone "awerner.myhome-server.de" in {
#   type master;
#   file "/var/lib/bind/db.awerner.myhome-server.de";
#   notify no;
#   allow-update { key "rndc-key" ; } ;
#};

#zone "mailgate.myhome-server.de" in {
#   type master;
#   file "/var/lib/bind/db.mailgate.myhome-server.de";
#   notify no;
#   allow-update { key "rndc-key" ; } ;
#};


key "rndc-key" {
    algorithm hmac-md5;
    secret "ixxxxxxxxx";
};
```

- **NOTE:** Note that the named configuration file is separate from the zone files at **/var/lib/bind/** ! This is is because UBUNTU has apparmor enabled by default and guards/blocks any filesystem access anywhere else by the daemon. In theory you could store the zone files whereever you like, but its easier for me to follow the distribution's recommendation.

4. Forward lookup zone for lan domain (ipv4 + ipv6):

/var/lib/bind/db.lan

```
$ORIGIN .
$TTL 72800  ; 20 hours 13 minutes 20 seconds
lan         IN SOA    freiburg.lan. hostmaster.freiburg.lan. (
                20222130   ; serial
                86400      ; refresh (1 day)
                7200       ; retry (2 hours)
                604800     ; expire (1 week)
                172800     ; minimum (2 days)
                )
            NS    freiburg.lan.
            MX    10 freiburg.lan.
$ORIGIN lan.
cluster         A    192.168.0.100
cluster-node1       A    192.168.0.101
cluster-node2       A    192.168.0.102
cluster-node3       A    192.168.0.103
cluster2        A    192.168.0.110
cluster2-node1      A    192.168.0.111
cluster2-node2      A    192.168.0.112
cluster2-node3      A    192.168.0.113
freiburg        A    192.168.0.1
            AAAA    fd00:feed:dead:beef::1
printer-hpcplj1525n A    192.168.0.8
farblaser       CNAME    printer-hpcplj1525n
probook-wlan        A    192.168.0.3
tv-panasonic-sz     A    192.168.0.13
tv-sz           CNAME    tv-panasonic-sz
wlan-pl-ap-sz       A    192.168.0.252
```

5. Reverse lookup zone for lan ipv4 IPs:

/var/lib/bind/db.192.168.0.0

```
$ORIGIN .
$TTL 172800 ; 2 days
0.168.192.in-addr.arpa  IN SOA    freiburg.lan.
hostmaster.freiburg.lan. (
                20211583   ; serial
                86400      ; refresh (1 day)
                7200       ; retry (2 hours)
                604800     ; expire (1 week)
                172800     ; minimum (2 days)
                )
            NS    freiburg.lan.
$ORIGIN 0.168.192.in-addr.arpa.
1           PTR    freiburg.lan.
100         PTR    cluster.lan.
```

```
101             PTR     cluster-node1.lan.
102             PTR     cluster-node2.lan.
103             PTR     cluster-node3.lan.
110             PTR     cluster2.lan.
111             PTR     cluster2-node1.lan.
112             PTR     cluster2-node2.lan.
113             PTR     cluster2-node3.lan.
12              PTR     lab.lan.
13              PTR     tv-panasonic-sz.lan.
2               PTR     wz.lan.
252             PTR     wlan-pl-ap-sz.lan.
3               PTR     probook-wlan.lan.
$TTL 3600   ; 1 hour
50              PTR     ubuntu-server-2004.lan.
52              PTR     Nexus-6.lan.
$TTL 172800 ; 2 days
8               PTR     printer-hpcplj1525n.lan.
```

6. Check config for errors and restart the DNS Server:

```
named-checkconf
rm -v /var/lib/bind/*.jnl # remove offending journal zones
systemctl restart named
systemctl status named
```

7. Check that **named** only listens to interfaces/ip addresses that you like to serve:

```
ss -nptul | grep -i named
```

8. Fix Me! IPv6 reverse lookup still missing Fix Me!

```
FIXME
```

Enable routing (ip forwarding) incl. firewalling and NAT for ipv4 AND ipv6 using firewalld :

NOTE: Almost everywhere you look around on the interwebs you will find documents and tutorials claiming that using NAT (nat6) and/or IP masquerading on IPv6 is a “no go”, saying this is a thing for IPv4 only and that you should not use it. However i think its totally valid and a good idea to still use it on IPv6 for most home and small office use. It prevents your ISP sniffing around in your LAN and/or renders them unable to count the number of computers/devices on your network. So i prefer to use it on IPv6 just as i use(ed) it with IPv4!

Have a good read at:

- https://mirrors.bieringer.de/Linux+IPv6-HOWTO/
- https://firewalld.org/documentation/
- https://firewalld.org/documentation/man-pages/firewalld.richlanguage.html
- https://github.com/firewalld/firewalld/issues/29

This is how i enable routing and firewalling on my Linux router/box including nat and ip masquerading:

1. Enable IPv4 ip forwarding (routing):

```
cat << EOF > /etc/sysctl.d/55-my-ipv4-routing.conf

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Do not accept ICMP redirects (prevent MITM attacks)
net.ipv4.conf.all.accept_redirects = 0

# Log Martian Packets
net.ipv4.conf.all.log_martians = 1

EOF
```

2. Enable IPv6 ip forwarding (routing):

```
cat << EOF > /etc/sysctl.d/55-my-ipv6-routing.conf

# Uncomment the next line to enable packet forwarding for IPv6
#  Enabling this option disables Stateless Address Autoconfiguration
#  based on Router Advertisements for this host
net.ipv6.conf.all.forwarding=1

# Do not accept ICMP redirects (prevent MITM attacks)
net.ipv6.conf.all.accept_redirects = 0

EOF
```

3. I am assuming all your network interfaces are already configured at this point.
4. Install firewalld:

```
apt install -yy firewalld
```

5. Lets see if its aleady running and if the service is enabled:

```
systemctl status firewalld
# expected result:
#      service must be enabled and running
#
#● firewalld.service - firewalld - dynamic firewall daemon
#     Loaded: loaded (/lib/systemd/system/firewalld.service; enabled;
vendor preset: enabled)
```

```
#     Active: active (running) since xxxxxxxxxx xxxxxx ago
# ...
```

6. Have a glimpse about what zones and rules are already in place out of the box. If you want to check a specific firewall zone only, you can do that as well. For my scenario the following **two zones are of use: external, internal** :

```
firewall-cmd --list-all-zones
firewall-cmd --list-all --zone=internal
firewall-cmd --list-all --zone=external
```

7. First im going to tell the firewalld what zone is supposed to become my “default zone” for new (future) network interfaces. Thats a security point! If someone suddenly plugs in some USB network interface (or a tethering smartphone) for example, its supposed to be “as safe” as our **external** zone , which is supposed to be the network interface facing the evil internet:

```
firewall-cmd --set-default-zone=external
```

8. Next we need to define which network interfaces are which. Meaning, we need to tell firewalld which physical (and/or virtual) interface is part of which firewall zone. The network **interface facing the internet** is supposed to be part of the **unsecure external zone**. The **interface facing to our LAN** is supposed to be assigned to the **internal** zone, as its usually more secure and we want to allow more access to it:

```
# Any interwebs interfaces goes "external" zone
firewall-cmd --permanent --add-interface=enp5s8 --zone=external
## in case of my usb tethered smartphone
firewall-cmd --permanent --add-interface=usb0 --zone=external
firewall-cmd --permanent --add-interface=usb1 --zone=external

# Any LAN (or bridge) interfaces goes "internal" zone
firewall-cmd --permanent --add-interface=br0 --zone=internal

# My physical LAN interface is not used (unconfigured)
# since i use BR0 instead. Therefore i can put it in
# the "block" zone, for security reasons.
firewall-cmd --permanent --add-interface=enp1s0 --zone=block
```

9. IPv4 masquerading has been enabled on the **“external”** zone by default. So IPv4 clients already can work across our Linux router. The **IPv6 masquerading is NOT enabled by default** and **requires the use of firewalld's “rich rules” to be added manually**. Since i prefer to have IPv6 NAT/MASQUERADING as well (because i dont want my internal IPv6 clients to have “global (public) IPv6 addresses”) we can enable it relative simply like this:

```
firewall-cmd --permanent --zone=external --add-rich-rule='rule
family=ipv6 masquerade'

# This creates an identical IPv6 masquerading iptable/ip6table rule!
# compare the output of iptable-save and ip6table-save
#     ip6tables-save | grep -i masq
#     iptables-save | grep -i masq
#         -A POST_external_allow ! -o lo -j MASQUERADE
```

```
#         -A POST_external_allow ! -o lo -j MASQUERADE
```

10. Now lets configure some rules, services and ports for our **external** zone, so that people from the interwebs can access our local network services on our Linux server:

```
firewall-cmd --permanent  --zone=external --add-service=dhcpv6-client
firewall-cmd --permanent  --zone=external --add-service=http
firewall-cmd --permanent  --zone=external --add-service=https

# ssh
firewall-cmd --permanent --zone=external --add-port=65000/tcp

# eye too pee
firewall-cmd --permanent --zone=external --add-port=19901/tcp
firewall-cmd --permanent --zone=external --add-port=19901/udp
```

11. Now lets configure some rules, services and ports for our **internal** zone, so that our devices and colleagues within our organisation can access our own local services on our Linux server. Of course only enable what you really need:

```
firewall-cmd --permanent --zone=internal --add-service=ssh
firewall-cmd --permanent --zone=internal --add-service=http
firewall-cmd --permanent --zone=internal --add-service=https
firewall-cmd --permanent --zone=internal --add-service=dns
firewall-cmd --permanent --zone=internal --add-service=dhcp
firewall-cmd --permanent --zone=internal --add-service=dhcpv6
firewall-cmd --permanent --zone=internal --add-service=imap
firewall-cmd --permanent --zone=internal --add-service=imaps
firewall-cmd --permanent --zone=internal --add-service=pop3
firewall-cmd --permanent --zone=internal --add-service=pop3s
firewall-cmd --permanent --zone=internal --add-service=smtp
firewall-cmd --permanent --zone=internal --add-service=smtps
firewall-cmd --permanent --zone=internal --add-service=smtp-submission
firewall-cmd --permanent --zone=internal --add-service=ntp
```

12. I prefer to have firewall LOGGING enabled and relatively verbose, so i can see at all time whats going on or why something didnt worked:

```
firewall-cmd --set-log-denied=all
```

13. **A NOTE ABOUT REJECT vs DROP default target (action):** By default firewalld will “actively reject” any traffic or attempt that is not allowed by our rules. That means the sender or attacker will receive an active message from our server, mentioning that his traffic has been blocked. That is good for debugging and “maybe” for internal use. But its usually better practice to switch that behavior to “DROP”. That means our firewall will no longer “actively tell”, but silently drop the packets. So an attacker cant tell for sure whats going on. It also saves us bandwidth, since every single denied packet would else have been answered, using our precious bandwidth:

```
firewall-cmd --permanent --zone=external --set-target=DROP
```

```
# default is:
# firewall-cmd --permanent --zone=external --set-target=default
```

14. Now Reload/Restart our freshly configured firewall (or reboot server), so it activates all the new rules and features:

```
firewall-cmd --reload
# or reboot
```

15. There *seems* to be no need for special Docker rules or setup, since Docker from version 20.10 on seems to be in harmony with firewalld. So... we will see about that soon i guess!
16. Fix Me!:

```
FIXME
```

17. Fix Me!:

```
FIXME
```

sshguard - Installation and configuration:

The very moment you expose any network service (open port) to the internet it will be found and attacked by bots and hackers. Especially when “well known ports” are used. Since we wanted to access our server via ssh from across the internet we had to configure and open a port for our SSH daemon. But even we used a non-standard TCP Port it's just a matter of time until the automatic password and brute force attacks will start on it.

So to detect and counter meassure these attacks we can use **sshguard**, which monitors failed login attempts on our ssh daemon and keeps track of the IP addresses used by these attackers. The moment an attacker attempt too many failed logins within a specified period of time it's IP address will be put on an internal black list and a “script” execution will be triggered that can configure a locally installed hostfirewall (e.g. firewalld in our case) to block this source IP from further accessing ANY services on this host.

Here we now install and configure **sshguard** for use with our previously installed firewalld:

1. **NOTE: For this manual firewalld has to be installed!**
2. Install **sshguard** package:

```
sudo apt install -yy sshguard
```

3. The main configuration file for **sshguard** is **/etc/sshguard/sshguard.conf**. We need to make changes there so it will make use of **firewalld**. Also i'd like to change some default thresholds and time periods to make it real hard:

```
# make backup first
cp -v /etc/sshguard/sshguard.conf{,.$(date +%F)}
```

```
# use firewalld
sed -i 's|^BACKEND.*$|BACKEND="/usr/lib/x86_64-linux-gnu/sshg-fw-
firewalld"|' /etc/sshguard/sshguard.conf

# remember every attack within 24h (24*60*60 secs)
sed -i '/^DETECTION_TIME=/s/=.*$/'=$((24*60*60))/
/etc/sshguard/sshguard.conf

# Ban/Block/Filter attackers IP for 1h (60*60 secs)
sed -i '/^BLOCK_TIME=/s/=.*$/'=$((60*60))/ /etc/sshguard/sshguard.conf
```

4. Finally we may want to configure some IPs and or hostnames which should NEVER be banned (whitelisting), even there has been seen attacks coming from those IPs. That said, be cautious with this, else it might undermine your intention:

```
echo "someHost.fqdn.tld" >> /etc/sshguard/whitelist
echo "yourAdminsPC.fqdn.tld" >> /etc/sshguard/whitelist
echo "yourNetworkIP/yourNetmask" >> /etc/sshguard/whitelist
```

5. Enable **sshguard** service to it runs after reboots:

```
systemctl enable sshguard
```

6. Finally restart service to activate it:

```
systemctl stop sshguard
systemctl start sshguard
```

7. **Handling Tip**: If one of your own IP addresses has accidently been banned/filtered you can **remove or “unban”** it as follows:
    1. List banned IPs using **ipset**:

    ```
    ipset list
    ```

    2. Remove IP from ipset:

    ```
    ipset del sshguard4 XXX.XXX.XXX.XXX
    # or
    ipset del sshguard6 XXXX:XXXX:XXXX::XXXX
    ```

    3. This should instantly allow new connections from the removed IP. However…sshguard does not actively monitor the “ipset” list and is unaware of the removal. That said it means that it won't put the offending IP back on the ipset list until the detention time is over. Else you would need to restart the sshguard service to make it forget everything and start over fresh.

Setting up free dynamic DNS hostnames using ddnss.de (ipv6 + ipv4)

1. Register at the free dyndns Provider ddnss.de, then find and choose one or more apropriate

hostnames for your Linux box.

2. Get yourself a secret “Update Key” from that website (your dashboard), so we can update our dyndns hostnames without using a loginname and password.
3. Now lets add a mechanism to our Linux box to periodically update our dyndns hostnames with what we have on board. Its not ideal this way, but quick n dirty. Here i decided to use a hourly cron job:

```
cat << 'EOF' > /etc/cron.hourly/updateDynDNS
#!/bin/bash
#
# VERSION 2020-01-09
#
# WARNING! Contains secret keys!
#

LOGFILE=/var/log/$(basename $0).log

if [ ! -f "$LOGFILE" ] ; then
    touch "$LOGFILE"
    chmod ugo+rw "$LOGFILE"
fi

# update ALL your dyndns hostnames automatically
# (autodetect ipv4 + ipv6)
#
#wget -O -
'http://ddnss.de/upd.php?key=putYourSecretKeyStringHere&host=all&mx=1'
2>>"$LOGFILE" 1>>"$LOGFILE" || {
    echo "$0:ERROR:Update failed. Check $LOGFILE for details!" >&2
}

# OR only update hostnames listed here AND only autoupdate
# your IPv6 address, while setting the IPv4 pointing to
# a specific IP address of your choice
# (maybe a rented ipv4 webserver somewhere else)
#
wget -O -
'http://ddnss.de/upd.php?key=putYourSecretKeyStringHere&host=hostname1.
myhome-server.de,hostname2.myhome-server.de,hostname3.myhome-
server.de&mx=1&ip=xxx.xxx.xxx.xxx' 2>>"$LOGFILE" 1>>"$LOGFILE" || {
    echo "$0:ERROR:Update failed. Check $LOGFILE for details!" >&2
}


EOF
```

4. Make our update script executable:

```
chmod -c ug+x /etc/cron.hourly/updateDynDNS
chmod -c o-rwx /etc/cron.hourly/updateDynDNS
```

5. Test if the dyndns update script worked by checking if your hostnames at the dyndns provider

are properly pointing to your ipv4 and/or ipv6 addresses now:

```
# test dns resolution
nslookup yourDynDnsHostnameHere.tld
.
.
.
# check the 'Address' lines for IPv4
# and IPv6 addresses. Compare them to
# what has been assigned to your router
# or linux box.
.
.
.
Address: XX.XXX.XX.X14
Name:   pyourDynDnsHostnameHere.tld
Address: 2a02:xxxx:xxxx:xxxx:feed:dead:beef:20
```

certbot and LetsEncrypt tls certificats Installation and configuration

letsencrypt.org offers official x509 certificats for free, everyone can create and use for webservers, database connections etc. These are only valid for 90 days or so, but since this CA offers the “ACME” protocol (IETF standard), we can automate the process of getting them and updating them. The official tool to handle this is called “certbot”, which we are using here. But there are many other methods as well. See letsencrypt.org if interested.

This is how i did it:

1. Install **certbot** tool and some of its plugins:

```
aptitude install -y python3-certbot
```

2. Kickstart letsencrypt **with NO Webserver running on port 80** at the moment:

```
certbot certonly \
  --noninteractive \
  --domain yourDdnssDomainName \
  --domain www.yourDdnssDomainName \
  --email yourAdministrativeEmailContact@gmail.com \
  --staple-ocsp \
  --no-eff-email  \
  --standalone \
  --agree-tos
```

3. Else kickstart letsencrypt **with Apache already running** on port 80:

```
certbot certonly \
```

```
  --noninteractive \
  --webroot-path /var/www/yourWebRootPath/ \
  --domain yourDdnssDomainName \
  --domain www.yourDdnssDomainName \
  --email yourAdministrativeEmailContact@gmail.com \
  --staple-ocsp \
  --no-eff-email  \
  --webroot \
  --agree-tos
```

4. As console messages should mention, our freshly made TLS certificates for our FQDN have been places in a subdirectory at **/etc/letsencrypt/live/** in pretty much every variation there is. List them with:

```
ls -laRh /etc/letsencrypt/live/
```

5. **PRO TIP:** To view the contents of your certificate file in a more **human readable format** and for you to check if all your additional hostnames are also in there, you can use the openssl tool like this:

```
openssl x509 -text  -in /etc/letsencrypt/live/yourFqdnHere/cert.pem  |
more
```

6. Now all we need to do is configure our network services to use these new certificate files. Either “copy” them , link to them (symbolic links) or just point absolutely to them, as long your service is able to read them.
7. **IMPORTANT:** Until now this was just a “single shot action”. But **our new certificates will expire** within 90 days or so, if we dont take repetitive action. To **update our certificates** (and private key) all we need to do is reissue the command again OR use an even shorter version shown later, like once a month or so. This should be enough to keep us flying. We use cron for this and create us a little cron job, that executes on every 1st day of month:

```
cat << 'EOF' >/etc/cron.monthly/certbot-renew
#!/bin/sh
set -e

# stopping/restarting apache is
# only required if you kickstarted
# letsencrypt with --standalone
# mode, to prevent port conflicts
# since certbot will attempt to
# spin up its own little
# temporary webserver on port 80
# for a few seconds!

#systemctl stop apache2
certbot --quiet renew
# restart webserver for new certificates to
# take effect.
systemctl restart apache2
EOF
```

```
chmod +x /etc/cron.monthly/certbot-renew
```

8. Whenever you need to check what's going on with **certbot** have a look in its log file at **/var/log/letsencrypt/letsencrypt.log** !
9. **IF YOU NEED TO START OVER:** If you want to start over and reset/discard everything, just delete the whole **/etc/letsencrypt/** directory and run your certbot command again.

Apache2 Webserver installation and configuration:

This section is independent from the IP version used! For IPv4 and IPv6 its the same.

Since i want to serve both, plain html and other simple files directly by the webserver AND also be able to serve/process PHP apps, im going to install and configure php version 7.4 as well. This enables the use for DokuWiki for example.

1. Install Apache and PHP74:

```
aptitude update && aptitude install -y \
  php7.4 \
  apache2 \
  php7.4-zip \
  php7.4-readline \
  php7.4-gmp \
  php7.4-gd \
  php7.4-cli \
  php7.4-common \
  php7.4-bz2 \
  php7.4-xml \
  php7.4-mbstring
```

2. Enable the Apache2 Service:

```
systemctl enable apache2
systemctl status apache2
```

3. On ubuntu Server out of the box Apache has been enabled to look and serve pages and files from within the **/var/www/html/** directory. This is defined through the configuration file at **/etc/apache2/sites-enabled/000-default.conf**. Ubuntu put a demo page there, named **index.html**. To test if Apache is working so far, you can aim your webbrowser at it, like...

```
firefox http://freiburg.lan/ &
```

A **“Apache2 Ubuntu Default Page”** should come up. So your Apache webserver works so far. Still not able to process PHP files nor supporting TLS/SSL (https) just yet.

1. **IMPORTANT: Since Firefox and similar professional web browsers these days decided to prefer the TLS secured https over http by default, its important to tell it explicitly when NOT to use https by prefixing the hostname with**

'http://' ! Else the browser may try to connect using https only, which will fail at this point in time, since we did not configure TLS/SSL with our webserver just yet!

4. Now for PHP content we prepare a small php test page for Apache, like this:

```
cat << "EOF" > /var/www/html/test.php
<?php phpinfo(); ?>
EOF
```

When Apache processes this file, its supposed to spit out lots and lots of information about the PHP environment it found. That would be prove that our Apache is PHP enabled.

5. Test if Apache is already PHP enabled yet:

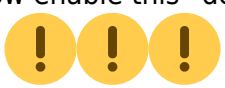
```
firefox http://freiburg.lan/test.php &
```

If you now see a nice large table even with the PHP logo in the top right it means that your Apache Installation is already PHP enabled and working already! (Thanks to **/etc/apache2/sites-enabled/000-default.conf** ! This file later will be a good template for our own configuration file)

6. Lets test if the default (Ubuntu's) Apache configuration also enabled “index” PHP pages, by renaming our **test.php** to **index.php** and remove or rename the former **index.html**, so there is no conflict:

```
cd /var/www/html
mv -v test.php index.php
mv -v index.html index.html.disabled
ls -l
#
#-rw-r--r-- 1 root root 10918 Oct  8 06:38 index.html.disabled
#-rw-r--r-- 1 root root    20 Oct  8 07:14 index.php
#root@yourHost:/var/www/html#
firefox http://freiburg.lan/ &
```

It the PHP info table still comes up while we did not specify the page name on the webbrowser it means that Apache has been configured to serve us the **index.php** file. Thats good! So our Apache webserver is pretty much ready to serve any kind of php app now. Just remove all the test files from **/var/www/html/** and put your html or php web app there and there you g“continue and adding an exception”o. Make sure that the Apache service account is able to read all the files there. Else you will end up with “access denied” error messages or some sort.

7. UBUNTU by default also provides a minimal default SSL configuration for apache (**/etc/apache2/sites-available/default-ssl.conf**) , including a self-signed ssl/tls certificate (**/etc/ssl/certs/ssl-cert-snakeoil.pem**) and key (**/etc/ssl/private/ssl-cert-snakeoil.key**) file. Since more and more modern webbrowsers expect us to provide SSL/TLS enabled webbrowser, we now enable this “default ssl configuration” that comes with ubuntu's apache.
    1.  **This is NOT the safest Apache or SSL/TLS configuration! This is just the UBUNTU “out of the box” TLS/SSL configuration, good for 1st Steps testing. NOT SAFE for any productive environment.**
8. Enable the prepared “default-ssl.conf” apache configuration:

```
a2ensite default-ssl.conf
```

9. Enable Apache's ssl module:

```
a2enmod ssl
```

10. Restart Apache deamon after change of configuration:

```
systemctl restart apache2
```

11. Now test the TLS/SSL access with your webbrowser:

```
firefox https://freiburg.lan/ &
```

    1. **Since we are only working with a “self signed TLS certificate” provided by UBUNTU so far, the webbrowser by default WILL NOT TRUST our webserver, thus complaining or warning you as the user about lack of trust. That's expected, but not a problem!** If YOU trust your own web server just allow your web browser to trust it too, by “continue and adding an exception”. From now on its supposed to no longer block access and you should be able to see your (default) website from earlier.

12. **NOTE: To enable “official TLS server certificates” on your Apache web server so that any web browser will “trust us” we'll need official (globally available) FQDN for our server as well!** That means, that we most likely cannot get any official CA to issue TLS server certificates for our “internal host name” like **freiburg.lan**. So at this point we assume that this host also already has some sort of proper external DNS host name as well. Like **awerner.myhome-server.de** or similar and we assume that matching server certificates has already been acquired and are in place as prepared in the earlier step certbot-and-letsencrypt-tls-certificats-installation-and-configuration, so we can use them now.
13. Lets modify the **default-ssl.conf** now and install our new official TLS certificates into this configuration, instead of using a **self signed certificate**:

```
# always make a backup first!
cp -v /etc/apache2/sites-available/default-ssl.conf{,.$(date +%F)}

#
# backup the original "SSLCertificateFile" line by
# duplicate + comment it out, then
# replace path to our own certificate file
#
# NOTE: i use the "fullchain.pem" file here, instead of
#       the plain certificate, because else i would have
#       to provide an additional keyword/line with the
#       CA-chain as well. Using the "fullchain.pem" which
#       includes certificate + ca-chain solves that.
#
sed -i
"s/^\([[:space:]]*\)\(SSLCertificateFile[[:space:]]*.*$\)/\1#\2\n\1SSLC
ertificateFile \/etc\/letsencrypt\/live\/yourFqdnHere\/fullchain.pem/"
```

```
/etc/apache2/sites-available/default-ssl.conf



#
# backup the original "SSLCertificateKeyFile" line by
# duplicate + comment it out, then
# replace path to our own certificate file
#
sed -i
"s/^\([[:space:]]*\)\(SSLCertificateKeyFile[[:space:]]*.*$\)/\1#\2\n\1S
SLCertificateKeyFile
\/etc\/letsencrypt\/live\/yourFqdnHere\/privkey.pem/"
/etc/apache2/sites-available/default-ssl.conf
```

14. Restart Apache service and check its status and log:

```
systemctl restart apache2
systemctl status apache2
```

15. If the service is running without trouble its time to visit its website and see if the webbrowser now recognizes the **“official server certificate”** (by letsEncrypt), by NOT presenting us any security warning or error message within the browser address bar:

```
firefox https://yourFqdnHere/ &
```

Apache2 Webserver - Make it bind only to specific IP addresses, not everyone:

If you run network services like the Apache on your bare metal machine AND later want to add additional network services through Linux containers (Docker|Kubernetes or similar) you might run into port conflicts (port already in use by your Apache server). As usual TCP/UDP ports can only be attached or bind by **one server process**, so either Apache or your container service can bind to it, not both at the same time.

The default Apache configuration files provided by UBUNTU 20 are build to let Apache **bind to ALL local IP addresses available to your machine**. To prevent that behavior we need to modify our Apache configuration files and tell it exactly which IP address to bind to.

See for more details:

- https://httpd.apache.org/docs/2.4/bind.html
- https://httpd.apache.org/docs/2.4/vhosts/examples.html

In this example we modify the configuration file “**/etc/apache2/ports.conf**”, so Apache will no longer bind to every IP address on the system, but only specific ones.

1. Backup existing configuration file:

```
cp -v /etc/apache2/ports.conf{,.$(date +%F)}
```

2. Replace the generic **Listen** statements with our more specific ones. Note that we need to add every IP adress (IPv4 and IPv6) where we want Apache to listen or “bind” to. Expecially if you have multiple network interfaces and you want to access your Apache from multiple directions:

```
cat << 'EOF' > /etc/apache2/ports.conf
# If you just change the port or add more ports here, you will likely
also
# have to change the VirtualHost statement in
# /etc/apache2/sites-enabled/000-default.conf
#
#Listen 80
#
#<IfModule ssl_module>
#   Listen 443
#</IfModule>
#
#<IfModule mod_gnutls.c>
#   Listen 443
#</IfModule>
#

# HTTP / TCP 80
#   INTERNAL (LAN) IPs
Listen 192.168.0.1:80
Listen [fd00:feed:dead:beef::1]:80
#   EXTERNAL (INTERNET/DMZ) IPs
Listen 192.168.178.20:80
Listen [2a02:xxxx:xxxx:xxxx:feed:dead:beef:20]:80

# HTTPS / TCP 443
<IfModule ssl_module>
    #    INTERNAL (LAN) IPs
    Listen 192.168.0.1:443
    Listen [fd00:feed:dead:beef::1]:443
    #    EXTERNAL (INTERNET/DMZ) IPs
    Listen 192.168.178.20:443
    Listen [2a02:xxxx:xxxx:xxxx:feed:dead:beef:20]:443
</IfModule>

<IfModule mod_gnutls.c>
    # INTERNAL (LAN) IPs
    Listen 192.168.0.1:443
    Listen [fd00:feed:dead:beef::1]:443
    #EXTERNAL (INTERNET/DMZ) IPs
    Listen 192.168.178.20:443
    Listen [2a02:xxxx:xxxx:xxxx:feed:dead:beef:20]:443
</IfModule>
```

```
EOF
```

DokuWiki installation and configuration on Apache2

DokuWiki, a great simple to install Wiki without the need of a database server, recommends to use some additional precautions and configurations for security reasons. So we are going to follow these recommendations.

1. According to the DokuWiki Homepage it requires the Apache module **mod_rewrite** to be enabled to make the **.htaccess** files work, that come with DW. Therefore we enable this mod here:

```
a2enmod rewrite
```

2. Download and unpack DokuWiki to a directory under **/var/www/** named after your virtual named host:

```
mkdir -p /var/www/awerner.myhome-server.de
cd /var/www/awerner.myhome-server.de

wget https://download.dokuwiki.org/src/dokuwiki/dokuwiki-stable.tgz

tar -xvf dokuwiki-stable.tgz
rm -v dokuwiki-stable.tgz
mv -v dokuwiki-*/* dokuwiki-*/.* .
rmdir -v dokuwiki-*
```

3. Fix filesystem permissions (owners) so the Apache service can own and change files here if needed:

```
chown -cR www-data:www-data .
```

4. Add a virtual host configuration for Apache that handles port 80 http requests only:

```
cat << 'EOF' > /etc/apache2/sites-available/awerner.myhome-
server.de.conf
<VirtualHost *:80>
    ServerAdmin webmaster@awerner.myhome-server.de

    ServerName awerner.myhome-server.de
    ServerAlias www.awerner.myhome-server.de

    DocumentRoot /var/www/awerner.myhome-server.de

    <Directory /var/www/awerner.myhome-server.de>
        Options Indexes FollowSymLinks MultiViews
        #AllowOverride None
        AllowOverride Options AuthConfig FileInfo Limit
        Order allow,deny
        allow from all
```

```
    </Directory>

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn
    ServerSignature Off

    ErrorLog /var/log/apache2/awerner.myhome-server.de.error.log
    CustomLog /var/log/apache2/awerner.myhome-server.de.access.log
combined

        # allow letsEncrypt to access the challenge file
        # unencrypted (without redirection to https)
        # anything else redirect to https://
        RewriteEngine On
        RewriteCond %{REQUEST_URI} !^/\.well\-known/acme\-challenge/
        #RewriteRule ^(.*)$ https://%{HTTP_HOST}$1 [R=301,L]
</VirtualHost>

EOF
```

5. Enable our new Apache site configuration:

```
a2ensite awerner.myhome-server.de.conf
```

6. Restart Apache Service to activate new configuration:

```
systemctl restart apache2.service
systemctl status apache2.service
```

7. Test access via **http (Port80)** on DokuWiki and start with the important 1st steps to setup and secure the web app.  **NOTE the http:// prefix!** Modern browsers often skip http entirely and use https instead, if the protocoll is omited:

```
firefox http://awerner.myhome-server.de &

# expected result: plain DokuWiki Homepage shown
```

8. Since this web app is not yet set up properly, we need do this NOW, by manually visiting the **install.php** page and enter some things. Follow DokuWiki setup instructions at https://www.dokuwiki.org/ :

```
firefox http://awerner.myhome-server.de/install.php &
```

9. After basic setup, DokuWiki mentions that now you should **delete the install.php** file now:

```
rm -v /var/www/awerner.myhome-server.de/install.php
```

10. DokuWiki now is up and running. But yet only accessable **via http only**, which is unsecure, since content can and often is transfered over the networks in clear text. Pretty much any attacker can not only spy, but often can even swap out content from within the data stream

without you noticing. So the use of **https (port 443) is highly recommended instead!!** That is what we are going to do next.

11. Add a new virtual host configuration for Apache that handles port 443 https requests only:

```
cat << 'EOF' > /etc/apache2/sites-available/awerner.myhome-server.de-
ssl.conf
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerAdmin webmaster@awerner.myhome-server.de

        ServerName awerner.myhome-server.de
        ServerAlias www.awerner.myhome-server.de

        DirectoryIndex index.html index.cgi index.pl index.php
index.xhtml index.htm default.htm default.html
        DocumentRoot /var/www/awerner.myhome-server.de

        <Directory /var/www/awerner.myhome-server.de>
            Options Indexes FollowSymLinks MultiViews
    #        AllowOverride None
            AllowOverride Options AuthConfig FileInfo Limit
            Order allow,deny
            allow from all
        </Directory>

        # Possible values include: debug, info, notice, warn, error,
crit,
        # alert, emerg.
        LogLevel warn
        ServerSignature Off

        ErrorLog /var/log/apache2/awerner.myhome-server.de-
ssl.error.log
        CustomLog /var/log/apache2/awerner.myhome-server.de-
ssl.access.log combined


        #   SSL Engine Switch:
        #   Enable/Disable SSL for this virtual host.
        SSLEngine on

        #   A self-signed (snakeoil) certificate can be created by
installing
        #   the ssl-cert package. See
        #   /usr/share/doc/apache2/README.Debian.gz for more info.
        #   If both key and certificate are stored in the same file,
only the
        #   SSLCertificateFile directive is needed.
        #SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
        SSLCertificateFile /etc/letsencrypt/live/awerner.myhome-
server.de/fullchain.pem
```

```
        #SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key
        SSLCertificateKeyFile /etc/letsencrypt/live/awerner.myhome-
server.de/privkey.pem

        #   Server Certificate Chain:
        #   Point SSLCertificateChainFile at a file containing the
        #   concatenation of PEM encoded CA certificates which form the
        #   certificate chain for the server certificate. Alternatively
        #   the referenced file can be the same as SSLCertificateFile
        #   when the CA certificates are directly appended to the
server
        #   certificate for convinience.
        #SSLCertificateChainFile /etc/apache2/ssl.crt/server-ca.crt

        #   Certificate Authority (CA):
        #   Set the CA certificate verification path where to find CA
        #   certificates for client authentication or alternatively one
        #   huge file containing all of them (file must be PEM encoded)
        #   Note: Inside SSLCACertificatePath you need hash symlinks
        #         to point to the certificate files. Use the provided
        #         Makefile to update the hash symlinks after changes.
        #SSLCACertificatePath /etc/ssl/certs/
        #SSLCACertificateFile /etc/apache2/ssl.crt/ca-bundle.crt

        #   Certificate Revocation Lists (CRL):
        #   Set the CA revocation path where to find CA CRLs for client
        #   authentication or alternatively one huge file containing
all
        #   of them (file must be PEM encoded)
        #   Note: Inside SSLCARevocationPath you need hash symlinks
        #         to point to the certificate files. Use the provided
        #         Makefile to update the hash symlinks after changes.
        #SSLCARevocationPath /etc/apache2/ssl.crl/
        #SSLCARevocationFile /etc/apache2/ssl.crl/ca-bundle.crl

        #   Client Authentication (Type):
        #   Client certificate verification type and depth.  Types are
        #   none, optional, require and optional_no_ca.  Depth is a
        #   number which specifies how deeply to verify the certificate
        #   issuer chain before deciding the certificate is not valid.
        #SSLVerifyClient require
        #SSLVerifyDepth  10

        #   SSL Engine Options:
        #   Set various options for the SSL engine.
        #   o FakeBasicAuth:
        #     Translate the client X.509 into a Basic Authorisation.
This means that
        #     the standard Auth/DBMAuth methods can be used for access
control.  The
```

```
        #     user name is the `one line' version of the client's X.509
certificate.
        #     Note that no password is obtained from the user. Every
entry in the user
        #     file needs this password: `xxj31ZMTZzkVA'.
        #   o ExportCertData:
        #     This exports two additional environment variables:
SSL_CLIENT_CERT and
        #     SSL_SERVER_CERT. These contain the PEM-encoded
certificates of the
        #     server (always existing) and the client (only existing
when client
        #     authentication is used). This can be used to import the
certificates
        #     into CGI scripts.
        #   o StdEnvVars:
        #     This exports the standard SSL/TLS related `SSL_*'
environment variables.
        #     Per default this exportation is switched off for
performance reasons,
        #     because the extraction step is an expensive operation and
is usually
        #     useless for serving static content. So one usually
enables the
        #     exportation for CGI and SSI requests only.
        #   o OptRenegotiate:
        #     This enables optimized SSL connection renegotiation
handling when SSL
        #     directives are used in per-directory context.
        #SSLOptions +FakeBasicAuth +ExportCertData +StrictRequire
        <FilesMatch "\.(cgi|shtml|phtml|php)$">
                SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
                SSLOptions +StdEnvVars
        </Directory>

        #   SSL Protocol Adjustments:
        #   The safe and default but still SSL/TLS standard compliant
shutdown
        #   approach is that mod_ssl sends the close notify alert but
doesn't wait for
        #   the close notify alert from client. When you need a
different shutdown
        #   approach you can use one of the following variables:
        #   o ssl-unclean-shutdown:
        #     This forces an unclean shutdown when the connection is
closed, i.e. no
        #     SSL close notify alert is send or allowed to received.
This violates
        #     the SSL/TLS standard but is needed for some brain-dead
```

```
browsers. Use
        #     this when you receive I/O errors because of the standard
approach where
        #     mod_ssl sends the close notify alert.
        #   o ssl-accurate-shutdown:
        #     This forces an accurate shutdown when the connection is
closed, i.e. a
        #     SSL close notify alert is send and mod_ssl waits for the
close notify
        #     alert of the client. This is 100% SSL/TLS standard
compliant, but in
        #     practice often causes hanging connections with brain-dead
browsers. Use
        #     this only for browsers where you know that their SSL
implementation
        #     works correctly.
        #   Notice: Most problems of broken clients are also related to
the HTTP
        #   keep-alive facility, so you usually additionally want to
disable
        #   keep-alive for those clients, too. Use variable
"nokeepalive" for this.
        #   Similarly, one has to force some clients to use HTTP/1.0 to
workaround
        #   their broken HTTP/1.1 implementation. Use variables
"downgrade-1.0" and
        #   "force-response-1.0" for this.
        # BrowserMatch "MSIE [2-6]" \
        #        nokeepalive ssl-unclean-shutdown \
        #        downgrade-1.0 force-response-1.0

        # to improve tls/ssl security according to
https://ssl-config.mozilla.org/
        # enable HTTP/2, if available
        Protocols h2 http/1.1

        # HTTP Strict Transport Security (mod_headers is required)
(63072000 seconds)
        Header always set Strict-Transport-Security "max-age=63072000"

        # intermediate configuration
        SSLProtocol             all -SSLv3 -TLSv1 -TLSv1.1
        SSLCipherSuite          ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-
RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-
GCM-SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-
POLY1305:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384
        SSLHonorCipherOrder     off
        SSLSessionTickets       off
```

```
    </VirtualHost>

    SSLUseStapling On
    SSLStaplingCache "shmcb:logs/ssl_stapling(32768)"

</IfModule>


EOF
```

12. Enable our new Apache site configuration AND enable more ssl/tls related security (Apache) modules our ssl configuration file needs:

```
a2enmod ssl socache_shmcb rewrite headers
a2ensite awerner.myhome-server.de-ssl.conf
```

13. Restart Apache Service to activate new configuration:

```
systemctl restart apache2.service
systemctl status apache2.service
```

14. Test access via **https (Port 443)** on DokuWiki and see if the browser is satisfied with our SSL/TLS certificate:

```
firefox https://awerner.myhome-server.de &

# expected result:
#    plain DokuWiki Homepage shown.
#    No security warnings in browser!!
```

15. Finally we better make sure, that any incoming unencrypted http (port 80) request will be redirected to our https url, for better security. We prepared such a line in out http config already. We just need to “activate” it:

```
# always make a backup first!
cp -v /etc/apache2/sites-available/awerner.myhome-
server.de.conf{,.$(date +%F)}

#
# activate/uncomment the redirect (rewrite) statement to https
#
sed -i 's/^\([[:space:]]*\)#\(RewriteRule.*\)/\1\2/'
/etc/apache2/sites-available/awerner.myhome-server.de.conf
```

16. Again restart service and check status + check if browser now gets automatically redirected to **https://** , when we access the website using http:// protocol only:

```
systemctl restart apache2.service
systemctl status apache2.service

firefox http://awerner.myhome-server.de &
```

```
# expected result:
#    WATCH BROWSERS ADDRESS LINE!
#    we entered http:// , but browser
#    must now show a SECURED connection!
#    No security warnings in browser!!
#    Also we expect the website to appear just fine.
```

17. Our DokuWiki installation is complete.

Change Timezone:

For whatever reason it seems that UBUNTU Server 20 stopped asking us about what Timezone we are in, while installation. So it may or may not happen to you, that it comes up with the “etc/UTC” Timezone. Thats ok if you prefer to have it that way.

I prefer the local timezone. In my case its “Europe/Berlin”.

NOTE: This string is CASE SENSITIVE!!

1. To check which timezone is configured type:

```
timedatectl
```

2. To list all available timezones enter:

```
timedatectl list-timezones
```

3. To set/change your timezone type:

```
timedatectl set-timezone Europe/Berlin
```

4. check again by rebooting, then:

```
timedatectl
```

Enable NTP Timesynchronisation / Network Time (Client) :

Keeping time is often critical. Therefore its included and enabled by default on UBUNTU Server 20.04. However, you might want to change the NTP Server its using, in case you run your own NTP timeserver on your network or simply want to use another one. This is how i did it.

1. Backup the default configuration first:

```
cp -v /etc/systemd/timesyncd.conf{,.$(date +%F)}

# check status BEFORE
timedatectl show
```

```
timedatectl timesync-status


echo "NTP=your.prefered.ntp.time-source.host.net" >>
/etc/systemd/timesyncd.conf
systemctl restart systemd-timesyncd.service


# check status AFTER
timedatectl show
timedatectl timesync-status
```

2. **NOTE: *NTPSynchronized=yes*** means its working now.

Setting up 'unattended-upgrades' :

UBUNTU comes with some 'unattended-upgrades' partly enabled. But on some UBUNTU LXC containers there might it be disabled by default. So this is how i like my (non enterprise) UBUNTU to be handling automatic upgrades:

1. Install 'unattended-upgrades' it not already installed:

```
apt install -y unattended-upgrades
```

2. Replace the default configuration file that comes with 'unattended-upgrades':

```
cat << 'EOF' > /etc/apt/apt.conf.d/50unattended-upgrades
// Automatically upgrade packages from these (origin:archive) pairs
//
// Note that in Ubuntu security updates may pull in new dependencies
// from non-security sources (e.g. chromium). By allowing the release
// pocket these get automatically pulled in.
Unattended-Upgrade::Allowed-Origins {
    "${distro_id}:${distro_codename}";
    "${distro_id}:${distro_codename}-security";
    // Extended Security Maintenance; doesn't necessarily exist for
    // every release and this system may not have it installed, but if
    // available, the policy for updates is such that unattended-
upgrades
    // should also install from here by default.
    "${distro_id}ESMApps:${distro_codename}-apps-security";
    "${distro_id}ESM:${distro_codename}-infra-security";
    "${distro_id}:${distro_codename}-updates";
//  "${distro_id}:${distro_codename}-proposed";
//  "${distro_id}:${distro_codename}-backports";
};

// Python regular expressions, matching packages to exclude from
upgrading
Unattended-Upgrade::Package-Blacklist {
```

```
    // The following matches all packages starting with linux-
//  "linux-";

    // Use $ to explicitely define the end of a package name. Without
    // the $, "libc6" would match all of them.
//  "libc6$";
//  "libc6-dev$";
//  "libc6-i686$";

    // Special characters need escaping
//  "libstdc\+\+6$";

    // The following matches packages like xen-system-amd64, xen-
utils-4.1,
    // xenstore-utils and libxenstore3.0
//  "(lib)?xen(store)?";

    // For more information about Python regular expressions, see
    // https://docs.python.org/3/howto/regex.html
};

// This option controls whether the development release of Ubuntu will
be
// upgraded automatically. Valid values are "true", "false", and
"auto".
Unattended-Upgrade::DevRelease "auto";

// This option allows you to control if on a unclean dpkg exit
// unattended-upgrades will automatically run
//   dpkg --force-confold --configure -a
// The default is true, to ensure updates keep getting installed
Unattended-Upgrade::AutoFixInterruptedDpkg "true";

// Split the upgrade into the smallest possible chunks so that
// they can be interrupted with SIGTERM. This makes the upgrade
// a bit slower but it has the benefit that shutdown while a upgrade
// is running is possible (with a small delay)
Unattended-Upgrade::MinimalSteps "true";

// Install all updates when the machine is shutting down
// instead of doing it in the background while the machine is running.
// This will (obviously) make shutdown slower.
// Unattended-upgrades increases logind's InhibitDelayMaxSec to 30s.
// This allows more time for unattended-upgrades to shut down
gracefully
// or even install a few packages in InstallOnShutdown mode, but is
still a
// big step back from the 30 minutes allowed for InstallOnShutdown
previously.
// Users enabling InstallOnShutdown mode are advised to increase
```

```
// InhibitDelayMaxSec even further, possibly to 30 minutes.
//Unattended-Upgrade::InstallOnShutdown "false";

// Send email to this address for problems or packages upgrades
// If empty or unset then no email is sent, make sure that you
// have a working mail setup on your system. A package that provides
// 'mailx' must be installed. E.g. "user@example.com"
Unattended-Upgrade::Mail "root";

// Set this value to one of:
//    "always", "only-on-error" or "on-change"
// If this is not set, then any legacy MailOnlyOnError (boolean) value
// is used to chose between "only-on-error" and "on-change"
Unattended-Upgrade::MailReport "only-on-error";

// Remove unused automatically installed kernel-related packages
// (kernel images, kernel headers and kernel version locked tools).
//Unattended-Upgrade::Remove-Unused-Kernel-Packages "true";

// Do automatic removal of newly unused dependencies after the upgrade
Unattended-Upgrade::Remove-New-Unused-Dependencies "true";

// Do automatic removal of unused packages after the upgrade
// (equivalent to apt-get autoremove)
Unattended-Upgrade::Remove-Unused-Dependencies "true";

// Automatically reboot *WITHOUT CONFIRMATION* if
//  the file /var/run/reboot-required is found after the upgrade
Unattended-Upgrade::Automatic-Reboot "true";

// Automatically reboot even if there are users currently logged in
// when Unattended-Upgrade::Automatic-Reboot is set to true
Unattended-Upgrade::Automatic-Reboot-WithUsers "true";

// If automatic reboot is enabled and needed, reboot at the specific
// time instead of immediately
//  Default: "now"
//Unattended-Upgrade::Automatic-Reboot-Time "02:00";

// Use apt bandwidth limit feature, this example limits the download
// speed to 70kb/sec
//Acquire::http::Dl-Limit "70";

// Enable logging to syslog. Default is False
Unattended-Upgrade::SyslogEnable "true";

// Specify syslog facility. Default is daemon
// Unattended-Upgrade::SyslogFacility "daemon";

// Download and install upgrades only on AC power
// (i.e. skip or gracefully stop updates on battery)
```

```
// Unattended-Upgrade::OnlyOnACPower "true";

// Download and install upgrades only on non-metered connection
// (i.e. skip or gracefully stop updates on a metered connection)
// Unattended-Upgrade::Skip-Updates-On-Metered-Connections "true";

// Verbose logging
// Unattended-Upgrade::Verbose "false";

// Print debugging information both in unattended-upgrades and
// in unattended-upgrade-shutdown
// Unattended-Upgrade::Debug "false";

// Allow package downgrade if Pin-Priority exceeds 1000
// Unattended-Upgrade::Allow-downgrade "false";

// When APT fails to mark a package to be upgraded or installed try
adjusting
// candidates of related packages to help APT's resolver in finding a
solution
// where the package can be upgraded or installed.
// This is a workaround until APT's resolver is fixed to always find a
// solution if it exists. (See Debian bug #711128.)
// The fallback is enabled by default, except on Debian's sid release
because
// uninstallable packages are frequent there.
// Disabling the fallback speeds up unattended-upgrades when there are
// uninstallable packages at the expense of rarely keeping back
packages which
// could be upgraded or installed.
// Unattended-Upgrade::Allow-APT-Mark-Fallback "true";

EOF
```

3. Check and enable Service if nessesary:

```
systemctl status unattended-upgrades

systemctl enable --now unattended-upgrades
```

4. This should automatically update/upgrade the OS once a day including automatic reboot (instantly) if necessary. Since this is part of the **cron.daily** it's usually started between 0600-0700 UTC.
5. To test run it manually in verbose mode:

```
unattended-upgrades --verbose
```

Mailserver MTA installation and configuration for status mails:

Without any installed MTA local status mails or error messages stay within the local server and cannot be externaly retreived.

Because i like my Linux Servers to be able to send and receive Emails via SMTP i usually install the Postfix MTA and some additional IMAP/IMAPS/POP3/POP3S daemons, so a server can send and receive emails (for a local user or the ROOT user) and/or received mails can be retrieved via a standard mailclient like Thunderbird, Gmail or similar.

This is how i usually do it:

1. Since Google Mail / GMAIL let us down on **SMTP AUTH** and in 2022 **oauth2** is not yet supported with postfix out of the box, we need to use a more classic free email provider service, that still supports plain simple **SMTP AUTH** to send at least “some” mails per month. Im my case i use the free service of https://www.sendinblue.com/. So get your own free account there. Good for up to 300 mail per month. Should be plenty for status mails.
2. Install Postfix MTA and Mailutils:

   ```
   apt install -y postfix mailutils libsasl2-modules sasl2-bin swaks vim
   ```

   Configure it using the text UI with the default values, even they are wrong for now. We will reconfigure it shortly with even more parameters.

3. Reconfigure Postfix using the **postconf** command like this:

   ```
   postconf "mynetworks=127.0.0.0/8 [::1]/128"
   postconf "mydestination = $(hostname -s),$(hostname -f),$(hostname -
   s).local,$(hostname -s).localdomain,127.0.0.1,[::1]"
   postconf "relayhost=[smtp-relay.sendinblue.com]:587"
   postconf "recipient_delimiter=+"
   postconf "mailbox_size_limit = 51000000"
   postconf "inet_protocols = all"
   postconf "inet_interfaces = all"

   postconf 'smtp_use_tls = yes'
   postconf 'smtp_tls_security_level = encrypt'
   postconf 'smtp_tls_CApath = /etc/ssl/certs/'

   postconf 'smtp_sasl_auth_enable = yes'
   postconf 'smtp_sasl_security_options ='
   postconf 'smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd'
   postconf 'smtp_sasl_mechanism_filter ='

   systemctl restart postfix
   systemctl status postfix
   ```

4. Store your SMTP AUTH login credentials for the mail relay:

```
# NOTE THE LEADING SPACE in the next line!
# Prevents it from being saved in bash history
 echo '[smtp-relay.sendinblue.com]:587
axel.werner.1973@gmail.com:seecreetPazzword' > /etc/postfix/sasl_passwd
chmod 600 /etc/postfix/sasl_passwd

postmap hash:/etc/postfix/sasl_passwd
```

5. Set the destination email address for the **root** account:

```
echo "root: axel.werner.1973@gmail.com" >> /etc/aliases

newaliases
```

   - You can check the outbound mailqueue with mailq to see if the mail is still pending. Or you can watch the logs “live” like this:

```
tail -n0 -f /var/log/messages /var/log/syslog /var/log/mail.* &
```

6. Test sending Email to local recipient:

```
echo "local testmail to ROOT" | mail -s "test email from $HOSTNAME to
root user" root
```

   Test Email should be found in your mailbox.

7. Test sending Email to external recipient:

```
echo "external testmail to gmail user" | mail -s "test email from
$HOSTNAME to gmail user" axel.werner.1973@gmail.com
```

   Another Email should be found in your mailbox.

8. 

:

```
FIXME
```

9. 

:

```
FIXME
```

10. 

:

```
FIXME
```

11. 

:

```
FIXME
```

12. Fix Me!:

```
FIXME
```

FIXME:

Fix Me!

FIXME:

Fix Me!

FIXME:

Fix Me!

linux, ubuntu, server, ipv4, ipv6, static, networking, bridge, apache, router, dhcp, dns, ddns, apache, ssl, tls, letsencrypt, ca, certificates, firewall, iptables, hostfirewall, docker