# OpenSUSE

imx6, software, opensuse, ignition

## Overview

| | |
|---|---|
| **Name:** | **OpenSUSE** |
| **Picture:** |  |
| **Active** | Yes |
| **Purpose** | General Linux |
| **Ignition install** | Yes |
| **Compatibility** | All IMX6 MicroSoms |
| **More information** | |
| **Maintained by** | Unknown |
| **Forum** | Solid-Run Forum |
| **Website** | opensuse.org |

## Description

OpenSUSE is a popular Desktop Distribution featuring fully integrated Gnome and KDE desktops, as well as partly integrated XFCE, E17 and more. This page gives information on how to install openSUSE on the Cubox-i minicomputer as well as details on the current status. Be sure to check this forum discussion for further Information.

## Tumbleweed

### Installation

### Manual Installation

OpenSUSE Tumbleweed is supplied in 5 image flavors:

- JeOS - a minimal install (useful to set up a server)
- E20 - featuring the Enlightenment desktop
- XFCE - featuring the XFCE desktop
- LXQt - featuring the LXQt desktop
- KDE - featuring the KDE desktop
- X11 - featuring a basic X.org environment

In order to install them, as **root** extract the image onto your SD card (**replace sdX with the device name of your SD card**).

```
xzcat [image].xz | dd bs=4M of=/dev/sdX iflag=fullblock oflag=direct; sync
```

**WARNING:** all previous data on the SD card will be lost. Check first if the device you have selected is really your SD card!

The command `fdisk -l` may help

After that, you are ready to boot. Insert the SD card with the openSUSE image into your board and optionally connect the board to your PC via serial port (helpful for debugging; USB-TTL serial cable needed).

For serial port connection, under Linux you can use `screen`.

```
sudo screen /dev/ttyUSB0 115200
```

If needed, connect the board to your monitor (via DVI/HDMI), and finally power on the board.

Ethernet is configured to request an IP via DHCP, so check your DHCP server for the board IP if used. Otherwise, to check the IP you may use

```
wicked ifstatus all
```

Default login is *user*→**root**, *password*→**linux**. It works on serial console, via *ssh* (`ssh XXX.XXX.XXX.XXX -l root` where XXX.XXX.XXX.XXX is the local IP address of your board), via GUI.

For security reasons, you may decide to have *sshd* listen to a non-standard port and forbid **root** login. To do so, edit `/etc/ssh/sshd_config`, in particular the options *Port XXXX* where 'XXXX' is the port number and *PermitRootLogin yes/no* set to 'no'.

---

#### Boot from external hard drive

The easiest way to do that is probably to connect the hard drive and the SD card (already gone through the first boot) to your PC at the same time. **Do not mount them**, just check the device name with

```
fdisk -l
```

Let's assume that /dev/sdb represents your hard drive and /dev/sdc the SD card. Then execute

```
sudo dd bs=4M if=/dev/sdc of=/dev/sdb iflag=fullblock oflag=direct
```

**WARNING:** dd will wipe all data on the output device, so act carefully!

Once the operation is complete, just use any partition manager to expand the root partition to your preferred size and eventually create a separate home partition.

After partitioning, edit `/etc/fstab` according to your effective situation. For example:

```
#/dev/disk/by-id/mmc-00000_0xa12853e0-part2 / ext4 noatime,nobarrier 1 1
```

```
#/dev/disk/by-id/mmc-00000_0xa12853e0-part1 /boot ext3 defaults 1 2
#/dev/disk/by-id/mmc-00000_0xa12853e0-part3 swap swap defaults 0 0

/dev/sda2 / ext4 noatime,nobarrier 1 1
/dev/sda1 /boot ext3 defaults 1 2
/dev/sda3 swap swap defaults 0 0
/dev/sda5 /home ext4 defaults 1 2
```

Now you should make u-boot point to the external hard drive at boot time. To do so, connect the SD card to your PC and mount the BOOT partition. Open `boot.script` with your favorite text editor and change the `setenv bootargs` line from something like

```
setenv bootargs "root=/dev/disk/by-id/mmc-00000_0xa12853e0-part2
loader=uboot disk=/dev/disk/by-id/mmc-00000_0xa12853e0 resume=/dev/disk/by-
id/mmc-00000_0xa12853e0-part3   loglevel=3 splash=silent plymouth.enable=0
console=ttymxc0,115200n8 ${append}"
```

to

```
setenv bootargs "root=/dev/sda2 loader=uboot disk=/dev/sda resume=/dev/sda3
loglevel=3 splash=silent plymouth.enable=0  console=ttymxc0,115200n8
${append}"
```

where *sda2* is the root partition, *sda* is the external device name, *sda3* is the swap partition.

Convert `boot.script` into `boot.scr` with *mkimage* (in openSUSE this tool is available through the *u-boot-tools* package.

```
sudo mkimage -A arm -O linux -T script -C none -a 0 -e 0 -d
/path/to/boot.script /path/to/boot.scr
```

At this point you are ready. Connect the external hard drive to your cubox-i, insert the SD card and plug the power.

Should you not want wish to have a separate partition for `/boot`, there is an alternative method. First prepare the hard drive with all partitions defined and formatted, then connect it to your cubox-i and power this on. To copy the root file system, follow these steps (assuming that it must be copied to `/dev/sda1`:

```
mkdir /tmp/source /tmp/target
sudo mount /dev/mmcblk0p2 /tmp/source
sudo mount /dev/mmcblk0p1 /tmp/source/boot
sudo mount /dev/sda1 /tmp/target
(cd /tmp/source; sudo tar --backup -c *) |sudo tar -C /tmp/target -xv
```

After that, just follow the `/etc/fstab` and `mkimage` steps described above.

---

**Packman**

Packman offers various additional packages for openSUSE, especially but not limited to multimedia related applications and libraries that are on the openSUSE Build Service application blacklist. It's the largest external repository of openSUSE packages. Packman for ARM is comprised of the following three repositories:

- Essentials: provides codecs and audio and video player applications, to fulfill the most essential needs
- Multimedia: contains many more multimedia related applications
- Extra: additional non multimedia related applications, mostly network related

To add them, follow the instruction below:

```
sudo zypper ar -f
http://pmbs.links2linux.de:82/Essentials/openSUSE_Factory_ARM/ packman-
essentials
sudo zypper ar -f
http://pmbs.links2linux.de:82/Multimedia/openSUSE_Factory_ARM/ packman-
multimedia
sudo zypper ar -f http://pmbs.links2linux.de:82/Extra/openSUSE_Factory_ARM/
packman-extra
```

Then you will probably want to replace all currently installed packages with the versions provided by packman.

```
sudo zypper dup --from='packman-essentials'
sudo zypper dup --from='packman-multimedia'
sudo zypper dup --from='packman-extra'
```

## JeOS Administration Via Webmin

### Installation

To install Webmin, dowload the `tar` archive and extract its content.

```
wget http://www.webmin.com/download/webmin-current.tar.gz
tar -xvf webmin-current.tar.gz
```

Then move the content of the `webmin-${ver}` to `/usr/share/webmin` and launch the installation procedure. Just substitute 'X.Y.Z' with the actual webmin version.

```
ver="X.Y.Z"
sudo mv webmin-${ver} -T /usr/share/webmin
sudo chown -R root:root /usr/share/webmin
sudo sh /usr/share/webmin/setup.sh
```

The distro is not automatically detected, so choose SuSE Linux and your preferred setup.

### Configuration of Apache for Internet access

If you are accessing your Webmin server over the Internet, then you should definitely consider using SSL to prevent an attacker capturing your Webmin password. To do so, the simplest way install `perl-Net-SSLeay` and then enable SSL via the SSL Encryption of the Webmin configuration dashboard. By default Webmin uses a self-signed certificate located in `/etc/webmin/miniserv.pem`. We will be using this.

First install `apache`, make it start at boot time and then enable the necessary additional modules.

```
sudo zypper in apache2
sudo systemctl enable apache2
sudo a2enmod proxy
sudo a2enmod proxy_http
```

Then make it listen also on port 443. To do so, just add SSL to the Apache server flags and restart the server.

```
sudo vi /etc/sysconfig/apache2

   APACHE_SERVER_FLAGS="SSL"

sudo systemctl restart apache2
```

You can check whether Apache is listening on port 443 with

```
netstat -ln | grep -E ':80|443'
```

You should get this result:

```
tcp6       0      0 :::80                   :::*                    LISTEN
tcp6       0      0 :::443                  :::*                    LISTEN
```

### Webmin In A Virtual Host Via A Proxy

The virtual host configuration should look more or less like this:

```
<VirtualHost *:80>
    ServerName webmin.domain.com
        ServerAlias webmin.localdomain
    Redirect permanent / https://webmin.domain.com/
</VirtualHost>

<VirtualHost *:443>

        ServerName webmin.domain.com
        ServerAlias webmin.localdomain
        #   SSL Engine Switch:
        #   Enable/Disable SSL for this virtual host.
        SSLEngine on
```

```
        #   You can use per vhost certificates if SNI is supported.
        SSLCertificateFile /etc/webmin/miniserv.pem
        SSLCertificateKeyFile /etc/webmin/miniserv.pem

        <Proxy *>
                Require all granted
                #   For apache version 2.2 comment out the line above
                #   and uncomment the two lines below
                #Order deny,allow
                #Allow from all
        </Proxy>

        SSLProxyEngine On
        SSLProxyVerify none
        #   Should you have issues with the certificate Common Name and Name
checks,
        #   because webmin server uses different values than those of your
webserver,
        #   you may decide to disable them (you may even decide to switch
off the
        #   expiration check, in some cases)
        SSLProxyCheckPeerCN off
        SSLProxyCheckPeerName off
        #SSLProxyCheckPeerExpire off
        ProxyPass / https://localhost:10000/
        ProxyPassReverse / https://localhost:10000/
</VirtualHost>
```

Last but not least, you should add your server name (i.e.`webmin.domain.com`) and server alias (`webmin.localdomain`) as trusted referrers by adding this line to `/etc/webmin/config`.

```
referer=webmin.domain.com webmin.localdomain
```

**Webmin In A Sub-Directory Via A Proxy**

Alternatively, if you prefer to access via a sub-directory of your host, just add to the host .conf file the same directives above with proper `ServerName` value (e.g. `yourdomain.com`) and modified `ProxyPass` directives as such:

```
ServerName domain.com

ProxyPass /webmin https://localhost:10000/
ProxyPassReverse /webmin https://localhost:10000/
ProxyPass /webmin/ https://localhost:10000/
ProxyPassReverse /webmin/ https://localhost:10000/
```

To complete the configuration, you should modify `/etc/webmin/config` as follows:

```
referer=domain.com
webprefix=/webmin
```

```
webprefixnoredir=1
```

Finally, add the line `cookiepath=/webmin` to `/etc/webmin/miniserv.conf` and restart webmin by running `/etc/webmin/restart`.

## Setup Of An SFTP Server

The easiest way to setup an SFTP server is to make use of the option *ChrootDirectory* of `sshd`. To setup the chroot directory, you need to edit `/etc/ssh/sshd_config` and first of all make sure to use the internal SFTP server. Comment out any reference to `/usr/lib/ssh/sftp-server`.

```
# override default of no subsystems
#Subsystem  sftp    /usr/lib/ssh/sftp-server
Subsystem   sftp    internal-sftp
```

In order to limit sftp access to certain users, you could make use of a “Match rule block”. Any setting located after the match rule will apply only to that user/group (so make sure it is at the bottom of the configuration file). For instance, a match rule block could look like this:

```
Match group sftponly
   ForceCommand internal-sftp
   ChrootDirectory /home/%u
   X11Forwarding no
   AllowTcpForwarding no
```

In the example above:

- All users belonging to the *sftponly* group will be able to connect to your host via SFTP only.
- The `ForceCommand` option must always be the first of the block and makes sure that the restricted users can only use the SSHD for SFTP, so they don't have the possibility of opening a regular SSH session. This means that if your user is caught by the rule, you won't be able to login via ssh and SSHD will return a message like:

```
This service allows sftp connections only.
Connection to host closed.
```

- The `ChrootDirectory` option sets the new root directory for this user. %u can be used as username replacement and %h can be used for path to home directory. The use of %u is strongly recommended, especially if you are going to create separate users only for SFTP, without local login.
- `X11Forwarding` and `AllowTcpForwarding` are both set to no, to prevent the user from forwarding ports or starting remote X applications.

Save and reload the configuration with:

```
sudo rcsshd reload
```

The ChrootDirectory must be owned by the SFTP user and its permissions must be set to 700 (that is the SFTP user can read, write and execute; anybody else has no permissions) or 750 (that is the SFTP user can read, write and execute; group can read and execute; others have no permissions). Its

parent directory must be owned by *root*.

For further information, you can refer to https://en.wikibooks.org/wiki/OpenSSH/Cookbook/File_Transfer_with_SFTP.

**Example 1 (regular users with access to regular directories)**

Let's say you have a system with *user1*, *user2*, *user3*, etc. If you want to enable SFTP for them with the ability to surf a dedicated `/sftp/user_name` directory, follow these steps:

```
sudo groupadd sftponly                  #create the sftponly group
sudo usermod -aG sftponly user1         #add user1 to the sftponly group.
Repeat for all necessary users
sudo mkdir -p /sftp/user1               #create /sftp/user1 directory. Repeat
for all necessary users
sudo chown user1:sftponly /sftp/user1   #change ownership of user1's sftp
directory. Repeat for all necessary users
sudo chmod 700 /sftp/user1              #change permissions of user1's sftp
directory. Repeat for all necessary users
```

Then edit `/etc/ssh/sshd_config` setting the right sftp subsystem (see above) and defining the right “Match rule block”.

```
Match group sftponly
   ForceCommand internal-sftp
   ChrootDirectory /sftp/%u
   X11Forwarding no
   AllowTcpForwarding no
```

**Example 2 (alternate SFTP user with access to a veracrypt volume)**

Let's say you have a `veracrypt` encrypted volume that you want to access via SFTP but at the same time you don't want to give up the possibility to login via ssh. In this case you may decide to create an alternate user specific for the purpose.

```
sudo useradd -M -g sftponly -s /sbin/nologin sftp_user    #create the
sftp_user belonging to the sftponly group,
                                                          #without a home
directory and with no login permissions
sudo passwd sftp_user                                     #assign a password
to the newly created user
```

Then, if already mounted, unmount the veracrypt archive with `veracrypt -d /path/to/file/volume_file` and, if necessary change the *volume file* permissions and/or location so that it becomes accessible to *sftp_user*. You can finally mount it to your preferred location (`/sftp/sftp_user` for example) as *sftp_user* (note: the actual veracrypt command line options can vary depending on how you created the encrypted volume).

```
sudo -u sftp_user veracrypt -k "" --pim 0 --protect-hidden=no --fs-
options=iocharset=utf8 /path/to/file/volume_file /sftp/sftp_user
```

If you want to pass to *veracrypt* also the volume password with the `-p "PASSWORD"`option, it would be safer not to log the command in the bash history. In openSUSE the bash variable *HISTCONTROL* is set by default to *ignoreboth*, which tells bash not to log commands that start with spaces or repeated commands. If you changed that behaviour, you can obtain the same result by appending

```
; history -d $((HISTCMD-1))
```

to the command line. For example:

```
veracrypt [LIST OF OPTIONS]; history -d $((HISTCMD-1))
```

.

Veracrypt mounts the volume with permissions set to 700 and ownership set to the user who launched to command (in the example above we used *sudo -u* to switch user), so it has all required features to be an SFTP ChrootDirectory.

So at this point we just need to edit `/etc/ssh/sshd_config` setting the right sftp subsystem (see above) and defining the right "Match rule block".

```
Match group sftponly
   ForceCommand internal-sftp
   ChrootDirectory /sftp/%u
   X11Forwarding no
   AllowTcpForwarding no
```

Should you require to perform a `fsck` on the veracrypt volume, just unmount the volume with

```
veracrypt -d /path/to/file/volume_file
```

Then with the following command you can decrypt it without mounting it.

```
veracrypt -k "" --pim 0 --protect-hidden=no /path/to/file/volume_file --
filesystem=none
```

Find out where the volume is mapped with `veracrypt -l`. You should get a similar output:

```
1: /path/to/file/volume_file /dev/mapper/veracrypt1
```

Now you have all required information to run `fsck`.

```
sudo fsck -f /dev/mapper/veracrypt1
```

## Setup Of A Nextcloud Server

### Installation

Nextcloud is available in the distribution repository. The installation of the nextcloud package will pull

in Apache, MariaDB and a series of php modules. Not all php modules for optional features are installed together with the nextcloud package, so you can add them if needed.

```
sudo zypper in nextcloud
sudo zypper in php-opcache php-dom php-bz2 php-intl php-exif php-APCu php-
redis redis ffmpeg php-pcntl
```

Then enable MariaDB at boot and start the service.

```
sudo systemctl enable mysql
sudo systemctl start mysql
```

The `root` password is empty by default, so you should set one immediately.

```
mysqladmin -u root password new_password   # where new_password represents
the password you have chosen
```

Next step, you need to create a MariaDB database that can be managed by a Nextcloud-specific user.

```
mysql -u root -p   # you will be prompted for your MariaDB root password

CREATE DATABASE nextcloud_db;
# where nextcloud_db is the name of the database
GRANT ALL ON nextcloudb.* TO nc_user@localhost IDENTIFIED BY 'password';
# where nc_user is the name of Nextcloud-specific
# user and 'password' is the password you would
# like to assign to it
```

Then you can pass to fine-tuning `php.ini` located in `/etc/php7/apache2/`. For example, you may decide to change the following default values.

```
post_max_size = 8M
upload_max_filesize = 2M
max_file_uploads = 20
max_input_time = 60
max_execution_time = 30
session.gc_maxlifetime = 1440
memory_limit = 128M
```

For example:

```
post_max_size = 50G
upload_max_filesize = 25G
max_file_uploads = 200
max_input_time = 3600
max_execution_time = 3600
session.gc_maxlifetime = 3600
memory_limit = 512M
```

Also make sure you enable OPcache in `php.ini`. Please note: all options are already there but may

have been commented out with a ; or may be set to a different value:

```
opcache.enable=1
opcache.enable_cli=1
opcache.interned_strings_buffer=8
opcache.max_accelerated_files=10000
opcache.memory_consumption=128
opcache.save_comments=1
opcache.revalidate_freq=1
```

Finally, setup Apache. First of all, enable the required modules.

```
sudo a2enmod php7
sudo a2enmod rewrite
sudo a2enmod headers
sudo a2enmod env
sudo a2enmod dir
sudo a2enmod mime
sudo a2enmod ssl
```

Then prepare the server directives. Below you can find an example where Nextcloud is accessed via `www.domain.com/nextcloud`. SSL encryption is enforced for security reasons.

```
<VirtualHost *:80>
    ServerName www.domain.com
    ServerAlias domain.com
    Redirect permanent / https://www.domain.com/
</VirtualHost>

<VirtualHost *:443>

    ServerName www.domain.com
    ServerAlias domain.com
    DocumentRoot "/srv/www/htdocs"
    #   SSL Engine Switch:
    #   Enable/Disable SSL for this virtual host.
    SSLEngine on

    #   You can use per vhost certificates if SNI is supported.
    SSLCertificateFile /path/to/certificate/certificate.pem
    SSLCertificateKeyFile /path/to/key/privkey.pem
        Alias /nextcloud "/srv/www/htdocs/nextcloud/"
        <Directory "/srv/www/htdocs/nextcloud">
        Options +FollowSymLinks
        AllowOverride All

        <IfModule mod_dav.c>
        Dav off
        </IfModule>

        <IfModule mod_headers.c>
```

```
        Header always set Strict-Transport-Security "max-age=15768000;
includeSubDomains; preload"
        </IfModule>
        SetEnv HOME /srv/www/htdocs/nextcloud
        SetEnv HTTP_HOME /srv/www/htdocs/nextcloud
        </Directory>

        <Directory "/srv/www/htdocs/nextcloud/data/">
          # just in case if .htaccess gets disabled
         Require all denied
        </Directory>
</VirtualHost>
```

Once done, add your domain(s) to the list of trusted domains by adding the following to the config array in `/srv/www/htdocs/nextcloud/config/config.php`.

```
'trusted_domains' =>
  array (
    0 => 'www.domain.com',
    1 => 'www.otherdomain.com',
  ),
```

Then enable Apache at boot and start the service.

```
sudo systemctl enable apache
sudo systemctl start apache
```

At this point you are almost done. Now that everything is configured on the server, open a browser and visit https://www.domain.com/nextcloud. You will be prompted to create an admin account. In addition you will need to fill in the details for the database. In our example:

- the database user is *nc_user*
- the password is the one you created with the user
- the host and port are *localhost:3306* (unless you defined a different port for `MariaDB`)
- the database name is *nextcloudb*

Click Finish Installation and you have a working installation of Nextcloud.

### Cache optimization

Since you will most likely use your Cubox-i just for a private home server it is not necessary to setup `Memcached`, the best option is to configure APCu for local data caching and `Redis` to cache transactional file locking. Let us first setup `redis` using its configuration example as a guide.

```
sudo cp /etc/redis/default.conf.example /etc/redis/nextcloud.conf
sudo chown root:redis /etc/redis/nextcloud.conf
```

Then edit `/etc/redis/nextcloud.conf` as follows.

```
unixsocketperm 770 # Uncomment this line
pidfile /var/run/redis/nextcloud.pid
logfile /var/log/redis/nextcloud.log
dir /var/lib/redis/nextcloud/
daemonize no
port 6379 # Each instance should be running on a different port.
```

Then you need to create the database directory, add `wwwrun` to the `redis` group and enable/start `redis`.

```
sudo install -d -o redis -g redis -m 0750 /var/lib/redis/nextcloud
sudo usermod -a -G redis wwwrun
sudo systemctl enable redis@nextcloud
sudo systemctl start redis@nextcloud
```

Then add the following to the config array in `/srv/www/htdocs/nextcloud/config/config.php`.

```
'memcache.local' => '\\OC\\Memcache\\APCu',
'memcache.locking' => '\\OC\\Memcache\\Redis',
'redis' =>
array (
  'host' => 'localhost',
  'port' => 6379,
),
```

Finally restart Apache.

```
sudo systemctl restart apache
```

**Mount via davfs2**

Nextcloud supports *Web Distributed Authoring and Versioning* protocol and a WebDAV share can be mounted vis `davfs2`. `davfs2` is indeed a FUSE file system driver that allows you to mount a WebDAV server as a local file system, like a disk drive. This way applications can access resources on a Web server without knowing anything about HTTP or WebDAV. `davfs2` is not provided by the standard repositories, so you should first enable the `filesystems` repository hosted at the openSUSE Build Service (OBS). Then refresh the repos and finally install *davfs2*.

```
sudo zypper ar -f
http://download.opensuse.org/repositories/filesystems/openSUSE_Tumbleweed/
filesystems
sudo zypper ref
sudo zypper in davfs2
```

Use `mount` to mount the WebDAV share.

```
sudo mount -t davfs -o noexec
```

```
https://www.domain.com/nextcloud/remote.php/webdav/ /mnt/dav/
```

You will be prompted for user id and password.

```
Please enter the username to authenticate with server
https://www.domain.com/nextcloud/remote.php/webdav/ or hit enter for none.
  Username: nextcloud_user
Please enter the password to authenticate user nextcloud_user with server
https://www.domain.com/nextcloud/remote.php/webdav/ or hit enter for none.
  Password:
/sbin/mount.davfs: warning: the server does not support lock
```

Use `umount` to unmount the WebDAV share.

```
sudo umount /mnt/dav
/sbin/umount.davfs: waiting while mount.davfs (pid XXXX) synchronizes the
cache .. OK
```

**Mount via fstab**

To mount via `fstab`, just add the following line to `/etc/fstab`:

```
https://www.domain.com/nextcloud/remote.php/webdav/ /mnt/dav/ davfs
user,rw,noauto 0 0
```

With the `noauto` option the share will be mount manually, whereas with the `user` option you allow normal users to mount it. At this point you can use `mount /mnt/dav/` without `sudo`, but be aware: a normal user must also belong to the *davfs2* group for the command to be successful. You can achieve that by issuing the following command:

```
sudo usermod -a -G davfs2 username
```

Otherwise, you will get the following error message:

```
/sbin/mount.davfs: user username must be member of group davfs2
```

You can restrict mounting permissions to a specific user by appending the `uid=` and `gid=` options to the `fstab` mount directive. For example:

```
https://www.domain.com/nextcloud/remote.php/webdav/ /mnt/dav/ davfs
user,rw,noauto,uid=user_id,gid=group_id 0 0
```

When mounting the WebDAV share as normal user, you could get the following error message:

```
/sbin/mount.davfs: program is not setuid root
```

`/sbin/mount.davfs` is just a symbolic link to `/usr/sbin/mount.davfs`. You can set the SUID bit via `chmod`.

```
sudo chmod u+s /usr/sbin/mount.davfs
```

If you prefer not to be prompted for user id and password, you can store such information in `/etc/davfs2/secrets` (for system-wide configuration) or in `~/.davfs2/secrets` (for user-specific configuration. Create it, if missing). Just add the following line:

```
https://www.domain.com/nextcloud/remote.php/webdav/ nextcloud_user password
```

#### Issues with certificates

If the common name of the SSL certificate you are using does not match that of your domain (e.g. because you are connecting via loopback interface 127.0.0.1 or via a private network IP address) the mount operation will require additional input:

```
/sbin/mount.davfs: the server certificate does not match the server name
[...]
Accept certificate for this session? [y,N] y
```

To fix it, edit `/etc/davfs2/davfs2.conf` (for system-wide configuration) or `~/.davfs2/davfs2.conf` (for user-specific configuration. Create it, if missing) add the following line:

```
trust_server_cert        /etc/davfs2/certs/my_cert.pem
```

Then copy your certificate to `/etc/davfs2/certs/`. If you are using a self-signed certificate, and consequently the certification authority is not trusted, you get a different message:

```
/sbin/mount.davfs: the server certificate is not trusted
[...]
Accept certificate for this session? [y,N] y
```

To fix it, follow the instructions above, but use the `trust_ca_cert` option, instead.

```
trust_ca_cert        /etc/davfs2/certs/my_cert.pem
```

## Setup Of A Mail Server (using Postfix, Dovecot and MariaDB)

#### Prerequisites

For this tutorial it is assumed that you own a root domain (`yourdomain.com`), a subdomain (FQDN) to be associated to the mail server (`mail.yourdomain.com`) and that you have set up an MX record pointing to the aforementioned subdomain. First of all, install all necessary packages.

```
zypper in postfix postfix-mysql dovecot dovecot-backend-mysql mariadb
mariadb-client
```

#### Preparation of MariaDB database

After you have set up MariaDB following the instructions in the openSUSE wiki, you can create the

*mailserver* database.

```
mysqladmin -p create mailserver
```

Then you need to create a MariaDB user able to access the *mailserver* database. To do that, log in as MariaDB *root* user.

```
mysql -u root -p
```

After entering the password you should get the MariaDB prompt.

```
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is XXX
Server version: 10.1.25-MariaDB openSUSE package

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]>
```

Create the *mailuser* with the following command:

```
MariaDB [(none)]> GRANT SELECT ON mailserver.* TO mailuser@localhost
IDENTIFIED BY 'mailpassword';
```

Then update the privileges.

```
MariaDB [(none)]> FLUSH PRIVILEGES;
```

The next step is to create the necessary tables. The first table, named `virtual_domains` contains all authorized domains.

```
CREATE TABLE `mailserver`.`virtual_domains` (
`id`  INT NOT NULL AUTO_INCREMENT,
`name` VARCHAR(50) NOT NULL,
PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The second one, named `virtual_users`, contains the list of email addresses each associated with a password and a domain.

```
CREATE TABLE `mailserver`.`virtual_users` (
`id` INT NOT NULL AUTO_INCREMENT,
`domain_id` INT NOT NULL,
`password` VARCHAR(106) NOT NULL,
`email` VARCHAR(120) NOT NULL,
PRIMARY KEY (`id`),
UNIQUE KEY `email` (`email`),
```

```
FOREIGN KEY (domain_id) REFERENCES `mailserver`.`virtual_domains`(id) ON
DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

The third one, named `virtual_aliases`, contains the aliases of each mail user.

```
CREATE TABLE `mailserver`.`virtual_aliases` (
`id` INT NOT NULL AUTO_INCREMENT,
`domain_id` INT NOT NULL,
`source` varchar(100) NOT NULL,
`destination` varchar(100) NOT NULL,
PRIMARY KEY (`id`),
FOREIGN KEY (domain_id) REFERENCES `mailserver`.`virtual_domains`(id) ON
DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Now you are ready to fill in the tables with data. First insert the root domain and the subdomain into the `virtual_domains` table.

```
INSERT INTO `mailserver`.`virtual_domains`
(`id` ,`name`)
VALUES
('1', 'yourdomain.com'),
('2', 'mail.yourdomain.com');
```

Then insert the email addresses associated to each domain.

```
INSERT INTO `servermail`.`virtual_users`
(`id`, `domain_id`, `password` , `email`)
VALUES
('1', '1', ENCRYPT('firstpassword', CONCAT('$6$', SUBSTRING(SHA(RAND()),
-16))), 'email1@yourdomain.com'),
('2', '1', ENCRYPT('secondpassword', CONCAT('$6$', SUBSTRING(SHA(RAND()),
-16))), 'email2@yourdomain.com'),
('3', '2', ENCRYPT('firstpassword', CONCAT('$6$', SUBSTRING(SHA(RAND()),
-16))), 'email1@mail.yourdomain.com'),
('4', '2', ENCRYPT('secondpassword', CONCAT('$6$', SUBSTRING(SHA(RAND()),
-16))), 'email2@mail.yourdomain.com');
```

Finally insert the alias to virtual user mapping into the `virtual_aliases` table.

```
INSERT INTO `servermail`.`virtual_aliases`
(`id`, `domain_id`, `source`, `destination`)
VALUES
('1', '1', 'alias@yourdomain.com', 'email1@yourdomain.com'),
('1', '2', 'alias@mail.yourdomain.com', 'email1@mail.yourdomain.com');
```

After that your database is ready and you can quit.

```
MariaDB [(none)]> QUIT;
```

Enable MariaDB at boot with

```
sudo systemctl enable mysql
```

### Postfix setup

Just to be on the safe side, back up the original configuration.

```
sudo cp /etc/postfix/main.cf /etc/postfix/main.cf.orig
sudo cp /etc/postfix/master.cf /etc/postfix/master.cf.orig
```

```
inet_interfaces = all
inet_protocols = ipv4
myhostname = mail.yourdomain.com
mydomain = yourdomain.com
myorigin = $mydomain
mydestination = localhost
smtpd_tls_security_level = may
smtpd_tls_cert_file = /path/to/your/cert/yourcert.pem
smtpd_tls_key_file = /path/to/your/privkey/yourprivkey.pem
smtpd_use_tls=yes
smtpd_tls_auth_only = yes
smtpd_sasl_type = dovecot
smtpd_sasl_path = private/auth
smtpd_sasl_auth_enable = yes
smtpd_recipient_restrictions =
permit_sasl_authenticated,permit_mynetworks,reject_unauth_destination
```

Then we need to tell `postfix` to deliver the mail to `dovecot` for all virtual domains, users and aliases listed inside the MariaDB tables. The parameters to get the information from the database are stored in separated configuration files.

```
virtual_transport = lmtp:unix:private/dovecot-lmtp
virtual_mailbox_domains = mysql:/etc/postfix/mysql_virtual_domains_maps.cf
virtual_mailbox_maps = mysql:/etc/postfix/mysql_virtual_mailbox_maps.cf
virtual_alias_maps = mysql:/etc/postfix/mysql_virtual_alias_maps.cf
```

Now edit `/etc/postfix/mysql_virtual_domains_maps.cf`, `/etc/postfix/mysql_virtual_mailbox_maps.cf`, and `/etc/postfix/mysql_virtual_alias_maps.cf` so that they have the following content.

```
sudo vi /etc/postfix/mysql_virtual_domains_maps.cf

user = mailuser
password = mailpassword
hosts = localhost
dbname = mailserver
query = SELECT 1 FROM virtual_domains WHERE name='%s'
```

```
sudo vi /etc/postfix/mysql_virtual_mailbox_maps.cf

user = mailuser
password = mailpassword
hosts = localhost
dbname = mailserver
query = SELECT 1 FROM virtual_users WHERE email='%s'
```

```
sudo vi /etc/postfix/mysql_virtual_alias_maps.cf

user = mailuser
password = mailpassword
hosts = localhost
dbname = mailserver
query = SELECT destination FROM virtual_aliases WHERE source='%s'
```

Then edit the `submission` section of `/etc/postfix/master.cf` as follows:

```
submission inet n       -       n       -       -       smtpd
   -o syslog_name=postfix/submission
   -o smtpd_tls_security_level=encrypt
   -o smtpd_sasl_auth_enable=yes
#   -o smtpd_tls_auth_only=yes
#   -o smtpd_reject_unlisted_recipient=no
#   -o
smtpd_client_restrictions=$mua_client_restrictionsemail1@yourdomain.com
#   -o smtpd_helo_restrictions=$mua_helo_restrictions
#   -o smtpd_sender_restrictions=$mua_sender_restrictions
#   -o smtpd_recipient_restrictions=
   -o smtpd_relay_restrictions=permit_sasl_authenticated,reject
#   -o milter_macro_daemon_name=ORIGINATING
```

At this point you are ready to start (if not yet) `postfix` and enable it at boot time.

```
sudo systemctl start postfix
sudo systemctl enable postfix
```

You can check whether `postfix` is able to query the *mailserver* MariaDB database with the following commands. email1@yourdomain.com

```
postmap -q yourdomain.com mysql:/etc/postfix/mysql_virtual_domains_maps.cf
postmap -q email1@yourdomain.com
mysql:/etc/postfix/mysql_virtual_mailbox_maps.cf
postmap -q alias@yourdomain.com
mysql:/etc/postfix/mysql_virtual_alias_maps.cf
```

The result of the first two commands should be '1', while the result of the last should be 'email1@yourdomain.com'. If so, the setup of `postfix` is done.

**Dovecot setup**

Again, just to be on the safe side, back up the original configuration.

```
sudo cp /etc/dovecot/dovecot.conf /etc/dovecot/dovecot.conf.orig
sudo cp /etc/dovecot/conf.d/10-mail.conf /etc/dovecot/conf.d/10-
mail.conf.orig
sudo cp /etc/dovecot/conf.d/10-auth.conf /etc/dovecot/conf.d/10-
auth.conf.orig
sudo cp /etc/dovecot/dovecot-sql.conf.ext /etc/dovecot/dovecot-
sql.conf.ext.orig
sudo cp /etc/dovecot/conf.d/10-master.conf /etc/dovecot/conf.d/10-
master.conf.orig
sudo cp /etc/dovecot/conf.d/10-ssl.conf /etc/dovecot/conf.d/10-ssl.conf.orig
```

Then edit `/etc/dovecot/dovecot.conf`, uncomment the *protocol* setting.

```
sudo vi /etc/dovecot/dovecot.conf

protocols = imap pop3 lmtp
```

In `/etc/dovecot/conf.d/10-mail.conf` define the mail location, the mail privileged group and enable the *vmail* user to login.

```
sudo vi /etc/dovecot/conf.d/10-mail.conf

mail_location = maildir:/srv/maildirs/%d/%n
mail_privileged_group = mail
first_valid_uid = 303
last_valid_uid = 303
first_valid_gid = 303
last_valid_gid = 303
```

Create the *maildirs* directory and its substructure.

```
sudo mkdir -p /srv/maildirs/yourdomain.com/{email1,email2}
sudo chown -R vmail:vmail /srv/maildirs/
```

Then edit `/etc/dovecot/conf.d/10-auth.conf` to enable MariaDB authorization.

```
sudo vi /etc/dovecot/conf.d/10-auth.conf

disable_plaintext_auth = yes
auth_mechanisms = plain login
#!include auth-system.conf.ext
!include auth-sql.conf.ext
```

In `/etc/dovecot/conf.d/auth-sql.conf.ext` edit the *passdb* and the *userdb* sections.

```
sudo vi /etc/dovecot/conf.d/auth-sql.conf.ext

passdb {
  driver = sql

  # Path for SQL configuration file, see example-config/dovecot-sql.conf.ext
  args = /etc/dovecot/dovecot-sql.conf.ext
}

userdb {
  driver = static
  args = uid=vmail gid=vmail home=/srv/maildirs/%d/%n
}
```

The parameters to connect to *mailserver* database are stored in `/etc/dovecot/dovecot-sql.conf.ext`. Uncomment and/or edit as follows:

```
sudo vi /etc/dovecot/dovecot-sql.conf.ext

driver = mysql
connect = host=127.0.0.1 dbname=mailserver user=mailuser
password=mailpassword
default_pass_scheme = SHA512-CRYPT
password_query = SELECT email as user, password FROM virtual_users WHERE
email='%u';
```

Then change the ownership and the group of the dovecot folder to *vmail* user.

```
sudo chown -R vmail:dovecot /etc/dovecot
sudo chmod -R o-rwx /etc/dovecot
```

Edit as follows the *service imap-login*, *service lmtp*, *service auth*, and *service auth-worker* sections in `/etc/dovecot/conf.d/10-master.conf`.

```
sudo vi /etc/dovecot/conf.d/10-master.conf

service imap-login {
  inet_listener imap {
    port = 0
  }
  inet_listener imaps {
    #port = 993
    #ssl = yes
  }

  # Number of connections to handle before starting a new process. Typically
  # the only useful values are 0 (unlimited) or 1. 1 is more secure, but 0
  # is faster. <doc/wiki/LoginProcess.txt>
  #service_count = 1
```

```
  # Number of processes to always keep waiting for more connections.
  #process_min_avail = 0

  # If you set service_count=0, you probably need to grow this.
  #vsz_limit = $default_vsz_limit
}

service lmtp {
  unix_listener /var/spool/postfix/private/dovecot-lmtp {
    mode = 0600
    user = postfix
    group = postfix
  }

  # Create inet listener only if you can't use the above UNIX socket
  #inet_listener lmtp {
    # Avoid making LMTP visible for the entire internet
    #address =
    #port =
  #}
}

service auth {
  # [...]
  unix_listener /var/spool/postfix/private/auth {
    mode = 0666
    user = postfix
    group = postfix
  }

  unix_listener auth-userdb {
    mode = 0600
    user = vmail
  }

  # Postfix smtp-auth
  #unix_listener /var/spool/postfix/private/auth {
  #  mode = 0666
  #}

  # Auth process is run as this user.
  user = dovecot
}

service auth-worker {
  # Auth worker process is run as root by default, so that it can access
  # /etc/shadow. If this isn't necessary, the user should be changed to
  # $default_internal_user.
  user = vmail
}
```

Finally edit `/etc/dovecot/conf.d/10-ssl.conf` to define name and location of your certificate and private key.

```
sudo vi /etc/dovecot/conf.d/10-ssl.conf

ssl_cert = </path/to/your/cert/yourcert.pem
ssl_key = </path/to/your/privkey/yourprivkey.pem
```

Start `dovecot`.

```
sudo systemctl start dovecot
```

Well done! Now you can configure your email client as follows:

- username: email1@yourdomain.com
- password: email1's password
- IMAP: yourdomain.com
- SMTP: yourdomain.com

## 13.2

### Installation

### Install using Ignition

OpenSUSE can be installed using the Ignition installer. Flash Ignition to an SD card, then use Ignition to download and install OpenSUSE automatically. As an alternative, get the images directly from http://files.jm0.eu/suse/, unpack and write to your sdcard.

### Repositories

### BSP

The BSP repository will provide a set of packages required by the Cubox-i and Hummingboard to operate properly which cannot be included in openSUSE for license reasons or required patches.

```
sudo zypper ar -f http://repo.maltegrosse.de/suse/13.2/bsp_cuboxi/ bsp-
cuboxi-13.2
```

### Packman

*##TODO##*

### Wifi

The Wireless LAN chip requires non-free firmware files which are part of openSUSE 13.2, as well as firmware configuration files that you can install from the BSP repository:

```
sudo zypper install cuboxi-firmware-wifi-config
```

#### Bluetooth

The bluetooth chip requires a firmware which is not part of openSUSE 13.2. You can install it from the BSP repository:

```
sudo zypper install cuboxi-firmware-bluetooth
```

In addition, the patchram utility has to be run.

*##TODO: document patchram usage##*

### 13.1

#### Working Features

- resizes rootfs automatically to use the complete SD-Card
- creates 512 MB swap partition
- Yast2 Firstboot Setup
- Accelerated OpenGL-ES

#### Installation

You can get the openSUSE images at http://files.jm0.eu/suse/. Unpack and write to sdcard using dd or win32diskimager.

#### Downloading manually

#### JeOS

This is a minimal commandline image to be used for servers or custom applications: openSUSE-13.1-ARM-JeOS-cuboxi.armv7l-1.12.1-Build68.2.raw.xz Use an SD-Card >= 2GB;

Be sure to attach a screen and keyboard at least during the first boot, to run the graphical yast wizard. If you need to go headless, you can follow these instructions for raspberry pi: rpi headless setup

#### E17

This is a desktop image featuring the Enlightenment 17 Desktop Environment: openSUSE-13.1-ARM-E17-cuboxi.armv7l-1.12.1-Build68.2.raw.xz Use an SD-Card >= 4GB

### XFCE

This is a desktop image featuring the XFCE Desktop Environment:openSUSE-13.1-ARM-XFCE-cuboxi.armv7l-1.12.1-Build68.2.raw.xz Use an SD-Card >= 4GB

### Installing manually

Simply write the downloaded image to your sdcard after uncompressing. This can be done in one step on Unix systems:

```
xzcat <yourimage.xz> | sudo dd of=/dev/sdX bs=1M; sync
```

*yourimage.xz* should be the file you downloaded and sdX the name of your sdcard device.

### Network Time

The latest images all have ntp configured to update the time at every startup and at runtime. It will start to magically work right after the firstboot yast wizard is done. If you want to configure ntp yourself, simpyl use YaST:

```
 # Install yast ntp module
sudo zypper install --no-recommends yast2-ntp-client
```

```
 # Configure NTP within yast CLI interface
sudo /sbin/yast ntp-client
# Or you can run the graphical YaST from your start menu, or with this
command:
xdg-su -c "/sbin/yast2 ntp-client"
```

You will want to ignore the message that ntp.conf has been changed externally and continue.

### OpenGL-ES

The Cubox-i contains an OpenGL-ES capable GPU by Vivante. Freescale provides binary-only implementations of Graphics libraries including but not limited to OpenGL 2, OpenGL-ES, EGL and OpenCL. The OpenGL implementation is rather rudimentary right now and not officially supported by Vivante. So far it has proven to be sufficient for running glxgears, and even bzflag has been running(only at 7fps, but thats not too bad). The OpenGL-ES implementations should be good but I have not done any thorough testing on them.

The Binary-only libraries for the Vivante GC2000 and GC880as contained in the i.MX6 SoCs can be installed by executing the following commands:

```
 # Add repository with the vivante packages
sudo zypper ar -f -G http://vps.jm0.eu/vivante vivante
```

```
 # install the required vivante libraries, the xorg driver, and a suitable
```

```
xorg config file
sudo zypper install --no-recommends xf86-video-vivante-3.10.17-1.0.1
vivante-xorg-config \
     gpu-viv-bin-mx6q-3.10.17-1.0.1-{dri-x11,libEGL-x11,libGAL-x11,libGL-
x11,libGLSLC,libGLESv1_CM,libGLESv2-x11}
# OPTIONALLY install these vivante libraries too
sudo zypper install --no-recommends \
     gpu-viv-bin-mx6q-3.10.17-1.0.1-
{libCLC,libGLES_CM,libOpenCL,libOpenVG_355,libVDK,libVIVANTE-x11}
```

Now, that was not enough! On openSUSE its a convention that users who need access to graphics hardware have to be in the video group. So, to make use of the graphics libraries, add yourself to the video group:

```
sudo /usr/sbin/usermod -a -G video <youruser>
```

I now suggest to make a reboot to make sure the system including the GPU is in a sane state.

Now you should have working OpenGL-ES, have fun with it!

#### Web Browser

Neither Mozilla Firefox nor Google Chrome are part of OpenSUSE 13.1 for arm. When openSUSE 13.1 was released, it was impossible to build them on the arm build bots because they took *very* long to build and required far too much RAM. At least Mozilla Firefox is expected to be included in openSUSE 13.2, chrome will likely not build on arm for quite a while.

#### Mozilla Firefox

Meanwhile, firefox can be built on x86 using qemu and a chroot, and I have created a repository with the latest Firefox package, and included it in the latest images.

A note about WebGL: You can enable webgl in about:config, but I recommend not to do it! The spinning cube demo at get.webgl.org works fine, but anything more advanced will cause your browser to crash. I have no idea why and I would have expected it to just work.

#### Hardware-accelerated Video De- and Encoding

The i.MX6 SoC has a VPU unit that can process h.264 and VC1. Freescale provides libraries to naje use of the VPU, and these can now be installed from the vivante repository above like this:

```
sudo zypper install --no-recommends firmware-imx-vpu libvpu libfslvpuwrap
gstreamer-plugins-1.0-imx
```

I suggest to also install the OpenGL-ES packages from above.

A simple way to test playback is this:

```
gst-launch-1.0 playbin uri=file://$PWD/yourvideo.mp4
```

## TODO List

- Audio
- udev rules for gpu, vpu and other special devices that are part of the i.MX6 SoCs
- firefox webgl stability
- chromium webgl support
- tty on /dev/mxc0
- ...

## External Links

- Solid-Run Forum
- opensuse.org
- openSUSE Factory ARM project and related images (see also the changelog)*.
- openSUSE Leap 15.0 ARM port project and related images (see also the changelog)
- openSUSE Leap 42.3 ARM port project and related images (see also the changelog)
- openSUSE Leap 42.2 ARM port project and related images (see also the changelog)
- Mailing list of openSUSE ARM project
- Apache configuration for Webmin

* Please, note that if the images listed at this location are outdated, you may download the latest directly from the OBS.