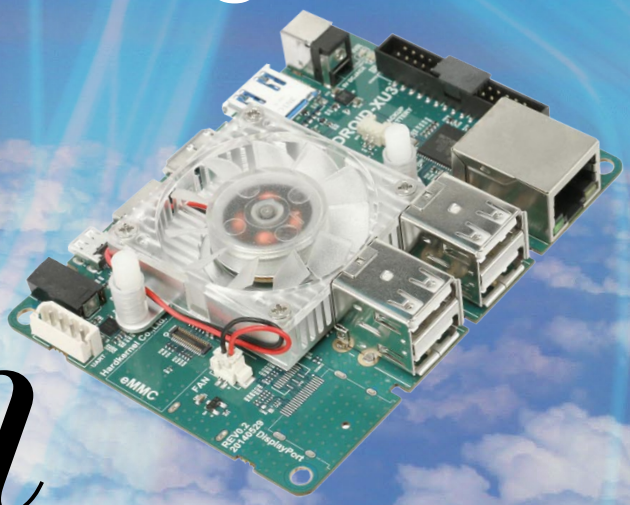
14.04 Repository Upgrade • Community Images • Graphics with Java

ODROID Magazine

Year Two
Issue #13
Jan 2015

Cloud Edition



Docker: Develop, Ship and Run Any Application, Anywhere with Containers

Seafile: Open Source Personal Cloud Software

OwnCloud: File Synchronization and Sharing Using Your Private ODROID Server

COMPARISON OF THE GAMING POWER OF THE ODROID-XU3 VS ODROID-U3

- Microsoft-Free Programming: Setting up an ASP.NET and Mono Server Stack
- GNU Radio: Wireless communications research and real-world radio systems

What we stand for.

We strive to symbolize the edge of technology,
future, youth, humanity, and engineering.

Our philosophy is based on Developers.
And our efforts to keep close relationships with
developers around the world.

For that, you can always count on having the quality
and sophistication that is the hallmark of our products.

Simple, modern and distinctive.
So you can have the best to accomplish
everything you can dream of.

HARDKERNEL

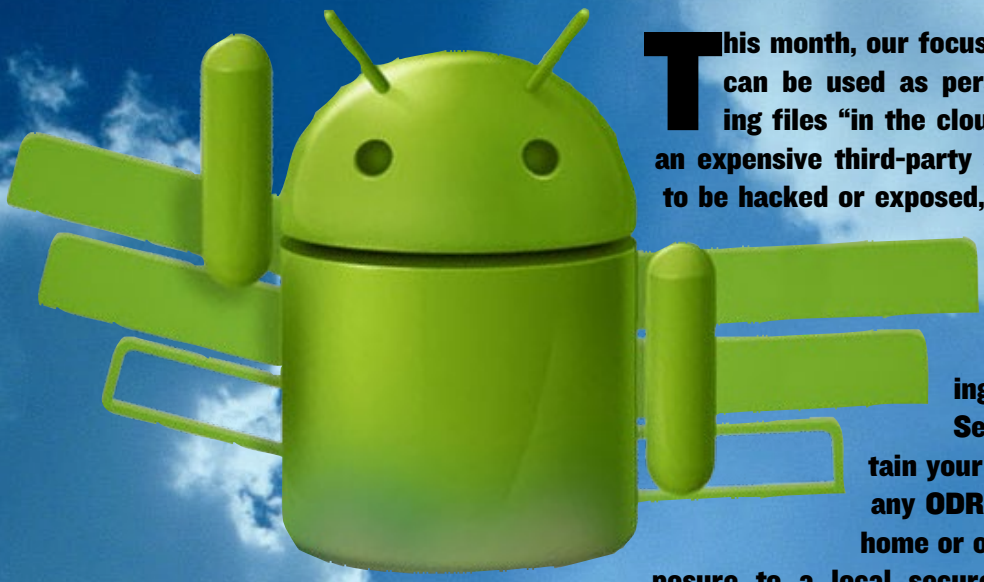
Pollin ELECTRONIC

We are now shipping the ODROID U3 devices to EU countries! Come and visit our online store to shop!

Address: Max-Pollin-Straße 1
85104 Pförring Germany

Telephone & Fax
phone : +49 (0) 8403 / 920-920
email : service@pollin.de

Our ODROID products can be found at
http://bit.ly/1tXPXwe



This month, our focus is on demonstrating how ODROIDs can be used as personal private cloud servers. Storing files "in the cloud" normally involves subscribing to an expensive third-party service, with the potential for data to be hacked or exposed, as has been demonstrated with recent news articles involving celebrity photos and corporate emails that were released to the public without permission. By installing and configuring software such as Seafile and OwnCloud, you can maintain your preferred level of security by using any ODROID device to host the files at your home or office. By doing so, you can limit exposure to a local secure intranet, so that the chance of an unknown intruder gaining access to them is greatly reduced.

A hot topic in world of cloud computing is Docker, which allows applications to be built on a platform of the developer's choice, then installed and run on nearly any architecture, including ODROIDs! As demonstrated in the latest Google I/O conference, Docker offers a highly stable environment in which to distribute and compartmentalize applications for scalability purposes. It enables apps to be quickly assembled from various components that are completely portable. Fred Meyer, a Docker expert who recently joined the ODROID Magazine team, presents the first part of an interesting series on getting Docker running on an ARM device.

Now that the ODROID-C1 has been available for a while, the ODROID forum has grown to nearly 8,000 members, with many suggestions, tips, and tutorials being posted daily. Tobias recently uploaded many of his game packages and useful software to the Hardkernel community server, and also put together a guide for connecting to his repository in order to get kernel updates via a simple apt-get command, which is a more convenient way to update custom ODROID software from Debian and Ubuntu.

As always, we bring you reviews of some fun games to play, including several Android programs, as well as a comparison of the gaming power of the ODROID-XU3 vs the ODROID-U3. Nanik continues his excellent series on Android Development with an introduction to the Zygote app, and we feature a tutorial on using your ODROID as a modern digital radio using the GNU radio package.

ODROID Magazine, published monthly at http://magazine.odroid.com, is your source for all things ODROIDian.
Hard Kernel, Ltd. • 704 Anyang K-Center, Gwanyang, Dongan, Anyang, Gyeonggi, South Korea, 431-815
Makers of the ODROID family of quad-core development boards and the world's first ARM big.LITTLE architecture based single board computer.
Join the ODROID community with members from over 135 countries, at http://forum.odroid.com, and explore the new technologies offered by Hardkernel at http://www.hardkernel.com.

HARDKERNEL

ODROID Magazine

Rob Roy, Chief Editor

I'm a computer programmer living and working in San Francisco, CA, designing and building web applications for local clients on my network cluster of ODROIDs. My primary languages are jQuery, Angular JS and HTML5/CSS3. I also develop pre-built operating systems, custom kernels and optimized applications for the ODROID platform based on Hardkernel's official releases, for which I have won several Monthly Forum Awards. I use my ODROIDs for a variety of purposes, including media center, web server, application development, workstation, and gaming console. You can check out my 100GB collection of ODROID software, prebuilt kernels and OS images at `http://bit.ly/1fsaXQs`.

Bo Lechnowsky, Editor

I am President of Respectech, Inc., a technology consultancy in Ukiah, CA, USA that I founded in 2001. From my background in electronics and computer programming, I manage a team of technologists, plus develop custom solutions for companies ranging from small businesses to worldwide corporations. ODROIDs are one of the weapons in my arsenal for tackling these projects. My favorite development languages are Rebol and Red, both of which run fabulously on ARM-based systems like the ODROID-U3. Regarding hobbies, if you need some, I'd be happy to give you some of mine as I have too many. That would help me to have more time to spend with my wonderful wife of 23 years and my four beautiful children.

Bruno Doiche, Senior Art Editor

Bruno has been securing his computing necromantic skills after bringing a fiber optics switch back to life, getting his Macintosh back from death, resurrecting a PS3, rescuing his fiancee's T400 with an old-school dd data transplant, and handling the cold innards of his steady job at the data center.

Nicole Scott, Art Editor

I'm a Digital Strategist and Transmedia Producer specializing in online optimization and inbound marketing strategies, social media directing, and media production for print, web, video, and film. Managing multiple accounts with agencies and filmmakers, from Analytics and Adwords to video editing and DVD authoring. I own an ODROID-U3 which I use to run a sandbox web server, live in the California Bay Area, and enjoy hiking, camping and playing music. Visit my web page at http://www.nicolecscott.com.

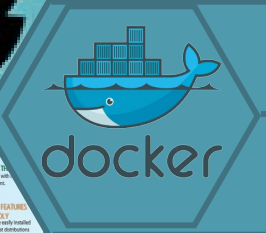
James LeFevour, Art Editor

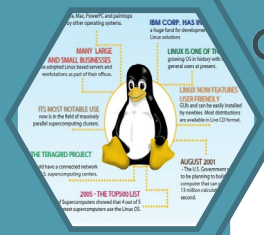
I am a Digital Media Specialist who is also enjoying freelance work in social network marketing and website administration. The more I learn about ODROID capabilities the more excited I am to try new things I'm learning about. Being a transplant to San Diego from the Midwest, I am still quite enamored with many aspects that I think most West Coast people take for granted. I live with my lovely wife and our adorable pet rabbit; the latter keeps my books and computer equipment in constant peril, the former consoles me when said peril manifests.

Manuel Adamuz, Spanish Editor

I am 31 years old and live in Seville, Spain, and was born in Granada. I am married to a wonderful woman and have a child. A few years ago I worked as a computer technician and programmer, but my current job is related to quality management and information technology: ISO 9001, ISO 27001, and ISO 20000. I am passionate about computer science, especially microcomputers such as the ODROID and Raspberry Pi. I love experimenting with these computers. My wife says I'm crazy because I just think of ODROIDs! My other great hobby is mountain biking, and I occasionally participate in semi-professional competitions.

INDEX





GNU RADIO

BRING YOUR PERSONAL RADIO BROADCASTS INTO THE 21ST CENTURY

by @denash

GNU Radio is a free and open-source software development toolkit that provides signal processing blocks to implement software radios. It can be used with readily-available low-cost external RF hardware to create software-defined radios, or without hardware in a simulation-like environment. It is widely used in hobbyist, academic and commercial environments to support both wireless communications research and real-world radio systems.

To use GNU Radio, boot up a Debian or Ubuntu distribution and follow these steps for installation:

1. Type the following into a Terminal window to install GNU Radio from the Debian repository:

```
$ wget -c \
http://ftp.debian.org/debian/
pool/main/q/qwtplot3d/libqwt-
plot3d-qt4-0_0.2.7+svn191-7_arm-
hf.deb
$ wget -c \
http://ftp.debian.org/debian/
pool/main/q/qwtplot3d/libqwt-
plot3d-qt4-dev_0.2.7+svn191-7_
armhf.deb
$ sudo dpkg -i libqwtplot3d-
qt4-*.deb
$ gnuradio
```

2. Use this pre-built script to build GNU Radio from source: http://bit.ly/IAWW3vr. Save it to a temporary folder under the name "build-gnuradio", then type the following after navigating to the temporary folder:

We didn't mod this classic radio, but don't think we wouldn't if we got our hands on it!

```
$ sudo chmod +x ./build-gnuradio
&& ./build-gnuradio && gnuradio
```

3. Follow these instructions to create your own build-gnuradio script:

Download the standard GNU Radio build script:

```
$ wget http://www.sbrac.org/files/
build-gnuradio
```

Create two local variables:

```
$ TEST=-DCMAKE_CXX_
FLAGS:STRING="-march=armv7-a
-mcpu=cortex-a9 -mfpu=neon \
-mfloat-abi=hard"

$ TEST2=-DCMAKE_C_FLAGS:STRING="-
march=armv7-a -mcpu=cortex-a9
-mfpu=neon \
-mfloat-abi=hard"
```

Search for every occurrence of cmake in the build-gnuradio script, and add "$TEST" "$TEST2" to the arguments. At around line 348, find this line:

```
for dir in /lib /usr/lib /usr/
lib64 /lib64 /usr/lib/x86_64-
linux-gnu /usr/lib/i386-linux-gnu
```

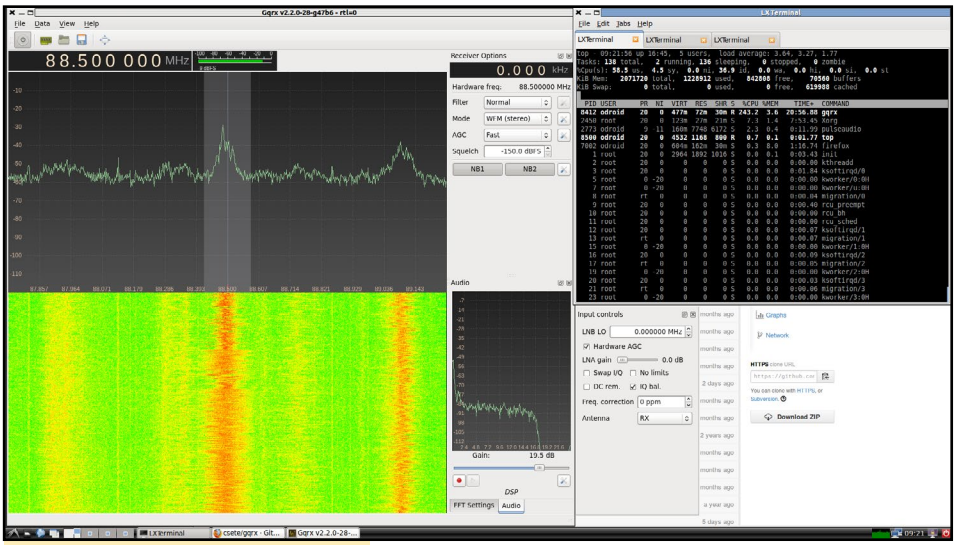
and append the following snippet:

```
/usr/lib/arm-linux-gnueabihf /
usr/lib/arm-linux-gnueabi
```

Finally, run the build script:

```
$ ./build-gnuradio
```

Note that gnuradio should not be compiled using the -j4 flag since GNU radio seems to break when built in parallel.

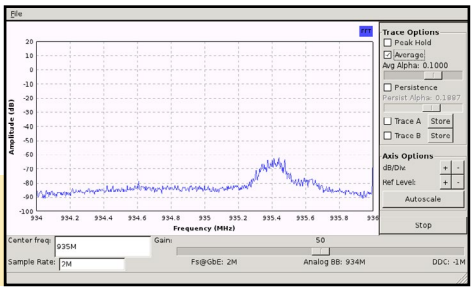


Running GQRX

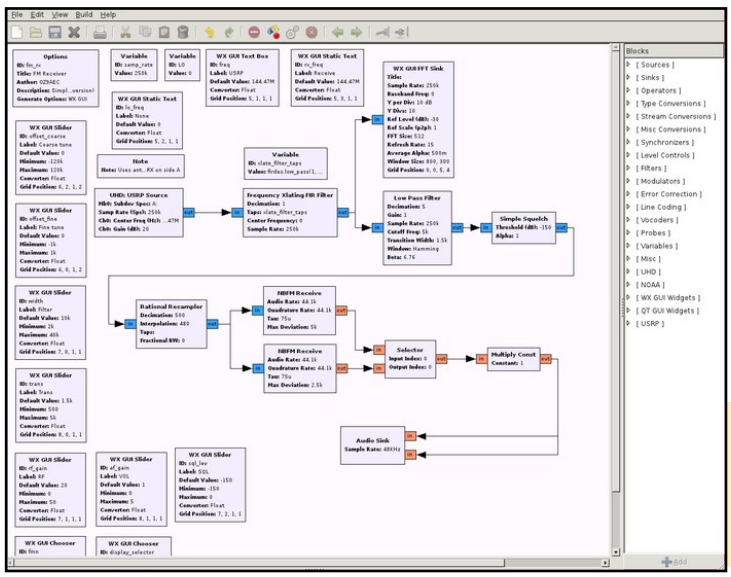
Adding microphone input

1. Run the "PulseAudio VolumeControl" application, found in the Applications menu
2. In the Configuration tab, select the "Analog Stereo + Analog Mono Input"
3. Install "GNOME ALSA Mixer" via the Ubuntu Software center and run it
4. Ignore the error pop-up. It seems to be a permission issue of saving configuration file
5. Check the following 3 items in the Mixer GUI: "MIC Bias VCM Bandgap", "MICI Mix", and "Left ADC Mixer MICI"
6. You can adjust the input gain or boost option with the "ADCL", "ADCL Boo", "MICI", and "MICI Boo" control bar
7. You can view the real time microphone input level in the Input Device tab of "PulseAudio VolumeControl"

More information about GNU Radio may be found at the home page at `http://gnuradio.org`.



Using the uhd_fft.py tool to observe a GSM downlink channel



Screenshot of the GRC with a narrow band FM receiver

METAL SLUG DEFENSE

A WHOLE NEW TAKE ON A BELOVED SERIES

by Bruno Doiche

I love the game Metal Slug, which I've played in the arcade, on my NEO GEO CD, on a cellphone, and emulated on every single computer I've ever had - even the POWER 4 and 5 servers I used to own.

However, after countless plays, I thought I'd seen everything that was supposed to be seen in the series, and suddenly... BAM! SNK released a defense game based on Metal Slug.

Take a look for yourself - it's a very fun game!

`http://bit.ly/1iH5Z2k`

MY VERY OWNCLOUD

KEEP YOUR FILES SECURE WITH A PERSONAL CLOUD SERVER

by Venkat Bommakanti

OwnCloud is an enterprise-quality file synchronization and sharing application that is hosted in your own data center, on your servers, using your own storage. OwnCloud provides universal file access through a single front-end to all of your systems, regardless of the particular architectures. Users can access company files on any device, anytime, anywhere, while IT can manage, control and audit file sharing activity to ensure that security and compliance measures are met.

In this article, I present the specifics of installing, configuring and using the most recent versions of the best-of-breed software that comprises a robust secure ownCloud solution, which consists of the following components:

Odroid XU3 or XU3 Lite with ITB USB3 storage (plus ITB backup) and Gigabit Ethernet
LEMP software stack (Linux 3.10.60, Enginx 1.4.6, MySQL 5.5.40, Php 5.5.9)
phpMyAdmin 4.0.10
ownCloud 7.0.4

Online documentation for ownCloud is skimpy, outdated and confusing in some cases, if not outright misleading, inaccurate and untested. A variety of information resources have been researched, and a working configuration has been painstakingly assembled in an effort to make your journey through the setup process as easy as possible.

Requirements

1. An ODROID XU3-Lite or XU3 board, with an appropriate power adapter. While this article targets an XU3-Lite, it can apply to a U3 or a CI as well.
2. A 16GB+ eMMC 5.0 modue or Class 10 MicroSD card with the latest XU3-Lite specific Lubuntu desktop image.
3. A ITB USB3 external HD such as a WD Ultra or Toshiba Canvio, used for primary ownCloud data storage. A second ITB USB3 external hard drive may also be added as a method of backing up the primary data.

Install Lubuntu

Install the latest XU3 image onto the eMMC card, and boot up the system with the HDMI display attached. Run the ODROID Utility and use it to expand the operating system partition. Reboot and run the ODROID Utility again, updating the kernel, video drivers and all other aspects of the system. Reboot one more time before continuing to the next step.

Prepare the system

Backup all the operating system files and software on the external USB3 HD's using the dd utility if desired. With the XU3 system powered down, do the following:

• Attach the primary external HD, which will be used to store the main ownCloud data, to the USB3 Host Type A port,
• Attach the secondary external HD, which will be used as the ownCloud data backup, to one of the many USB2 Type A ports. Since backups can be scheduled for off-peak hours, a USB2 connection will suffice for the backup drive.
• Attach the USB3 OTG cable to the USB3 port and attach the other end of the cable to the Gigabit Ethernet dongle. Attach the dongle to your home router using a regular Cat5E or Cat6 cable.

Since the two HDs are normally NTFS formatted out of the box, they should be detected and mounted automatically.

Install MySQL

Rather than using lightweight data management options through the default SQLite system, I have chosen the highly scalable and popular MySQL RDBMS for managing the administrative meta data of the ownCloud instance.

First, install MySQL software using the following command:

```
$ sudo apt-get install mysql-server mysql-client
```

Reboot the system and check the installation:

```
$ mysql -V
mysql Ver 14.14 Distrib 5.5.40, for debian-linux-gnu (armv7l) using read-
line 6.3
```

Setup the root password on first use:

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

Enter a password at the prompt and note it somewhere safely. For this example, I used “odroid” as the password for the root user. The installation can also be checked using the following SQL commands from the MySQL CLI:

```
mysql> SHOW VARIABLES LIKE “%version%”;
mysql> STATUS;
mysql> show databases;
mysql> select user,host from mysql.user;
mysql> exit
```

Alternatively, you can check the installation using the MySQL admin application like so:

```
$ mysqladmin -u root -p version
```

Install the system database and secure the installation using the following commands, after which the installed MySQL instance will be ready for use by ownCloud:

```
$ sudo mysql_install_db
```

```
$ sudo mysql_secure_installation
```

Install nginx

To create a robust and efficient setup, we have chosen the nimble nginx webserver over the default apache web server. You can refer to the August 2014 ODROID Magazine issue for specific instructions regarding nginx installation and configuration.

The general steps are as follows:

Install nginx using the command:

```
$ sudo apt-get install nginx-full
```

Next, check the username that owns the nginx installation, which will be used later:

```
$ sudo grep user /etc/nginx/nginx.conf user www-data;
```

Setup the SSL credentials using the commands (each command in a single line):

```
$ sudo cd /etc/nginx/ && sudo mkdir ssl
$ sudo openssl req -x509 -nodes -days 365 \
-newkey rsa:2048 -keyout \
/etc/nginx/ssl/nginx.key -out \
/etc/nginx/ssl/nginx.crt
```

Update the nginx configuration to address the needs of the SSL, PHP5 and ownCloud installations:

```
$ sudo cd /etc/nginx/sites-available
$ sudo cp default default-orig
$ sudo medit default
```

Replace the existing server { … } block with the following configuration. Each configuration snippet shown below should be on its own line:

```
...
# our php-handler - add this
upstream php-handler {
    server unix:/var/run/php5-fpm.sock;
}

# update section like so:
server {
    listen 80 default_server;
    listen [::]:80 default_server ipv6only=on;

    # ssl support
    listen 443 ssl;

    root /usr/share/nginx/html;

    # try php file execution first
    index index.php index.html;

    # Make site accessible from http://your-XU3-
host-ip-addr/
    server_name <your-XU3-host-ip-addr>;

            # ssl credentials
    ssl_certificate     /etc/nginx/ssl/nginx.crt;
    ssl_certificate_key /etc/nginx/ssl/nginx.key;

    # set max upload size
    client_max_body_size    10G;
    fastcgi_buffers         64 4K;
    client_body_buffer_size 2M;

    # setup calendar, contact, webdav options
    rewrite ^/caldav(.*)$ /remote.php/caldav$1 re-
direct;
    rewrite ^/carddav(.*)$ /remote.php/carddav$1
redirect;
    rewrite ^/webdav(.*)$ /remote.php/webdav$1 re-
direct;

    location = /robots.txt {
            allow all;
            log_not_found off;
            access_log off;
    }

    # diabling of .ht* checks doesn't work (from
here) for nginx.
    # so using /oc-data as the ownCloud data direc-
tory, instead of
# the typical data directory: /usr/share/nginx/html/
ownCloud/data.
# retained for future support when issue gets fixed in
ownCloud
    location ~ ^/(?:\.ht|oc-data|config|db_struc-
ture\.xml|README){
            deny all;
    }

    location / {
            # First attempt to serve request as
file, then
            # as directory, then fall back to dis-
playing a 404.
            try_files $uri $uri/ index.php;

            # The following 2 rules are only needed
with webfinger
            rewrite ^/.well-known/host-meta /pub-
lic.php?service=host-meta last;
            rewrite ^/.well-known/host-meta.json /
public.php?service=host-meta-json last;
            rewrite ^/.well-known/carddav /remote.
php/carddav/ redirect;
            rewrite ^/.well-known/caldav /remote.
php/caldav/ redirect;
            rewrite ^(/core/doc/[^\/]+/)$ $1/index.
html;
    }

    # redirect server error pages to the static
pages
    error_page 404 /404.html;
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
            root /usr/share/nginx/html;
    }

    # pass the PHP scripts to FastCGI server lis-
tening on fpm-socket
    location ~ \.php(?:$|/) {
            fastcgi_split_path_info ^(.+\.php)
(/.+)$;
            include fastcgi_params;
            fastcgi_param SCRIPT_FILENAME $docu-
ment_root$fastcgi_script_name;
                    # $fastcgi_path_info parse fails
in latest php5-fpm.  disable it.
            # fastcgi_param PATH_INFO $fastcgi_
path_info;
```

```
            fastcgi_pass php-handler;
            fastcgi_read_timeout 600;
      }

      # since "phpmyadmin" is a db-admin app, obfus-
cate it by using a
# random (or is it not) appname "OYA16z2-xFg" ;!).
make it use
            # a BASIC authentication dialog prior to
displaying its own login page.
            # the BASIC authentication credentials
are placed in the phpmyadmin_pass
            # file
            location /OYA16z2-xFg {
            auth_basic "PHPMyAdmin Login";
            auth_basic_user_file /etc/nginx/phpmyad-
min_pass;
      }
  }
  ...
```

Save the nginx config file after the above editing has been completed, then create placeholder HTML error pages if they don't already exist:

```
/usr/share/nginx/html/404.html
/usr/share/nginx/html/50x.html
```

Install php5

Type the following commands to install the prerequisites for PHP5:

```
$ sudo apt-get install autoconf automake autotools-
dev libtool curl
$ sudo apt-get install libcurl4-openssl-dev lbzip2
$ sudo apt-get install php5 php5-dev php5-cgi php5-
fpm php5-curl php5-gd
$ sudo apt-get install php5-mysql php5-gmp php5-
imagick php5-imap php5-intl
$ sudo apt-get install php5-ldap php5-mcrypt libm-
crypt-dev php-xml-parser
$ sudo apt-get install php5-xsl php-apc phpmyadmin
```

phpMyAdmin, a PHP-based tool, is a useful application for managing MySQL databases. During its installation, skip the web server config since nginx is not presented as an option. For the MySQL dbconfig-common config step, select "YES" and use your preferred secure password, which for simplicity in this example is "odroid". Make sure to use a more secure password for your own setup.

Note that in the nginx installation section earlier, we had already included the needed php5-fpm config. However, we will need to make a minor socket configuration change:

```
$ sudo cd /etc/php5/fpm/pool.d/
$ sudo cp www.conf www.conf-orig
$ sudo medit www.conf
```

Add the following socket configuration to the existing file, which is intended to match the existing nginx socket configuration that was established in the previous step:

```
listen = /var/run/php5-fpm.sock
```

The nginx document root folder is /usr/share/nginx/html, where you should create a sample php test file, which will be used later to test the PHP5 installation:

```
$ sudo cd /usr/share/nginx/html
$ su
# echo '<?php echo exec('whoami'); ?>' > info.php
# echo '<?php phpinfo(); ?>' >> info.php
```

Enhance file execution security by setting the following flags in the PHP5 configuration file:

```
$ sudo medit /etc/php5/fpm/php.ini
```

Set these options:

```
cgi.fix_pathinfo=0
display_errors = On
display_startup_errors = On
output_buffering = 0
```

Change the following options in the same file to suit your needs:

```
upload_max_filesize = 50M
max_file_uploads = 5
post_max_size = 60M
default_socket_timeout = 600
```

Save the changes. Note that some of the configuration changes above are related to the upcoming ownCloud installation. Next, apply the installed component configuration changes:

```
$ sudo service php5-fpm stop && sudo /etc/init.d/
mysql stop && sudo service nginx stop
$ sudo service nginx start && sudo /etc/init.d/mysql
start && sudo service php5-fpm start
```

Verify the installation by navigating a web browser to http://<XU3 IP address>/info.php. Because a simple PHP script was created earlier, the output should match the screenshot shown in Figure 1.

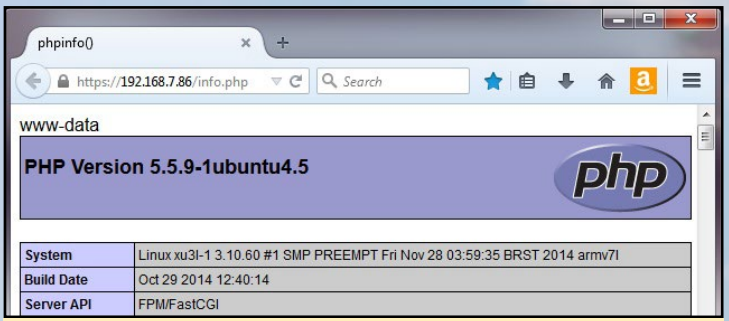


Figure 1 - PHP test page showing the info.php output

When you check the output, you may notice that a useful PHP5 module called mcrypt is disabled, which can be enabled using the following command:

```
$ sudo php5enmod mcrypt
```

Restart php5-fpm, mysql and nginx with the previous commands above and refresh the browser page to review the PHP information script. It should now show that the mcrypt module is enabled.

Install ownCloud

Create a placeholder directory in your home dir and navigate to it:

```
$ mkdir ownCloud && cd ownCloud
```

Download the latest ownCloud tarball for Linux and its corresponding md5 hash file:

```
$ wget https://download.ownCloud.org/community/own-
Cloud-7.0.4.tar.bz2
$ wget http://bit.ly/1GIchxr
```

Check integrity of the ownCloud file:

```
$ cat ownCloud-7.0.4.tar.bz2.md5
6d4a3f9275d1f2b2607e7e6484051d4c  -
$ md5sum ownCloud-7.0.4.tar.bz2
6d4a3f9275d1f2b2607e7e6484051d4c  ownCloud-7.0.4.tar.
bz2
```

If the md5sum numbers match, we are OK to install it. The authenticity of the file can also be checked using the instructions at http://bit.ly/13Nlfeu.

Expand the tarball to a new subdirectory:

```
$ mkdir ~/zBU/ownCloud/oc
$ cd ~/zBU/ownCloud/oc && cp ../ownCloud-7.0.4.tar.
bz2 .
$ tar -xjf ownCloud-7.0.4.tar.bz2
```

Next, move the expanded tarball contents to the nginx document root at /usr/share/nginx/html:

```
$ sudo mv ownCloud /usr/share/nginx/html/
```

The ownCloud installation comes with a sample php config file called config.sample.php. Make a copy of it and edit the copy to establish the necessary configuration:

```
$ cd /usr/share/nginx/html/ownCloud/config
$ sudo cp config.sample.php config.php
$ sudo medit ./config.php
```

Update the following sections, substituting the IP address for <XU3-host-ip-address> without using the "<" and ">" symbols:

```
'apps_paths' => array(
  array(
              'path'=> '/usr/share/nginx/html/own-
Cloud/apps',
      'url' => '/apps',
      'writable' => true,
      ),
  ),

'trusted_domains' =>
  array (
      'localhost',
      '<XU3-host-ip-address>',
  ),
```

Create the ownCloud data directory along with a data backup directory, then change the ownership and other attributes of the various ownCloud directories:

```
$ cd usr/share/nginx/html/ownCloud
$ sudo mkdir oc-data && sudo mkdir oc-data-bu
$ sudo chown -R root:root /usr/share/nginx/html/own-
Cloud/
$ sudo chown -R www-data:www-data /usr/share/nginx/
html/ownCloud/config/
$ sudo chown -R www-data:www-data /usr/share/nginx/
html/ownCloud/oc-data/
$ sudo chmod 0775 /usr/share/nginx/html/ownCloud/oc-
data/
$ sudo chown -R odroid:odroid /usr/share/nginx/html/
ownCloud/oc-data-bu/
$ sudo chmod 0775 /usr/share/nginx/html/ownCloud/oc-
data-bu/
$ sudo chown root:root /usr/share/nginx/html/own-
Cloud/.htaccess
$ sudo chown -R www-data:www-data /usr/share/nginx/
html/ownCloud/apps/
```

The config/, oc-data/ and apps/ subdirectories need to be under ownership of www-data, which was established earlier. Also, note that the backup directory oc-data-bu can be used by a cron-job (under the user odroid) to periodically backup the ownCloud data directory contents. This allows for the regular odroid user to restore data if necessary.

By default, the ownCloud installation presumes the use of Apache, and relies on the .htaccess file to ensure proper access restrictions, which interferes with the operation of nginx. To address this, we need to move the oc-data and oc-data-bu directories out of the nginx document directory structure to the system root "/":

```
$ sudo mv /usr/share/nginx/html/ownCloud/oc-data /
$ sudo mv /usr/share/nginx/html/ownCloud/oc-data-bu /
```

The ownership and permissions will remain unchanged, allowing ownCloud to properly access the directories. We can

now use these directories to create mount-points for the two USB3 external hard drives. Update the fstab file so that the mounts persist across every reboot:

```
$ cd /etc
$ sudo medit ./fstab
```

Add the following entries, each in its own single line:

```
# WD My Passport Ultra 1TB - external HD #1
/dev/sda1  /oc-data  ext4  defaults,errors=remount-
ro,noatime,nodiratime 0 2

# Toshiba Canvio 1TB - external HD #2
/dev/sdb1  /oc-data-bu ext4 defaults,errors=remount-
ro,noatime,nodiratime         0 2
```

After you have backed any existing data on the hard drives, you can use the gparted utility to reformat them to ext4 format, then reboot. The file-system entries should be verified:

```
$ df -h
   Filesystem        Size  Used Avail Use% Mounted on
   ...
   /dev/sda1         917G   72M  871G   1% /oc-data
   /dev/sdb1         917G   72M  871G   1% /oc-data-
bu
```

This ensures that ownCloud will use the reasonably fast and large 1TB USB3 hard drives. Although 1TB may not be enough for some purposes, it is definitely better than a few paltry gigabytes!

Configure phpmyadmin

Even though the phpMyAdmin installation was addressed in an earlier step, its configuration is incomplete. If the MySQL installation is reported as working using phpMyAdmin, we can safely assume that a major part of the overall installation is completed properly. Prepare the phpMyAdmin installation to be usable under nginx, then verify it:

```
$ sudo ln -s /usr/share/phpmyadmin /usr/share/nginx/
html
$ ls -ltr /usr/share/nginx/html
...
lrwxrwxrwx  1 root root   21 Dec 12 13:46 phpmyadmin
-> /usr/share/phpmyadmin
```

Next, create the basic authentication credentials:

```
$ openssl passwd
Password: birdsong
Verifying - Password:
Warning: truncating password to 8 characters
PUzMLT4M8HMDY
```

Then, create a password file:

```
$ cd /etc/nginx
$ sudo touch phpmyadmin_pass
$ sudo medit ./phpmyadmin_pass
```

Add the following on the first line and save the file:

```
zWarlock:PUzMLT4M8HMDY
```

Although basic authentication is used as an example, you may want to use a more robust method in your actual setup.

Normally, phpMyAdmin is accessible using the url: http://<XU3-Lite-ip-address>/phpmyadmin. However, since we used an obfuscation which was configured earlier, we will have to use the url http://<XU3-Lite-ip-address>/OYA16z2-xFg, which should display a pre-login dialog show as shown in Figure 2.

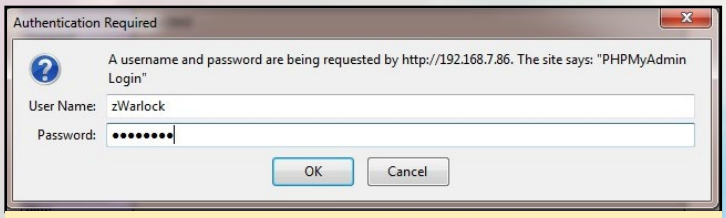


Figure 2: phpmyadmin pre-login screen

Use the same credentials specified in the nginx configuration earlier (username: zWarlock, password: birdsong). After clicking OK, it should take you to the actual phpMyAdmin login page similar to Figure 3.

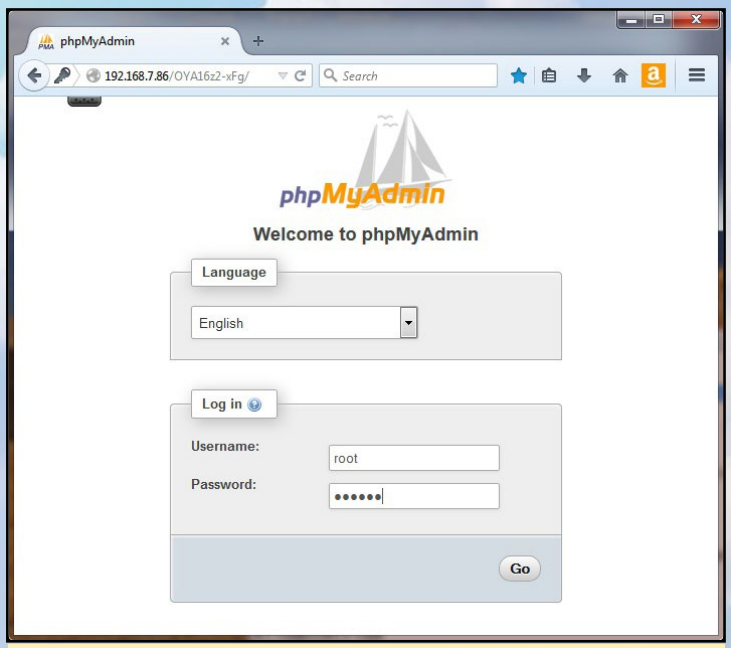


Figure 3: phpmyadmin login page

Note the use of the previously specified access credentials (username: root, password: odroid). Upon successful login, you will be presented with the home page as shown in Figure 4.

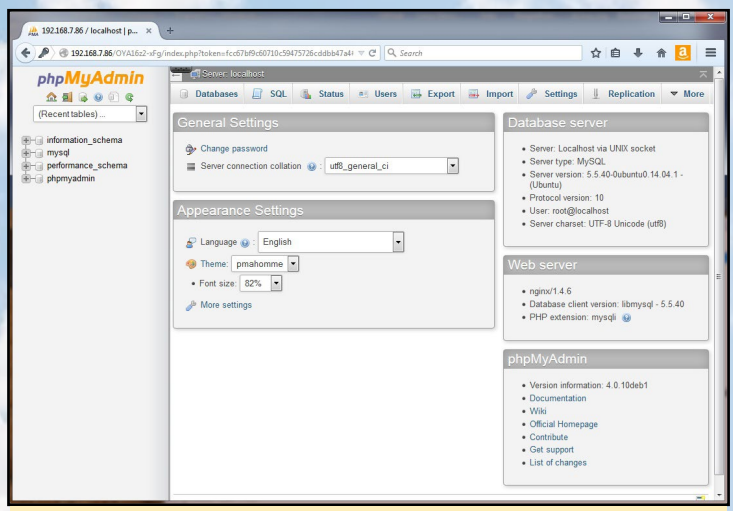


Figure 4: phpMyAdmin home page

Note the presence of the MySQL database in the pane to the left. Figure 5 shows the list of preliminary users in the database. Reboot the system prior to the final step.

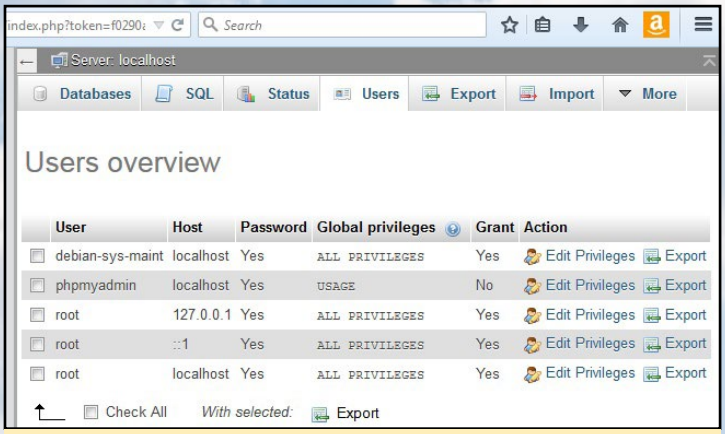


Figure 5: phpmyadmin showing preliminary users

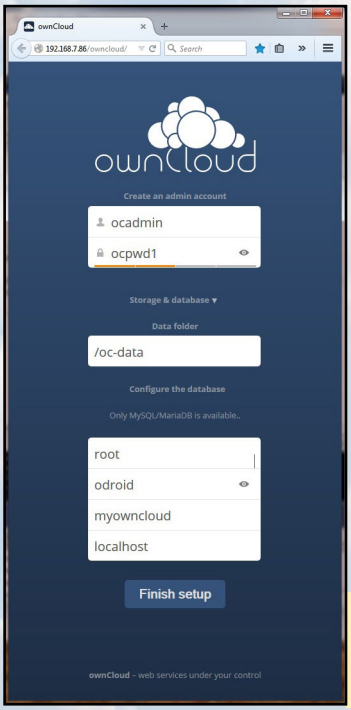


Figure 6: ownCloud penultimate step with data filled in

Complete the ownCloud setup

Access the ownCloud website using the url `http://<XU3-Lite-ip-address>/ownCloud`. You should be navigated to the ownCloud setup page. After the form is filled in with the desired information, it should look like the one shown in Figure 6. Note the setup of the credentials (username: ocadmin, password: ocpwd1).

Click on Finish Setup, which returns with a page listing the available desktop clients to be installed. We can check if the ownCloud database and users were configured properly using phpMyAdmin, as seen in Figure 7.

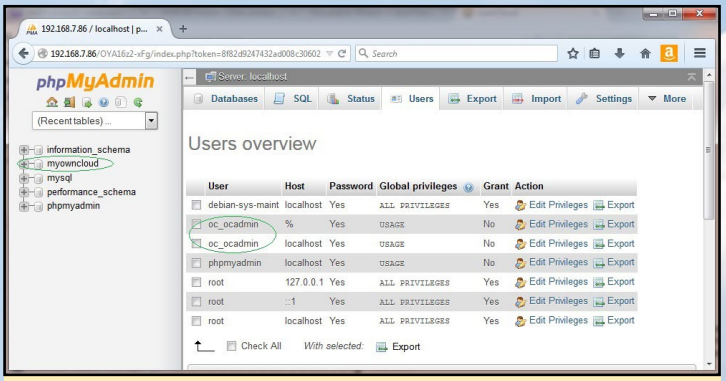


Figure 7: ownCloud database and users in phpMyAdmin

The ownCloud instance displays the page shown in Figure 8 indicating the available clients to be installed. Click on the Desktop app option, which displays a page similar to Figure 9.

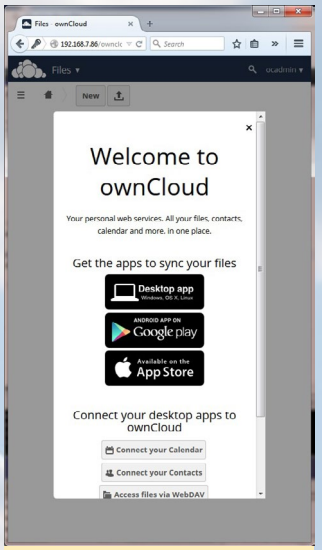


Figure 8: ownCloud client installation option

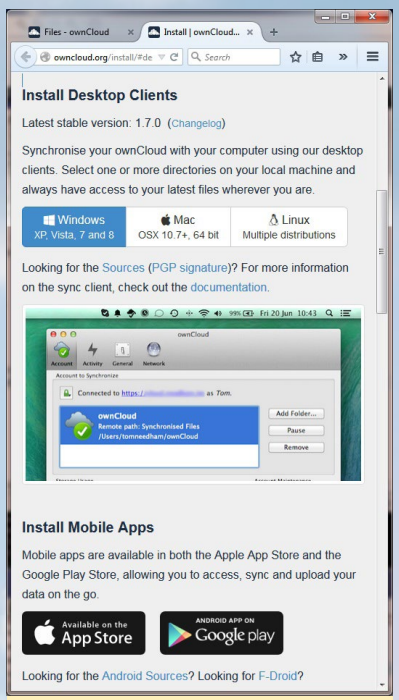


Figure 9: ownCloud desktop client install option

I selected the Windows option to be used on my Win7 system. You can select the option appropriate for your case. A prompt will appear to download the Desktop Client version 1.7.0. Select Save File, and after download is complete, run it to install the client. After a client login window appears, use the login credentials selected in Figure 6 (username: ocadmin, password: ocpwd1). A window will appear for the setup of local syncup as shown in Figure 10.

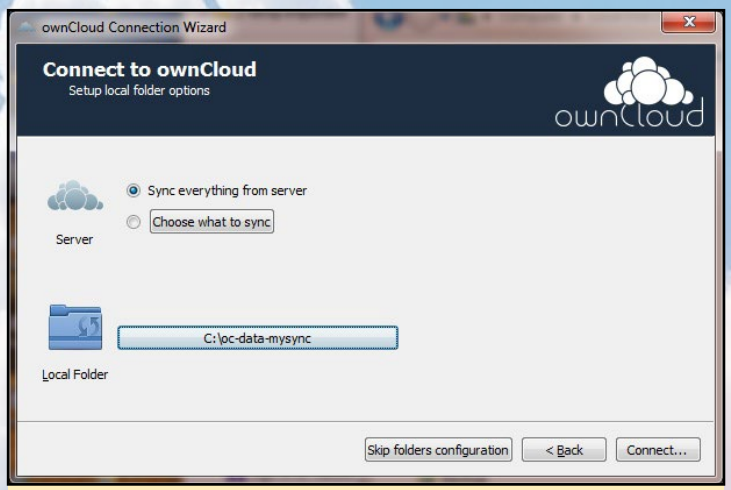


Figure 10: ownCloud client local syncup

Create a local directory at c:\oc-data-mysync to be used for the local syncup. Click Connect and wait for completion, which will display the final client screen. Filled in with relevant information, it should look like Figure 11.

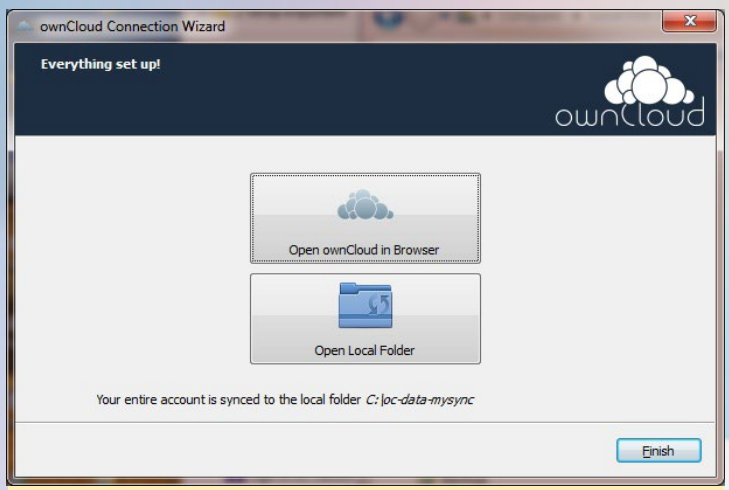


Figure 11: ownCloud client setup completion

Click on the "Open ownCloud in Browser" option, which will launch a browser-based ownCloud login page. Enter the credentials, and you should see a page like Figure 12. Finally, close the welcome window.

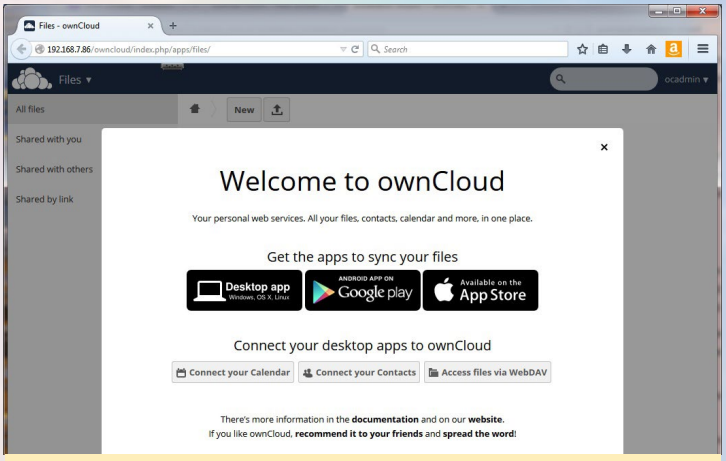


Figure 12: ownCloud client first login

The installation process modifies the ownCloud PHP configuration to the point where ownCloud disallows uploads of any files, and instead adding a special configuration for the use of an external third-party data storage application called ObjectStore. However, the whole purpose of a stand-alone cloud system is not to depend on external data storage. To address this issue, adjust the modified ownCloud PHP configuration to match those shown below:

```
$ sudo medit /usr/share/nginx/html/ownCloud/config/
config.php

'logtimezone' => 'America/Los_Angeles',
'log_rotate_size' => 1048576000, // ~1GB
'openssl' =>
array (
      'config' => '/usr/lib/ssl/openssl.cnf',
),
'mount_file' => 'oc-data/mount.json',
```

Next, remove the entire objectstore entry by searching for the "objectstore" tag, then deleting the entry. Save the file and reboot the system. Once the ODROID has rebooted, you can then proceed to create specialized folders for storing various types of files. I created the following folders, then uploaded some sample content to each of the folders for the purpose of verifying that ownCloud was running:

pix: to hold pictures
audio: to hold mp3 files
video. to hold mp4 and flv files

Figure 13 shows a popup window playing a video file through its native registered player directly within the ownCloud client session.

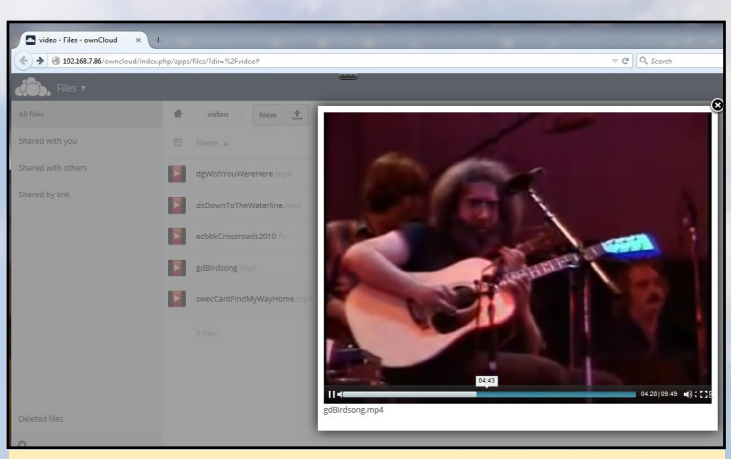


Figure 13: ownCloud client playing a video

To test that everything has been installed properly, you can perform one final check to see if the XU3 system actually has the files present, and is serving them correctly, by using the following commands:

HEAVENSTRIKE RIVALS

A CLASSIC RPG FOR THE FINAL FANTASY LOVER IN ALL OF US

by Bruno Doiche

A king of the RPG genre, Square Enix is a publisher that has consumed hours and hours of our lives with all of their games, so if you are running our latest Android release on your ODROID, don't miss the opportunity to take a look on Heavenstrike Rivals.

Enjoy a battle system designed especially for mobile devices that's easy to learn but with deep strategic possibilities, quick-entry player-vs-player combat, and hundreds of unique characters to collect, grow and evolve.

Challenge the world and ascend the rankings until you rule over all!

http://bit.ly/1BKpbrc


```
$ su
# cd /oc-data/ocadmin
# find .  -name "*"
.
./files
./files/audio
./files/audio/gdPeggyO.mp3
./files/audio/rre1759.mp3
./files/audio/jgBrokedownPalace.mp3
./files/video
./files/video/dgWishYouWereHere.mp4
./files/video/swecCantFindMyWayHome.mp4
./files/video/gdBirdsong.mp4
./files/video/ecbbkCrossroads2010.flv
./files/video/dsDownToTheWaterline.mp4
./files/pix
./files/pix/acharipicb.png
...
```

Wasn't that fun? Well, there you have it: one of the fastest and smallest high capacity scalable ownCloud systems available. Best of all, like nearly all ODROID-based projects, you can fit the whole server into your shirt pocket!

Additional configuration

- **Enable and test https access**
- **Enhance the configuration using the information at http://bit.ly/ItshZRW**
- **Develop a cron job using rsync to backup the ownCloud configuration, data, and relevant database information**
- **Delve deeper into apps built into ownCloud, such as calendaring**

install third-party apps to be hosted by ownCloud

- **Use external storage options such as Google Docs or Objectstore Swift**
- **And much more!**

For more information or questions regarding ownCloud, please visit the original information sources by clicking on any of the following websites:

- http://bit.ly/13Nlfeu
- http://bit.ly/1H8B8uo
- http://bit.ly/13NwWlo
- http://bit.ly/1rtFE33
- http://bit.ly/1tshZRW
- http://bit.ly/1kssJLF
- http://bit.ly/1D1R7s6
- http://bit.ly/1JWN0C6
- http://bit.ly/1Ez6ZXy

USER-CONTRIBUTED KERNEL REPOSITORY

MANAGE YOUR SOFTWARE PACKAGES WITH AUTOMATIC UPDATES USING APT-GET

by Tobias Schaaf

For some time now, I've been building Debian packages for easy installation of games and programs that I've ported to the ODROID, which are stored in the Hardkernel-sponsored server at http://bit.ly/13v98ly. You can manually download and install lots of software packages from this repository, which spares the trouble of compiling them on your own. However, I've recently improved the way in which these programs may be installed on your local system.

I've been experimenting with a simple Debian repository which will allow you to install packages with a simple "apt-get install" command, and to update already-installed packages with the "apt-get upgrade" command. I've recently developed it into a state where it's usable, and it is even able to update kernels with the "apt-get upgrade" command, which is a feature that was previously unavailable even from Hardkernel.

I want to share the current status of my first ODROID repository for people who want to try it out. I will most likely add more repositories whenever I see the need for it.

Getting started

Note that all of the following steps should be done as as root by typing "sudo su" and typing the administrative password, which is typically "odroid". First, navigate to the /etc/apt/sources.list.d/ directory on your distribution. Once in the directory, you can download any number of my repository files to this directory:

```
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-all-main.list
# main package list for all ODROIDs and all Distributions (Debian/Ubuntu)
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-all-U.list
# package list for ODROID U2/U3 devices and all Distributions (Kernel and
Headers)
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-all-X.list
# package list for ODROID X devices and all Distributions (Kernel and
Headers)
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-all-X2.list
# package list for ODROID X2 devices and all Distributions (Kernel and
Headers)
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-all-XU.list
# package list for ODROID XU devices and all Distributions (Kernel and
Headers)
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-all-XU3.list
# package list for ODROID XU3 devices and all Distributions (Kernel and
Headers)
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-all-C1.list
# package list for ODROID C1 devices and all Distributions (Kernel and
Headers)
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-all-testing.list
# package list all ODROID devices and all Distributions unstable packages
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-wheezy-main.list
# package list for all ODROID devices but for Debian Wheezy (ex. not for
Ubuntu 14.04)
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-wheezy-back-
ports.list
# package list for all ODROID devices but for Debian Wheezy (backports of
libraries ex. SDL2)
$ wget http://oph.mdrjr.net/meveric/sources.lists/meveric-wheezy-testing.
list
# package list for all ODROID devices but for Debian Wheezy (packages for
testing ex. XBMC 13)
```

Please make sure to only download the package lists suitable for your device. For instance, kernel updates for the ODROID-X2 won't work on an ODROID-U3.

Next, you need to download and install my signature key to tell the apt program that packages signed with that key are OK to use:

```
$ wget -O- http://oph.mdrjr.net/meveric/meveric.asc | apt-key add -
```

After that, you need to update the package lists with the following command:

```
$ apt-get update
```

If you've done everything right, it should run through without an issue, which means you are now ready to update and install packages via the "apt-get" command.

Kernel updates using apt-get

One of the most helpful features of the Debian repository is the option to update your kernels automatically via system updates. For this, I created a "meta-package" which will guarantee that your kernel will always be updated with the system updates. I will use the ODROID-U series as an example, but the same applies to ODROID-X and ODROID-X2, and other modern models as well.

First, make sure that you have the the following in your /etc/apt/sources.list.d/meveric.list file, and have already run the "apt-get update" command in order to download the most current package list:

```
$ deb http://oph.mdrjr.net/meveric/ all u
```

You can then install the following meta package to get all the Kernel updates that I provide:

```
$ apt-get install linux-headers-armhf-odroid-u
$ apt-get install linux-image-armhf-odroid-u
```

The headers package contains the header files for the kernel, which are sometimes needed if you want to compile your own kernel modules, such as when installing an external sound card or similar peripheral. The image package contains the actual kernel and modules. With these packages installed, you automatically get the newest kernel, and can automatically get updates using "apt-get".

Notes

When you already have one of my kernels installed, or whenever you receive an update, the previous kernel and header will be uninstalled before installing the new kernel. The system will complain about that, since you are uninstalling the currently running kernel and ask you if you want to stop this operation. You have to answer with "no" in order to continue with the installation.

Be careful with this step, because after removing the kernel, you should NOT restart the ODROID until the new kernel is installed, since your ODROID won't boot without a kernel. But don't worry, your ODROID will run indefinitely until you restart, even for days and weeks, which should give you plenty of time to fix any issues. If something goes wrong and the system gets restarted anyway, you can still repair your installation using another computer.

If you already have a kernel as a package installed created by me, you don't need to worry, since the steps above should work fine. If you have a kernel from HardKernel lower than version 3.8.13.26, you should be fine as well. However, if your kernel is labeled version 3.8.13.26 (type uname -a to see what kernel version you have) then you have to clean up the /boot directory first, since the kernel package might contain the same files that are already copied in your boot directory, which will prevent the package from installing.

```
$ rm -f /boot/*-3.8.13.26 for U3
$ rm -f /boot/*.3.10.51 # for XU3
```

Package list

This section contains a list of packages that can be found in my repository as of January 2015. The list will be updated without a corresponding announcement, so make sure to periodically check the forum thread listed at the end of this article if you want to know when packages have been added.

Package Name	Description
linux-headers-armhf-odroid-u	Meta Package for Kernel Headers of U devices
linux-image-armhf-odroid-u	Meta Package for Kernel Image of U devices
linux-headers-armhf-odroid-x	Meta Package for Kernel Headers of X devices
linux-image-armhf-odroid-x	Meta Package for Kernel Image of X devices
linux-headers-armhf-odroid-x2	Meta Package for Kernel Headers of X2 devices
linux-image-armhf-odroid-x2	Meta Package for Kernel Image of X2 devices
linux-headers-armhf-odroid-xu	Meta Package for Kernel Headers of XU devices
linux-image-armhf-odroid-xu	Meta Package for Kernel Image of XU devices
linux-headers-armhf-odroid-xu3	Meta Package for Kernel Headers of XU3 devices
linux-image-armhf-odroid-xu3	Meta Package for Kernel Image of XU3 devices
linux-headers-armhf-odroid-c1	Meta Package for Kernel Headers of C1 devices
linux-image-armhf-odroid-c1	Meta Package for Kernel Image of C1 devices
armagetronad-odroid-launcher	Meta Package for glshim version of Armagetron
chromium-bsu-odroid	Up-Down Shooting game using glshim
eduke32-odroid	Remake of Duke Nukem 3D using glshim
emulationstation-odroid	A graphical and themeable emulator front-end
freedroidrpg-odroid	Diablo game with Tux using glshim acceleration
hedgewars-odroid-launcher	Worms-like action game using glshim
libgl-odroid	glshim OpenGL -> OpenGL ES wrapper
libglew-odroid	libGLEW linked against glshim (for some games)
libglues-odroid	libGLU for OpenGL ES linked against glshim (needed for some games)
mario-odroid	Super Mario and Valves Portal mixed using glshim
neverball-odroid-launer	3D Puzzle game using glshim
neverputt-odroid-launcher	3D Puzzle/Golf game using glshim
opencpn-odroid	Naval map and route using glshim
shmupacabra-odroid	A hard and fast arcade shooter using glshim
smc-odroid	Super Mario Chronicles, using glshim
supertux2-odroid	Super Mario Clone with Tux using glshim
supertuxkart-odroid-launcher	Run SuperTuxCart (3D Mario Kart clone) glshim
sw-odroid	Shadow Warrior clone using glshim and OpenGL
valyriatear-odroid	Very nice looking RPG game
yquake2-odroid	Quake 2 remake in OpenGL using glshim
clementine-odroid	Music Player to organize your music and streams
mono-odroid	Mono (C#) lib and dev files for Debian Wheezy
retroarch-odroid	Retroarch Frontend for Libretro cores (Multi System Emulator)
xf86-video-armsoc-odroid	ARMSoC framebuffer drivers for Mali GPUs used for Exynos 4412 series

```
xbmc-odroid                 XBMC Gotham 13.2 for Debian Wheezy
antimicro-odroid            Tool for mapping keyboard events to gamepads and joysticks
clipgrab                    Tool to download movies from online websites, such as youtube or dailymotion
corsixth-odroid             Theme Hospital Clone (very funny Hospital simulation)
d1x-rebirth-odroid          Descent 1 Rebirth OpenGL ES version
d2x-rebirth-odroid          Descent 2 Rebirth OpenGL ES version
desmume-odroid              Nintendo DS/i Emulator
doom3-odroid                Famous 3D First Person Shooter
dosbox-odroid               ARMv7a optimized version of DOS Emulator
dunelegacy                  Dune 2 remake with enhanced features using SDL
etr-odroid                  Extreme Tux Racer OpenGL ES version
fheroes2-odroid             Heroes of Might and Magic 2 remake
flare-engine-odroid         Free/Libre Action Roleplaying Engine
ffmpeg-odroid               A complete, cross-platform solution to record, convert and stream audio and video
frogatto-odroid-720         Very good looking jump platformer where you play as a frog using GLES1 (720p binary)
frogatto-odroid-1080        Very good looking jump platformer where you play as a frog using GLES1 (1080p binary)
fs-uae                      Amiga Emulator with OpenGL ES 1 support
homeworldsdl-odroid         Port of the famous Real Time Space Strategy game Homeworld with OpenGL ES support
hurrican-odroid             Remake of the classic Turrican using OpenGL ES for lots of special effects
ioquake3-odroid             Open Source Quake 3 remake for OpenGL ES
ja2-stracciatella           Jagget Alliance 2 remake in SDL, allows to replay JA2 on your ODROID in FullHD
jk3-odroid                  Jedi Knight 3 - Jedi Academy for OpenGL ES
libsodium-odroid            easy-to-use encryption and decrytion library
mednafen-odroid             A MultiSystemEmulator which allows you to play GBA, NES, and many other console games
openggs-odroid              Great Giana Sisters remake..  C64 version, as well as total remake with different levels
openomf-odroid              Open Source remake of One Must Fall 2097
opentyrian-odroid           Arcade Shooter
openxcom-odroid             UFO: Enemy Unkown (X-COM: UFO Defence) remake with high resolution and new features
ppsspp-odroid               PlayStation Portable Emulator
retroarch-cores-good        Libretro cores for retroarch used in GameStation Turbo Image
retroarch-cores-bad         Additional libretro cores for retroarch not used in GameStation Turbo Image
rickyd-odroid               Rick Dangerous Clone using SDL2
scummvm-odroid              ScummVM Engine for multiple Adventure games
smw-bin                     Super Mario War - A fighting/Jump and Run inspired by Super Mario
smw-leveledit               Level Editor for Super Mario War
toppler-odroid              Toppler Tower is a Nebulus Clone in SDL
uqm-hd-odroid               Ur-Quan Master HD / HD remake of Ur-Quan Master (Star Control 2)

vcmi-odroid                 Heroes of Might and Magic III Engine to play HoMM3 on the ODROID
```

If you have questions about the repository, feel free to post on the original post in the ODROID forums at `http://bit.ly/1wEbfzC`, and I will try to help you whenever I can. It sounds complicated, but once the repository is setup properly, you can install and update programs, games and kernel with a simple "apt-get" command.

UPGRADE FROM 13.10 TO 14.04

STAY SECURE UNTIL APRIL 2019 WITH AN LTS RELEASE

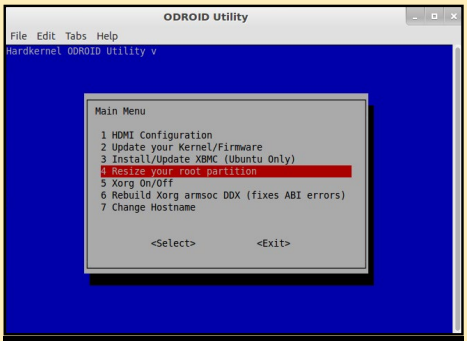
by Rob Roy

I've created a large public repository of Ubuntu 13.04 and 13.10 images over the past year, and have received several requests to upgrade them to Ubuntu 14.04. Because my library contains over 100GB of software and distributions, I have decided to share the instructions for upgrading any of my images to Trusty Tahr, which can be easily performed from the command line. Even though Ubuntu 13.10 no longer receives software and security updates, I personally still use the 13.10 versions because of their stability and reliability, but Ubuntu 14.04 is supported until April 2019, so it makes sense to upgrade to that version if you wish to have a secure system.

To begin, download any of my Ubuntu 13.10 images from the Hardkernel server at http://bit.ly/1rhHymu, copy it to eMMC or SD card, and boot it up. The first step is to download the ODROID Utility script:

```
$ sudo -s
$ wget -O /usr/local/bin/odroid-utility.sh \
https://raw.githubusercontent.com/mdrjr/\
odroid-utility/master/odroid-utility.sh
$ chmod +x /usr/local/bin/odroid-utility.sh
$ odroid-utility.sh
```

Select the option to resize the root partition, which will require a reboot. Once the desktop appears again, choose "Software Updates" from the Application menu, or type "sudo do-release-upgrade" in the Terminal window. Follow the prompts to upgrade, but don't reboot when asked. Run the ODROID Utility once again, and upgrade the kernel, firmware and video drivers.



The now classic ODROID-CI utility

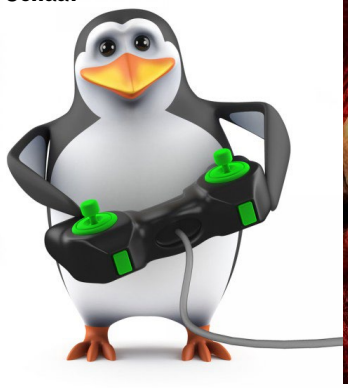
Once the ODROID Utility has completed its upgrades, reboot and verify that the new operating system has been installed by typing the following into a Terminal:

```
$ lsb_release -a
Distributor ID:    Ubuntu
Description:  Ubuntu 14.04.1 LTS
Release:    14.04
Codename:    trusty
```

LINUX GAMING

A COMPARISON OF THE GAMING POWER OF THE U3 VS XU3

by Tobias Schaaf



Recently, I was able to get a version of my popular gaming image ODROID GameStation Turbo working on the ODROID-XU3. Although it's not perfect, it gives a similar user experience to the U3 version. Now that I have the same image running on both U3 and XU3, it's time to compare both boards for their gaming power to determine which board performs better as a gaming platform, and what drawbacks there may be.

Overview

Obviously, the ODROID-XU3 has more USB2 ports, as well as a USB3 port, which is a big advantage over the U3, but also the eMMC module and hardware bus is a lot faster on the XU3. I used the command "hdparm -tT /dev/mmcblk0" to test the eMMC speed, and it reported that the read speed is about 80-90 MB/sec with an average of 84 MB/sec on the XU3. I then created a 4GB test file using the command "pv /dev/urandom > test.file" to evaluate the write capabilities, which utilized one core at 100% and reported a speed of about 4.2 MB/sec, which is not bad, considering that it's randomly generated data. After the file was created, I did another test by sending the newly generated file to /dev/null with the command "pv test.file > /dev/null", but this time, the results were somewhat different: the speed was reported at 117 MB/sec. I did the same test on a different XU3 with a different eMMC (one was 64GB, and the other was 16GB) with the exact same result. I redid the test on my Linux Laptop with a regular SATA hard drive, and got about 95 MB/sec, with the values sometimes dropping under 80 MB/sec. The ODROID gave a constant 115-117 MB/sec, so the read speed is very good. After that, I tested the write speed of the eMMC using the command "pv /dev/zero > test.file", which varied somewhat, but resulted in an average of about 30 MB/sec with spikes up to 35 MB/sec.

Notes

While I haven't done the exact same read/write test on the U3, since the speed of the eMMC is already known from previous tests, I did create a random data test file for comparison. What I noticed instantly was that the speed of the file creation was nearly the same. The U3 created the file at 4 MB/sec which is just about 5% slower than the XU3. However, on the XU3, the CPU temperature rose to 67-70°C with the fan constantly spinning on a high speed, whereas the U3 was running quietly at 50°C without even starting the fan.

Generally speaking, the XU3 is louder than the U3. Even at idle, the XU3 never goes under 55°C and the moment I started a task, one of the cores it goes up to 100% briefly, and the temperature jumps up to 65°C almost instantly.

The write speed of the XU3 is rather slow compared to its excellent read speed, and with a good microSD card you can probably achieve the same write speed as the eMMC. This also means that even if you're using the GigaBit USB3 LAN adapter you will never get more than 30 MB/sec when copying a file over the network.

Another fun fact (although not much of a performance test) are the results of the command "pv /dev/zero > /dev/null":

XU3 yields 3.2GB/sec @75°C CPU with the fan spinning full speed

U3 yields 3.5GB/sec @50°C with no fan spinning

XU-Lite yields 2.4GB/sec @56°C with no fan spinning

Another thing that I noticed is that the XU3 operating system is somewhat unstable. XBMC tends to crash the XU3 when switching between programs or movies too often. Even running games from the desktop can cause the XU3 to occasionally crash or hang. Although the performance of the XU3 is generally very good, this is somewhat of a downside, so you should position the XU3 in a way that you can easily restart it.

OpenGLES performance

The next experiment that I tried was running the glmark2-es2 demo in order to see how well the new Mali T628 of the XU3 performs compared to the Mali 400 of the U3. I was surprised to see that glmark2-es2 found OpenGLES 3.0 right on mark and was able to perform every single test there is while the U3 has some issues with a few tests.

I was also surprised in a different way with the results. In some tests, the ODROID-U3 was 5-10 FPS faster than the XU3, but the results varied a lot. In fact, both devices are slower than they should be, but I'm not sure what the problem is. While the U3 shows an average of 67 FPS, and went as high as 79 FPS in the test, I've seen the U3 performing much better in the past, with values of up to 109 FPS using the same benchmark. So something is slowing down this test, although it probably does not affect the overall experience.

The XU3, on the other hand shows an average of 66 FPS with the highest score at 73 FPS, but I've also seen values of 140 FP which indicates that the T628 in fact should have more power than it actually shows. Another anomaly is that the XU3 is unable to run the glmark2-es2 benchmark in full screen mode, which results in a still picture. However, the tests still seem to run in the background with a reported value of over 1500 FPS.

I also noticed that running the benchmark in window mode, but having the window in the background, has the same result with benchmark results of over 1500 FPS. I also wanted to run native OpenGLES games and compare the speed, so I used the ones that are the most demanding on the hardware, which are presented in the following sections.

Doom3

Doom3 has a timed demo which you can use to test the performance of your hardware. The demo runs through a level with different monsters, with a lot of different effects. It calculates the time the game needs to finish the demo, and gives an average FPS. On the XU3, the game had some slight issues. While turning quickly, some glitches appear such as tearing, with an unknown cause. But even with the glitches, the game is very much playable and gets a final result of 29 FPS, while the U3 gets 24.5 FPS without any of the glitches apparent in the XU3 tests. I'm not sure if the glitches are simply rendering issues, or if they are actually affecting performance, but even with the tearing, the XU3 performs about 18% faster than the U3 on this game.

Extreme Tux Racer, Homeworld, Jedi-Knight 3, Frogatto and UFO-AI

I haven't been able to find an FPS counter for Extreme Tux Racer, but I can tell by playing that the game runs full speed on both the U3 and XU3, but the XU3 has a video tearing issue whenever the camera moves. This turned out to be true for every game that I tried running natively under OpenGLES, even while using glshim. Homeworld, which uses OpenGLES 1.1, is working just fine. The tearing on the XU3 is still present, but nearly unnoticeable, since the camera never turns fast enough to make it visible. Jedi-Knight 3 started on the XU3, but was unable to draw a window, which means that only the game audio is working. Frogatto demonstrated the tearing issue as well, but runs smoothly besides that. The XU3 actually fixes an issue with transparency which is prominent on the U3, so the water looks better on the XU3 than on the U3.

UFO-AI surprised me by performing very well on the XU3. The U3 has issues with this game, which in my opinion, is the result of the texture buffer. You have to reduce the graphics a lot in order to be able to play it, and at some adjustment points, the graphics fail and the game crashes. It can only be played using a low texture resolution using the 256x256 pixel maps. If you're lucky, you can use 512x512, but it results in having graphical issues much earlier in the game. The XU3 can go up to 1024x1024 pixel maps, and seems to handle it well. But, starting at 2048x2048, the performance drops greatly when using battlescape mode, while the FPS counter remains at a steady 50 FPS while using the game menu and globescape mode. The U3 demonstrates far more issues than the XU3 when playing UFO-AI.

GLSHIM performance

Glshim is a wrapper for OpenGL which allows you to play certain OpenGL games on OpenGLES devices such as ODROID. It only supports OpenGL 1.x for now, and not all functions are available. Some games that use OpenGL are playable, but there are quite a lot that actually work properly. Therefore, glshim is a good test of performance, especially since some of the programs have high hardware demands.

Eduke32, SuperTux2, Chromium B.S.U., Hedgewars and Secret Maryo Chronicles

Eduke has some issues on the XU3. For instance, when running the game in full screen in the same resolution as the desktop, I receive an EGL error and I don't see anything on the monitor. However, I can run it in windows mode, but that drops the frame rate down to about 27FPS. When I use a different resolution for the game than the desktop resolution, the game starts with a slightly misplaced image, but it holds about 40-49 FPS with an average of about 47 FPS. On the U3, the game runs without issues at a steady 60FPS.

SuperTux 2 has acceptable performance on both devices. The XU3 suf

fers from the tearing issue while scrolling, but the U3 version is running fine. On the U3, this game runs at an average of 68 FPS, while on the XU3 it runs between 58 and 62 FPS.

My patched version of Chromium B.S.U. runs very well on the U3 at 1080p with about 50 FPS, although during play the FPS slowly decreases. On the XU3, the frame rate can go as high as 55 FPS, but sometimes also drop to 44 FPS. The game is still very playable, but has the same issues that I had encounter with other games, i.e., the game does not run in full screen at the same resolution as the desktop.

Hedgewars also does not work in full screen with the desktop resolution. Choosing a different resolution results in about 45-49 FPS while in window mode at 1360x786 resolution. Between 22 and 27 FPS can be achieved on the XU3, while on the U3 it runs in 1920x1080 full screen at a steady 60 FPS, and in window mode, it shows 40 FPS.

Secret Maryo Chronicles does not have an FPS counter, so my impressions are based on the look and feel. U3 performs awesomely at 1080p with full details, and the game is very smooth. Using the window mode at 1360x768 resolution was still good, but I could feel that it was struggling a little. On the XU3, there still existed the screen resolution issue mentioned above, but performance was acceptable. In fact, the window mode feels somewhat faster on the XU3 than on the U3.

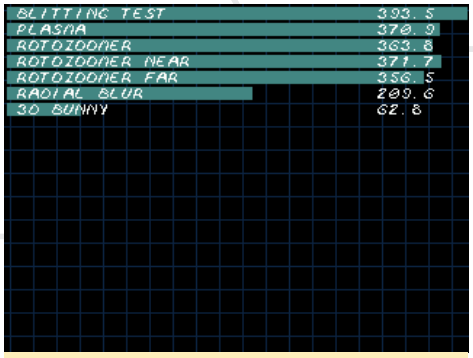
OpenGLES conclusion

I was disappointed with the OpenGLES performance of the XU3. I think it might just be an issue with OpenGLES 1.1, since Doom3 was in fact running faster than on the U3, but that could also due to the CPU power of the XU3. Unfortunately, there are only a small number of games that use OpenGLES 2.0 or even 3.0 on Linux, so it's hard to compare them with each other.

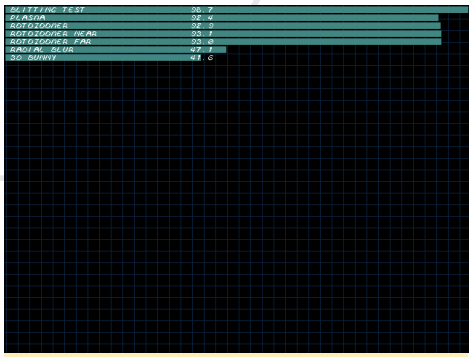
SDL Performance

Since OpenGLES showed poor performance in my experiments, I was looking forward to trying out the SDL performance, since its speed is mostly reliant on the power of the CPU. My assumption was that SDL should be better on the XU3 than on the U3.

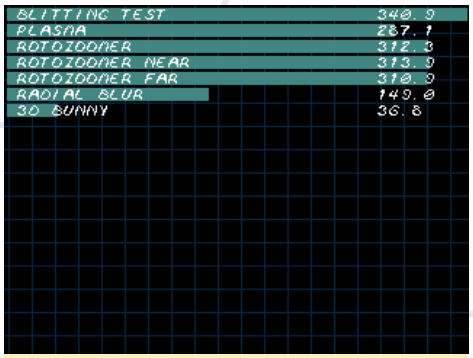
I ran a few games such as Jagged Alliance 2, Dune Legacy and freedroid RPG, as well as an SDL benchmark called gpmark for comparing he performance of the XU3 with the U3. As suspected, the performance of the XU3 is higher than on the U3. It even solves the issue with the full screen resolution which I encountered with OpenGLES applications, meaning I can run games



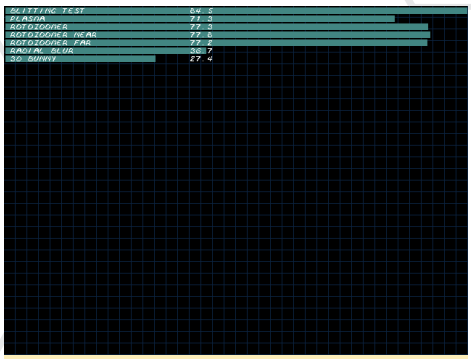
GPMark results 320x240 XU3



GPMark results 320x240 U3



GPMark results 640x480 XU3



GPMark results 640x480 U3

in the same resolution as the desktop using SDL.

Emulators

Emulators are a very good test for performance as well, since they often require a lot of CPU power along with some graphical power in order to emulate different systems. A very good example for this is Retroarch, which is a front-end for the well-known libretro cores, which uses different technologies to emulate different systems such as SNES, NDS, GBA, 3DO and many others. It uses OpenGLES 2.0 in order to display the content using hardware acceleration, but also uses SDL to draw the content, OpenAL for sound and udev for controller input. Having so many different technologies working together is rather demanding on the hardware, so it's expected that the XU3 performs better, since it has a more powerful CPU.

Retroarch 3DO Emulation

I chose a few CPU-intensive cores of Retroarch for testing, and one of the newest add-ons for Retroarch is the 3DO emulator, which is typically very demanding on the CPU. I used Super Street Fighter II to try out the performance of both devices. For some reason, I was unable to take screenshots directly from the U3 so I can only include pictures from the XU3.

Retroarch NDS Emulation

Although NDS is not the newest

Street Fighter II intro shows 49.2 FPS on the XU3 vs. 29 FPS on the U3

Street Fighter II title screen shows 58.7 FPS on the XU3 vs. 42.2 FPS on the U3

Street Fighter II gameplay shows 48.8 FPS on the XU3 vs. 27.8 FPS on the U3

core available for Retroarch, it has some occasional 3D rendering, which can be very demanding. With the recently added JIT compiler for ARM boards, NDS emulation runs nearly perfect on all ODROIDs, with some room for improvement. I used Rune Factory 3 and Bleach the 3rd Phantom for testing. Rune Factory 3 uses 3D characters and Bleach has some heavy background scrolling which are both very CPU-intensive.

Phantom, where you can choose to talk to friends and allies. In the background, the Bleach logo is constantly scrolling which eats up CPU power like crazy, giving 50 FPS on the XU3 vs. 42 FPS on the U3)

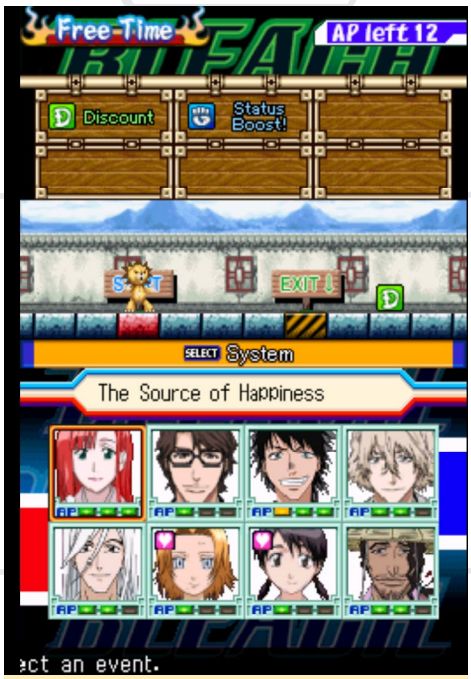
I could have performed more tests with the NDS emulator, but it's pretty clear that Retroarch performs better on

Rune Factory 3 title screen shows nearly 60 FPS on both the XU3 and U3

Rune Factory 3 gameplay shows 57.5 FPS on the XU3 vs. 45 FPS on the U3

the XU3 than on the U3, which is not surprising, since the XU3 CPU is much more powerful than the U3 CPU. The higher CPU power can utilize up to 50% more speed, but has on average, about 20-25% more speed than on the U3.

FS-UAE Amiga Emulation



A scene in Bleach the 3rd Phantom, where you can choose to talk to friends and allies. In the background, the Bleach logo is constantly scrolling which eats up CPU power like crazy, giving 50 FPS on the XU3 vs. 42 FPS on the U3

Although FS-UAE relies heavily on OpenGLES, it performs surprisingly well on the XU3. I discovered no issues with it, and the performance is very good. I chose a more demanding game for this test called Banshee. The AGA version requires expanded memory and a fast CPU.

I set both ODROIDs to emulate an Amiga A1200 with a 68020 CPU at the fastest speed possible, including 2MB chip memory and 4GB fast memory using Kickstart 2.04. The XU3 once again wins over the U3 through pure CPU power. While the game on the XU3 runs well in full speed without lags, the game is very slow on the U3, with stuttering sound and laggy gameplay.

Banshee is a great multiplayer cooperative game for the Amiga

PPSSPP Playstation Portable Emulation

PPSSPP is one of the best emulators available, and the performance and graphics it provides are just stunning, allowing you to play beautiful 3D games in high resolution. For this test, I disabled the frame skip option in order to see what the actual frame rate output and speed were using a 2x resolution.

I played Asphalt Urban GT2 and Ragnarok Tactics as an example, since I've traditionally used these games for testing the performance of the PPSSPP emulator while developing my GameStation Turbo images. Asphalt Urban GT2 is very demanding on the hardware, but will actually get slower if you increase frame skipping, and also has other degradation issues which, in other games, would otherwise boost performance.

Asphalt Urban GT2 on the ODROID-XU runs great with about 30-35 FPS on the XU3, and gets only about 13 FPS on the U3, but can run better on the U3 with the right settings

Ragnarok Tactics is a cute anime-style RPG/TBS game in the world of Ragnarok Online. Back when the PPSSPP emulator was still hard to get to working properly on the ODROID, it was one of the first game that I tried. There were different types of errors found in running the game, such as characters that were always facing in one direction no matter what, and the gameplay was not very fluent. However, I'm impressed at how far the performance of the game has come so far.

At the last moment, I decided to test one more game. Since people tend to

Ragnarok Tactics running in steady 60 FPS on the XU3 and 57 FPS on the U3 with waterfalls on the screen and many polygon models

enjoy fighting games, I chose Soul Calibur. I could have used Tekken 6 like HardKernel uses for their demos, but I find Tekken to be a rather dull fighting game with only a little action and mediocre graphics. I prefer Soul Calibur over Tekken, since even back on DreamCast, Soul Calibur always had stunning graphics, with lens flare reflections, very fluent character movement, swords, staffs, and all kinds of weapons.

Soul Calibur runs at 60 FPS on the XU3 and at 60 FPS on the U3 - so much for performance comparison!

Even though Soul Calibur runs at the same speed on the XU3 as on the U3, I feel that it's somewhat faster on the XU3. The menus react better, although after a short initial shock, the U3 is just as fast as his big brother. All in all, PPSSPP shows how well an emulator can leverage the hardware. PPSSPP even has an option to use OpenGLES 3.0, which theoretically should work even better on the XU3 and would offer more effects. However, the PPSSPP project is in a big restructuring phase right now because they are switching from SDL to SDL2, which unfortunately caused the newest version of PPSSPP to be temporarily unable to run on the ODROID platform.

Similar to the other emulators, PPSSPP makes it obvious that the XU3 still has reserve power during intensive emulation, whereas the U3 is often at its limit. Which means that, rather than 2x resolution, you would probably be able to use a 3x resolution on the XU3, which should enhance the graphics even more, making the games look like you are playing them on an Xbox 360.

Final thoughts

Although the performance of the XU3 is incredible, it has many flaws. XBMC is not working correctly, and the MFC decoder functions in XBMC are simply a clever hack, forcing the system to use MFC rather than checking to see if it's actually available.

OpenGLES seems to be somewhat broken on the XU3, even though version 3.0 is supported. Native OpenGLES games, as well as glshim, seem to run slower on the XU3 than on the U3, although the specs say it should have perform better on the XU3. Only Doom3, while somewhat glitchy, was actually able to use the higher performance of the XU3 to improve graphics performance.

This leads me to the conclusion that OpenGLES 2.0 (and probably 3.0 as well) are working fine on the XU3, while the OpenGLES 1.1 performance is worse on the XU3 than on the U3. Issues with vsync, screen resolution and tearing indicate that there are some incompatibilities with the XU3, which is probably an issue with the xf86-video-armsoc driver. It seems unable to handle the different modes as well as it can on the U3, which means that there's probably a solution if someone is able to fix the xf86-video-armsoc driver for the XU3.

On the other hand, when you start 3D acceleration through SDL, like some of the emulators do, the performance is very nice and there are no residual issues. The XU3 clearly shows that higher CPU power gives an advantage when it comes to emulating other systems, and the XU3 does a really good job with most

emulators.

Therefore, I would suggest that, as long as you want to play native OpenGLES games or games with glshim, stick with the U3 until the issues on the XU3 are resolved. However, for emulation, the XU3 is awesome and highly recommended, since all emulators can make use of the powerful XU3 CPU, giving better results, on the order of 15-50% over the U3.

Unfortunately, the XU3 crashes or freezes rather often, which diminishes its gaming experience. A workaround is to use an eMMC module, which allows the XU3 to reboot quickly, and if you can deal with having to occasionally restart the computer in order to keep playing, it's the perfect device for gaming and/or to use as a desktop replacement. Interestingly, the games available for the XU3 are stable, since the XU3 never crashes during gameplay, but only on starting or exiting. So at least while you play, you are safe from losing your progress.

All tests were done on Debian Wheezy using ODROID GameStation Turbo, so I can't say if the games that I tested would perform differently using Ubuntu 14.04. I'm also still in the process of evaluating Debian Jessie to see if it solves some of the issues mentioned above, so there may be room for improvement resulting from switching to an updated operating system.

You don't want to encounter Tobias in an online gaming session - he is extremely ARMed and dangerous!

COMMUNITY IMAGES

by Rob Roy

Hardkernel produces many pre-built images for use with the U3 and XU3 such as Android and Ubuntu, and some ODROIDians have created special-purpose distributions based on the official releases and shared them with the open-source community. Here is a short list of popular contributions that have been released on the ODROID forums:

OpenELEC
U3/XU3: http://bit.ly/1t6fWgr

Gamestation Turbo
U3: http://bit.ly/1nVvQqz
XU3: http://bit.ly/1ASFO5O

Cyanogenmod 11
U3: http://bit.ly/1ASG8BL
XU3: http://bit.ly/1qMA6Oq

Max2Play
U3: http://bit.ly/1HMovDY

Trusty Dev Centre
U3: http://bit.ly/1t6h1ov

Ubuntu Server
U3: http://bit.ly/1CMYC8K

Debian
U3: http://bit.ly/13zNTiG

Robotics (ROS + OpenCV + PCL)
U3: http://bit.ly/16TLG3V
XU3: http://bit.ly/1xlEPbZ

Android Pocket Rocket
U3: http://bit.ly/1H2Legq
XU3: http://bit.ly/1wrlB0L

Arch Linux (ALARM)
U3: http://bit.ly/1wOEzng

Kali Linux
U3/XU3: http://bit.ly/1sZsZ7x

DOCKER: DEVELOP, SHIP AND RUN ANY APPLICATION, ANYWHERE

PART I - GETTING STARTED WITH CONTAINERS

by Fred Meyer

Docker is a platform for developers and sysadmins to develop, ship, and run applications. Docker lets you quickly assemble applications from components and eliminates the friction that can come when shipping code, and lets you get your code tested and deployed into production as fast as possible. It consists of the following components:

- The Docker Engine, which is a lightweight and powerful open source container virtualization technology combined with a workflow for building and containerizing applications.
- The Docker Hub (https://hub.docker.com), which is a SaaS service for sharing and managing application stacks.

With Docker (https://www.docker.com/whatisdocker), you can manage to host many different applications on your single ODROID box concurrently, which becomes very easy to maintain. I have been running a miniDLNA Docker for several weeks now, and it is absolutely stable, serving music to my home. With Docker, you can run many popular Linux applications, such as:

- owncloud
- lamp
- openstack (dockenstack)
- node.js
- roundcube
- serviio DLNA/Server
- madsonic
- webproxy/webfilter
- DHCP/DNS-Server, like dnsmasq
- cloudprint (using cups)
- and many more

Linux distributions supported by Docker

Everything runs inside its own, lightweight Docker container. The Linux "system" inside each container can be based on CentOS, Ubuntu, Fedora, or ARCH Linux (to name a few) as personal preference, or as required by the application. This approach makes efficient use of the ODROID resources and at the same time keep your base/host operating system (OS) clean. Docker ensures that if something goes wrong with a single application, none of the other application containers will be affected,

Installing and configuring Docker is the first step toward a stable system

XU3 kernel requires a rebuild, and you will find the instructions on how to do that further down this article. You can also use ARCH Linux for the XU3, which comes with ready support in the kernel and a more upstream version of Docker.

With your ODROID up and running, install the Docker binaries from the main repository:

```
Ubuntu
$ sudo apt-get install docker.io

ARCH Linux
$ pacman -S docker
```

Base image

I suggest beginning with an Ubuntu 14.04 based image, as this is also the required base for building Docker from source. In general, this first step is ex-

plained at http://bit.ly/1tn21Z9. For the XU3, on ARM, some other tweaks are required.

A faster start for obtaining a base image is through the public Docker image repository, called the Docker Hub, which is available at http://bit.ly/1y1SMvO. I also added my manually built base images to the Docker Hub, and if you want to skip the steps of producing it for yourself, you can access and download them easily.

Ubuntu runs Docker well, and is the preferred operating system for many users

Note that the current versions of Docker and Docker Hub are not aware of the architecture for which the image has been built. All standard images are intended for the x86 architecture, and the autobuild feature offered by the Docker registry is only available for x86. However, Docker is Linux-based, and since Linux supports many architectures, other developers have added images from other architectures to the repository.

A well-known common naming convention has been established, where the contributors cite the architecture of the image inside the image name. For the ODROID architecture, look for images carrying "armhf" in the name while browsing the repository.

For convenience, my pre-built Ubuntu Trusty 14.04 base image is available through the public Docker repository at https://registry.hub.docker.com/u/hominidae/armhf-ubuntu. Type the following to fetch it for your own builds and Docker projects:

```
$ sudo docker pull hominidae/
armhf-ubuntu
```

Next, in order to run a test of your freshly created container by simply viewing the lsb-release file inside, type:

```
$ sudo docker run hominidae/arm-
hf-ubuntu cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=14.04
DISTRIB_CODENAME=trusty
DISTRIB_DESCRIPTION="Ubuntu
14.04"
```

If you'd like to build your image from scratch instead, the following sections illustrate how to do so, using an ODROID to perform the compilation.

Building from scratch

There is a way to create ARM based Docker images from an x86 Docker host, running a version of qemu-arm-static, as described at http://bit.ly/1CNgX5O, but I suggest doing this on your ODROID host instead.

Hardcore Linux hackers will find installing Docker to be a walk on the beach

Install debootstrap

The debootstrap utility is required in order to create a basic Debian/Ubuntu tarball. Follow the recommended steps for installation on your Linux distribution. For example, in ARCH Linux, install yaourt first.

Create minbase tree and tarball

On the command line, type the following, which will produce an ubuntu:trusty tree under the directory "ubuntu" relative to where you run the command:

```
$ sudo debootstrap --ver-
bose --variant=minbase \
--include=iproute,iputils-ping
--ARCH armhf trusty ./ubuntu \
http://ports.ubuntu.com/ubuntu-
ports/
```

Modify the sources list

Copying the sources.list from the XU3 stock ubuntu:14.04 image into your newly created ubuntu:trusty tree is a good start for creating a Docker image that is able to maintain itself through running a simple "apt-get update && apt-get upgrade". Type the following command into a Terminal window:

```
$ sudo cp /etc/apt/sources.list
./ubuntu/etc/apt/
```

The following line will create and add the tree/tarball as an image, named "ubuntu" and tagged "latest" to your local Docker repository on your host:

```
$ sudo tar -C ubuntu -c .  | sudo
docker import - ubuntu
```

Next, run a test of your freshly created container by typing the following command into a Terminal window:

```
$ sudo docker run ubuntu cat \
/etc/lsb-release \
DISTRIB_ID=Ubuntu \
DISTRIB_RELEASE=14.04 \
DISTRIB_CODENAME=trusty \
DISTRIB_DESCRIPTION=\
"Ubuntu 14.04"
```

Tweak the image

You normally will use a Dockerfile (http://bit.ly/1x8oBnN) to begin customizing and enhancing your base image. In the Dockerfile, you will instruct Docker to install further applications. During this stage, only the docker command is running inside the container, without any services (such as initd/systemd), which will make the install commands for certain applications/services fail.

Ubuntu Tweak is an application designed to make Ubuntu configuration easier

To circumvent this, you need to apply some tweaks/adjustments, as shown in the example at http://bit.ly/13KBsRm:

```
$ sudo docker run ubuntu echo
'#!/bin/sh' > /usr/sbin/policy-
rc.d \
   && echo 'exit 101' >> /usr/
sbin/policy-rc.d \
   && chmod +x /usr/sbin/policy-
rc.d \
   \
   && dpkg-divert --local --re-
name --add /sbin/initctl \
   && cp -a /usr/sbin/policy-rc.d
/sbin/initctl \
   && sed -i 's/^exit.*/exit 0/'
/sbin/initctl \
   \
   && echo 'force-unsafe-io' > /
etc/dpkg/dpkg.cfg.d/docker-apt-
speedup \
   \
   && echo 'DPkg::Post-Invoke {
"rm -f /var/cache/apt/ARCHives/*.
deb /var/cache/apt/ARCHives/
partial/*.deb /var/cache/apt/*.
bin || true"; };' > /etc/apt/apt.
conf.d/docker-clean \
   && echo 'APT::Update::Post-
Invoke { "rm -f /var/cache/apt/
ARCHives/*.deb /var/cache/apt/AR-
CHives/partial/*.deb /var/cache/
apt/*.bin || true"; };' >> /etc/
apt/apt.conf.d/docker-clean \
   && echo 'Dir::Cache::pkgcache
""; Dir::Cache::srcpkgcache "";'
>> /etc/apt/apt.conf.d/docker-
clean \
   \
   && echo 'Acquire::Languages
"none";' > /etc/apt/apt.conf.d/
docker-no-languages \
   \
   && echo 'Acquire::GzipIndexes
"true"; Acquire::CompressionTypes
::Order:: "gz";' > /etc/apt/apt.
conf.d/docker-gzip-indexes
```

Finalize the image

To save your image, it's necessary to commit the changes from above and tag the image before proceeding to the next step. First, fetch the latest container-id from the last run, then use that container-id (the first 3 digits will suffice) to commit the changes and tag the resulting new image:

```
$ sudo docker ps -l
$ sudo docker commit <id> ubun-
tu:14.04
```

Exploring Docker

You now have a working Ubuntu Trusty 14.04 base armhf-image to start from. Run this command to view your available images:

```
$ sudo docker images
```

At some point, especially whenever you start a new project, it is a good practice to bring your Ubuntu container up to date by adding the following line inside your Dockerfile:

```
RUN apt-get update && apt-get
upgrade
```

Also you'll want to save your image, snapshotting its "system" for further reuse, by typing:

```
$ sudo docker save <image-id>
<name>.tar
```

You can explore further docker-cli commands at http://bit.ly/13KDxwN.

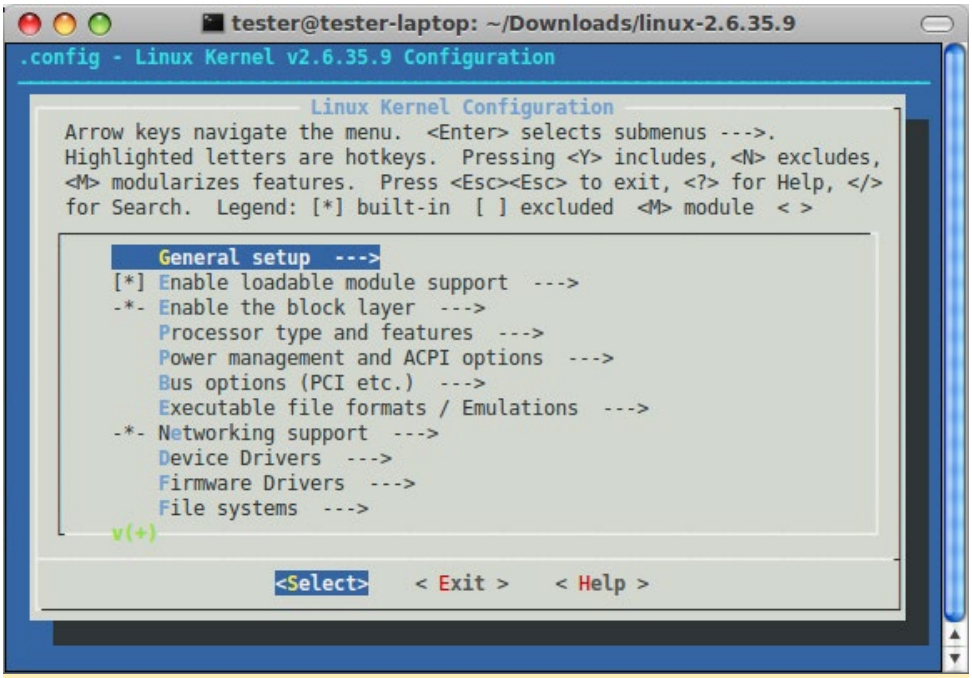
Upgrading Docker binaries on your host

Docker is still under development, so new features are constantly added, and the API is continually evolving. The good thing about Docker is that now you can upgrade your ODROID host to the latest version of the binaries if you wish. You can rebuild Docker, since it builds inside a ubuntu:14.04 container, and create/install the binaries by following the instructions at http://bit.ly/16U9epg.

As previously mentioned, Docker is not aware of the architecture on which it's installed, which means that the Docker container to build the docker binaries isn't either. In order to run the build on your ODROID host, you will have to do two tweaks to the source obtained from git.

The Dockerfile will build GO (https://golang.org) for all known/supported architectures. This obviously assumes that the build is running on x86-based machines, and that it is able to do a cross-compile. Just remove all architectures in the Dockerfile except for "Linux/ARM".

Secondly, the Docker daemon that



An example of the make menu config application, which is one of the essentials steps when building a kernel. There are a lot of options available, so take your time!

will build the binaries has a hard-coded check for the presence of the amd64 architecture in its code. You'll first need to remove the check for the amb64 architecture by editing the file docker/daemon/daemon.go in the source tree from git, in order to get a working Docker daemon.

Building the kernel

Ubuntu will be the default host OS for many users, using the kernel supplied by the Hardkernel team. The actual kernel build, however, does not include the proper settings to support Docker. The following steps will fix these settings, beginning with checking whether your kernel is already enabled for Docker.

Install lxc

Although lxc is not needed for Docker, since Docker comes with its own implementation called libcontainer, this package comes with a little tool to check/probe your kernel config.

```
$ sudo apt-get install lxc
```

Now check whether your running kernel needs a modified config, and if a rebuild is required:

```
$ lxc-checkconfig
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
Network namespace: enabled
Multiple /dev/pts instances: enabled

--- Control groups ---
Cgroup: enabled
Cgroup clone_children flag: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc ---
Veth pair device: enabled
Macvlan: enabled
Vlan: enabled
File capabilities: enabled
```

Should you see any of the above kernel parameters as not-enabled, you'll need to prepare a new kernel config and build a kernel from it. Note that you can point the lxc-checkconfig tool towards a kernel config-file, allowing you to test a kernel without having to boot into it:

```
usage: $ CONFIG=/path/to/config /
usr/bin/lxc-checkconfig
```

Prepare a suitable kernel config and build the kernel

First, fetch a kernel build tree. Note that the instructions shown here are for kernel 3.10.y. The instructions for kernel compilation are already laid out in the ODROID Wiki at http://bit.ly/1ATKTLh. Go to the "Linux" Section further down on that Wiki page, and read the section titled "Kernel Rebuild Guide" with the following additional steps:

1. In build-step 2 from the Wiki, during menuconfig, do the following config steps:

a. under Filesystems entry, disable support for XFS.

Kernel 3.10.y has a dependency config bug in its build tree...you won't be able to enable the next part, until you disable XFS.

b. under General -> Namespaces, enable "User Namespaces"

c. under General -> cgroup, enable all options

d. under Devices -> Character Devices,

enable "support for multiple dev/pts instances"

e. save the config and exit from menuconfig

f. re-check the new config:

```
$ CONFIG=./.config /usr/bin/lxc-
checkconfig
```

Hopefully your new configuration has all required features enabled now.

2. Continue to build and install the

kernel, as laid out in the Wiki.

3. After you have booted into your newly built kernel, check your kernel configuration again, type "lxc-checkconfig".

Congratulations! After completing these steps, you now have a kernel suitable for using containerized Apps with Docker on your ODROID.

Notes

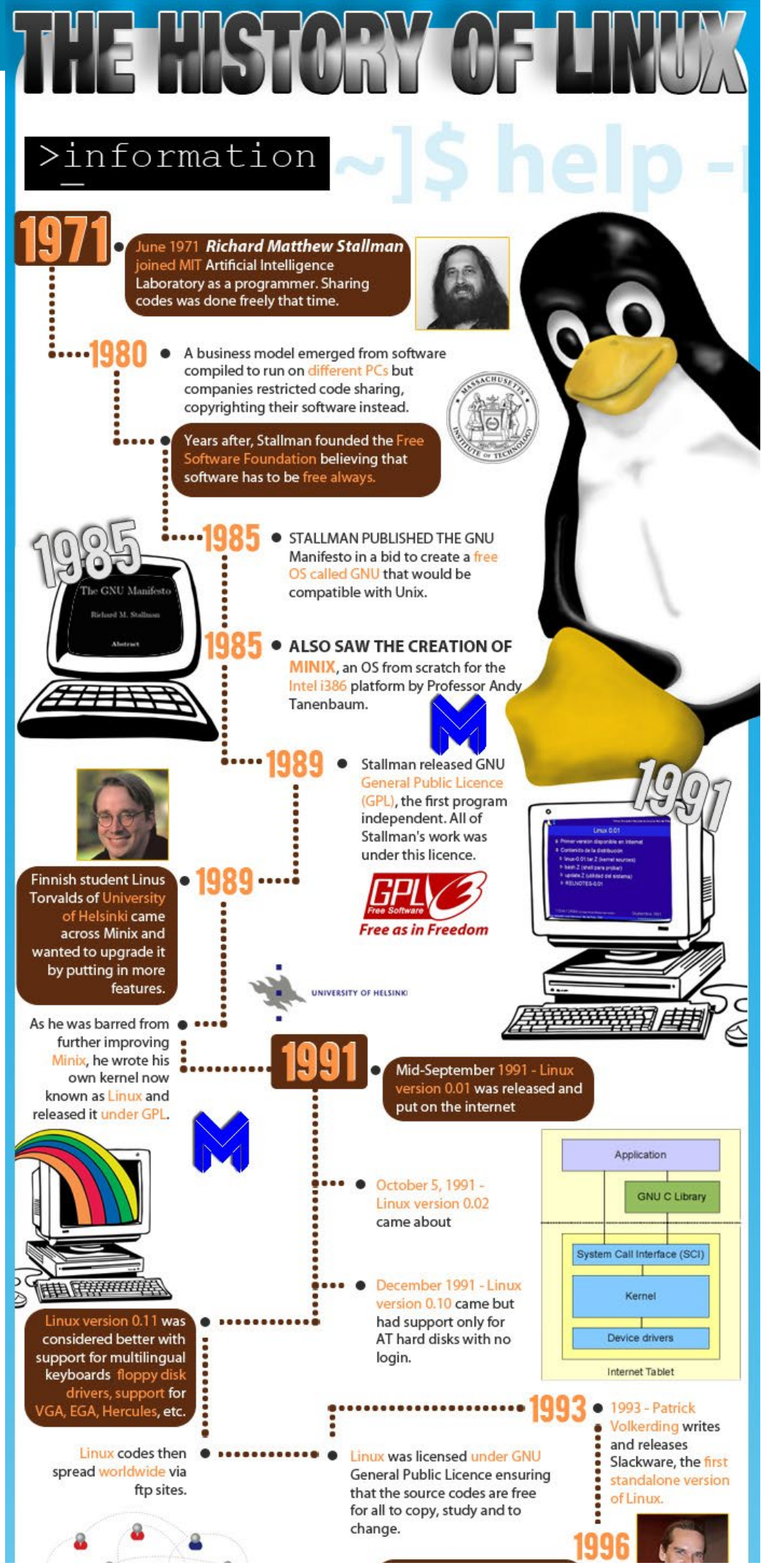
When enabling Docker in kernel 3.10.y, your host will lose the capability to support the XFS filesystem format. This is the main reason why the team at Hardkernel has not supplied a Docker-enabled kernel (yet).

If you have access to an ARCH Linux image for the XU3, there is a shortcut for establishing a working kernel config. Since ARCH Linux for ODROID-XU3 comes with a Docker-enabled kernel 3.10.y already, you can extract the config from there. Using a running ARCH Linux install, type the following in a terminal:

```
$ zcat /proc/config.gz > .config-arch && CONFIG=./.config-arch / usr/bin/lxc-checkconfig
```

Finally, fetch the .config-arch file and inject it into step 2 of the kernel build guide from the Wiki.

In the next part of this series, I will introduce some of my pre-built Docker images so that you can get your Docker system up and running even faster.



SETTING UP ASP.NET AND MONO

BUILDING A MICROSOFT-FREE SERVER STACK

by Rui Carmo

These are my notes on setting up a bleeding-edge Mono runtime and ASP.NET vNext on ODROID devices with hardware floating point.

Building Mono from Source

```
# I'm going to do everything as root
su -
# grab minimal dependencies
apt-get install build-essential mono-runtime autoconf libtool automake
# import required certificates
mozroots --import --ask-remove --machine

# Now check out the Mono tree (this alone will take ages if you have a slow SD card)
git clone git://github.com/mono/mono.git
cd mono
./autogen.sh --prefix=/usr/local
# grab the bootstrap compiler
make get-monolite-latest
# now is a nice time to go off and take a long stroll by the beach
make
# use this instead if you have distcc like me, it will speed up building the native bits:
# DISTCC_NODES="node1 node2 node3 node4 localhost" make -j5 CC=distcc
# now install it locally
make install
# Should report 3.10.1 (or above) and hardware floating point
mono --version
```

Sample output

```
Mono JIT compiler version 3.10.1 (master/8da186e Sat Oct 25 19:32:35 WEST 2014)
Copyright (C) 2002-2014 Novell, Inc, Xamarin Inc and Contributors. www.mono-project.com
        TLS:           __thread
        SIGSEGV:       normal
        Notifications: epoll
        Architecture:  armel,vfp+hard
        Disabled:      none
        Misc:          softdebug
        LLVM:          supported, not enabled.
        GC:            sgen
```

LLVM support

It's possible to set up the Mono LLVM fork to have Mono use LLVM instead of its built-in JIT, but it requires picking the right Git branch and passing both --enable-llvm=yes to autogen.sh and --llvm to mono itself which is not very useful, since the trade-offs in RAM/performance are debatable. Start up time, in particular, seems to take a sizable hit.

vNext

This script uses myget.org to fetch the nightly vNext package builds by Eilon Lipton, who works at Microsoft, so your mileage may vary depending on how stable the nightlies are.

```
# grab K tools
curl https://raw.githubusercontent.com/aspnet/Home/master/kvminstall.sh | sh && source ~/.kre/kvm/kvm.sh
kvm upgrade

# add the package repo certificates
sudo certmgr -ssl -m https://nuget.org
sudo certmgr -ssl -m https://www.myget.org
mozroots --import --sync
# run the samples
git clone https://github.com/aspnet/home
cd cd home/samples/HelloWeb
kpm restore -s https://www.myget.org/F/aspnetvnext/
```

For further questions regarding setting up .NET on an ARM device, please refer to the original article at http://bit.ly/1AZH3hW. This article was brought to you under the Creative Commons license (http://bit.ly/1jsHqrq).

ANDROID DEVELOPMENT: THE POWER OF ZYGOTE

by Nanik Tolaram

Building apps has become easier since the early days of Android development, and there are plenty of resources on the Internet that you can use to learn more. As an Android developer, you may understand the variety of APIs that are made available for your application, but sometimes you need to stop and think about how the app actually runs inside Android. Which part of Android is taking care of the app, and what is controlling it? This article will try to answer these questions.

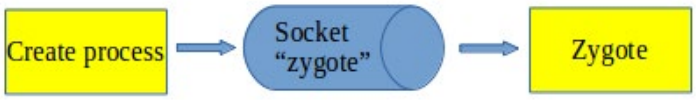
Zygote

We know that Android uses the Java Virtual Machine to run apps, and that this virtual machine is called dalvik, which was renamed art in Lollipop/Android 5.0. Dalvik is an implementation of a Java VM, but it is not the service that controls the launching of your application. There is another small component that controls the end-to-end process which is called Zygote.

Let's take a look what Zygote means in Wikipedia: "....In multicellular organisms, it is the earliest developmental stage of the embryo. In single-celled organisms, the zygote divides to produce offspring, usually through mitosis, the process of cell division." Ignoring the relevancy of the Wikipedia quote to biology, we can see that zygote is the replication of cells, which in the Android world means the replication of a process. In summary, Zygote takes care of the instantiation and replication of processes in conjunction with the virtual machine.

Every time you execute a Java application inside Android, you are triggering the launch process. Internally, the launch process is a straightforward yet multi-layered procedure, since it involves a number of different components talking and connecting with each other. At the highest level, the process works as shown in Figure 1.

Figure 1 : Application creation process



When you launch an application, you are instructing Android to create/fork a process, and this is taken care of by sending a socket request to Zygote when you start Android for the first time during the execution of the init process. Please refer to the December issue of ODROID Magazine at http://bit.ly/1x2sg6z for a further explanation of the init process. One of the main tasks of init is to launch Zygote, which makes it reside in memory waiting for an incoming instruction via its opened socket.

Zygote Init

Here is the init process command that launches Zygote:

```
service zygote /system/bin/app_process -Xzygote /system/bin --zygote --start-system-server
    class main
    socket zygote stream 660 root system
    onrestart write /sys/android_power/request_state wake
    onrestart write /sys/power/state on
    onrestart restart media
    onrestart restart netd
```

The above service command instructs Android to run zygote with the appropriate permissions. Note that the actual application is called app_main, but the service is referred to as Zygote. The following explains the different parameters passed to app_main:

--zygote: instructs the app_main application to run the program in zygote mode, where it initializes the environment and opens a socket.

--start-system-server: this is to instruct the app_main application to start the system server which requires different kinds of handling than normal applications. The system server contains several components that will be run as part of the init process.

The app_main application is used to launch apps, and is also used to launch Android's internal services. You can say that the app_main application is the "one-size-fits-all-app" for bootstrapping applications inside Android.

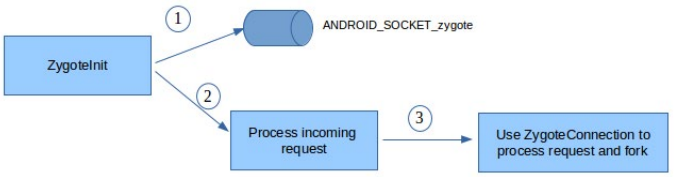


Figure 2 : ZygoteInit Initialization

Figure 2 shows the ZygoteInit class initialized during the init process, with two main steps:

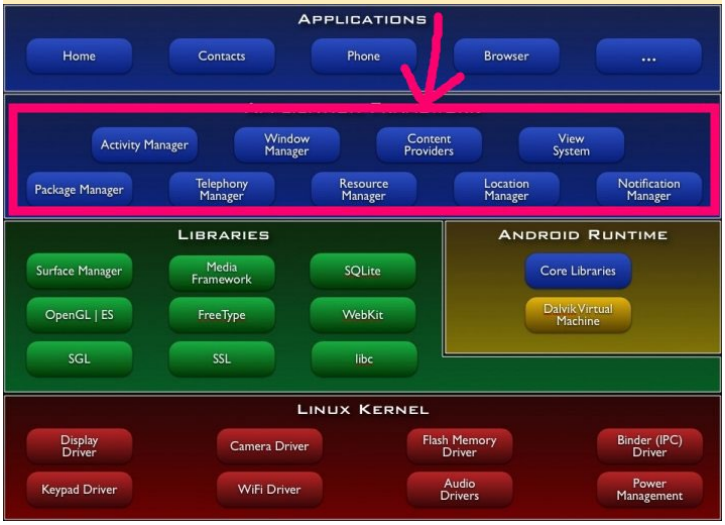
1. Create a socket called "ANDROID_SOCKET_zygote", which is used to receive incoming requests.
2. Process incoming requests to launch new applications and fork processes.

The zygoteinit class is the primary class that takes care of all zygote-related functionality, including preparing the environment for the new app to use, while the ZygoteConnection class is used to handle the incoming socket requests.

Launching Apps

Knowing that Zygote is the component that takes care of launching Android app, we can go a bit deeper by looking at the different classes involved in making this "magic" happen.

Figure 3 : Android Architecture



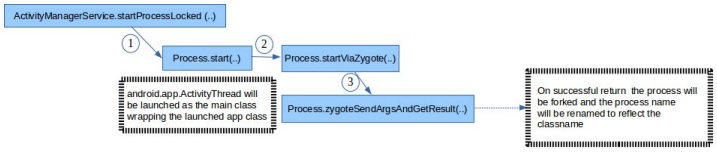


Figure 4 : App launch flow

As seen in Figure 3, we will look at the highlighted section of the Android architecture.

Specifically, we are going to review the Activity Manager, since this is the main service that takes care of the activity lifecycle of an Android app. The flow diagram in Figure 4 shows the classes that are involved when you want to run an app:

- ActivityManagerService is the main service inside Android that takes care of activities inside Android applications
- The Process class is responsible for mapping apps and processes created inside Linux

Steps 2 and 3 of Figure 4 shows the interaction with Zygote via the ANDROID_SOCKET_zygote socket as described in Figure 1. As you can imagine, without Zygote you wouldn't be able to execute your app, and the whole Android system would be rendered useless. Zygote is just a small component in the whole Android framework that helps the ActivityManager to launch applications in memory.

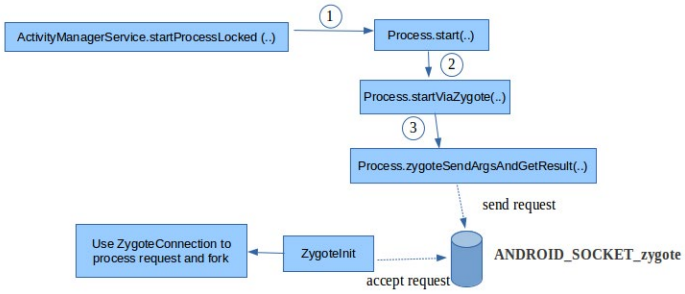


Figure 5 : Complete Zygote Flow

In summary, we can see that the whole interaction between the different layers will look like Figure 5. With a deeper understanding of Zygote comes better insight at how your app is executed inside Android.

FANCY GRAPHICS WITH JAVA: POIJU

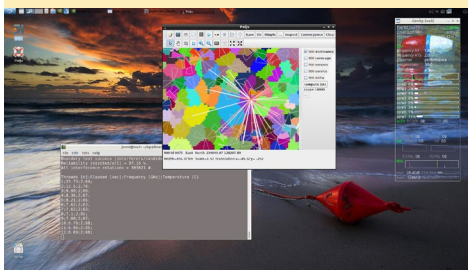
by Jussi Opas

In many applications, it is valuable to visualize the objects of interest using graphics; for instance, a floor layout, a drive route on a map, a design schema of a board layout, or a tool path of a numerical control program. For such purposes, primitives such as lines, rectangles, ellipsoids, or free form paths are used. Another means to create graphics are raster images, where each pixel contains an RGB color and sometimes also an alpha transparency value.

We created a sample application called Poiju, which runs on an ODROID-XU3, to help us visualize graphical representations of our experiments. In this article, we demonstrate the graphics capabilities of the application, present an overview of its drawing and imaging functions, and show how parallelisation with the octa-core processor of the ODROID improves computation performance using domain algorithms.

Overview

A desktop with the Poiju application and Conky monitor on an ODROID-XU3



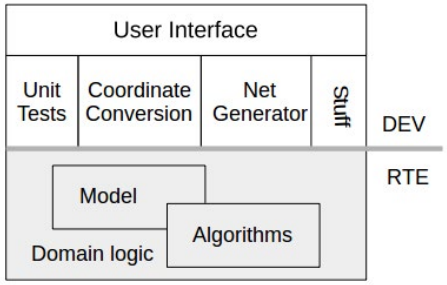
The Finnish word Poiju means a buoy. It is a sea marker that stays in its location on the water's surface, and keeps on moving by wind and waves. The Poiju application, as used in this article, models radio network and computational models in order to mimic network behavior. The application has been developed partially for actual use, and partially for fun and experiments.

The domain logic consisting of spatially relevant elements and related algorithms as deployed into a runtime environment (RTE). The algorithms are enumerated as:

- propagation model
- combined coverage
- dominance
- service area
- interference
- parallelization

The informal architecture of the Poiju application is shown in Figure 2. The development environment is used locally for verifying the domain logic as well as various other experiments.

Figure 2 - Poiju architecture



Swing and AWT

Visualization can be used to enable an application user or tester to verify the correctness of computations and intermediate results. Java offers its own methods to perform visualization. Java's Swing and AWT packages are capable of converting line drawings into graphics content as a Graphics class, and painting of Image instances into a graphical context. Painting in the UI is done using a view coordinate system whose origin is top left cornered, with the y-axis directed downwards and the x-axis directed to the right.

The AffineTransform class of Java is used to translate, rotate and scale view objects into their correct locations. Meanwhile, domain objects are mapped using another coordinate system. For instance, the location of geographical objects may be given as longitude and latitude or as northing and easting at a relevant UTM zone. The spatial cells of Poiju are first created as implementations of Shape objects and are painted only after that. Therefore, all shape functionality is available. For instance, it is easy to ask determine whether a mouse click hits an item on a map. Let's

have a look on how Java does drawing on an ODROID-XU3 while modeling multi-node networks.

Drawing

The figure below illustrates painting of cells with four different methods: border as aliased or with anti-aliasing on, and the content of the sectors with or without a fill. Visually, one would select

Figure 3 - 4 methods of cell painting

the last method, where anti-aliasing is on and the content is filled.

Intuitively, one would assume that aliased borders without filling is the fastest. One could also assume that filling takes more time. Then, it would be logical to determine that anti-aliasing is more time consuming, and that filling with anti-aliasing would be the slowest method. To decide which of the methods to use one can, of course, test it with a real-world application by collecting actual painting times in order to get accurate information. Painting times of 10000 cells are shown in the table below:

Line style and fill method	Time [sec]
Border aliased	0.41
Border aliased and fill	5.7
Border anti-aliasing	1.01
Border anti-aliasing and fill	1.7

The surprise is that painting with aliased lines with content fill is essentially slower than any other method, since it takes 5.7 seconds to paint. If the rule for using a progress meter is 2 seconds, then the application should use it while painting is performed. Performance-wise, it is feasible to use painting with anti-aliasing and fill, but an attractive outlook can be reached also with anti-aliasing without filling. We also recorded the time to draw 305000 interference lines. The results are shown in the table below:

Line style	Time [sec]
Aliased	9.9
Anti-aliasing	19.4

As seen in the table, in one second, 30800 aliased or 15700 anti-aliased lines can be drawn. Keep in mind that the Swing component is not thread safe. Therefore, only one thread can draw lines at a time ,and multiple cores cannot be used to accelerate painting.

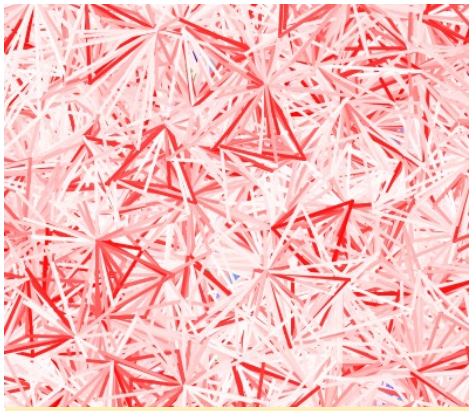
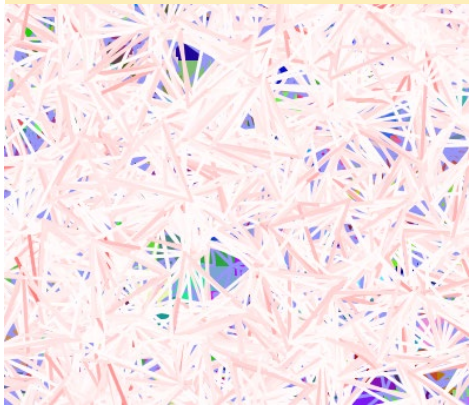


Figure 4 - Interference lines

As shown in Figure 4, all interference lines have been painted into a single image. Interference is represented as probability (CIP) and its magnitude is shifted on the interval [0, 100] %. The stronger the interference, the darker the red color. Respectively, a light color means that the interference is low between two cells. Painting of all interference lines at a time does not have a practical use case, but we show it here to demonstrate the graphical capability of the Poiju application.

All interference lines are computed

Figure 5 - FEP lines



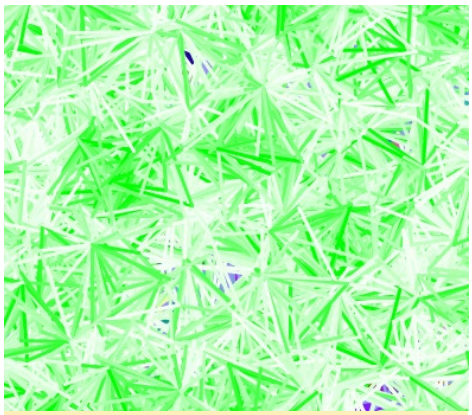


Figure 6 - ARP lines

with another interference method in Figure 5. The lines are lighter here, because the used mathematical method (FEP) is different from the previous image.

Interference lines in Figure 6 are shown as average received power (ARP), whose value interval is [0, 63] and the unit is RXLEV. A white to green color gradient has been used.

Images

With Java the other means to produce graphics is to create images. Once an image has been rendered, its painting is very fast. We cannot measure it with the System.currentTimeMillis method, because painting takes less than a millisecond. Therefore, time is used significantly only when the domain algorithms compute the content of a raster. Figure 6 shows what kind of graphics we can produce with image functionality.

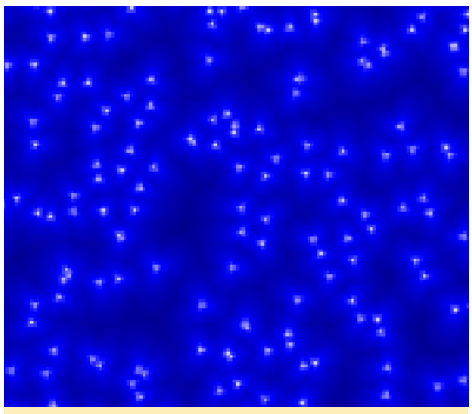


Figure 7 - Combined coverage

Combined coverage looks as if lighthouses were located around the reserved space. Each pixel has the field strength

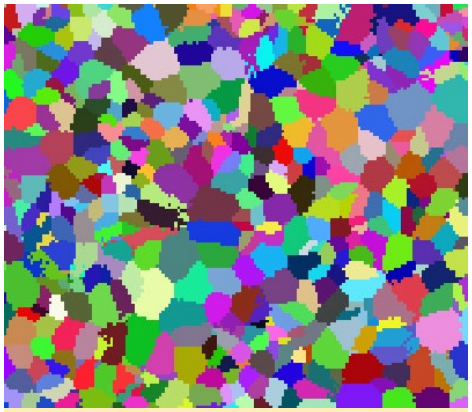


Figure 8 - Dominance

value of the strongest cell. Alternatively, coloring could be made using a different color gradient.

Dominance raster contains the long

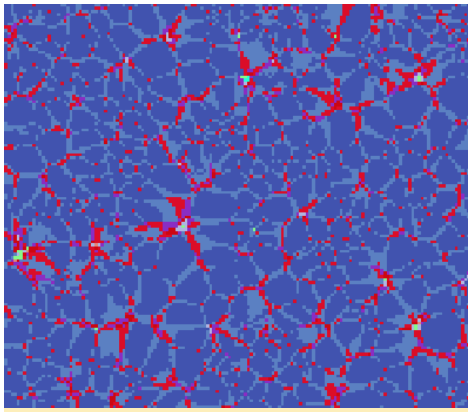


Figure 9 - Servers

identifier of the strongest cell at each pixel. Coloring is made so that a random color is given to each identifier.

Figure 9 shows how many cells are serving at each pixel. A handover margin of 2 dB has been used for the image. Blue pixels are served by one cell, light blue are served by two cells, red pixels by three cells, and so on. In a streaming technology network, multiple servers would mean a soft handover area, where data is transferred to a mobile by several cells at a time.

Layers

It is useful to use both images and drawing in the same image. It can be made by painting several layers of data into the same graphics context. For instance, there may be a image in the background, and then one or more layers are

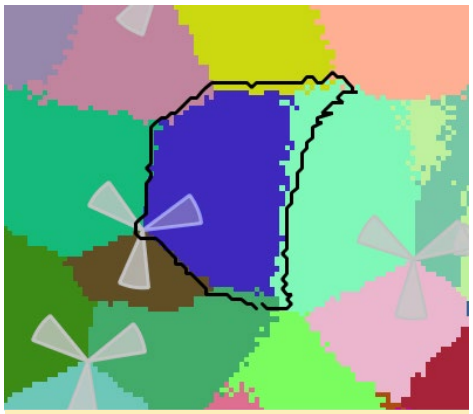


Figure 10 - Dominance and service area

drawn on top of it.

With the available graphics, a network can be inspected more closely. In

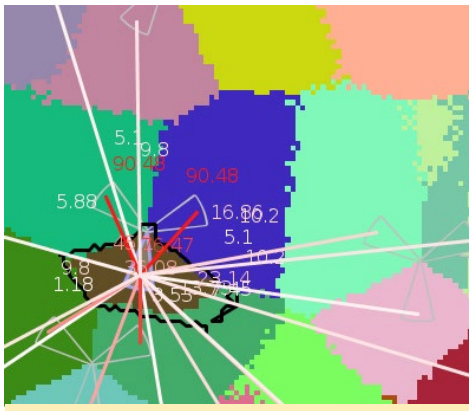
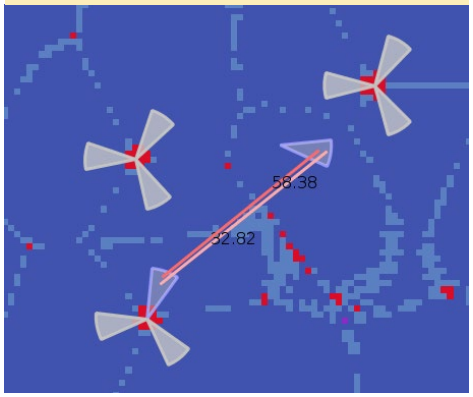


Figure 11 - Interference with values

Figure 10, the dominance area of selected cell is shown with blue color. The black border line shows that the service area is larger than the dominance area.

In Figure 11, interference lines of the selected cell have been painted with interference probability values. Also, the color of the painted values is changing in accordance with interference values. It is also possible to inspect interference values between selected cells. Figure 12

Figure 12 - Interference between two cells



shows a situation where two cells have been selected.

In the background, the number of servers' raster handover margin is 0 decibels. Hence, there are only a few pixels with several serving signals. Mutual interference between cells is unbalanced, since interference in one direction is bigger than the other direction (33% vs 58%). The interference line color changes accordingly.

Many dimensions

In an early phase of development, the application supported only line drawing capabilities and tables showing interference values. However, by using only tables, it is not possible to say whether the computed values are correct. Conversely, by line drawing on a map, it's immediately apparent that the values are sometimes biased or wrong. Not all interference relations were counted, but the specification function insisted that it had been given correct formulas, so it follows that the implementation function had made mistakes.

A software solution to this kind of problem is to develop graphics that show correctness or incorrectness of the applied mathematics and its implementations. The graphics must be created within the system and using the same language that is used in the implementation. Also, the graphical verification must be built into the environment that implements the formula. This is because Matlab has some functions that do not exist in the production language. A methodological question was whether it is possible to reasonably compute interference relation without making pixel traversals. The specification function wanted to avoid a pixel-based implementation, because previous implementations had been too slow, and development times had been too long.

At that time of development, test engineers were inspecting interference lines in a static network layout. It was

not possible to move cells or rotate their antennas. However, in a full black box test, one should go through all possible configurations. To implement that, we did the following experiment: two cells are located close to each others and the bearing of main antenna is changed incrementally with one degree. While doing so we got 360 lines, with each having 360 values. The resulting cell layout and resulting values are illustrated in Figure 13.

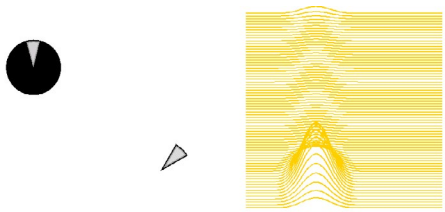


Figure 13 - An early interference graph

The black circle in the top left corner illustrates that an interfering cell has been drawn 360 times, once per each sample degree configuration. On the basis of the figure in the right side, two cells seem to interfere each other only when they are directed towards each others. This is erroneous, because interference is caused by the carrier cell when an interfering cell is shooting from behind. Based on this, the computation should be redone and corrected.

Figure 14 - Circle opposite 1-point stroke

After correction, the mutual interference relation between two cells is shown in Figure 12. The interference relation between two cells is continuous, and a back-shooting and side-shooting interfering cell causes interference to the carrier cell. We propose that other projects, old or new, could also use this triple axis method, <carrier bearing, interferer bearing, interference value>, to verify their definition and implementation.

Parallelization

Interference computation can be parallelized. ODROID-XU3 has 4 big and 4 little cores, which allows a Java program to run 8 threads simultaneously. Dominance and combined coverage for 10000 cells is computed in 0.4 seconds by XU3. A profile of parallel interference computation test with varying thread amount configurations is shown Figure 15. Interference for 10000 cells can be computed in less than 7 seconds. Meanwhile, drawing of those lines takes more time - 10 or 19 seconds depending on the aliasing style.

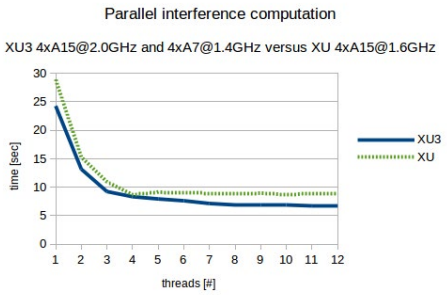
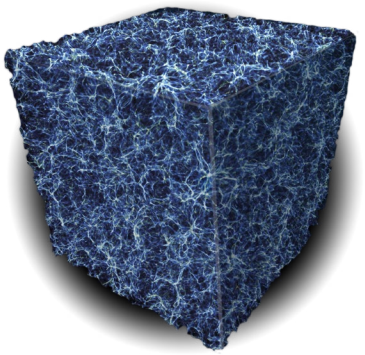


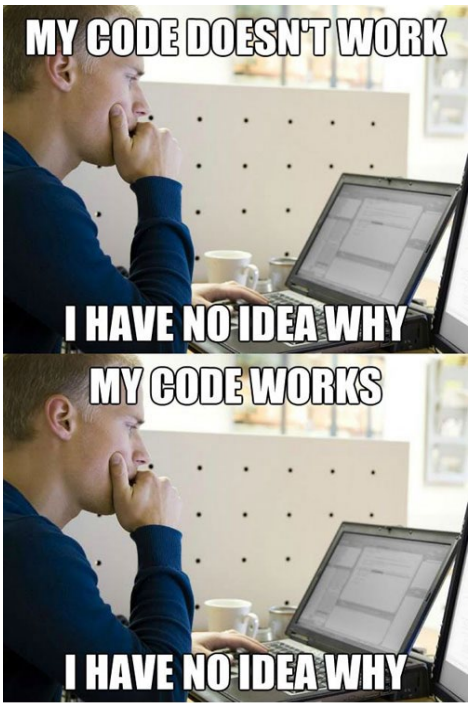
Figure 15 - XU and XU3 parallel interference computation

We can also compare heterogeneous multiprocessing capability of XU3 to the cluster switching used by the XU. The XU3 can run 8 threads at same time while XU runs 4 threads. Since the XU3 runs at 2 GHZ, it is faster already with one thread because the XU uses A15 cores which run at 1.6 GHz. With these figures, we say that there is no reason to avoid pixel-based implementations because of performance reasons.

Conclusions

We have shown that with Java, fancy graphics can be produced. Graphical visualization serves a solid purpose in showing what an application does. In domain logic, visualization is used to validate the functionality of the application during development. The current trend in software development is that testing is based on unit tests and on test automation. It does not justify the kind of work that has been shown here, which is considered as something that should be avoided. In spite of that, we have written a separate application with a diversity of purposes in mind: to verify model and algorithms, to learn, and just for fun. Meanwhile, we can also ensure that the domain models and algorithms work flawlessly using graphics. We have found that the octa-core ODROID-XU3 is an excellent platform for implementing and exercising parallelization experiments.





SEAFILE

PERSONAL CLOUD SOFTWARE

by @tlankford

Seafile is a cloud service that allows you to create and share a private cloud library of files with friends or colleagues. Files get synced for all users, so that if one person edits a document or makes any changes to it, Seafile automatically updates the changes for everyone in the group. It's safe, because you use your own servers, and reliable because Seafile saves everything, and you can even restore items that have been accidentally deleted. It's also secure because the files can be encrypted with a password.

This tutorial is an overview of the initial installation and setup of a Seafile server on an ODROID-U3. Seafile is a great platform for hosting a blog, small business server or family media server when coupled with an external 250GB USB hard drive.

Required components

The item list for this project is fairly short. I will explain what it is that I am using and what parts are interchangeable or optional. This can be set up as headless server, or with a monitor if you would like. I prefer to look at the first boot on the monitor.

- ODROID-U3 with power supply and HDMI cable
- Ethernet cable or Wifi adapter
- 16GB (or larger) eMMC or SD card as the boot media
- 250GB Linux-compatible hard drive
- Monitor, keyboard and mouse, which can be removed after the initial installation

Setup the image

I used SD formatter to format the cards and the Win32 disk imager from Hardkernel to write an Ubuntu image to the boot media. I do not want to go into too much detail as there are already lots of tutorials on how to do this.

First boot

Although the ODROID can be started as a headless node, and is ready for SSH on first boot, I like to have the desktop up for making early changes. Plug in the eMMC or SD card, monitor, usb dongle for keyboard and mouse, ethernet cable and apply power. If you have to enter a password and username they are as follows:

User = ODROID
Password = ODROID

The ODROID boots fast and the desktop appears immediately upon login. The first thing that I do is run the ODROID utility which is Hardkernel's version of the raspi-config command. I usually choose to change the hostname first, then expand the root partition and install the updated xorg files. If we were primarily using the desktop, we would also update the window manager and video drivers, but since we are not, we can skip that part. When you are finished with this step, you can reboot so that changes will take effect. Upon rebooting, launch a Terminal window so that so we can make a few changes. First, run the following command to check for an ethernet connection:

```
$ sudo ifconfig
```

We are mainly looking for eth0 and its IP address. Make a note of the IP address, then enter this command to customize the keyboard configuration:

```
$ sudo dpkg-reconfigure keyboard-configuration
```

Choose the UTF-8 configuration appropriate for your location by using the spacebar to deselect the default and select the preferred one. Next, enter the command:

```
$ sudo dpkg-reconfigure tzdata
```

This command allows you to choose the appropriate time zone, which is fairly self explanatory. Next, enter the command:

```
$ sudo nano /etc/ssh/sshd.conf
```

This will launch the nano editor so you can make changes to the script. In the default Hardkernel images, I have found that nano is not installed by default, so you can substitute vi for the nano command if you are more com-

fortable with the vi editor. If your prefer the nano editor, then run the command:

```
$ sudo apt-get install nano
```

After starting the editor, scroll down until you see the following lines and change them to match the following example:

```
Port 22
Protocol 2
PermitRootLogin yes
```

For security, you will want to change the PermitRootLogin option to "no" after the initial configuration. Save the file, then bind the configuration using the following commands, making note of the IP address:

```
$ ifdown eth0
$ ifup eth0
```

Finally, type the following to apply the changes:

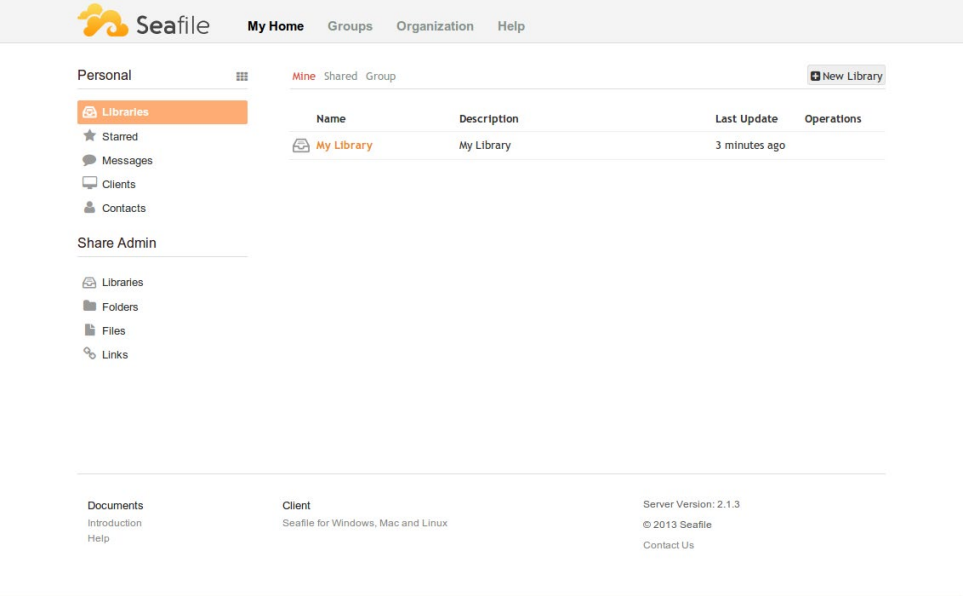
```
$ sudo reboot
```

Create a test sample

When you plug in the USB drive, the Ubuntu desktop should have a prompt to open the drive in the file manager. We want to save the drive path for later. For this example, the directory was /media/ODROID/OneTouch 4 Mini. You can change the location where the drive is mounted, but the default is fine for our purposes. Next, open a Terminal window and type the command, which gives a list of the available drives:

```
$ cat /proc/mounts
```

Ignore the drives listed containing with the letters "mmc", and look for the drive that is named /dev/sda*. The * is probably going to be a partition number 1. If there are multiple partitions mounted, then we will see them listed in order. Make sure to only have one external drive plugged in before running this command. Next, we want to see if the drive is usable as is, or if we are going to have to rewrite the drive. To test its usability, type the following command, using the drive path given by the file manager in the above step:




```
$ sudo nano mkdir /media/ODROID/
OneTouch 4\Mini/test
```

If you get a read-only error, then you will have to reformat, which requires a few simple steps. There are several ways to go about this, but we are going to use a program called parted in this tutorial. Be very careful and pay attention to what you are doing, since the parted application can overwrite the boot drive and corrupt the operating system, so be very careful when working with the drive scripts. First, type the following command, using the partition name found using the preceding "cat" command:

```
$ umount /dev/sda1
```

Finally, type the following to apply the changes:

```
$ sudo reboot
```

This will unmount the partition so that we do not get a "drive in use" error. Next, start the parted application and select the appropriate partition:

```
$ parted
$ select /dev/sda
```

The "select" command is very important to make sure we are using the right drive. We do not want to mess with the boot drive at all. Next, type the command, which will prompt to give a name to the drive. For this example, the drive will be named "labserv":

```
$ mklabel msdos
labserv
```

Next, type:

```
$ mkpart
```

When prompted, respond with the following answers:

```
primary
ext4
0%
100%
```

Finally, type the following to apply the changes:

```
$ sudo reboot
```

This specifies the partition layout, and you can change these values if you

wish to make multiple partitions, but for simplicity, we will only be writing a single partition in this example. Next, type the following to check that the partition has been written properly, then exit the partition editor:

```
print
quit
```

Finally, designate a file system for the partition by typing:

```
$ mke2fs -v -L labserv -t ext4 /
dev/sda1
$ umount /dev/sda
```

Once the drive has been unmounted, unplug it from the ODROID, wait a moment, then plug the drive back in. Check the file manager to see if the drive path for the hard drive is mounted properly. Since my example uses the label "labserv", the file manager shows that it is mounted in the directory /media/ODROID/labserv. In a Terminal window, type the following command, substituting the name that you chose for your drive for "labserv":

```
$ mkdir /media/ODROID/labserv/
test
```

If there are no errors, then remove the test folder by typing:

```
rm -rf /media/ODROID/labserv/test
```

If the test is successful, that means that the drive is formatted properly and ready for use with the server. Shutdown the ODROID so that you can unplug the monitor and keyboard by typing:

```
$ sudo shutdown -h now
```

Installing MySQL

For the rest of the steps, it's necessary to login using SSH. I am not going into much detail regarding SSH, since there are many available tutorials on the specifics. I used PuTTY to establish the connection by installing it from http://bit.ly/1jsQjnt. After PuTTY launches, you will be asked for the IP address of the ODROID from the earlier step. Make sure that the port is set to 22. Press enter and wait for the Terminal window to appear, then login as ODROID with the password ODROID. You should see the same Terminal prompt that you would see if using the native Terminal application on the desktop, which looks like this:

```
ODROID@ODROID~$
```

If you see this prompt, it means that you are in your home directory. We have made a few changes, so we want to run the following commands, answering "y" when prompted:

```
sudo apt-get update
sudo apt-get upgrade
```

Next, initiate the installation of MySQL by running the command:

```
$ sudo apt-get install mysql-
server mysql-client php5-mysql
```

You will be prompted to set a password for the user "root". After typing your chosen password, run two more commands, typing "y" for each of the answers, except in response to changing the root password. Make sure to remember the root password, since it will be used during the setup of the Seafile server.

```
$ sudo mysql_installation_db
$ sudo mysql_server_installation
```

Installing Seafile

There are two dependencies that the software will for during installation so we are going to preempt this error by pre-installing them, answering "y" when prompted:

```
$ sudo apt-get install python-
setuptools python-simplejson
```

Next, install the Seafile server application, which offers a nice interface to the Seafile server using a standard browser. Create a temporary directory called seafile to store the files:

```
$ mkdir seafile
$ cd seafile
```

Next, download and unpack the package into the newly created directory:

```
$ wget https://bitbucket.org/hai-
wen/seafile/downloads/\
seafile-server_3.0.4_pi.tar.gz
$ tar xzvf seafile_server*
$ cd seafile_server*
$ ./setup-seafile-mysql.sh
```

Follow the prompts and just choose defaults, except for two items: when asked for server name, enter your IP address, choosing the default ports. Also, choose to set up new MySQL databases. After all the questions are answered, you should see a list that looks like this:

```
Server Name: Your Server Name
IPAddress: <ODROID IP address>
ccnet server port: 10001
seafile data: /media/ODROID/lab-
serv/seafile-data
seafile server: 12001
HTTP server: 8082
MySQL server: 3306
ccnet database: ccnet-db
seafile database: ccnet-db
seahub database: seahub-db
```

During the setup, you may be asked to enter to the MySQL root password that was chosen earlier. Next, start the Seafile server and seahub application by typing the following commands into a Terminal window:

```
$ ./seafile.sh
$ ./seahub.sh
```

When asked to enter the administrative email, type in your preferred email address, then set a password for the browser login, which should be written down for future use. Once the Seahub application is setup, you can check it using a local browser by navigating to http://<IP address>:8000, where <IP address> matches the one from the earlier "ifconfig" command. Finally, login and check out your new cloud server!

For further information about setting up Seafile on the ODROID, please refer to the original article at http://bit.ly/1rIb9Te, or the Seafile home page at http://seafile.com.

The man in the cloud wanted us to tell you that he enjoys passing the time by browsing through your collection of funny cat memes and gym selfies

HISTORY OF ODROIDS

by Rob Roy

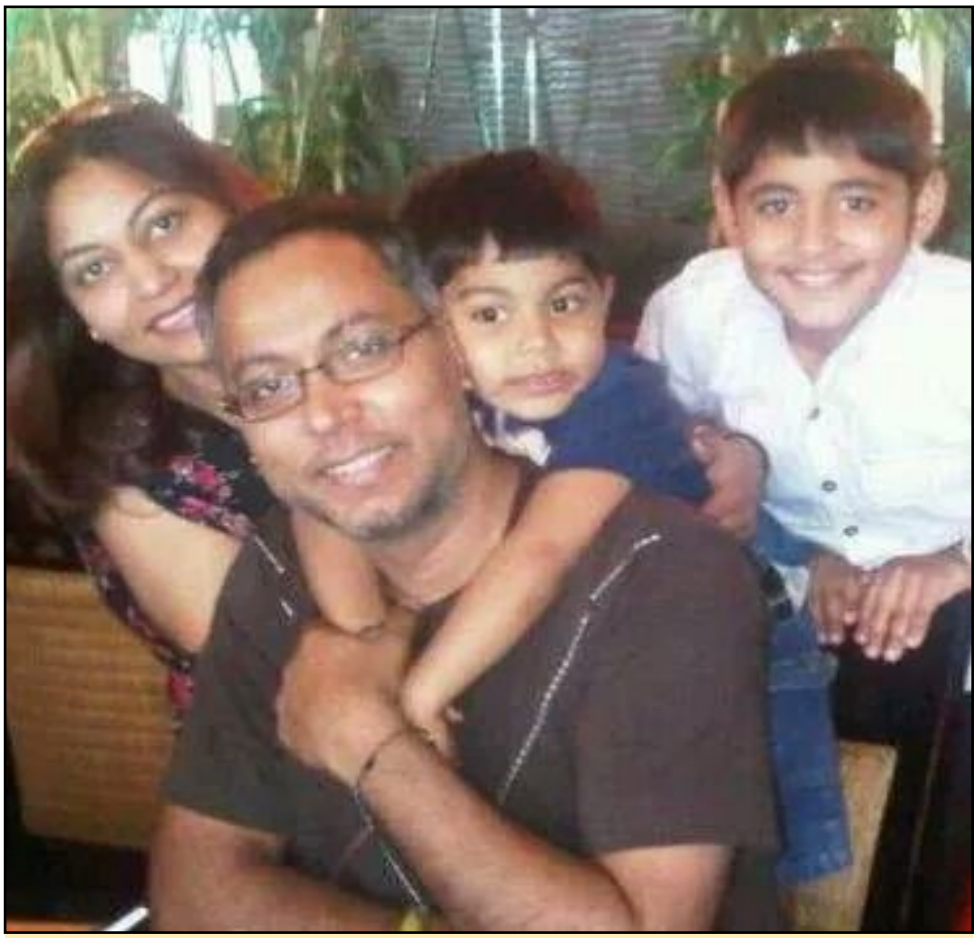
Wikipedia maintains a history of Hardkernel's products, which includes a chart of the specifications for every ODROID device since 2009. Check it out at http://en.wikipedia.org/wiki/Odroid:

Name	CPU
Odroid 2009	Samsung S5PC100 833 MHz, Cortex-A8
ODROID-U2	1.7 GHz Exynos 4412
ODROID-X2	
ODROID-U3	1.7 GHz Exynos 4412 Prime
ODROID-XU	Exynos 5410 Octa big.LITTLE ARM Cortex-A15 @ 1.6 GHz quad-core and ARM Cortex-A7 @ 1.2 GHz quad-core CPUs
ODROID-XU3/XU3-Lite	Exynos 5422 Octa big.LITTLE ARM Cortex™-A15 @ 2.0Ghz (Lite @ 1.8Ghz) quad core and Cortex™-A7 quad core CPUs
ODROID-W[3] ***(discontinued)***	Broadcom BCM2835 ARM11 @ 700 MHz
ODROID-C1[4]	Amlogic S805, 4× Cortex-A5 @ 1.5 GHz

MEET AN ODROIDIAN

NANIK TOLARAM: JAVA JEDI

edited by Rob Roy



Nanik surrounded by his family. Kudos for happiness above all else!

Please tell us a little about yourself.

I live in Sydney, Australia, and am married to my best friend with our 2 beautiful and playful boys who are 5 and 10 years old. On a day to day basis, I work as an Android Platform Engineer building a custom Android platform called ScreenerOS (`http://bit.ly/1wjixnr`) that is compatible with both x86 and ARM platforms.

How did you get started with computers?

I started working with computers when I was 9 years old. My first computer was an Apple computer (can't remember the model) and my first exposure to programming was with the BASIC language. I started taking a deep interest in computers when, at 11 years old, I got infected by a boot sector virus called Denzuko, and that's the time that I started learning everything I needed to know in order to clean the virus. Since then, I realised that I have a passion in learning how things work inside computers. I wrote my first computer book when I was 17 years old, which was a C++ programming book.

What drew you to the ODROID platform?

I came across ODROIDs when I was doing research on embedded open source hardware systems, and found out that it can run Android, and the price was reasonable. It was at the same time that I came across a posting in the forum that they were looking for an Android columnist, so I jumped at the chance, as I was also looking for an avenue to share my Android knowledge with the community. I guess it's my own way to give back to the community since I've learned a lot from the open source community.

Which ODROID is your favorite?

The ODROID-U3 is my all time favourite, because it's small and powerful!

How did you become so proficient in Java?

I started learning Java back in 2004. I found out about Java by coincidence when I was doing research about the internals of virtual machines and the potential that they can bring. I discovered a lot about Java and J2EE by reading the implementation source code of the various Java Specification Request (JSR) documents. I've been working with Java ever since, as both a hobby and professionally.

What hobbies and interests do you have apart from computers?

Since I was young, I have always had a keen interest in tropical fish and breeding them, especially goldfish and discus. A new hobby that I've picked up lately is making electronic projects with my boys, as well as woodworking, since I love to design things from scratch.

Are you involved with any other computer projects unrelated to the ODROID?

I'm actively involved in doing presentations for the Sydney Android community (http://bit.ly/1EbknRo), and also getting myself involved with the MinnowBoard Max communities.

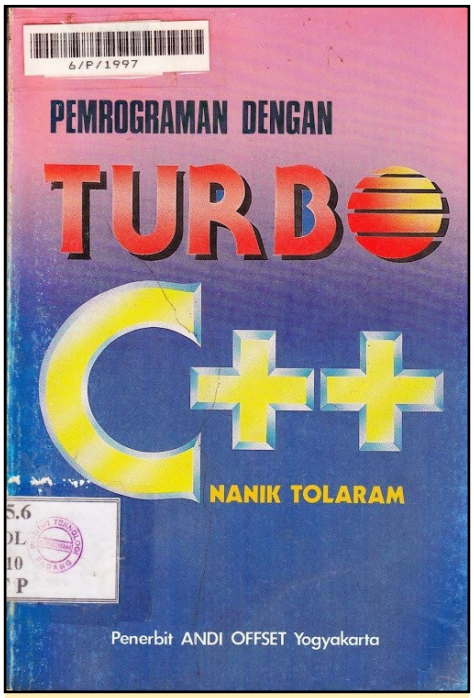
What type of hardware innovations would you like to see for future Hardkernel boards?

One thing that I find currently lacking is the availability of sensor accessories for boards specifically for Android. We know that most Android devices such as phones and tablets have varieties of sensors which do not exist for many development boards.

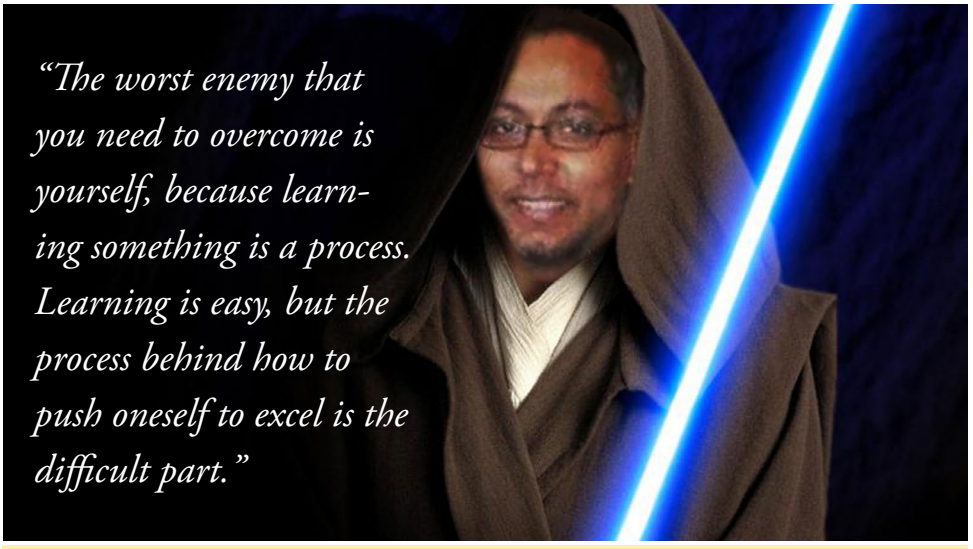
What advice do you have for someone want to learn more about programming?

Learning programming has never been easier nowadays, as there are so many good websites and communities that can help out with any kind of problems. Open source has changed the way in which people are learning, and it's a good opportunity for someone to learn what interests them.

I always tell people that the worst enemy that you need to overcome is yourself, because learning something is a process. Learning is easy, but the process behind how to push oneself to excel is the difficult part. Build the motivation and desire to learn programming since programming is a combination of science and art. Read lots and lots of source code because the best documentation is inside the source. Making mistakes is the best way to learn - if you don't make mistakes, that means you haven't succeeded. Understanding how certain things work is more important than knowing a certain programming language. Think of programming as the tool that you use to drive, because without knowing how to drive you won't be able to arrive at your goal.



Alongside having a happy family, Nanik is also an accomplished writer



Just in case you are in doubt that Nanik is a Jedi, just read his quote