Albrecht Weinert

Ubuntu for Docker

A b s t r a c t and I n t r o d u c t i o n

This technical report is about installing and using Docker on Linux machines.

Target machines

The concrete first target platform was **Ubuntu 16.04** on a **Fujitsu** server:

- image ubuntu-16.04.1-server-amd64.iso [07.11.2016 699.400.192] as of now with all apt-get updates
- server Fujitsu RX200S5 with two LSI MegaRaid extenders.

At its time the **Fujitsu RX200S5** was a full grown powerful server, and still is. The one of them reported here on did years of good service as file server and in many other roles in an AD domain with Windows Server 2003. Migrating to newer Windows distributions it took only some updates to loose the RAID extenders and no newly installed modern Windows brought them back – but Ubuntu 16.04 (server only, see [29]) did. We completely deleted the machine's Windows past and put pure (non graphical) Ubuntu server on it. This server shall provide a bunch of services in at least two different networks.

A second sort of target machines to use will be Raspberry and mechanical compatibles, alone or as a little farm.

Motivation and goals

As outlined here and in [29] Microsoft's tendency to to break existing applications (24/7 server, too), hardware by loosing driver support and long grown HMI habits motivated our bringing more and more Linux WS and even full grown servers to what was once a pure middle sized Windows (AD 6000 user, 400 WS accounts) site.

Going Ubuntu/Linux is not paradise regained by the way: As some of MS' developments and decisions seem to be motivated by the aim to drive users from Windows to Linux – so do some people in the Linux community seem to want and do all to drive the converts back.

On said server we almost revived all roles it had in its former Windows live. Nevertheless some roles and much comfort is still lacking. Experience showed painfully that beyond some point filling those holes by adding (apt-getting) to the server endangers its integrity.

The worst example respectively experience is the lack of a decent HMI and graphical tools in Ubuntu Server. On the other hand when either using a non server distro or adding graphics to a running server one looses respectively destroys server features, as e.g. hardware RAID.

Some impressive Docker presentations (by Peter Rossbach et al.) let us hope Docker containers may be a way to provide a greater variety and flexibility of services here. It might be a way to break the current confinements of the Ubuntu server OS not modifying it on one end nor resorting to the burden of virtual machines on the other.

On the content

In **Part I** we describe the basics and the installation on the first target machine **Fujitsu RX200S5**.

Part II deals with some use cases and using other smaller targets.

Find most References, Abbreviations, a short paragraph on **Using names** as well as a collection of useful Linux commands in [29], this won't be repeated here.

Some useful docker related commands, Docker references and this report's **Table of Contents** find in the **Appendix**.

P A R T I

Docker basics

Docker introduces containers to

- encapsulate applications, libraries or even kernels,
- giving them a restricted set of virtual or mapped real resources,
- including even process, user and group IDs but
- isolate it from the (non mapped) rest of the system from the container.

The host and host operating system is available and used, making those containers a quite thin and lightweight and fast layer.

On top of this Docker CLI and containers comes a bunch of tools for creating, deleting, updating etc. many and co-oparating containers, managing their (virtual) infrastructure even over the boundary of one single host machine. Without all these extras, i.e. with the Docker CLI alone in one machine one can create, organise run and utilise containers. The cry for more and complicated orchestrating tools will come only after success or even some enthusiasm. Hence, in the state of just starting with and trying to understand Docker we don't care about the higher level tools yet.

The real base for this thin but effective isolation layer is a Linux feature called “namespaces”. In Windows nothing comparable/compatible exists, effectively binding Docker containers to Linux hosts and, as containers are a thin layer over the host OS, restricting contained applications to Linux ones.

As of Windows Server 2016 the namespace and containment features, which, of course, Windows NT and descendants always had, as well as WSL where tuned to be more Docker compatible. So, on the surface you may get something very (not distinguishably) alike. Under the surface it is totally different. Nevertheless this and especially Windows 2016 micro server is quite interesting for the Docker user. On the other hand, as our use of Linux and of Ubuntu servers ([29]) is mostly motivated by getting some Microsoft independence we won't care about Docker on Windows, yet, and stay in Docker's original habitat.

Linux namespaces

Namespaces exist since 2002 in the 2.4 kernel and became really useful since the 3.8 kernel. The feature can be used by itself e.g. for “jailing” an application, but seldom are. Their main merit is being the base for containers.

Namespaces wrap/isolate a (part of a) global resource in a way that a process within the namespace sees its own isolated instance of the global resource – maybe mapped to other names and numbers.

At present (kernel >= 4.10) there are 7 kinds of namespaces:

1 ● mnt mount
2 ● pid process ID
3 ● net network
4 ● ipc interpocess communication
5 ● utc hostname and domain
6 ● user user ID
7 ● cgroup control group

More namespace kinds, for time and system logging, are proposed and may come in future.

The term “namespace” may mean the feature, the namespace kind (1..7) and the concrete namespace (set) a process is in. What is meant should be clear from the context.

On boot Linux starts on a global namespace, with all seven kinds. This global namespace set has all the machines resources of the respective kind in it. And Linux from boot will put all its processes

in this global set. The mechanism is implemented by the way Linux handles its process descriptors and by the way processes find resources.

Obviously, namespaces are nothing superimposed to Linux, they are one of the OS' fundamentals.
Note: An example for unfittingly superimposing are ACLs which never made in in the mainstream to superimpose the initial meaning of the infamous nine bits keeping Linux well below Windows standard.
On the other hand, with not too old Linuxes you just have namespaces mostly without noticing.

Each process is subdirectory of /proc/ named by its ID number:

```
dir /proc/
```

gives some 1000 processes most of them root.root (excerpt)

```
dr-xr-xr-x  9 root             root             0 2017-04-21 15:54  1
dr-xr-xr-x  9 root             root             0 2017-04-21 15:57  100
dr-xr-xr-x  9 root             root             0 2017-04-21 15:57  1035
dr-xr-xr-x  9 root             root             0 2017-04-21 15:57  1036
dr-xr-xr-x  9 root             root             0 2017-04-21 16:01  10694
dr-xr-xr-x  9 root             root             0 2017-04-21 16:01  10696
dr-xr-xr-x  9 root             root             0 2017-04-21 15:57  11
dr-xr-xr-x  9 root             root             0 2017-04-26 12:37  14416
dr-xr-xr-x  9 weinert          weinert          0 2017-04-26 12:37  14426
dr-xr-xr-x  9 systemd-timesync systemd-timesync 0 2017-04-21 15:57  1884
dr-xr-xr-x  9 root             root             0 2017-04-21 15:57  1897
dr-xr-xr-x  9 daemon           daemon           0 2017-04-21 15:57  1911
dr-xr-xr-x  9 syslog           syslog           0 2017-04-21 15:57  1915
dr-xr-xr-x  9 messagebus       messagebus       0 2017-04-21 15:57  1941
dr-xr-xr-x  9 root             root             0 2017-04-24 11:09  23046
dr-xr-xr-x  9             1984 root             0 2017-04-24 11:39  23189
dr-xr-xr-x  9 openldap         openldap         0 2017-04-21 15:58  2426
dr-xr-xr-x  9 Debian-exim      Debian-exim      0 2017-04-21 16:01  2460
dr-xr-xr-x  9 root             root             0 2017-04-21 16:01  2471
dr-xr-xr-x  9 root             root             0 2017-04-21 16:01  2601
dr-xr-xr-x  9 libvirt-dnsmasq  dip              0 2017-04-21 15:58  2621
dr-xr-xr-x  9 weinert          weinert          0 2017-04-21 16:01  2746
dr-xr-xr-x  9 weinert          weinert          0 2017-04-21 16:00  2747
dr-xr-xr-x  9 www-data         www-data         0 2017-04-21 16:07  32455
dr-xr-xr-x  9 www-data         www-data         0 2017-04-21 16:07  32458
```

Now looking at one of them (say 32458) reveals (excerpt):

```
sudo dir /proc/32458/
```

```
lrwxrwxrwx 1 root     root     0 2017-04-21 16:07 exe -> /usr/sbin/apache2
dr-x------ 2 root     root     0 2017-04-26 12:51 fd
-rw-r--r-- 1 root     root     0 2017-04-26 12:51 loginuid
dr-x------ 2 root     root     0 2017-04-26 12:51 map_files
-r--r--r-- 1 root     root     0 2017-04-26 12:51 maps
-rw------- 1 root     root     0 2017-04-26 12:51 mem
-r--r--r-- 1 root     root     0 2017-04-26 12:51 mountinfo
-r--r--r-- 1 root     root     0 2017-04-26 12:51 mounts
dr-xr-xr-x 5 www-data www-data 0 2017-04-26 12:51 net
dr-x--x--x 2 root     root     0 2017-04-26 12:51 ns
```

Of interest is the subdirectory ns: `sudo dir /proc/32458/ns/`

```
lrwxrwxrwx 1 root root 0 2017-04-26 12:57  cgroup -> cgroup:[4026531835]
lrwxrwxrwx 1 root root 0 2017-04-26 12:57  ipc    -> ipc:[4026531839]
lrwxrwxrwx 1 root root 0 2017-04-26 12:57  mnt    -> mnt:[4026531840]
lrwxrwxrwx 1 root root 0 2017-04-26 12:57  net    -> net:[4026531957]
lrwxrwxrwx 1 root root 0 2017-04-26 12:57  pid    -> pid:[4026531836]
lrwxrwxrwx 1 root root 0 2017-04-26 12:57  user    -> user:[4026531837]
lrwxrwxrwx 1 root root 0 2017-04-26 12:57  uts     -> uts:[4026531838]
```

reveals all 7 namespace kinds under the short name given above. Each is a link pointing to the “real” namespace used. In the exemplary case or “32458” all are the global ones. Would we have other namespaces defined some or all of it could point to a restricted one.

Such a namespace can die. And by some tools will be killed when no one points/links to it. That's the pattern of Java garbage collection. Example:

Creating, listing and deleting a network namespace

```
sudo ip netns add myns01
ip netns list
dir /var/run/netns/
sudo ip netns delete myns01
ip netns list
```

The first command creates a new net namespace named “myns01”, which will be created as new (empty) sub-directory of /var/run/netns/. The second and third command proof its existence. The forth command kills it. The killing is verified by repeating the second command. The killing would have failed had someone (a process) used this namespace by linking to it.

We could have used this (empty) namespace to confine an application from networking or – in other words – put it in a no-network jail. And we will do so now.

Creating and using a namespace to jail an application

ifconfig -a

will show 8 network devices in our PD321S server including the loopback (lo) device and the “real” ones ens1f1, ens1f1 etc. Now the experiment (all putty remote at the headless server):

```
weinert@pd321s:~$ sudo ip netns add myns01
weinert@pd321s:~$ sudo ip netns exec myns01 bash
root@pd321s:~# ifconfig -a
lo        Link encap:Lokale Schleife
          LOOPBACK  MTU:65536  Metrik:1
          RX-Pakete:0 Fehler:0 Verloren:0 Überläufe:0 Fenster:0
          TX-Pakete:0 Fehler:0 Verloren:0 Überläufe:0 Träger:0
          RX-Bytes:0 (0.0 B)  TX-Bytes:0 (0.0 B)
root@pd321s:~# exit

weinert@pd321s:~$ ifconfig -a
          ….   all network devices are here again    …..............
```

Here we create the empty network space again and use it to run another bash. In this respect we were jailed. All other resources including access to the full file system and all else we still had. This was to be expected and verified by some simple experiments not documented here.

We had a first look at Linux namespaces and a simple direct use case: a serviceable “network jail”.

`sudo unshare -n  bash` would, by the way, have done the same.

Install Docker, first experiment

The following commands get docker and one simple image installed and running in a container:

```
sudo apt install docker.io
sudo docker images
sudo docker run -i -t ubuntu:14.04 bash
```

The first command installs Docker on our server. That was it!

The second one shows the images we do have locally, which are yet none.

The third run command when issued the very first time on our server gets a headless Ubuntu 14.04 plus almost nothing as image. Then it runs this image as bash instead of the the server one's in our putty session.

```
weinert@pd321s:~$ sudo docker run -i -t ubuntu:14.04 bash
root@58ce1d2a7512:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
root@58ce1d2a7512:/# exit
exit
weinert@pd321s:~$
```

Not bringing us further than having an older headless Ubuntu hiding the actual one we stop using this container by exit, as shown. Now

```
sudo docker images
```

```
REPOSITORY          TAG      IMAGE ID        CREATED         SIZE
ubuntu              14.04    302fa07d8117    11 days ago     188 MB
```

shows that we kept the (small) image in the local Docker repository and can run it again and again. Doing so and experimenting a bit reveals that (by default) we're now completely jailed by the container:

- ifconfig reveals a loopback and a dead eth0.
- A file system is there but – after each new start – empty as freshly installed and with no relation to the host machine's file system.
- Listing all processes reveals the named standard ones but none of the server PD321S' hundreds of numbered ones seen above. And so on.

To sum up and compare to the simple pure namespace experiment above:

In a container we are totally jailed by default.
Using namespaces directly we're totally open except for the properties of the namespace kinds we explicitly make and supply to the process in question.

With Docker images and containers we'll have the full spectrum with more comfort as when having to use namespaces ourselves.

A second experiment, something with graphic HMI

On our headless and GUI-less server we are more or less confined to putty. Is Docker a way to add graphical HMI best via RDP?
The following was just found more or less undocumented (and not understood) on the web. To use it and to show it here was driven by the wish to have something graphical on our headless (stone age HMI and editors; see [29]) on our big Ubuntu 16.04 server. In this respect the following is totally disappointing, but nevertheless a nice piece of voodoo.

Do get something graphical, even usable by browser, we found this:

```
sudo docker pull consol/ubuntu-xfce-vnc
sudo docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	14.04	302fa07d8117	11 days ago	188 MB
consol/centos-xfce-vnc	latest	9b2977baddea	13 days ago	1.173 GB
consol/ubuntu-xfce-vnc	latest	a58904bf5b9b	13 days ago	1.156 GB

```
sudo docker run -it -p 5901:5901 -p 6901:6901 consol/centos-xfce-vnc bash
```

This command brings us some strange messages and a strange bash shell behaving quite differently from the one we left. Anyway, exit brings us back.

```
sudo docker run -d -p 5901:5901 -p 6901:6901 --restart=always consol/ubuntu-xfce-vnc
```

```
wget http://ipinfo.io/ip -qO -   ; useless
```

The recipe's first command gets us a web service running for ever by the first command.

As said in the recipe the last command shall give us this services IP address, which it did not in our case. Here by using an external service – en lieu de just ifconfig ! – the cook tries to be too clever. This fails a) in an isolated environment with no internet and b) (our case) behind proxies and/or gateways the external service will render one their IP addresses.

Instead of a potentially misleading recipe a documentation saying “Use an IP which your server is visible by from the client” would have been better. In our case they are are just the server one's in its different networks: those of ens1f0 (192.168.89.6), ens1f1 (193.175.115.6) etc..

From any workstation with a browser the URL (Chrome on Windows7 e.g.)
http://193.175.115.6:6901/vnc_auto.html?password=vncpassword

gives us graphical access to a strange Ubuntu distribution; see Figure 1.

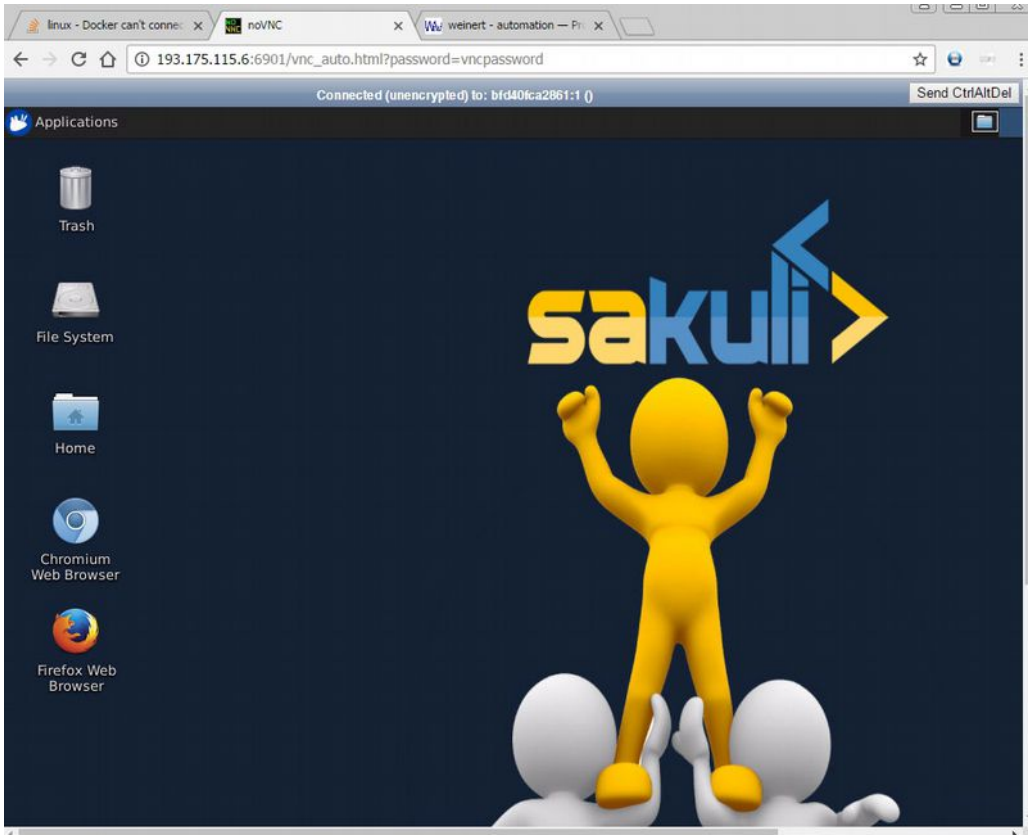

Fig. 1: sakuli

To all uninitiated this is sheer voodoo. Nevertheless the blue upper bunny button labelled “Applications” gives us an ugly shell (without clipboard support).

Here

```
cat /etc/*-release
```

will tell us this is an Ubuntu 16.04 LTS. Well, that the one we use in many places, with GUI and, alas, on servers without. Why does all look so … loony?

What we see is a permanently running singleton instance. Getting there with another browser (also over another LAN and on another machine, if you like)

http://192.168.89.6:6901/vnc_auto.html?password=vncpassword

gets all to the same share state, where the other one sees what you type.

Nice demo, but not what we are looking for:

- A (RDP) way to add a decent HMI with clipboard support and all
- added to our headless server
- not by installing anything (Xserver or the like), lest to spoil a well running server, but
- by a very clever Docker image, adding all that side effect free.

Well, still searching, googling – n o result yet.

Part I's intermediate results

We acquired a very basic understanding of Linux namespaces and of Docker based upon.
With namespaces “basic” means “very basic” as our little experiment just used the less complicated namespace kind in the most simple way. Nevertheless, one can isolate an application from networking – a real use case – this way.

We didn't look at other namespace kinds. Symptoms are, network being the only one that can be utilised by command line alone.

We installed Docker on a “real big” server, the got two docker images and used both.

In theory, we no just have to get useful Docker images made by other, clever people and utilise them.

Getting appetite by the first successful experiments, we wanted an image providing a decent HMI via RDP to our Ubuntu server. For the admin no jailing of any resources would necessary, quite the contrary, for other users one could think of many isolation levels.
But, as of yet, we didn't find such thing – tips welcome.

P A R T II

Part I was on acquiring the basics of namespaces and Docker.

The target was a server machine Fujitsu RX200S5 with hardware RAID (LSI) running Ubuntu 16.04 LTS server (headless).

Part II deals with more and real use cases, other smaller target machines down to Raspberry and maybe higher managing / orchestrating tools.

An Ubuntu for RDP

After some searching we found https://hub.docker.com/r/jumanjiman/xrdp/. By

```
sudo docker build --rm -t jumanjiman/xrdp
sudo docker run -d --name dc-xrdp -p 3389:3389 tomestos/xrdp
```

and by running an .rdp file (C:\util\remoteDT\pd321s_X.rdp) containing among others

```
full address:s:192.168.89.6
username:s:ubuntu
```

+ logging in as ubuntu:ubuntu we see figure 2by RDP.

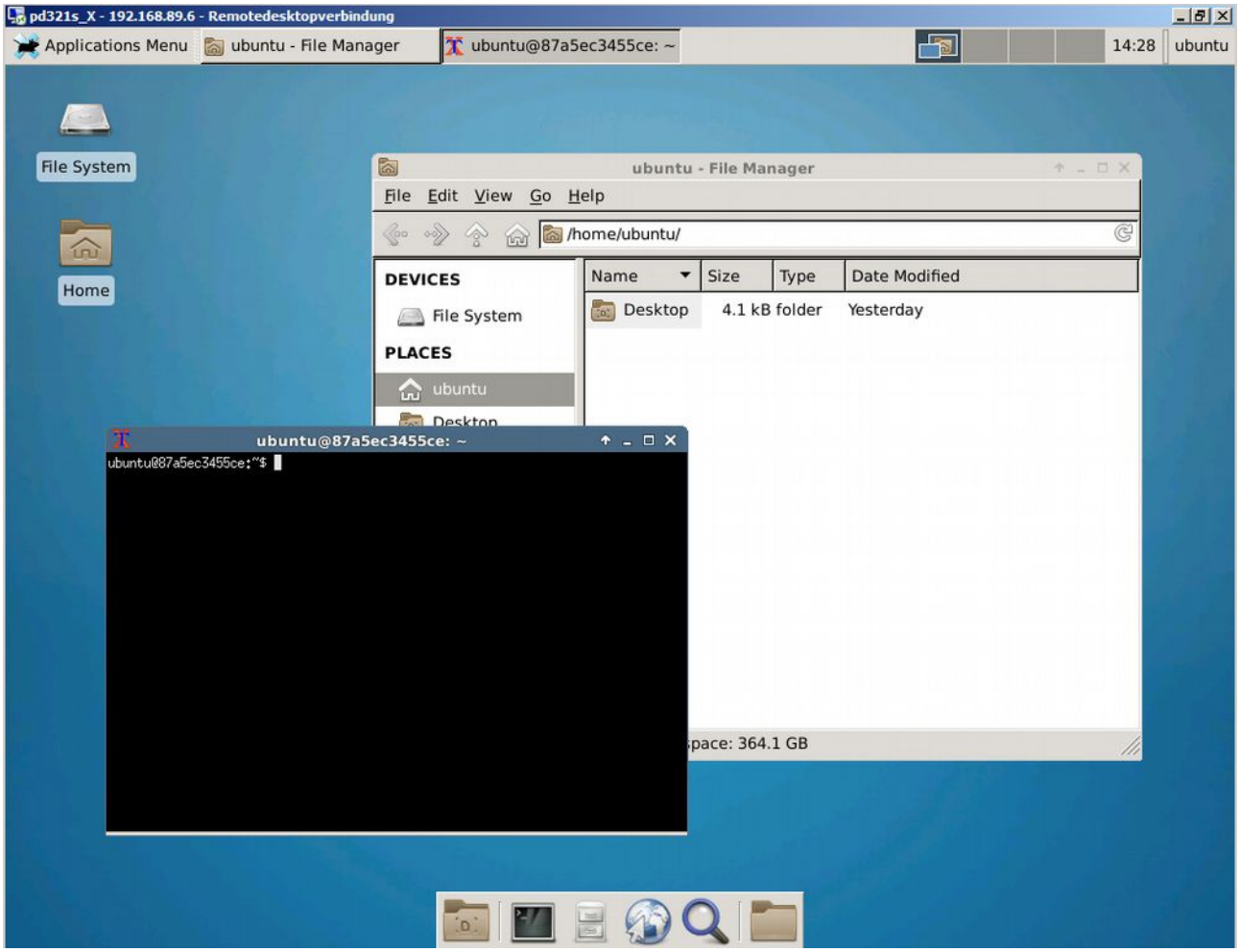

Fig. 2: Xubuntu by Xfce 4.10

It is, of course not, the GUI wrapper around our server's Ubuntu, but we hoped it would shine a light on the way there. What we got is a camouflaged Ubuntu 14.04.5LTS to play with.

The distribution has

- a horrible (non bash, non configurable) “terminal”,
- an as horrible explorer presenting itself as Thunar and
- no clipboard support at all neither within the system nor between it and the RDP host system, Windows 7 professional in our case.

The later point might be a basic bug of the Linux RDP server (xfce) as RDPing from an Ubuntu machine makes it no better.

Having full clipboard support with RDP is indispensable for any remote job and no Windows administrator would ever imaging such thing without. No or fragile clipboard support is a big flaw.

The good side of this image / container is its

- being stateful (at least for one singleton instance)

Logging out and in again brings you back to right where you left.

Wouldn't it be for the clipboard issue this would be a nice playground to get acquainted with less advanced Linux distros and tools.

A second example for Docker usage found at https://docs.docker.com/compose/wordpress/ is

A WordPress confined

A more useful images / containers is a WordPress to play with. On the server (with putty) do:

```
mkdir myWordpress
myWordpress
touch docker-compose.yml
```

Then (best via FileZilla and a GUI editor) open docker-compose.yml and copy the recipe in:

```
version: '2'
services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress
   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

In this approach we can describe multiple co-operating containers, in this docker.com example one for the database and one for the WordPress standard webserver. To make it happen we, additionally, need:

```
sudo apt install docker-compose
```

Afterwards:

```
sudo docker-compose up -d
```

gets and makes all. Going to PD321S:8000 brings us to our new WordPress server (in a box, sorry in two containers). You start with so so called “5 minutes installation” and play around, change desings, users, user rights, no integration to existing user base/ Id management (might be an insurmountable WordPress flaw) and else.

It is impressive and recommendable for practising rusted WordPress skills or trying out something new or potentially stupid. But this useful example is not for production as having no support for https, no integratibility with existing web servers &c.

Docker on small targets – Raspberry to start with

Someone with Docker on Raspberry experience recommended special Docker fit Raspberry images. This would avoid problems (yet unclear which?) one otherwise would probably run into. Well why not follow a good advice?

First experience – beware of pirates

Our first trial was HypriotOS. As we must burn a Raspberry image on a µSD we put our Ubuntu WS/laptop to work on the first steps.

Use a browser or wget on
https://github.com/hypriot/image-builder-rpi/releases/download/v1.4.0/hypriotos-rpi-v1.4.0.img.zip
to put
-rw-rw-r-- 1 weinert weinert 240254583 2017-05-04 13:53 `hypriotos-rpi-v1.4.0.img.zip`

to ~/Downloads. Use jar or the explorer to unzip it to
-rw-r--r-- 1 weinert weinert 1048576000 2017-03-19 16:35 `hypriotos-rpi-v1.4.0.img`

Use lsblk to see the µSD inserted + their partitions (auto) mounted. Use the names seen to un-mount any partitions and burn the image by something like:

```
sudo umount /dev/mmcblk0p1
cd ~/Downloads
sudo dd of=/dev/mmcblk0  if=hypriotos-rpi-v1.4.0.img bs=4M
```

The last command may take quite a while.

Put the µSD just burned to a Raspberry, connect mouse, keyboard, LAN and monitor and, not least but last, power.

It works and boots .. but we can't log in neither by pi:raspberry nor (starting to anticipate evil) pi:raspberrz. In one blog we learned – after a while of searching – that against all habits and against the documentation it would be pirate:hypriot which worked as pirate:hzpriot in the end.

Scotty, beam me up!

All toils to set the thing to German keyboard failed by un-availability of the standard tools or by … the wrong keyboard. Trying to enter complicated commands or stone age editors with wrong keyboard handling by the OS is no good for the blood pressure.

What a b..llsh.t: A distro made to avoid Docker problems, one would otherwise have, brings other hard ones, so it can't be used at all from the beginning – before even getting near to Docker at all.

Two notes to avoid discussion:

Note 1: I apologise for the a b..llsh.t. Imagine a less drastic way to express my judgement.

Note 2: Yes I did hear the advice “Gét än American kezbøard¡” when getting near a special type of Docker wizzard – very nice people, most of them – often enough.
No, I never won't (as long as live and work in Europe, Asia, Africa ... and not in the US)!
Here, this advice shows a lack of professionalism of the advisor or, at least, his willingness to tolerate other's bad work by an in-acceptable surrender. This attitude renounces any claim to be part of something being an alternative to MS systems.

Of course, `ifconfig -a` (after finding – in the wild or having a numeric one) revealed the IP address our fresh Raspberry 3 got from our DHCP. From putty.exe, one could verify without keyboard struggling that raspy-config wasn't there and couldn't be got.
A file /etc/default/keyboard does not exist.

```
sudo dpkg-reconfigure locales
```

worked and allowed to make de_DE.ISO-8859-1 and de_DE.UTF-8, but, as expected, had no effect on keyboard settings. And dozens of other tips did fail, as did making a suitable /etc/default/keyboard and

```
sudo invoke-rc.d keyboard-setup start
```

and/or reboot.

Hence, to sum up:
Only when always running headless and GUI-less (i.e. remote-puttied, only) this pirate's OS may work. Otherwise forget this distribution.

Testing Raspbian lite

We got the impression HypriotOS was based on Raspbian lite. And before making above criticisms more public if all is not lite's fault. We never used lite as the only motivation for it seemed using extremely small µSDs.

So we gave it a try:

- download the image and unzip
- put a small (8GB) µSd in the slot and check the mount
- burn the image and put in in a Rasberry 3
- get the machine to mouse, keyboard, LAN, monitor and at last power.

Well the keyboard is wrong at start, so use pi:rasberrz at first log in.
But raspi-config is there and operational. Hence use it to

- set your locale
- the keyboard layout
- the time zone
- your country's WiFi settings
- enable using all the rest of the µSD as disk
- and enable ssl
  Do not forget. ssl is there but disabled. raspi-config was the only way found to enable it over reboots. (The usual alternatives didn't work.)

To round up give it an update upgrade (takes time) and try if all survives re-boots. The re-boots are quite fast; that may be a better motivation for lite.). As we never must burn a Raspberry image on a µSD we put our Ubuntu WS/laptop to work on the first steps.

From the right ssl settings in raspi-config on, you may use putty. Best, start with giving the new baby a nice setting in the registry, replacing the outright ugly dark on black. Having this setting stored as lite42, e.g., you may start your remote session by

```
putty.exe -load lite42 -l pi
```

best making a nice icon bearing that command. As said on Windows this would be the registry ([HKEY_CURRENT_USER\Software\SimonTatham\PuTTY\Sessions\lite42] in our example); on an Ubuntu WS/laptop it' s a text file, but command and icon making is essentially the same.

Now we have a raspberry 3 with a lightweight “lite” GUI-less OS, hoping it be a base for some “Docker on small machine” experiments or real applications.

Docker comes to Raspberry

From [40] we learn that after the pirates heroic preparational work an Docker now officially supports Raspberry respectively ARM.

The link get.docker.com (from [40] gives an sophisticated installation script that's worth reading. On our well prepared “Raspberry lite” (see above) we let it go by:

```
curl -sSL get.docker.com | sh
sudo usermod -aG docker pi
```

As expected from reading the first command takes a while and brings lots of messages. Use the second one when tired of decorating every docker command by sudo. Beware of some security riscs when putting the user used I the docker group. The membership would change after reboot.

Commands like

```
sudo docker images
sudo docker ps
sudo docker ps -a
```

with or without sudo show we do have Docker with nothing else yet. Just as “proof of installation we may get and run an Alpine Linux (whatever that is) image as proposed by [41]:

```
docker run -ti armhf/alpine:3.5 /bin/sh
docker run armhf/alpine:3.5 date
```

Notwithstanding the “Unable to find image 'armhf/alpine:3.5' locally” the installation and start is surprisingly fast. (I've to confess, it took a while realising “/ # “ being no stuck download but the Alpine prompt.) Give Alpine something to do and exit back after the first command. Figure 3 shows the result.

```
pi@raspberryLite:~ $ docker run -ti armhf/alpine:3.5 /bin/sh
Unable to find image 'armhf/alpine:3.5' locally
3.5: Pulling from armhf/alpine
6a3203a774aa: Pull complete
Digest: sha256:cde15a7e720fee4ef997b028fd6a32d680370f41950e27c00db5984ee709b41b
Status: Downloaded newer image for armhf/alpine:3.5
/ # ls
bin    dev    etc    home   lib    media  mnt    proc   root   run    sbin   srv
/ # ls bin
ash            chown          dnsdomainname  fgrep          iostat         makemi
base64         conspy         dumpkmap       fsync          ipcalc         mkdir
bbconfig       cp             echo           getopt         kbd_mode       mknod
busybox        cpio           ed             grep           kill           mktemp
cat            date           egrep          gunzip         ln             more
catv           dd             false          gzip           login          mount
chgrp          df             fatattr        hostname       ls             mountp
chmod          dmesg          fdflush        ionice         lzop           mpstat
/ # ls home
/ # exit
pi@raspberryLite:~ $ docker run armhf/alpine:3.5 date
Fri May  5 15:20:11 UTC 2017
pi@raspberryLite:~ $
```

Fig. 3: Alpine Linux on “Raspberry lite”

Repeating the list command above tell us we have one image and two terminated processes with our Docker.

More useful examples, own developments

To be done:

- Apache 2

Having multiple Apaches in Docker images and cloning the productive one to play with should be a common use case. To be done and tried in a “real” local server.
In case of / after success to be tried with rented “virtual” servers,.

- Ubuntu Subversion

Having Apache 2.4 Ubuntu 16.04 ready and running, we have SVN running on top of it in many cases. Light weight virtualising / cloning this too, as, hopefully, with Apache2 alone would open an experimental field do develop hooks, GIT-integration and much more.

Making Docker containers behave

Over the time we got many recipes for mending Linux/Ubuntu nuisances (some of them found in [29] and else). Those repairs deal with

- Having other JDK than the Oracle one
- Resurrecting the other one on updates or installs
- Not having the standard (Microsoft) fonts for exchange with partners and colleagues (including one self)
- Making dark on light work instead of the ugly black shells Having Apache 2.4 Ubuntu 16.04 ready and running, we have SVN running on top of it in many cases.
- and much more

Depending on the image and level of confinement the repairs on the basic OS stay in effect. But often they won't. In some cases the same repairs just repeated will work.

In all other cases we'll need new recipes.

The result – and where we are with Docker

Our work with docker – if we should continue this path – is just at the beginning and this report is in no way complete. As Docker is both promising and widely used, I'm almost sure we will go on.

In Docker events and on the web you'll find many impressive Docker applications. But most of them are fixed examples, and some sheer hacking or, as sometimes said above. voodoo. This seems a bit unfair. Having seen Docker workshops with some 40 participants all on their Linux laptops, following the presented recipe (by one seeming to have guru status), 10 failing, no chance to mend, almost no one caring … it may be worse.

In an environment, to run 24/7, where money flows from customers partly to service providers, on real time or process control application, where safety or security comes in, and the like: this working by example, getting some hardly documented syntax and semantic of configuration files is not the base to work from.

And getting a base quality even near to it is harder than promoted. The loveable freedom and playfulness of this open source community brings a lot of items in the repositories that may have been good in just very narrow case and very few environments. At least there seems not much QA of any sort.

One idea behind Unix was avoiding monolithic tools and doing one thing, only, and doing it well. The downside is one may have to organise the co-operation of the small tools in complicated pipes or scripts to achieve what others do with one programme. But if the latter won't suit, the flexibility pays. The same approach we see with micro-services and containers. But, alas, by just searching, downloading, using mostly we found small things done badly.

Hence, the charming idea of “pull and use” is dead, at least for productive use. When using Docker professionally one must learn the basics, the tools, the idiosyncrasies of the configuration files and so on and more often than not build one's own distros and images. Well, business as usual.

To be fair: Just make that or take this and enjoy/play with it is a good thing by itself.
And, perhaps not made obvious here, we enjoyed it, too.

A p p e n d i x

Miscellaneous commands

This is more or less an anthology of useful and proven tips.
Some common Linux commands please find at the same place (Appendix) in [29].
Here we have just namespace and Docker related command.

Namespaces

Create, list, delete network namespaces:

```
sudo ip netns add  nameSpaceName
ip netns list
sudo ip netns delete  nameSpaceName
```

Use a network namespace with an application:

```
sudo ip netns exec nameSpaceName  application
```

Common checks and maintenance

See all network interfaces:

```
ifconfig -s -a
ip link
```

See all Docker images in local repository:

```
sudo docker images
```

See all Docker containers running locally:

```
sudo docker ps
```

See all Docker containers existing locally:

```
sudo docker ps -a
```

Stop and remove a Docker container:

```
sudo docker stop  name
sudo docker remove  name
```

Find all Raspberries in the private net (192.168.89.*)

```
sudo apt install nmap
sudo nmap  -sP  192.168.89.0/24 | awk '/^Nmap/{ip=$NF}/B8:27:EB/
{print ip}'
```

Abbreviations

Abbreviation found in the Appendix of [29] will not be repeated here. Search there, please.

ACL Access control list

CLI Command line interpreter/interface

IP Internet protocol

QA Quality assurance

WSL Windows Subsystem for Linux

Table of Content

References

[1..28] see [29]

[29] Albrecht Weinert, Ubuntu for remote services, Report, November 2016,
a-weinert.de/pub/ubuntu4remoteServices.pdf

[30] Albrecht Weinert, Ubuntu for docker, Report, April 2017,
This paper (the last actual version): a-weinert.de/pub/ubuntu4docker.pdf

[40] Matt Richardson, Docker comes to Raspberry Pi
blog, August 2016 https://www.raspberrypi.org/blog/docker-comes-to-raspberry-pi/

[41] Alex Ellis, Get Started with Docker on Raspberry Pi
blog, August 2016 http://blog.alexellis.io/getting-started-with-docker-on-raspberry-pi/

Dr. Albrecht Weinert is computer science professor at
Bochum University of Applied Sciences or Hochschule Bochum.
He is founder and director of
MEVA-Lab – Laboratory for versatile distributed applications –
as well as of the service provider weinert - automation.
albrecht@a-weinert.de

Rev. 08- 08.05.2017