Linux Containers and the Future Cloud

Rami Rosen
ramirose@gmail.com

About me

- Bio: Rami Rosen, a Linux kernel expert, author of a recently published book, “Linux Kernel Networking - Implementation and Theory”, 648 pages, Apress, 2014.

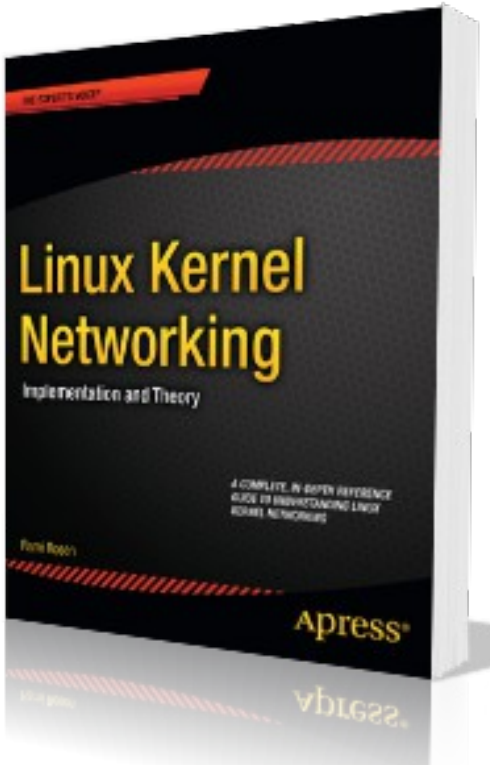


Table of Contents

General

- Lightweight process virtualization is not new.
  - Solaris Zones.
  - BSD jails.
  - AIX WPARs (Workload Partitions).
  - Linux-based containers projects.
- Why now ?
  - Primarily because kernel support is now available (namespaces and cgroups).
    - Kernel 3.8 (released in February 2013).

Scope and Agenda

- This lecture is a sequel to a lecture given in May 2013 in Haifux:

  “Resource Management in Linux”, http://www.haifux.org/lectures/299/

Agenda:

- Namespaces and cgroups – the Linux container building blocks.
- Linux-based containers (focusing on the LXC Open Source project, implementation and some hands-on examples).
- Docker – an engine for creating and deploying containers.
- CRIU (Checkpoint/restore in userspace)

Scope:

  - We will not deal in depth with security aspects of containers.
  - We will not deal with containers in Android.

Namespaces

There are currently 6 namespaces:

- mnt (mount points, filesystems)
- pid (processes)
- net (network stack)
- ipc (System V IPC)
- uts (hostname)
- user (UIDs)
  - Eric Biederman talked about 10 namespaces in OLS 2006.
    - security namespace and security keys namespace will probably not be developed (replaced by user namespaces)
    - time namespace.
    - RFC for device namespaces was recently sent (Android; Cellrox).

6 New CLONE Flags

6 new flags to the *clone()* system call:

CLONE_NEWNS	**2.4.19**	CAP_SYS_ADMIN
CLONE_NEWUTS	**2.6.19**	CAP_SYS_ADMIN
CLONE_NEWIPC	**2.6.19**	CAP_SYS_ADMIN
CLONE_NEWPID	**2.6.24**	CAP_SYS_ADMIN
CLONE_NEWNET	**2.6.29**	CAP_SYS_ADMIN
CLONE_NEWUSER	**3.8**	No capability is required

3 System calls for handling namespaces

- Three system calls are used for namespaces:
- *clone()* - creates a new process and a new namespace; the process is attached to the new namespace.
  - Process creation and process termination methods, *fork()* and *exit()* methods, were patched to handle the new namespace CLONE_NEW* flags.
- *unshare()* - does not create a new process; creates a new namespace and attaches the current process to it.
  - The *unshare()* system call was added in 2005:
  - see “new system call, unshare” : http://lwn.net/Articles/135266/
- *setns(int fd, int nstype)* - a new system call was added, for joining an existing namespace. Available only from kernel 3.0 – see ***man 2 setns.***

- Namespaces do *not* have names in the kernel.
  - With ip netns sub command, we can assign names to network namespaces, but these are kept in userspace and not in the kernel.
- An inode is created when the namespace is created, for each namespace.
- ls -al /proc/<pid>/ns

Namespaces implementation

A member named nsproxy was added to the process descriptor , struct task_struct.

- nsproxy includes 5 inner namespaces:
  - uts_ns, ipc_ns, mnt_ns, pid_ns, net_ns.
- Notice that user ns (user namespace) is missing in this list.
  it is a member of the credentials object (struct cred) which is a member of the process descriptor, task_struct.
- It is better, in terms of performance, than having an object to each namespace, and incrementing the reference counter of each when forking.
- A method named task_nsproxy(struct task_struct *tsk), to access the nsproxy of a specified process. (include/linux/nsproxy.h)

- There is an **initial, default namespace** for each of the 6 namespaces.

Userspace additions

Userspace additions to support namespaces:

- **IPROUTE** package
  - some additions like *ip netns add/ip netns del, ip link set ... netns,* and more (will be discussed in later slides).
- **util-linux** package
  - *unshare* util with support for all the 6 namespaces.
  - *nsenter* – a wrapper around *setns()*.
- **shadow/shadow-utils** (for supporting user namespaces)
  - Still not integrated in all distros.

UTS namespace

- UTS (Unix timesharing) namesapce
  - Very simple to implement.

  A member named uts_ns was added (uts_namespace object) to the nsproxy structure. The uts_ns object includes a **new_utsname** object, which includes these members:

- sysname
- nodename
- release
- version
- machine
- domainname

UTS namespace -contd.

- A method called utsname() was added to fetch the new_utsname object of the uts namespace associated with the current process:

```
static inline struct new_utsname *utsname(void)
{
  return &current->nsproxy->uts_ns->name;
}
```

- The new implementation of gethostname():

```
SYSCALL_DEFINE2(gethostname, char __user *, name, int, len)
{
  struct new_utsname *u;
   ...
   u = utsname();
   if (copy_to_user(name, u->nodename, i))
     errno = -EFAULT;
   ...
}
```

Similar approach was taken in the ***uname()*** and the ***sethostname()*** system calls.

- **Question: What is missing in UTS implementation shown above?**

UTS namespace -contd.

- **Answer:** UTS procfs entries, like:
  - /proc/sys/kernel/hostname
  - /proc/sys/kernel/domainname
- With IPC namespace, the principle is the same; it is simply a bigger namespace.

Network Namespaces

- A network namespace is logically another copy of the network stack, with its own routes, firewall rules, and network interfaces.
- The network namespace is represented by **struct net** (defined in *include/net/net_namespace.h*).

Struct net includes all network stack ingredients, like:

- The loopback device.
- SNMP stats. (netns_mib)
- All network tables:routing, neighboring, etc.
- All sockets
- /proc and /sys entries.

Network Namespaces implementation

A network interface belongs to exactly one network namespace.

- This addition was made to the *net_device* structure:
  - *struct net *nd_net;*

  for the Network namespace this network device is inside.
- A method was added: *dev_net(const struct net_device *dev)*

  to access the *nd_net* namespace associated with the specified network device.

  A socket belongs to exactly one network namespace.
- Added ***sk_net*** to struct *sock* (also a pointer to *struct net*), for the Network namespace this socket is inside.
- Added *sock_net()* and *sock_net_set()* methods (get/set network namespace of a socket)

Network Namespaces – example:

- Create two namespaces, called "myns1" and "myns2":
- ***ip netns add myns1***
- ***ip netns add myns2***
  - ***Implementation: see netns_add() in ipnetns.c (iproute2)***
- Delete “myns1” network namespace is done by:
- ***ip netns del myns1***

  Notice that after deleting a namespace, all its **migratable** network devices are moved to the default network namespace.
- You can monitor addition/removal of network namespaces by:
  - ***ip netns monitor***

- prints one line for each addition/removal event which occurs.

- You list the network namespaces (which were added via “ ip netns add”) by:
- ***ip netns list***
  - this simply reads the namespaces under:

    /var/run/netns
- You can find the pid (or list of pids) in a specified network namespace by:
  - ***ip netns pids namespaceName***
- You can find the network namespace of a specified pid by:
  - ***ip netns identify #pid***

- You can move the *eth0* network interface to *myns1* network namespace by:
  - ***ip link set eth0 netns myns1***
  - *This triggers changing the network namespace of the net_device to “myns1”.*
  - *It is handled by the* ***dev_change_net_namespace()*** method*, net/core/dev.c.*
- **You can start a shell in the new namespace by:**
  - ***ip netns exec myns1 bash***
  - ***Running now ifconfig -a in myns1 will show eth0 and the loopback device.***

- You can move a network device to the default, initial namespace by:

  ip link set eth0 netns 1

- In a network namespace, which was created by “ip netns add”, network applications which look for files under /etc, will first look in /etc/netns/myns1/, and then in /etc.
- For example, if we will add the following entry "192.168.2.111 www.dummy.com"
- in /etc/netns/myns1/hosts, and run:
- ping www.dummy.com
  - we will see that we are pinging 192.168.2.111.

PID namespaces

- Added a member named pid_ns (pid_namespace object) to the nsproxy object.
- Processes in different PID namespaces can have the same process ID.
- When creating the first process in a new namespace, its PID is 1.
- Behavior like the “init” process:
  - When a process dies, all its orphaned children will now have the process with PID 1 as their parent (**child reaping**).
  - Sending **SIGKILL** signal does not kill process 1, regardless of which namespace the command was issued from (initial namespace or other PID namespace).

PID namespaces – contd.

- pid namespaces can be nested, up to 32 nesting levels. (MAX_PID_NS_LEVEL).
- See: multi_pidns.c, Michael Kerrisk, from http://lwn.net/Articles/532745/.

Cgroups (Control Groups)

- **cgroups** (control groups) subsystem is a Resource Management solution providing a generic process-grouping framework.
- This work was started by developers from Google (primarily Paul Menage and Rohit Seth) in **2006** under the name "process containers"; in **2007**, renamed to "Control Groups".
  - Maintainers: Li Zefan (Huawei) and Tejun Heo.
  - The memory controller (memcg), which is probably it is the most complex, is maintained separately (4 maintainers)
  - **Namespaces** provide a per process resource **isolation** solution.
  - **Cgroups**, on the other hand, provides resource **management** solution (handling groups).
- **Systemd** is a replacement for SysV init scripts. It uses aggressive parallelization capabilities in order to start services. It was integrated into Fedora since Fedora 15. It is based on DBUS messages.
  - Fedora systemd uses cgroups.
  - **Ubuntu** does not have systemd yet, but will have it in the future as it was decided that **Debian** will use systemd.
  - RHEL 7 will use systemd. (RHEL 7 is to be be released during 2014; it is based on Fedora 19).
  - Systemd-nspawn: uses namesapces/cgroups to create containers (A tool for testing/debugging of systemd that should run inside a container; no need for config files, easy to use; not meant to compete with LXC or Libvirt LXC or Docker)

Cgroups implementation

- The implementation of cgroups requires a few, simple hooks into the rest of the kernel, none in performance-critical paths:
  - In boot phase (**init/main.c)** to preform various initializations.
  - In process creation and destroy methods, ***fork()*** and ***exit().***
  - A new file system of type "cgroup" (VFS)
  - Process descriptor additions (struct task_struct)
  - Addition of *procfs* entries:
    - For each process: */proc/pid/cgroup.*
    - System-wide: */proc/cgroups*

cgroups and kernel namespaces

Note that the **cgroups** is not dependent upon namespaces; you can build cgroups **without** namespaces kernel support, and vice versa.

There was an attempt in the past to add "ns" subsystem (ns_cgroup, namespace cgroup subsystem); with this, you could mount a namespace subsystem by:

mount -t cgroup -ons.

This code was **removed** in 2011 (by a patch by Daniel Lezcano), in favor of using user namespaces instead.

See:

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/commit/?id=a77aea92010acf54ad785047234418d5d68772e2

cgroups VFS

- Cgroups uses a Virtual File System
  - All entries created in it are not persistent and deleted after reboot.
- All cgroups actions are performed via filesystem actions (create/remove directory, reading/writing to files in it, mounting/mount options).
- Usually, cgroups are mounted on ***/sys/fs/cgroup*** *(for example, by systemd)*
  - *This is not mandatory; mounting cgroups can be done also on every path in the filesystem.*

In order to use a cgroups filesystem (browse it/attach tasks to cgroups, etc) it must be mounted.

The control group can be mounted anywhere on the filesystem. Systemd uses /sys/fs/cgroup.

When mounting, we can specify with mount options (-o) which subsystems we want to use.

There are 11 cgroup subsystems (controllers) (kernel 3.9.0-rc4 , April 2013); **two** can be built as modules. (All subsystems are instances of **cgroup_subsys** struct)

cpuset_subsys - defined in kernel/cpuset.c.

freezer_subsys - defined in kernel/cgroup_freezer.c.

mem_cgroup_subsys - defined in mm/memcontrol.c; Aka memcg - memory control groups.

blkio_subsys - defined in block/blk-cgroup.c.

net_cls_subsys - defined in net/sched/cls_cgroup.c (can be built as a kernel module)

net_prio_subsys - defined in net/core/netprio_cgroup.c (can be built as a kernel module)

devices_subsys - defined in security/device_cgroup.c.

perf_subsys (perf_event) - defined in kernel/events/core.c

hugetlb_subsys - defined in mm/hugetlb_cgroup.c.

cpu_cgroup_subsys - defined in kernel/sched/core.c

cpuacct_subsys - defined in kernel/sched/core.c

devices cgroup controller

- Also referred to as :***devcg*** (devices control group)
- devices cgroup provides enforcing restrictions on opening and mknod operations on device files.
- 3 files: **devices.allow, devices.deny, devices.list.**
  - **devices.allow** can be considered as devices whitelist
  - **devices.deny** can be considered as devices blacklist.
  - **devices.list** available devices.
- Each entry is 4 fields:
  - **Type**: can be a (all), c (char device), or b (block device).
    - All means all types of devices, and all major and minor numbers.
  - **Major number.**
  - **Minor number.**
  - **Access:** composition of 'r' (read), 'w' (write) and 'm' (mknod).

devices - example

/dev/null major number is 1 and minor number is 3 (You can fetch the major/minor number from Documentation/devices.txt)

mkdir /sys/fs/cgroup/devices/0

By default, for a new group, you have full permissions:

cat /sys/fs/cgroup/devices/0/devices.list

a *:* rwm

echo “a *:* rmw” > /sys/fs/cgroup/devices/0/devices.deny

This denies rmw access from all devices.

echo $$ > /sys/fs/cgroup/devices/0/tasks

echo "test" > /dev/null

bash: /dev/null: Operation not permitted

echo “a *.* rwm” > /sys/fs/cgroup/devices/0/devices.allow

This adds the 'a *:* rwm' entry to the whitelist.

echo "test" > /dev/null

Now there is no error.

What will be the future cloud infrastructure?

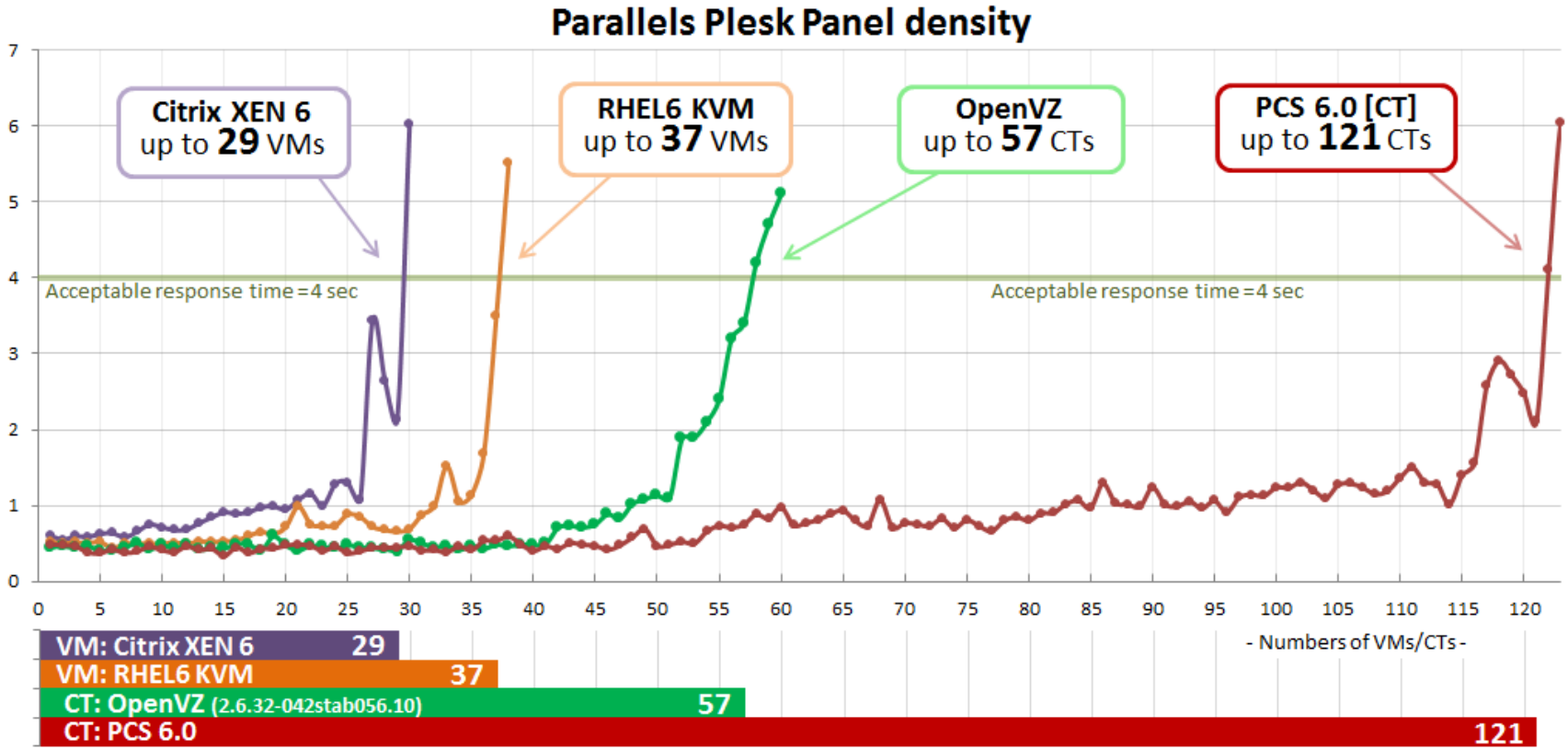
- **Two types of virtualization:**
  - Running VMs: (Xen, Kvm) another OS instance by Hardware Virtualization/Para Virtualization solutions
  - Lightweight Process level (Containers) (Aka Virtual Environment (**VE**) and Virtual Private Server (**VPS**).
- **Will VMs disappear from clouds infrastructure ? Probably not.**
  - With VMs, you can run different kernels on different guests; with containers you must use the same kernel.
  - VM may support non-Linux OSs such as Microsoft Windows, etc. You cannot run Windows in a container.
  - Security.
    - OpenVZ: security audit, performed by Solar Designer in 2005 (http://openvz.org/Security)

- **Containers advantages:**
  - **Nesting** in containers is supported out of the box, though it was not tested thoroughly.
  - **Start-up and Shut-down time**: With process-level virtualization, starting and shutting down a container is faster than with VMs like Xen/KVM.
  - Test: starting 4 Fedora containers as daemons in less then half a second on a laptop.
  - **Density** - you can deploy more containers on a host then VMs (see next in the density slides).
    - Take into account that twice the numbers of containers on a host leads to a higher overall profit.
- "The hypervisor is here to stay":
  - http://mailman.cs.huji.ac.il/pipermail/linux-il/2013-September/010663.html

- LXCBENCH: http://projects.genivi.org/lxcbench
  - Based on Phoronix Test Suite (PTS), see: http://www.phoronix-test-suite.com.
    - GENIVI Alliance organization project.
- You can have more containers on a host then kvm/xen Vms.
- **“Lightweight Virtualization with Linux Containers (LXC)”** - Jérôme Petazzoni
  - The 5th China Cloud Computing Conference, June 2013
  - ● 10-100 virtual machines
  - ● 100-1000 container
  - http://www.ciecloud.org/2013/subject/07-track06-Jerome%20Petazzoni.pdf

- “**Containers and The Cloud: Do you need another virtual environment**”, a lecture given in Enterprise End User Summit, May 2013, by James Bottomley, CTO, Server Virtualization for Parallels.
- Thanks fto James Bottomley from Parallels for giving an explicit permission to show the following chart from his lecture slides:
- http://events.linuxfoundation.org/sites/events/files/eeus13_bottomley.pdf
- The benchmarks were performed on a Xeon server.

Chart of Densities



Containerization is the new virtualization

Containers are in use by many PaaS (Platform as a Service) companies; to mention a few -

- **dotCloud** (which changed later its name to docker): https://www.dotcloud.com/
- **Parallels** - http://www.parallels.com
- **Heroku** - https://www.heroku.com/
- **Pantheon** - https://www.getpantheon.com/
- **OpenShift** of Red Hat: https://www.openshift.com/
- more.

What is a container?

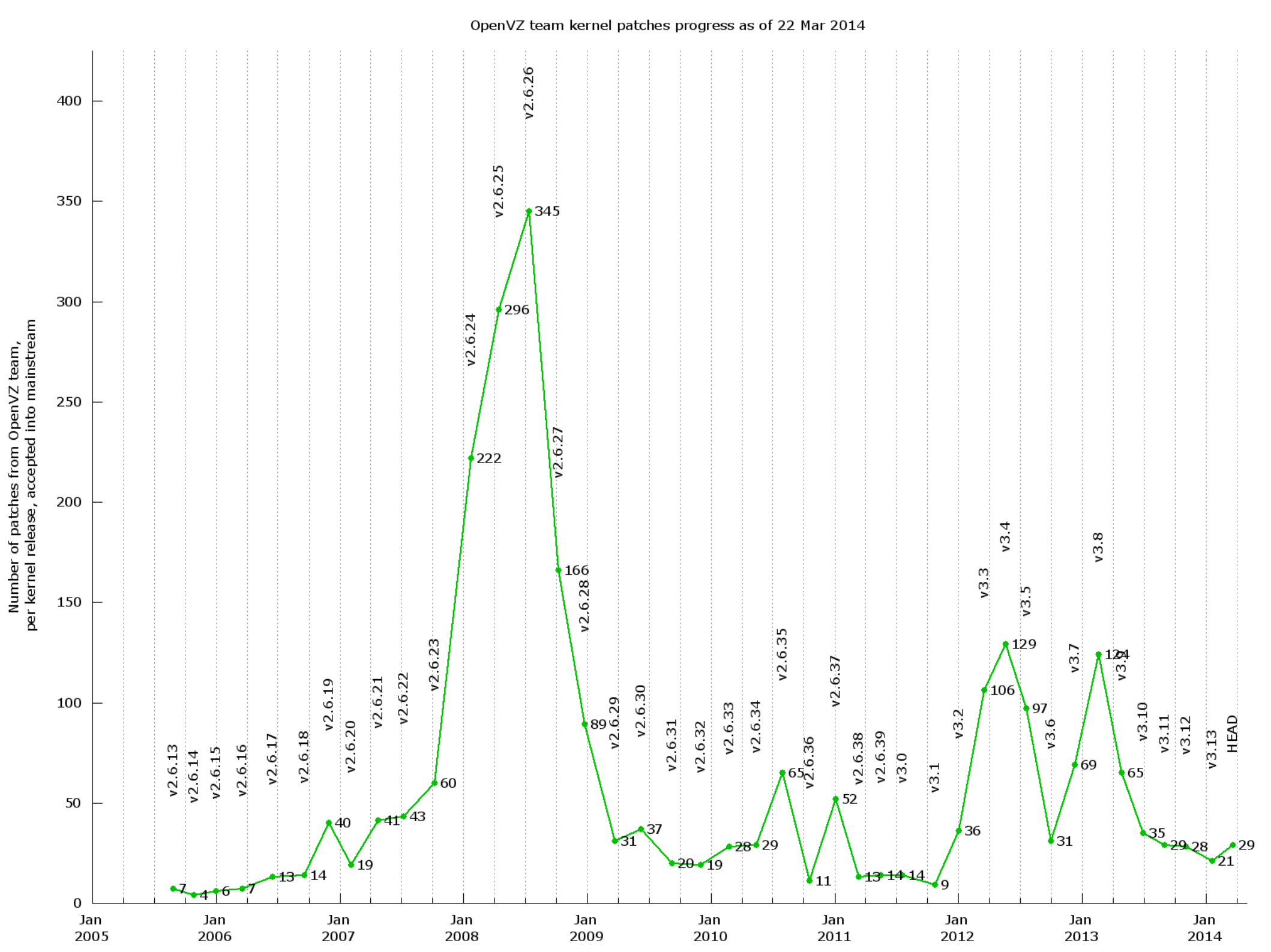
- There is no spec specifying what a container should implement.
- There are several Linux-based Containers projects:
- The **Google containers** (still in Beta phase):
  - https://github.com/google/lmctfy
  - Based on cgroups.
  - It is being considered to add namespaces usage in the future.
- **Vserver**:http://linux-vserver.org/
  - There are no plans to upstream it.
  - Some of the interfaces were defined more than ten years ago.

The OpenVZ project

- **OpenVZ** (based on modified kernel):
- Origins: a company called SWsoft, which was Founded in 1997
  - 2001 - Virtuozzo was released (a proprietary virtualization product)
  - 2005 – OpenVZ (Open Virtuozzo) released (Linux only)
  - 2008 – SWsoft merged into parallels.
  - OpenVZ uses cgroups (originally it used a different resource management mechanism called **bean counters)**.
    - Using **ulimt** is not enough.
  - OpenVZ is sponsored by a hosting and cloud services company named Parallels (http://www.parallels.com)
  - vzctl user space tool.
  - In production
  - Features which are not in mainline yet like **vSwap** or **ploop** (it is not upstream)

The OpenVZ project

- OpenVZ (with the modified kernel) uses 5 namespaces out of the 6. User namespace is not yet used in it.
- Cgroups are not used yet.
  - There is an effort to move to cgroups in the upcoming OpenVZ kernel (which will be RHEL7-based).
- The following plot was contributed by Kir Kolyshkin, and it was plotted using a script from OpenVZ wiki, using *gnuplot*: https://wiki.openvz.org/File:Kernel_patches_stats.png
- For comparison, for kernel **3.14** there are **1233** patches from **Intel** (http://lwn.net/Articles/589728/).


OpenVZ team kernel patches progress as of 22 Mar 2014
Number of patches from OpenVZ team,
per kernel release, accepted into mainstream
0
50
100
150
200
250
300
350
400
Jan 2005
Jan 2006
Jan 2007
Jan 2008
Jan 2009
Jan 2010
Jan 2011
Jan 2012
Jan 2013
Jan 2014
v2.6.13
v2.6.14
v2.6.15
v2.6.16
v2.6.17
v2.6.18
v2.6.19
v2.6.20
v2.6.21
v2.6.22
v2.6.23
v2.6.24
v2.6.25
v2.6.26
v2.6.27
v2.6.28
v2.6.29
v2.6.30
v2.6.31
v2.6.32
v2.6.33
v2.6.34
v2.6.35
v2.6.36
v2.6.37
v2.6.38
v2.6.39
v3.0
v3.1
v3.2
v3.3
v3.4
v3.5
v3.6
v3.7
v3.8
v3.10
v3.11
v3.12
v3.13
HEAD
7
4
6
7
13
14
40
19
41
43
60
222
296
345
166
89
31
37
20
19
28
29
65
11
52
13
14
14
9
36
106
129
97
31
69
124
65
35
29
28
21
29

LXC containers

- **LXC containers:**
- Website: http://linuxcontainers.org/
- Background – a french company called Meiosys which developed an HPC product called “MetaCluster” with checkpoint/restore bought by IBM in 2005. Originally based on a kernel patch which was not merged, so rewritten from scratch in LTC.
- Their CKRM solution (resource management) was rejected.
- An Open Source project; over 60 contributors. Maintainers: Serge Hallyn(Ubuntu), Stéphane Graber(Ubuntu)
- Only on Linux, but can run on many platforms.
- Why now ?
  - User namespaces were integrated into mainline since 3.8 kernel (February 2013).
  - CRIU 1.0 version was released only recently.
- In the past, several projects refrained from using LXC since it was immature (for example, NOVA of OpenStack:
  - “Using OpenVZ to build a PaaS with Openstack's Nova”, Daniel Salinas, Rackspace, LPC 2013. This might change now.
- **Lxc-libvirt**
- not a single line of code in common between LXC and libvirt lxc.
  - virsh – command line tool

  Maintained by Daniel P. Berrange, RedHat.

LXC containers – contd.

- **There is no data structure in the kernel representing a container. A container is a userspace construct (*lxc_container struct*)**
- **There are several patches in the kernel to support checkpoint/restore of processes in general (though these processes can be also containers).**

  A container is a Linux userspace process.
- Created by the ***clone()*** system call.
- When a new container is created, it is **always** with these two namespaces:
  - PID namespace (CLONE_NEWPID flag is set).
  - MNT namespace (CLONE_NEWNS flag is set).

  If lxc.network.type = none in the container config file, then CLONE_NEWNET is not set in the flags which are passed to ***clone()***.
- In such a case, we will have the same network namespace as the host, and see the same network interfaces as the host that created it.

Templates

There are **templates** for creating a container (under ***/usr/share/lxc/templates*** in Fedora). These templates are **shell scripts**.

- In LXC 1.0 there are 14 templates, and in LXC 0.9, there are 11 templates.
- The simplest container is lxc-busybox.
- Fedora templates - Michael H. Warfield, Stéphane Graber, Serge Hallyn, others.
- Oracle template - Dwight Engen, others.

Templates glitches and quirks

- With **busybox** containers, you must use the -k (send SIGKILL) in order to terminate the container, or use busybox from git, where a patch to enable getting **SIGPWR** was applied.
- With Fedora, lxc-stop takes a lot of time. Adding -k will make it quicker (but terminate ungracefully).
- In the past, creating Fedora containers prior to Fedora 15 (with the -r flag) had some issues.
- The name of the bridge interfaces are different for fedora (virbr0) and Ubuntu (lxcbr0).
- Setting lxc.stopsignal to SIGKILL (or some other signal) in a Fedora container config file does not work.

Creating and Destroying a Container

- *Examples for creating a container:*

lxc-create -t fedora -n myFedoraCT

=> *The container will be created under **/var/lib/lxc/myFedoraCT***

(This is true when using a distro package like with yum install lxc or apt-get install lxc. When building from source, when using –prefix=/usr, the container will also be created in that path).

You can also create containers of older Fedora distros, with the -r flag. For example, create fedora 18 distro with:

lxc-create -t fedora -n fedora18 -- -R 18

Creating and Destroying a Container – contd.

Or creating Ubuntu release:

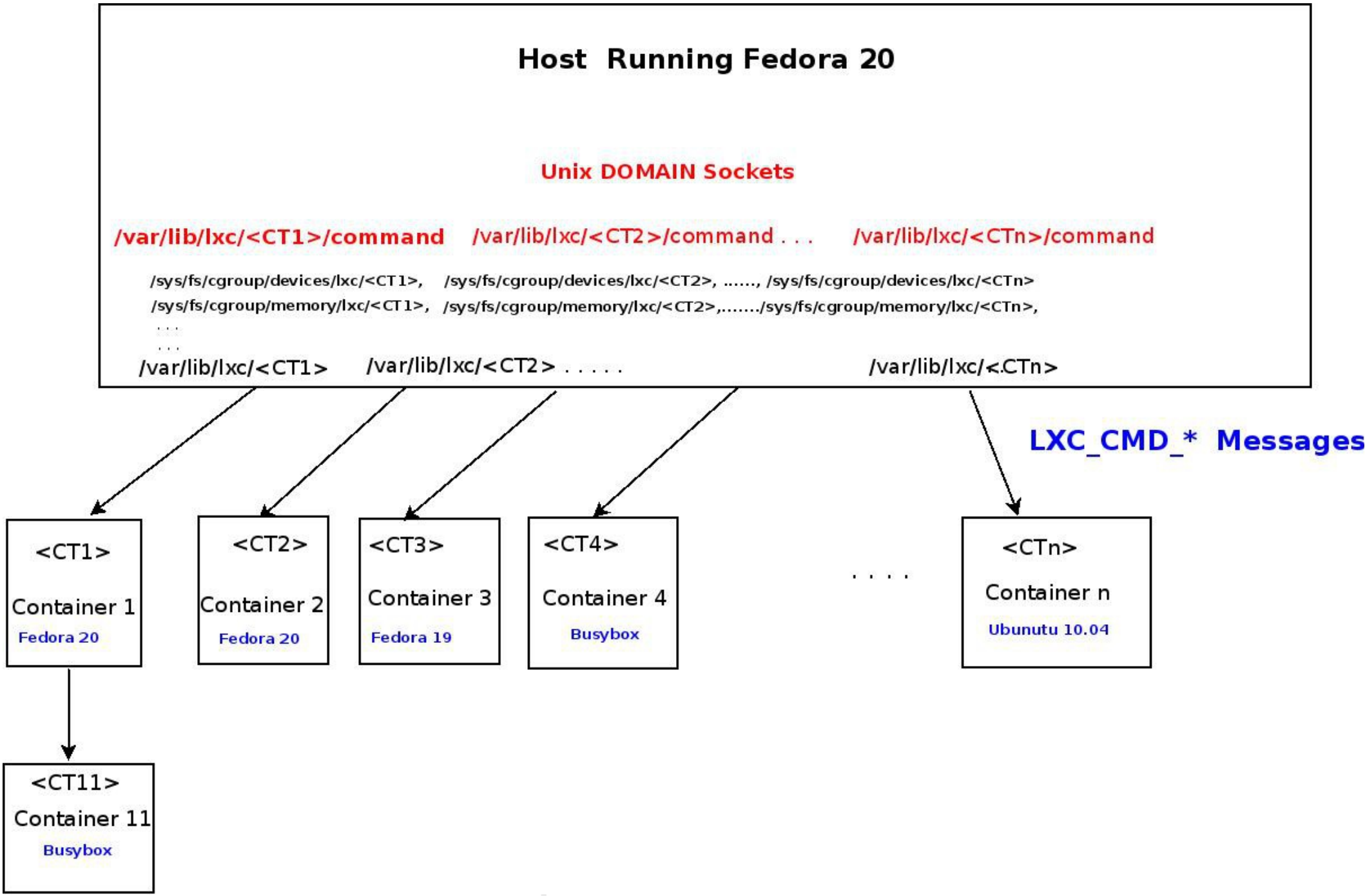
lxc-create -t fedora -n ubuntuCT -- -r raring

lxc-create -t busybox -n myBusyboxCT

- Example of removing a container:

lxc-destroy -n busyboxCT

=> */var/lib/lxc/myFedoraCT will be removed.*


Host Running Fedora 20
Unix DOMAIN Sockets
/var/lib/lxc/<CT1>/command
/var/lib/lxc/<CT2>/command . . .
/var/lib/lxc/<CTn>/command
/sys/fs/cgroup/devices/lxc/<CT1>, /sys/fs/cgroup/devices/lxc/<CT2>, ..., /sys/fs/cgroup/devices/lxc/<CTn>
/sys/fs/cgroup/memory/lxc/<CT1>, /sys/fs/cgroup/memory/lxc/<CT2>,...../sys/fs/cgroup/memory/lxc/<CTn>,
/var/lib/lxc/<CT1>
/var/lib/lxc/<CT2> . . . . .
/var/lib/lxc/<CTn>
LXC_CMD_* Messages
<CT1>
Container 1
Fedora 20
<CT2>
Container 2
Fedora 20
<CT3>
Container 3
Fedora 19
<CT4>
Container 4
Busybox
. . . .
<CTn>
Container n
Ubunutu 10.04
<CT11>
Container 11
Busybox

Unix sockets Commands

- There are 7 commands which can be sent by Unix domain sockets from a host to a container (but not vice verse):
- LXC_CMD_CONSOLE (console)
- LXC_CMD_STOP (stop)
- LXC_CMD_GET_STATE (get_state)
- LXC_CMD_GET_INIT_PID (get_init_pid)
- LXC_CMD_GET_CLONE_FLAGS (get_clone_flags)
- LXC_CMD_GET_CGROUP (get_cgroup)
- LXC_CMD_GET_CONFIG_ITEM (get_config_item)

LXC management

- ***Running*** *a container named busyboxCT:*
  lxc-start -n busyboxCT
- *This in fact creates with* ***clone()*** *a new process (with at least a new PID and MNT namespace) , and subsequently calls* ***execvp()*** *with* ***/sbin/init*** *of the busybox container*
- *A* ***python3*** *short script to start a container named fedoraNew:*

```
#!/usr/bin/python3

import lxc

container = lxc.Container("fedoraNew")
container.start()
```

Stopping *a container named busyboxCT:*

- *lxc-stop -n busyboxCT*
  - *This sends a SIGPWR signal.*
  - *You can also perform a non-graceful shutdown with SIGKILL by:*

lxc-stop -n busyboxCT -k

- *lxc-start -n <containerName> -l INFO -o /dev/stdout*
  - *For showing log messages with logpriority of INFO.*
- *lxc-start is for system containers.*
- *lxc-execute is for application containers.*
- *Monitoring a container:*

  lxc-monitor -n busyboxCT

'busyboxCT' changed state to [STOPPING]

'busyboxCT' changed state to [STOPPED]

- *lxc-ls --active*

Shows all running containers.

- *lxc-clone* – for copy on write.
- *lxc-snapshot.*

Freezing a container

- *lxc-freeze -n <containerName>*
- Will write “frozen” to */sys/fs/cgroup/freezer/lxc/<containerName>/freezer.state*
- *lxc-unfreeze -n <containerName>*
- Will write “THAWED” to */sys/fs/cgroup/freezer/lxc/<containerName>/freezer.state*

- ***Displaying info about a container is done with***

  lxc-info -n bbCT:

  Name: bbCT

  State: RUNNING

  PID: 415

  IP: 192.168.122.85

  . . .

- The pid is the process ID as seen from the host.

LXC releases

- **LXC 1.0** was released in February (20.2.2014).
  - Available in: http://linuxcontainers.org/downloads/
- About 10 months after LXC-0.9.0 was released, in April 2013.
- LXC is included in **Ubuntu 14.04** (Trusty Tahr) LTS (Long Term Support):
- Ubuntu 14.04 comes with 5 years of security and bug fixes updates: end of life in April 2019.

LXC Templates

These are the 14 shell templates available in LXC 1.0 (there were 11 in LXC 0.9.0).

lxc-busybox – **only 376 lines.**

lxc-centos

lxc-cirros

lxc-debian

lxc-download

lxc-fedora – **1400 lines**

lxc-gentoo

lxc-openmandriva

lxc-opensuse

lxc-oracle

lxc-plamo

lxc-sshd

lxc-ubuntu

lxc-ubuntu-cloud

Containers configuration

- If lxc.utsname is not specified in the configuration file, the hostname will be the same as of the host that created the container.

  When starting the container with the following entries:

  lxc.network.type = phys

  lxc.network.link = em1

  lxc.network.name = eth3

  The em1 network interface, which was in the host, will be moved to the new container, and will be named eth3. You will no longer see it in the host. After stopping the container, it will return to the host.

Setting Container Memory by cgroups

Setting cgroups entries can be done from the host by one of three ways.

For example, for setting the container memory, we can use:

1) In the container configuration file:

lxc.cgroup.memory.limit_in_bytes = 512M

- Then:

cat /sys/fs/cgroup/memory/lxc/fedoraCT/memory.limit_in_bytes

536870912

(instead of the default, which is 18446744073709551615 bytes)

2) lxc-cgroup -n fedoraCT memory.limit_in_bytes 512M

3) echo 512M > /sys/fs/cgroup/memory/lxc/fedoraCT/memory.limit_in_bytes

- *lxc-cgroup -n fedora devices.list*
  - *displays the allowed devices to be used on a container called “fedora”:*

*c *:* m*

*b *:* m*

c 1:3 rwm

c 1:5 rwm

c 1:7 rwm

c 5:0 rwm

c 1:8 rwm

c 1:9 rwm

c 136: rwm*

c 5:2 rwm

Disabling the out of memory killer

- You can disable the out of memory killer with memcg:
- *echo 1 > /sys/fs/cgroup/memory/0/memory.oom_control*
- This **disables** the oom killer.
- *cat /sys/fs/cgroup/memory/0/memory.oom_control*
  - *oom_kill_disable 1*
  - *under_oom 0*

Security

- Anatomy of a user namespaces vulnerability (March 2013)
  http://lwn.net/Articles/543273/
  - CVE 2013-1858
  - It occurred when calling *clone()* with both CLONE_FS and CLONE_NEWUSER. The fix was to disable such mask.
  - See: https://www.mail-archive.com/stable@vger.kernel.org/msg34470.html
- RHEL 6 and 5 **disabled** user namespaces (CONFIG_USER_NS):

  https://bugzilla.redhat.com/show_bug.cgi?id=921448#c1
- RHEL 7: user namespaces are enabled in the kernel (for ABI comparability) but disabled in userspace.
- In Fedora 20 it is disabled, will probably be enabled in Fedora 21.

  Daniel J Walsh (SELinux expert), Red Hat.
  - Linux Security Summit (LSS), 2012: http://lwn.net/Articles/515034
  - Libvirt-sandbox (libvirt-lxc is too complex).
- “LXC is not yet secure. If I want real security I will use KVM”.
- Daniel P. Berrangé, September 2011, https://www.berrange.com/tags/containers/

Security – contd

- **seccomp** (secure computing mode)
  - The seccomp kernel layer was developed by Andrea Arcangeli
  - By setting lxc.seccomp to a path for a seccomp policy file, you can define **seccomp policy** (which means to specify the numbers of the system calls which are allowed).
  - **seccomp2**: specifying the system calls by name.
- **capabilities**
  - You can disable capability in a container by setting lxc.cap.drop in the container.
  - For example:

    lxc.cap.drop=sys_module
  - disables CAP_SYS_MODULE, so **insmod** from within a container will fail.
  - See man 7 capabilities

Security – contd

- **Config entries:**
- For example, by default, /proc is mounted as rw, so the following sequence will reboot the host:

  echo 1 > /proc/sys/kernel/sysrq

  echo b > /proc/sysrq-trigger
- Setting lxc.mount.auto = proc:mixed in the config file will mount **/proc** as read-write, but remount **/proc/sys** and **/proc/sysrq-trigger** as read-only.
- You can also use **lxc.mount.auto = sys:ro** to mount **/sys** as read only.
- There are other options.

- **unprivileged containers** are containers which are created by a **non-root** user.
  - **requires kernel 3.13 or higher.**
  - **User namespaces should be enabled in the kernel.**
  - A lot of work was done in order to provide support for **unprivileged containers.**
  - **/etc/subuid**
  - **/etc/subgid**

Unprivileged containers

- links:
  - Introduction to unprivileged containers,
  - post 7 out of 10 in the LXC 1.0 blog post series by Stéphane Graber:

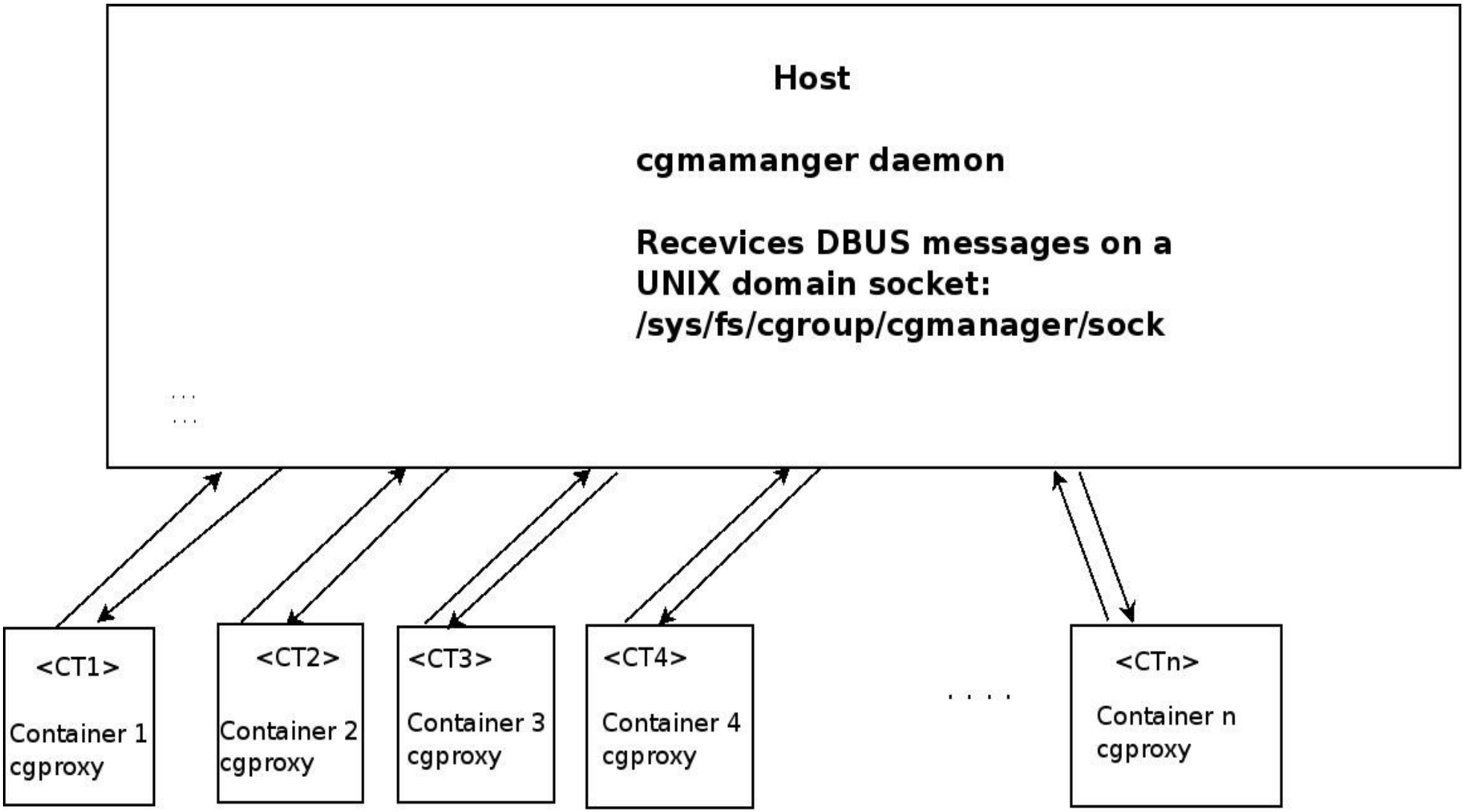
    https://www.stgraber.org/2014/01/17/lxc-1-0-unprivileged-containers/
  - Semi-Unprivileged Linux Containers (LXC) on Debian 7 Stable, Assaf Gordon:

    http://crashcourse.housegordon.org/LXC-semi-unprivileged-containers.html

Cgmanager – The cgroup manager

- A daemon which is started early and opens a UNIX domain socket on /sys/fs/cgroup/cgmanager/sock; developed by Serge Hallyn.
- In order to create cgroups, DBUS messages (DBUS method calls) are sent over this socket.
- Will **Systemd** use the cgmanager ?

Cgmanger – Diagram



Cgmamanger – Example

- **Example (based on *lxc/src/tests/lxc-test-unpriv*)**

```
cat /proc/cgroups
#subsys_namehierarchy num_cgroups enabled
cpuset 3 2 1
cpu 4 5 1
cpuacct 5 2 1
memory 6 2 1
devices 7 5 1
freezer 8 2 1
net_cls 9 2 1
blkio 10 2 1
perf_event 11 2 1
hugetlb 12 2 1
```

Create cgroups called “test” with DBUS messages - example

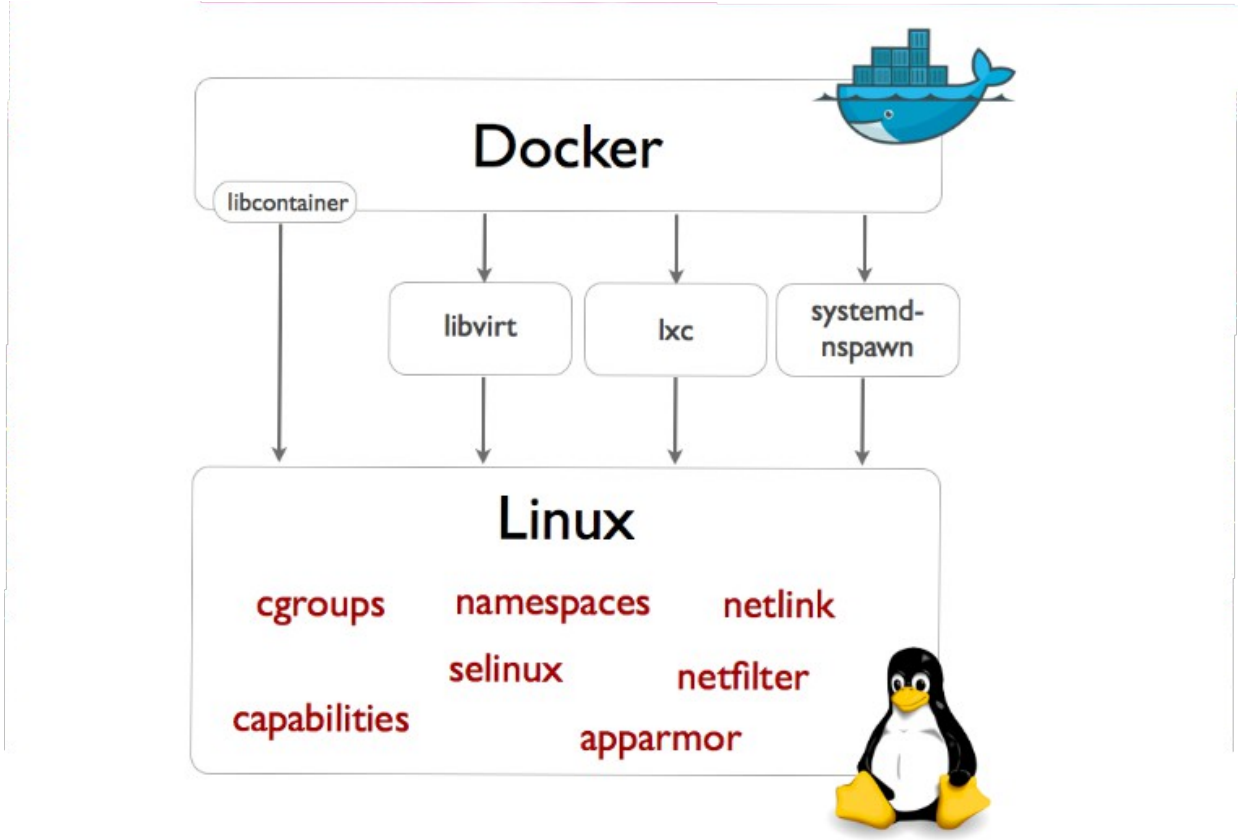
```
#!/bin/sh

TUSER="test"
if [ -e /sys/fs/cgroup/cgmanager/sock ]; then
    for d in $(grep -v ^# /proc/cgroups | awk '{print $1}'); do
     dbus-send \
       --address=unix:path=/sys/fs/cgroup/cgmanager/sock \
       --type=method_call
       /org/linuxcontainers/cgmanager org.linuxcontainers.cgmanager0_0.Create \
       string:$d string:$TUSER
    done
fi
```

Docker

- **Docker** is a Linux container engine.
- An open Source project; first release: 3/2013, by dotCloud (a company which later changed its name to Docker Inc). Written first in ***Python***, and later in a programming language called ***Go*** (initially developed by Google).
- https://www.docker.io/
  - git repository: https://github.com/dotcloud/docker.git

- **Docker 0.8 –** release in February 2014 (includes Mac OSX support).
- **Docker 0.9 –** released in 10.3.14.
  - New default driver: **libcontainer**. https://github.com/dotcloud/docker/tree/master/pkg/libcontainer
  - By default, does **not** use LXC at all to create containers, but uses cgroups/namespaces directly.
  - Does not support currently user namespaces.
  - In order to switch to working with LXC, you should run: ***docker -d -e lxc***
  - *The following diagram is from the Docker website (Thanks to Jerome Petazzoni from Docker Inc. for giving explicit permission to show it)*

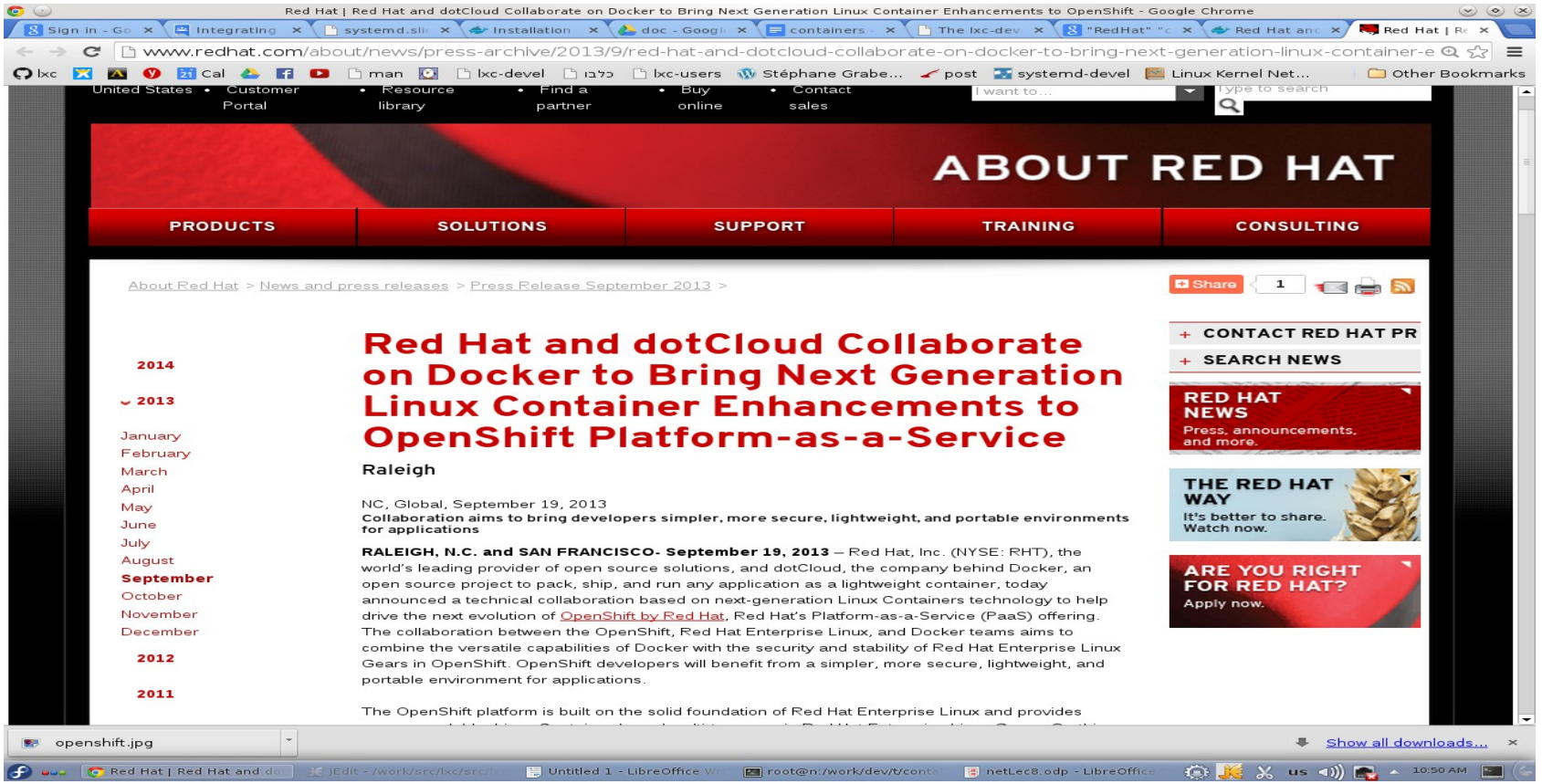

Docker
libcontainer
libvirt
lxc
systemd-
nspawn
Linux
cgroups
namespaces
netlink
selinux
netfilter
capabilities
apparmor

Docker 1.0

- **Docker 1.0** – intended for Q2 of 2014.
  - Still probably not production ready.
  - Anyhow, according to Solomon Hykes (Docker CTO), non privileged containers (using user namespaces) are **not** a pre-requisite for Docker (https://groups.google.com/forum/#!topic/docker-dev/MoIDYDF3suY)

Docker and RedHat collaboration

In **September 2013**, RedHat announced collaboration with dotCloud. There was a Docker package for Ubuntu, based on LXC.



Docker and RedHat collaboration

Initially, the plan was that RedHat will add code to Docker, for using XML-based libvirt-lxc, which is also used by RedHat in Xen, KVM and other virtualization solutions.

Dan Walsh recently (Red Hat Czech conference 2014) suggested to use systemd-nspwan container instead of Libvirt-LXC / LXC.

Systemd-nspwan is relatively short (3170 lines)

- SELinux support to **systemd-nspwan** was added by him.

Docker - examples

- *On Fedora 20:*
  - *yum install docker-io*
  - *Start the docker daemon: systemctl start docker.sevice*
  - *docker run -i -t ubuntu /bin/bash*
  - *This runs a **root** Ubuntu container, which was prepared beforehand by the Docker team. You **cannot** modify the **root** container images in the docker repository.*

Docker – examples (contd.)

- *docker stop* <container ID> – to stop a container. (you can get the container ID by docker ps).
- Run *docker help* to get the list of available commands.
- Docker has some similarities to git.
  - As opposed to git, which is text-based, Dokcer deals with binaries.
  - No rebase or merger operations.
- For example, if you will run from inside a container:

  touch /x.txt

  touch /y.txt

  yum install net-tools

  and then exit the container and enter again, you will **not see** these changes (as opposed to what happens when you work with containers by the *lxc-start* tool)
- b0814f648e7b is the container ID, which you can get by *docker ps*
- *docker diff <containerID>* will show the changes you made.

First trial

docker diff shows:

- *'A' -> Add*
- *'D' -> Delete*
- *'C' -> Change*

docker commit b0814f648e7b fedora

docker push fedora

2014/01/25 19:31:04 Impossible to push a "root" repository. Please rename your repository in <user>/<repo> (ex: <user>/fedora)

- *docker commit b0814f648e7b rruser/fedora*
- *You will need to enter credentials:*

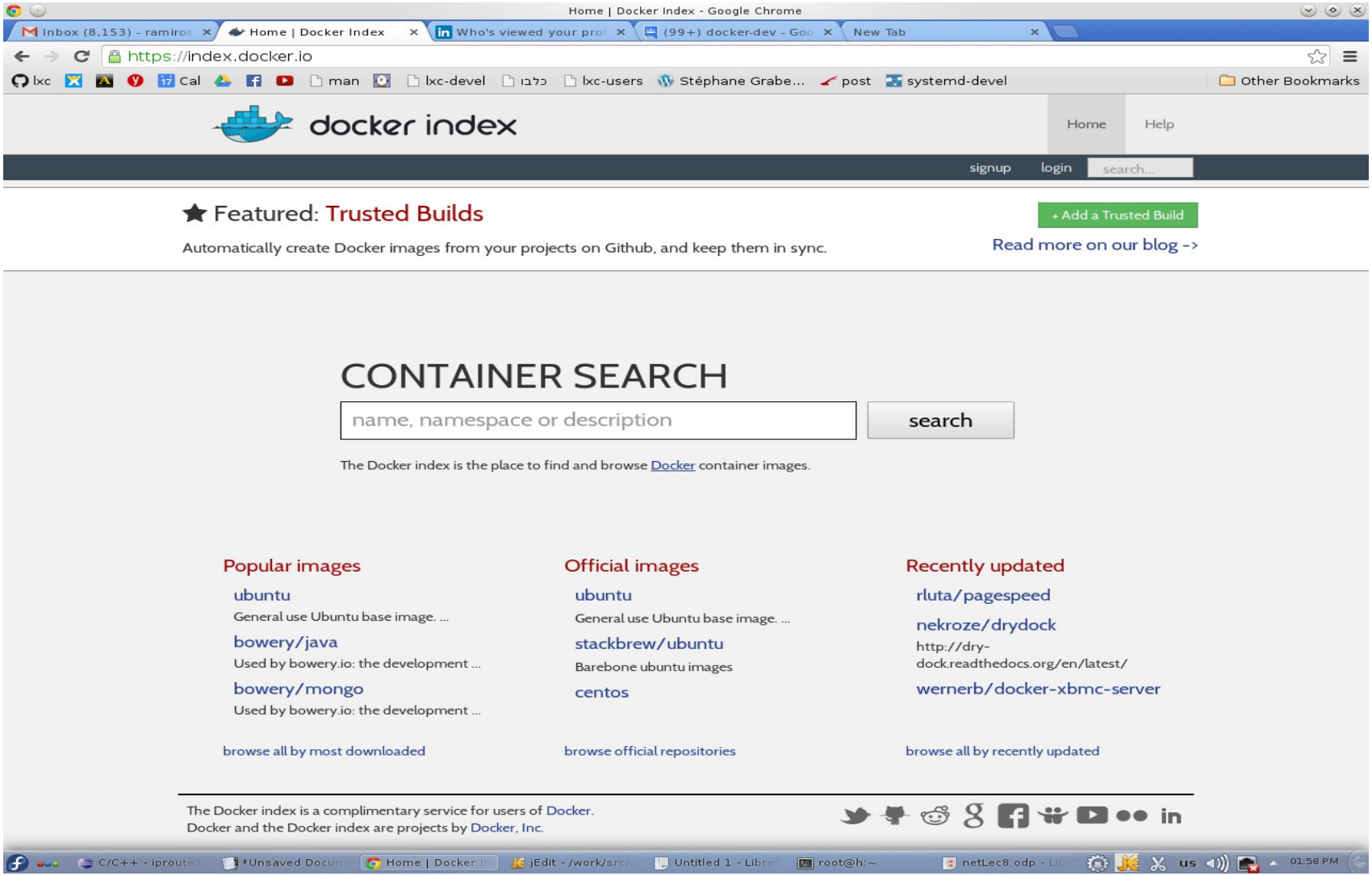
  Please login prior to push:

  Login against server at https://index.docker.io/v1/

  Username:

- *As in git, you can add -m “message text”.*
- *You can create an account free registration in: https://index.docker.io/account/signup/*
- *You use the username/password you get for pushing the image to the docker repository.*
- *You can install a private docker registry server.*

Docker public registry: https://index.docker.io



Dockerfile

- The Dockerfile is composed of a sequence of commands
- It has a simple syntax for automating building images.
- Dockerfile entries start with an uppercase commands.
- The first line in a Dockerfile must start with “FROM”, specifying base image, like Fedora or Ubuntu.
- **Example of a simple Dockerfile:**

  FROM fedora

  MAINTAINER Rami Rosen

  RUN yum install net-tools

- We can use the docker build command to create containers according to a specified Dockerfile:

  docker build -t rruser/fedora .

  Dockerfile tutorial in :http://www.docker.io/learn/dockerfile/

Docker and LXC

- Docker before 0.9 use LXC to create and manage containers.
  - Docker versions before 0.9 do **not** use LXC GO bindings.
  - Docker 0.9 (3/14) by default does not use LXC at all to create containers, but uses cgroups/namespaces directly (libcontainer), without using any of the LXC userspace commands.
  - In order to switch to working with LXC, you should run the docker daemon thus:
    - ***docker -d -e lxc***
  - Using libcontainer: This will remove the burden of supporting many LXC versions.
  - https://github.com/dotcloud/docker/tree/master/pkg/libcontainer

- From: http://docs.docker.io/en/latest/

“Please note Docker is currently under heavy development. It should not be used in production (yet).”

CRIU

- Checkpoint/Restore for Linux in userspace (Pavel Emalyanov is the team leader.)
- An OpenVZ project; Licensed under GPLv2.
- http://www.criu.org/Main_Page
- **Why do we need Checkpoint/Restore? Maintenance:**
  - Installing a new kernel.
  - HW addition/fixes or HW maintenance
  - Load Balancing.
  - Recovery from disaster by snapshots.

2010: the kernel community rejected Oren Laddan checkpoint/restore kernel patches . There was in fact a joint effort of OpenVZ, who had their own kernel implementation, IBM, Oren Laddan, and a couple of others; after it was not accepted, as it was too complicated, Pavel Emalyanov started to push the hybrid approach: C/R mostly in userspace, with much less code in kernel.

- V19 of the patch included **27,000-line diff** from 2.6.33-rc8(http://lwn.net/Articles/375855/)
- Following this rejection, OpenVZ team decided to stop work on checkpoint/restore in kernel, and to open the **CRIU** project, for checkpoint/restore in userspace.

CRIU – contd.

- More than 150 kernel patches (some of them used also outside of CRIU, for example UNIX sockets and the SS tool).
  - Usually wrapped in #ifdef CONFIG_CHECKPOINT_RESTORE blocks.
  - See: http://criu.org/Commits
- C/R tool 0.1 - July 2012: the first release of the checkpoint-restore tool: http://lwn.net/Articles/507796/
- C/R tool 0.2 – September 2012: Support for dump and restore a simple LXC container.

  http://lwn.net/Articles/517079/
- C/R tool 1.0 – November 2013.
- C/R tool 1.1 – January 2014.
- C/R tool 1.2 – February 2014.
  - p.haul (Process haul) git branch - migration of containers and also arbitrary processes.

    - U-Haul is a moving equipment company in the USA...

CRIU-Example

- ***Checkpointing*** *is done by:*

  criu dump -t $PID -D images

  - ***-D*** *tells the folder to put the generated image files.*
- ***Restoring*** *is done by:*

  criu restore -D images $PID

CRIU is intended to support checkpoint/restore of:

- *OpenVZ containers*
- *LXC containers*
- *Docker containers*
- *More in the future.*

CRIU plugins

- Intergrated into CRIU 1.1
  - Intended for external dependencies.

- External Unix socket (test/unix-callback/)
- External bind mounts (test/mounts/ext/)
- External files (test/ext-links/)
- More.
- See: http://criu.org/Plugins
- CRIU doesn't support (currently) cgroups, hugetlbfs, fusectl, configfs, mqueue,debugfs file systems in containers

Summary

- We have discussed various ingredients of Linux-based containers:
  - namespaces enable secure/functional separation
  - cgroups allow resource usage control
  - LXC automates it using config files and userspace commands.
  - The Docker engine simplifies creating and deploying containers into the cloud.
  - CRIU adds the ability to save/restore processes and containers in particular.

Tips

- How can I know that I am inside a containers ?
  - Look at *cat /proc/1/environ*
    - *With LXC containers, this will show container=lxc*
  - If it a Fedora container, *systemd-detect-virt* will show: lxc.

Thank you!