LinuXContainer

Nutzung unprivilegierter LinuXContainer als normaler Nutzer

Dirk Geschke

Linux User Group Erding

28. Januar 2015

Gliederung

1. Einleitung
2. Namespaces
3. Zwischenfazit
4. Control Groups
5. Aufbau unprivilegierter LinuXContainer
6. All together
7. Ausblick
8. Fazit

Allgemeines zu LXC

- leichtgewichtige Virtualisierungslösung
- Isolation von anderen Prozessen
- vergleichbar unter Linux mit OpenVZ, VServer, UML
- vergleichbar mit Solaris Zones
- vergleichbar mit BSD jails
- gemeinsamer Kernel
- verschachtelbar

Allgemeines zu LXC

- verbesserter `chroot()` (`pivot_root()`)
- Isolation dank Namespaces
- Ressourcenverwaltung via cgroups
- Absicherung möglich durch
  - Seccomp
  - AppArmor
  - SELinux
  - Capabilities
- User Namespaces seit Kernel 3.12
  $\Longrightarrow$ unprivilegierte Container sind möglich: LXC 1.0
  $\Longrightarrow$ `root` im Container $\neq$ `root` im Hostsystem möglich

Vor- und Nachteile

Vorteile:

+ wenig Ressourcen notwendig, mehr VMs als bei KVM
+ sehr performant
+ minimales System möglich
+ sehr gut isoliert
+ nativer Kernelsupport
+ kein CPU-Support notwendig

Nachteile:

- nur Hostkernel verwendbar, da gemeinsamer Kernel
- nur Linux lauffähig
- nur gleiche Architektur lauffähig

Anwendungsbereiche

- Testumgebung
  - viele Systeme via Netzwerk
  - systemnahe Entwicklung
  - Test ohne Buildumgebung
  - Installation und Test von beliebiger Software
- Absicherung möglicherweise unsicherer Dienste: Webserver, Browser, . . .
- . . .

Namespaces im Linuxkernel

- Isolierung: Prozesse haben unterschiedliche Sicht auf das System
- kein Hypervisor notwendig
- 2 neue System Calls: `setns()` und `unshare()`

  setns betritt einen existierenden Namespace

  unshare erzeugt neuen Namespace und setzt den aktuellen Prozess in diesen
- Erweiterung von `clone()` um entsprechende Flags

Notabene: Namespaces haben im Kernel keinen Namen!

Namespaces im Linuxkernel

- müssen im Kernel aktiviert werden
- jeder Namespace ist über einen inode repräsentiert, z.B:

```
$ ls -l /proc/self/ns
total 0
... ipc -> ipc:[4026531839]
... mnt -> mnt:[4026531840]
... net -> net:[4026531955]
... pid -> pid:[4026531836]
... user -> user:[4026531837]
... uts -> uts:[4026531838]
```

Tools für das Userland

iproute	`ip` mit Option `netns` für Netzwerk-Namespaces
util-linux	`unshare`, `nsenter`
lxc	`lxc-unshare`, `lxc-usernsexec`
shadow	`newuidmap`, `newgidmap` für User-Namespaces

6 Namespaces

uts	Unix Time Sharing
ipc	Inter Process Communication
net	Netzwerk Namespace
mnt	Mount Namespace
pid	Prozess Namespace
user	User Namespace

Unix Time Sharing Namespace

Hier können pro Namespace die `uname` Werte gesetzt werden:

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

Beispiel:

```
unshare -u /bin/bash; hostname ...
```

IPC Namespace

Die Inter-Prozess-Kommunikation kann hierdurch separiert werden:

- semaphoren
- shared memory
- message queues

Diese sind in anderen Namespaces nicht zu sehen.
(außer auf dem Host)

UTS Namespace ist *per default* eine Kopie des Originals
IPC Namespace ist *per default* leer!

Network Namespace

- eigene Routen
- eigene Firewallregeln
- eigene Netzwerkinterfaces
- Devices gehören exakt zu einem Namespace
- `ip netns` richtet diese in `/var/run/netns/` ein
- `ethtool -k` *device* zeigt Namespace an (`netns-local: ...`)

Network Namespace

Einige nützliche Kommandos:

ip netns monitor überwacht Erstellung/Löschung neuer Namespaces

ip netns add *ns* Erstellen eines Namespaces

ip netns del *ns* Löschen eines Namespaces

ip netns list Auflisten aller Namespaces

ip netns pids *ns* Welche PIDs gehören zum Namespace?

ip netns identify *pid* Welcher Namespace gehört zur PID?

ip link set *device* ns *ns* Verschieben eines Devices in Namespace

ip netns exec *ns command* Ausführen eines Kommandos im Namespace

Network Namespace

Beispiel:

1. Verschieben von `eth0` nach *myns1*:

```
# ifconfig eth0
# ip netns add myns1
# ip link set eth0 netns myns1
# ifconfig eth0 ⟶ no such device
```

2. Netzwerkinterface im *myns1* Namespace:

```
# ip netns exec myns1 /bin/bash
# ifconfig eth0
```

3. Zurückschieben von `eth0`:

```
# ip link set eth0 netns 1
```

veth: Virtuelles Ethernetpaar

Mit `veth` kann ein virtuelles Ethernetpaar erstellt und in Namespaces verteilt werden:

- `ip nets exec myns1 bash`
- `ip link add name veth0 type veth peer name veth0_peer`
- Nun sind 3 Interfaces in *myns1* vorhanden:
  1. Loopbackinterfaces `lo`, hat jeder Namespace
  2. Ethernetschnittstelle `veth0`
  3. Ethernetschnittstelle `veth0_peer`
- `ip link set dev veth0_peer netns myns2`
- `ip link set dev veth0_peer name veth0`
- Netzwerk Namespaces verwenden `/etc/netns/`*myns1*`/` statt `/etc/`, z.B. für `hosts` oder `resolv.conf`

Mount Namespace

- *per se* alle mounts sichtbar
- mounts in Namespace sind für andere nicht sichtbar
- Kommando `mount` wertet *per default* `/etc/mtab` aus
- Datei `/proc/mounts` ist per Namespace
- funktioniert **nicht** mit `systemd`: verwendet *shared flag*
- Lösung: `mount --make-rprivate -o remount ...`

PID Namespace

- gleiche PID ist in unterschiedlichen Namespaces verwendbar
- erster Prozess bekommt immer PID 1

```
# lxc-unshare -s PID /bin/bash
# echo $$
1
```

- dieser Prozess erbt verwaiste Prozesse (*à la* `init`)
- dieser Prozess ist auch mit SIGKILL nicht beendbar
- PID im Parent sichtbar, jedoch mit **realer** PID

User Namespace

- unterschiedliche UID, GID und Capabilities
- im Parent sind diese via `/proc/`*pid*`/uid_map` sicht- und steuerbar
- `/etc/subuid` und `/etc/subguid` listen erlaubten Bereich pro Nutzer: *username*:*1.UID/GID*:*Zahl der UIDs/GIDs*
- einziger Namespace ohne `CAP_SYS_ADMIN`, also nicht-`root`, nutzbar

```
$ lxc-unshare -s USER /bin/bash
$ id -u
65534
```

Finden der PID und `echo 0 1000 1 > /proc/17498/uid_map`:

```
$ id -u
0
```

Zwischenfazit

1. User Namespace erzeugen und `root` im Namespace werden
2. alle anderen Namespaces sind nun erstellbar
3. *bind-mount* von *Devices* in neue Umgebung
4. `pivot_root()` in neue Umgebung
5. `mount` von `/proc` und `/sys` in neuer Umgebung

Es fehlt noch etwas:

Ressourcenverwaltung $\longrightarrow$ *cgroups*

cgroups

- ursrprünglich von Google (2006) für Android: *process containers*
- seit Ende 2007 heißen sie cgroups
- sind in `/proc/cgroups` und `/proc/`*pid*`/cgroup` einsehbar
- 4 Dateien pro cgroup: `tasks, cgroup.procs, cgroup.event_control, notify_on_release`
- plus spezielle Dateien pro cgroup–Ressource
- klassisch verwaltet über `/sys/fs/cgroup/`*resource-type*`/`*user-class*
- Verschieben eines Prozesses in cgroup durch schreiben der PID in die `tasks`-Datei
- Redesign seit 2013: zentrale Verwaltung der cgroups durch einen Prozess (ab Kernel 3.16)

cgroup Ressource-Typen

cpuset Verwendung der CPUs + zugehörigem Speicher (NUMA)

cpuacct Accounting des CPU-Verbrauchs pro cgroup user-class

devices Begrenzung der Zugriffe auf Device-Dateien via `devices.allow` und `devices.deny`:

- **r** read
- **w** write
- **m** mknod

sowie Typen:

- **a** all devices
- **c** character devices
- **b** block devices

zum Beispiel `/dev/null` erlauben, alles andere verbieten

```
echo 'c 1:3 mr' > /sys/.../devices.allow
echo a > /sys/.../devices.deny
```

cgroup Ressource-Typen

freezer Einfrieren ganzer Prozessgruppen auf einmal

hugetlb Begrenzung der Verwendung von HugeTLB

memory Begrenzung von RAM, SWAP aber auch TCP-Buffer und OOM-Control

net_cls Taggen von Netzwerkpaketen, Verwendung der Tags via `tc` oder `iptables`

net_prio Piorisierung von Netzwerkpaketen

blkio Drosselung von Diskzugriffen, zwei Verfahren:

1. zeitbasierte Wichtung, erfordert `CFQ`–Scheduler (`blkio.weight`)
2. Drosselung durch Obergrenze I/O–Durchsatz (`blkio.throttle`)

Besonderheiten von cgroups

- generell mit Vorsicht zu verwenden
  $\Longrightarrow$ Lasterhöhung, Perfomanceverluste
- memory–cgroup benötigt Kernelbootparameter
  `cgroup_enable=memory`
  Grund: Performance–Impact wenn nur aktiv!
- in Zukunft nicht mehr direkt setzbar, zentrales Tool zur Verwaltung von cgroups, z.B. `systemd`
- `systemd` benötigt cgroups zur Überwachung von Dämonen, Ressourcenkontrolle ist nur sekundär:

All your cgroups are belong to us!

`cgmanager`, ein alternativer cgroup–Manager

- läuft auf dem Host unter `root`–Kennung
- mounted cgroups in eigenem Namespace, für andere nicht sichtbar
- stellt `unix socket` zur Kommunikation bereit
- verwendet D-Bus zur Kommunikation, ähnlich wie `systemd`
- stellt Verbindung zwischen UID, GID und PID mit cgroup dar
- `cgm` als Kommandozeilentool für D-Bus–Operationen mit `cgmanager`

Erstellen eines unprivilegierten Containers

Voraussetzungen: aktuelles `shadow`-Paket, aktuellen Kernel

1. Erlauben eines Benutzers Netzwerkinterfaces auf dem Host anzulegen in `/etc/lxc/lxc-usernet`, z.B.:

   ```
   lxcuser veth lxcbr0 10
   ```

   Die Werte sind: Benutzer, Interfacetyp, Bridge auf dem Host, maximale Anzahl an Interfaces.
   `lxc-usernet` ist das einzige SUID-Programm!
2. Sicherstellen, dass der Benutzer subUIDs und subGIDs haben darf (`/etc/subuid` und `/etc/subgid`), z.B.:

   ```
   $ grep lxcuser /etc/subuid /etc/subgid
   /etc/subuid:lxcuser:100000:65537
   /etc/subgid:lxcuser:100000:65537
   ```

Erstellen eines unprivilegierten Containers

3. starten als root von `cgmanager` und der Kommandos

```
# cgmanager -daemon -m name=systemd
# cgm create controller cgroup
# cgm chown controller cgroup uid gid
```

Der *controller* kann `all` sein, *uid* und *gid* sind die IDs des unprivilegierten Nutzers, z.B.:

```
# cgm create all wheezy
# cgm chown all wheezy 1001 1001
```

4. verschieben der aktuellen Shell des Nutzers in die cgroup:

```
$ cgm movepid all wheezy $$
```

Erstellen eines unprivilegierten Containers

5. Erstellen von Verzeichnissen und kopieren einer default-Konfiguration:

```
$ mkdir ~/.config/lxc
$ cp /etc/lxc/default.conf ~/.config/lxc
```

Anfügen von zwei Zeilen in Konfigurationsdatei

```
lxc.id_map = u 0 100000 65536
lxc.id_map = g 0 100000 65536
```

6. Erstellen des Containers durch Download–Template:

```
$ lxc-create -t download -n wheezy
```

Optional, kann noch die Distribution, das Release und die Architektur mit angegeben werden:

```
... -- -d debian -r wheezy -a amd64
```

Erstellen eines unprivilegierten Containers

7. Starten des Containers:

   ```
   $ lxc-start -n wheezy
   ```

   Das kann durch `--daemon` in den Hintergrund gesendet werden.

8. mit dem Container verbinden und man ist `root` im Container:

   ```
   $ lxc-attach -n wheezy
   ```

   Das kann notwendig sein um das Netzwerk zu konfigurieren, per default ist es DHCP. Es kann aber auch so ein root-Passwort gesetzt werden!

Details...

- Container landen per default hier:

  `~/.local/share/lxc/`*container*`/`
- im Verzeichnis `rootfs` befindet sich die Wurzel des Containers
- dort gibt es eine Konfigurationsdatei `config` für den Container, hier können diverse Einstellungen vorgenommen werden, z.B.:
  - Begrenzung des Speichers:

    ```
    lxc.group.memory.limit_in_bytes = 1G
    lxc.group.memory.memsw.limit_in_bytes = 2G
    ```
- *systemd*-basierte Distributionen funktionieren derzeit noch nicht (richtig)

Nun passt alles zusammen

1. User Namespace wird als erstes generiert
2. in diesem Namespace gibt es nun `root`-User
3. weitere Namespaces generieren, als `root` im User Namespace!
4. Netzwerkinterfacepaar `veth` wird erstellt, eine Hälfte wird an den Host *verschoben* und dort in die `bridge` eingehängt.
5. bindmount relevante Dateien/Verzeichnisse nach `/dev`
6. pivot_root in das Container `rootfs`
7. mounten von `/proc` und `/sys`
8. cgroups zur Ressourcenverwaltung
9. starten von /sbin/init im Container

Ein Blick nach vorne

- LXD: zentraler Dienst zum Verwalten von Containern auf verschiedenen Hosts:
  - Kommandozeilentool `lxc` zur Verwaltung lokal wie remote
  - Live-Migration ist damit möglich
  - OpenStack Nova plugin $\longrightarrow$ *compute nodes* basierend auf Container
- lxcfs: ein *fuse*-Dateisystem zur *Emulation* von cgroups für z.B. `systemd` $\longrightarrow$ spezielle Dateien als Mountpunkte für cgroup-fuse-mounts

Fazit

- unprivilegierte LinuXContainer sind eine ressourcenschonende Containervirtualisierung
- erhöhte Sicherheit durch unprivilegierten Benutzer
- Auslagern von vielleicht kritischen Diensten
- ideale Testumgebung für viele Fälle
- Applikationslösung durch *unabhängige* Abhängigkeiten

Praxis && Fertig :-)