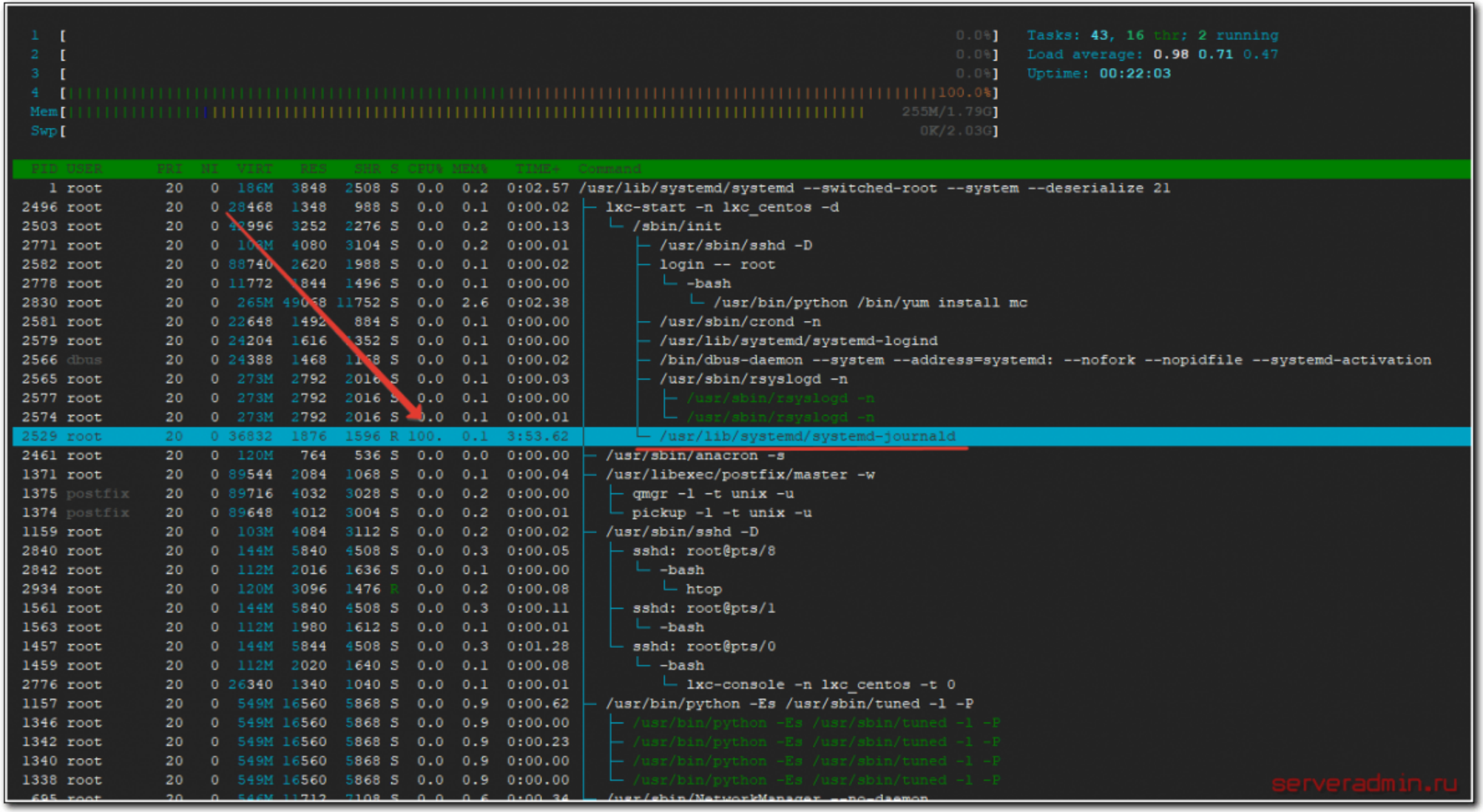

1 [0.0%]
2 [0.0%]
3 [0.0%]
4 [100.0%]
Mem[255M/1.79G]
Swp[0K/2.03G]
Tasks: 43, 16 thr; 2 running
Load average: 0.98 0.71 0.47
Uptime: 00:22:03
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
1 root 20 0 186M 3848 2508 S 0.0 0.2 0:02.57 /usr/lib/systemd/systemd --switched-root --system --deserialize 21
2496 root 20 0 28468 1348 988 S 0.0 0.1 0:00.02 lxc-start -n lxc_centos -d
2503 root 20 0 /sbin/init
2771 root 20 0 /usr/sbin/sshd -D
2582 root 20 0 88740 2620 1988 S 0.0 0.1 0:00.02 login -- root
2778 root 20 0 11772 -bash
2830 root 20 0 265M /usr/bin/python /bin/yum install mc
2581 root 20 0 22648 1492 884 S 0.0 0.1 0:00.00 /usr/sbin/crond -n
2579 root 20 0 24204 1616 /usr/lib/systemd/systemd-logind
2566 dbus 20 0 24388 1468 /bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
2565 root 20 0 273M 2792 /usr/sbin/rsyslogd -n
2577 root 20 0 273M 2792 /usr/sbin/rsyslogd -n
2574 root 20 0 273M 2792 /usr/sbin/rsyslogd -n
2529 root 20 0 36832 1876 1596 R 100. 0.1 3:53.62 /usr/lib/systemd/systemd-journald
2461 root 20 0 120M 764 536 S 0.0 0.0 0:00.00 /usr/sbin/anacron -s
1371 root 20 0 89544 2084 1068 S 0.0 0.1 0:00.04 /usr/libexec/postfix/master -w
1375 postfix 20 0 89716 4032 3028 S 0.0 0.2 0:00.00 qmgr -l -t unix -u
1374 postfix 20 0 89648 4012 3004 S 0.0 0.2 0:00.01 pickup -l -t unix -u
1159 root 20 0 103M 4084 3112 S 0.0 0.2 0:00.02 /usr/sbin/sshd -D
2840 root 20 0 144M 5840 4508 S 0.0 0.3 0:00.05 sshd: root@pts/8
2842 root 20 0 112M 2016 1636 S 0.0 0.1 0:00.00 -bash
2934 root 20 0 120M 3096 1476 R 0.0 0.2 0:00.08 htop
1561 root 20 0 144M 5840 4508 S 0.0 0.3 0:00.11 sshd: root@pts/1
1563 root 20 0 112M 1980 1612 S 0.0 0.1 0:00.01 -bash
1457 root 20 0 144M 5844 4508 S 0.0 0.3 0:01.28 sshd: root@pts/0
1459 root 20 0 112M 2020 1640 S 0.0 0.1 0:00.08 -bash
2776 root 20 0 26340 1340 1040 S 0.0 0.1 0:00.01 lxc-console -n lxc_centos -t 0
1157 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.62 /usr/bin/python -Es /usr/sbin/tuned -l -P
1346 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.00 /usr/bin/python -Es /usr/sbin/tuned -l -P
1342 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.23 /usr/bin/python -Es /usr/sbin/tuned -l -P
1340 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.00 /usr/bin/python -Es /usr/sbin/tuned -l -P
1338 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.00 /usr/bin/python -Es /usr/sbin/tuned -l -P
serveradmin.ru

Я давно хотел посмотреть какую-нибудь систему контейнеризации, отличную от docker, так как очень его не люблю. Решил в итоге остановиться для начала на lxc, познакомиться, рассказать вам, как установить и настроить lxc на CentOS 7. Впечатления у меня от него не однозначные, расскажу обо всем по порядку.

Содержание:

Введение

Сразу предупреждаю, что данная статья это мое знакомство с lxc контейнерами. Я не имею практического опыта работы с ними, поэтому бездумно брать и применять мои рекомендации не стоит.

Не буду много писать о том, что такое контейнеры, чем они отличаются от виртуальных машин, и чем отличаются системы контейнеров (lxc, docker, openvz и др.) друг от друга. Все это можно найти в интернете на сайтах самих продуктов. Расскажу все своими словами.

1. Главное отличие контейнера от виртуальной машины в том, что он запускается на том же уровне железа, что и хостовый сервер. Нет необходимости в виртуальных устройствах. Контейнер видит исходное железо. Это плюс к производительности. Используется ядро исходной системы и изоляция ресурсов внутри системы.
2. Контейнер может масштабироваться по ресурсам до целого физического сервера. Например, контейнер может использовать тот же диск и файловую систему, что и хостовая машина. Нет необходимости разбивать диск на кусочки, как с виртуальными машинами и давать каждому по кусочку. В итоге, кто-то может свой диск вообще не использовать, а кому-то не хватит. С контейнерами можно поступить проще - все используют общий диск с хостом. Ограничений для диска как в виртуальной машине тем не менее все равно можно устанавливать. То же самое можно сделать с процессором и памятью.

Я почти не работал с контейнерами, кроме докера. С ним приходится работать только в связке с разработчиками. Для своих целей я его не использую, так как для моих задач он мне кажется неудобным. Не хочется сейчас здесь раскрывать эту тему, может быть в другой раз в статье о докер, если таковая будет. Но в целом докер я не люблю, особенно в продакшене.

Расскажу о том, почему мне захотелось посмотреть на контейнеры вместо виртуальных машин, которые использую повсеместно уже много лет.

1. Как уже написал ранее, привлекает возможность использовать ресурсы хостовой машины напрямую. Взял системный диск с корнем / на 1Тб и все контейнеры его используют пока есть место.
2. Легкий бэкап и доступ к файлам в контейнерах. Посмотреть файлы в контейнере можно просто зайдя в директорию контейнера с хостовой машины. Они все хранятся в открытом виде. Так их очень удобно бэкапить с помощью rsync, или каким-то другим способом.
3. Легко копировать, разворачивать, управлять контейнерами. Они занимают мало места, можно руками с хоста поправить какой-то конфиг в системе контейнера.

Например, у меня есть достаточно мощные VDS серверы от ihor. 2 ядра, 8 гигов, 150Гб диск. Иногда то, что там размещается, не использует полностью ресурсы виртуальной машины. Хочется как-то их занять, но при этом никак не затрагивать основные проекты на виртуальной машине. Иногда хочется какие-то тестовые среды создавать для сайтов и пробовать новые версии софта. Для этого надо отдельную виртуалку брать. Вместо этого я решил попробовать lxc контейнеры.

Использовать lxc контейнеры в плане сети можно двумя способами:

1. Заказывать каждому контейнеру отдельный внешний IP, настраивать для контейнера bridge на реальном сетевом интерфейсе и выпускать его напрямую в интернет. Этот вариант подходит, если есть ip адреса или не жалко для них денег. Так работать удобнее всего.

2. Настраивать виртуальный bridge, настраивать NAT и проброс портов с хостовой машины внутрь контейнеров. Не очень удобно, но тем не менее вполне рабочий вариант.

Я расскажу про оба способа, так как проверял и то и другое. Настраивать все будем на CentOS 7.

Если у вас еще нет своего сервера с CentOS 8, то рекомендую мои материалы на эту тему:

- Установка CentOS 8.
- Настройка CentOS.

Настройка сети для LXC контейнеров

Начнем с настройки сети для контейнеров. Нам понадобится пакет bridge-utils. Установим его:

```
# yum install bridge-utils
```

Настроим виртуальный бридж, который будут использовать только контейнеры внутри своей виртуальной сети - 10.1.1.1/24. Для этого создаем в директории */etc/sysconfig/network-scripts* файл *ifcfg-virbr0* следующего содержания:

```
# mcedit /etc/sysconfig/network-scripts/ifcfg-virbr0
```

```
DEVICE=virbr0
BOOTPROTO=static
IPADDR=10.1.1.1
NETMASK=255.255.255.0
ONBOOT=yes
TYPE=Bridge
```

```
NM_CONTROLLED=no
```

После изменения сетевых настроек лучше перезагрузиться. Проверим, что получилось:

```
# ip a
```

```
[root@kw etc]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: ens18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 0a:c1:98:50:00:e3 brd ff:ff:ff:ff:ff:ff
    inet 95.169.190.64/24 brd 95.169.190.255 scope global ens18
       valid_lft forever preferred_lft forever
    inet6 fe80::8c1:98ff:fe50:e3/64 scope link
       valid_lft forever preferred_lft forever
3: virbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP qlen 1000
    link/ether fe:e4:5d:b2:da:e1 brd ff:ff:ff:ff:ff:ff
    inet 10.1.1.1/24 brd 10.1.1.255 scope global virbr0
       valid_lft forever preferred_lft forever
    inet6 fe80::c0d1:59ff:fea5:f3ae/64 scope link
       valid_lft forever preferred_lft forever
```


Дальше нам может помочь статья по настройке шлюза на centos, так как часть функционала шлюза нам нужно будет реализовать на хосте. А именно:

- Включить маршрутизацию пакетов между сетевыми интерфейсами

- Настроить NAT для виртуальной сети контейнера
- Настроить проброс портов в контейнеры

Включаем маршрутизацию пакетов. Для этого в файл */etc/sysctl.conf* добавляем строку в самый конец:

```
net.ipv4.ip_forward = 1
```

Применяем изменение:

```
# sysctl -p
```

Теперь основное - настройка iptables. Вообще, она сходу не берется и чаще всего у людей возникают вопросы по работе, если настраивают первый раз. В CentOS 7 по-умолчанию установлен firewalld. Я не использую его и всегда отключаю. Не потому, что он плохой и неудобный, а потому, что привык работать с iptables напрямую, у меня много готовых конфигурация для него.

Отключаем firewalld:

```
# systemctl stop firewalld
# systemctl disable firewalld
```

Устанавливаем службы iptables:

```
# yum install iptables-services
```

Рисуем конфиг для iptables. Пример взял из статьи про настройку шлюза для локальной сети, только изменил адрес виртуальной сети и имя интерфейса. По сути нам нужно то же самое. Привожу конфиг с рабочего сервера:

```
# mcedit /etc/iptables.sh
```

```
#!/bin/bash
#
# Объявление переменных
export IPT="iptables"

# Интерфейс который смотрит в интернет
export WAN=ens18
export WAN_IP=95.169.190.64

# lxc сеть
export LAN1=virbr0
export LAN1_IP_RANGE=10.1.1.1/24

# Очистка всех цепочек iptables
$IPT -F
$IPT -F -t nat
$IPT -F -t mangle
$IPT -X
$IPT -t nat -X
$IPT -t mangle -X

# Установим политики по умолчанию для трафика, не соответствующего ни одному из правил
$IPT -P INPUT DROP
$IPT -P OUTPUT DROP
$IPT -P FORWARD DROP

# разрешаем локальный траффик для loopback
$IPT -A INPUT -i lo -j ACCEPT
```

```
$IPT -A OUTPUT -o lo -j ACCEPT

# Разрешаем исходящие соединения самого сервера
$IPT -A OUTPUT -o $WAN -j ACCEPT

# Разрешаем доступ из lxc наружу и обратно
$IPT -A FORWARD -i $LAN1 -o $WAN -j ACCEPT
$IPT -A FORWARD -i $WAN -o $LAN1 -j ACCEPT
$IPT -A INPUT -i $LAN1 -j ACCEPT
$IPT -A OUTPUT -o $LAN1 -j ACCEPT

# Включаем NAT
$IPT -t nat -A POSTROUTING -o $WAN -s $LAN1_IP_RANGE -j MASQUERADE

# Пробрасываем порты в контейнер lxc_centos
$IPT -t nat -A PREROUTING -p tcp --dport 23543 -i ${WAN} -j DNAT --to 10.1.1.2:22
$IPT -t nat -A PREROUTING -p tcp --dport 80 -i ${WAN} -j DNAT --to 10.1.1.2:80
$IPT -t nat -A PREROUTING -p tcp --dport 443 -i ${WAN} -j DNAT --to 10.1.1.2:443

# Состояние ESTABLISHED говорит о том, что это не первый пакет в соединении.
# Пропускать все уже инициированные соединения, а также дочерние от них
$IPT -A INPUT -p all -m state --state ESTABLISHED,RELATED -j ACCEPT
# Пропускать новые, а так же уже инициированные и их дочерние соединения
$IPT -A OUTPUT -p all -m state --state ESTABLISHED,RELATED -j ACCEPT
# Разрешить форвардинг для уже инициированных и их дочерних соединений
$IPT -A FORWARD -p all -m state --state ESTABLISHED,RELATED -j ACCEPT

# Включаем фрагментацию пакетов. Необходимо из за разных значений MTU
$IPT -I FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu

# Отбрасывать все пакеты, которые не могут быть идентифицированы
# и поэтому не могут иметь определенного статуса.
$IPT -A INPUT -m state --state INVALID -j DROP
```

```
$IPT -A FORWARD -m state --state INVALID -j DROP

# Приводит к связыванию системных ресурсов, так что реальный
# обмен данными становится не возможным, обрубаем
$IPT -A INPUT -p tcp ! --syn -m state --state NEW -j DROP
$IPT -A OUTPUT -p tcp ! --syn -m state --state NEW -j DROP

# Рзрешаем пинги
$IPT -A INPUT -p icmp --icmp-type echo-reply -j ACCEPT
$IPT -A INPUT -p icmp --icmp-type destination-unreachable -j ACCEPT
$IPT -A INPUT -p icmp --icmp-type time-exceeded -j ACCEPT
$IPT -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

# Открываем порт для ssh
$IPT -A INPUT -i $WAN -p tcp --dport 22 -j ACCEPT

# Записываем правила
/sbin/iptables-save  > /etc/sysconfig/iptables
```

> Не забудьте поменять имена сетевых интерфейсов и ip адреса. Я не рекомендую настраивать фаервол, если у вас нет доступа к консоли сервера. Так вы можете потерять управление сервером.

В моем примере показан проброс порта ssh, http и https внутрь контейнера с ip адресом 10.1.1.2. Дальше в примерах я его создам.

Делаем скрипт */etc/iptables.sh* исполняемым:

```
# chmod 0740 /etc/iptables.sh
```

Запускаем iptables и добавляем в автозагрузку:

```
# systemctl start iptables.service
# systemctl enable iptables.service
```

Выполняем скрипт с правилами:

```
# /etc/iptables.sh
```

Проверяем установленные правила:

```
# iptables -L -v -n
```

```
[root@kw etc]# iptables -L -v -n
Chain INPUT (policy DROP 5 packets, 333 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     all  --  lo     *       0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  virbr0 *       0.0.0.0/0            0.0.0.0/0
   23  1724 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0           state INVALID
    1   104 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp flags:!0x17/0x02 state NEW
    0     0 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0           icmptype 0
    0     0 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0           icmptype 3
    0     0 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0           icmptype 11
    0     0 ACCEPT     icmp --  *      *       0.0.0.0/0            0.0.0.0/0           icmptype 8
    0     0 ACCEPT     tcp  --  ens18  *       0.0.0.0/0            0.0.0.0/0           tcp dpt:22

Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 TCPMSS     tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp flags:0x06/0x02 TCPMSS clamp to PMTU
    0     0 ACCEPT     all  --  virbr0 ens18   0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  ens18  virbr0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
    0     0 DROP       all  --  *      *       0.0.0.0/0            0.0.0.0/0           state INVALID

Chain OUTPUT (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     all  --  *      lo      0.0.0.0/0            0.0.0.0/0
   22 12848 ACCEPT     all  --  *      ens18   0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  *      virbr0  0.0.0.0/0            0.0.0.0/0
    0     0 ACCEPT     all  --  *      *       0.0.0.0/0            0.0.0.0/0           state RELATED,ESTABLISHED
    0     0 DROP       tcp  --  *      *       0.0.0.0/0            0.0.0.0/0           tcp flags:!0x17/0x02 state NEW
[root@kw etc]#
```


Проверяем NAT и проброс портов:

```
# iptables -L -v -n -t nat
```

```
[root@kw etc]# iptables -L -v -n -t nat
Chain PREROUTING (policy ACCEPT 111 packets, 10077 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 DNAT       tcp  --  ens18  *       0.0.0.0/0            0.0.0.0/0           tcp dpt:23543 to:10.1.1.2:22
    0     0 DNAT       tcp  --  ens18  *       0.0.0.0/0            0.0.0.0/0           tcp dpt:80 to:10.1.1.2:80
    0     0 DNAT       tcp  --  ens18  *       0.0.0.0/0            0.0.0.0/0           tcp dpt:443 to:10.1.1.2:443

Chain INPUT (policy ACCEPT 3 packets, 180 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 3 packets, 211 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain POSTROUTING (policy ACCEPT 3 packets, 211 bytes)
 pkts bytes target     prot opt in     out     source               destination
    0     0 MASQUERADE  all  --  *      ens18   10.1.1.0/24          0.0.0.0/0
[root@kw etc]#
```


Это я разобрал пример, когда у контейнеров своя виртуальная сеть, без прямого доступа во внешнюю. Если же они будут бриджем выходить наружу с прямым ip, то iptables вообще трогать не надо. Достаточно включить маршрутизацию пакетов между интерфейсами, создать bridge и добавить туда реальный интерфейс сервера. Контейнерам подключать этот бридж. Работает так же, как и bridge в proxmox.

Создаем конфиг для нового бриджа:

```
# mcedit /etc/sysconfig/network-scripts/ifcfg-virbr1
```

```
DEVICE=virbr1
BOOTPROTO=static
IPADDR=192.168.13.25
NETMASK=255.255.255.0
GATEWAY=192.168.13.1
DNS1=192.168.13.1
ONBOOT=yes
TYPE=Bridge
NM_CONTROLLED=no
```

И приводим конфиг основного сетевого интерфейса к такому виду:

```
# mcedit /etc/sysconfig/network-scripts/ifcfg-eth0
```

```
TYPE=Ethernet
BOOTPROTO=none
DEVICE=eth0
ONBOOT=yes
NM_CONTROLLED=no
BRIDGE=virbr1
```

После изменения сетевых настроек сервер лучше перезагрузить. Будем считать, что с сетью разобрались и все настроили. Еще раз напоминаю:

- **virbr0** - виртуальный бридж, который создает виртуальную локальную сеть для контейнеров. Выход во внешнюю сеть они осуществляют с помощью хоста, где настроен NAT и проброс портов с помощью iptables.
- **virbr1** - бридж, который включает в себя реальный физический интерфейс хоста. С помощью этого бриджа контейнеры получают прямой доступ во внешнюю сеть.

Установка LXC на CentOS 7

Первым делом подключаем репозиторий epel:

```
# yum install epel-release
```

Теперь устанавливаем сам lxc:

```
# yum install debootstrap lxc lxc-templates lxc-extra
```

Проверим готовность системы к работе lxc:

```
# lxc-checkconfig
```

Все должно быть enable, кроме двух строк:

```
newuidmap is not installed
newgidmap is not installed
```

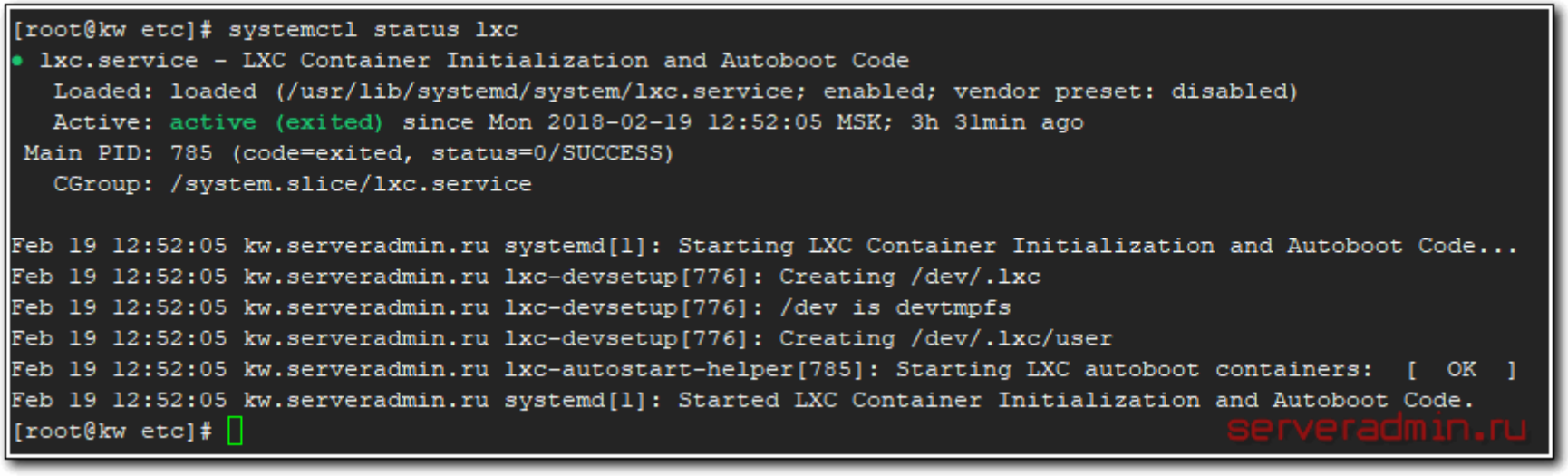
```
[root@kw etc]# lxc-checkconfig
Kernel configuration not found at /proc/config.gz; searching...
Kernel configuration found at /boot/config-3.10.0-693.17.1.el7.x86_64
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
User namespace: enabled
newuidmap is not installed
newgidmap is not installed
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
Bridges: enabled
Advanced netfilter: enabled
CONFIG_NF_NAT_IPV4: enabled
CONFIG_NF_NAT_IPV6: enabled
CONFIG_IP_NF_TARGET_MASQUERADE: enabled
CONFIG_IP6_NF_TARGET_MASQUERADE: enabled
CONFIG_NETFILTER_XT_TARGET_CHECKSUM: enabled

--- Checkpoint/Restore ---
checkpoint restore: enabled
CONFIG_FHANDLE: enabled
CONFIG_EVENTFD: enabled
CONFIG_EPOLL: enabled
```

Запускаем lxc и добавляем в автозагрузку:

```
# systemctl start lxc
# systemctl enable lxc
```

Проверяем:

```
# systemctl status lxc
```



Все в порядке, lxc сервис установлен и запущен. Переходим к созданию и настройке контейнеров.

Создание и настройка lxc контейнеров

Создадим новый контейнер с названием lxc_centos под управлением системы centos.

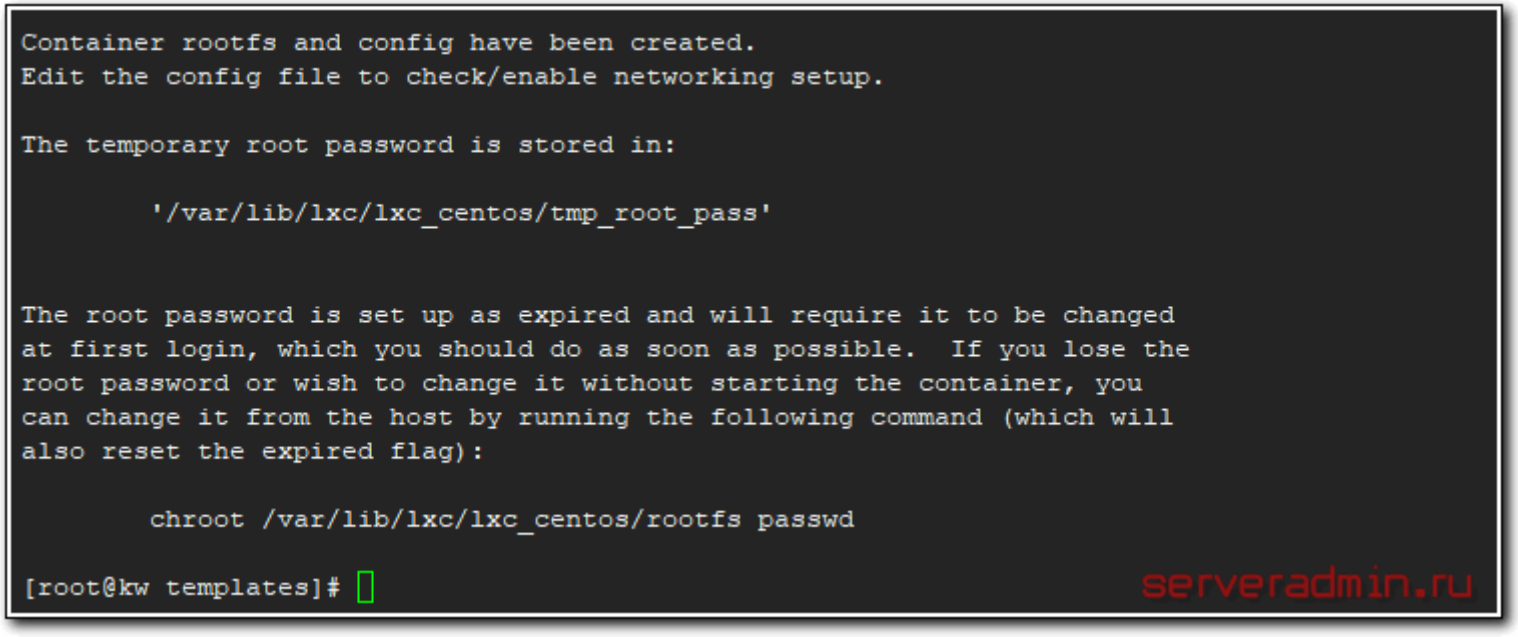
```
# lxc-create -n lxc_centos -t centos
```

После ключа -t указывается название шаблона. Список доступных шаблонов для установки можно посмотреть в */usr/share/lxc/templates/*

```
# ll /usr/share/lxc/templates
```

```
[root@kw templates]# ll /usr/share/lxc/templates
total 340
-rwxr-xr-x. 1 root root 10579 Oct 20 20:21 lxc-alpine
-rwxr-xr-x. 1 root root 13537 Oct 20 20:21 lxc-altlinux
-rwxr-xr-x. 1 root root 10839 Oct 20 20:21 lxc-archlinux
-rwxr-xr-x. 1 root root  9677 Oct 20 20:21 lxc-busybox
-rwxr-xr-x. 1 root root 29971 Oct 20 20:21 lxc-centos
-rwxr-xr-x. 1 root root 10486 Oct 20 20:21 lxc-cirros
-rwxr-xr-x. 1 root root 18342 Oct 20 20:21 lxc-debian
-rwxr-xr-x. 1 root root 18064 Oct 20 20:21 lxc-download
-rwxr-xr-x. 1 root root 49438 Oct 20 20:21 lxc-fedora
-rwxr-xr-x. 1 root root 28253 Oct 20 20:21 lxc-gentoo
-rwxr-xr-x. 1 root root 13965 Oct 20 20:21 lxc-openmandriva
-rwxr-xr-x. 1 root root 13882 Oct 20 20:21 lxc-opensuse
-rwxr-xr-x. 1 root root 35540 Oct 20 20:21 lxc-oracle
-rwxr-xr-x. 1 root root 12233 Oct 20 20:21 lxc-plamo
-rwxr-xr-x. 1 root root  6851 Oct 20 20:21 lxc-sshd
-rwxr-xr-x. 1 root root 24133 Oct 20 20:21 lxc-ubuntu
-rwxr-xr-x. 1 root root 11641 Oct 20 20:21 lxc-ubuntu-cloud
[root@kw templates]#
```


После установки контейнера, нам нужно указать свой root пароль к нему.



Для этого выполните в консоли команду и укажите новый пароль:

```
# chroot /var/lib/lxc/lxc_centos/rootfs passwd
```

Новый контейнер располагается по адресу */var/lib/lxc/lxc_centos*. В этой директории лежит файл config. Приводим его к следующему виду:

```
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = virbr0
lxc.network.hwaddr = fe:89:c3:04:aa:38
```

```
lxc.rootfs = /var/lib/lxc/lxc_centos/rootfs
lxc.network.name = eth0
lxc.network.ipv4 = 10.1.1.2/24
lxc.network.ipv4.gateway = 10.1.1.1
lxc.include = /usr/share/lxc/config/centos.common.conf
lxc.arch = x86_64
lxc.utsname = lxc_centos
lxc.autodev = 1
```

Зададим некоторые сетевые настройки в самом контейнере. Добавим dns серверы в */etc/resolv.conf*:

```
# mcedit /var/lib/lxc/lxc_centos/rootfs/etc/resolv.conf
```

```
nameserver 77.88.8.1
nameserver 8.8.4.4
```

Конфиг сетевого интерфейса приводим к следующему виду:

```
# mcedit /var/lib/lxc/lxc_centos/rootfs/etc/sysconfig/network-scripts/ifcfg-eth0
```

```
DEVICE=eth0
BOOTPROTO=none
ONBOOT=yes
HOSTNAME=lxc_centos
NM_CONTROLLED=no
```

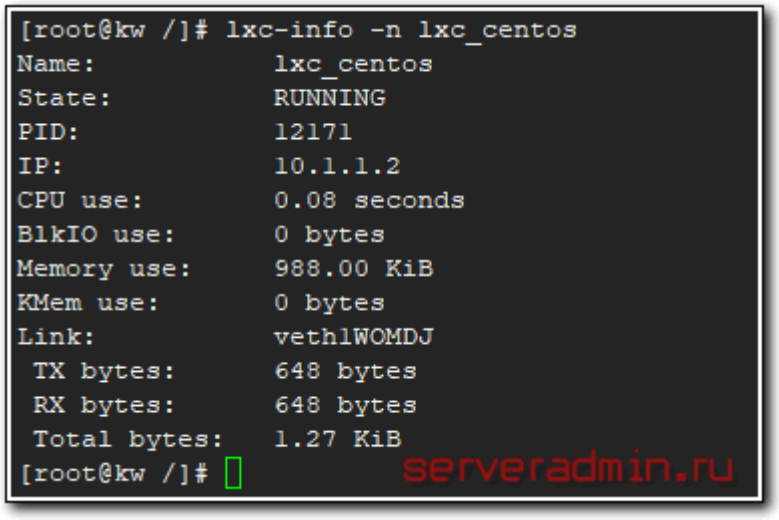
```
TYPE=Ethernet
```

Настройка lxc контейнера завершена. Запускаем его:

```
# lxc-start -n lxc_centos -d
```

Посмотрим состояние контейнера:

```
# lxc-info -n lxc_centos
```



Подключаемся к консоли контейнера:

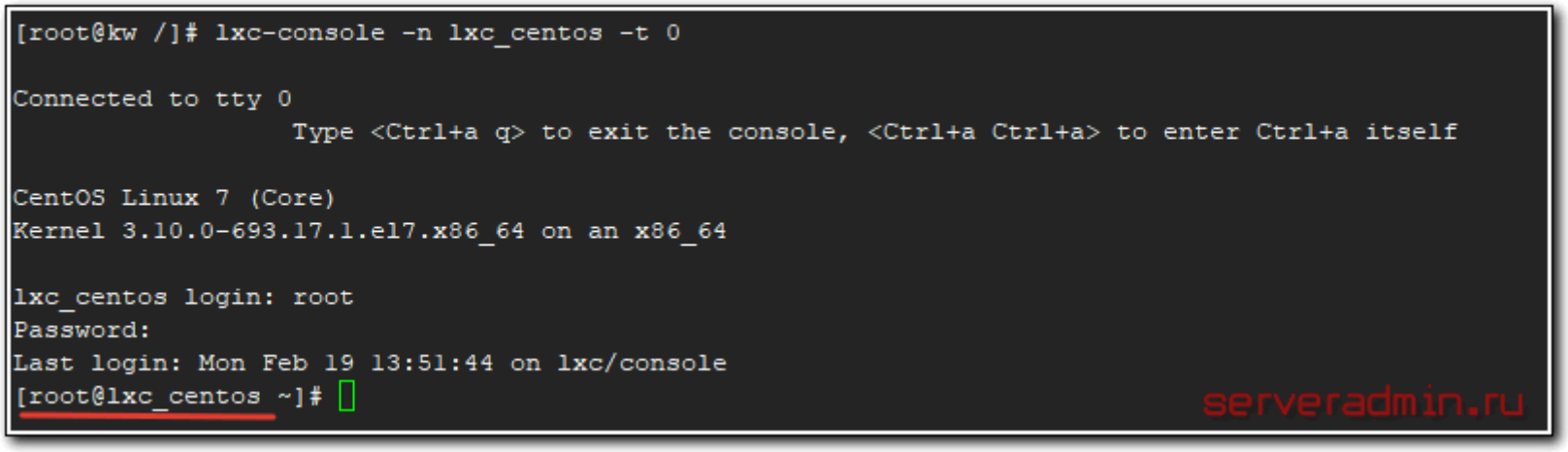
```
# lxc-console -n lxc_centos -t 0
```

Обращаю внимание на ключ -t 0. Если его не указать, то при подключении к консоли, вы будете пытаться подключиться к tty 1, на котором не будет никакого ответа. Вы увидите на экране:

```
Connected to tty 1
                Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter Ctrl+a itself
```

Ничего сделать больше не сможете, кроме как отключиться, нажав сначала Ctrl+a, затем отдельно q. Обращаю на это внимание, так как не очевидно, как нужно набирать эту комбинацию. Ключом -t мы указываем нулевую консоль и успешно к ней подключаемся. Для возврата к хостовой системе, нужно нажать Ctrl+a, затем отдельно q.

Если после подключения к консоли контейнера вы не видите экрана приветствия, нажмите Enter на клавиатуре.



На этом настройка lxc контейнера с centos 7 закончена. Проверяйте сеть, должно быть все в порядке. Чтобы подключиться по ssh к контейнеру, необходимо подключиться к порту 23543 хоста. При условии, что вы взяли мой пример настройки iptables в самом начале.

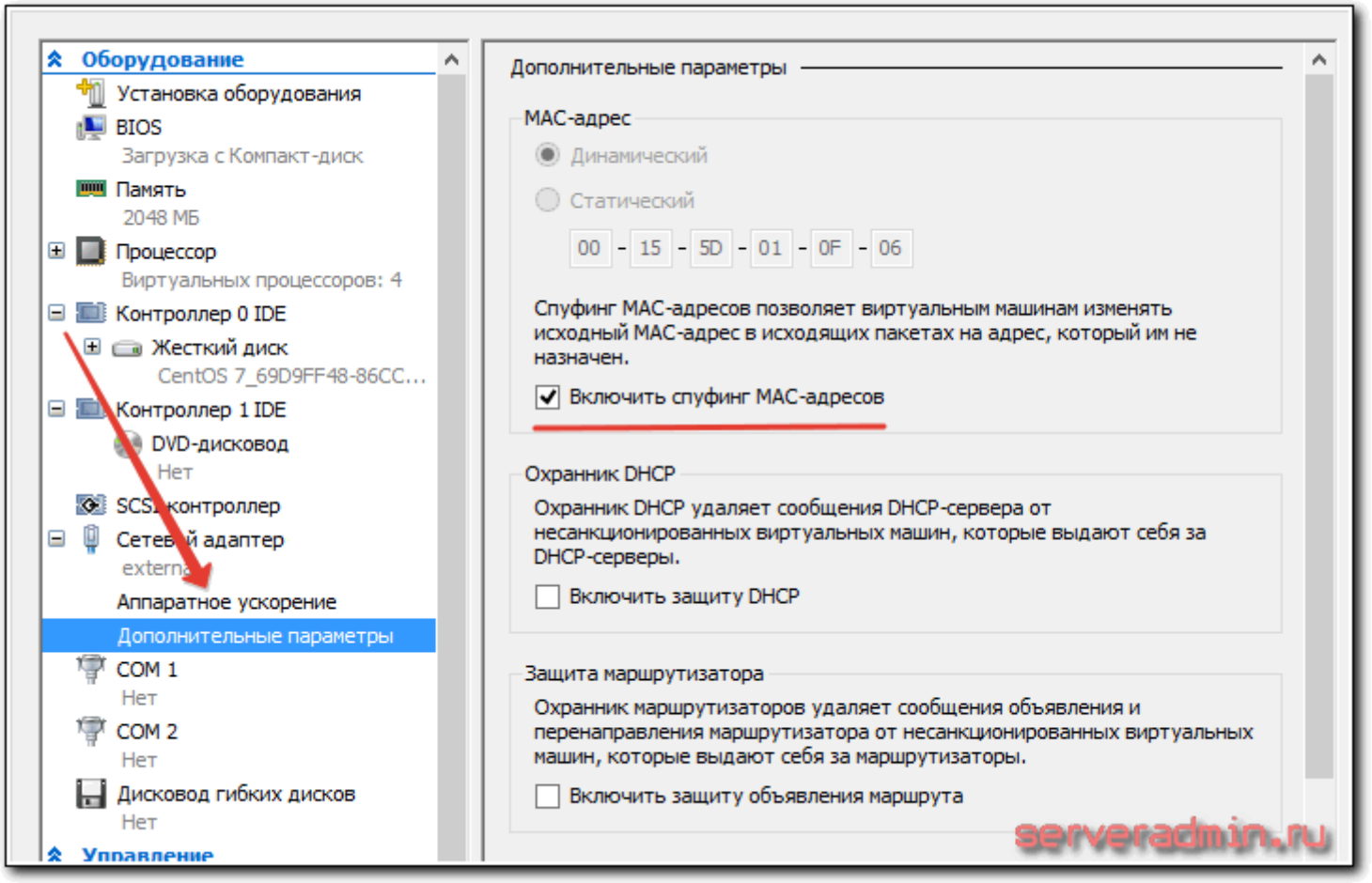
Выше я привел пример использования виртуального бриджа virbr0 для контейнера. Если вы используете бридж с физическим интерфейсом для прямого доступа контейнера во внешнюю сеть, настройки контейнера будут следующие:

```
lxc.rootfs = /var/lib/lxc/lxc_centos/rootfs
lxc.network.type = veth
lxc.network.flags = up
lxc.network.link = virbr1
lxc.network.hwaddr = fe:89:c3:04:aa:38
lxc.network.name = eth0
lxc.network.ipv4 = 192.168.13.44/24
lxc.network.ipv4.gateway = 192.168.13.1
lxc.network.veth.pair = veth-01
lxc.include = /usr/share/lxc/config/centos.common.conf
lxc.arch = x86_64
lxc.utsname = lxc_centos
lxc.autodev = 1
```

В данном случае 192.168.13.0/24 - внешняя сеть для хоста. В моем случае это локальная сеть тестового окружения. Если у вас будет реальный ip адрес на бридже, то контейнеру понадобится еще один реальный внешний ip адрес.

Обращаю внимание на очень важный нюанс, который мне стоил нескольких часов разбирательств. Для написания этой статьи я использовал виртуальную машину на hyper-v, у которой единственный сетевой интерфейс был бриджем во внешнюю сеть. Когда я пробрасывал контейнер через этот интерфейс с помощью virbr1, у меня ничего не работало. Контейнер видел только хост и ничего за его пределами. Я многократно проверял настройки, но никак не мог понять, почему не работает.

Оказалось, что по-умолчанию, гипервизор выпускает во внешнюю сеть только те устройства, mac адреса которых он знает. В данном случае это только мак адрес виртуальной машины. Контейнеры же на этой машине имеют другие мак адреса, которые гипервизору неизвестны, и он не выпускал их пакеты во внешнюю сеть. Конкретно в hyper-v такое поведение можно изменить в свойствах сетевого адаптера виртуальной машины:



После этого бридж во внешнюю сеть нормально заработал и контейнеры получили в нее доступ. Такое же поведение будет на виртуальных машинах с proxmox. По-умолчанию, контейнеры не получат доступа во внешнюю сеть. У меня не было под рукой доступа к настройкам гипервизора proxmox на той машине, где я проверил. Не стал подробно разбираться, но имейте ввиду этот момент. С vmware, кстати, будет то же самое. Контейнеры не выйдут во внешнюю сеть.

Дальше пойдет рассказ о проблемах, с которыми я столкнулся в процессе работы с lxc контейнерами.

Проблемы и ошибки

Не устанавливается httpd

Сразу скажу, что в качестве хостовой системы и шаблонов для lxc контейнеров я использовал только centos. Возможно в других системах указанных мной ошибок не будет. Первое с чем сразу же столкнулся было невозможность установить пакет httpd. Была вот такая ошибка:

```
# yum install httpd
```

```
Running transaction
  Installing : httpd-2.4.6-67.el7.centos.6.x86_64
1/1
Error unpacking rpm package httpd-2.4.6-67.el7.centos.6.x86_64
error: unpacking of archive failed on file /usr/sbin/suexec;5a8adbd2: cpio: cap_set_file
  Verifying  : httpd-2.4.6-67.el7.centos.6.x86_64
1/1

Failed:
  httpd.x86_64 0:2.4.6-67.el7.centos.6

Complete!
```

В интернете полно информации по подобной ошибке в контейнерах centos. Она встречается не только в lxc, но и в docker. В докере ее каким-то образом в определенный момент исправили, в lxc до сих пор воспроизводится и я не уверен, что исправление будет.

Суть ошибки в том, что существуют некие ограничения ядра для работы file capabilities. Я не вникал подробно в эти file capabilities, не понимаю до конца

сути ошибки, только поверхностно. Подробно ошибка разобрана тут - https://github.com/lxc/lxd/issues/1245. Так как решением проблемы предлагается перевести контейнер в privileged режим, когда хостовый рут и контейнерный имеют одинаковые системные id, то примерно понятно, в чем суть ошибки.

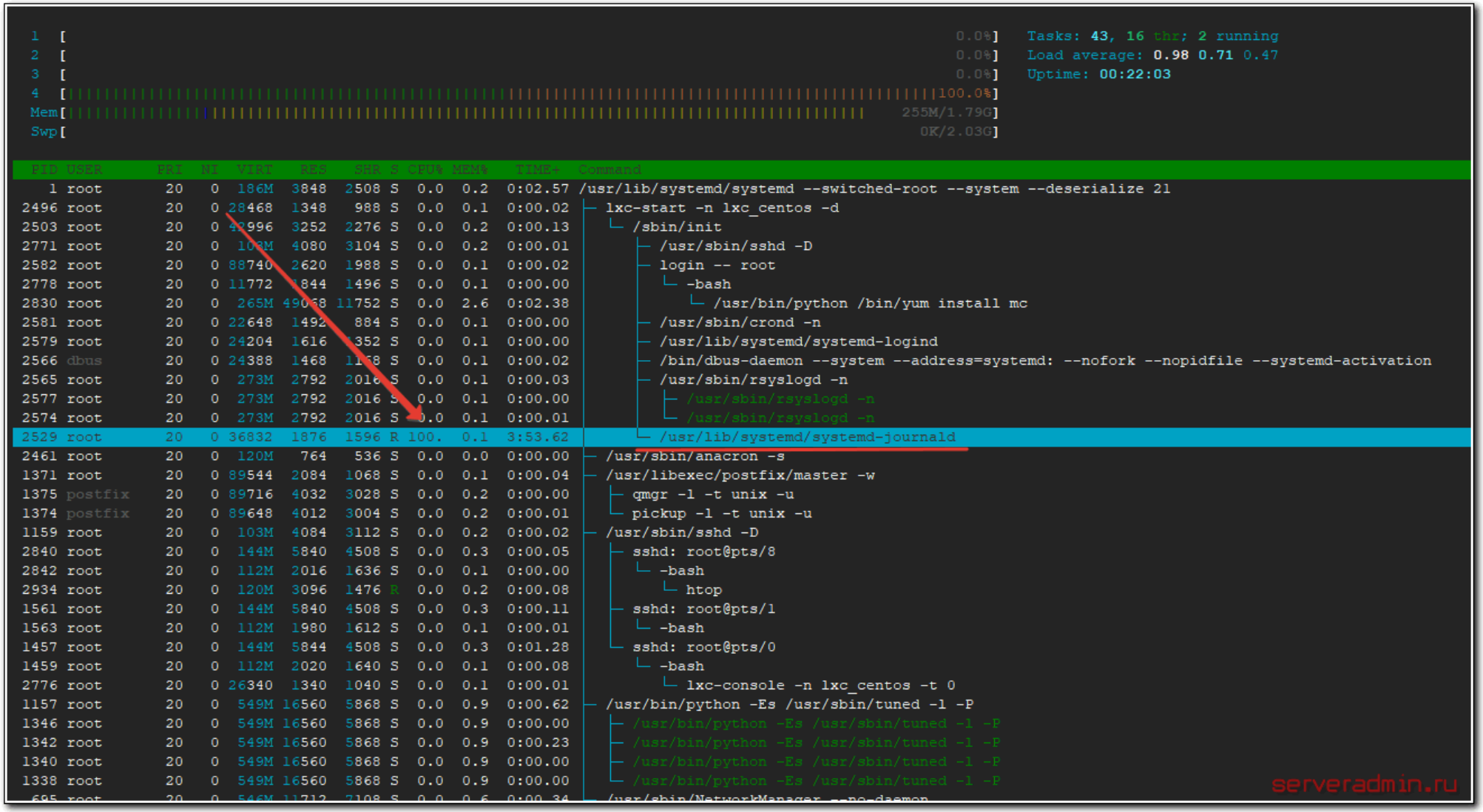
В общем, я не стал переводить контейнер в privileged режим, а поступил следующим образом. Зачрутился с хостовой машины в контейнер и установил httpd оттуда. Выполняем на хосте:

```
# chroot /var/lib/lxc/lxc_centos/rootfs
# yum install httpd
```

Теперь можно зайти в контейнер и проверить, что httpd установлен и нормально работает. Это рабочее решение, когда вы администратор и хоста и контейнеров. Но если вы кому-то отдаете контейнеры под управление, то либо вам придется самим решать ошибки владельцев контейнеров, либо искать какое-то другое решение.

Зависает контейнер и нагружает cpu хоста

Следующая неприятная ошибка, с которой столкнулся сразу же после начала тестирования lxc контейнеров. Контейнер через несколько минут после запуска зависал. Я не мог его ни остановить, ни удалить. При этом на самом хосте процесс /usr/lib/systemd/systemd-journald на 100% нагружал одно ядро cpu.


1 [0.0%]
2 [0.0%]
3 [0.0%]
4 [100.0%]
Mem[255M/1.79G]
Swp[0K/2.03G]
Tasks: 43, 16 thr; 2 running
Load average: 0.98 0.71 0.47
Uptime: 00:22:03
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
1 root 20 0 186M 3848 2508 S 0.0 0.2 0:02.57 /usr/lib/systemd/systemd --switched-root --system --deserialize 21
2496 root 20 0 28468 1348 988 S 0.0 0.1 0:00.02 ├─ lxc-start -n lxc_centos -d
2503 root 20 0 43996 3252 2276 S 0.0 0.2 0:00.13 │ └─ /sbin/init
2771 root 20 0 107M 4080 3104 S 0.0 0.2 0:00.01 │ ├─ /usr/sbin/sshd -D
2582 root 20 0 88740 2620 1988 S 0.0 0.1 0:00.02 │ ├─ login -- root
2778 root 20 0 11772 1844 1496 S 0.0 0.1 0:00.00 │ │ └─ -bash
2830 root 20 0 265M 49068 11752 S 0.0 2.6 0:02.38 │ │ └─ /usr/bin/python /bin/yum install mc
2581 root 20 0 22648 1492 884 S 0.0 0.1 0:00.00 │ ├─ /usr/sbin/crond -n
2579 root 20 0 24204 1616 1352 S 0.0 0.1 0:00.00 │ ├─ /usr/lib/systemd/systemd-logind
2566 dbus 20 0 24388 1468 1168 S 0.0 0.1 0:00.02 │ ├─ /bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
2565 root 20 0 273M 2792 2016 S 0.0 0.1 0:00.03 │ ├─ /usr/sbin/rsyslogd -n
2577 root 20 0 273M 2792 2016 S 0.0 0.1 0:00.00 │ │ ├─ /usr/sbin/rsyslogd -n
2574 root 20 0 273M 2792 2016 S 0.0 0.1 0:00.01 │ │ └─ /usr/sbin/rsyslogd -n
2529 root 20 0 36832 1876 1596 R 100. 0.1 3:53.62 │ └─ /usr/lib/systemd/systemd-journald
2461 root 20 0 120M 764 536 S 0.0 0.0 0:00.00 ├─ /usr/sbin/anacron -s
1371 root 20 0 89544 2084 1068 S 0.0 0.1 0:00.04 ├─ /usr/libexec/postfix/master -w
1375 postfix 20 0 89716 4032 3028 S 0.0 0.2 0:00.00 │ ├─ qmgr -l -t unix -u
1374 postfix 20 0 89648 4012 3004 S 0.0 0.2 0:00.01 │ └─ pickup -l -t unix -u
1159 root 20 0 103M 4084 3112 S 0.0 0.2 0:00.02 ├─ /usr/sbin/sshd -D
2840 root 20 0 144M 5840 4508 S 0.0 0.3 0:00.05 │ ├─ sshd: root@pts/8
2842 root 20 0 112M 2016 1636 S 0.0 0.1 0:00.00 │ │ └─ -bash
2934 root 20 0 120M 3096 1476 R 0.0 0.2 0:00.08 │ │ └─ htop
1561 root 20 0 144M 5840 4508 S 0.0 0.3 0:00.11 │ ├─ sshd: root@pts/1
1563 root 20 0 112M 1980 1612 S 0.0 0.1 0:00.01 │ │ └─ -bash
1457 root 20 0 144M 5844 4508 S 0.0 0.3 0:01.28 │ └─ sshd: root@pts/0
1459 root 20 0 112M 2020 1640 S 0.0 0.1 0:00.08 │ └─ -bash
2776 root 20 0 26340 1340 1040 S 0.0 0.1 0:00.01 │ └─ lxc-console -n lxc_centos -t 0
1157 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.62 ├─ /usr/bin/python -Es /usr/sbin/tuned -l -P
1346 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.00 │ ├─ /usr/bin/python -Es /usr/sbin/tuned -l -P
1342 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.23 │ ├─ /usr/bin/python -Es /usr/sbin/tuned -l -P
1340 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.00 │ ├─ /usr/bin/python -Es /usr/sbin/tuned -l -P
1338 root 20 0 549M 16560 5868 S 0.0 0.9 0:00.00 │ └─ /usr/bin/python -Es /usr/sbin/tuned -l -P
serveradmin.ru

Решение проблемы следующее. Добавляем в конфиг контейнера параметр:

```
lxc.kmsg = 0
```

Перезапускаем контейнер. Заходим в него и удаляем /dev/kmsg (именно в контейнере, не на хосте!!!)

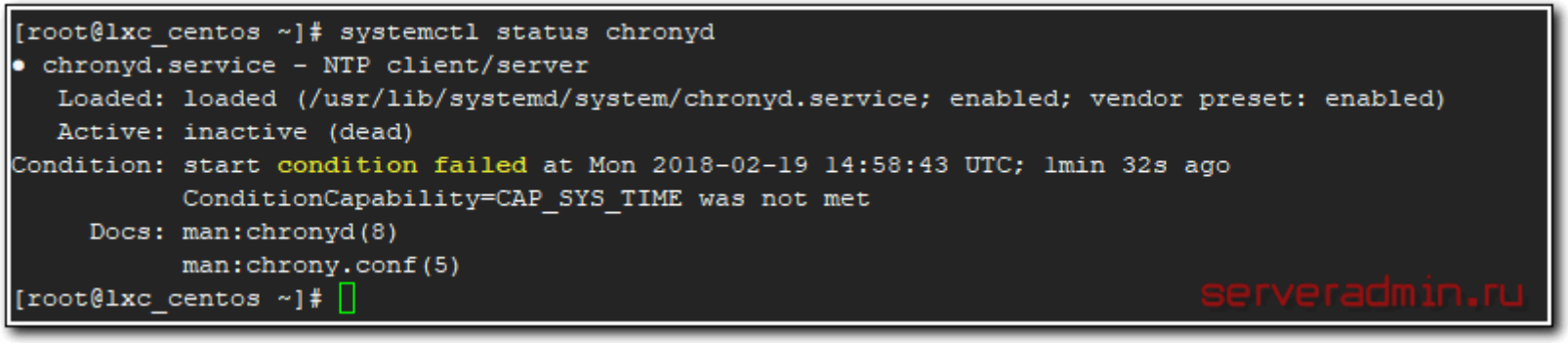
```
# rm -f /dev/kmsg
```

После этого контейнеры стали работать стабильно и перестали зависать. Я установил bitrix-env и развернул сайт. Все заработало без проблем с нормальной скоростью.

Не работает chronyd

После установки и запуска chronyd в lxc контейнере с centos 7 получаем ошибку:

```
ConditionCapability=CAP_SYS_TIME was not met
```



Тут я уже немного утомился ковыряться в ошибках lxc и понял, что не хочу использовать в работе эти контейнеры. Но все же собрался с силами и погуглил еще немного. Как оказалось, это не ошибка, это ограничение работы в контейнере. Условием работы chronyd является доступ к системному вызову adjtimex(). У контейнера в не privileged режиме нет этого доступа, поэтому он и не запускается.

Контролирует эту ситуацию параметр

```
ConditionCapability=CAP_SYS_TIME
```

в конфиге systemd службы chronyd в контейнере - */etc/systemd/system/multi-user.target.wants/chronyd.service*. Если убрать этот параметр и запустить службу, получим ошибку:

```
adjtimex(0x8001) failed : Operation not permitted
```

В общем, контейнер без привилегированного режима управлять временем не может. Это архитектурная особенность работы контейнеров, с этим ничего не поделать. Надо на хосте следить за временем.

Заключение

> Не понравилась статья и хочешь научить меня администрировать? Пожалуйста, я люблю учиться. Комментарии в твоем распоряжении. Расскажи, как сделать правильно!

Я сделал небольшой обзор возможностей lxc контейнеров. Не затронул такие вопросы как ограничение ресурсов, бэкап и восстановление контейнеров. Я еще не тестировал это, поэтому не хочется писать о том, что сам еще не пробовал, а зафиксировать уже полученные знания хочется.

В целом, вывод по lxc неоднозначный. С одной стороны, вроде все удобно и даже работает. Но с дугой стороны, такие нерешенные проблемы, как зависание контейнеров мне не понятны. Почему оно не заработало из коробки - не ясно. Кроме того, периодически я получал зависания таких команд как lxc-info. Хочешь получить инфу по контейнеру, а в ответ тишина. Просто висит команда в консоли и не выводит ничего. Оставлял на час - без изменений.

После перезапуска контейнера показывает нормально. Такие явные ошибки очень настораживают и предостерегают от использования в продакшене.

Потестирую дальше на своих небольших проектах. Если все будет нормально, то дополню статью новым материалом. В принципе, если дальше будет работать стабильно, можно кое-где использовать. В целом, контейнеры очень интересная технология, которая будет отличным дополнением виртуальным машинам, особенно если весь хост используется для однотипной нагрузки, к примеру, под web сайты. Можно их безопасно изолировать с минимальным оверхедом по ресурсам, чего не скажешь о виртуальных машинах даже в минимальной конфигурации.

Если у вас есть опыт работы с lxc или какими-то другими контейнерами, но не докером, я с ним сам много работал и в целом знаком, прошу поделиться в комментариях.