



ОНЛАЙН-ОБРАЗОВАНИЕ

# Основы контейнеризации в Linux

**namespaces & cgroups**

**Александр Румянцев**



# Для чего нужна контейнеризация?



# Для чего нужна контейнеризация?

- Разделение ресурсов сервера
- Песочница для критичных сервисов
- Запуск нескольких одинаковых сервисов при унификации управления
- Разделение доступа и предоставление полного окружения разным непривилегированным пользователям



# Первые механизмы. Шаг 1.

**chroot** - подмена корня файловой системы для группы процессов

Появился в 1979 году в AT&T Version 7 Unix, а в 1982 - в 4.2BSD

**Недостатки:**

- пространство процессов общее
- сеть общая
- отсутствие ограничения ресурсов

Фактически служит для целей изоляции процессов (например, до сих пор `bind` опционально штатно во многих дистрибутивах запускается в `chroot`, многие демоны перед понижением привилегий делают `chroot` в пустую директорию) или пользователей (`chroot` используется, например в демоне `ftp`, ограничивая анонимного пользователя)

Разработчики и пользователи IBM S/390 смотрят с насмешкой



# Первые механизмы. Шаг 2.

**jail - первые контейнеры в FreeBSD**  
**Появились в 1999-2000 годах в FreeBSD 4.0**

**Достоинства:**

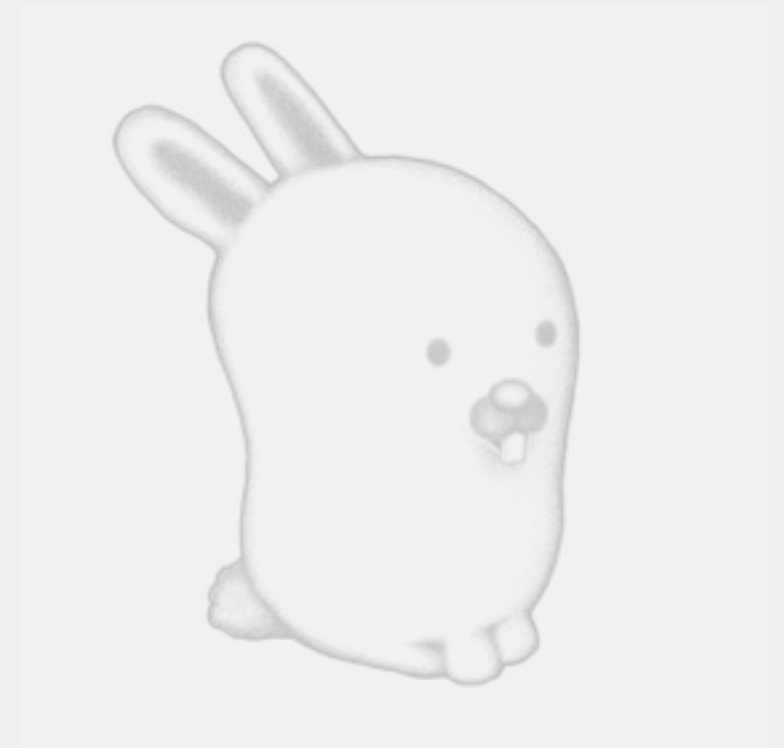
- **изоляция сети (но нет изоляции loopback)**
- **некоторая изоляция процессов**

**Недостатки:**

- **отсутствие ограничения ресурсов**

**openvz - первое появление в 2000 году, до прода дошло уже в 2.6, примерно в 2002 году, сложная судьба из-за неудачных попыток коммерциализации**

**Разработчики и пользователи IBM S/390 смотрят с удивлением**  
**Разработчики** (они же пользователи) **Plan9 смотрят с грустью**



# Time to pivot(). Namespaces.

Все механизмы обладали одним недостатком: это были костыли и палки, проросшие глубоко в ядро. В какой-то момент это осознали и в 2002 году, в ядре 2.4.19 появился Namespaces API, состоящий из трех системных функций:

- `clone (2)` - аналог `fork()`
- `setns (2)` - подключение к namespace существующего процесса
- `unshare (2)` - изменение контекста текущего приложения

которые доступны пользователю с привилегиями `CAP_SYS_ADMIN`, или, проще говоря, `root`.

**Namespace** - новый атрибут процесса



# Namespace

**Абстракция над общими ресурсами, позволяющая разным процессам иметь разное представление о тех глобальных ресурсах, которыми они распоряжаются**

**На данный момент существуют следующие пространства имён  
(`man 7 namespaces`):**

**Cgroup** - используется как атрибут, корневой узел дерева cgroup  
**IPC** - IPC

**Network** - сетевой стек: интерфейсы, таблицы маршрутизации, etc

**Mount** - (**CLONE\_NEWNS**, `xe-xe`) - аналог chroot

**PID** - пространство номеров процессов

**User** - пространство пользователей

**UTS** - изолированный hostname и NIS domain name





# Утилиты для работы с namespaces

- **unshare (1)** - запуск процесса в новом namespace
- **nsenter (1)** - подключение к namespace существующего процесса
- **/proc/<pid>/ns/**



## Mount namespace

Можно разделять общие каталоги, классический пример в до-systemd эпоху - изоляция /tmp (в systemd уже есть параметр PrivateTmp)

Chroot делается в mount namespace через `mount -o bind` или `pivot_root`



# Network Namespace

Можно уже делать свои контейнеры. Но лучше - использовать разные таблицы маршрутизации для разных процессов.

```
root@host:/# ip link add veth0 type veth peer name veth1
root@host:/# ip link set veth1 netns netns1
root@host:/# ip addr add 172.16.99.1/24 dev veth0
root@host:/# ip link set veth0 up
root@host:/# ip netns exec netns1 ip addr add 172.16.99.100/24 dev veth1
root@host:/# ip netns exec netns1 ip link set lo up
root@host:/# ip netns exec netns1 ip link set veth1 up
root@host:/# ip netns exec netns1 ip route add default via 172.16.99.1
```



## User & PID namespaces

Уже появились сильно позже конкретно для контейнеров

User Namespace, позволяет иметь к каждому контейнеру своих системных пользователей, включая рута. А через механизм subuid (5) можно сопоставить непривилегированных пользователей хост системы с root контейнера.

Pid Namespace - тот же смысл, позволяет в каждом контейнере иметь свою систему инициализации с PID=1, а так же полностью изолированные процессы (процессы, не входящие в namespace не "видны")

/etc/subuid и /etc/subgid - конфигурация для выделения отдельного пространства идентификаторов



**Ну всё, можно делить машину на контейнеры. Oh wa...**

**Но тут контейнеры начинают мечтать обо всех доступных ресурсах системы**



# Вначале было слово

**Словом было - process accounting, появившийся давно (1999 год), но сильно расширенный в ядре 2.6.20 (2007 год) параллельно с появлением KVM (совпадение? не думаю!) - подсистема, позволившая вести учёт ресурсов по каждому процессу, которой мы пользуемся в atop/sar со своей админской стороны. Ну а основы социализма нам говорят, что кроме учета должен быть и контроль.**

**Примерно тогда же в 2006 году внутри Google началась разработка process containers с усовершенствования механизма cgroup, который позволял ограничить процессы по времени исполнения.**



# control groups

**Подсистема ядра, позволяющая создавать иерархические группы процессов, каждая из которых обладает своим набором счётчиков.**

**В корне иерархии (дерева) - набор контроллеров для учёта и контроля групп.**

**Процесс может принадлежать разным группам под разными контроллерами, но не может быть в разных ветвях одного контроллера (при этом он будет в группе верхней иерархии)**

**systemd имеет свою ветку, не имеющую контроллера задачей которого является просто группировка процессов. Главный процесс заносится в новую группу, а потомки (см слайд "порождение процессов") появляются в той-же группе. Это позволяет управлять сервисом без знания PID главного процесса и не оставляя неприкаянных демонов. При завершении всех процессов в группе systemd об этом уведомляется.**



# Контроллеры

в CentOS 7 мы имеем:

**blkio:** управление доступной полосой при доступе к блочным устройствам

**cpu:** управление доступом к ресурсам процессора

**cpuacct:** аккаунтинг cpu; используется совместно с контроллером cpu

**cpuset:** выделение отдельных процессоров группе

**devices:** ограничение доступа к устройствам

**freezer:** заморозка процессов (спасибо команде OpenVZ)

**memory:** ограничение памяти для группы

**net\_cls:** шейпинг (man 8 tc)

**perf\_event:** интерфейс для perf

**hugetlb:** ограничение работы с huge pages

**pid:** ограничение числа процессов





# Интерфейс

**В отличие от namespaces, cgroups имеет простой интерфейс для управления. Он реализован в виде псевдо-FS cgroup**

**Для того, что бы начать использовать cgroups, нужно смонтировать вершину иерархии в любой каталог, в качестве опции передав имена контроллеров, которые будут обслуживать иерархию.**

**В CentOS 7 все возможные контроллеры смонтированы под `/sys/fs/cgroup/`**



# Основные файлы

`/proc/<pid>/cgroup` (read only) — список групп к которым принадлежит процесс

Иерархия `/sys/fs/cgroup/*`

`cgroup.clone_children` — позволяет передавать дочерним контрольным группам свойства родительских

`tasks` — содержит список PID всех процессов, включённых в контрольные группы

`cgroup.procs` — содержит список TGID групп процессов, включённых в контрольные группы

`cgroup.event_control` — позволяет отправлять уведомления в случае изменения статуса контрольной группы

`release_agent` — содержится команда, которая будет выполнена, если включена опция `notify_on_release`. Может использоваться, например, для автоматического удаления пустых контрольных групп

`notify_on_release` — содержит булеву переменную (0 или 1), включающую (или наоборот отключающую), выполнение команду, указанной в `release_agent`.



# Недостатки

Разные контроллеры разрабатывались разными группами людей, в результате:

- мы имеем несколько иерархий
- в одних и тех же подсистемах ядра присутствуют части разных контроллеров
- разные контроллеры по-разному понимают наследование групп

Отдельный анекдот: процесс, ограниченный по памяти продолжает видеть в `/proc/meminfo` всю доступную память, что иногда приводит к казусам. Решается с помощью псевдо-ФС `lxcfs` из комплекта LXC



## До основания, а затем

С 2015 года разрабатывается cgroups v2, (присутствует в ядре начиная с версии 4.5) которая представляет единую иерархию, в которой контроллеры на лету добавляются в группу.

~~Но пока там всего три контроллера: blkio, memory и PID, а также начальная поддержка cpuset~~

~~Ждем, надеемся, верим.~~

Начиная с Linux 5.2 (07.07.2019) в cgroups v2 добавили контроллер устройств



# Зачем это всё

**Все системы (lxc, docker, rkt, много их нынче развелось) контейнеризации используют только описанные механизмы.**

**Не все системы контейнеризации предоставляют нужные функции, особенно мониторинга, динамического ограничения ресурсов etc.**

**В любой контейнер вы можете зайти с помощью nsenter вне зависимости от используемой системы.**

**Любой контейнер вы можете дополнительно промониторить или ограничить вне зависимости от используемой системы.**



# Бонус

## Контейнеризация из коробки в три шага: systemd-nspawn

### 1. Готовим образ

```
yum --installroot=/var/lib/machines/test --releasever=7 -y install centos-release systemd passwd yum iproute  
:> /var/lib/machines/test/etc/sysconfig/network  
echo -e 'pts/0\npts/1\npts/2\n' >> /var/lib/machines/test/etc/securetty  
chroot /var/lib/machines/test passwd
```

### 2. Включаем сервис

```
machinectl enable test
```

### 3. Запускаем машину

```
machinectl start test
```

### 4. Логинимся

```
machinectl login test
```





**Спасибо  
за внимание!**

**Вопросы?**