GeekBrains

Механизмы контрольных групп

Контейнеризация

Урок 2

Оглавление

Введение

Всем привет! На этом уроке мы познакомимся с механизмами контрольных групп: узнаем, что это такое и для чего используется и изучим составляющие, посредством которых происходит ограничение ресурсов.

На этом уроке

1. cgroups – появление механизма процесс модификации
2. Архитектура и составляющие механизма
3. Примеры управления группами
4. Недостатки cgroups

Вступление

В продолжение темы рассмотрения механизмов контейнеризации, предлагается рассмотреть вторую довольно важную составляющую - механизмы контрольных групп. Для полноценного запуска контейнеризированного приложения недостаточно лишь изоляции ресурсов. В случае запуска приложения в изолированном окружении, нужно быть уверенным, что приложение получит достаточное количество ресурсов, что приложение будет потреблять только необходимые ресурсы и ничего более. Это необходимо для сохранения стабильной работы как самого приложения, так и хостовой системы в целом.

Давайте немного поясню на примере. Предположим, что у нас есть сайт, который состоит из визуальной части, то есть web-интерфейса и базы данных (не важно какой). У каждого приложения есть свои требования к системе: программный и аппаратный. Программные требования мы пока опустим, а вот аппаратные обсудим.

Предположим, что для корректной работы сайта необходим двухъядерный процессор с тактовой частотой 1 Гц и минимум 1 Гб ОЗУ. Если сайт будет работать на голом железе - никаких проблем. Берем простой компьютер и настраиваем. В случае же, если мы работаем с контейнерами, необходима не только изоляция ресурсов, но и гарантированная выдача аппаратных ресурсов. Просто запустив приложение изолированно и, параллельно запуская другие, может случиться так, что нашим компонентам сайта попросту не хватит аппаратных ресурсов, что приведет к замедлению работы в лучшем случае, в худшем - к полной неработоспособности нашего сайта.

Для решения этой задачи в ядре ОС Linux уже встроен специальный механизм: механизм контрольных групп (cgroups (control groups)).

Историческая справка

Разработка данного механизма была начата инженерами из Google в 2006 году и изначально имела иное название - **контейнеры процессов**, а уже после, в 2007 году, был переименован в **механизм контрольных групп**. Связано это было с тем, что мировое сообщество двояко расшифровывало термин **контейнер** в изначальном определении.

Изначальное назначение утилиты было довольно простым: усовершенствование имеющегося механизма **cpuset**, который отвечал за распределение процессорного времени и оперативной памяти между всеми

запущенными процессами и задачами. Однако, со временем проект перерос себя и был переориентирован.

Данная технология начала встраиваться в официальные версии ядра несколько позже (в 2008 году), начиная с версии 2.6.24, когда была доработана и протестирована. С момента добавления в ядро, разработка значительно ускорилась и было добавлено много дополнительных возможностей, а механизм начал использоваться в технологии инициализации **systemd**, являясь при этом ключевым элементом в реализации системы виртуализации на уровне операционной системы **LXC**.

LXC-подсистема контейнеризации

Прежде чем мы продолжим, стоит кратко рассказать о системе контейнеризации, которая встраивалась в систему еще с 2008 года.

Данная подсистема позволяет запускать несколько изолированных друг от друга экземпляров ОС Linux на одном узле. Важно отметить, что данная система не использует технологию виртуализации, то есть не создает виртуальных машин. Система создает виртуальное окружение с собственным пространством процессов и сетевым стеком.

Возможности и использование

Основной целью механизма является предоставление единого программного интерфейса к ряду средств управления процессами: начиная с контроля просто единичного управления процессом, заканчивая полной виртуализацией на уровне системы (пример - LXC).

Механизм предоставляет ряд возможностей:

- Ограничение ресурсов (как виртуальных, так и физических) - использование памяти и процессорного времени.
- Приоритизация. Разным группам приложений можно выделить различное количество ресурсов процессора и, например, пропускной способности ввода-вывода (дисковой подсистемы).
- Регистрация затрат тех или иных ресурсов приложением либо группой приложений.

- Изоляция. Предполагается, что приложения используют распределенное пространство имен таким образом, что процессы одной группы недоступны другой. Та же участь касается и сетевой инфраструктуры и прочих ресурсов.
- Управление. Возможность перезагрузки процессов, создания дополнительных квот и ограничений.

Механизм состоит из двух главных частей: ядра (cgroup core) и подсистем. Список подсистем зависит от версии ядра. Основные компоненты следующие:

- blkio — подсистема устанавливает квоты на чтение и запись с блочных устройств.
- cpuacct — формирует отчеты об использовании ресурсов процессора.
- cpu — предоставляет доступ процессам в рамках контрольной группы к ресурсам процессорной подсистемы.
- cpuset — распределяет задачи в рамках контрольной группы между имеющимися процессорными ядрами.
- devices — предоставляет доступ или же, наоборот, блокирует доступ к устройствам.
- freezer — приостанавливает и возобновляет выполнение задач в рамках контрольной группы.
- memory — занимается выделением памяти для групп процессов.
- net_cls — помечает сетевые пакеты специальным тэгом, который в дальнейшем позволяет идентифицировать пакеты, порождаемые определённой задачей в рамках контрольной группы.
- netprio — используется для динамической установки приоритетов по трафику.
- ns - используется для группировки процессов в отдельное пространство имен, где процессы могут взаимодействовать между собой и при этом будут изолированы от внешних процессов.
- pids — используется для ограничения количества процессов в рамках контрольной группы.
- unified — автоматически монтирует файловую систему в каталог /sys/fs/cgroup/unified при запуске системы.

Подсистемы представляют собой модули ядра, каждый из которых отвечает за выделение системных ресурсов контрольным группам. Разумеется, можно по отдельности запрограммировать подсистемы с целью создания индивидуального подхода к управлению группами процессов. Интерфейс программирования задокументирован в файле:

```
    https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.tx
t
```

Объекты состояния могут содержать параметры подсистем для каждой контрольной группы и представляются в виде псевдофайлов в виртуальной файловой системе. О них мы поговорим позднее.

Каждая подсистема представляет собой директорию с набором управляющих файлов, в которых и прописываются все настройки.

Помимо специализированных файлов, каждая директория содержит в себе набор управляющих файлов:

- 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 (Thread Group Id) 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.

Перед рассмотрением каждой подсистемы в отдельности, введем определение псевдофайла.

Псевдофайл - файл, который не представляет из себя файл в базовой файловой системе на диске. Это файлы, которые автоматически создаются для

представления какого-либо другого объекта в виде файла с целью возможности взаимодействия с ним. Простой пример **/dev/sda**.

Теперь же давайте рассмотрим каждую подсистему в отдельности.

blkio (block I/O)

blkio - это подсистема управления процедурами ввода/вывода блочных устройств. Для ограничения доступа приложению или процессу записываются значения в псевдофайлы. Подсистема содержит огромное количество параметров. Предлагается рассмотреть наиболее важные:

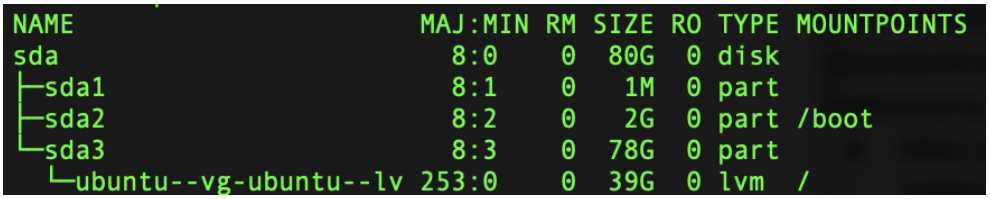
- blkio.weight - позволяет определить относительный вес (в значениях от 100 до 1000) ввода-вывода контрольной группы. В данном случае, чем больше значение, тем больше ресурсов получит приложение или процесс. Аналогию можно провести довольно простую: при значении **100** приложение получит возможность производить 100 операций чтения/записи в секунду. Можно определить дополнительное значение для отдельных устройств в отдельный псевдофайл: blkio.weight_device. Пример определения параметра в псевдофайл выглядит следующим образом:

```
echo 200 > blkio.weight
```

- blkio.weight_device - определяет относительный вес для конкретного устройства, которое доступно в файловой системе. Этот параметр позволяет переопределить общее значение blkio.weight. Переопределение выглядит следующим образом:

```
echo 8:0 200 > blkio.weight_device
```

Вывод такой в связи с тем, что устройство /dev/sda соответствует в списке устройств номеру 8:0. Проверить это можно с помощью команды **lsblk:**



Как можно видеть, значения, вводимые нами, совпадают со значениями, выводимыми системой.

cpu

Эта подсистема отвечает за управление доступом контрольных групп к процессорам системы. Доступ регламентируется параметрами, которые так же, как и в предыдущем случае, записываются в псевдофайлы. Принцип такой же: один параметр - один псевдофайл. Рассмотрим наиболее значимые:

- cpu.shares - целочисленный параметр, которые определяет относительную величину доступного процессорного времени. Например: есть две контрольные группы. У одной из них это значение равно единице, у второй - 2. Это значит, что процессы из второй контрольной группы будут получать вдвое больше процессорного времени при выполнении той или иной задачи.

- cpu_rt_runtime_us - этим параметром определяется максимальный период времени (в микросекундах), в течение которого задания того или иного процесса могут использовать процессорные ресурсы. Данный параметр очень важен, так как это ограничение позволяет предотвратить монопольное использование ресурсов одной подгруппой.

- cpu._rt_period_us - здесь же определяется интерфал, по истечении которого приложения из контрольной группы получат доступ к процессорным ресурсам.

  Пример довольно прост: если нам необходимо, чтобы процессы из группы имели доступ к процессору длиной в 5 секунд каждые 10 секунд, необходимо задать следующие значения:

```
cpu.rt_runtime_us 4000000

cpu.rt_period_us 10000000
```

cpuacct

Данная подсистема создает отчеты о занятости ресурсов процессора. Всего имеется три вида отчетов:

- cpuacct.stat - этот отчет возвращает число циклов процессора (величина измерения - USER_HZ), которые были затрачены на обработку заданий контрольной группы. Учитывается пользовательский и системный режим.

- cpuacct.usage - возвращает суммарное время (единица измерения - наносекунды), в течение которого ресурсы процессора были заняты обработкой заданий контрольной группы.

- cpuacct.usage_percpu - как и прошлый отчет, возвращает время, в течение которого ресурсы были заняты обработкой всех заданий контрольной группы, но попроцессорно.

cpuset

Подсистема, отвечающая за выделение ресурсов процессора контрольным группам. Также, как и ранее, каждый параметр хранится в отдельном псевдофайле. Например:

- cpuset.cpus - параметр, определяющий количество процессоров, к которым могут обращаться процессы в контрольной группе. Записывать в файл можно как диапазон используемых процессоров (0-2), так и какие-то отдельные через запятую. Пример: если в файле записано какое-либо число (для простоты - 0), то приложение будет иметь доступ только к этому процессору. Остальные не будут задействованы для этой задачи. Важно отметить, что этот же процессор может быть задействован для решения и других задач. Если мы хотим, чтобы он был задействован только для решения нашей задачи, используйте следующий параметр.
- cpuset.cpu_exclusive - этим параметром можно задать, возможность совместного использования процессоров, которые были перечислены ранее.

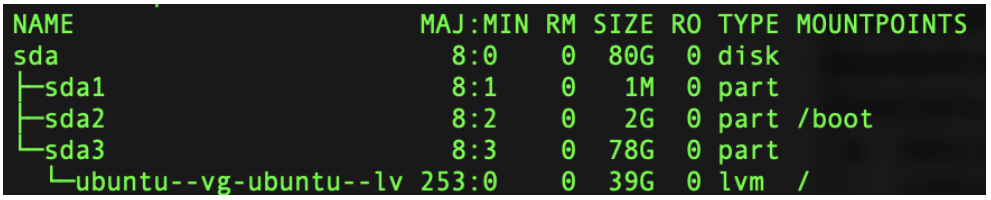
devices

Эта подсистема отвечает за управление доступом к устройствам. Она включена в ОС Linux относительно недавно в отличие от ранее рассмотренных и имеет наименьшее число возможных параметров:

- devices.allow - в этот псевдофайл записываются устройства, к которым разрешен доступ в рамках контрольной группы. Каждая запись содержит по четыре поля: **тип устройства, старший номер, младший номер, режим доступа**. На первый взгляд кажется, что все сложно, но нет. Давайте рассмотрим каждое поле с примерами:
    - тип устройства. Здесь могут быть следующие значения:
        - a - применяется ко всем символьным и блочным устройствам
        - b - блочное устройство
        - c - символьное устройство (ссылка на устройство, на файл)

- старший и младший номера - они разделяются двоеточием и идентифицируют конечное устройство в ОС Linux. Например, значение 8:1 обозначает следующее:
    - 8 - старший номер, обозначающий диски SCSi
    - 1 - младший номер, обозначающий первый раздел на первом диске

  Явно этот пример мы уже видели ранее, но сейчас можно освежить память:



  - режим доступа - определяется доступ к устройству, которое будет иметь контрольная группа, Доступны следующие варианты:
    - r - доступ разрешен только на чтение
    - w - доступ разрешен на чтение и запись
    - m - разрешение доступа на создание файлов, если они не существуют

- devices.deny - полная противоположность предыдущей опции. Запрещает доступ к устройствам. Формат записи идентичный предыдущему.
- devices.list - в этом псевдофайле будут устройства, для которых было настроено управление доступом.

freezer

Из названия может быть понятно, что эта подсистема отвечает за остановку и возобновление заданий контрольной группы. У нее есть всего один псведофайл:

- freezer.state - статус подсистемы. Она имеет несколько допустимых значений:
  - frozen - задания приостановлены
  - freezing - система в стадии приостановки задач
  - thawed - возобновление работы заданий в группе

Важное замечание: в файл можно записать лишь два значения: **frozen** и **thawed**. Значение **freezing** нельзя записать и его система сама записывает в файл. Это состояние - переходное для процесса.

memory

Подсистема создает отчеты об использовании ресурсов оперативной памяти. Также позволяет наложить ограничения с помощью ряда параметров. Рассмотрим несколько:

- memory.stat - возвращает статистику использования памяти.
- memory.usage_in_bytes - отображается суммарный размер памяти, которая занята процессами контрольной группы.
- memory.max_usage_in_bytes - максимальный размер памяти, доступный процессам контрольной группы.
- memory.limit_in_bytes - здесь задается максимальный размер памяти, включая файл подкачки.
- memory.failcnt - этот файл является счетчиком случаев достижения лимита, который задается в memory.limit_in_bytes.

net_cls

Эта подсистема присваивает сетевым пакетам идентификатор, который помогает контроллеру идентифицировать пакеты, поступающие из контрольной группы. Также можно изменить настройки так, чтобы пакетам из различных групп назначался разный приоритет.

Единственный параметр **net_cls.classid** представляет из себя шестнадцатиричное значение, которое идентифицирует обработчик трафика.

Практическое знакомство: проверка теории

Итак, изучив теорию, давайте перейдем к практике! Для начала давайте создадим новую директорию в папке:

```
mkdir /sys/fs/cgroup/cpuset/testgroup1
```

А дальше давайте просмотрим список файлов и каталогов в папке. Вывод будет довольно интересный:

```
# ll /sys/fs/cgroup/cpuset/testgroup1/
total 0
drwxr-xr-x 2 root root 0 Oct 27 08:51 ./
drwxr-xr-x 3 root root 0 Oct 27 08:50 ../
-r--r--r-- 1 root root 0 Oct 27 08:51 cgroup.controllers
-r--r--r-- 1 root root 0 Oct 27 08:51 cgroup.events
-rw-r--r-- 1 root root 0 Oct 27 08:51 cgroup.freeze
--w------- 1 root root 0 Oct 27 08:51 cgroup.kill
-rw-r--r-- 1 root root 0 Oct 27 08:51 cgroup.max.depth
-rw-r--r-- 1 root root 0 Oct 27 08:51 cgroup.max.descendants
-rw-r--r-- 1 root root 0 Oct 27 08:51 cgroup.procs
-r--r--r-- 1 root root 0 Oct 27 08:51 cgroup.stat
-rw-r--r-- 1 root root 0 Oct 27 08:51 cgroup.subtree_control
-rw-r--r-- 1 root root 0 Oct 27 08:51 cgroup.threads
-rw-r--r-- 1 root root 0 Oct 27 08:51 cgroup.type
-rw-r--r-- 1 root root 0 Oct 27 08:51 cpu.pressure
-r--r--r-- 1 root root 0 Oct 27 08:51 cpu.stat
-rw-r--r-- 1 root root 0 Oct 27 08:51 io.pressure
-rw-r--r-- 1 root root 0 Oct 27 08:51 memory.pressure
```

Как можно видеть, вместе с директорией сразу создался и набор файлов. Но не только! Создавая директорию по этому пути, мы создали и контрольную группу, которая тут же прошла инициализацию, создав набор файлов.

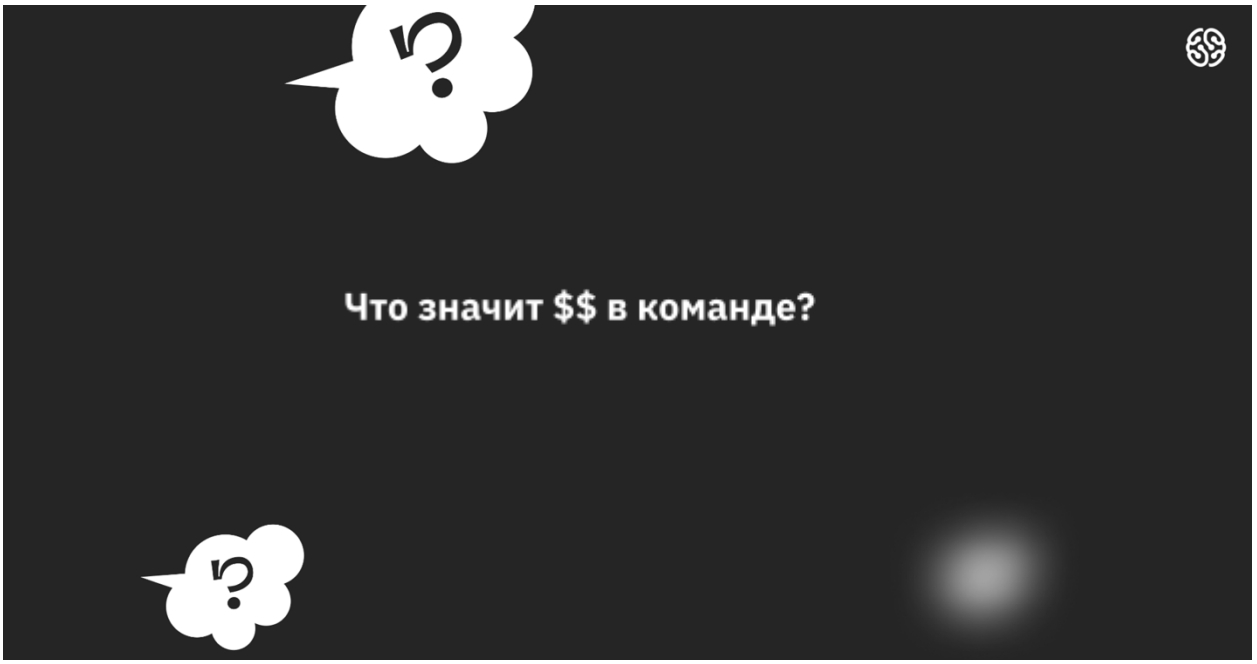
Помимо просто создания файлов (они же - псевдофайлы), они тут же заполняются информацией:

```
# cat /sys/fs/cgroup/cpuset/testgroup1/cpu.stat
usage_usec 0
user_usec 0
system_usec 0

# cat /sys/fs/cgroup/cpuset/testgroup1/cpu.pressure
some avg10=0.00 avg60=0.00 avg300=0.00 total=0
full avg10=0.00 avg60=0.00 avg300=0.00 total=0
```

Вот здесь уже видим изученные нами в теоретической части значения и цифры. Как минимум, видны файлы статистики о загруженности процессора. Можем пойти дальше и посмотреть информацию о процессе, выполняемом нашей текущей командной оболочкой. Для этого предлагается выполнить следующую команду:

```
    # cat /proc/$$/status | grep 'allowed'
    Cpus_allowed:   2
    Cpus_allowed_list:   0-1
    Mems_allowed:
00000000,00000000,00000000,00000000,00000000,00000000,00000000,00
000000,00000000,00000000,00000000,00000000,00000000,00000000,0000
0000,00000000,00000000,00000000,00000000,00000000,00000000,000000
00,00000000,00000000,00000000,00000000,00000000,00000000,00000000
,00000000,00000000,00000001
    Mems_allowed_list:   0
```



Ответ: такими символами обозначается PID процесса, который выполняется нашей текущей командной оболочкой.

Также в команде имеется **grep** - вывод строк, которые содержат сочетание символов, содержащееся в кавычках. Вывод уже интереснее - в нем мы видим уже известные нам параметры: cpus_allowed, mems_allowed. В выводе видно, что для процесса доступны к исполнению 2 процессорных ядра с номерами 0 и 1. Также в выводе содержится информация о памяти, доступной данному процессу.

Контейнеры и LXC

Из приведенного ранее примера должно быть примерно понятно для чего нужны контрольные группы: в них помещаются какие-либо процессы и над ними проводятся различные манипуляции - ограничение ресурсов, запрет на пользование ресурсами либо наоборот разрешение.

Давайте сделаем всю эту теорию более понятной! Разберем как именно используются cgroups в современных инструментах контейнеризации. Для этого предлагаю использовать LXC, так как он более прозрачно позволяет управлять разрешениями и запретами. С его помощью можно будет делать отсылки на теорию на любом шаге.

Прежде, чем мы запустим первый контейнер, сделаем небольшую отсылку к механизму работы LXC. Этот инструмент появился сравнительно недавно и имеет некоторые отличия от Docker. В каких случаях в целом удобно применять LXC?

1. Простота и наследственность: когда у нас уже есть какой-либо готовый сервис на выделенном сервере и этот сервис нужно правильно упаковать и передать разработчикам, тестировщикам либо в целом - другой команде. Да или даже просто - произвести перенос с 1 сервера на другой.
2. LXC очень похож на обычную виртуальную машину с точки зрения настройки, но потребляет ресурсов как обычный Docker. Это позволяет довольно легко настраивать сервисы LXC, так как они очень похожи на сервисы Linux.

Установка контейнера довольно простая:

```
sudo apt-get install lxc debootstrap bridge-utils lxc-templates

sudo lxc-create -n test123 -t ubuntu -f /usr/share/doc/lxc/examples/lxc-veth.conf

lxc-start -d -n test123
```

Давайте дадим пояснения командам выше. Первая команда устанавливает необходимые пакеты через пакетный менеджер. Также она устанавливает набор “заготовок” - templates (тэмплейтов). По сути, это образы операционных систем, которыми можно будет в дальнейшем воспользоваться. Отсюда нам нужен образ ubuntu. Вторая создает контейнер с именем **test123** из образа **ubuntu**, также указывается файл конфигурации. Третья команда запускает контейнер в режиме демона (флаг -d).

Чтобы убедиться, что контейнер действительно создался и работает, можно подключиться к нему и посмотреть как и что происходит:

```
lxc-attach -n test123
```

Собственно, далее предлагается посмотреть на изменения в уже известной нам папке:

```
ls /sys/fs/cgroup
```

Тут мы видим, что появились папки, относящиеся непосредственно к нашему контейнеру:

```
# ll /sys/fs/cgroup/
total 0
dr-xr-xr-x 16 root root 0 Oct 28 08:35 ./
drwxr-xr-x  9 root root 0 Oct 20 08:49 ../
drwxr-xr-x  2 root root 0 Oct 28 08:35 lxc.monitor.test123/
drwxr-xr-x  6 root root 0 Oct 28 08:35 lxc.payload.test123/
```

Почему и был выбран LXC? Вся конфигурация и работа максимально близка к работе с системными ресурсами. Для каждого нового контейнера будет создаваться отдельная директория с набором файлов:

```
ls /sys/fs/cgroup/lxc.payload.test123/
```

Теперь еще один интересный момент: заниматься выделением и ограничением ресурсов можно как напрямую - через файлы, описанные выше, так и через специальные команды **lxc**:

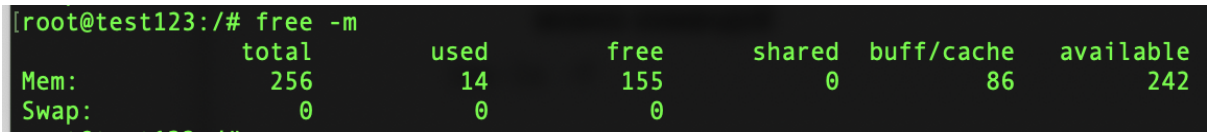
```
/var/lib/lxc/test123/config
```

В этот файл можно добавлять различные параметры. Например:

```
lxc.cgroup2.memory.max = 256M
lxc.start.auto  =  1
```

Эти строки сделают два действия:

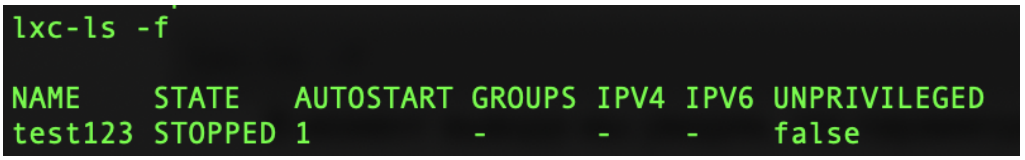
1. Ограничат память контейнера 256 мегабайтами. Проверить это можно внутри контейнера командой free -m.



2. Контейнер будет автоматически запускаться при старте системы. Проверить это можно командой

```
lxc-ls -f
```

В момент вывода мы увидим, что параметр “автозапуск” включен:



На данном этапе могут возникнуть вопросы, так как в интернете встречается много литературы и мануалов, которые предлагают ограничивать контейнеры несколько иным способом:

```
memory.limit_in_bytes 400
```

Этот вариант касается установки параметров для первой версии контрольных групп. И, опять же, большинство инструкций в интернете касаются именно настройки cgroup v1. На данном моменте заострено внимание в связи с тем, что при подготовке урока было потрачено немало времени, пытаясь найти причину неработоспособности ограничения памяти способом, который, казалось бы, везде описан. Вторая версия контрольных групп вышла в 2016 году и немного позднее начала встраиваться в ОС семейства Linux. Основное отличие между группами состоит в том, что вторая версия сформирована на основании предоставления единого продуманного и универсального интерфейса к управлению правами и ограничениями. Первая версия довольно беспорядочна и последовательна в связи с тем, что изначально разрабатывалась как просто механизм для ограничения системных ресурсов, без необходимости работы с контейнерами, а уже позднее, к имеющемуся функционалу стали добавлять дополнительный.

Во второй версии контроллеры компонентов жестко иерархичны и ведут себя стандартизировано. Также в первой версии отсутствовало наследование правил родительской контрольной группы, которая появилась в версии 2.

Резюмируя о разнице версий, можно сказать, что главная проблема первой версии состояла в неправильной ориентации, при которой разные подсистемы подключались к разным иерархиям контрольных групп, в то время как вторая версия использует только одну.

По поводу включения второй версии cgroups в ОС: вторую версию начали включать с версии 4.5, а с версии ядра 4.6 начали появляться патчи для отключения поддержки первой версии при загрузке ядра.

Заключение

Итак, на прошедших уроках мы рассмотрели механизм работы контрольных групп (они же - cgroups) и механизм пространства имен (namespaces). Теперь мы точно готовы установить и начать использовать Docker: создавать контейнеры, запускать их с различными параметрами. Чем на будущих уроках мы и займемся!

Термины, используемые в лекции

cgroup — это механизм для иерархической организации процессов и распределения системных ресурсов. Контрольные группы позволяют образовывать иерархические группы процессов с заданными ресурсными свойствами и обеспечивает программное управление ими.