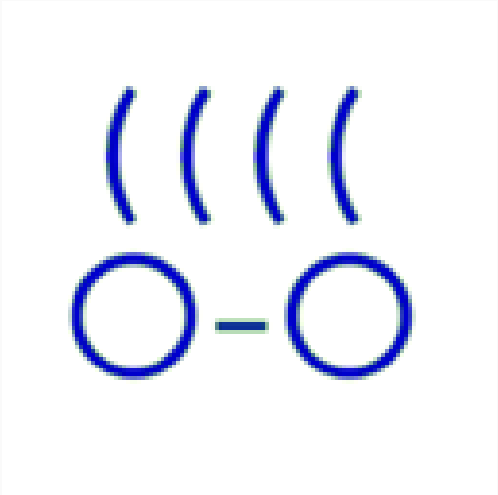
cgroup v2

第 10 回コンテナ型仮想化の情報交換会@東京

加藤泰文

2016-10-29

自己紹介



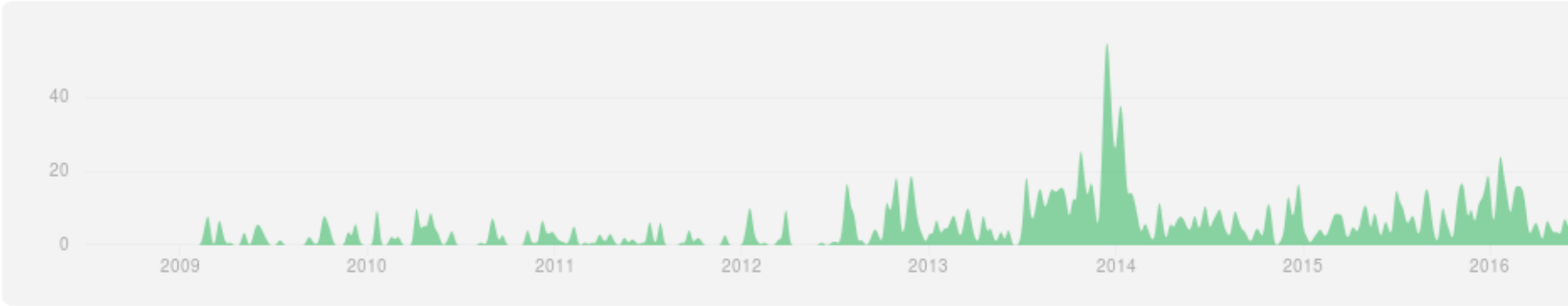
加藤泰文

- http://www.ten-forward.ws/
- @ten_forward
- http://gplus.to/tenforward
- https://github.com/tenforward
- http://d.hatena.ne.jp/defiant/ (技術ブログ)

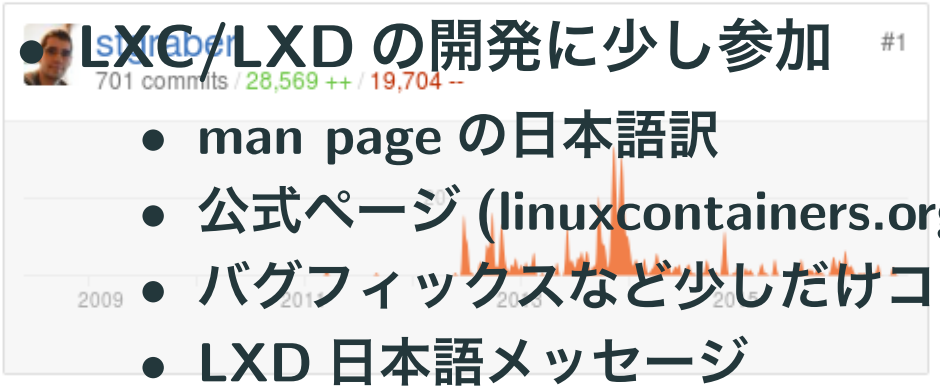
自己紹介

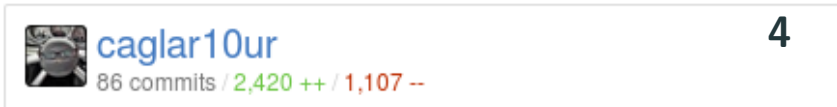
- Plamo Linux メンテナ
- LXC で学ぶコンテナ入門　ー軽量仮想化環境を実現する技術
  gihyo.jp で連載

自己紹介



- LXC/LXD の開発に少し参加
  - man page の日本語訳
  - 公式ページ (linuxcontainers.org) 翻訳
  - バグフィックスなど少しだけコードにも貢献
  - LXD 日本語メッセージ





今日の目標

目標

- cgroup v2 の基本の基本を紹介する

cgroup おさらい

cgroup とは

プロセスをグループ化し、グループに対してリソース制限を行う。コンテナ専用の仕組みではない。

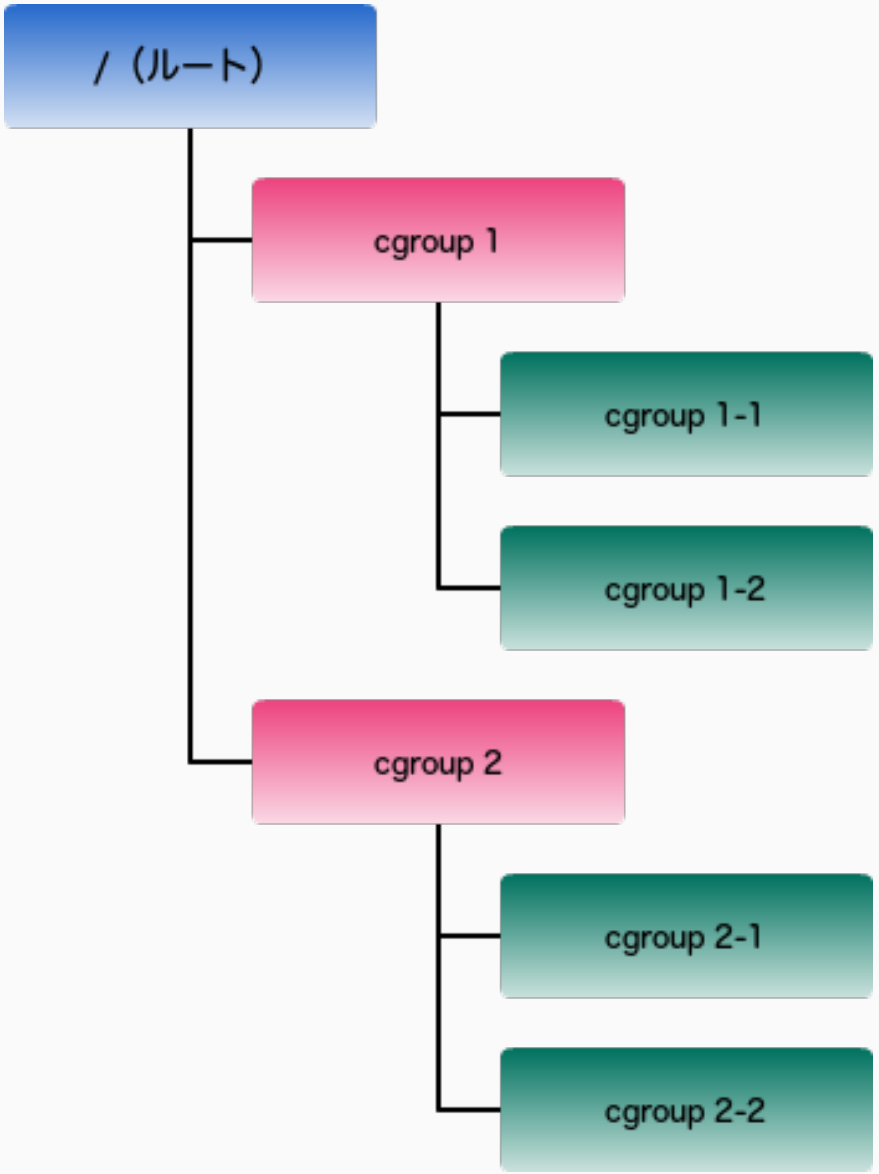
- 機能ごとにサブシステム (コントローラ) に分かれる
- cgroupfs をマウントしてディレクトリでグループを表す
- ディレクトリ内のファイルを読み書きすることで操作を行う
- 現時点で広く利用されている cgroup v1 と、4.5 カーネルで stable になった cgroup v2 がある

用語

cgroup v2 文書で明確に定義されました。

- cgroup は小文字
- cgroup or cgroups?
  - 単数形は
    - 機能を表す際
    - 修飾子として (“cgroup controllers” のように)
  - 複数形は
    - 明確に複数の cgroup を示すときに使う

cgroup の階層構造



cgroup v1 の特徴

- 複数の階層を持てる。cgroupfs を複数マウントできる
- グループへの登録はスレッド単位
- cgroupfs ツリーのどのレベルのノード (ディレクトリ) にもタスクが登録できる

cgroup v1 問題点

複数階層

複数階層

- **任意の数の階層を作成でき、それぞれにいくつものコントローラが所属できる**
- **柔軟で便利**

のはず…

コントローラの制限

ところが

- コントローラはひとつの階層にしか所属できない
  - 複数の階層で使えると便利なコントローラ (例えば freezer) が、特定の階層でしか使えない
- 一度ある階層に属したコントローラは移動できない
- 同じ階層に属しているコントローラは同じ構造でなければならない

結局

柔軟性なんてそんなになかった

- コントローラごとに階層を作成するのが一般的になった
  - 密接に関係し、同じようなグループで扱う意味のあるコントローラだけ同じ階層に属させることに

コントローラ間の関係

- コントローラ間の連携がない
  - 複数コントローラで連携して動作させられない
- コントローラによって動きもバラバラ
  - cpuset の cgroup.clone_children ファイル
  - memory の memory.use_hierarchy ファイル
  - などなど…

タスクの扱い

- どの階層のノードにもタスクが所属できる
  - 親子の cgroup どちらにもタスクが属している場合のリソース割り振りとかカオス
- タスクの単位がスレッド単位
  - コントローラによっては意味がない

cgroup v2 概要

歴史

- 3.16 で “unified control group hierarchy” として導入
  - `__DEVEL__sane_behavior` オプションでマウントしてた (まともな振る舞いオプション!!)
- (参考)
  - The unified control group hierarchy in 3.16 (lwn.net)
  - Linux カーネルのすべて: cgroup の再設計 (linux.com)
- 4.5 で stable に

特徴

- 単一階層構造
- 管理はプロセス単位
- v1 と共存できる
  - 一部のコントローラは v2 で、他のコントローラは v1 で利用とかできる
- 操作は v1 と同じ (ディレクトリ管理、ファイルへの読み書きで操作)

実装

- 4.8 時点で memory、io、pids のみ
  - ドキュメントに cpu の記載ありますがマージされていません

cgroup v2 操作

cgroup v2 の利用

- マウントする

```
# mount -t cgroup2 cgroup2 /sys/fs/cgroup
```

cgroup の作成・削除

- マウント直後は root cgroup のみ存在
- ディレクトリの作成によりグループを作成する

```
# mkdir cgroup_name
```

- ディレクトリの削除によりグループを削除する

```
# rmdir cgroup_name
```

  - 子 cgroup もなく、プロセスもない cgroup のみ削除できる

プロセスを cgroup に所属させる

- PID を `cgroup.procs` に書き込む

```
# echo $PID > /path/to/cgroup/cgroup.procs
```

- プロセスがどの cgroup に属するかは`/proc/$PID/cgroup` にリストされる

```
# cat /proc/$$/cgroup
0::/cgroup_name
```

cgroup の状態監視

- **root 以外の cgroup には `cgroup.events` ファイルが存在**

```
# cat cgroup.events
populated 0 (自分もしくは子孫の cgroup にプロセスが存在しないときは populated が 0)
# echo $$ > cgroup.procs (プロセスを追加すると…)
# cat cgroup.events
populated 1 (プロセスが存在するときは populated が 1)
```

- **`populated` の値が変化するとファイルが変化したイベント発生 (poll,dnotify,inotify)**

```
$ inotifywait -m /sys/fs/cgroup/test01/cgroup.events
(test01 にプロセスを追加する)
/sys/fs/cgroup/test01/cgroup.events MODIFY
```

コントローラの制御

- 各 cgroup で使用できるコントローラは
  cgroup.controllers にリストされる

```
# cat cgroup.controllers
io memory pids
```

- 子 cgroup で使用したいコントローラは
  cgroup.subtree_control で制御する

```
（使いたいコントローラには"+"を、消去したい・使わないコントローラには"-"をつける）
# echo "-pids +memory +io" > cgroup.subtree_control
# cat cgroup.subtree_control
io memory
```

子 cgroup を持つ場合の制約

- 自身がプロセスを持たないときだけ、子供にリソースを分配できる
  - プロセスを持たない cgroup のみ、`cgroup.subtree_control` ファイルでコントローラを有効にできる
  - root はこの制約を受けない

```
# cat cgroup.procs
3541
3577
# mkdir child
# echo "+io" > cgroup.subtree_control
bash: echo: write error: Device or resource busy
# echo $$ > /sys/fs/cgroup/cgroup.procs
(プロセスを root へ戻し現グループから削除)
# echo "+io" > cgroup.subtree_control
# cat cgroup.subtree_control
io
```

非特権ユーザへの権限委譲

- cgroup のディレクトリと cgroup.procs ファイルへの書き込み権限を与え、非特権ユーザに権限委譲する
  - そのグループ以下は自由にリソース配分できる

リソース分配の方法

- Weights
  - グループ間の比率
- Limits
  - 設定量までリソースを使用できる (オーバーコミット可能)
- Protections
  - (祖先が制限を超えていない限り) 割り当てが保証される (オーバーコミット可能)
- Allocation
  - 有限リソースの割り当て (オーバーコミット不可)

コントロールファイル

ファイル名

- ウェイトでのリソース分配を行う場合、ファイルは “weight”
- 絶対値でのリソース保証、制限の場合、ファイルはそれぞれ “min”、“max”
- ベストエフォートのリソース保証、制限の場合、ファイルはそれぞれ “low”、“high”

リソース制限の方法

- v1 とほぼ同じ
  - それぞれのコントローラの使い方を参照しましょう
- ウェイトの場合、デフォルト値が 100、範囲は 0〜10000
- デフォルト値が設定できる場合 “default” とキーワードを使い登録

```
echo "default 100" > control_file
echo "8:0 150" > control_file ("8:0"の制限値を 150 に)
echo "8:0 default" > control_file ("8:0"の制限値をデフォルトに戻す)
```

- 制限なしの場合は “max” を使う

設定例

```
# cat pids.max（プロセス数の制限値の表示）
max（デフォルト値）
# echo "2" > pids.max（プロセス数の制限値を 2 に設定）
# cat pids.max
2
# echo $$ > cgroup.procs
# cat pids.current（現在のプロセス数の表示）
2
# ( echo "test" | cat )
bash: fork: retry: No child processes
bash: fork: retry: No child processes
bash: fork: retry: No child processes
bash: fork: retry: No child processes
bash: fork: Resource temporarily unavailable
Terminated
# echo "max" > pids.max（プロセス数の制限を外す）
# cat pids.max
max
# ( echo "test" | cat )
test
```

cgroup v2 のステータス

- stable だけど
- CPU コントローラが反対にあってマージされていない
  - The case of the stalled CPU controller (lwn.net)
  - [Documentation] State of CPU controller in cgroup v2 (lwn.net) (Tejun Heo 氏によるまとめ)