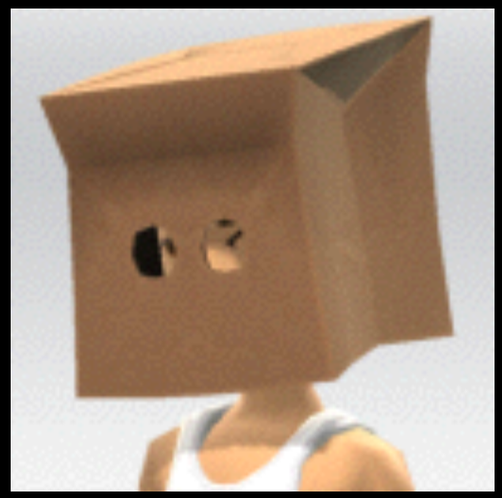
つくって学ぶ
Linuxコンテナの裏側

ENOG39 Meeting
1 Jul 2016

自己紹介

- Hayato Imai(@hayajo)
  - ウォータセル株式会社
    - アグリノート
    - インフラ担当



アジェンダ

- Linuxコンテナ概要
- コンテナエンジンをつくろう
- まとめ

Linuxコンテナ概要

ハードウェア仮想化と OSレベル仮想化（1）

ハードウェア仮想化

- 複数の仮想化されたハードウェアを動かす仮想化方式
- VMWare, VirtualBox, Xen, KVM, Hyper-Vなどで採用されている

OSレベル仮想化

- 複数の仮想化されたOS環境を動かす仮想化方式
- Container(Linux), Jail(FreeBSD), Zone(Soralis)と呼ばれている

ハードウェア仮想化と OSレベル仮想化（2）

	ハードウェア仮想化	OSレベル仮想化
OS	自由	ホスト共通
フットプリント	大きい	小さい
集積度	低い	高い
起動スピード	遅い	速い
隔離レベル	高い	低い

Linuxコンテナとは

- OSレベル仮想化を基本とした軽量の仮想化技術
- 同一カーネルのプロセス群をグループ化
- 他のグループとは独立した環境で動作

あたかも別のシステム上でプロセスを実行しているように見せる機能です。

コンテナを構成する主な技術

「コンテナ」と呼ばれる一つの技術でできているわけではなく、
下記に挙げるような複数の技術の組み合わせで構成される。

- Namespace
- Control Group(cgroup)
- Capability
- chroot/pivot_root
- bind mount
- Union Filesystem
- Seccomp
- MAC
- veth, macvlan, ipvlan

and more.

代表的なコンテナ実装

- Docker(runC)
  https://www.docker.com/
- LXC
  https://linuxcontainers.org/
- rkt(CoreOS Rocket)
  https://coreos.com/rkt/
- systemd
  https://www.freedesktop.org/wiki/Software/systemd/

軽量コンテナ実装

- jailing
  https://github.com/kazuho/jailing
- droot
  https://github.com/yuuki/droot
- mincs
  https://github.com/mhiramat/mincs
- Firejail
  https://firejail.wordpress.com/
- pflask
  https://github.com/ghedo/pflask

コンテナエンジンを
つくろう

コンテナエンジン "yapC"

https://github.com/hayajo/yapC/tree/yapc8oji-2016mid

```
$ sudo yapc /bin/bash
$ sudo YAPC_CPU_QUOTA=50000 yapc \
/bin/bash -c "yes >/dev/null"
$ sudo YAPC_CAPS="cap_net_raw" yapc ping 127.0.0.1
$ sudo YAPC_ROOT=centos yapc yum --help
```

- シェルスクリプト
- リソースを分離した環境でプログラムを実行
- CPUやメモリなどのシステムリソースを制限可能
- コンテナ内の権限を制限可能
- rootファイルシステムを指定可能

yapCの実行環境

- Ubuntu 16.04（カーネル 4.4.0）
  - Vagrantfie同梱
    https://atlas.hashicorp.com/boxcutter/boxes/ubuntu1604
- コンテナ内でbashとcapsh(libcap)の実行環境が必要

yapCに必要な要素

1. リソースの分離
2. リソースの制限
3. 権限の制限
4. rootファイルシステムの変更

1. リソースの分離

Namespaceと呼ばれるカーネルの機能で実現します。

Namespaceとは

- マウントポイント、PIDなどのリソースを分離し、プロセスに対して専用のグローバルリソースを持っているかのように見せる

- すべてのプロセスはいずれかのネームスペースに属している

- ネームスペースを分離しない場合は親プロセスのネームスペースを引き継ぐ（fork, CoW）

- /proc/<PID>/ns/ で確認できる

Namespaceの種類

名前	カーネル	概要
Mount	2.4.19	マウントポイントの集合を分離する。
UTS	2.6.19	ホスト名、NISドメイン名を分離する。
IPC	2.6.19	SysV IPCオブジェクト、POSIXキューを分離する。
PID	2.6.24	PID空間を分離する。
Network	2.6.26	ネットワークに関連するシステムリソースを分離する。
User	3.8	UID/GIDなどを分離する。
Cgroup	4.6	Cgroupルートディレクトリを分離する

Namespaceの操作

- clone(2)
  新しいプロセスを生成してネームスペースを分離する

- unshare(2)
  現在のプロセスのネームスペースを分離する(*)

- setns(2)
  指定したプロセスのネームスペースを変更する(*)

- unshare(1)
  unshare(2)のCLIインターフェース

(*) PIDネームスペースは子プロセスのネームスペースが分離/移動される

yapCのNamespace実装

https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.1

- unshare(1)でリソースを分離
- Mount, UTS, IPC, PIDネームスペースを分離

https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.1#L13

yapCのNamespaceデモ

```
$ sudo /vagrant/yapc.1 /bin/bash
CONT# ps auxf
CONT# hostname yapc; hostname
```

- ✔ PIDが独立していることを確認
- ✔ ホスト名変更がホストへ影響がないことを確認

2. リソースの制御

Control Group(cgroup)と呼ばれるカーネルの機能で実現します。

Contorl Group(cgroup)とは

- プロセスをグループ化し、グループに存在するプロセスに対してCPU時間、メモリ使用量、ブロックデバイスの入出力帯域などのリソースの管理を行う
- cgroupを動的に再定義することも可能

cgroupファイルシステム

- cgroupfsをマウントし、ディレクトリによる階層構造でグループを表現

- cpu, memoryなど単一のリソースをサブシステムと呼ぶ

- 異なる複数の階層を持つことが可能
  （cpu, memoryなどのサブシステムごと）

- 複数のサブシステムを組み合わせた階層構造も作製可能
  （cpu+memoryなど）

- 一般的には/sys/fs/cgroup/以下にマウント

サブシステムの種類（1）

サブシステム	カーネル	概要
cpu	2.6.24	CPU実行時間の制御
cpuacct	2.6.24	CPUレポートの生成
cpuset	2.6.24	CPUコアの割当
devices	2.6.26	デバイスファイルのアクセス制御
freezer	2.6.28	プロセスの一時停止/再開
memory	2.629	メモリ上限の制御

サブシステムの種類（2）

サブシステム	カーネル	概要
net_cls	2.6.29	ネットワークパケットへのタグ付け
blkio	2.6.33	ブロックデバイスの入出力制御
pref_event	2.6.39	prefツールでモニタリングの制御
net_prio	3.3	ネットワークトラフィックの優先度を制御
hugetlb	3.6	サイズの大きい仮想メモリページの再読
pids	4.3	プロセス上限の制御

cgroupの操作

- /sys/fs/cgroupディレクトリの操作
- cgroup-tools(libcgroup-tools)

yapCのcgroup実装

https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.2

- `cgcreate/cgset/cgdelete`でcgroupを操作
  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.2#L13-L14
  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.2#L24-L27
- 子プロセスとして起動するコンテナのリソースを制御するために、`unshare`では自身のスクリプト（`$0`）を呼び出してPIDチェック
  （PID Namespace + forkなので、コンテナでのPIDは1となる）
  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.2#L8-L22
- `trap`でプロセス終了時に作成したグループを削除
  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.2#L17

yapCのcgroupデモ

```
$ sudo YAPC_CPU_QUOTA=50000 \
/vagrant/yapc.2 /bin/bash -c "yes >/dev/null"
```

- ✔ ホストで**top**を実行し、上記コンテナプロセスのCPU使用率が50%付近になることを確認

3. 権限の制限

Capabilityと呼ばれるカーネルの機能で実現します。

Capabilityとは（1）

- 細分化したroot権限をプロセス、ファイルに割り当てる
  - 通常は一般ユーザー権限（非特権）で動くか、root権限（特権）で動くかの2種類
  - 動作している特権プロセスの脆弱性により、コンピューターを自由に操作されてしまう恐れ
- プロセスに脆弱性があったとしても影響の範囲を狭めることができる

Capabilityとは（2）

例）set-user-ID-rootではないpingプログラムに
CAP_NET_RAWケーパビリティを与える

```
vagrant@vagrant:~$ ls -l /bin/ping
-rwsr-xr-x 1 root root 44168 May  7  2014 /bin/ping
vagrant@vagrant:~$ cp /bin/ping .
vagrant@vagrant:~$ ls -l ./ping
-rwxr-xr-x 1 vagrant vagrant 44168 Jun 29 23:51 ./ping
vagrant@vagrant:~$ ./ping 127.0.0.1
ping: icmp open socket: Operation not permitted
vagrant@vagrant:~$ sudo setcap CAP_NET_RAW+ep ./ping
vagrant@vagrant:~$ getcap ./ping
./ping = cap_net_raw+ep
vagrant@vagrant:~$ ./ping -c 3 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.014 ms
...
```

Capabilityの種類（１）

- 現在37種類
  http://man7.org/linux/man-pages/man7/capabilities.7.html

Capabilityの種類（2）

Dockerでデフォルトで有効になるCapability（14種類）

ケーパビリティ	概要
CAP_AUDIT_WRITE	カーネル監査のログにレコードを書き込む
CAP_CHOWN	ファイルの UID とGID を任意に変更する
CAP_DAC_OVERRIDE	ファイルのread/write/execの権限チェックをバイパスする
CAP_FOWNER	ファイルのUIDとGIDを変更する
CAP_FSETID	ファイルが変更されたときにsuidとsgidビットをクリア しない
CAP_KILL	シグナルを送信する際に権限チェックがバイパスする
CAP_MKNOD	mknod(2)でスペシャル・ファイルを作成する
CAP_NET_BIND_SERVICE	ウェルノウンポートをバインドする
CAP_NET_RAW	RAWソケットとPACKETソケットの使用する
CAP_SETFCAP	ファイルケーパビリティを設定する
CAP_SETGID	プロセスのGIDと追加のGIDリストを操作する
CAP_SETPCAP	プロセスのケーパビリティを操作する
CAP_SETUID	プロセスのUIDを操作する
CAP_SYS_CHROOT	chroot(2)を呼び出す

Capability Set

プロセス、ファイルごとにケーパビリティセットを持つ

ケーパビリティセット	プロセス	ファイル	説明
許可（Prm）	✔	✔	EffとInhで持つことが許可されるケーパビリティの集合。一度OFFにしたものは自力で再セット不可
継承（Inh）	✔	✔	execve(2)した際に継承するケーパビリティの集合
実効（Eff）	✔	✔	実際に判定されるケーパビリティの集合 （ファイルでは1ビット）
バウンディング（Bnd）	✔		獲得できるケーパビリティを制限するための集合
環境（Amb）	✔		特権のないプログラムをexecve(2)した際に保持されるケーパビリティの集合

Capability Setの確認

プロセスのケーパビリティセットの確認

```
$ cat /proc/self/status | grep ^Cap
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
$ sudo cat /proc/self/status | grep ^Cap
CapInh: 0000000000000000
CapPrm: 0000003fffffffff
CapEff: 0000003fffffffff
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
```

ファイルのケーパビリティセットの確認

```
$ getcap ./ping
./ping = cap_net_raw+ep
```

Capabilityの求め方

```
P'(Amb) = (file is privileged) ? 0 : P(Amb)
P'(Prm) = (P(Inh) & F(Inh))
          | (F(Prm) & cap_bset)
          | P'(Amb)
P'(Eff) = F(Eff) ? P'(Prm) : P'(Amb)
P'(Inh) = P(Inh)
```

P : execve(2)前のプロセスのケーパビリティセット
P' : execve(2)後のプロセスのケーパビリティセット
F : ファイルのケーパビリティセット
cap_bset : プロセスのバウンディングセット

Capabilityの操作

- prctl(2)
- capset(2), capget(2)
- setxattr(2), getxattr(2)
- libcap（推奨）

yapCのCapability実装

https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.3

- capsh(8)でケーパビリティを操作

  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.3#L45-L49

- デフォルトのケーパビリティセットはDockerを参考（ただしCAP_NET_RAWは除く）

  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.3#L6

yapCのCapabilityデモ

```
$ sudo /vagrant/yapc.3 ping 127.0.0.1
```

✔ pingができないことを確認

```
$ sudo YAPC_CAPS="cap_net_raw" \
/vagrant/yapc.3 ping 127.0.0.1
```

✔ pingができることを確認

4. rootファイルシステムの変更

chrootや**pivot_root**で実現します。
また、**overlayfs**も利用します。

chroot/pivot_rootとは

chroot

- プロセスのルートディレクトリを変更する
- パス名解決時のルートディレクトリが変更される

pivot_root

- プロセスのrootファイルシステムを入れ替える
- 元のrootファイルシステムをアンマウントできる

pivot_rootの条件

pivot_rooで指定する新しいファイルシステムと（***new_root***）と元のファイルシステムの移動先（***put_old***）は以下の制限がある

- ディレクトリでなければならない
- *new_root*と*put_old*は現在のrootと同じファイルシステムにあってはならない
- *put_old*は*new_root*以下になければならない
- 他のファイルシステムが*put_old*にマウントされていてはならない

chroot/pivot_rootの操作

- chroot(2), chroot(8)
- pivot_root(2), pivot_root(8)

overlayfsとは

- Union Filesystemのひとつ
- ディレクトリを重ねあわせて1つのファイルシステムに見せる
- copy-on-writeでファイルを差分管理する
- カーネル3.18から取り込まれた

overlayfsのlower, upper, work

lower
- 下層のディレクトリ。読み取り専用
- ファイルシステムのベースとなるディレクトリ

upper
- 上層のディレクトリ。書き込み可能
- 新規作成、更新されたファイルはここに書き出される

work
- 作業用ディレクトリ
- upperと同じファイルシステムに存在する必要がある

overlayfsの操作

- mount(2), mount(8)

```
$ CLONE_DIR=$(mktemp -d)
$ for d in upper work root; do mkdir $CLONE_DIR/$d; done
$ sudo mount \
-t overlay \
-o lowerdir=/,upperdir=$CLONE_DIR/upper,workdir=$CLONE_DIR/work \
overlayfs \
$CLONE_DIR/root
```

yapCのpivot_root, overlayfs実装

https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.4

- overlayfs(mount(8))
  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.4#L35-L39
- pivot_root(8)
  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.4#L70-L75
- bind_mount
  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.4#L41-L59
  - ファイルやディレクトリなどをファイルシステムの別の場所で見えるようにする
  - symlinkと違い`chroot`/`pivot_root`による制約がない
  - hardlinkと違いディレクトリもOK

yapCのpivot_root, overlayfs デモ（1）

```
$ sudo /vagrant/yapc.4 /bin/bash
CONT# touch /yapc; ls -l /yapc
```

- ✔ ホストで/yapcが存在しないことを確認
- ✔ ホストで/tmp/yapc-<PID>.XXXXXX/upper/yapc が存在することを確認

yapCのpivot_root, overlayfs デモ（2）

```
$ sudo YAPC_ROOT=centos /vagrant/yapc.4 /bin/bash
CONT# yum install -y epel-release; yum install -y sl; sl
```

- ✔ 事前にdocker exportでcentosのrootファイルシステムアーカイブを作製。ホストの~/centosへ展開する

  docker export $(docker create centos) > centos.tar
- ✔ yumコマンドが使えることを確認

Appendix.

ネットワークの分離

Network Namespaceで実現します。
分離されたネットワーク間の接続には**veth**を利用します。

Network Namespaceが分離するリソース

- ネットワークデバイス
- IPv4/v6 プロトコルスタック
- ルーティングテーブル
- ファイアウォール
- /proc/net/
- /sys/class/net/
- ポート番号

Network Namespaceの操作

- ip-netns(8)
  - `ip netns [list]`
    ネットワークネームスペースを一覧表示
  - `ip netns add`
    ネットワークネームスペースの作製
  - `ip netns del`
    ネットワークネームスペースの削除
  - `ip netns exec`
    ネットワークネームスペースを指定してコマンド実効
  - and more.

vethとは

- 仮想的なイーサネットインターフェースのペアを作成して接続する

vethの操作

- ip-link(8)

```
$ sudo ip link add \
name veth0 \
type veth \
peer name veth1
```

yapCの Network Namespaceとveth実装

https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.a

- ip-netns(8), ip-link(8)

  https://github.com/hayajo/yapC/blob/yapc8oji-2016mid/yapc.a#L28-L37

  - `ip netns exec`で`/sys/`がアンマウントされてしまう…
    - cgroupが設定できない
  - `pivot_root`でホストの`/run(/var/run)`が見えなくなる…
    - Network Namespaceが利用できない
  - 上記２点から実装が苦しい感じに（良い案がありましたら教えて下さい）

yapCの Network Namespaceとvethデモ（1）

```
$ sudo \
YAPC_NET=1 YAPC_CAPS="cap_net_raw,cap_net_admin" \
/vagrant/yapc.a \
ip link
```

- ✔ コンテナ内で独立したネットワークインターフェースが見えることを確認

yapCの Network Namespaceとvethデモ（2）

1. ブリッジを作成する

```
$ sudo ip link add name yapc0 type bridge
$ sudo ip link set dev yapc0 up
$ sudo ip a add 10.0.0.1/24 \
broadcast 10.0.0.255 \
label yapc0 \
dev yapc0
```

yapCの Network Namespaceとvethデモ（2）

2. ネットワークネームスペースを有効にしたコンテナでbashを実行

```
$ sudo \
YAPC_NET=1 YAPC_CAPS="cap_net_raw,cap_net_admin" \
/vagrant/yapc.a \
/bin/bash
```

yapCの Network Namespaceとvethデモ（3）

3. ホスト側のvethをブリッジに登録する

```
$ ip link # デバイス名を確認
$ sudo ip link set dev vethXXXXXXX up
$ sudo ip link set dev vethXXXXXXX master yapc0
```

4. コンテナ側のvethにIPアドレスを割り当る

```
CONT# ip link # eth0の存在を確認

CONT# ip link set dev eth0 up
CONT# ip a add 10.0.0.10/24 dev eth0
```

yapCの Network Namespaceとvethデモ（4）

```
CONT# ping 10.0.0.1
```

✔ ホストにpingができることを確認

まとめ

まとめ

- Linuxコンテナは様々な機能の組み合わせでできている
- リソースの分離
  - Namespace
- リソースの制限
  - cgroup
- 権限の制限
  - Capability
- rootファイルシステムの変更
  - chroot/pivot_root, overlayfs

まとめ

機会があればコンテナのセキュリティまわりの
機能や対策について話せれば...

- Dockerでホストを乗っ取られた
  http://qiita.com/titilat/items/f4f94a0cee9049125156

- 非特権コンテナ
- MAC
- Seccomp
- PR_SET_NO_NEW_PRIVS