chrootとnetwork namespaceでつくる簡易コンテナ

第11回 コンテナ型仮想化の情報交換会@大阪

自己紹介

- id:masayoshi
- はてな@京都
- Webオペレーションエンジニア
- 大学時代はSDN関連の研究

今日話すこと

- 自作コンテナのモチベーション
- chroot ✕ network namespace ✕ UTS namespace

今日話すこと

- TenForward氏の詳細な解説で基礎技術を理解し、
- 私の雑な発表でコンテナ自作に興味を持ってもらい、
- udzura氏のhaconiwaでぜひチャレンジして欲しい

コンテナ自作のモチベーション

- Linuxコンテナの勉強
    - 基礎部分、実装によらない共通技術の勉強
- 既存コンテナ技術の再確認
    - 作って使ってみると違いなどがよく分かる
- 手元でのネットワークテスト環境
    - 細かく変更するので自分で触りやすい方が良い

chroot

network namespace

UTS namespace

なんでこの3つ?

- 合わせて使うとシンプルだが意外とおもしろい物が動かせる
  - 学生のとき研究でnetwork nsをよく使っていた
  - ネットワークで遊ぶときはこの構成を使っている
- chroot namespaceの一部のみの組み合わせは多くなさそう
  - 1つ1つや全てを組み合わせた実装例は色々ある

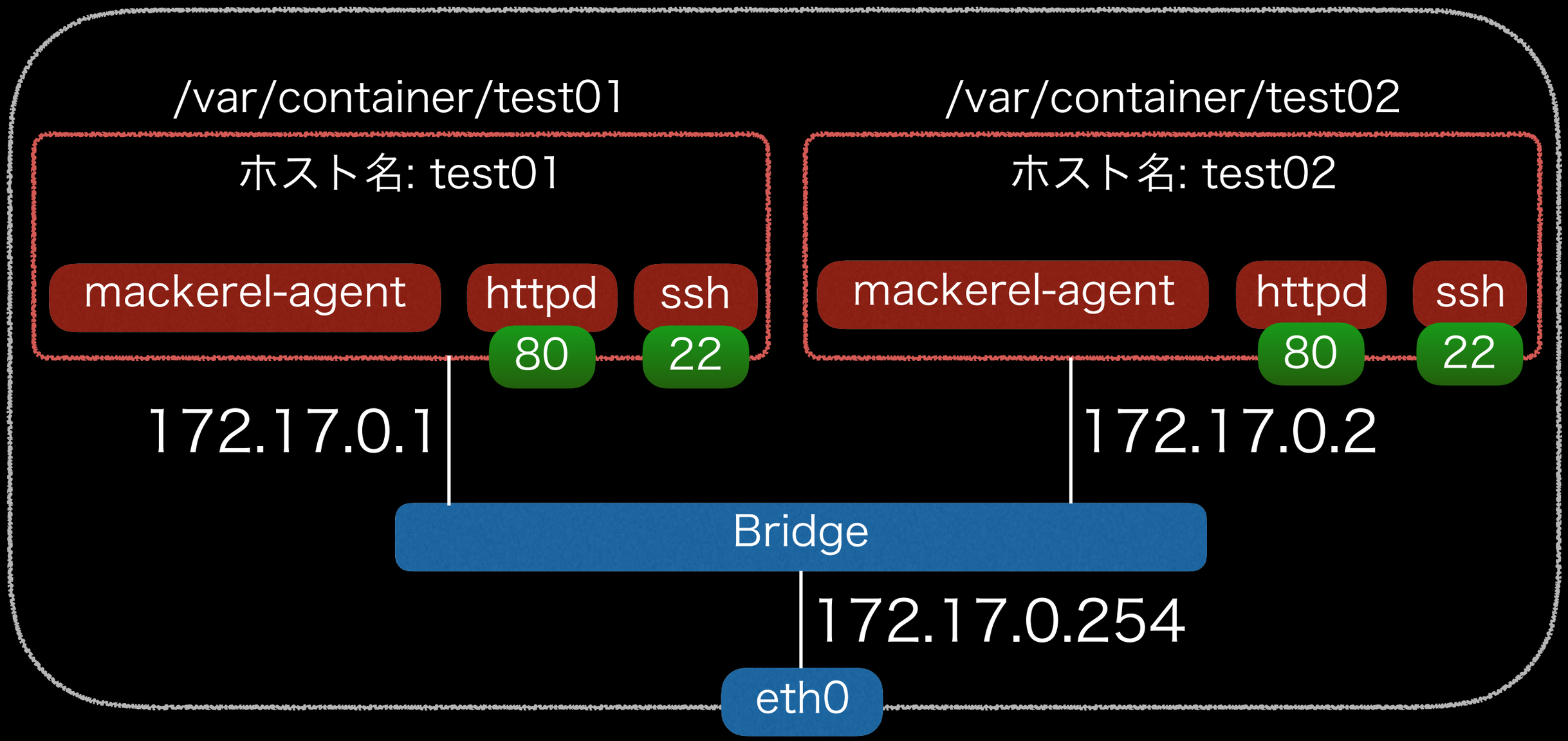
この3つでも面白いことが出来る

- chroot
    - docker exportなどの展開されたイメージの実行
- network namespace
    - 特定のIPアドレス + ポートでのLISTEN
- UTS namespace
    - 管理上の利便性

例えば

- apache + mackerel-agent + ssh なコンテナ
    - Webサーバ
    - 監視用エージェントとsshが動作
        - アプリケーション + 監視 + 管理

- 同一の物理サーバで上記のコンテナを複数起動可能
    - networkはLinux Bridgeでブリッジ接続


/var/container/test01
ホスト名: test01
mackerel-agent
httpd
ssh
80
22
172.17.0.1
/var/container/test02
ホスト名: test02
mackerel-agent
httpd
ssh
80
22
172.17.0.2
Bridge
172.17.0.254
eth0

```
touch /var/run/utsns/test01
unshare --uts=/run/utsns/test01 hostname test01
ip netns add test01
ip link add name test01-br type veth peer name test01-ct
brctl addif br0 test01-br
ip link set test01-ct netns test01
ip netns exec test01 ip addr add 172.17.0.1/24 dev test01-ct
ip netns exec test01 ip link set lo up
ip netns exec test01 ip link set test01-ct up
ip link set test01-br up
ip netns exec test01 ip route add default via 172.17.0.254
mount -t proc proc /mnt/test01/proc
mount --rbind /sys /mnt/test01/sys
mount --make-rslave /mnt/test01/sys
mount --rbind /dev /mnt/test01/dev
mount --make-rslave /mnt/test01/dev
```

UTS

```
touch /var/run/utsns/test01
unshare --uts=/run/utsns/test01 hostname test01
```

network

```
ip netns add test01
ip link add name test01-br type veth peer name test01-ct
brctl addif br0 test01-br
ip link set test01-ct netns test01
ip netns exec test01 ip addr add 172.17.0.1/24 dev test01-ct
ip netns exec test01 ip link set lo up
ip netns exec test01 ip link set test01-ct up
ip link set test01-br up
ip netns exec test01 ip route add default via 172.17.0.254
```

chroot

```
mount -t proc proc /mnt/test01/proc
mount --rbind /sys /mnt/test01/sys
mount --make-rslave /mnt/test01/sys
mount --rbind /dev /mnt/test01/dev
mount --make-rslave /mnt/test01/dev
```

touch /var/run/utsns/test01
unshare --uts=/run/utsns/test01 hostname test01
ip netns add test01
ip link add name test01-br type veth peer name test01-ct
brctl addif br0 test01-br
ip link set test01-ct netns test01
ip netns exec test01 ip addr add 172.17.0.1/24 dev test01-ct
ip netns exec test01 ip link set test01-ct up
ip link set test01 lo up
ip netns exec test01 ip route dd default via 172.17.0.254
mount -t proc proc /mnt/test01/proc
mount --rbind /sys /mnt/test01/sys
mount --make-rslave /mnt/test01/sys
mount --rbind /dev /mnt/test01/dev
mount --make-rslave /mnt/test01/dev

コンテナ作成に16コマンド

(imageの作成は除く)

デモしながら見ていく

imageの作成

- dockerならdocker export で
    - build, shipはdockerでやると楽そう
        - 今回はrun部分で遊ぶ
- dockerなしならdebootstrapなど
    - 今回はdebootstrapで作成したやつを利用

namespaceの永続化

- /proc/[PID]/ns 配下にある特殊ファイル

/proc/[PID]はプロセスが消えるとなくなるので永続化が必要

```
lrwxrwxrwx 1 root root 0  6月 17 02:18 ipc -> ipc:[4026531839]
lrwxrwxrwx 1 root root 0  6月 17 02:18 mnt -> mnt:[4026531840]
lrwxrwxrwx 1 root root 0  6月 17 02:18 net -> net:[4026531969]
lrwxrwxrwx 1 root root 0  6月 17 02:18 pid -> pid:[4026531836]
lrwxrwxrwx 1 root root 0  6月 17 02:18 uts -> uts:[4026531838]
```

namespaceの永続化

- bindマウントをつかって永続化する

  mount --bind /run/utsns /run/utsns
  mount --make-shared /run/utsns
  unshare -u mount --bind /proc/self/ns/uts /run/utsns/test01

- 最近のunshareコマンドは永続化が楽

  unshare --uts=/run/utsns/test01

UTS namespace

- 主に管理のため
- コンテナに入ったときとか
  - シンプルに使えるのでお気軽

```
touch /var/run/utsns/test01
unshare --uts=/run/utsns/test01 hostname test01
```

Networkの作成

- veth作ってbridgeに接続
    - TenForward氏によるデモがありそうなので省略

- (私は)色々変更することが多い
    - Linux BridgeをOpen vSwitchにしたり
    - 自作のソフトウェアルータに接続したり
    - KVMのVMと接続したり
    - 複数NIC + mptcp環境

Networkの作成

- Networkはポータビリティに影響が出やすい
- 一時期dockerが頑張ってた
    - VXLANによるoverlay Networkなど
- 改善すべき箇所がたくさんある面白い分野
    - オフローディング, SR-IOVなど高速化
    - VXLANなどのプロトコル技術

chroot環境の作成

- proc, sys, devなどをmountする

```
mount -t proc proc /mnt/test01/proc
mount --rbind /sys /mnt/test01/sys
mount --make-rslave /mnt/test01/sys
mount --rbind /dev /mnt/test01/dev
mount --make-rslave /mnt/test01/dev
```

systemd環境ではbindマウントがSHAREDになったので
rslaveしておかないとumount -Rした時におかしくなる

コンテナ内でのプロセスの実行

- nsenterをつかってnamespaceをattach
- その上でchrootする

```
nsenter --net=/run/netns/test01 \
        --uts=/run/utsns/test01 \
        chroot /mnt/test01 \
        /etc/init.d/nginx start
```

同様にsshなども起動する

コンテナ内でのプロセスの実行

- chroot配下ではsystemdは動作しないので注意が必要
- chrootの代わりにsystemd-nspawnを使って動かす方法もある
  - その場合後述のPID namespaceを使うことになる

PID namespace

- PID分離すると生成された子プロセスはその空間ではinit(PID=1)となる
    - initが死ぬと悲しいことになるので維持する必要がある
    - より良いinitを求める旅が始まる
    - docker 1.13で runに initオプションが付いてそう
    - また/sbin/init を実行する、しないといった選択肢も増える

- 今回の用途ではいらないので省いた
    - 実際には必要となることが多い
    - 上記理由で気軽にやるなら省くと楽

PID namespaceを利用しないと...

- ps 結果が分離されない
    - コンテナ内、コンテナ外から見放題
- initにぶら下がるdaemon
    - UST, networkのnamespaceは分離されている
    - プロセス生成時に分離しなければ継承される
    - プロセスの終了をどうするか
- ss -N test01 -tlpなどでLISTENを確認すると分離されていることがわかる

ps 結果

```
root      2845  nginx: master process /usr/sbin/nginx
http      2848  \_ nginx: worker process
http      2849  \_ nginx: worker process
http      2850  \_ nginx: worker process
http      2851  \_ nginx: worker process
root     29348 nginx: master process /usr/sbin/nginx
http     29349  \_ nginx: worker process
http     29350  \_ nginx: worker process
http     29351  \_ nginx: worker process
http     29352  \_ nginx: worker process
```

ss結果

```
sudo ss -N test01 -ltp
State      Recv-Q Send-Q    Local Address:Port    Peer Address:Port
LISTEN     0      128    *:http    *:*    users:(("nginx",pid=2851,fd=6))
LISTEN     0      128    :::http    :::*    users:(("nginx",pid=2851,fd=7))

sudo ss -N test02 -ltp
State      Recv-Q Send-Q    Local Address:Port    Peer Address:Port
LISTEN     0      128    *:http    *:*    users:(("nginx",pid=29352,fd=6))
LISTEN     0      128    :::http    :::*    users:(("nginx",pid=29352,fd=7))
```

curl

curl 172.17.0.1
 test01 container nginx

curl 172.17.0.2
 test02 container nginx

SSH

ssh 172.17.0.1
  cat /etc/debian_version
  8.8

ssh 172.17.0.2
  cat /etc/debian_version
  stretch/sid

まとめると...

- image内のライブラリバージョンで動作
- コンテナ内で複数のアプリケーションを起動
- 異なるIPアドレスで通信
- 簡単なアプリケーションを実行するぐらいはできそう
- 比較的枯れているものしか使っていない + シンプルなので安定はしていそう

いけてない箇所

- いけてない箇所の既存コンテナ技術での解決法と自分で実装する際の解決方を比較すると楽しい
  - image管理
  - Netowork構成
  - PIDの管理, プロセスの処理方法
  - リソース制限
  - セキュリティ

- imageの管理機能
    - snapshotや、バージョニングは?
    - imageの移動はどうする?
- Network構成
    - サブネットも固定でIPも手動なので移動はどうする？
    - 異なるサブネットとの通信は?
- PID分離、プロセス処理
    - PIDは分離する or しない?
    - コンテナ内のinitの処理
    - システムコンテナ? アプリケーションコンテナ?

まとめ

- コンテナ自作は気軽にできる
  - 何がコンテナかという話はあるが
    namespaceは気軽に使える
- 既存コンテナ技術への理解が深まる
  - たりないところも見えてくる
- 一回は触っておくと良さそう