

クラウド基盤構築演習

第一部

クラウド基盤を支えるインフラ技術

～ 第8回 サーバ仮想化環境構成・管理演習

ver1.1 2012/05/01

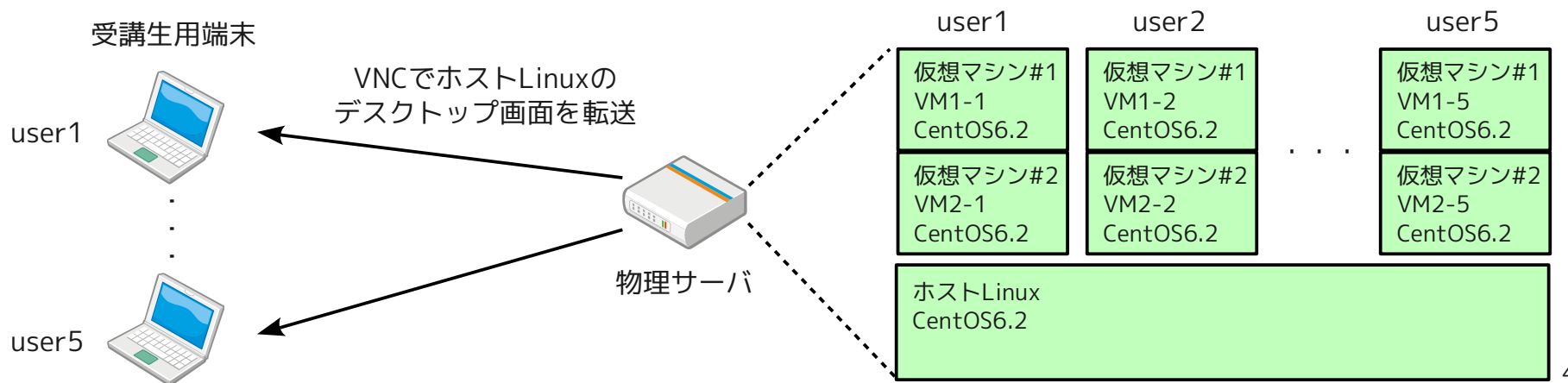
# 目次

- 仮想マシン構成演習
- 仮想ネットワーク構成演習
- cgroupsによるリソース制御演習
- (オプション) LXCによるコンテナ構成演習

# 演習環境の説明

# 演習環境 (1)

- 受講生（最大）5名ごとに演習用の物理サーバが割り当てられています。
  - 各受講生は、自分に割り当てられた「物理サーバ（IPアドレス）」と「ログインユーザ（user1～user5）」を確認してください。
- 各物理サーバには、ホストLinuxとして、CentOS6.2が導入されています。このホストLinuxのデスクトップ画面をVNCで受講生用端末に表示して演習を行います。
  - VNC接続の方法は、別途インストラクタよりガイドがあります。
- この演習では、Linux KVMによる仮想化環境を利用して、CentOS6.2をゲストOSとする仮想マシンを「受講生1名につき2台」作成します。
  - 各受講生は自分が作成する仮想マシンについて、「仮想マシン名、ホストネーム、IPアドレス」の割り当てルール（次ページ参照）を確認してください。

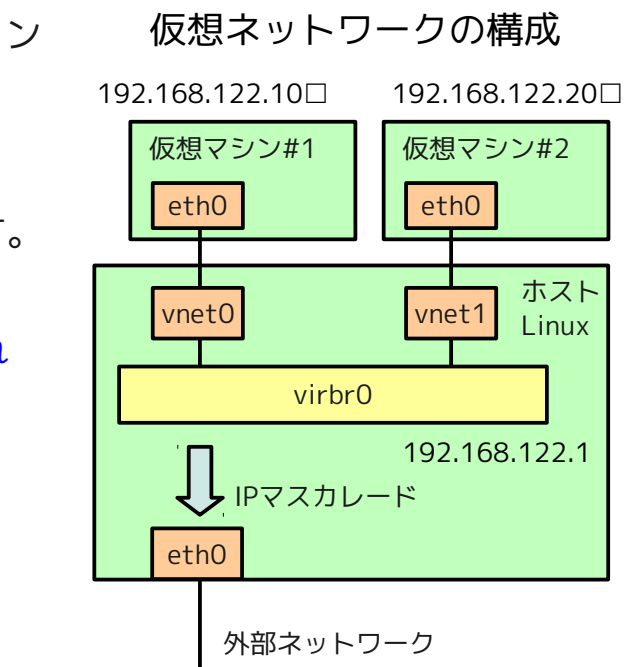


## 演習環境 (2)

- 演習で作成する仮想マシンは、ホストLinux上の仮想ブリッジによるプライベートネットワークに接続されます。
  - 仮想マシンから外部ネットワークには、IPマスカレードで接続します。外部ネットワークから仮想マシンに接続することはできません。
  - ホストLinuxから仮想マシンにログインすることは可能です。
- 仮想マシンを使用する際は、次のどちらかで接続します。
  - ホストLinuxで「virt-manager」を起動して、仮想マシンのコンソール画面を開く。
  - ホストLinuxから仮想マシンにSSHでログインする
  - 仮想マシン名、ホストネーム、IPアドレスは下表を使用します。  
□には、割り当てられたユーザ番号 (1~5) が入ります。

※演習手順において、□で示された部分も同様にユーザ番号 (1~5) を入れてください。

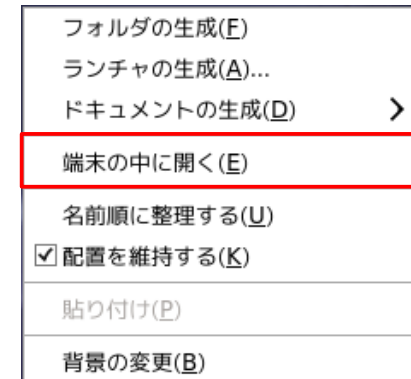
| 仮想マシン名           | ホストネーム | IPアドレス<br>/ネットマスク                 | デフォルトゲートウェイ   |
|------------------|--------|-----------------------------------|---------------|
| 仮想マシン#1<br>VM1-□ | vm1-□  | 192.168.122.10□<br>/255.255.255.0 | 192.168.122.1 |
| 仮想マシン#2<br>VM2-□ | vm2-□  | 192.168.122.20□<br>/255.255.255.0 | 192.168.122.1 |



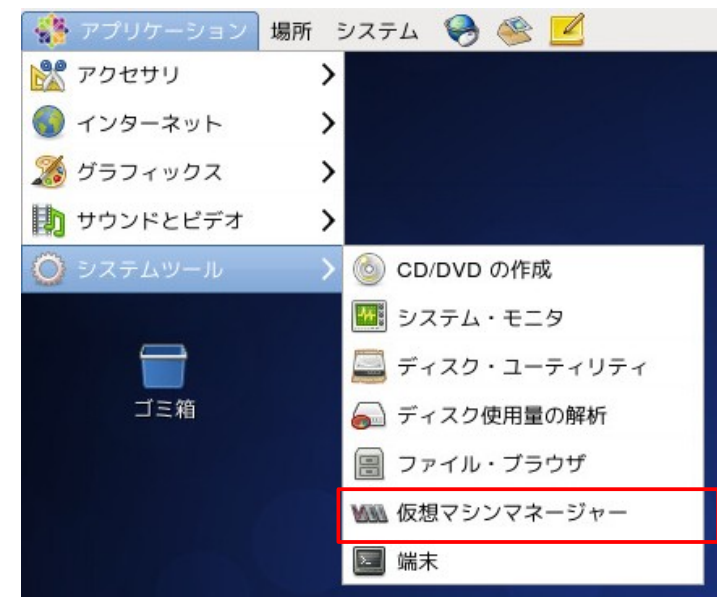
## 演習環境 (3)

- ホストLinuxでコマンド端末を開くには、デスクトップを右クリックして「端末の中を開く」を選択します。
- 「virt-manager」を起動するには、コマンド端末で「virt-manager」を実行するか、デスクトップ左上の「アプリケーション」メニューから「システムツール→仮想マシンマネージャー」を選択します。
- 「Firefox」を起動するには、コマンド端末で「firefox」を実行するか、デスクトップ上部のアイコン（「システム」メニューの右横）をクリックします。
- ホストLinux上では、CentOS6.2のインストールメディアの内容がHTTPで公開されています。ホストLinuxのFirefoxから次のURLにアクセスして、内容を確認してください。
  - <http://192.168.122.1/repo>

デスクトップの  
右クリックメニュー



デスクトップのアプリケーションメニュー



# 仮想マシン構成演習

# 演習内容

- この演習では、次の作業を行います。
  - 仮想マシン#1 (VM1-□) のCPUモデルをデフォルトから変更します。
    - ゲストOSから見えるCPUの機能フラグが変化することを確認します。
  - 仮想マシン#1 (VM1-□) の仮想CPUと仮想メモリの割り当てを変更します。
    - virt-managerを使用する方法とvirshコマンドを使用する方法を行います。
  - 仮想マシン#1 (VM1-□) に新規の仮想ディスクを接続します。
    - virt-managerを使用して行います。



# CPUモデルの変更 (1)

- 仮想マシン#1 (VM1-□) のCPUモデルをデフォルトから変更します。
  - ホストLinuxのコマンド端末で、ホストLinuxから見える物理CPUの機能フラグを確認します。

```
# cat /proc/cpuinfo | grep flags | head -1
flags      : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat pse36 clflush
dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs
bts rep_good xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx smx
est tm2 ssse3 cx16 xtpr pdcm sse4_1 sse4_2 x2apic popcnt aes xsave avx lahf_lm ida arat epb
xsaveopt pln pts dts tpr_shadow vnmi flexpriority ept vpid
```

- これは「第2世代Core i5プロセッサ (Westmere)」での出力例です。

- VM1-□にログインして、ゲストOSから見える仮想CPUの機能フラグを確認します。

```
[root@vm1-□ ~]# cat /proc/cpuinfo | grep flags | head -1
flags      : fpu de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pse36 clflush mmx fxsr
sse sse2 syscall nx lm unfair_spinlock pni cx16 hypervisor lahf_lm
```

- 物理CPUが持つ機能フラグがすべて見えているわけではありません。

- VM1-□を停止します。

```
[root@vm1-□ ~]# poweroff
```

# CPUモデルの変更 (2)

- ホストLinuxでvirt-managerを起動して、VM1-□の仮想コンソールを開いた後、左上の ⓘ ボタンを押して仮想マシンの構成画面を表示します。
- 左の「Processor」メニューを選択して、「Configuration」を開きます。「Copy host CPU configuration」を押すと、「モデル」に物理CPUのモデル名が入ります。
  - 「CPU Features」を開くと各機能フラグを個別に設定することもできます。
- 「適用」を押した後に、▶ ボタンを押してVM1-□を起動します。
- VM1-□にログインして、ゲストOSから見える仮想CPUの機能フラグを確認します。

```
[root@vm1-□ ~]# cat /proc/cpuinfo | grep flags | head -1
flags          : fpu vme de pse tsc msr pae mce cx8 apic
mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ss
syscall nx lm constant_tsc unfair_spinlock pni ssse3 cx16
sse4_1 sse4_2 x2apic popcnt aes hypervisor lahf_lm
```

- ゲストOSから見える機能フラグが増えていることが分かります。
- VM1-□を停止します。

```
[root@vm1-□ ~]# poweroff
```

- 以上で「CPUモデルの変更」は完了です。



# 仮想CPUと仮想メモリの割り当て変更 (1)

- 仮想マシン#1 (VM1-□) の仮想CPUと仮想メモリの割り当てを変更します。
  - ホストLinuxでvirt-managerを起動して、VM1-□の仮想マシン構成画面を表示します。
  - 左の「Processor」メニューを選択して、「Current allocation」を「4」に変更して「適用」を押します。
  - 左の「Memory」メニューを選択して、「Current allocation」を「2048」に変更して「適用」を押します。



# 仮想CPUと仮想メモリの割り当て変更 (2)

- VM1-□を起動してログインした後に、ゲストOSから認識されている仮想CPUの個数とメモリ量を確認します。

```
[root@vm1-□ ~]# cat /proc/cpuinfo | grep processor
processor      : 0
processor      : 1
processor      : 2
processor      : 3

[root@vm1-□ ~]# free
              total        used         free       shared    buffers     cached
Mem:          2054688      198820      1855868           0       10524       65880
-/+ buffers/cache:      122416      1932272
Swap:          524280           0         524280
```

- VM1-□を停止します。

```
[root@vm1-□ ~]# poweroff
```

- ホストLinuxで、rootユーザからvirshコマンドで仮想マシン名を確認します。

```
# virsh list --all
Id 名前                状態
-----
- VM1-□              シャットオフ
- VM2-□              シャットオフ
```

- rootユーザから仮想マシン「VM1-□」の構成情報を編集します。

```
# virsh edit VM1-□
```

- VM1-□の構成情報のXMLファイルがviエディタで開きます。

# 仮想CPUと仮想メモリの割り当て変更 (3)

- メモリ容量と仮想CPU数の値を下記のように変更して保存した後に、viエディタを終了します。

```
<domain type='kvm'>
  <name>VM1-□</name>
  <uuid>10926b90-1fed-d301-a8c4-63111eb8f88a</uuid>
  <memory>1048576</memory>          ← この行を修正
  <currentMemory>1048576</currentMemory> ← この行を修正
  <vcpu>2</vcpu>                    ← この行を修正
  <os>
```

- virshコマンドでVM1-□を起動します。

```
# virsh start VM1-□
ドメイン VM1-□ が起動されました
```

- VM1-□にログインして、ゲストOSから認識されている仮想CPUの個数とメモリ量を確認します。

```
[root@vm1-□ ~]# cat /proc/cpuinfo | grep processor
processor      : 0
Processor     : 1

[root@vm1-□ ~]# free
              total        used         free      shared    buffers     cached
Mem:          1020692     182036     838656           0       10532      65856
-/+ buffers/cache:      105648     915044
Swap:          524280           0      524280
```

- VM1-□を停止します。

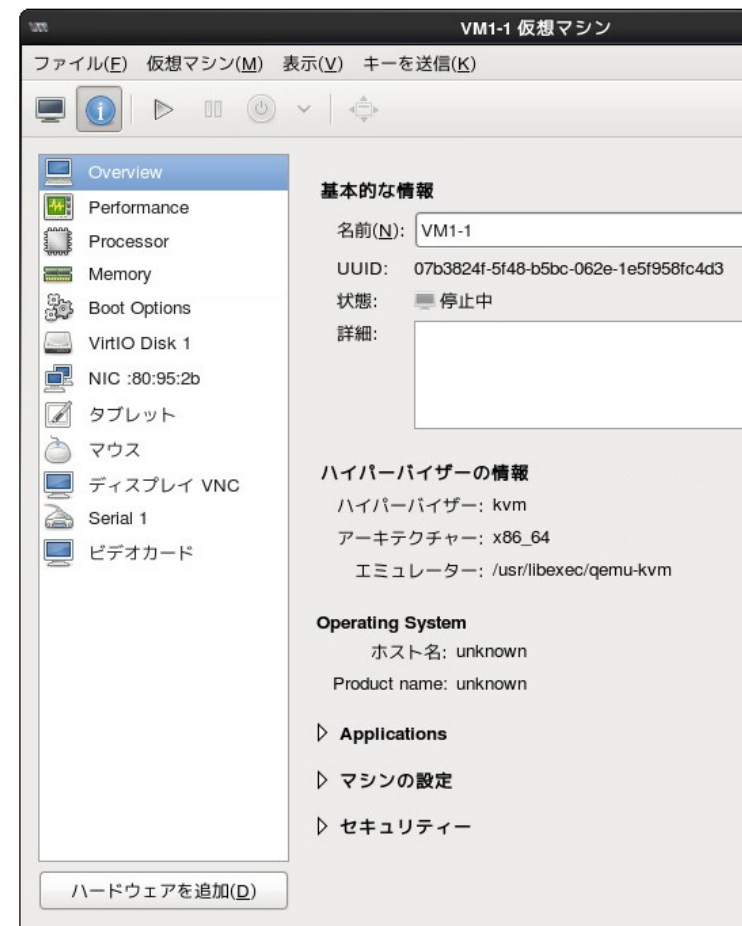
```
[root@vm1-□ ~]# poweroff
```

- 以上で「仮想CPUと仮想メモリの割り当て変更」は完了です。

# 新規仮想ディスクの接続 (1)

## ■ 仮想マシン#1 (VM1-□) に新規の仮想ディスクを接続します。

- ホストLinuxでvirt-managerを起動して、VM1-□の仮想マシン構成画面を表示します。
- 「ハードウェアを追加」を押します。
- 左のメニューで「Storage」を選んで、下図のように指定して「完了」を押します。
  - デバイスの種類に「Virtio Disk」を選択するとvirtio形式のエミュレーションが行われます。



# 新規仮想ディスクの接続 (2)

- 仮想マシン構成画面に「VirtIO Disk 2」が追加されたことを確認して、VM1-□を起動します。
- VM1-□にログインして、ゲストOSから認識されているディスクを確認します。

```
[root@vm1-□ ~]# lsblk -l
NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda   252:0    0    8G  0 disk
vda1  252:1    0   200M  0 part /boot
vda2  252:2    0   512M  0 part [SWAP]
vda3  252:3    0   7.3G  0 part /
vdb   252:16   0    8G  0 disk
```

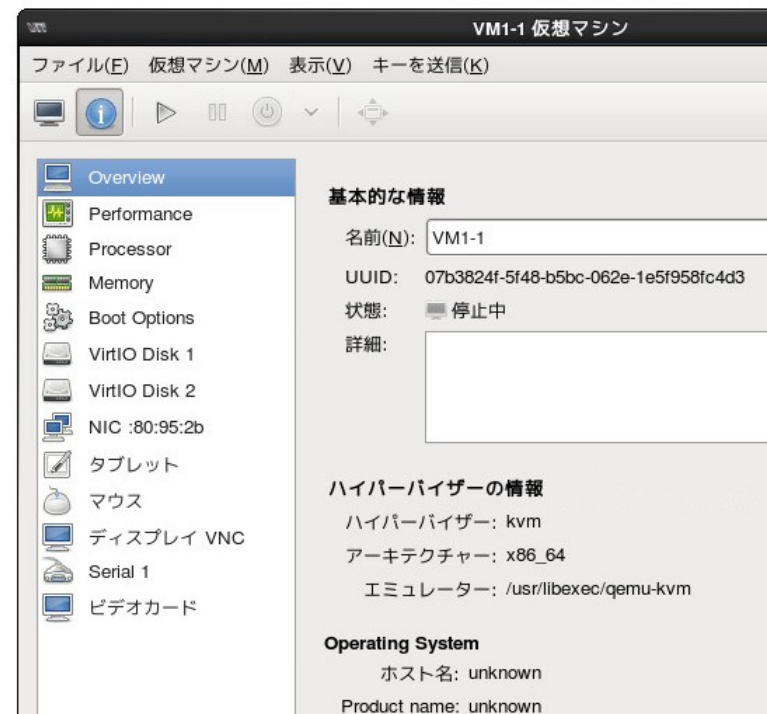
- 8GBの「vdb」が追加されていることが分かります。

- VM1-□を停止します。

```
[root@vm1-□ ~]# poweroff
```

- 以上で「新規仮想ディスクの接続」は完了です。

- 以上で「仮想マシン構成演習」は完了です。



メモとしてお使いください

[illegible]



メモとしてお使いください

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

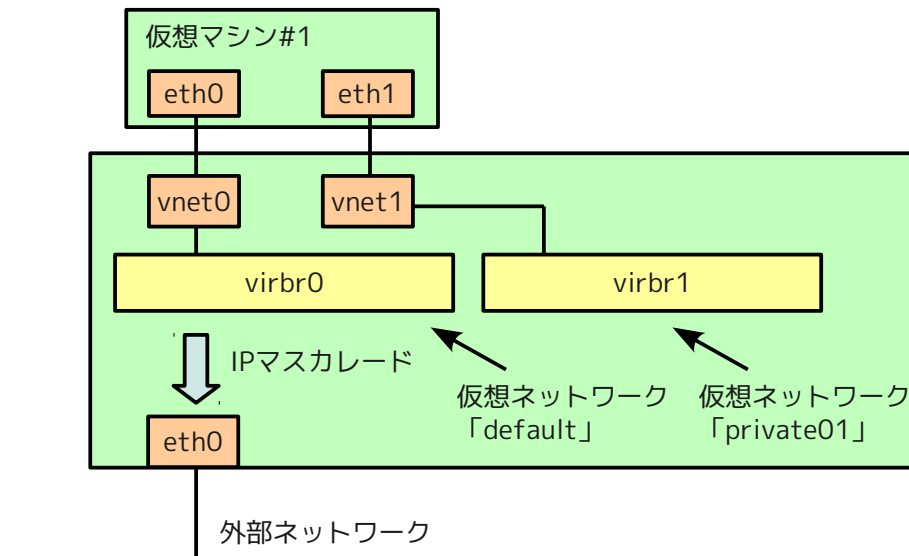
メモとしてお使いください

[illegible]

# 仮想ネットワーク構成演習


# 演習内容

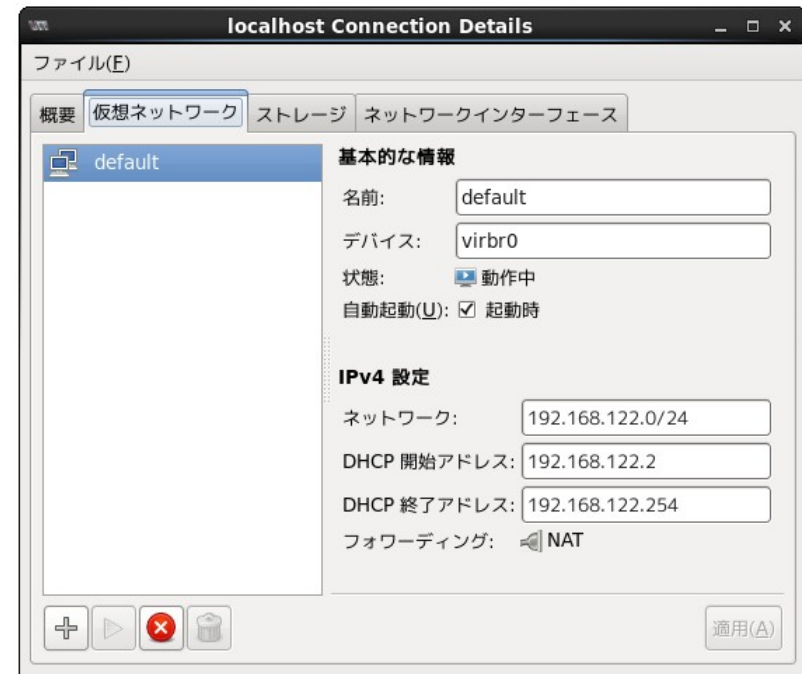
- この演習では、次の作業を行います。
  - 新規の仮想ネットワーク「private01」を作成します。
    - virt-managerを使用して行います。
  - 仮想マシン#1（VM1-□）に仮想NICを追加して、「private01」に接続します。
    - virt-managerを使用して行います。
  - 仮想ネットワークと仮想ブリッジの関係を確認します。
    - ホストLinux上で仮想ブリッジ、およびTAPデバイスの状態を確認します。



# 新規仮想ネットワークの作成 (1)

この節の作業はuser1の受講生が代表して実施します。

- 新規の仮想ネットワーク「private01」を作成します。
  - ホストLinuxでvirt-managerを起動して「localhost(QEMU)」をダブルクリックして、ホストLinuxの構成画面を開きます。
  - 「仮想ネットワーク」のタブを選択して、 ボタンを押します。
  - 新規仮想ネットワーク作成のウィザード画面が開くので、「進む」で次に進みます。



# 新規仮想ネットワークの作成 (2)

この節の作業はuser1の受講生が代表して実施します。

- 仮想ネットワークの名前の指定画面では「private01」を入力して次に進みます。
- IPアドレス範囲の指定画面では「192.168.100.0/24」のまま次に進みます。
- DHCP範囲の選択画面では「192.168.100.128～192.168.100.254」のまま次に進みます。
  - この仮想ネットワークにDHCP機能を提供しない場合は「DHCPを有効に」のチェックをはずします。

**新規の仮想ネットワークを作成**

**仮想ネットワークの名前の指定**

仮想ネットワークの名前を選択してください:

ネットワーク名(N):

例: network1

**IPv4 アドレス領域の指定**

仮想ネットワークで利用する IPv4 アドレス範囲を選択してください:

ネットワーク(N):

ヒント: ネットワークは、IPv4 プライベートアドレスの範囲から選択する必要があります。例 10.0.0.0/8、172.16.0.0/12、192.168.0.0/16

ネットマスク: 255.255.255.0  
ブロードキャスト: 192.168.100.255  
ゲートウェイ: 192.168.100.1  
サイズ: 256 アドレス  
種類: プライベート

キャンセル(C) 戻る(B) 進む(E)

**DHCP 範囲の選択**

DHCP サーバーが仮想ネットワークに接続された仮想マシンに割り当てるアドレスの範囲を選択してください。

DHCPを有効に(E) ☒

開始(S):

終了(N):

Tip: 仮想マシンに静的ネットワークを設定できるように一部のアドレスを予約しておきたい場合を除いて、これらのパラメータはデフォルト値のままにしておいて構いません。

キャンセル(C) 戻る(B) 進む(E)

# 新規仮想ネットワークの作成 (3)

この節の作業はuser1の受講生が代表して実施します。

- 物理ネットワークへの接続画面では、「隔離された仮想ネットワーク」のまま次に進みます。
  - これは、ホストLinux上で、この仮想ネットワークから、外部ネットワーク/他の仮想ネットワークへのパケット転送を行わないこと意味します。つまり、この仮想ネットワークに接続した仮想マシン間のみ通信が可能になります。ただし、ホストLinuxとの通信は可能です。
- 設定内容の確認画面が表示されるので「完了」を押します。

新規の仮想ネットワークを作成

### 物理ネットワークへの接続

この仮想ネットワークを物理ネットワークに接続するかどうか指定してください。

☒ 隔離された仮想ネットワーク(I)  
☐ 物理ネットワークにフォワード(W)

宛先(D): いずれかの物理デバイス

モード(M): NAT

キャンセル(C)    戻る(B)    進む(F)

新規の仮想ネットワークを作成

### ネットワークを作成する準備ができました

サマリー

ネットワーク名: private01

IPv4 ネットワーク

ネットワーク: 192.168.100.0/24  
 ゲートウェイ: 192.168.100.1  
 ネットマスク: 255.255.255.0

DHCP

開始アドレス: 192.168.100.128  
 終了アドレス: 192.168.100.254

フォワーディング

接続: 隔離されたネットワーク

キャンセル(C)    戻る(B)    完了(F)

# 新規仮想ネットワークの作成 (4)

この節の作業はuser1の受講生が代表して実施します。

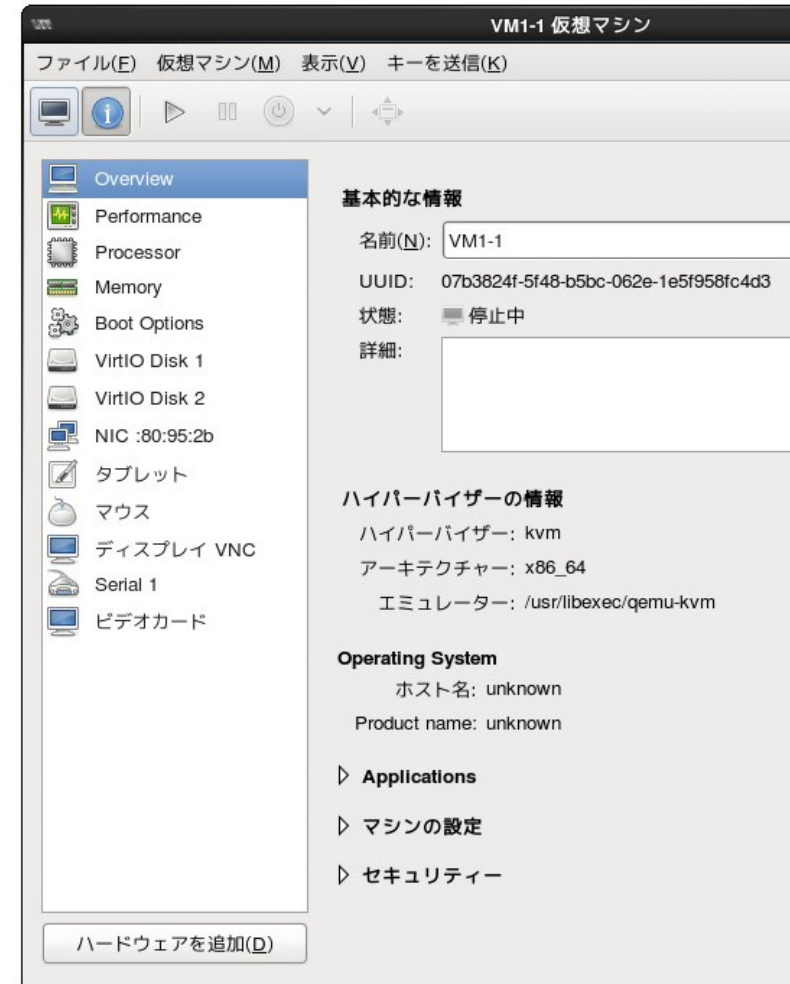
- ホストLinuxの構成画面に仮想ネットワーク「private01」が追加されていることを確認します。
- 以上で「新規仮想ネットワークの作成」は完了です。
  - ここで、ホストLinuxの構成画面は閉じて構いません。





# 新規仮想NICの追加 (1)

- 仮想マシン#1 (VM1-□) に仮想NICを追加して、「private01」に接続します。
  - ホストLinuxでvirt-managerを起動して、VM1-□の仮想マシン構成画面を表示します。
  - 「ハードウェアを追加」を押します。
  - 左のメニューで「Network」を選んで、下図のように指定して「完了」を押します。
    - デバイスモデルに「virtio」を選択するとvirtio形式のエミュレーションが行われます。



# 新規仮想NICの追加 (2)

- 仮想マシン構成画面に2つ目の「NIC」が追加されたことを確認して、VM1-□を起動します。
- VM1-□にログインして、「eth1」が認識されていることを確認します。

```
[root@vm1-□ ~]# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 52:54:00:C5:23:13
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:0 (0.0 b)
```

- 「/etc/sysconfig/network-scripts/ifcfg-eth1」を次の内容で作成します。

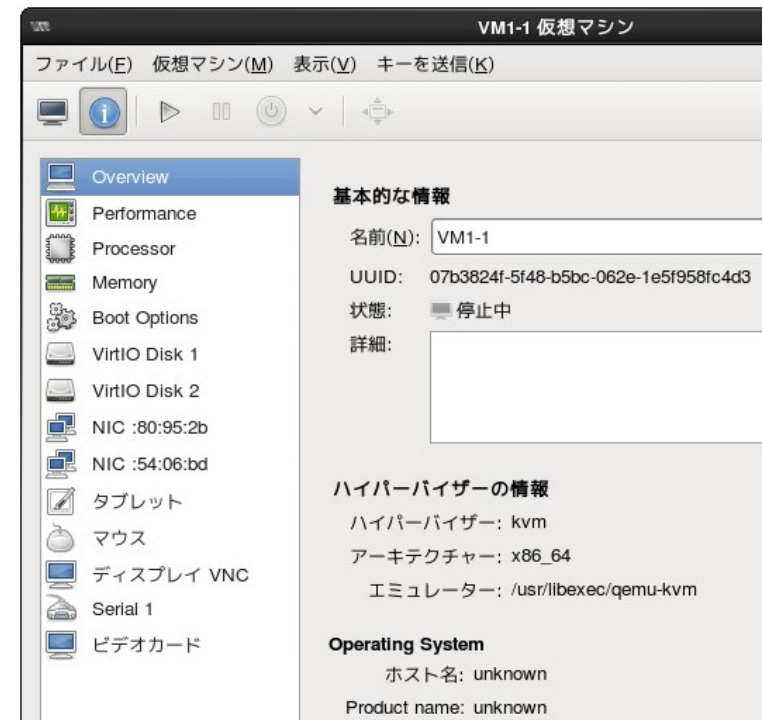
/etc/sysconfig/network-scripts/ifcfg-eth1

```
DEVICE=eth1
BOOTPROTO=dhcp
NM_CONTROLLED=no
ONBOOT=yes
TYPE=Ethernet
```

- eth1を有効化します。

```
[root@vm1-□ ~]# ifup eth1

eth1 のIP情報を検出中... 完了。
```



# 新規仮想NICの追加 (3)

- DHCPでeth1にIPアドレスがアサインされていることを確認します。

```
[root@vm1-□ ~]# ifconfig eth1
eth1      Link encap:Ethernet  HWaddr 52:54:00:C5:23:13
          inet addr:192.168.100.253  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:fec5:2313/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:129 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7528 (7.3 KiB)  TX bytes:1124 (1.0 KiB)
```

- 以上で「新規仮想NICの追加」は完了です。

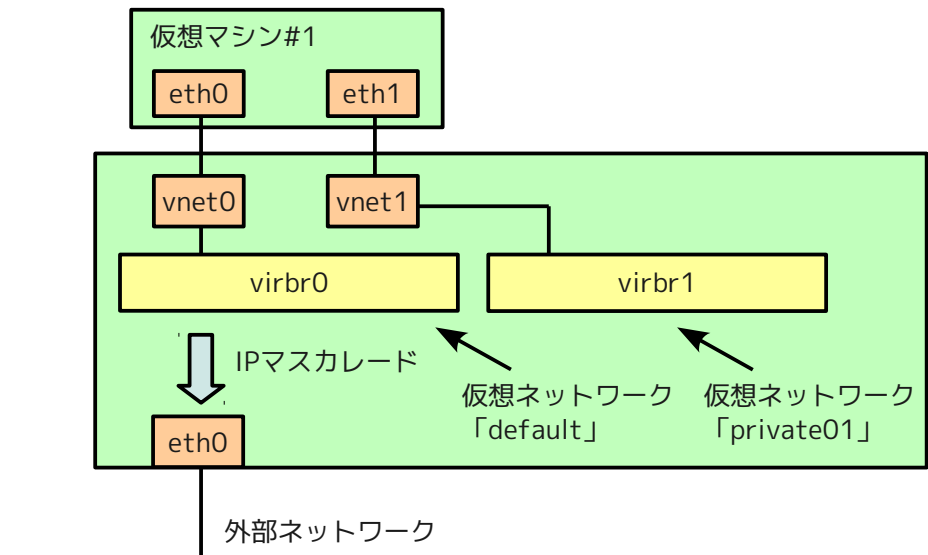
# 仮想ブリッジの状態確認 (1)

## ■ 仮想ネットワークと仮想ブリッジの関係を確認します。

– ホストLinuxで仮想ブリッジの状態を確認します。

```
# brctl show
bridge name      bridge id                STP enabled  interfaces
virbr0           8000.5254001889ef        yes          virbr0-nic
                                                         vnet0
virbr1           8000.525400cd2e90        yes          virbr1-nic
                                                         vnet1
```

- 「virbr0」「virbr1」の2個の仮想ブリッジがあることが分かります。
- 「interfaces」はこのブリッジに接続されているデバイスで、「vnet0」「vnet1」は図のTAPデバイスになります。



# 仮想ブリッジの状態確認 (2)

- 仮想ネットワークと仮想ブリッジの対応を確認します。ホストLinuxのrootユーザで次のコマンドを実行します。
  - 最初のコマンドは仮想ネットワークの一覧を表示しています。
  - 次のコマンドは、それぞれの仮想ネットワークの構成情報をXML形式で出力しています。
  - 赤囲みの部分から対応する仮想ブリッジの名前が分かります。

```
# virsh net-list --all
名前                状態      自動起動
-----
default              動作中    yes
private01             動作中    yes

# virsh net-dumpxml default
<network>
  <name>default</name>
  <uuid>63e94d28-5bc0-4752-80bc-ab8d09b7bc23</uuid>
  <forward mode='nat' />
  <bridge name='virbr0' stp='on' delay='0' />
  <mac address='52:54:00:18:89:EF' />
  <ip address='192.168.122.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.122.2' end='192.168.122.254' />
    </dhcp>
  </ip>
</network>

# virsh net-dumpxml private01
<network>
  <name>private01</name>
  <uuid>6eaf2a41-140e-0073-46ea-fd93ed829995</uuid>
  <bridge name='virbr1' stp='on' delay='0' />
  <mac address='52:54:00:FE:CC:50' />
  <ip address='192.168.100.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.100.128' end='192.168.100.254' />
    </dhcp>
  </ip>
</network>
```

# 仮想ブリッジの状態確認 (3)

- ホストLinuxで仮想ブリッジのIPアドレスを確認します。

```
# ifconfig virbr0
virbr0    Link encap:Ethernet  HWaddr 52:54:00:18:89:EF
          inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:152110 errors:0 dropped:0 overruns:0 frame:0
          TX packets:264118 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:16201062 (15.4 MiB)  TX bytes:743220547 (708.7 MiB)

# ifconfig virbr1
virbr1    Link encap:Ethernet  HWaddr 52:54:00:FE:CC:50
          inet addr:192.168.100.1  Bcast:192.168.100.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:7 errors:0 dropped:0 overruns:0 frame:0
          TX packets:38 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:916 (916.0 b)  TX bytes:6919 (6.7 KiB)
```

- VM2-□を起動します。

```
# virsh start VM2-□
ドメイン VM2-□ が起動されました
```

- 再度、仮想ブリッジの状態を確認します。

```
# brctl show
bridge name      bridge id                STP enabled  interfaces
virbr0           8000.5254001889ef       yes          virbr0-nic
                 vnet0
                 vnet2
virbr1           8000.525400cd2e90       yes          virbr1-nic
                 vnet1
```

- virbr0にVM2-□のTAPデバイス「vnet2」が追加で接続されていることが分かります。

## 仮想ブリッジの状態確認 (4)

- 以上で「仮想ブリッジの状態確認は完了」です。
- 以上で「仮想ネットワーク構成演習」は完了です。

メモとしてお使いください

[illegible]



メモとしてお使いください

[illegible]

メモとしてお使いください

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

# cgroupsによるリソース制御演習

# 演習内容

- この演習では、次の作業を行います。
  - cgroupsにより、仮想マシン#1 (VM1-□) と仮想マシン#2 (VM2-□) のCPU使用を制限します。
    - VM1-□、VM2-□が使用する物理CPUを制限します。
    - VM1-□、VM2-□に対するCPUの優先順位の設定を行います。
    - VM1-□、VM2-□に対するCPU利用率の上限設定を行います。
  - cgroupsにより、仮想マシン#1 (VM1-□) と仮想マシン#2 (VM2-□) に対するI/O帯域の上限設定を行います。

# 使用する物理CPUの制限 (1)

この節の作業はuser1の受講生が代表して実施します。

- 仮想マシン#1 (VM1-□) と仮想マシン#2 (VM2-□) のCPU使用を制限します。
  - VM1-□とVM2-□が起動していない場合は、ここで起動します。
  - コマンド端末からVM1-□、VM2-□にログインして、それぞれ、CPUを使用しつづけるプロセスを起動します。

```
[root@vm1-□ ~]# cat /dev/zero > /dev/null
```

```
[root@vm2-□ ~]# cat /dev/zero > /dev/null
```

- ホストLinuxでCPUの使用状況を確認します。

```
# top

top - 23:07:24 up 25 days, 11:12,  4 users,  load average: 2.03, 2.08, 1.30
Tasks: 288 total,   2 running, 286 sleeping,   0 stopped,   0 zombie
Cpu0  : 98.3%us,  1.7%sy,   0.0%ni,   0.0%id,   0.0%wa,   0.0%hi,   0.0%si,   0.0%st
Cpu1  :  5.6%us,  0.7%sy,   0.0%ni,  92.4%id,   1.3%wa,   0.0%hi,   0.0%si,   0.0%st
Cpu2  :100.0%us,  0.0%sy,   0.0%ni,   0.0%id,   0.0%wa,   0.0%hi,   0.0%si,   0.0%st
Cpu3  :  5.0%us,  0.7%sy,   0.0%ni,  94.4%id,   0.0%wa,   0.0%hi,   0.0%si,   0.0%st
Mem:   7937076k total, 2999044k used, 4938032k free,  373696k buffers
Swap:  2097144k total,   2808k used, 2094336k free,  925436k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 31901 qemu      20   0 1376m 370m 3672 S 100.4   4.8    7:54.61 qemu-kvm
 26566 qemu      20   0 1377m 392m 3692 S 100.1   5.1    9:09.24 qemu-kvm
 14657 root       20   0  768m  85m  15m S   8.0   1.1   11:30.07 python
 12244 root       20   0 1042m  27m 5608 S   3.0   0.4   58:46.53 libvirtd
```

- 「1」を押してCPUごとの使用率を表示します。
- 2個のqemu-kvmプロセスが異なる物理CPUを専有していることが分かります。

# 使用する物理CPUの制限 (2)

この節の作業はuser1の受講生が代表して実施します。

- ホストLinuxのrootユーザーで次を実行して、VM1-□が使用する物理CPUをCPU#1のみに制限します。

```
# cd /cgroup/cpuset/libvirt/qemu/VM1-□
# echo "1" > cpuset.cpus
# cat cpuset.cpus
1
```

- 同様に、VM2-□が使用する物理CPUをCPU#1のみに制限します。

```
# cd /cgroup/cpuset/libvirt/qemu/VM2-□
# echo "1" > cpuset.cpus
# cat cpuset.cpus
1
```

- ホストLinuxでCPUの使用状況を確認します。

```
# top

top - 23:12:09 up 25 days, 11:17, 5 users, load average: 2.22, 2.12, 1.51
Tasks: 293 total, 2 running, 291 sleeping, 0 stopped, 0 zombie
Cpu0  : 2.0%us, 0.7%sy, 0.0%ni, 96.7%id, 0.7%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu1  :100.0%us, 0.0%sy, 0.0%ni, 0.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu2  : 7.3%us, 0.7%sy, 0.0%ni, 92.1%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu3  : 1.3%us, 0.0%sy, 0.0%ni, 97.3%id, 1.3%wa, 0.0%hi, 0.0%si, 0.0%st
Mem:   7937076k total, 3004116k used, 4932960k free, 373768k buffers
Swap:  2097144k total, 2808k used, 2094336k free, 925528k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 31901 qemu       20   0 1376m 370m 3672 S  50.2   4.8   12:26.09 qemu-kvm
 26566 qemu       20   0 1377m 392m 3692 S  49.9   5.1   13:40.94 qemu-kvm
 14657 root        20   0  768m  85m  15m S   8.3   1.1   11:53.11 python
 12244 root        20   0 1042m  27m 5608 S   3.3   0.4   58:54.92 libvirtd
```

- 2個のqemu-kvmプロセスが共にCPU1を占有して、それぞれの使用率が50%になっていることが分かります。

# 使用する物理CPUの制限 (3)

この節の作業はuser1の受講生が代表して実施します。

- ホストLinuxのrootユーザで次を実行して、VM1-□のCPU優先順位をデフォルト値の2倍に変更します。

```
# cd /cgroup/cpu/libvirt/qemu/VM1-□
# cat cpu.shares
1024
# echo "2048" > cpu.shares
# cat cpu.shares
2048
```

- ホストLinuxでCPUの使用状況を確認します。

```
# top

top - 23:16:34 up 25 days, 11:21, 5 users, load average: 2.34, 2.26, 1.72
Tasks: 293 total, 3 running, 290 sleeping, 0 stopped, 0 zombie
Cpu0  :  2.0%us,  0.7%sy,  0.0%ni, 96.4%id,  1.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  : 100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  7.0%us,  0.7%sy,  0.0%ni, 92.4%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  1.7%us,  0.3%sy,  0.0%ni, 96.7%id,  1.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   7937076k total, 3004464k used, 4932612k free, 373828k buffers
Swap:  2097144k total,  2808k used, 2094336k free, 925764k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 26566 qemu      20   0 1377m 392m 3692 S   66.8   5.1   15:55.12 qemu-kvm
 31901 qemu      20   0 1376m 370m 3672 S   33.2   4.8   14:36.95 qemu-kvm
 14657 root       20   0  768m  85m  15m R    8.3   1.1   12:14.63 python
 12244 root       20   0 1042m  27m 5608 S    3.0   0.4   59:02.86 libvirtd
```

- 2個のqemu-kvmプロセスがCPU1を占有していますが、それぞれの使用率の割合が1:2に分かれています。

# 使用する物理CPUの制限 (4)

この節の作業はuser1の受講生が代表して実施します。

- ホストLinuxのrootユーザで次を実行して、VM1-□のCPU優先順位をデフォルト値に戻します。

```
# cd /cgroup/cpu/libvirt/qemu/VM1-□
# echo "1024" > cpu.shares
```

- ホストLinuxのrootユーザで次を実行して、VM1-□のCPU使用時間の上限を10%に設定します。  
(「cpu.cfs\_quota\_us」 = 「cpu.cfs\_period\_us」 \* 10%)

```
[root@kakinoha VM1-□]# cd /cgroup/cpu/libvirt/qemu/VM1-□
[root@kakinoha VM1-□]# cat cpu.cfs_period_us
100000
[root@kakinoha VM1-□]# echo "10000" > cpu.cfs_quota_us
```

- ホストLinuxでCPUの使用状況を確認します。

```
[root@kakinoha ~]# top

top - 23:20:26 up 25 days, 11:25, 5 users, load average: 1.92, 2.19, 1.81
Tasks: 293 total, 3 running, 290 sleeping, 0 stopped, 0 zombie
Cpu0  :  2.0%us,  0.7%sy,  0.0%ni, 97.0%id,  0.3%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu1  :100.0%us,  0.0%sy,  0.0%ni,  0.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu2  :  5.6%us,  0.7%sy,  0.0%ni, 93.7%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Cpu3  :  2.7%us,  0.3%sy,  0.0%ni, 96.0%id,  1.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   7937076k total, 3005760k used, 4931316k free, 373860k buffers
Swap:  2097144k total,  2808k used, 2094336k free, 925824k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU  %MEM    TIME+  COMMAND
 31901 qemu       20   0 1376m 370m 3672 S  90.0   4.8   16:18.38 qemu-kvm
 26566 qemu       20   0 1377m 392m 3692 R  10.3   5.1   18:06.06 qemu-kvm
 14657 root        20   0  768m  85m  15m S   8.3   1.1   12:33.62 python
 12244 root        20   0 1042m  27m 5608 S   3.3   0.4   59:09.76 libvirtd
```

- 2個のqemu-kvmプロセスがCPU1を占有していますが、一方の使用率が10%に制限されています。



## 使用する物理CPUの制限 (5)

この節の作業はuser1の受講生が代表して実施します。

- VM1-□、VM2-□のcatコマンドをCtrl+Cで停止します。
- 以上で「使用する物理CPUの制限」は完了です。

# ディスクI/O帯域の制限 (1)

この節の作業はuser1の受講生が代表して実施します。

- 仮想マシン#1 (VM1-□) と仮想マシン#1 (VM2-□) のディスクI/O帯域を制限します。
  - VM1-□、VM2-□を起動します。
  - ホストLinuxで仮想ディスクイメージを配置したデバイス (sda) のデバイス番号を確認します。

```
[root@centos62 ~]# lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
loop0 7:0 0 4.1G 0 loop /mnt/iso/CentOS62
sda 8:0 0 2.5T 0 disk
├─sda1 8:1 0 4.8M 0 part
├─sda2 8:2 0 465.7G 0 part /
├─sda3 8:3 0 89.4G 0 part [SWAP]
├─sda4 8:4 0 90.3G 0 part
└─sda5 8:5 0 1.8T 0 part /mnt
```

- 「sda」のデバイス番号は「8:0」と分かります。
- ホストLinuxで次を実行して、VM1-□とVM2-□の「sda」に対するI/O帯域の上限をそれぞれ50MB/Secと10MB/Secに設定します。

```
# cd /cgroup/blkio/libvirt/qemu/VM1-□
# echo "8:0 50000000" > blkio.throttle.write_bps_device
# cd /cgroup/blkio/libvirt/qemu/VM2-□
# echo "8:0 10000000" > blkio.throttle.write_bps_device
```

# ディスクI/O帯域の制限 (2)

この節の作業はuser1の受講生が代表して実施します。

- VM1-□にログインして、ディスクI/O速度を確認します。

```
[root@vm1-□ ~]# dd if=/dev/zero of=/tmp/tmp0 bs=1M count=500 oflag=direct
500+0 records in
500+0 records out
524288000 bytes (524 MB) copied, 10.5281 s, 49.8 MB/s
```

- これは、500MBのファイルを作成しています。「oflag=direct」はディスクキャッシュを使用せずに直接にディスクに書き込むオプションです。
  - 約10秒でファイル作成が完了して、50MB/SecのI/O速度になっていることが分かります。
- VM2-□にログインして、同様にディスクI/O速度を確認します。

```
[root@vm2-□ ~]# dd if=/dev/zero of=/tmp/tmp0 bs=1M count=500 oflag=direct
500+0 records in
500+0 records out
524288000 bytes (524 MB) copied, 52.4254 s, 10.0 MB/s
```

- 約50秒でファイル作成が完了して、10MB/SecのI/O速度になっていることが分かります。
- 以上で「ディスクI/O帯域の制限」は完了です。
- 以上で「cgroupsによるリソース制御演習」は完了です。

メモとしてお使いください

[illegible]

メモとしてお使いください

[illegible]

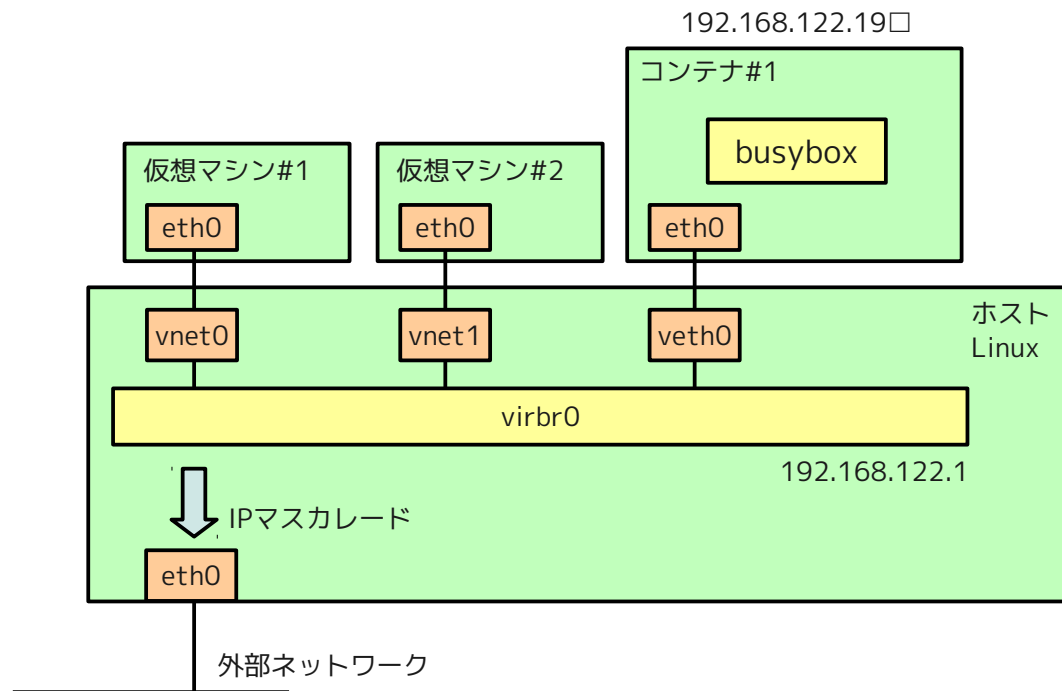
メモとしてお使いください

This image shows a blank sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

# (オプション) LXCによるコンテナ構成演習

# 演習内容

- この演習では、次の作業を行います。
  - ホストLinux上で、LXC（Linuxコンテナ）によるアプリケーションコンテナを作成します。
    - コンテナ内部では、busyboxによる簡易Webサーバを起動します。
    - コンテナの作成には、libvirt（virshコマンド）を使用します。





# 仮想ルートファイルシステムの用意 (1)

- アプリケーションコンテナの仮想ルートファイルシステムを用意します。

- ホストLinuxのrootユーザで、次のコマンドを実行します。

```
# mkdir -p /export/lxcguest□/bin
# mkdir -p /export/lxcguest□/html
# cd /export/lxcguest□/bin
# cp /sbin/busybox ./
# for i in echo grep ifconfig kill ps route test which cat false head ls pwd sh true date
find httpd ip ping rm sleep wget; do ln -s busybox $i; done
```

- ここでは、busyboxの動作に必要な最小限の環境を「/export/lxcguest□」以下に用意しています。
- コンテナの起動時に最初に実行するスクリプト「/export/lxcguest□/bin/init」を次の内容で作成します。

/export/lxcguest□/bin/init

```
#!/bin/sh
ifconfig eth0 192.168.122.19□
route add default gw 192.168.122.1 eth0
httpd -h /html
while [[ true ]]; do
    sh
done
```

- これは、IPアドレスの設定と簡易Webサーバの起動を行います。最後の「sh」は、仮想コンソールで使用するシェルになります。
- 「/export/lxcguset□/bin/init」に実行権を設定します。

```
# chmod u+x /export/lxcguest□/bin/init
```

# 仮想ルートファイルシステムの用意 (2)

- 簡易Webサーバで公開するHTMLファイル「/export/lxcguest□/html/index.html」を次の内容で用意します。

/export/lxcguest□/html/index.html

```
<h1>Welcome to lxcguest□</h1>
```

- 以上で「仮想ルートファイルシステムの用意」は完了です。

# コンテナの定義と実行 (1)

- アプリケーションコンテナ「lxcguest□」を定義して、起動します。
- コンテナ定義のXMLファイル「/root/work/lxcguest□.xml」を次の内容で作成します。

/root/work/lxcguest□.xml

```
<domain type='lxc'>
  <name>lxcguest□</name>
  <memory>200000</memory>
  <os>
    <type>exe</type>
    <init>/bin/init</init>
  </os>
  <devices>
    <interface type='network'>
      <source network='default'>
    </interface>
    <console type='pty'>
    <filesystem type='mount'>
      <source dir='/export/lxcguest□'>
      <target dir='/'>
    </filesystem>
  </devices>
</domain>
```

- ホストLinuxで次のコマンドにより、コンテナを定義します。

```
# cd /root/work
# virsh -c lxc:/// define lxcguest□.xml
ドメイン lxcguest□ が lxcguest□.xml から定義されました

# virsh -c lxc:/// list --all
Id 名前                状態
-----
- lxcguest□          シャットオフ
```

# コンテナの定義と実行 (2)

- 次のコマンドでコンテナを起動します。

```
# virsh -c lxc:/// start lxcguest□
ドメイン lxcguest□ が起動されました

# virsh -c lxc:/// list --all
Id 名前                      状態
-----
12123 lxcguest□                実行中
```

- コンテナの仮想コンソールに接続して、コンテナ内部の状態を確認します。

```
# virsh -c lxc:/// console lxcguest□
Connected to domain lxcguest□
エスケープ文字は ^] です
# ps -ef
PID    USER      TIME  COMMAND
  1  0          0:00  /bin/sh /bin/init
  8  0          0:00  httpd -h /html
  9  0          0:00  sh
 10  0          0:00  ps -ef

# ls -l /
drwxr-xr-x  2 0          0          4096 Jan 21 00:18 bin
drwxr-xr-x  3 0          0          220 Jan 21 00:18 dev
drwxr-xr-x  2 0          0          4096 Jan  4 05:25 html
dr-xr-xr-x 307 0          0           0 Jan 21 00:18 proc
drwxr-xr-x  7 0          0           0 Dec 26 02:55 selinux
drwxr-xr-x 13 0          0           0 Dec 26 02:55 sys
```

- コンテナ内部では、最初に起動する「/export/lxcguest□/bin/init」がPID 1のプロセスとなり、その子プロセスのみが見えます。
- ファイルシステムは「/export/lxcguset01」がルートファイルシステムとして見えます。

# コンテナの定義と実行 (3)

- コンテナ内部の状態確認の続きです。

```
# ifconfig
eth0      Link encap:Ethernet  HWaddr 52:54:00:FF:76:26
          inet addr:192.168.122.19 Bcast:192.168.122.255  Mask:255.255.255.0
          inet6 addr: fe80::5054:ff:feff:7626/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:51 errors:0 dropped:0 overruns:0 frame:0
          TX packets:6 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3578 (3.4 KiB)  TX bytes:552 (552.0 B)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

# route -n
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
192.168.122.0    0.0.0.0         255.255.255.0    U        0      0        0 eth0
0.0.0.0          192.168.122.1  0.0.0.0          UG       0      0        0 eth0
#
```

← [Ctrl+]を押す

- コンテナ内部には仮想NIC「eth0」がアサインされています。これは、ホストLinux上の「veth0」と接続されています。
- 仮想コンソールへの接続は、「[Ctrl+]」で終了します。

# コンテナの定義と実行 (4)

- ホストLinuxでFirefoxを起動して、URL「http://192.168.122.191/」を開きます。



- コンテナ内部の簡易Webサーバに接続できることが分かります。
- 以上で「コンテナの定義と実行」は完了です。

# ホストLinuxからの確認

- ホストLinux上でコンテナの状態を確認します。
  - ホストLinux上でコンテナを実行するプロセスを確認します。

```
# pstree
...
├─libvirt_lxc──init──httpd
│               └─sh
...
```

- libvirt\_lxcから起動される子プロセスがコンテナ内部から見えるプロセスになります。

- 仮想ブリッジの状態を確認します。

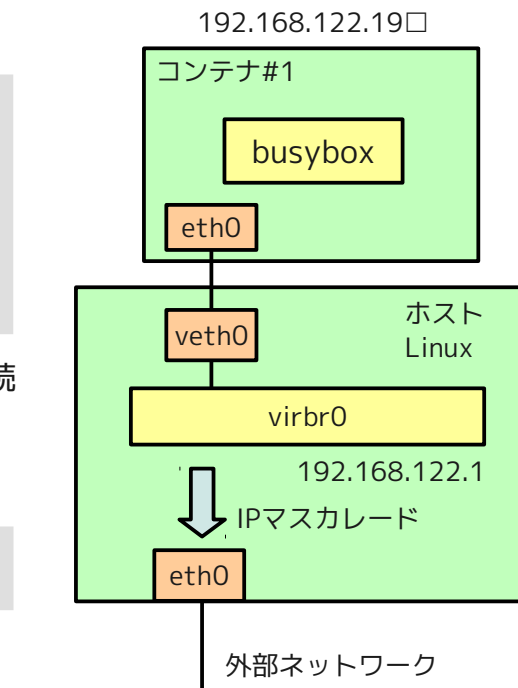
```
# brctl show
bridge name      bridge id        STP enabled  interfaces
virbr0           8000.5254001889ef  yes         veth0
                virbr0-nic
                vnet0
                vnet2
                virbr1-nic
                vnet1
virbr1           8000.525400cd2e90  yes
```

- 「virbr0」に接続された「veth0」がコンテナ内部の仮想NIC「eth0」に接続されています。

- コンテナ「lxcguest□」を停止します。

```
# virsh -c lxc:/// destroy lxcguest□
ドメイン lxcguest□ は強制停止されました
```

- 以上で「ホストLinuxからの確認」は完了です。



メモとしてお使いください

[illegible]



メモとしてお使いください

[illegible]

メモとしてお使いください

[illegible]

# Top SE

---

EDUCATION PROGRAM FOR TOP SOFTWARE ENGINEERS

