KVM / libvirt で構築した

at+link クラウド（IaaS）の実装ウラ話

[エーティーリンク]

at+link

Agenda

- 自社紹介
- at+link クラウドとは
- libvirt を使った仮想ネットワークの実装
- libvirt を使ったファイアウォールの実装
- LC (Load Control) のアルゴリズム
- sLVM(shared Logical Manager)の概要
- 最後に

自社紹介

「(株)リンクは、何をやっている会社ですか？」

「メインは、専用レンタルサーバサービス(16年)です」
http://www.at-link.ad.jp/

→ 富山県のサーバ・アプライアンス機器メーカである
エーティーワークス社との共同事業

→ Slackware 2.0 で始めた。今は RHEL と CentOS が多い。

→ データセンタ 4箇所/スタッフ 約120人/サーバ 1万台の監視・運用体制/規模

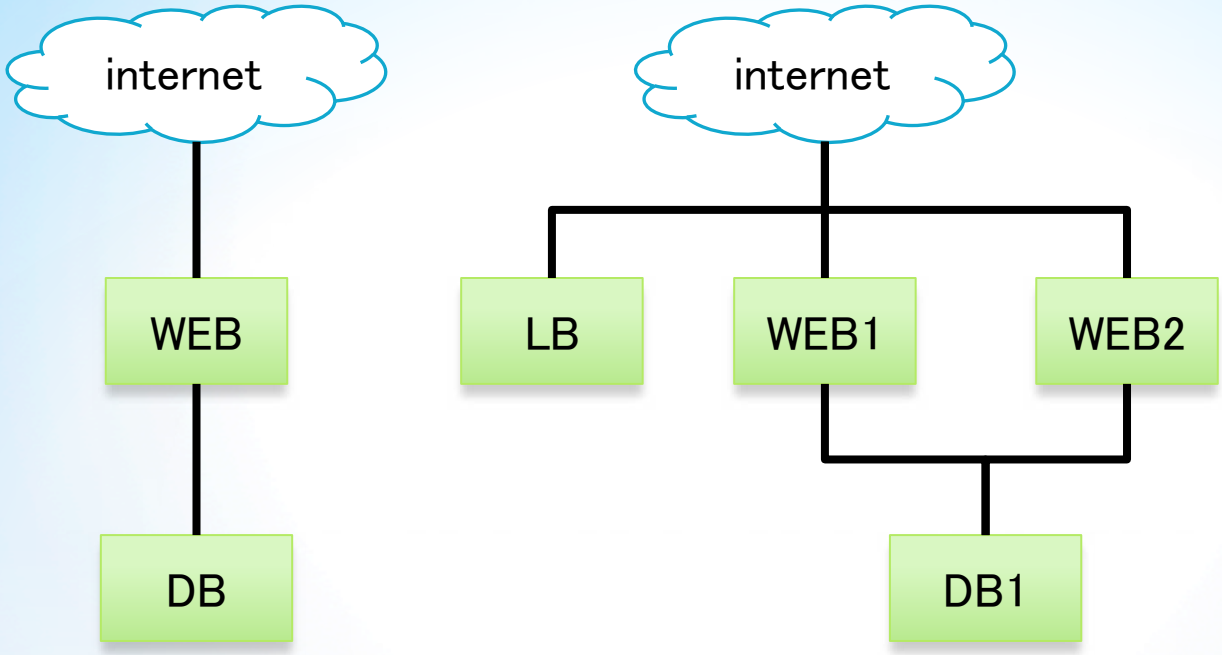
→ その他、コロケーションやクラウドもやっています。

at+link クラウドとは

・サービス的な話　http://www.at-link.ad.jp/cloud/

・技術的な話

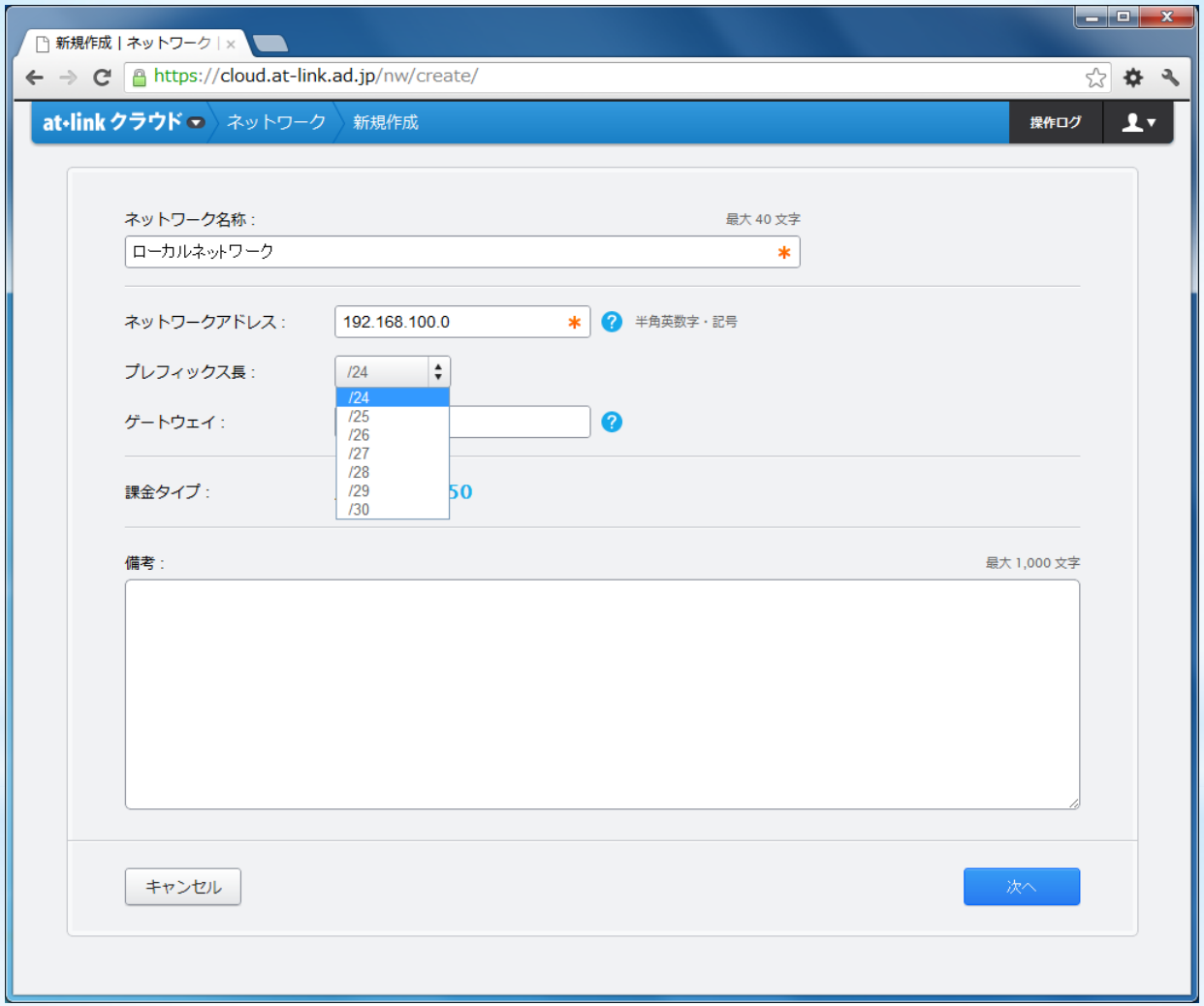
libvirt を使った
仮想ネットワークの実装



コントロールパネルの操作で、このような構成を作ることができる

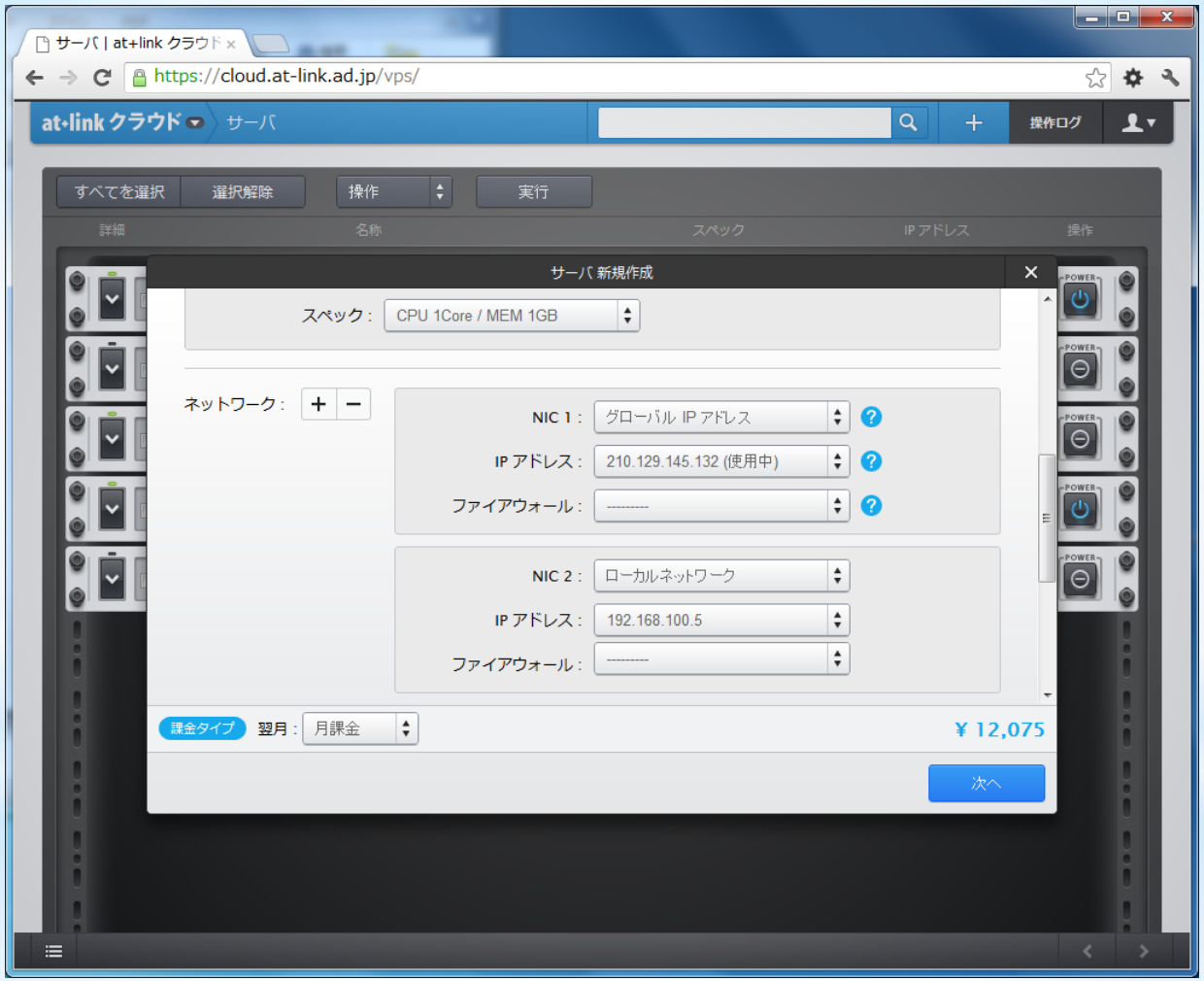
at-link

ネットワーク作成画面



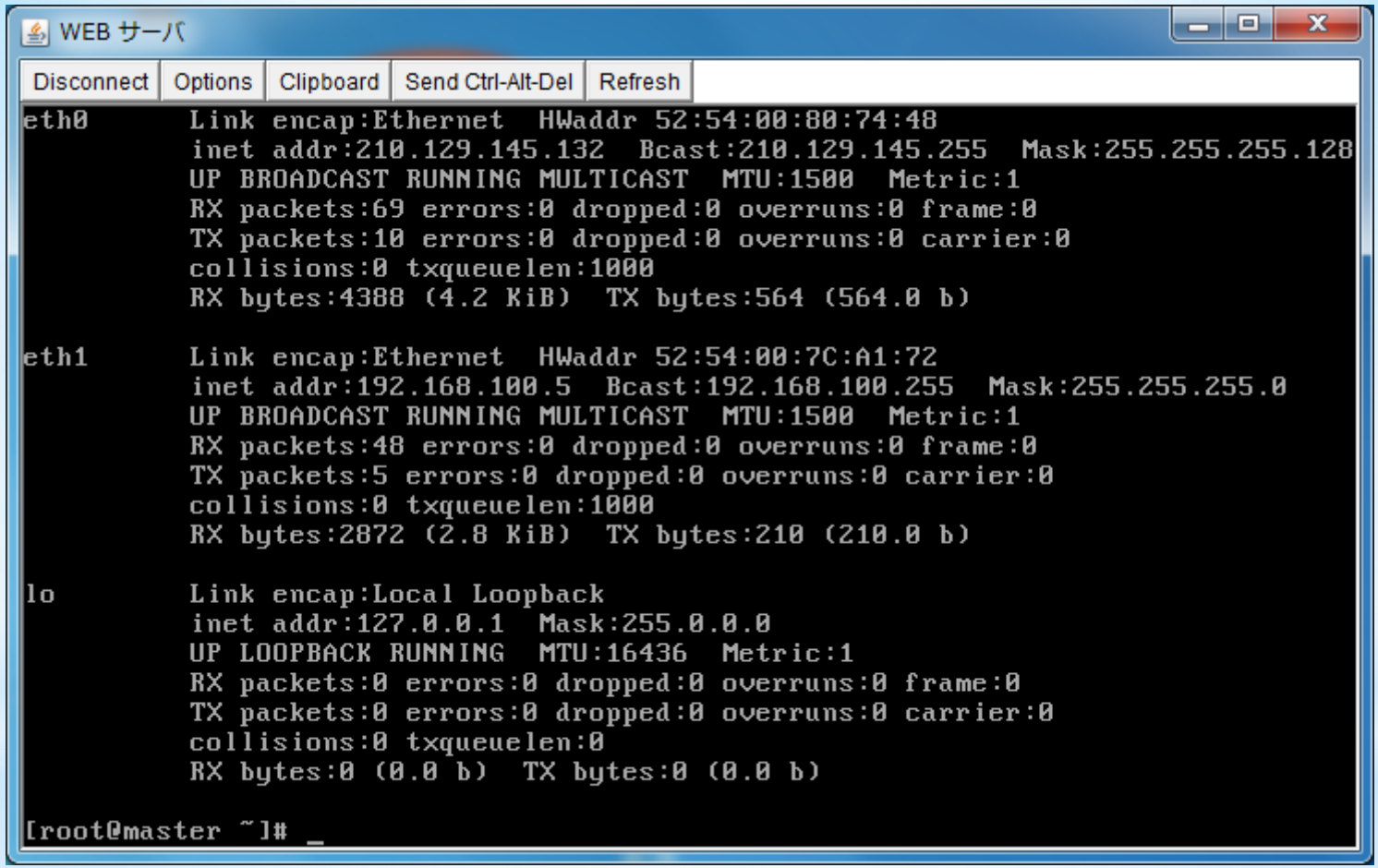
at+link

VMのインターフェースにネットワークを紐付ける



at-link

VM の インターフェースの状態



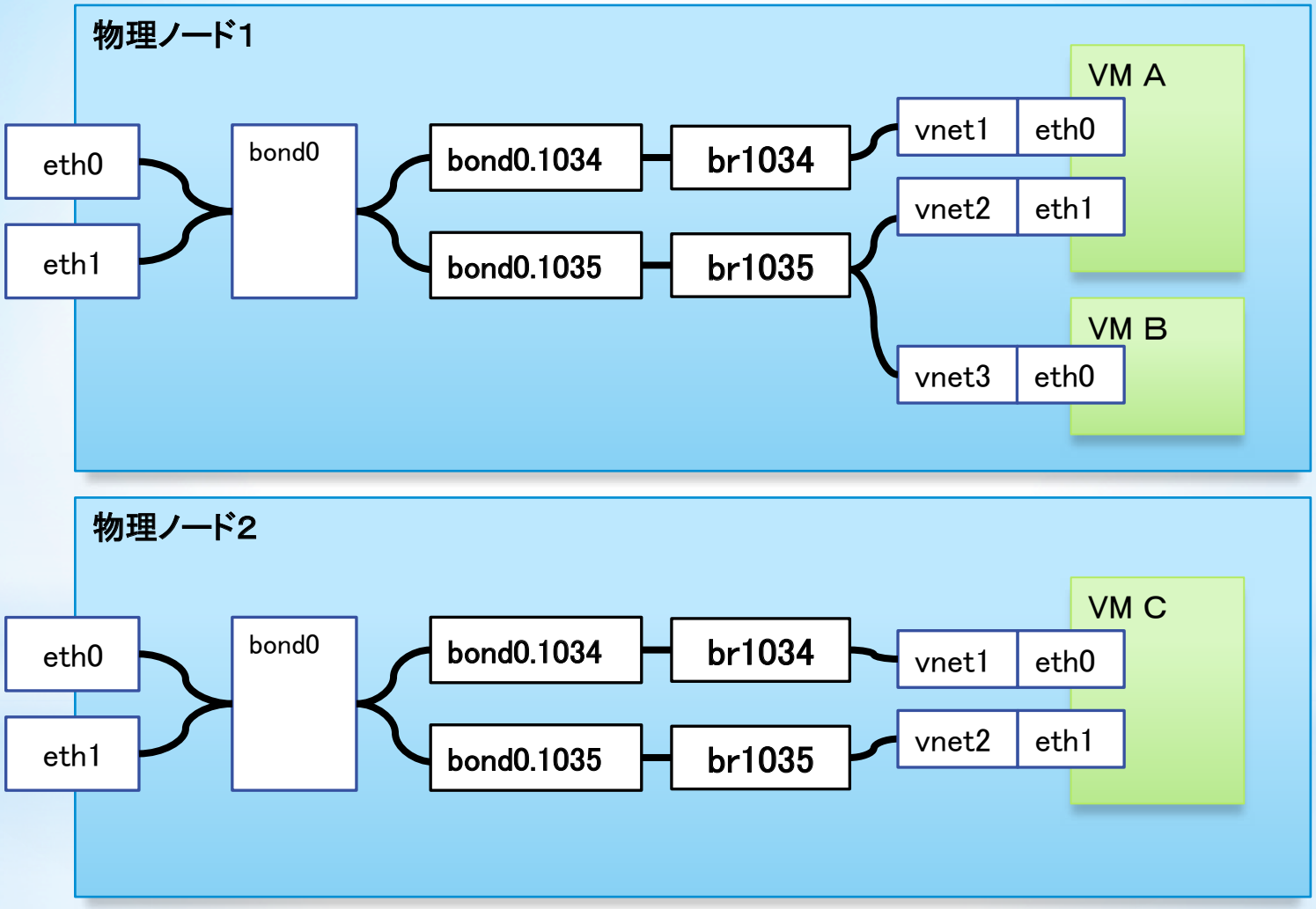
at+link

at+link クラウドの仮想ネットワーク

ブリッジ + VLAN

at+link

at+link クラウドの仮想ネットワーク（イメージ図）



at+link

仮想ネットワーク作成ソースコード

```
1    def add(self, node_id, vlan_num, vlan_name, pconn = None):
2            ifacename=self._get_ifacename(node_id)
3            conn = self._virt_connect(node_id)              物理ノードの libvirt に接続
4            br_name = 'br%04d' % vlan_num
5            xml = self._gen_xml(ifacename, vlan_num, br_name)   XMLを生成
6            vlan_ptf = conn.interfaceDefineXML(xml,0)       ブリッジと VLAN を定義（作成）
7            vlan_ptf.create(0)                              ブリッジと VLAN を起動
8            vlan_active = vlan_ptf.isActive()
9            if vlan_active:
10                   return ( 0, 'Create vlan and boot success')
11           else:
12                   return ( 2, 'Create vlan occur error' )
```

libvirt の制御は物理ノード単位。
at+link クラウドでは、複数ノードを扱うので、処理が必要な各ノードに対してこの操作を行う。

at-link

interfaceDefineXML（libvirt.py）

```
class virConnect:
    def __init__(self, _obj=None):
        if _obj != None:self._o = _obj;return
        self._o = None
                  :
                  :
    def interfaceDefineXML(self, xml, flags):
        """Define an interface (or modify existing interface configuration).

        Normally this change in the interface configuration is immediately
        permanent/persistent, but if virInterfaceChangeBegin() has been
        previously called (i.e. if an interface config transaction is
        open), the new interface definition will only become permanent if
        virInterfaceChangeCommit() is called prior to the next reboot of
        the system running libvirtd. Prior to that time, it can be
        explicitly removed using virInterfaceChangeRollback(), or will be
        automatically removed during the next reboot of the system running
        libvirtd. """
        ret = libvirtmod.virInterfaceDefineXML(self._o, xml, flags)
        if ret is None:raise libvirtError('virInterfaceDefineXML() failed', conn=self)
        __tmp = virInterface(self, _obj=ret)
        return __tmp
```

★ libvirt API

at-link

create （libvirt.py）

```
class virInterface:
    def __init__(self, conn, _obj=None):
        self._conn = conn
        if _obj != None:self._o = _obj;return
        self._o = None
                  :
                  :
    def create(self, flags):
        """Activate an interface (i.e. call "ifup").

        If there was an open network config transaction at the time this
        interface was defined (that is, if virInterfaceChangeBegin() had
        been called), the interface will be brought back down (and then
        undefined) if virInterfaceChangeRollback() is called.
        p * """
        ret = libvirtmod.virInterfaceCreate(self._o, flags)                    ★ libvirt API
        if ret == -1: raise libvirtError ('virInterfaceCreate() failed', net=self)
        return ret
```

at-link

XML の生成

```
1  def _gen_xml(self, ifacename, vlan_num, br_name):
2              interface_value = ifacename
3              if vlan_num>0:
4                          tag_value = str(vlan_num)
5                          interface_name = interface_value + '.' + tag_value
6                          xml = """¥
7                          <interface type='bridge' name='%s'>
8                                      <start mode='onboot'/>
9                                      <bridge delay='0'>
10                                             <interface type='vlan' name='%s'>
11                                                     <vlan tag='%s'><interface name='%s'/></vlan>
12                                             </interface>
13                                     </bridge>
14                         </interface>""" %¥
15                          (br_name, interface_name, tag_value, interface_value)
16             else:
17                         xml = """¥
18                         <interface type='bridge' name='%s'>
19                                     <start mode='onboot'/>
20                                     <bridge delay='0'>
21                                             <interface name='%s'/>
22                                     </bridge>
23                         </interface>""" %¥
24                          (br_name, interface_value)
25             return xml
```

at-link

実際のXML

```
1   virsh # iface-dumpxml br1034
2   <interface type='bridge' name='br1034'>
3     <protocol family='ipv6'>
4       <ip address='fe80::21b:21ff:febe:9207' prefix='64'/>
5     </protocol>
6     <bridge>
7       <interface type='vlan' name='bond0.1034'>
8         <vlan tag='1034'>
9           <interface name='bond0'/>
10        </vlan>
11      </interface>
12      <interface type='ethernet' name='vnet4'>
13        <mac address='fe:54:00:90:e5:5e'/>
14      </interface>
15    </bridge>
16  </interface>
```

XML のフォーマットは、libvirt のドキュメント(http://libvirt.org/format.html)を参照。
使用例も含め、記載されている。

at-link

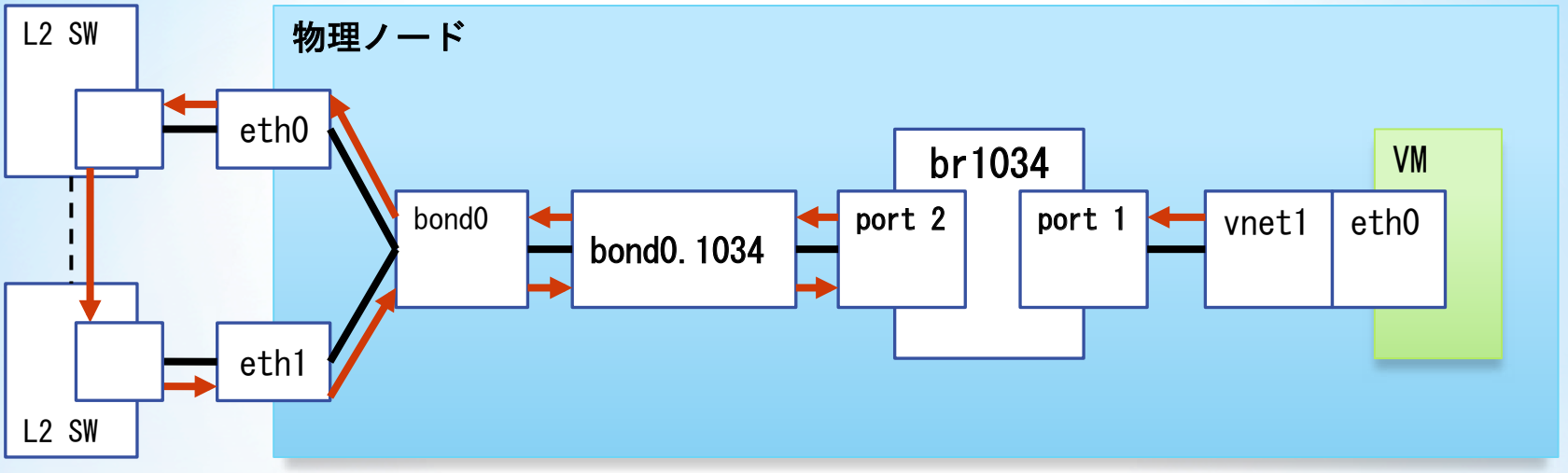
ハマったこと

・ブリッジ + VLAN + ボンディングの場合、VM の通信ができない。

・ボンディングの片方の NIC を down させると事象は発生しない。

at-link

結果には　RHEL 6 のバグ
(https://bugzilla.redhat.com/show_bug.cgi?id=623199)



・ブリッジは、内部的に転送用のデータベースを持つ。

・VM が送信したイーサネットフレームをbr1034 は port1 にVMのeth0 のMAC があると学習。

・正常であれば、eth1 でフレームを破棄するが、バグによって br1034 まで到達。

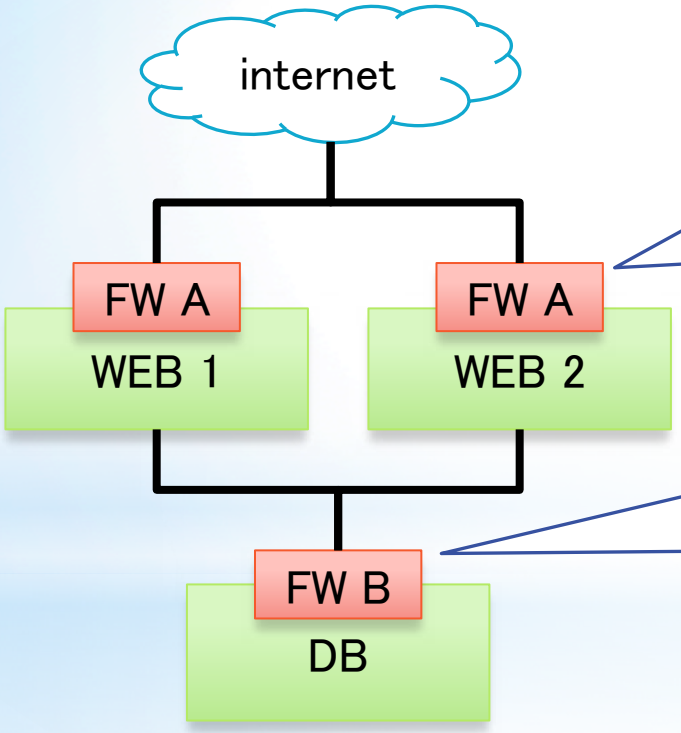
・br1034 は、VM の eth0 の MAC アドレスは、port2 に存在すると間違えた学習を行う。

libvirt を使った
ファイアウォールの実装

at+link

at+link クラウドのファイアウォール機能

インバウンドのパケットフィルタリング



TCP 80 ,443 を全てのIP アドレスで許可

TCP 22 を特定の IP アドレスのみ許可

ICMP特定の IP アドレスのみ許可

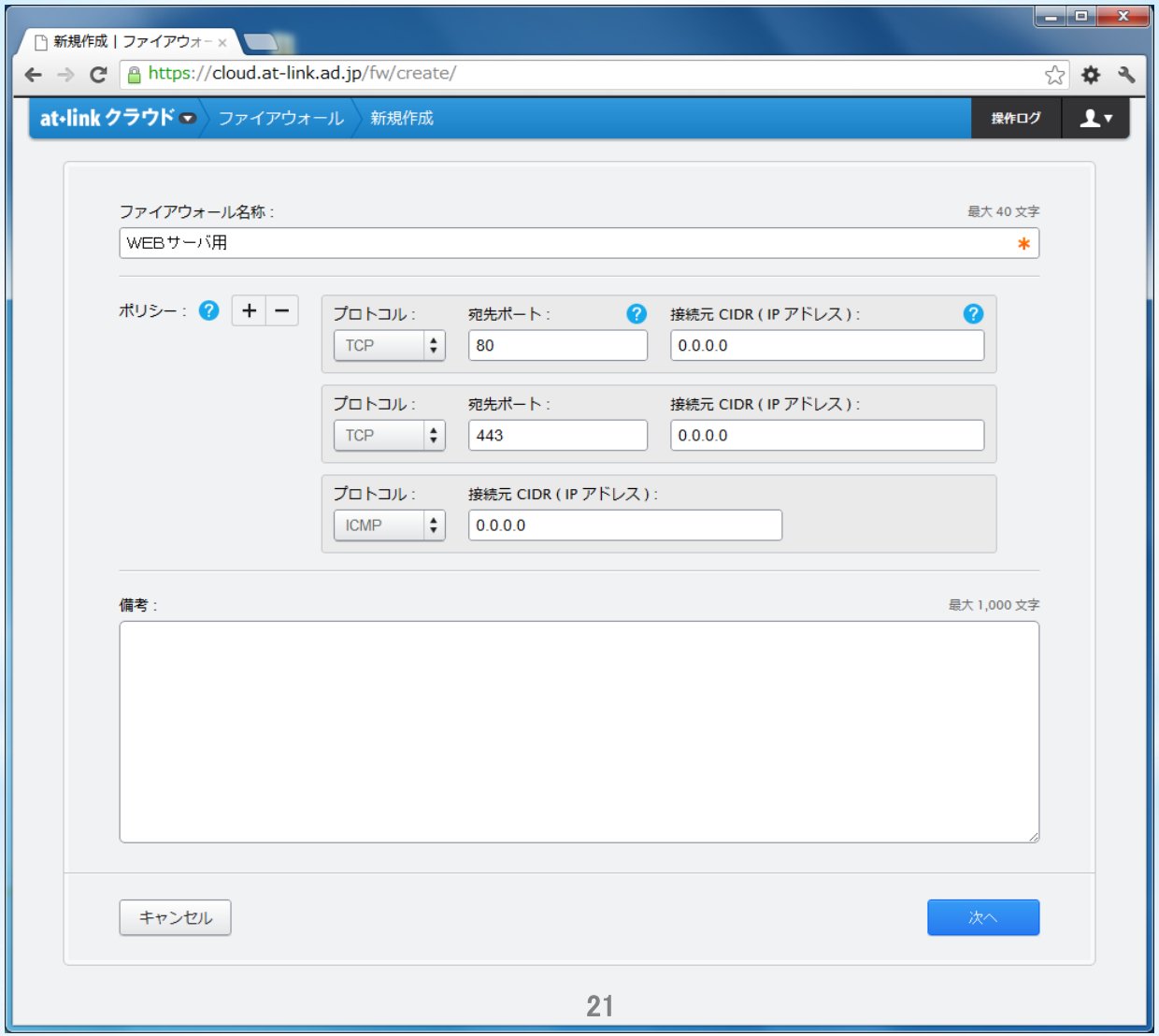
TCP 3306 をWEB1,WEB2 のローカル IP のみ許可

TCP 22　WEB1 ,WEB2 のローカル IP のみ許可

ICMP　WEB1 ,WEB2 のみ許可

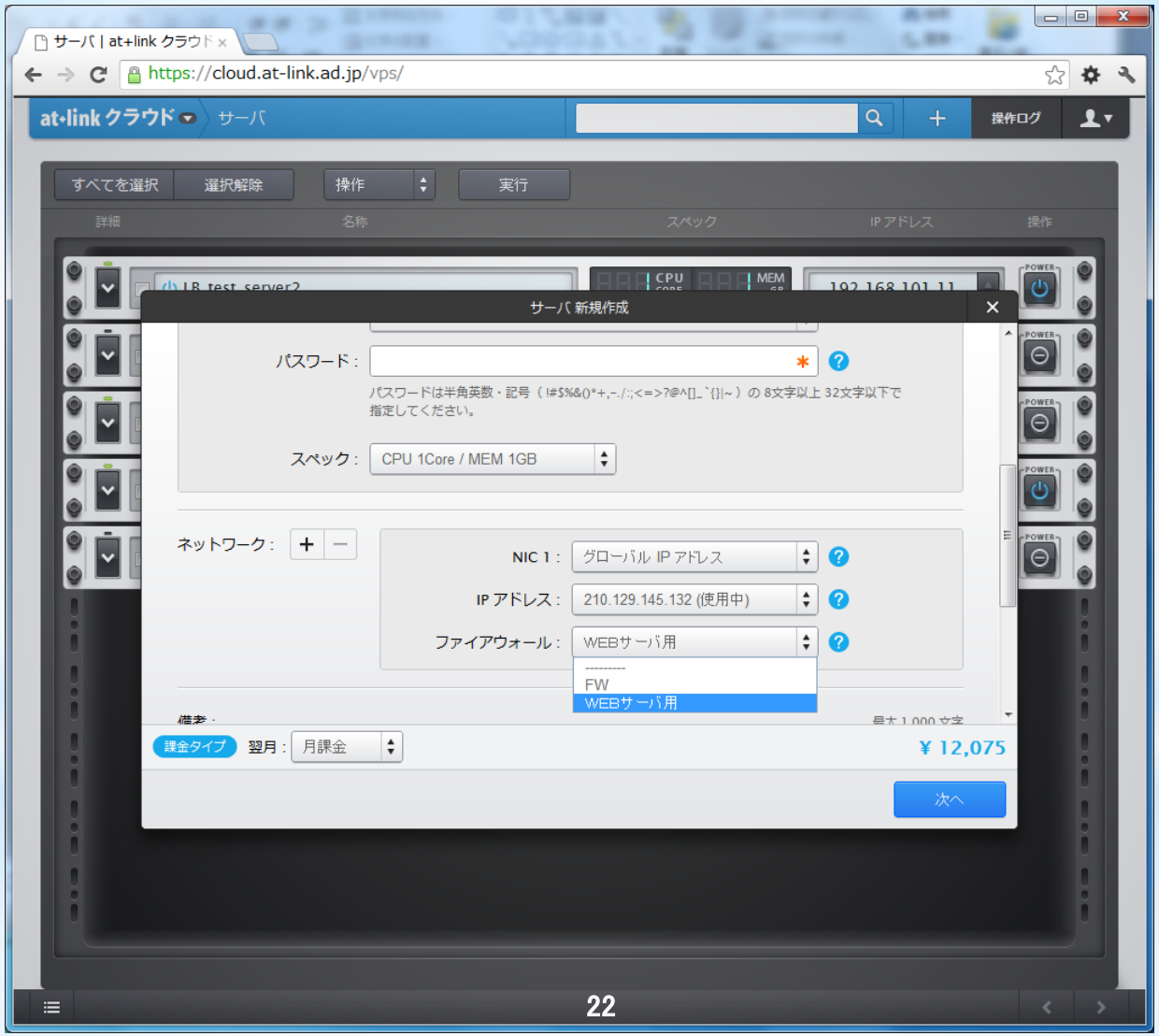
at-link

ファイアウォールを作る画面



at+link

ファイアウォールを適用する画面



at+link

at+link クラウドのファイアウォール

libvirt の nwfilter 機能

at-link

nwfilter 作成ソースコード

```
1    conn = db._virt_connect( node_uuid )          物理ノードの libvirt に接続
2    xmls=policy.get_interface_policy_xml(obj_uuid)    XMLを生成
3    for xml in xmls:
4            conn.nwfilterDefineXML(xml)           nwfilter の定義（作成）
```

at-link

nwfilterDefineXML関数（libvirt.py）

```
class virConnect:
    def __init__(self, _obj=None):
        if _obj != None:self._o = _obj;return
        self._o = None
                :
    def nwfilterDefineXML(self, xmlDesc):
        """Define a new network filter, based on an XML description
        similar to the one returned by virNWFilterGetXMLDesc() """
        ret = libvirtmod.virNWFilterDefineXML(self._o, xmlDesc)          ★ libvirt API
        if ret is None:raise libvirtError('virNWFilterDefineXML() failed', conn=self)
        __tmp = virNWFilter(self, _obj=ret)
        return __tmp
```

at-link

XML の生成

```
1 def get_interface_policy_xml(self,interface_uuid):
2                 :
3                 allxml=[]
4                 inrefs=[]
5                 for uuid,direction in ret:
6                                 allxml.append(self.get_policy_xml(uuid))
7                                 inrefs.append("<filterref filter='%s'/>" % uuid)
8           xml="""<filter name='%s' chain='root'>
9           <uuid>%s</uuid>
10          <rule action='accept' direction='inout'>
11          <all state='ESTABLISHED,RELATED'/>
12          </rule>
13          <rule action='drop' direction='out' priority='10'>
14          <mac match='no' srcmacaddr='$MAC' />
15          </rule>
16          <rule action='accept' direction='inout' priority='600' >
17          <arp /></rule>""" % (interface_uuid,interface_uuid)
18          if len(inrefs)==0:
19              xml=xml+"""<rule action='accept' direction='in' priority='100'><all/></rule>"""
20          else:
21              xml=xml+"¥n".join(inrefs)+"""<rule action='drop' direction='in' priority='1000'><all/></rule>"""
22          xml=xml+"</filter>"
23          allxml.append(xml)
24          return allxml
```

at-link

FW を適用したVMのXML（インターフェース部分のみ）

```
# virsh dumpxml '45c2cb40-0824-4b45-aec3-be5c2438d360'
<domain type='kvm' id='5'>
                :
   <interface type='bridge'>
     <mac address='52:54:00:7d:e6:2e'/>
     <source bridge='br1034'/>
     <target dev='vnet0'/>
     <model type='virtio'/>
     <filterref filter='4eb1f34e-83d8-49f9-98fa-d8f6086c0106'/>  ★
     <alias name='net0'/>
     <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
   </interface>
   :
</domain>
```

at-link

filter のXML

```
# virsh nwfilter-dumpxml '4eb1f34e-83d8-49f9-98fa-d8f6086c0106'
<filter name='4eb1f34e-83d8-49f9-98fa-d8f6086c0106' chain='root'>
  <uuid>4eb1f34e-83d8-49f9-98fa-d8f6086c0106</uuid>
  <rule action='accept' direction='inout' priority='500'>
    <all state='ESTABLISHED,RELATED'/>
  </rule>
  <rule action='drop' direction='out' priority='10'>
    <mac match='no' srcmacaddr='$MAC'/>
  </rule>
  <rule action='accept' direction='inout' priority='600'>
    <arp/>
  </rule>
<filterref filter='64cf28be-db24-4be8-95b8-e6eb94305b20' />  ★  指定したルール
  <rule action='drop' direction='in' priority='1000'>
    <all/>
  </rule>
  <rule action='accept' direction='out' priority='100'>
    <all/>
  </rule>
</filter>
```

at-link

指定したルールのXML

```
# virsh nwfilter-dumpxml '64cf28be-db24-4be8-95b8-e6eb94305b20'
<filter name='64cf28be-db24-4be8-95b8-e6eb94305b20' chain='root'>
  <uuid>64cf28be-db24-4be8-95b8-e6eb94305b20</uuid>
  <rule action='accept' direction='in' priority='500'>
    <icmp srcipaddr='0.0.0.0' srcipmask='0'/>
  </rule>
  <rule action='accept' direction='in' priority='500'>
    <tcp srcipaddr='0.0.0.0' srcipmask='0' dstportstart='443'/>
  </rule>
  <rule action='accept' direction='in' priority='500'>
    <udp srcipaddr='0.0.0.0' srcipmask='0' dstportstart='18000'/>
  </rule>
</filter>
```

at-link

libvirt の nwfiler

・実体は、物理ノードの iptables 。

・指定したルールは、vnet＊に対して適用される。

LC（ Load Control ）のアルゴリズム

at-link

Load Control :

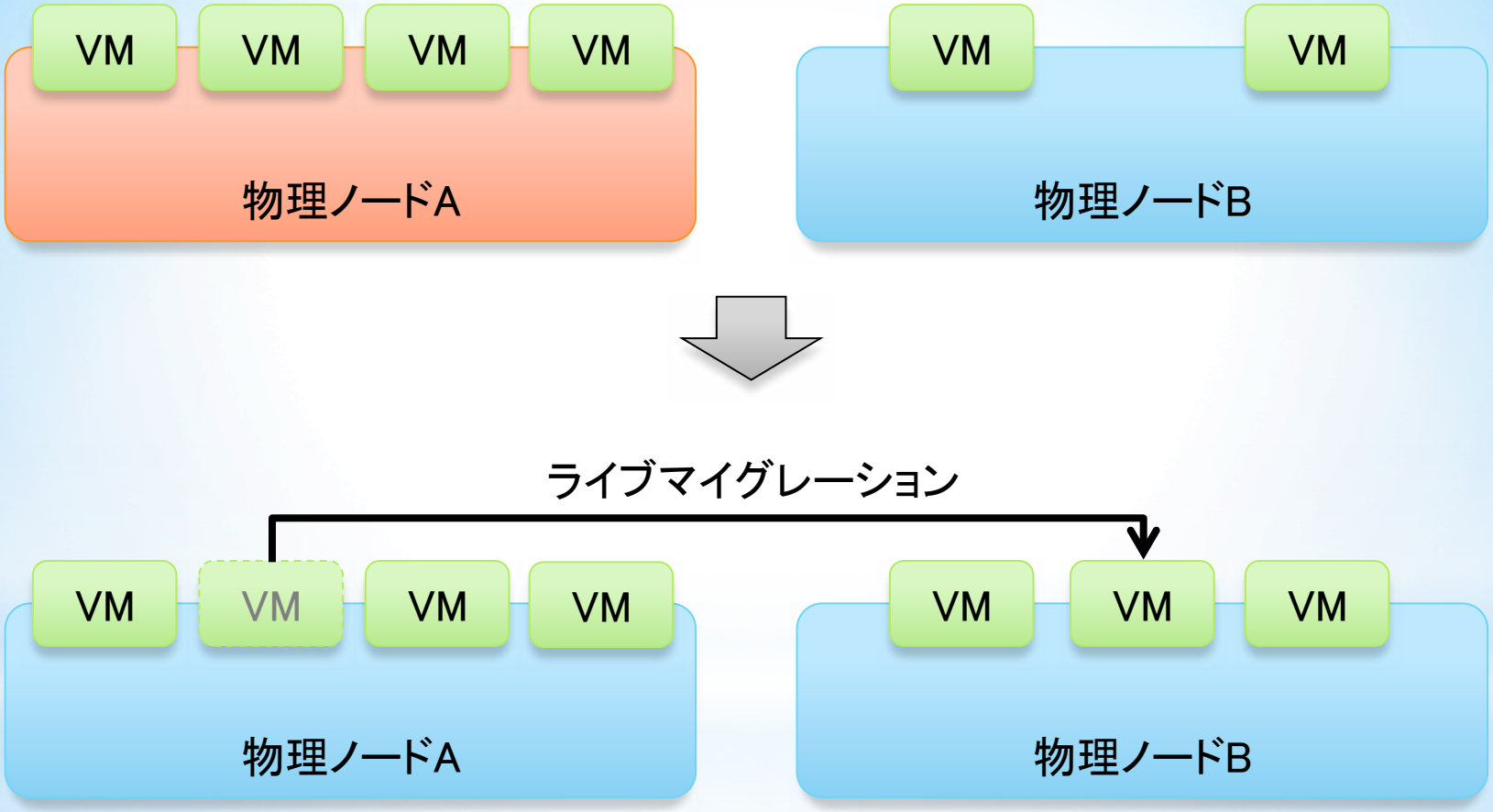
高負荷状態の物理ノードで稼働している VM を、

自動的に他の物理ノードに移動させる機能。

特定の物理ノードで処理能力が不足することを回避する。

at-link

LC のイメージ



処理の流れ

・高負荷状態にある物理ノードを検知

・移動先候補となる物理ノードの選定

・移動させる VM 候補の選定

・移動先物理ノード／移動させるVMの決定

・ライブマイグレーション実行

高負荷状態にある物理ノードの検知、移動先候補となる物理ノードの選定

・負荷状況の1分間平均を蓄積

・一定時間以上に渡り閾値を上回る場合、高負荷状態とみなす。

・一定時間以上に渡り閾値を下回る場合、移動先候補とみなす。

at-link

移動させる VM 候補の選定

VM の リソース使用量などから**適切な** VM を選定

リソース使用量が多い VM
移動先の物理ノードが高負荷となり、連鎖的にマイグレーションが発生する。

リソース使用量が少ない VM
負荷が下がらないため、複数 VMのマイグレーションが発生する。

at-link

移動させるVM／移動先物理ノードを決定

移動候補のVMを、移動先候補の VM に移動させた場合の

状態をシミュレートし、最終的に決定。

ライブマイグレーション実行

・ライブマイグレーション実行

・ライブマイグレーション実行後は、

一連のフローを開始するまで一定時間の間隔をあける

sLVM（shared Logical Manager）の概要

at-link

sLVM （ shared Logical Manager ）:

複数のストレージ領域を、複数の物理ノードで共有する機能。

独自に開発・実装。

clvm / corosync を利用していたが、

--

・突然物理ノードからストレージが見えなくなる（ virsh pool-list が無応答 ）。

・意図しない フェンス（遮断） が働く。

・問題発生時の復旧が簡単ではない。

--

により、利用し続けることを断念。。

at-link

ポイントとなる機能

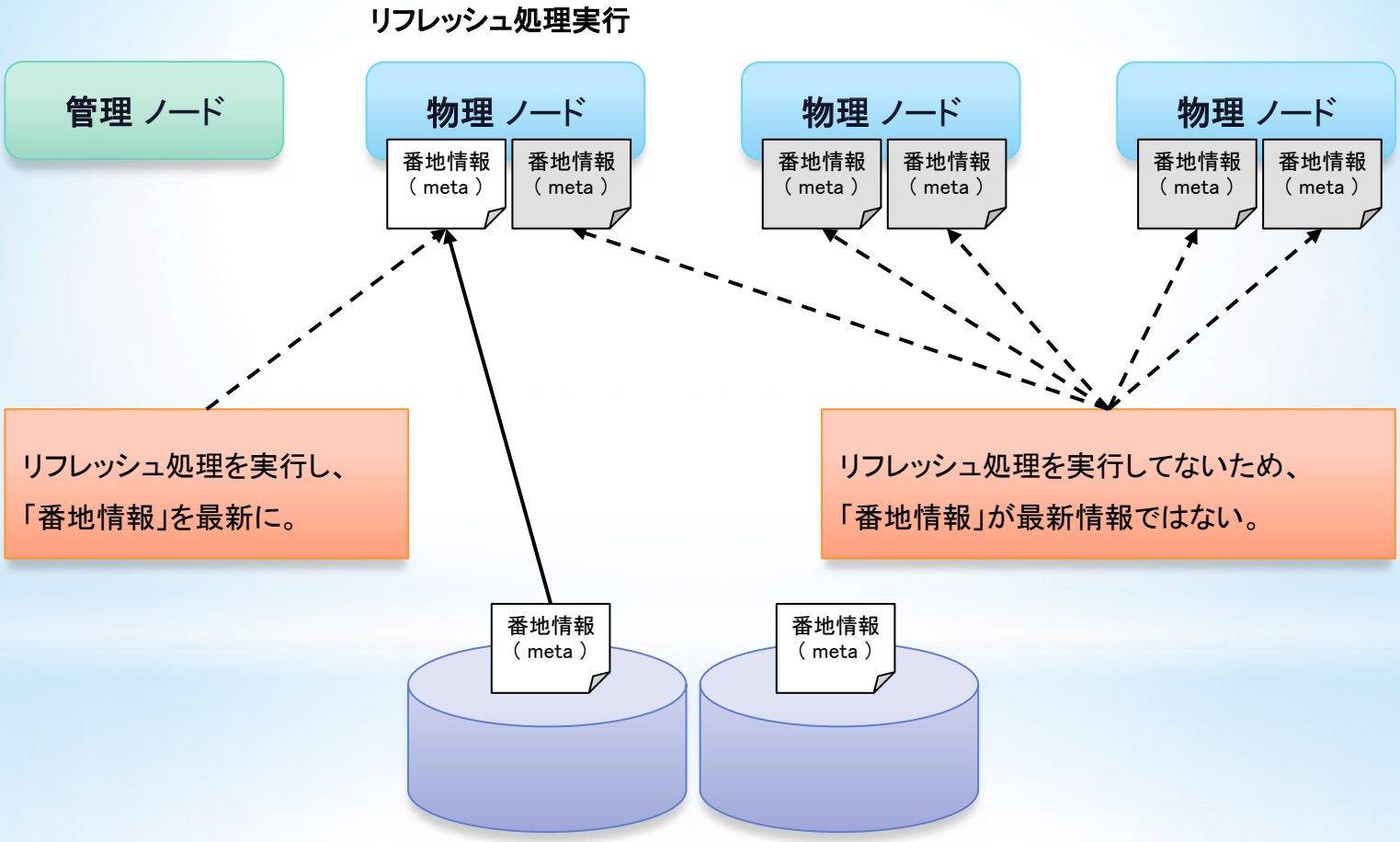
複数の物理ノードで、ストレージ領域の「番地情報」を共有する機能の実現。

clvm / corosync は、

常に最新の「番地情報」 を共有・維持する実装。

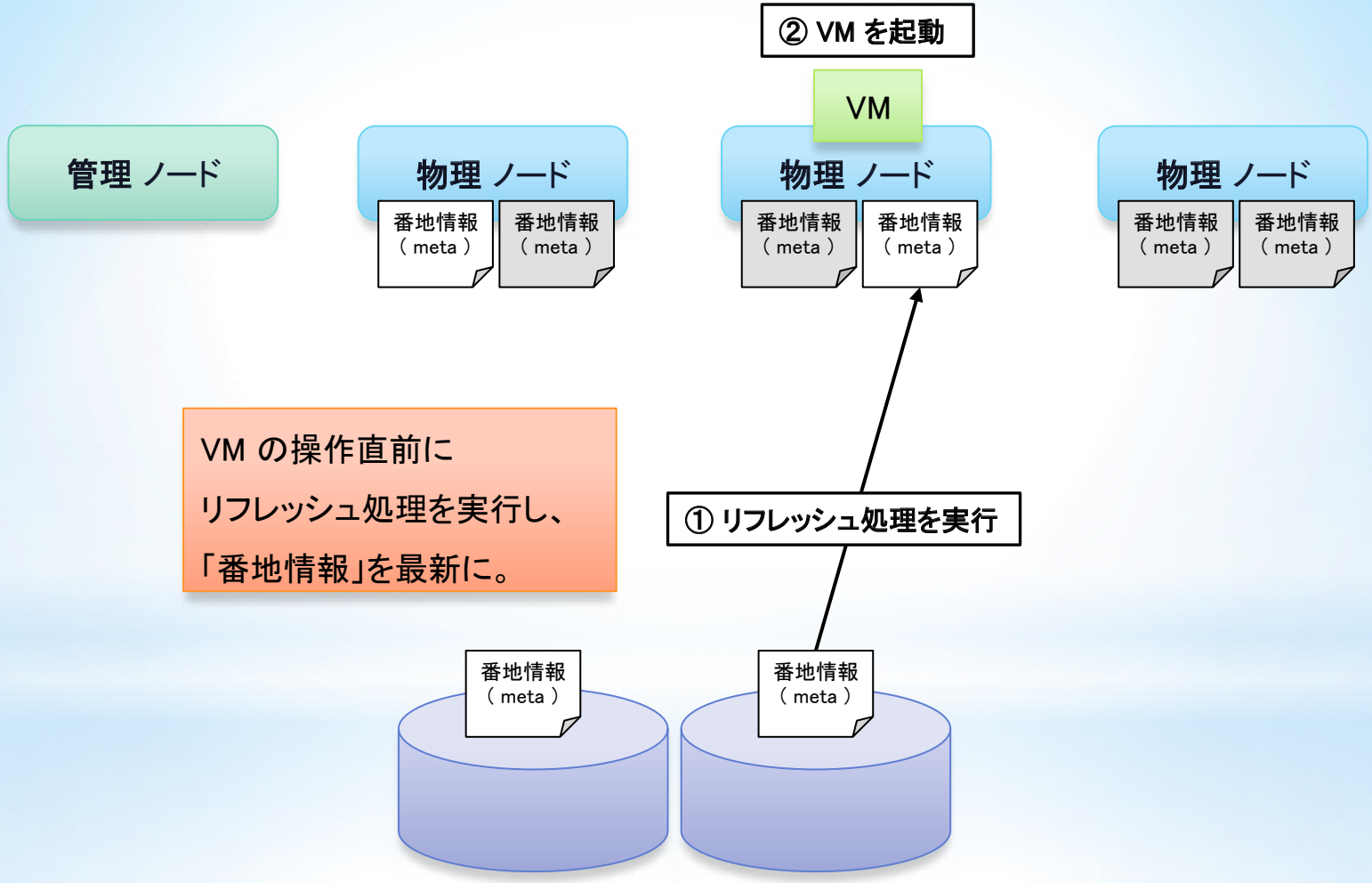
at-link

sLVM の「番地情報」取得処理（リフレッシュ処理）



at-link

VM起動時の処理



at-link

sLVM のポイント

常に最新の「番地情報」保つのではなく、

必要なタイミングで最新の「番地情報」を取り込む。

シンプルな仕組みのため、

物理ノードの増減など、メンテナンスも容易に。

at+link クラウドでは、無償のトライアル環境を提供しています。

期間　　：　1ヶ月
サーバ数　：　3台まで
スペック　　：　CPU 2core / MEM　2GB まで

お申し込み　：

http://www.at-link.ad.jp/cloud/　の「トライアルを申し込む」