

Pacemaker + KVMで 仮想化クラスタリング ～仮想化連携機能のご紹介～

2011年11月20日 OSC2011 Tokyo/Fall

Linux-HA Japan

中平 和友



本日のお話

- ① Linux-HA Japanについて
- ② 仮想化環境のクラスタ化について
- ③ 仮想化連携機能について
- ④ インストール・設定方法
 - 仮想化ホスト・ゲスト共通の作業
 - 仮想化ホストの作業
 - 仮想化ゲストの作業
- ⑤ 仮想化連携機能を使ってみよう

①

Linux-HA Japanについて



Linux-HA Japan URL

<http://linux-ha.sourceforge.jp/>

(一般向け)

<http://sourceforge.jp/projects/linux-ha/> (開発者向け)



Pacemaker情報の公開用として
随時情報を更新中です。

Pacemakerリポジトリパッケージ最新版
(1.0.11-1.2.2)はここで入手可能です。

Linux-HA Japanメーリングリスト

日本におけるHAクラスタについての活発な意見交換の場として「Linux-HA Japan日本語メーリングリスト」も開設しています。

Linux-HA-Japan MLでは、Pacemaker、Heartbeat3、Corosync DRBDなど、HAクラスタに関連する話題は歓迎！

• ML登録用URL

<http://linux-ha.sourceforge.jp/>
の「メーリングリスト」をクリック

• MLアドレス

linux-ha-japan@lists.sourceforge.jp

※スパム防止のために、登録者以外の投稿は許可制です



②

仮想化環境のクラスタ化について

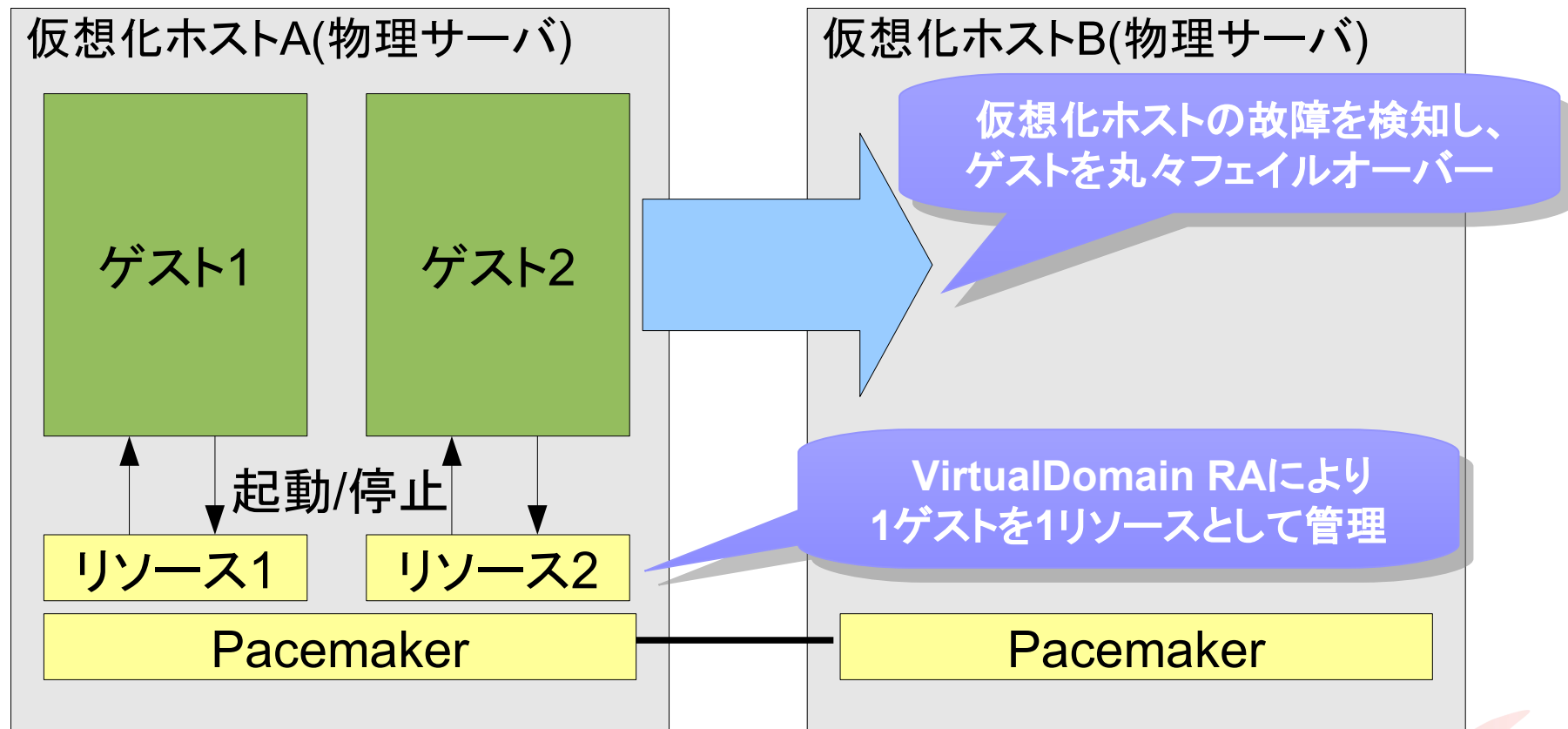


クラスタの構成パターン

- その1: 仮想化ホストをクラスタ化
 - 仮想化ホストOS上にPacemakerをインストール
 - 仮想化ホストの故障を検知し、**ゲスト単位でフェイルオーバー**
- その2: 仮想化ゲストをクラスタ化
 - 仮想化ゲストOS上にPacemakerをインストール
 - ゲスト内でリソースを監視し、**リソース単位でフェイルオーバー**
- その3: 両者の組み合わせ

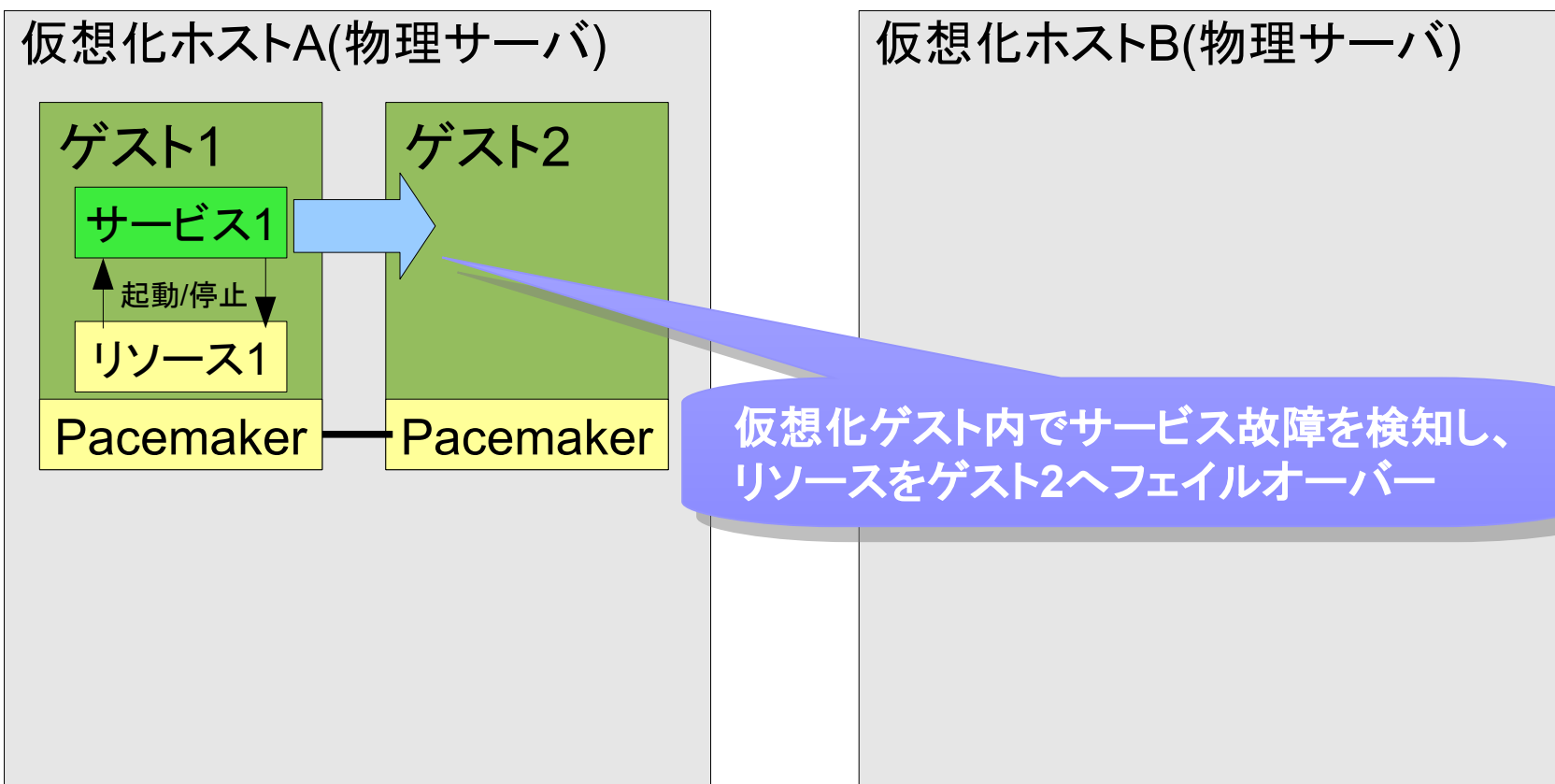
その1: 仮想化ホストのクラスタ化

- ホストOS上のPacemakerが、ゲストをリソースとして管理



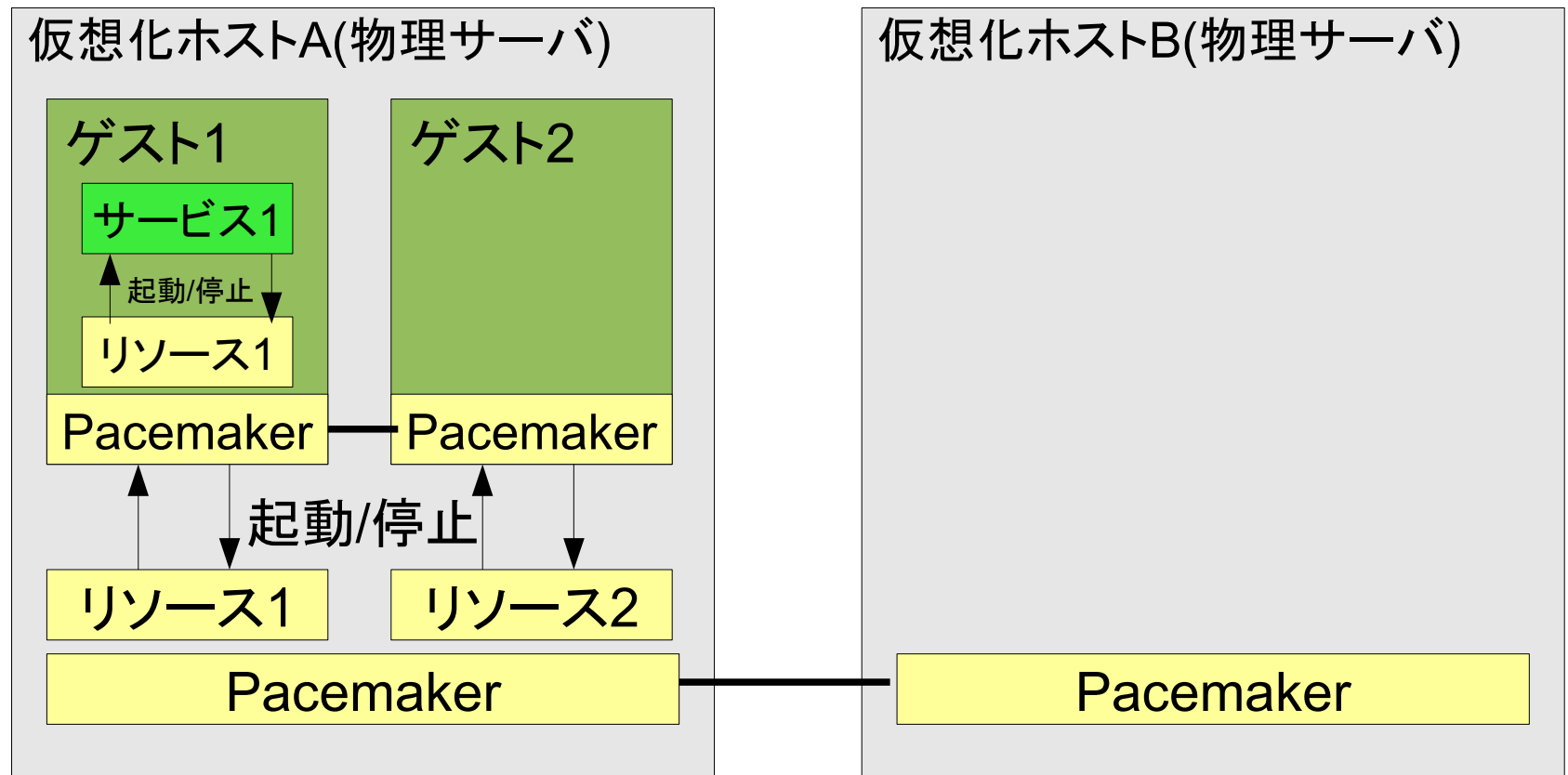
その2: 仮想化ゲストのクラスタ化

- ゲストOS上のPacemakerが、サービスを管理



その3: 両者の組み合わせ

- ホストのH/W故障、ゲストのサービス故障共に対応可能



③

仮想化連携機能について



開発の背景 (1/2)

仮想化ホストの改善

- 仮想化ホストの管理作業を楽にしたい！
 - 例：ゲストを新規追加するとき、Pacemaker側では以下のような設定を追加する必要があります
 - crmコマンドがあるとはいえ、設定作業は大変！

```
primitive prmVMCTL_guest1 ocf:extra:VirtualDomain \  
  params config="/etc/libvirt/qemu/guest1.xml" hypervisor="qemu:///system" migration_transport="ssh" \  
  meta allow-migrate="false" target-role="Stopped" \  
  op start interval="0" timeout="120s" on-fail="restart" \  
  op monitor interval="10s" timeout="30s" on-fail="restart" \  
  op stop interval="0" timeout="90s" on-fail="fence" \  
  op migrate_to interval="0" timeout="300s" on-fail="fence" \  
  op migrate_from interval="0" timeout="240s" on-fail="restart" \  
location locVMCTL_guest1_default_ping_set prmVMCTL_guest1 \  
  rule $id="locVMCTL_guest1_default_ping_set-rule" -inf: not_defined default_ping_set or \  
  default_ping_set lt 100 \  
location locVMCTL_guest1_diskcheck_status prmVMCTL_guest1 \  
  rule $id="locVMCTL_guest1_diskcheck_status-rule" -inf: not_defined diskcheck_status or \  
  diskcheck_status eq ERROR \  
location locVMCTL_guest1_host1_ACT prmVMCTL_guest1 200: host1 \  
colocation colVMCTL_guest1_clnDiskd1 inf: prmVMCTL_guest1 clnDiskd1 \  
colocation colVMCTL_guest1_clnPingd inf: prmVMCTL_guest1 clnPingd \  
order odrVMCTL_guest1_clnDiskd1 0: clnDiskd1 prmVMCTL_guest1 symmetrical=false \  
order odrVMCTL_guest1_clnPingd 0: clnPingd prmVMCTL_guest1 symmetrical=false
```

開発の背景 (2/2)

仮想化ゲストの改善

- 仮想化環境のゲスト内でも**STONITH**を使いたい！
 - **STONITH**: スプリットブレインやリソース停止故障発生時に相手ノードを強制的に停止する機能
 - 既存のSTONITHプラグインは、特定ホスト上のゲストしか落とせない！（ゲストが別ホストへ移動するとNG）

これらの課題を解決するため、2つのツールを開発しました

ツールその1 : vm-ctl

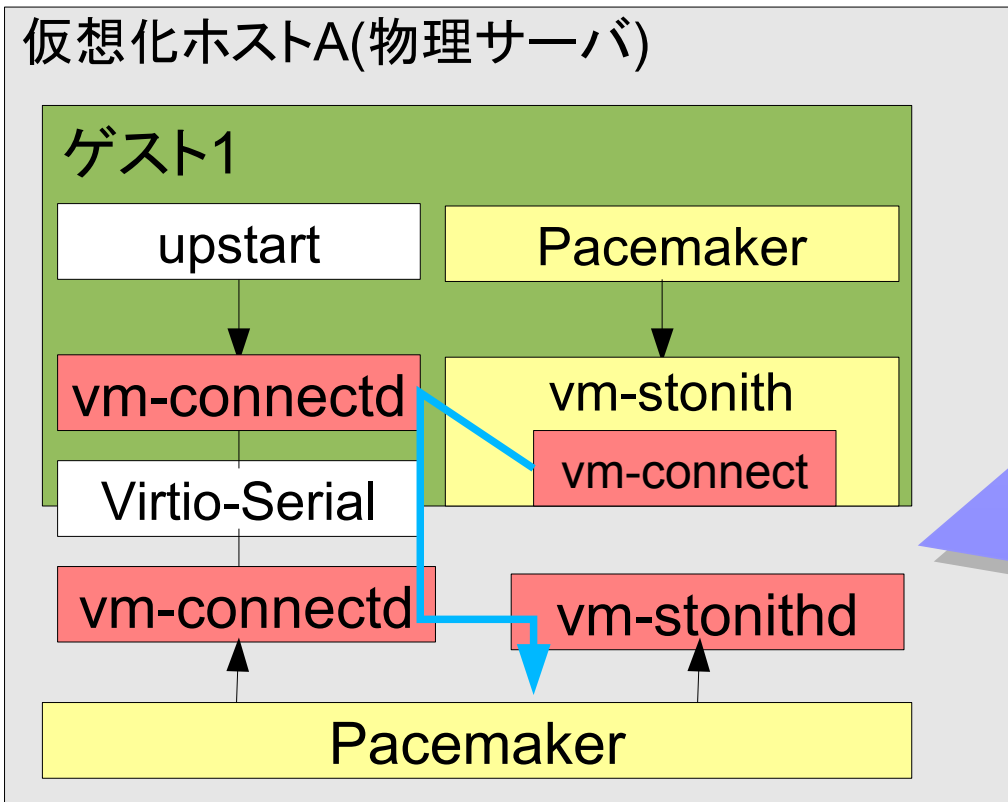
■ 仮想マシンリソース制御機能

- 仮想化ホストで使う crmコマンドのラッパーシェル
- Pacemakerのゲスト管理用設定をコマンド一発で実行します
 - ゲストをクラスタへ追加
 - ゲストをクラスタから削除
 - ゲストの起動
 - ゲストの停止
 - ゲストのホスト間移動(ライブマイグレーション)
 - ゲストの起動ノードの指定

ツールその2: pm_kvm_tools

■ 仮想環境連携機能

- 仮想化ゲスト<->ホスト間の通信機能を提供
- ゲストからホストと連携したSTONITHが可能になります



- vm-connectd
ゲスト-ホスト通信デーモン
- vm-connect
メッセージ送信コマンド
- vm-stonithd
STONITH実行デーモン

④

インストール・設定方法



前提条件 (1/2)

- 仮想化連携機能の動作環境
 - OS: RHEL6.0以降と、その互換OS(Scientific Linux, etc..)
 - HA: Pacemaker-1.0.10以降
(後述のリポジトリパッケージ 1.0.11-1.2.2.el6.x86_64がお勧め)
- 仮想化ホストで virshが利用可能であること
 - ゲストの起動/停止が virsh経由で可能な状態
- 仮想化ホストとゲストのOSはインストール済みとします
 - ゲストの作成手順は今回は省略します
 - virt-managerの使い方などはまた別の勉強会で...

前提条件 (2/2)

- ゲストのDiskイメージは、共有ストレージに配置します
また Pacemakerでマウント管理はしません
 - 例: /var/lib/libvirt/images をホスト起動時にNFSマウント
複数ホスト間で同時マウントできればNFS以外でもOK
→ ライブマイグレーションを可能にするため
- ゲストのドメイン定義がホスト間で同期していること
host1# scp /etc/libvirt/qemu/hoge.xml host2:/etc/libvirt/qemu/
host2# virsh define /etc/libvirt/qemu/hoge.xml

インストール・設定の流れ

- 仮想化ホスト・ゲスト共通で実施する作業
 - Pacemakerリポジトリパッケージのインストール
 - Pacemaker初期設定
- 仮想化ホストで実施する作業
 - Pacemaker監視設定・STONITH設定
 - vm-ctl設定
 - pm_kvm_tools設定
- 仮想化ゲストで実施する作業
 - Pacemaker監視設定・STONITH
 - pm_kvm_tools設定

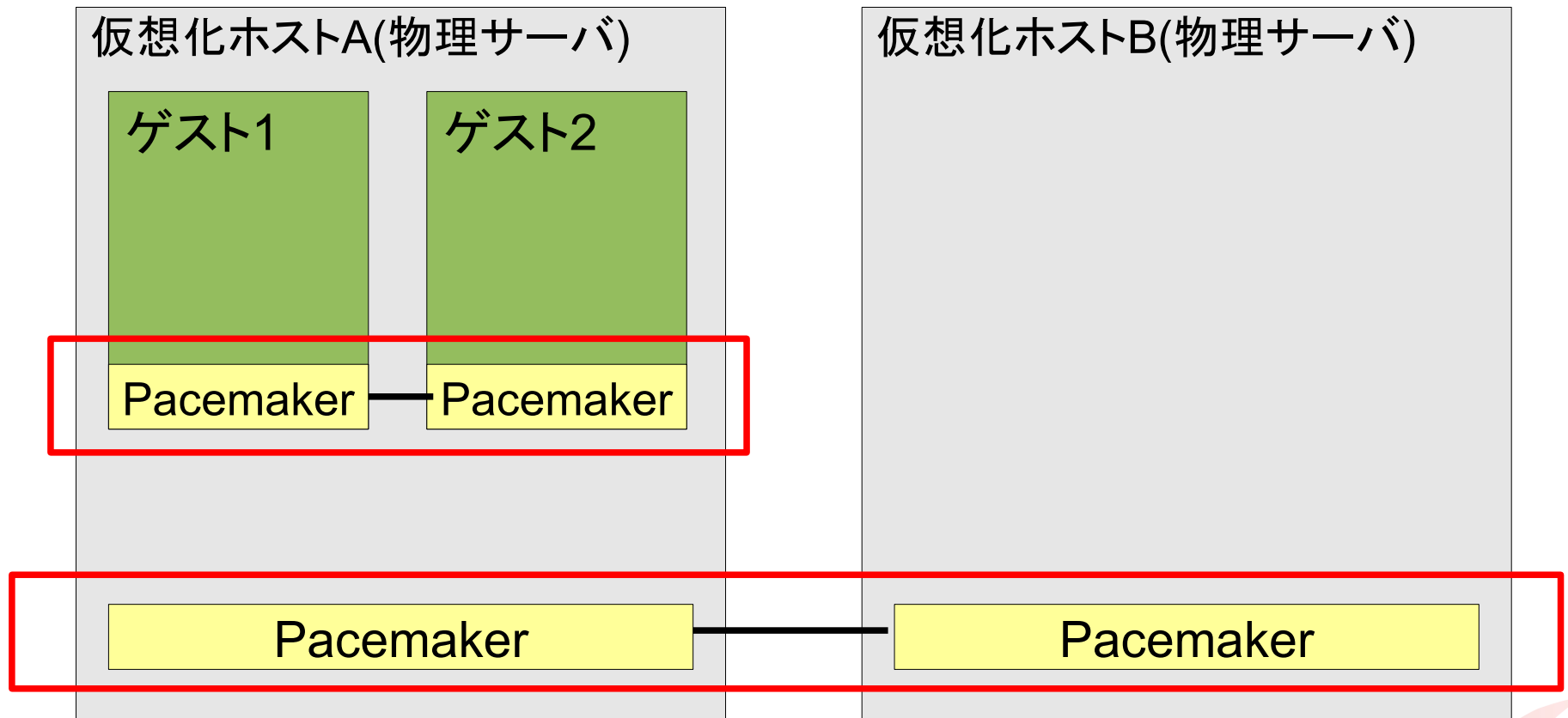
④—1

インストール・設定方法 ～仮想化ホスト・ゲスト共通で実施する作業～



Pacemakerをインストールします

- ホストとゲストで手順は同じです。



Pacemakerインストール方法の種類

1. yum を使ってネットワークインストール
 - Pacemaker本家(clusterlabs) の yumのリポジトリを使用
 - サーバにインターネット接続必須
2. ローカルリポジトリ + yum を使ってインストール
 - Linux-HA Japan 提供のリポジトリパッケージを使用
 - Linux-HA Japan オリジナルパッケージも含まれる
3. rpm を手動でインストール
 - 沢山のrpmを個別にダウンロードする必要あり
4. ソースからインストール
 - 最新の機能をいち早く試せる
 - コンポーネントが多いので、コンパイルは面倒

仮想化連携機能はこの中に
入っています

～ ローカルリポジトリ + yum を使ってインストール ～

(サーバにインターネット接続環境がなくてもOK!)

■ 1. Pacemakerリポジトリパッケージをダウンロード

Linux-HA Japan 提供の Pacemakerリポジトリパッケージを sourceforge.jp からダウンロードしておきます。

pacemaker-1.0.11-1.2.2.el6.x86_64.repo.tar.gz
をダウンロード

SourceForge.jp > ソフトウェアを探す > Linux-HA Japan > 概要

Linux-HA Japan

本ページはLinux-HA Japan 開発者向けサイトです。プロジェクトのメインサイトはこちらです <http://linux-ha.sourceforge.jp/>
Linux-HA Japanプロジェクトは、Linux上で高可用性システムを構築するための部品として、オープンソースの、クラスターソースマネージャ、クラスター通信レイヤ、ブロックデバイス複製、その他、さまざまなアプリケーションに対応するための数多くのリソースエージェント、などを、日本国内向けに維持管理、支援等を行っているプロジェクトです。
主な製品として、Pacemaker、Heartbeat、Corosync、DRBD等を取り扱っています。

[Linux-HA Japanの詳細情報へ](#)

[Linux-HA Japanのインストール方法](#)

[Linux-HA Japanの使い方](#)

最終更新日: 2010-08-23 12:44

開発メンバー: [ksk](#), [t-matsuo](#), [takayukitanaka](#), [b-oka](#), [bellche](#), [hideoyamauchi](#), [iidayuus](#), [ikedaj](#), [inouekazu](#), [jsugura](#), [kmi](#), [kitateishi](#), 他6名 [一覧]

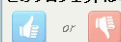
[その他の情報](#)

開発者向けページ

No Image Available

[他の画像を見る]

このプロジェクトはオススメ?



仮想化連携機能追加版は
9/16リリース

ダウンロード

最終更新日: 2010-06-18 07:21

～ ローカルリポジトリ + yum を使ってインストール ～

■ 2. Pacemaker リポジトリパッケージを展開

sourceforge.jp からダウンロードしたリポジトリパッケージを /tmp 等のディレクトリで展開します。

```
# cd /tmp
# tar zxvf pacemaker-1.0.11-1.2.2.el6.x86_64.repo.tar.gz
pacemaker-1.0.11-1.2.2.el6.x86_64.repo/
pacemaker-1.0.11-1.2.2.el6.x86_64.repo/repodata/
pacemaker-1.0.11-1.2.2.el6.x86_64.repo/repodata/primary.xml.gz
pacemaker-1.0.11-1.2.2.el6.x86_64.repo/repodata/repomd.xml
pacemaker-1.0.11-1.2.2.el6.x86_64.repo/repodata/filelists.xml.gz
pacemaker-1.0.11-1.2.2.el6.x86_64.repo/repodata/other.xml.gz
pacemaker-1.0.11-1.2.2.el6.x86_64.repo/rpm/
:
```

インストールするRPMファイルと
repoファイル等が展開されます

～ ローカルリポジトリ + yum を使ってインストール ～

■ 3. ローカルyumリポジトリを設定

展開したrepoファイルをローカルyumリポジトリとして設定します

```
# cd /tmp/pacemaker-1.0.11-1.2.2.el6.x86_64.repo/  
# vi pacemaker.repo
```

```
[pacemaker]  
name=pacemaker  
baseurl=file:///tmp/pacemaker-1.0.11-1.2.2.el6.x86_64.repo/  
gpgcheck=0  
enabled=1
```

パッケージを展開したディレクトリを指定
(デフォルトは /tmp なので、/tmpに tar.gzファイルを
展開したのならば修正不要)

～ ローカルリポジトリ + yum を使ってインストール ～

■ 4. yumでインストール！

Linux-HA Japanオリジナルパッケージも同時にインストールします。

```
# yum -c pacemaker.repo install pacemaker-1.0.11 pm_crmgen pm_diskd  
pm_logconv-hb pm_extras pm_kvm_tools vm-ctl
```

- **pm_kvm_tools-1.0-1.el6.x86_64.rpm** ……仮想化連携ツール
- **vm-ctl-1.0-1.el6.noarch.rpm** ……仮想マシンリソース制御ツール
- **pm_crmgen-1.1-1.el6.noarch.rpm** ……crm用設定ファイル編集ツール
- **pm_diskd-1.0-1.el6.x86_64.rpm** ……ディスク監視アプリとRA
- **pm_logconv-hb-1.1-1.el6.noarch.rpm** ……ログ変換ツール
- **pm_extras-1.1-1.el6.x86_64.rpm** ……その他オリジナルRA 等

クラスタ制御部基本設定

/etc/ha.d/ha.cf

- クラスタ制御部の基本設定ファイル
- クラスタ内の全ノードに同じ内容のファイルを設置

```
pacemaker on

debug 0
udpport 694
keepalive 2
warntime 7
deadtime 10
initdead 48
logfacility local1

bcast eth1 # クラスタ間通信LANの
bcast eth2 # IF名は適宜変更する

node host1 # ホスト名も適宜変更する
node host2

watchdog /dev/watchdog
respawn root /usr/lib64/heartbeat/ifcheckd
```

pm_extrasをインストールし、この ifcheckd の設定を追加すればインターコネクトLANの接続状況も確認可能です

/etc/ha.d/authkeys

- ノード間の「認証キー」を設定するファイル
- クラスタ内の全ノードに、同じ内容のファイルを配置
- 所有ユーザ/グループ・パーミッションは root/root ・ rw---- に設定

```
auth 1  
1 sha1 hogehoge
```

これも基本的に
Heartbeat2 と
設定は同じです

認証キー: 任意の文字列

認証キーの計算方法: sha1, md5, crcを指定可

/etc/syslog.conf

- 必須の設定ではないが、多くのログが /var/log/messages に出力されるため出力先を個別のファイルに変更するのがお勧め
- 以下は /var/log/ha-log への出力例
- 設定変更後は、syslogの再起動が必要

```
*.info;mail.none;authpriv.none;cron.none;local1.none  
/var/log/messages
```

```
  :  
  (省略)
```

```
  :  
local1.info
```

```
/var/log/ha-log
```

ha.cf で設定したlogfacility 名

ここまでいけば、
Pacemakerが起動できます！

```
# /etc/init.d/heartbeat start
```

← 各ノードで実行

```
Starting High-Availability services:
```

[OK]

Pacemakerの起動確認

- `crm_mon` コマンドで起動状態を確認できます

```
# crm_mon
```

```
=====
Last updated: Fri Nov 18 16:41:46 2011
Stack: Heartbeat
Current DC: host2 (44d8daf5-03e5-4a59-825f-27a964b12407) - partition
with quorum
Version: 1.0.11-1554a83db0d3c3e546cfd3aaff6af1184f79ee87
2 Nodes configured, unknown expected votes
6 Resources configured.
=====

Online: [ host1 host2 ]
```

- この段階では、まだ何のリソースも定義されていません

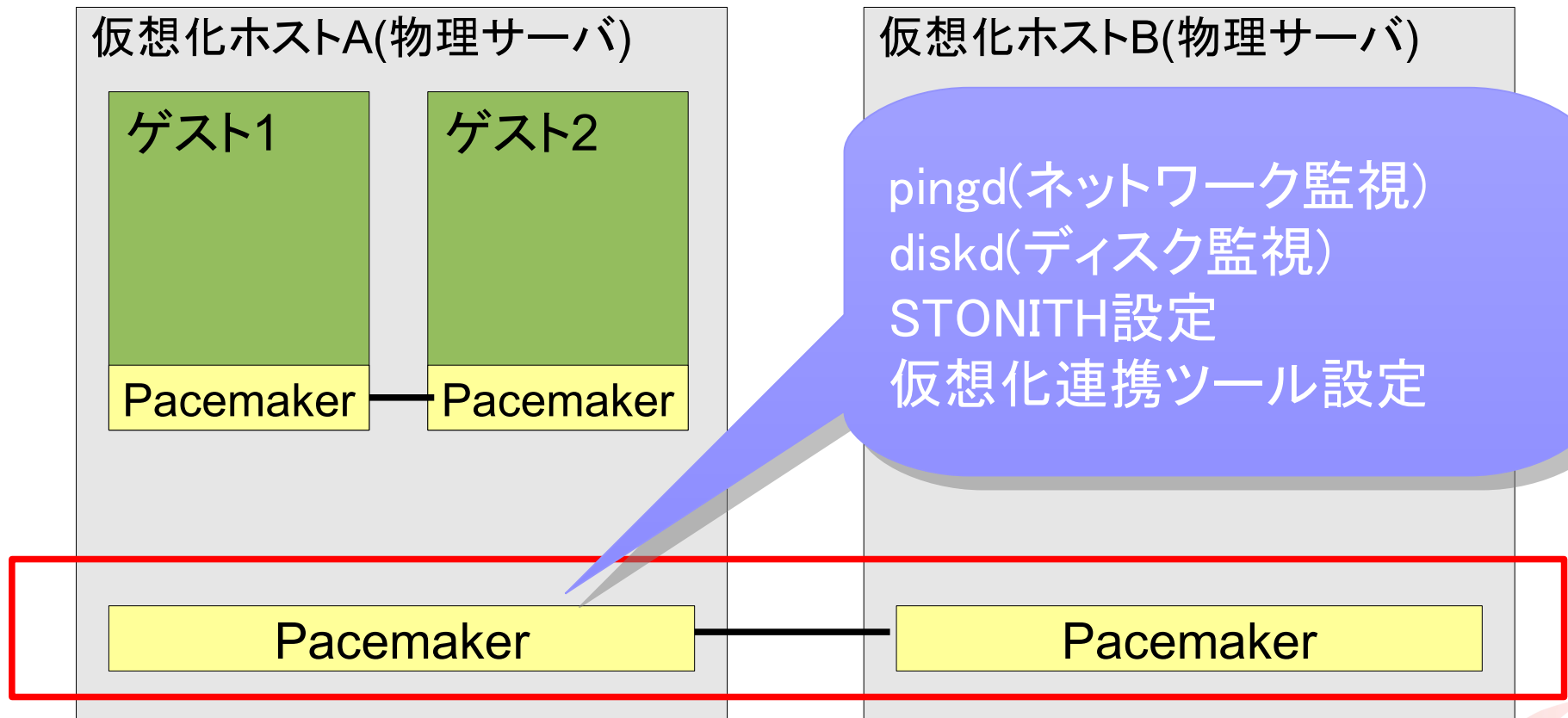
④—2

インストール・設定方法 ～仮想化ホストで実施する作業～



ホスト側のPacemakerを設定します

- 各種監視設定、仮想化連携ツール設定を追加



Pacemaker本体の追加設定(1/3)

- pingd監視を有効にします

```
# vim /tmp/pingd.crm
```

```
primitive pingCheck ocf:pacemaker:pingd \  
    params \  
        name="default_ping_set" \  
        host_list="192.168.xxx.xxx" \  
        multiplier="100" \  
    op start interval="0s" timeout="60s" on-fail="restart" \  
    op monitor interval="10s" timeout="60s" on-fail="restart" \  
    op stop interval="0s" timeout="60s" on-fail="ignore" \  
clone clnPingd pingCheck
```

```
# crm configure load update /tmp/pingd.crm
```

Pacemaker本体の追加設定(2/3)

- diskd監視を有効に設定します

```
# vim /tmp/diskd.crm
```

```
primitive diskCheck1 ocf:pacemaker:diskd \  
    params \  
        name="diskcheck_status" \  
        device="/dev/sda" \  
        interval="10" \  
    op start interval="0s" timeout="60s" on-fail="restart" \  
    op monitor interval="10s" timeout="60s" on-fail="restart" \  
    op stop interval="0s" timeout="60s" on-fail="ignore" \  
clone clnDiskd1 diskCheck1
```

```
# crm configure load update /tmp/diskd.crm
```

Pacemaker本体の追加設定(3/3)

■ STONITHを有効に設定します

```
# vim /tmp/stonith.crm
```

(長文のため個別のSTONITHリソース定義は省略)
(付録Aにcrm設定例全文を掲載します)

```
group stonith-host1 helper-host1 ipmi-host1 meatware-host1  
group stonith-host2 helper-host2 ipmi-host2 meatware-host2
```

```
location rsc_location-stonith-host1 stonith-host1 \  
    rule $id="rsc_location-stonith-host1-rule" -inf: #uname eq host1  
location rsc_location-stonith-host2 stonith-host2 \  
    rule $id="rsc_location-stonith-host2-rule" -inf: #uname eq host2
```

```
# crm configure load update /tmp/stonith.crm
```

vm-ctl基本設定(1/2)

- /etc/vm-ctl.confを環境に合わせて編集します

```
#####  
# 基本設定内容 #  
#####  
# 仮想マシン定義ファイルディレクトリ  
vm_cfg_dir="/etc/libvirt/qemu"  
# VM設定ファイル(ドメイン定義)拡張子  
vm_cfg_ext=".xml"  
# ライブマイグレーションのデフォルトでの有効/無効  
vm_allow_migrate="off"  
# STONITH設定(ゲストリソースのon_fail設定を"fence"に指定します)  
vm_stonith="on"  
# Vm制御OCF(Linux-ha japan提供のリソースエージェントを指定します)  
vm_ocf=ocf:extra:VirtualDomain  
# crmadmin タイムアウト値  
crmadmin_timeout=10000  
(次ページへ続く)
```

vm-ctl基本設定(2/2)

- /etc/vm-ctl.confを環境に合わせて編集します

(前ページの続き)

pingd制約(pingdクローンリソース名 属性名)

vm_pingd1=(**clnPingd default_ping_set**)

diskd制約(diskdクローンリソース名 属性名)

vm_diskd1=(**clnDiskd1 diskcheck_status**)

#vm_diskd2=(clnDiskd2 diskcheck_status_internal)

vm-managerd制約(リソース名 属性名)

vm_managerd1=(clnVmManagerd operator_check_status)

vm-stonithd制約(リソース名)

vm_stonithd1=(**clnVmStonithd**)

(後略)

※ リソースオペレーション設定の変更は非推奨です

pingd/diskd設定にあわせて
記述する必要があります

pm_kvm_tools基本設定(1/3)

- vm-connectdをPacemakerのリソースとして登録します

※ vm-connectdはホスト<->ゲスト間の通信用デーモン

```
# vim /tmp/vm-connectd.crm
```

```
primitive prmVmConnectd ocf:extra:vm-anything \  
  params \  
    binfile="/usr/sbin/vm-connectd" \  
    cmdline_options="-t host -d /var/lib/libvirt/qemu/" \  
    login_shell="false" \  
    op start interval="0s" timeout="60s" on-fail="restart" \  
    op monitor interval="10s" timeout="60s" on-fail="restart" \  
    op stop interval="0s" timeout="60s" on-fail="ignore" \  
  
clone clnVmConnectd prmVmConnectd
```

```
# crm configure load update/tmp/vm-connectd.crm
```

pm_kvm_tools基本設定(2/3)

- vm-stonithdをPacemakerのリソースとして登録します

```
# vim /tmp/vm-stonithd.crm
```

```
primitive prmVmStonithd ocf:extra:vm-anything \  
  params \  
    binfile="/usr/sbin/vm-stonithd" \  
    cmdline_options="-c 'openssl des-ede3 -d -base64 -k vmstonith' -i" \  
    login_shell="false" \  
    op start interval="0s" timeout="60s" on-fail="restart" \  
    op monitor interval="10s" timeout="60s" on-fail="restart" \  
    op stop interval="0s" timeout="60s" on-fail="ignore" \  
  
clone clnVmStonithd prmVmStonithd \  
order rsc_order-clnVmConnectd-clnVmStonithd 0: clnVmConnectd \  
clnVmStonithd symmetrical=true
```

```
# crm configure load update/tmp/vm-stonithd.crm
```



pm_kvm_tools基本設定(3/3)

- ホスト<->ゲスト通信用のvirtio-serial設定を追加します

```
# vim /etc/libvirt/qemu/<ゲスト名>.xml
```

```
<domain type='kvm'>
  <name>guest1</name>
  (省略)
  <devices>
  (省略)
  <channel type='unix'>
    <!-- ゲスト毎にホストとの通信用ソケットファイル名を一意に指定 -->
    <source mode='bind' path='/var/lib/libvirt/qemu/guest1'/>
    <target type='virtio' name='vmconnectd'/>
  </channel>
</devices>
</domain>
```

※ 仮想化連携機能(STONITH)を使わないゲストは設定不要

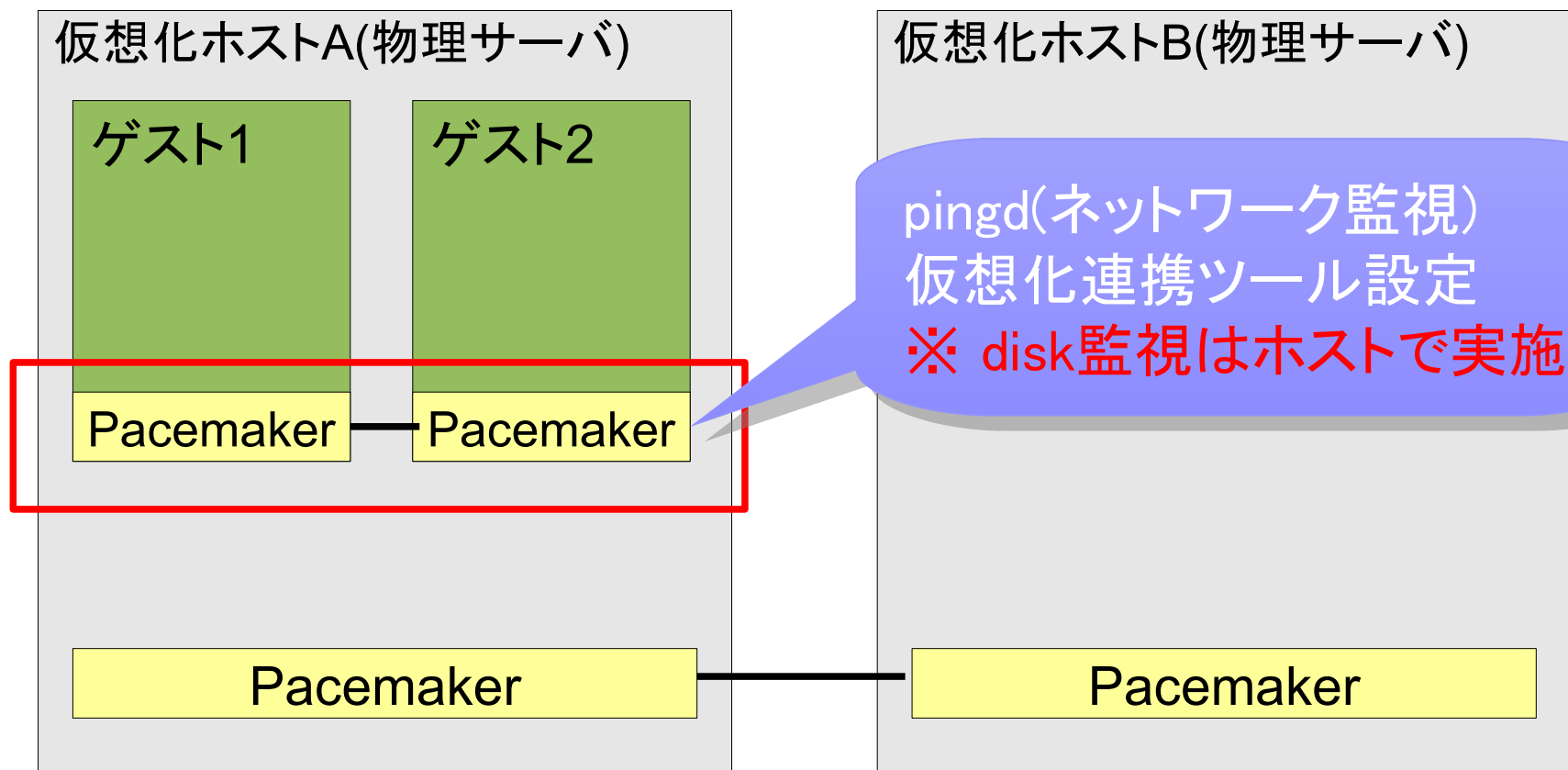
④—3

インストール・設定方法 ～仮想化ゲストで実施する作業～



ゲスト上のPacemakerを設定します

- 各種監視設定、仮想化連携ツール設定を追加



Pacemaker追加設定(1/2)

- pingd監視を有効にします

```
# vim /tmp/pingd.crm
```

```
primitive pingCheck ocf:pacemaker:pingd \  
    params \  
        name="default_ping_set" \  
        host_list="192.168.xxx.xxx" \  
        multiplier="100" \  
    op start interval="0s" timeout="60s" on-fail="restart" \  
    op monitor interval="10s" timeout="60s" on-fail="restart" \  
    op stop interval="0s" timeout="60s" on-fail="ignore" \  
clone clnPingCheck pingCheck
```

```
# crm configure load update /tmp/pingd.crm
```

Pacemaker追加設定(2/2)

■ STONITHを有効に設定します

```
# vim /tmp/stonith.crm
```

(長文のため個別のSTONITHリソース定義は省略)
(付録Bにcrm設定例全文を掲載します)

```
group stonith-guest1 helper-guest1 vm-stonith-guest1 meatware-guest1  
group stonith-guest2 helper-guest2 vm-stonith-guest2 meatware-guest2
```

```
location rsc_location-stonith-guest1 stonith-guest1 \  
    rule $id="rsc_location-stonith-guest1-rule" -inf: #uname eq guest1  
location rsc_location-stonith-guest2 stonith-guest2 \  
    rule $id="rsc_location-stonith-guest2-rule" -inf: #uname eq guest2
```

```
# crm configure load update /tmp/stonith.crm
```

pm_kvm_tools基本設定(1/1)

- vm-connectdをupstartで起動するよう設定します

```
# vim /etc/init/vm-connectd.conf
```

```
# vm-connectd
#
# Starts vm-connectd included in pm_kvm_tools package,
# it's for GUEST environment.

start on runlevel [2345]

env HA_logfacility=local1

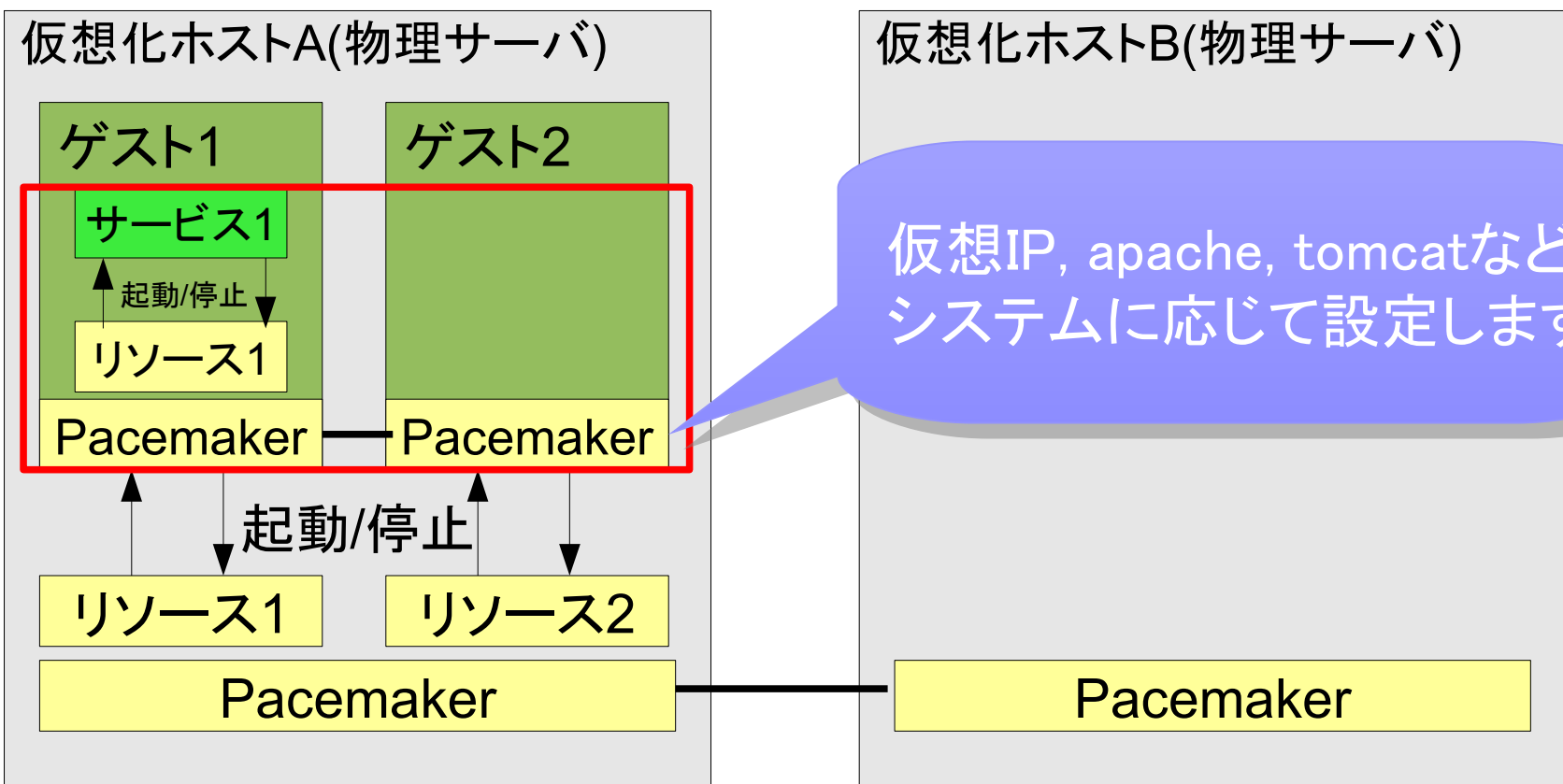
respawn
exec /usr/sbin/vm-connectd -t guest
```

※ 仮想化連携機能(STONITH)を使わないゲストは設定不要



ゲスト上のリソース設定を追加

- Pacemakerで管理したいサービスを登録します



以上でインストール・設定完了です



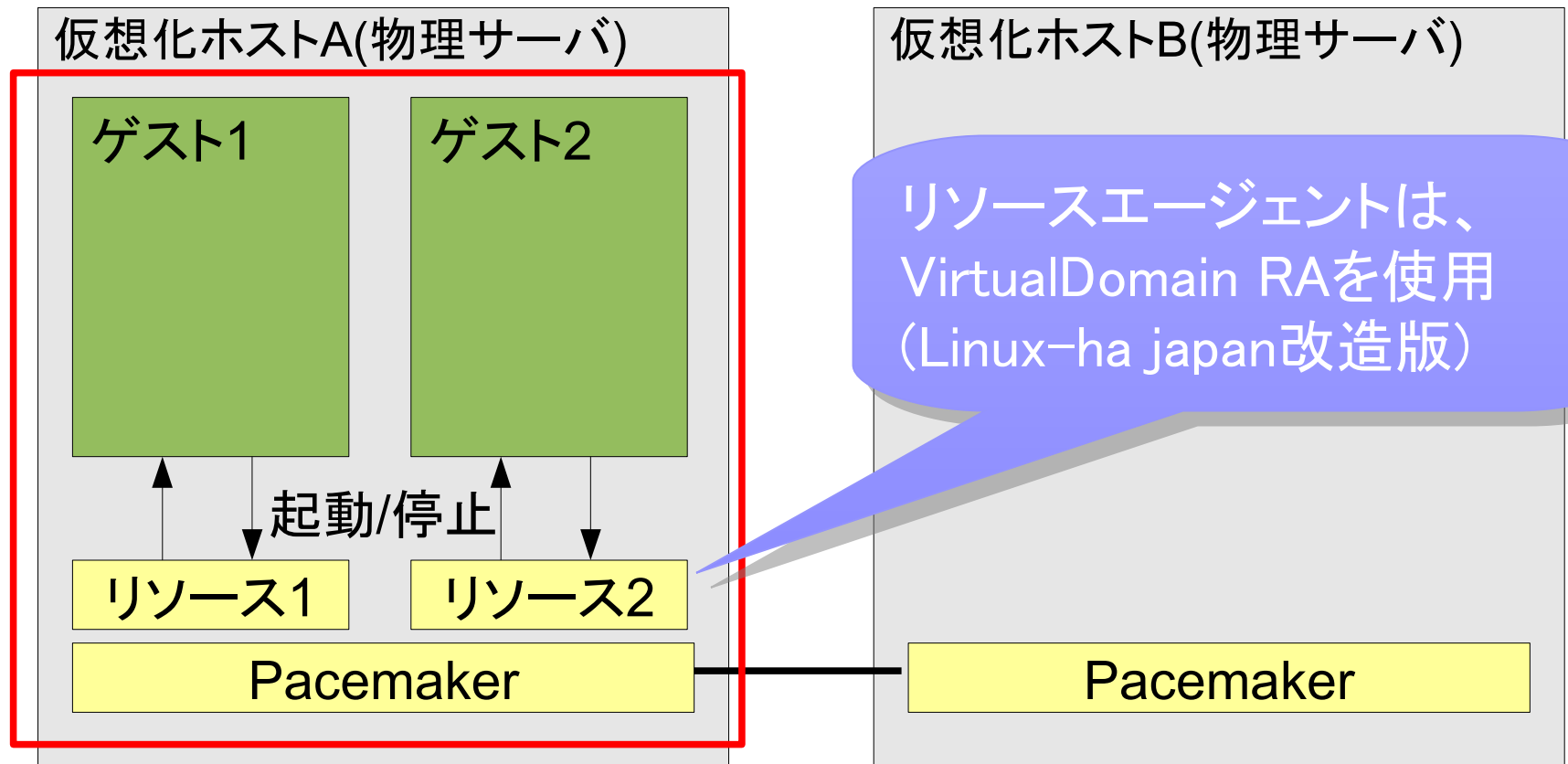
⑤

仮想化連携機能を使ってみよう
～仮想化ゲストをPacemakerで管理する～



ゲストをPacemaker管理下へ登録

- Pacemakerのリソースとして、ゲストを管理します



ゲストをリソースとして登録(1/3)

■ 以下のコマンドを実行します

```
# vm-ctl resource add guest1, guest2 -a hostA
```

書式:

```
vm-ctl resource add <domain_name> [,<domain_name>]  
-a <active_node> [-m on|off]
```

オプション:

-a(必須) 優先的にリソースを起動させるノード名
-m live migrationの有無(on又はoffの指定が可能)
未指定時はvm-ctl.confのvm_allow_migrateの設定に従う

ゲストをリソースとして登録(2/3)

- コマンド一発で、以下の設定が追加されます
 - ゲストリソースの定義(start/stop/monitor処理など)
 - 優先起動ホストの設定
 - pingd故障検知時のフェイルオーバー設定
 - diskd故障検知時のフェイルオーバー設定

ゲストをリソースとして登録(3/3)

- こんなcrm設定が2ゲスト分追加されます！

```
primitive prmVMCTL_guest1 ocf:extra:VirtualDomain \  
  params config="/etc/libvirt/qemu/guest1.xml" hypervisor="qemu:///system" migration_transport="ssh" \  
  meta allow-migrate="false" target-role="Stopped" \  
  op start interval="0" timeout="120s" on-fail="restart" \  
  op monitor interval="10s" timeout="30s" on-fail="restart" \  
  op stop interval="0" timeout="90s" on-fail="fence" \  
  op migrate_to interval="0" timeout="300s" on-fail="fence" \  
  op migrate_from interval="0" timeout="240s" on-fail="restart" \  
location locVMCTL_guest1_default_ping_set prmVMCTL_guest1 \  
  rule $id="locVMCTL_guest1_default_ping_set-rule" -inf: not_defined default_ping_set or \  
  default_ping_set lt 100 \  
location locVMCTL_guest1_diskcheck_status prmVMCTL_guest1 \  
  rule $id="locVMCTL_guest1_diskcheck_status-rule" -inf: not_defined diskcheck_status or \  
  diskcheck_status eq ERROR \  
location locVMCTL_guest1_host1_ACT prmVMCTL_guest1 200: host1 \  
colocation colVMCTL_guest1_clnDiskd1 inf: prmVMCTL_guest1 clnDiskd1 \  
colocation colVMCTL_guest1_clnPingd inf: prmVMCTL_guest1 clnPingd \  
colocation colVMCTL_guest1_clnVmStonithd inf: prmVMCTL_guest1 clnVmStonithd \  
order odrVMCTL_guest1_clnDiskd1 0: clnDiskd1 prmVMCTL_guest1 symmetrical=false \  
order odrVMCTL_guest1_clnPingd 0: clnPingd prmVMCTL_guest1 symmetrical=false \  
order odrVMCTL_guest1_clnVmStonithd 0: clnVmStonithd prmVMCTL_guest1 symmetrical=false
```

Pacemaker管理下のゲストを起動

- 以下のコマンドで、ゲストを起動します

```
# vm-ctl resource start guest1, guest2
```

書式:

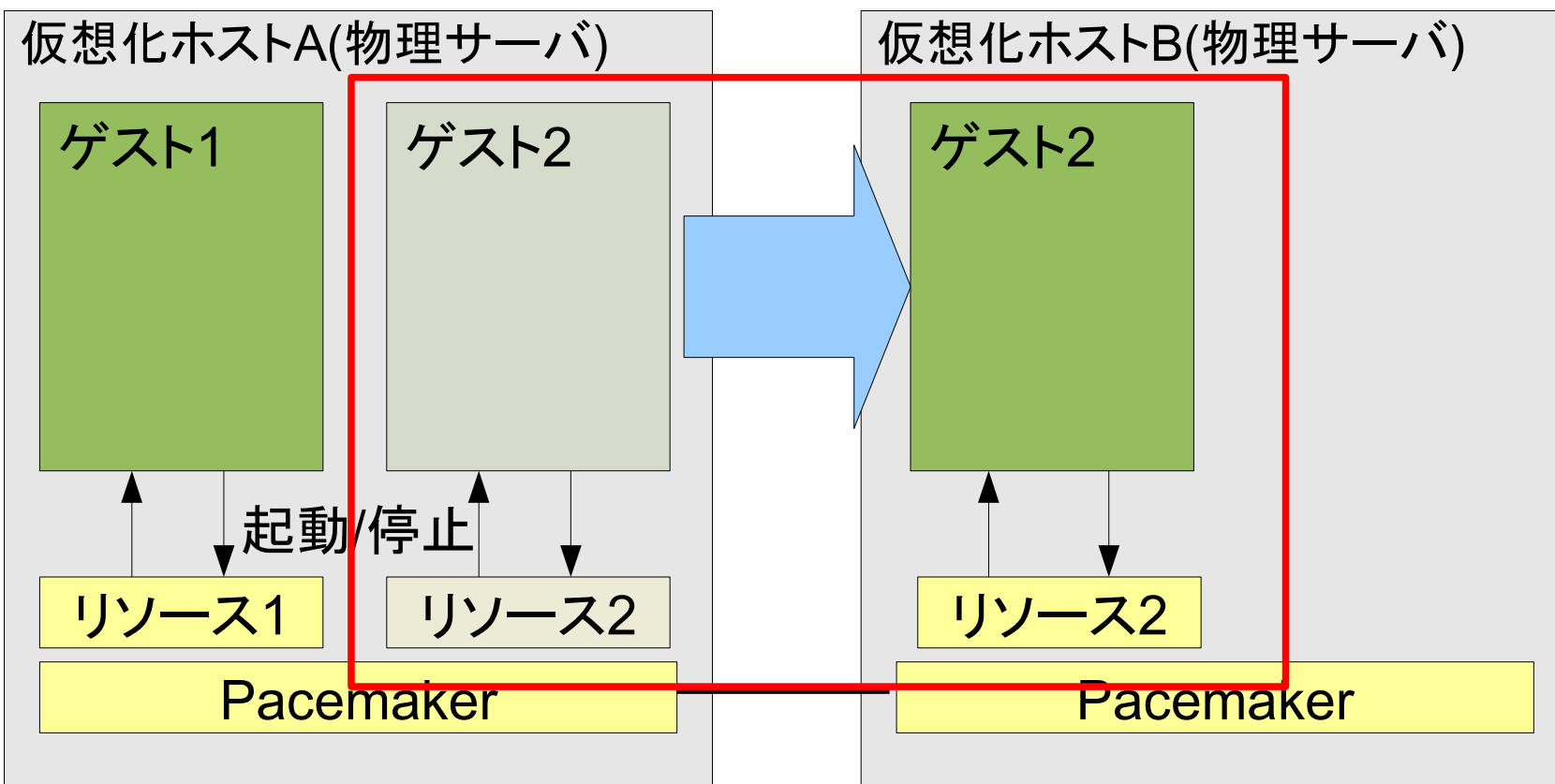
```
vm-ctl resource start <domain_name>[,<domain_name>]
```

オプション: なし



ゲストを別ホストへ移動

- `vm-ctl`コマンドで移動が可能です



ゲストを別ホストへ移動

- 以下のコマンドで、ゲストを移動させる

```
# vm-ctl resource move guest2 -n hostB
```

書式:

```
vm-ctl resource move <domain_name> [-n <node_name>]
```

オプション:

-n リソースを起動させるノード名。このオプションを指定しない場合は Pacemakerの起動優先度が最も高いノードに移動します。

ゲストの優先起動ホストの変更

- 以下のコマンドで、今ゲストが起動しているホストが優先起動ホストに設定されます。

```
# vm-ctl location move guest2
```

書式:

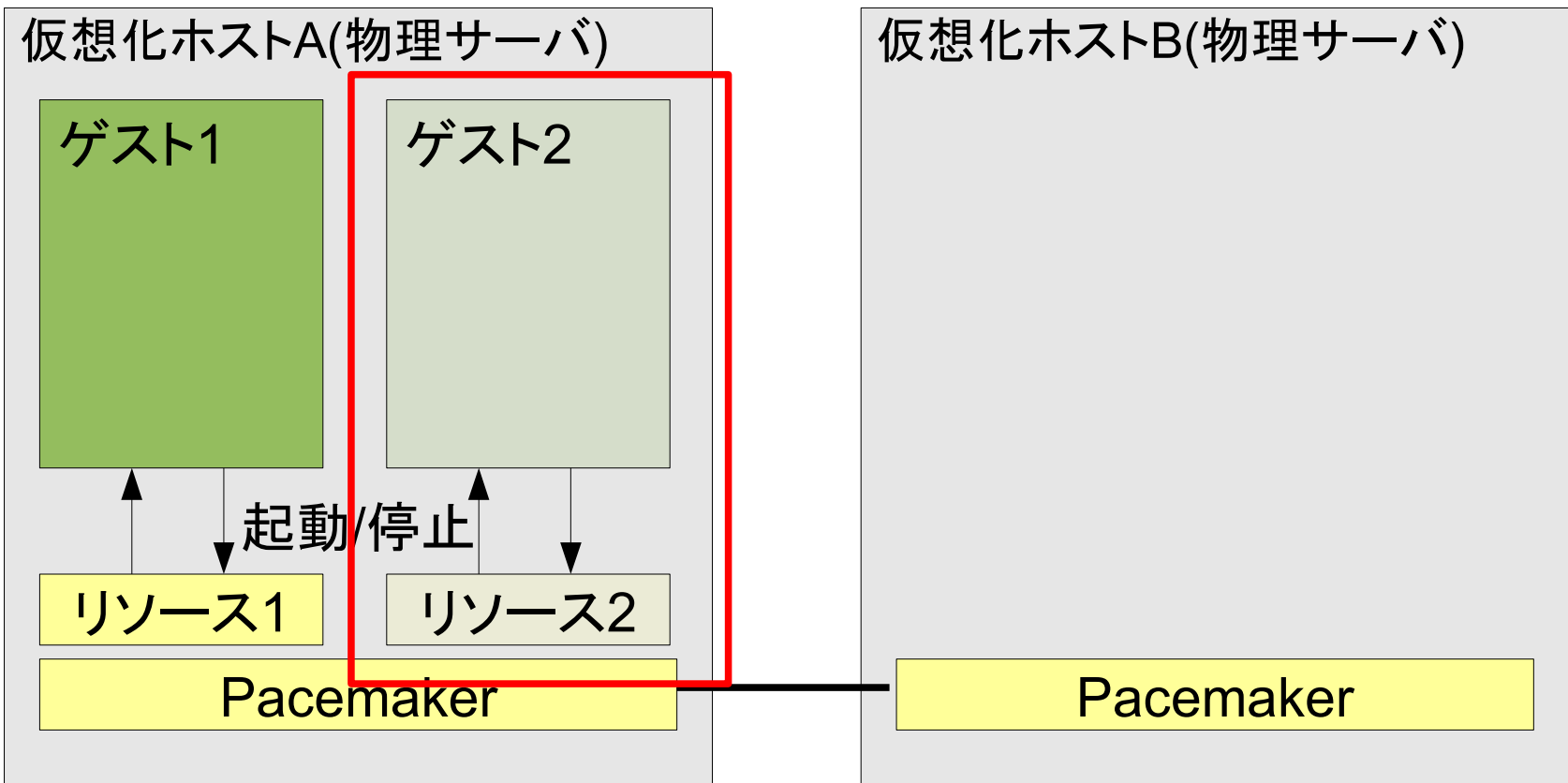
```
vm-ctl location move <domain_name> | -b <node_name>
```

オプション:

-b 変更元ノード名を指定し、当該ノード上の全ゲストの配置制約を変更

ゲストの停止・Pacemaker登録削除

- `vm-ctl`コマンドで停止・登録削除が可能です



ゲストの停止

- 以下のコマンドで、ゲストを停止します

```
# vm-ctl resource stop guest2
```

書式:

```
vm-ctl resource stop <domain_name>[,<domain_name>]
```

オプション: なし



Pacemaker管理下からゲストを削除

- 以下のコマンドで、ゲストが管理外となります

```
# vm-ctl resource delete guest2
```

書式:

```
vm-ctl resource delete <domain_name>[,<domain_name>]
```

オプション: なし

- ゲストリソースに関連したcrm設定をすべて削除
- ゲストのイメージファイル・ドメイン定義は残ります

以上です。
ご清聴ありがとうございました。



付録

付録A:ホストのSTONITH設定(1/4)

■ stonithリソースを以下のとおり定義します

```
# vim /tmp/stonith.xml
```

```
primitive helper-host1 stonith:external/stonith-helper \  
  params \  
    priority="1" \  
    stonith-timeout="40" \  
    hostlist="host1" \  
    dead_check_target="192.168.xxx.xxx 192.168.yyy.xxx" \  
    standby_check_command="/usr/sbin/crm_resource -r prmVMCTL_guest1 -W | grep -q `hostname`" \  
 \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="10s" timeout="60s" \  
  op stop interval="0s" timeout="60s"
```

(次ページへ続く)

host1に割り当てられているIPアドレスのうち、host2側からping到達可能なものをすべてここに列挙します。
このIPアドレスすべてがping不達になると、host1は既に停止していると判定します。

standby_check_commandの -rオプションに、現用機で稼動しているゲストリソースのうち、代表1つのリソース名を指定します。
スプリットブレイン発生時、ここで指定したリソースが稼動しているホストを現用機とみなします。

付録A:ホストのSTONITH設定(2/4)

■ (前ページからの続き)

```
primitive helper-host2 stonith:external/stonith-helper \  
  params \  
    priority="1" \  
    stonith-timeout="40" \  
    hostlist="host2" \  
    dead_check_target="192.168.xxx.zzz 192.168.yyy.zzz" \  
    standby_check_command="/usr/sbin/crm_resource -r prmVMCTL_guest1 -W | grep -q `hostname`" \  
 \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="10s" timeout="60s" \  
  op stop interval="0s" timeout="60s" \  
primitive ipmi-host1 stonith:external/ipmi \  
  params \  
    priority="2" \  
    stonith-timeout="60" \  
    hostlist="host1" \  
    ipaddr="192.168.xxx.xxx" \  
    userid="IPMIUser" \  
    passwd="passwd" \  
    interface="lanplus" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="3600s" timeout="60s" \  
  op stop interval="0s" timeout="60s"
```

環境に応じて、以下の設定を変更します。
ipaddr : IPMIデバイスの接続先IP
userid : IPMIデバイスのログインユーザ名
passwd : IPMIデバイスのログインパスワード

(次ページへ続く)

付録A: ホストのSTONITH設定(3/4)

■ (前ページからの続き)

```
primitive ipmi-host2 stonith:external/ipmi \  
  params \  
    priority="2" \  
    stonith-timeout="60" \  
    hostlist="host2" \  
    ipaddr="192.168.xxx.xxx" \  
    userid="IPMIUser" \  
    passwd="passwd" \  
    interface="lanplus" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="3600s" timeout="60s" \  
  op stop interval="0s" timeout="60s"
```

```
primitive meatware-host1 stonith:meatware \  
  params \  
    priority="3" \  
    stonith-timeout="600" \  
    hostlist="host1" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="3600s" timeout="60s" \  
  op stop interval="0s" timeout="60s"
```

(次ページへ続く)

付録A:ホストのSTONITH設定(4/4)

■ (前ページからの続き)

```
primitive meatware-host2 stonith:meatware \  
  params \  
    priority="3" \  
    stonith-timeout="600" \  
    hostlist="host2" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="3600s" timeout="60s" \  
  op stop interval="0s" timeout="60s" \  
  
group stonith-host1 helper-host1 ipmi-host1 meatware-host1 \  
group stonith-host2 helper-host2 ipmi-host2 meatware-host2 \  
  
location rsc_location-stonith-host1 stonith-host1 \  
  rule $id="rsc_location-stonith-host1-rule" -inf: #uname eq host1 \  
location rsc_location-stonith-host2 stonith-host2 \  
  rule $id="rsc_location-stonith-host2-rule" -inf: #uname eq host2
```

付録B: ゲストのSTONITH設定(1/4)

- stonithリソースを以下のとおり定義します

```
# vim /tmp/guest-stonith.xml
```

```
primitive helper-guest1 stonith:external/stonith-helper \  
  params \  
    priority="1" \  
    stonith-timeout="40" \  
    hostlist="guest1" \  
    dead_check_target="192.168.xxx.xxx 192.168.yyy.xxx" \  
    standby_check_command="/usr/sbin/crm_resource -r guest_resource1 -W | grep -q `hostname`" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="10s" timeout="60s" \  
  op stop interval="0s" timeout="60s" \  
primitive helper-guest2 stonith:external/stonith-helper \  
  params \  
    priority="1" \  
    stonith-timeout="40" \  
    hostlist="guest2" \  
    dead_check_target="192.168.xxx.zzz 192.168.yyy.zzz" \  
    standby_check_command="/usr/sbin/crm_resource -r guest_resource1 -W | grep -q `hostname`" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="10s" timeout="60s" \  
  op stop interval="0s" timeout="60s"
```

(次ページへ続く)

付録B: ゲストのSTONITH設定(2/4)

■ (前ページからの続き)

```
# vim /tmp/guest-stonith.xml
```

```
primitive vm-stonith-guest1 stonith:external/vm-stonith \  
  params \  
    priority="2" \  
    stonith-timeout="30s" \  
    hostlist="guest1:U2FsdGVkX1/0NmOPdK77shIGkagLA5RdgVghb7MdCdaggiLLrS01Fw==" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="3600s" timeout="60s" \  
  op stop interval="0s" timeout="60s" \  
primitive vm-stonith-guest2 stonith:external/vm-stonith \  
  params \  
    priority="2" \  
    stonith-timeout="30s" \  
    hostlist="guest2:U2FsdGVkX19OO1zVKCGneLBCaGTaGLZ7gLQiNnpLxRAcmJUOjnZrYg==" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="3600s" timeout="60s" \  
  op stop interval="0s" timeout="60s"
```

(次ページへ続く)

hostlistには、“ゲスト名:暗号化したゲストリソース名”を指定します。
“暗号化したゲストリソース名”の作成手順は、付録B(4/4)で解説します。

付録B: ゲストのSTONITH設定(3/4)

■ (前ページからの続き)

```
primitive meatware-guest1 stonith:meatware \  
  params \  
    priority="3" \  
    stonith-timeout="600" \  
    hostlist="guest1" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="3600s" timeout="60s" \  
  op stop interval="0s" timeout="60s" \  
primitive meatware-guest2 stonith:meatware \  
  params \  
    priority="3" \  
    stonith-timeout="600" \  
    hostlist="guest2" \  
  op start interval="0s" timeout="60s" \  
  op monitor interval="3600s" timeout="60s" \  
  op stop interval="0s" timeout="60s" \  
  
group stonith-guest1 helper-guest1 vm-stonith-guest1 meatware-guest1 \  
group stonith-guest2 helper-guest2 vm-stonith-guest2 meatware-guest2 \  
  
location rsc_location-stonith-guest1 stonith-guest1 \  
  rule $id="rsc_location-stonith-guest1-rule" -inf: #uname eq guest1 \  
location rsc_location-stonith-guest2 stonith-guest2 \  
  rule $id="rsc_location-stonith-guest2-rule" -inf: #uname eq guest2
```

付録B: ゲストのSTONITH設定(4/4)

■ 暗号化したゲストリソース名の作成

- 仮想化ホスト上で、以下のコマンドを実行します

```
hostA# echo "prmVMCTL_guest1" | openssl des-ede3 -e -base64 -k vmstonith  
U2FsdGVkX18Gh0VsgX6ze9TaOkigwXAYX3weRM8q2HFG+ppSGNhUqg==
```

```
hostA# echo "prmVMCTL_guest2" | openssl des-ede3 -e -base64 -k vmstonith  
U2FsdGVkX19OO1zVKCGneLBCaGTaGLZ7gLQINnpLxRAcmJUOjnZrYg==
```

各ゲストに対応する、ホスト上のPacemakerが管理するゲストリソース名を指定します。ゲストから送信されたSTONITH要求メッセージはホストで復号され、ホストのPacemakerが対象ゲストリソースを停止することで、STONITHが実現されます。

なお、ゲストリソース名をそのまま vm-stonithプラグインの設定に書かせない理由は、複数のゲスト利用ユーザがいる場合、あるユーザが自分のゲストリソース名から他人のゲストリソース名を推測し、STONITHを撃つような事態を防止するためです。