

CRM-CLIでHAクラスタ を自在に制御しよう！

2013年10月19日 OSC2013 Tokyo/Fall
Linux-HA Japan
東 一彦

本日のお話

- ① Pacemakerって何？
- ② CRMでPacemakerを設定する
- ③ CRM設定を読む
- ④ 制約を駆使してみる(デモ)
- ⑤ Linux-HA Japan について



①

Pacemakerって何？

Pacemakerはオープンソースの
HAクラスタソフトです。

High Availability = 高可用性

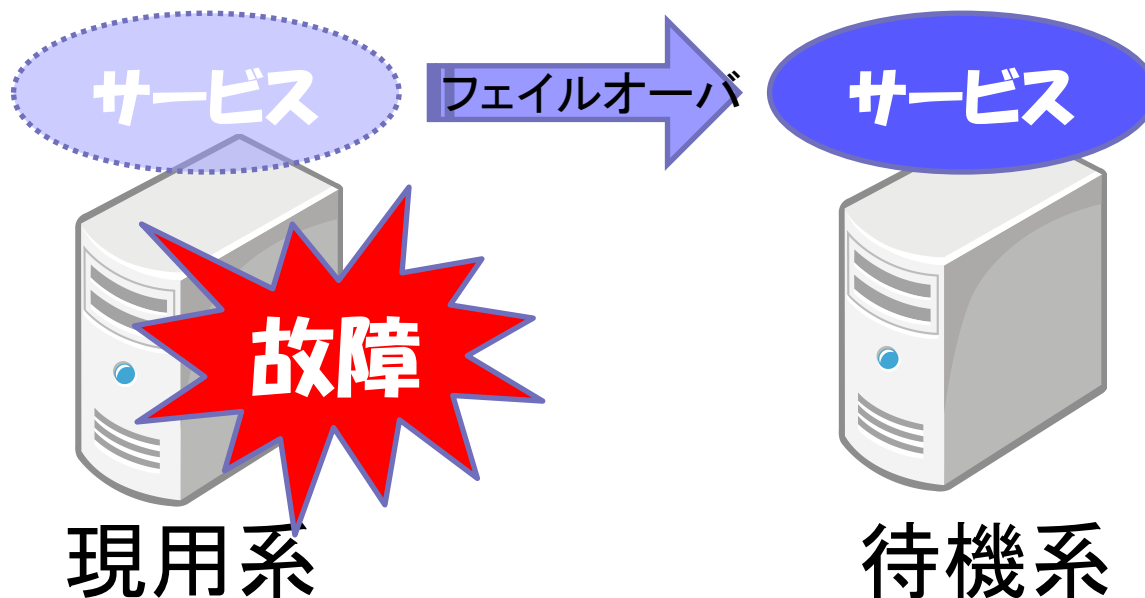
つまり

サービス継続性

一台のコンピュータでは得られない
高い信頼性を狙うために、
複数のコンピュータを結合し、
ひとまとまりとしたシステムのこと



HAクラスタを導入すると、
故障で現用系でサービスができなくなったとき
に、自動で待機系にサービスを起動させます
→このことを「**フェイルオーバー**」と言います

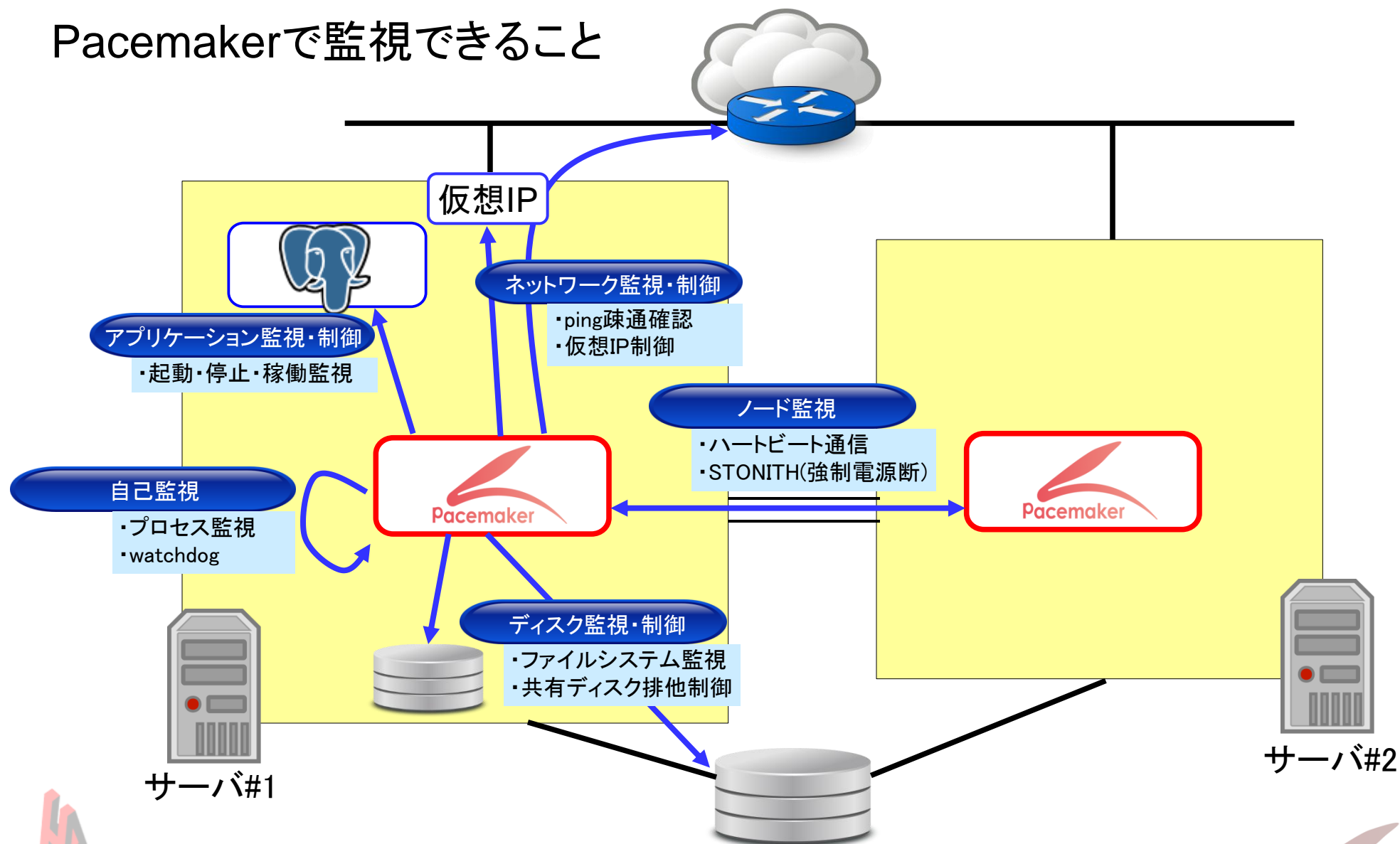




は
このHAクラスタソフトとして
実績のある「Heartbeat」と
呼ばれていたソフトの後継です。

何を監視できるの？

Pacemakerで監視できること



少しでも用語を紹介

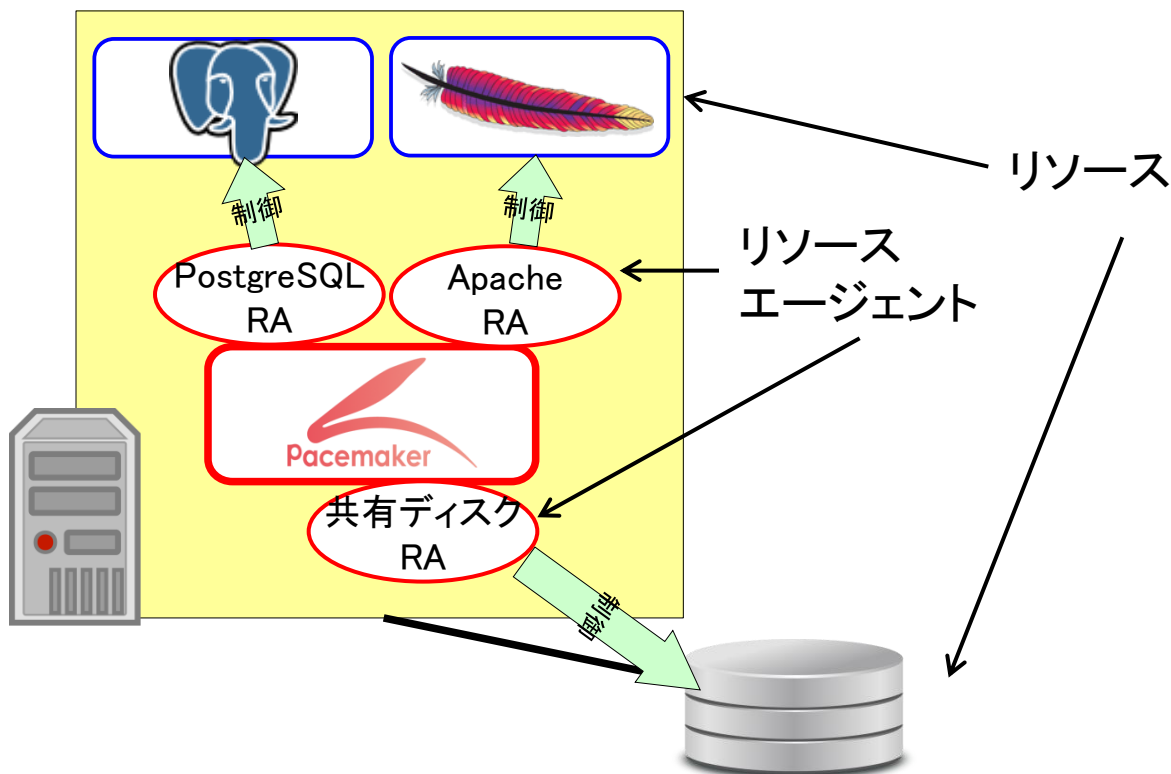
□ Pacemakerが起動/停止/監視を制御する対象をリソースと呼ぶ

■ 例) Apache, PostgreSQL, 共有ディスク, 仮想IPアドレス etc...

□リソースの制御はリソースエージェント(RA)を介して行う

■ RAが各リソースの操作方法の違いをラップし、Pacemakerで制御できるようにしている

□ 多くはシェルスクリプト





②

CRMでPacemakerを設定する



CRM-CLIって？

CRM-CLIは、
Cluster Resource Manager -
Command Line Interface
の略です。

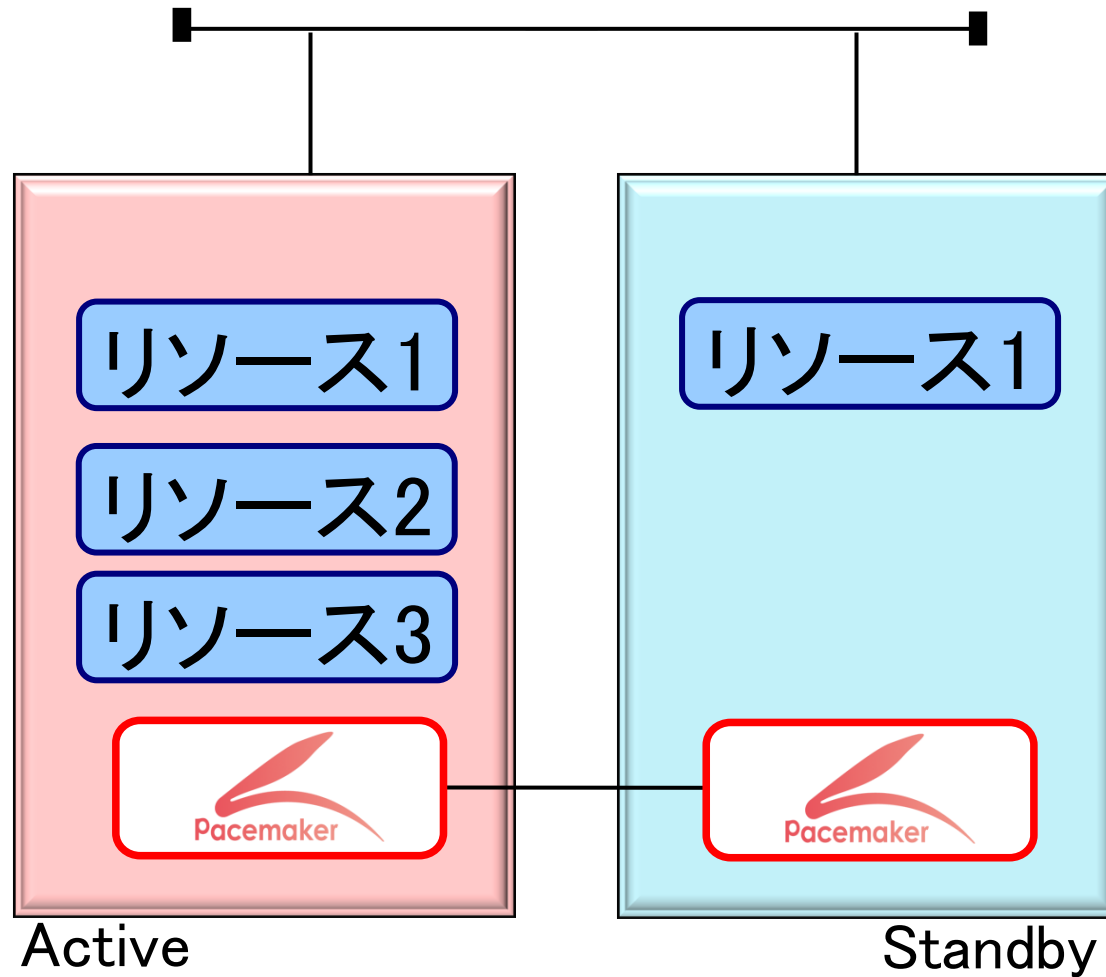
Pacemakerは大きく2つの制御部に
分かれています。



- ・・・「リソース」をどう配置、制御するかを決める
- ・・・他ノードの監視、制御を行う

そのうち「リソース制御部」を制御する
インターフェースがCRM-CLIです。

突然ですが、
こんなクラスタを組みたいとします。



ちょっと抽象的すぎるかもしれませんね。

では、リソース1～3が以下に対応しているとしたらどうでしょう？

リソース1=ネットワーク(NW)監視

リソース2=ファイルシステムのマウント・監視

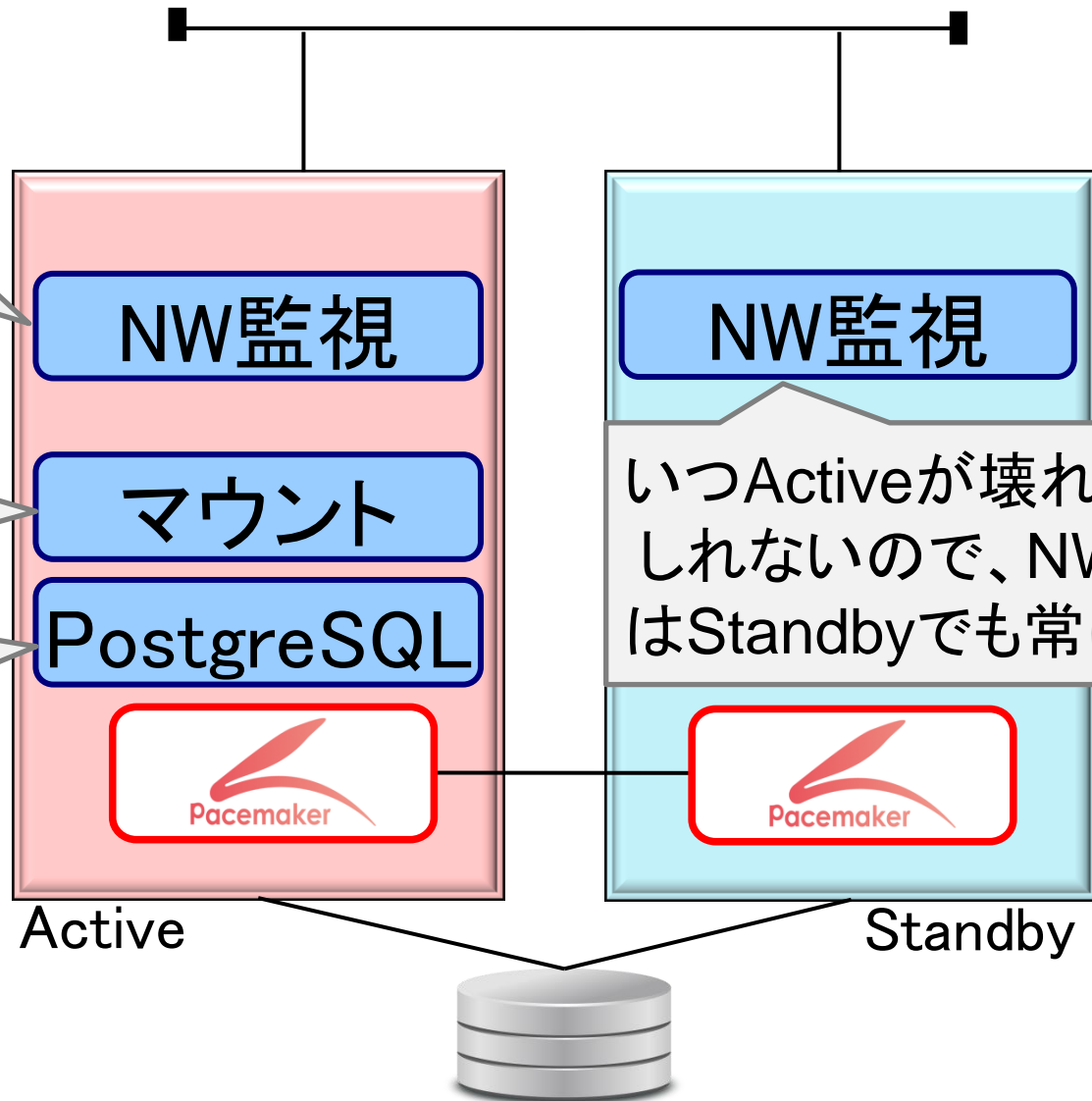
リソース3=DBMS(PostgreSQL)




① ネットワークに
異常がないことを確認
してから..

② DBファイルのある
ファイルシステムを
マウントし..

③ PostgreSQLを起動





PacemakerでDBをクラスタ化する際の、
典型的な構成です。※

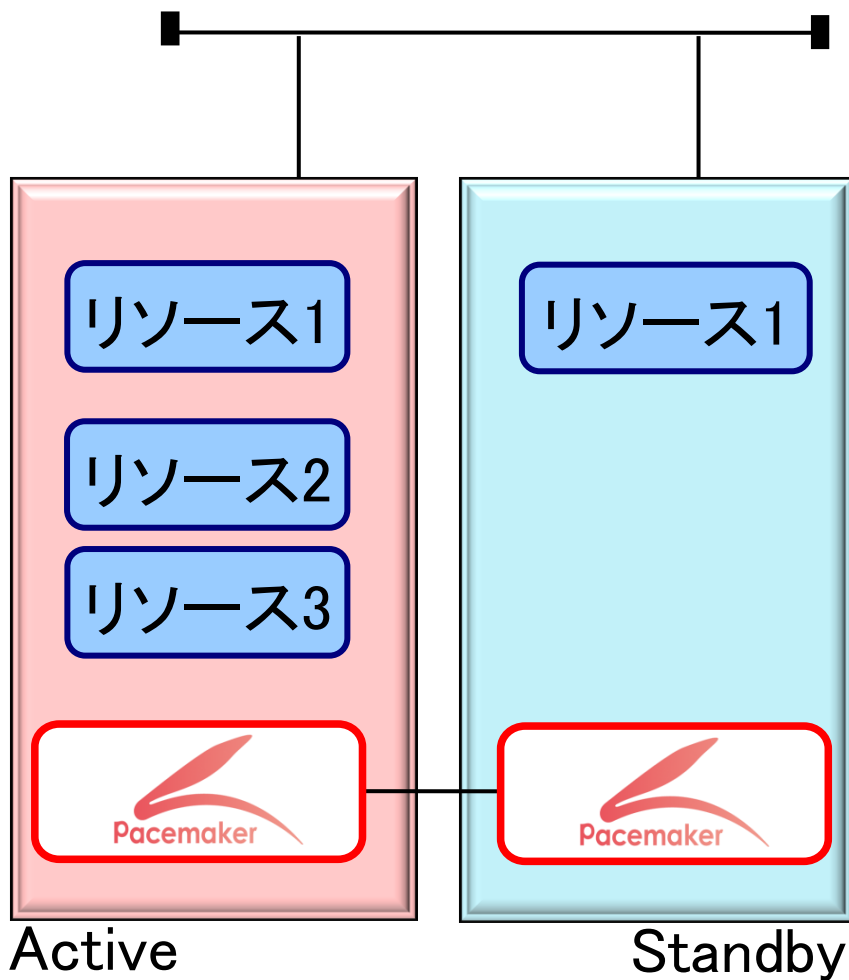
このように、リソース間の起動順や、
起動場所を指定することは、非常に重要です。

そういったことを定義、設定するのが
CRM設定です。



③ CRM設定を読む

さっきの動作を定義したCRM設定を
見てみましょう。



```
property no-quorum-policy="ignore" stonith-enabled="false" ¥
        crmd-transition-delay="2s"
```

```
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"
```

```
primitive resource1 ocf:heartbeat:Dummy ¥
    op start interval="0s" timeout="300s" on-fail="restart" ¥
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥
    op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource2 ocf:heartbeat:Dummy ¥
    op start interval="0s" timeout="300s" on-fail="restart" ¥
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥
    op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource3 ocf:heartbeat:Dummy ¥
    op start interval="0s" timeout="300s" on-fail="restart" ¥
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥
    op stop interval="0s" timeout="300s" on-fail="block"
```


```
clone clnResource resource1
```

```
group grp resource2 resource3
```

```
location loc1 grp ¥
    rule 200: #uname eq pm01 ¥
    rule 100: #uname eq pm02
```

```
colocation col1 INFINITY: grp clnResource
```

```
order order1 0: clnResource grp symmetrical=false
```



このCRM設定、よく見ると行頭は以下の
8種類のコマンドのいずれかになっています。

- property
- rsc_defaults
- primitive
- clone
- group
- location
- colocation
- order

この8コマンドがあれば、たいがいの構成は
設定できます。



8コマンドを(強引に)3つに分類

- property
 - rsc_defaults
- } クラスタ全体の共通設定。
-
- primitive
 - clone
 - group
- } どんなリソースを使用するかを定義。
クラスタ設定の基本。
-
- location
 - colocation
 - order
- } リソースの起動順や、場所などを「制約」する。
使いこなせばどんなクラスタもへっちゃら！？



というわけで、
先ほどのCRM設定を読んでいます。

```

property no-quorum-policy="ignore" stonith-enabled="false" ¥
    crmd-transition-delay="2s"

rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"

primitive resource1 ocf:heartbeat:Dummy ¥
    op start interval="0s" timeout="300s" on-fail="restart" ¥
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥
    op stop interval="0s" timeout="300s" on-fail="block"

primitive resource2 ocf:heartbeat:Dummy ¥
    op start interval="0s" timeout="300s" on-fail="restart" ¥
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥
    op stop interval="0s" timeout="300s" on-fail="block"

primitive resource3 ocf:heartbeat:Dummy ¥
    op start interval="0s" timeout="300s" on-fail="restart" ¥
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥
    op stop interval="0s" timeout="300s" on-fail="block"

clone clnResource resource1

group grp resource2 resource3

location loc1 grp ¥
    rule 200: #uname eq pm01 ¥
    rule 100: #uname eq pm02

colocation col1 INFINITY: grp clnResource

order order1 0: clnResource grp symmetrical=false

```

青字はSTONITHを使用しないことを表します。あとはおまじないです。

・・・いきなりさぼってすみません。

・・・ですが話すと長くなるので、今回は「おまじない」として触れないことにします。

知りたい人は以下を参照してください。

<http://linux-ha.sourceforge.jp/wp/archives/3786>

```
property no-quorum-policy="ignore" stonith-enabled="false" ¥  
crmd-transition-delay="2s"
```

```
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"
```

```
primitive resource1 ocf:heartbeat:Dummy ¥  
  op start interval="0s" timeout="300s" on-fail="restart" ¥  
  op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
  op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource2 ocf:heartbeat:Dummy ¥  
  op start interval="0s" timeout="300s" on-fail="restart" ¥  
  op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
  op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource3 ocf:heartbeat:Dummy ¥  
  op start interval="0s" timeout="300s" on-fail="restart" ¥  
  op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
  op stop interval="0s" timeout="300s" on-fail="block"
```

```
clone clnResource resource1
```

```
group grp resource2 resource3
```

```
location loc1 grp ¥  
  rule 200: #uname eq pm01 ¥  
  rule 100: #uname eq pm02
```

```
colocation col1 INFINITY: grp clnResource
```

```
order order1 0: clnResource grp symmetrical=false
```

primitiveコマンドを使用して、
使用するリソースを3つ定義
しています。

primitiveコマンド とは・・

概要 : クラスタ内で使用する**リソースを定義**します。

書式 : primitive リソースID RA名
[meta リソースの動作設定...]
[params RAに渡すパラメータ...]
[op start|stop|monitor オペレーション時の設定...]

リソースID : 任意の英数字で一意的な名前を設定します。

RA名 : 使用するRA(ファイル)を指定します。

[]内の説明は省略します。

知りたい方は以下をご覧ください。

<http://linux-ha.sourceforge.jp/wp/archives/3855>

```
property no-quorum-policy="ignore" stonith-enabled="false" ¥  
    crmd-transition-delay="2s"
```

```
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"
```

```
primitive resource1 ocf:heartbeat:Dummy ¥  
    op start interval="0s" timeout="300s" on-fail="restart" ¥  
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
    op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource2 ocf:heartbeat:Dummy ¥  
    op start interval="0s" timeout="300s" on-fail="restart" ¥  
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
    op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource3 ocf:heartbeat:Dummy ¥  
    op start interval="0s" timeout="300s" on-fail="restart" ¥  
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
    op stop interval="0s" timeout="300s" on-fail="block"
```

```
clone clnResource resource1
```

```
group grp resource2 resource3
```

```
location loc1 grp ¥  
    rule 200: #uname eq pm01 ¥  
    rule 100: #uname eq pm02
```

```
colocation col1 INFINITY: grp clnResource
```

```
order order1 0: clnResource grp symmetrical=false
```

} **cloneコマンド**を使用して、
resource1をクローン化しています。

cloneコマンド とは・・

概要：指定したリソースが、**複数ノードで同時に起動する**ようにします。

→同じリソースが複製(クローン)されるイメージ

書式： clone クローンリソースID リソースID...
[meta クローンリソース動作設定...]

クローンリソースID :任意の英数字で一意的な名前を設定します。
リソースID :クローン化するリソースをIDで指定します。

[]内の説明は省略します。

知りたい方は以下をご覧ください。

<http://linux-ha.sourceforge.jp/wp/archives/3855>

```
property no-quorum-policy="ignore" stonith-enabled="false" ¥  
    crmd-transition-delay="2s"  
  
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"  
  
primitive resource1 ocf:heartbeat:Dummy ¥  
    op start interval="0s" timeout="300s" on-fail="restart" ¥  
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
    op stop interval="0s" timeout="300s" on-fail="block"  
  
primitive resource2 ocf:heartbeat:Dummy ¥  
    op start interval="0s" timeout="300s" on-fail="restart" ¥  
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
    op stop interval="0s" timeout="300s" on-fail="block"  
  
primitive resource3 ocf:heartbeat:Dummy ¥  
    op start interval="0s" timeout="300s" on-fail="restart" ¥  
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
    op stop interval="0s" timeout="300s" on-fail="block"  
  
clone clnResource resource1  
  
group grp resource2 resource3  
  
location loc1 grp ¥  
    rule 200: #uname eq pm01 ¥  
    rule 100: #uname eq pm02  
  
colocation col1 INFINITY: grp clnResource  
  
order order1 0: clnResource grp symmetrical=false
```

} **groupコマンド**を使用して、
resource2,3をグループ化し
ています。

groupコマンド とは・・

概要：指定したリソースが「必ず同一ノードで」
「指定した順に」起動するよう設定します。
→リソース達が仲良しグループになるイメージ

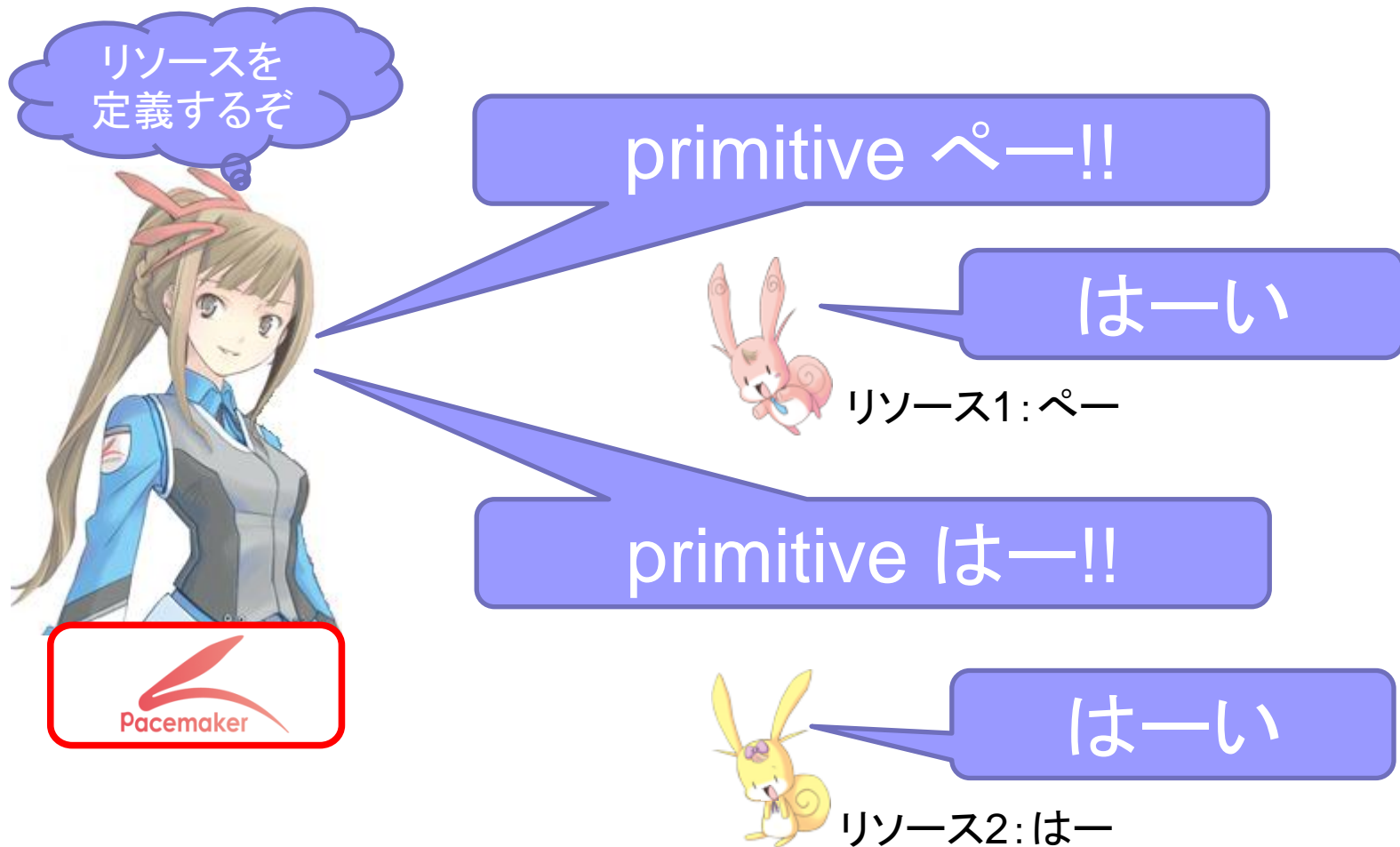
書式： group グループID リソースID...

グループID :任意の英数字で一意な名前を設定します。
リソースID :グループ化するリソースをIDで指定します。

詳細を以下記事で説明しています！

<http://linux-ha.sourceforge.jp/wp/archives/3855>

primitiveのイメージ



cloneのイメージ

ぺーちゃんが
2人必要！



clone ペー!!



はい

リソース1: ペー



Active



Standby

groupのイメージ

2人わズッ友..
一緒にいてほしいな



group ぺー はー!!



友情パワー

はーい



Active



Standby

改めて CRM設定を読んでみる

```
property no-quorum-policy="ignore" stonith-enabled="false" ¥  
crmd-transition-delay="2s"
```

```
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"
```

```
primitive resource1 ocf:heartbeat:Dummy ¥  
op start interval="0s" timeout="300s" on-fail="restart" ¥  
op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource2 ocf:heartbeat:Dummy ¥  
op start interval="0s" timeout="300s" on-fail="restart" ¥  
op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource3 ocf:heartbeat:Dummy ¥  
op start interval="0s" timeout="300s" on-fail="restart" ¥  
op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
op stop interval="0s" timeout="300s" on-fail="block"
```

```
clone clnResource resource1
```

```
group grp resource2 resource3
```

```
location loc1 grp ¥  
rule 200: #uname eq pm01 ¥  
rule 100: #uname eq pm02
```

```
colocation col1 INFINITY: grp clnResource
```

```
order order1 0: clnResource grp symmetrical=false
```

おまじない。

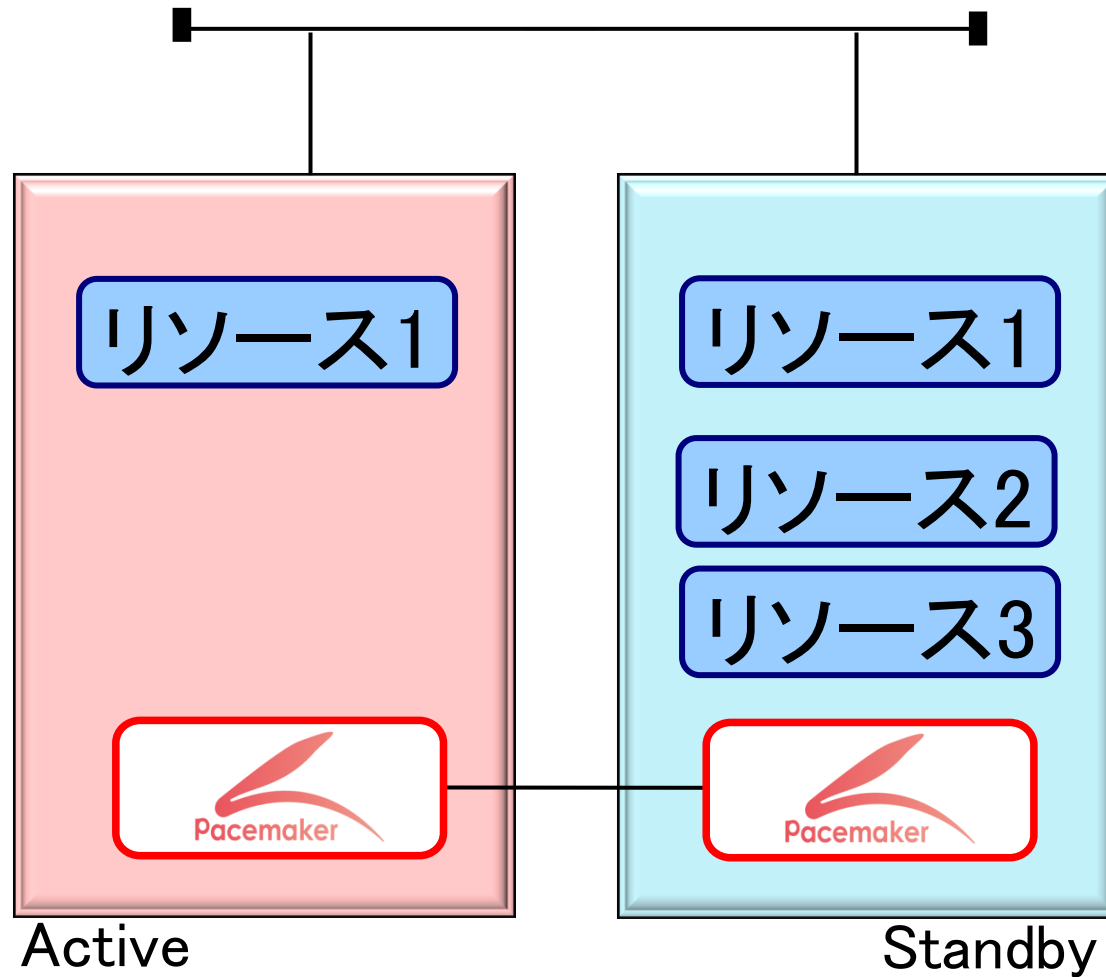
リソースを3つ使用することを定義。

どこで起動するかは、ここでは不定。

resource1を全ノードに複製(クローン)

resource2,3をグループ化し
同じノードでの起動を指定

ここまでの、ある程度
制御できているが..



惜しい！！

確かに、1と2の間の順序は
clone,groupでは指定していませんでした。

2,3の起動場所も不定です。

そこで必要になるのが「**制約**」！！


```
property no-quorum-policy="ignore" stonith-enabled="false" ¥  
crmd-transition-delay="2s"
```

```
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"
```

```
primitive resource1 ocf:heartbeat:Dummy ¥  
op start interval="0s" timeout="300s" on-fail="restart" ¥  
op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource2 ocf:heartbeat:Dummy ¥  
op start interval="0s" timeout="300s" on-fail="restart" ¥  
op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource3 ocf:heartbeat:Dummy ¥  
op start interval="0s" timeout="300s" on-fail="restart" ¥  
op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
op stop interval="0s" timeout="300s" on-fail="block"
```

```
clone clnResource resource1
```

```
group grp resource2 resource3
```

```
location loc1 grp ¥  
rule 200: #uname eq pm01 ¥  
rule 100: #uname eq pm02
```

```
colocation col1 INFINITY: grp clnResource
```

```
order order1 0: clnResource grp symmetrical=false
```

この3つです！

locationコマンド とは・・

概要：**任意のルール**とそのときのスコア値を設定することで、リソース配置を制約します。

(代表的な)書式:

```
location ID リソースID ¥  
rule スコア: ルール式
```

ID	:任意の英数字で一意的な名前を設定します。
リソースID	:制約を付与するリソースをIDで指定します。
スコア	:ルールを満たした場合のスコア値
ルール式	:ノード名、属性値等を「eq」「lt」「gt」等の比較演算子で比較し真の場合にスコアを適用

スコア とは・・

あるリソースを、どちらのノードで起動させるかを決めるための、Pacemaker内部の評価値※1。
値がより大きい方でリソースは起動する。

範囲: $-\text{INFINITY} < \text{負の整数} < 0 < \text{正の整数} < \text{INFINITY}$
(マイナス無限大) (無限大)

新リソース
追加



どちらで起動
させよう..?

ノード1はリソースが
もういるな。
→スコア減(-100)※2

よし
スコアを計算だ。

ノード2は故障
しているな。
→スコア減(-1000000)※2

ノード1



ノード2



※1 ptest -lsコマンドで実際のスコア値を確認できます。
※2 スコアの具体的な値は例のため正確ではありません。

colocationコマンド とは・・

概要：指定したリソースが**同一ノードで起動すること**に対し、スコアを設定します。

(代表的な)書式：

```
colocation ID スコア: リソースID1 リソースID2 ...
```

ID	:任意の英数字で一意な名前を設定します。
スコア	:設定するスコア値
リソースID	:制約を付与するリソースをIDで指定します。 1のリソースに、2のリソースが同一ノード上に必要となります。

orderコマンド とは・・

概要: リソースの**起動順序**に対しスコアを設定します。

(代表的な)書式:

```
order ID スコア: リソースID1 リソースID2 ...
```

ID	:任意の英数字で一意な名前を設定します。
スコア	:設定するスコア値
リソースID	:制約を付与するリソースをIDで指定します。 1のリソース起動後、2のリソースが起動。

locationのイメージ

location loc1 pe ¥
rule 200: #uname eq node01 ¥
rule 100: #uname eq node02

- ✓ 「#uname」でノード名を評価できる。
- ✓ 「eq」「lt」「gt」等の比較演算子がある。

pe追加



スコアの
大きい方へ..

ノード1

ノード2

location loc2 pe ¥
rule -INFINITY: attribute lt 100

- ✓ スコア「-INFINITY」は**禁止**を意味する。
- ✓ 属性値を評価できる

pe追加



禁止されて
ない方へ..

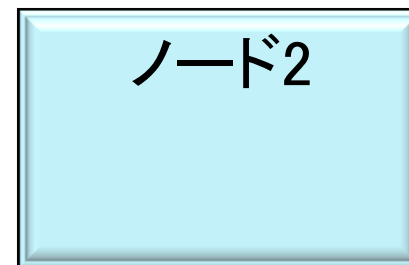
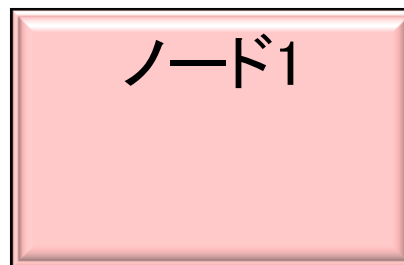
ノード1
attribute = 0

ノード2
attribute = 100

colocation,orderのイメージ

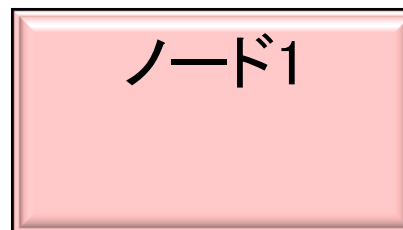
colocation col1 INFINITY: pe ha

- ✓ peの起動にhaが必要となる
- ✓ スコア「INFINITY」は**強制**を意味する。



order order1 0: pe ha symmetrical=false

- ✓ pe→haの順で起動
- ✓ スコア「0」は**Pacemakerへのアドバイス**
- ✓ symmetrical=trueだと停止順は起動の逆順に(falseは無効)



```
property no-quorum-policy="ignore" stonith-enabled="false" ¥  
crmd-transition-delay="2s"
```

```
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"
```

```
primitive resource1 ocf:heartbeat:Dummy ¥  
op start interval="0s" timeout="300s" on-fail="restart" ¥  
op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource2 ocf:heartbeat:Dummy ¥  
op start interval="0s" timeout="300s" on-fail="restart" ¥  
op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
op stop interval="0s" timeout="300s" on-fail="block"
```

```
primitive resource3 ocf:heartbeat:Dummy ¥  
op start interval="0s" timeout="300s" on-fail="restart" ¥  
op monitor interval="10s" timeout="60s" on-fail="restart" ¥  
op stop interval="0s" timeout="300s" on-fail="block"
```

```
clone clnResource resource1
```

```
group grp resource2 resource3
```

```
location loc1 grp ¥  
rule 200: #uname eq pm01 ¥  
rule 100: #uname eq pm02
```

```
colocation col1 INFINITY: grp clnResource
```

```
order order1 0: clnResource grp symmetrical=false
```

ノード名pm01の方をpm02
より優先する(200>100)

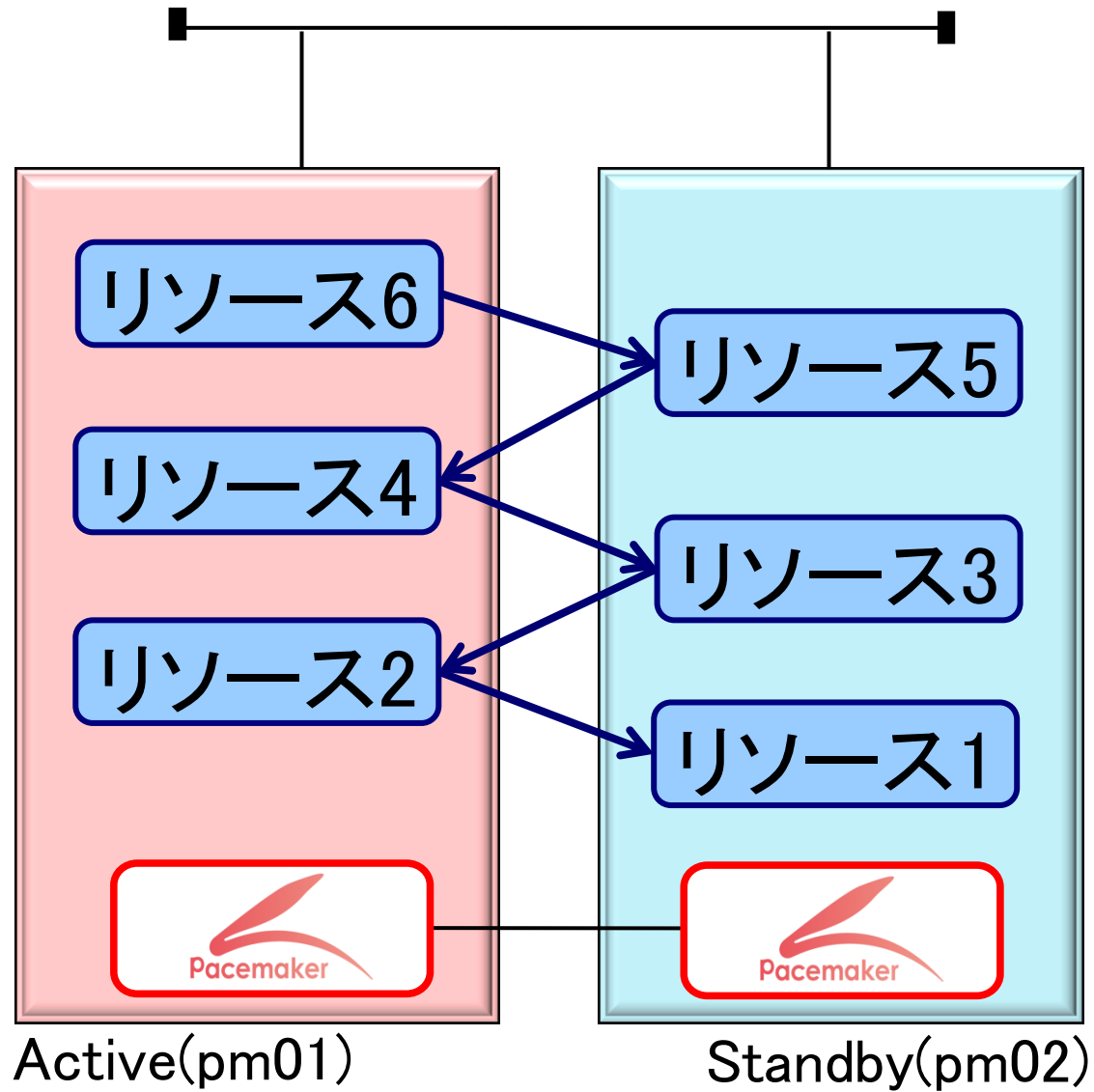
grpとclnResourceは必ず
同居させる。

clnResource→grpの順で
起動する。(停止順は不定)

④

制約を駆使してみる(デモ)

突然ですが、
こんなクラスタを組みたいとします。
(パート2)



名付けて「雷(いかづち)」！

実際の現場でこんな構成はないかもしれませんが・・・

どんなCRM設定になるでしょうか？
順に考えていきましょう。



まずはリソースが6つということで、以下の設定が必要そうです。

```
property no-quorum-policy="ignore" stonith-enabled="false" crmd-transition-delay="2s"
```

```
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"
```

「おまじない」

```
primitive resource1 ocf:heartbeat:Dummy ¥
```

```
op start interval="0s" timeout="300s" on-fail="restart" ¥
```

```
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
```

```
op stop interval="0s" timeout="300s" on-fail="block"
```

⋮

リソース1に対応する
リソース定義

同じように6回繰返し

```
primitive resource6 ocf:heartbeat:Dummy ¥
```

```
op start interval="0s" timeout="300s" on-fail="restart" ¥
```

```
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
```

```
op stop interval="0s" timeout="300s" on-fail="block"
```

リソース6に対応する
リソース定義

これだけでは、順番も場所も
決まっていないので、
6つのリソースがばらばらの順に、
任意のノードで起動します。※

そこで、「制約」！！

※実際には、2ノードに分散して起動すると思います。
しかし厳密には「不定」です。

まずは、こんな制約が必要そう・・・

└──→ resource1,3,5の同居を強制

```
colocation col1 INFINITY: resource1 resource3 resource5  
colocation col2 INFINITY: resource2 resource4 resource6
```

└──→ resource2,4,5の同居を強制

これで、resource1,3,5と2,4,6は
それぞれ同じノードで起動するはず。

やってみます。

ちゃんとpm01とpm02にきれいに分かれてくれました。

```
# crm_mon -rfA
```

～抜粋～

同居

同居

resource1	(ocf::heartbeat:Dummy): Started pm01
resource2	(ocf::heartbeat:Dummy): Started pm02
resource3	(ocf::heartbeat:Dummy): Started pm01
resource4	(ocf::heartbeat:Dummy): Started pm02
resource5	(ocf::heartbeat:Dummy): Started pm01
resource6	(ocf::heartbeat:Dummy): Started pm02

でも起動の順番がばらばら・・・

```
[root@pm01 ~]# grep -e "lrmd.*operation start" /var/log/ha-log  
pm01 ～略～ operation start[9] on resource3 ～略  
pm01 ～略～ operation start[8] on resource1 ～略  
pm01 ～略～ operation start[10] on resource5 ～略
```

```
[root@pm02 ~]# grep -e "lrmd.*operation start" /var/log/ha-log  
pm02 ～略～ operation start[10] on resource6 ～略  
pm02 ～略～ operation start[8] on resource2 ～略  
pm02 ～略～ operation start[9] on resource4 ～略
```

というわけで順番を制約。

order order1 INFINITY: resource6 resource5 ~ resource2 resource1

これで、
6→5→4→3→2→1
の順で起動するはず。

やってみます。

6から順番に起動しました。

(両ノードのログを時系列に並べています)

17:15:55 pm02 ～略～ operation start[8] on **resource6** ～略

17:16:04 pm01 ～略～ operation start[8] on **resource5** ～略

17:16:13 pm02 ～略～ operation start[10] on **resource4** ～略

17:16:20 pm01 ～略～ operation start[10] on **resource3** ～略

17:16:26 pm02 ～略～ operation start[12] on **resource2** ～略

17:16:34 pm01 ～略～ operation start[12] on **resource1** ～略

でもresource6がpm02で起動しています・・

というわけで起動場所を制約。

```
location loc1 resource6 ¥
```

```
rule 200: #uname eq pm01 ¥
```

```
rule 100: #uname eq pm02
```

```
location loc2 resource5 ¥
```

```
rule 200: #uname eq pm02 ¥
```

```
rule 100: #uname eq pm01
```

これで
resource6はホストpm01が、
resource5はホストpm02が、
優先されるはず。



6がpm01で、順番通りに起動しました。

(両ノードのログを時系列に並べています)

17:44:35 pm01 ～略～ operation start[8] on **resource6** ～略

17:44:43 pm02 ～略～ operation start[8] on **resource5** ～略

17:44:52 pm01 ～略～ info: operation start[10] on **resource4** ～略

17:45:02 pm02 ～略～ info: operation start[10] on **resource3** ～略

17:45:12 pm01 ～略～ info: operation start[12] on **resource2** ～略

17:45:23 pm02 ～略～ info: operation start[12] on **resource1** ～略

「雷」の要件を満たすことができました。



デモ2

スコア0とINFINITYの違いを確認

property no-quorum-policy="ignore" stonith-enabled="false" ¥
crmd-transition-delay="2s"

rsc_defaults resource-stickiness="INFINITY" ¥
migration-threshold="1"

primitive resource1 ocf:heartbeat:Dummy ¥
op start interval="0s" timeout="300s" on-fail="restart" ¥
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
op stop interval="0s" timeout="300s" on-fail="block"

primitive resource2 ocf:heartbeat:Dummy ¥
op start interval="0s" timeout="300s" on-fail="restart" ¥
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
op stop interval="0s" timeout="300s" on-fail="block"

location loc1 resource1 ¥
rule 200: #uname eq pm01 ¥
rule 100: #uname eq pm02

colocation col1 **0**: ¥
resource1 resource2

property no-quorum-policy="ignore" stonith-enabled="false" ¥
crmd-transition-delay="2s"

rsc_defaults resource-stickiness="INFINITY" ¥
migration-threshold="1"

primitive resource1 ocf:heartbeat:Dummy ¥
op start interval="0s" timeout="300s" on-fail="restart" ¥
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
op stop interval="0s" timeout="300s" on-fail="block"

primitive resource2 ocf:heartbeat:Dummy ¥
op start interval="0s" timeout="300s" on-fail="restart" ¥
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
op stop interval="0s" timeout="300s" on-fail="block"

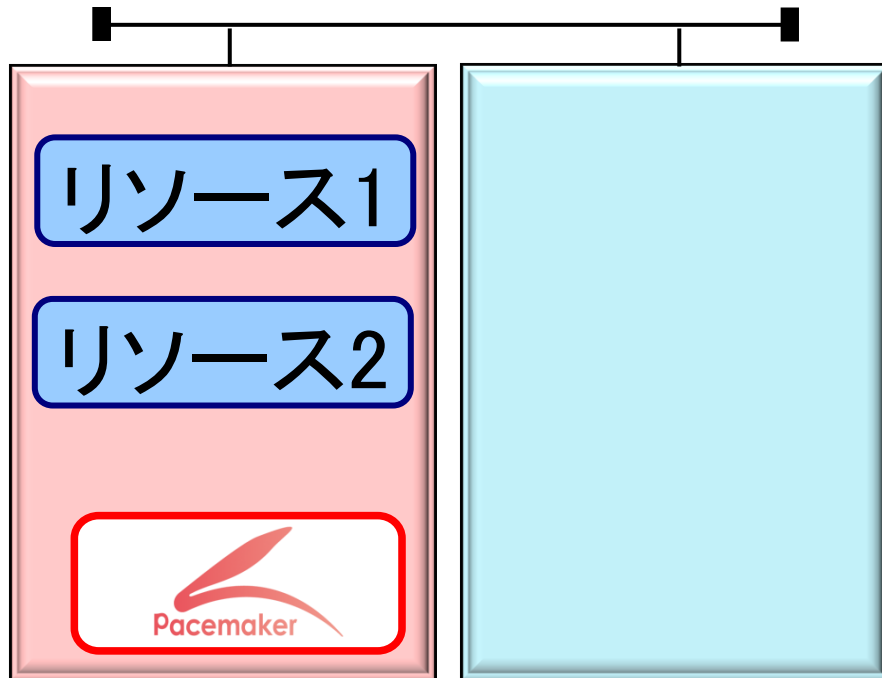
location loc1 resource1 ¥
rule 200: #uname eq pm01 ¥
rule 100: #uname eq pm02

colocation col1 **INFINITY**: ¥
resource1 resource2

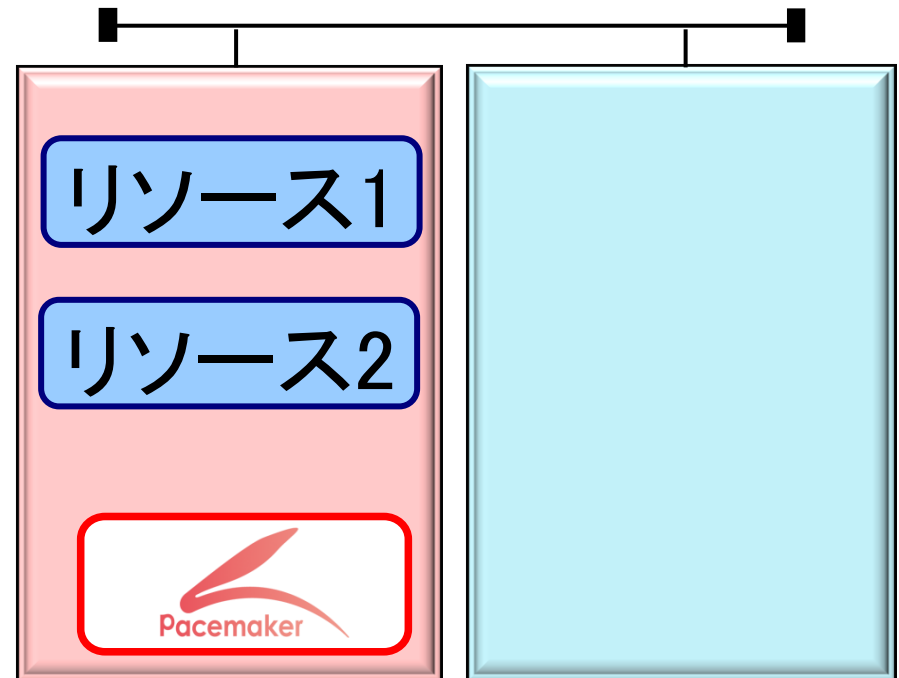
↑ どちらも「1の起動には2が必要」と制約 ↓
どう違うか？



「0」の場合



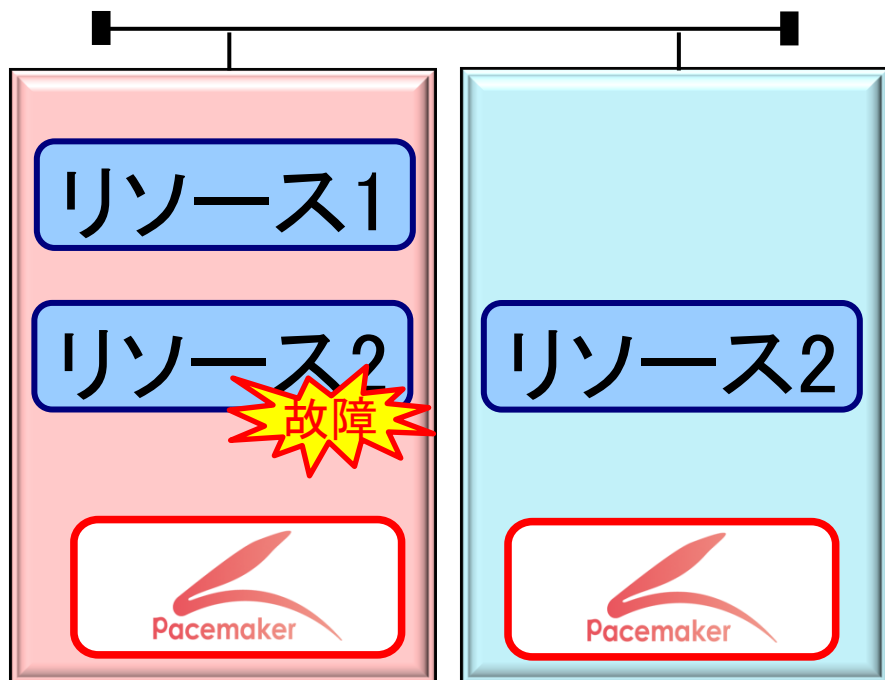
「INFINITY」の場合



同じに見えます

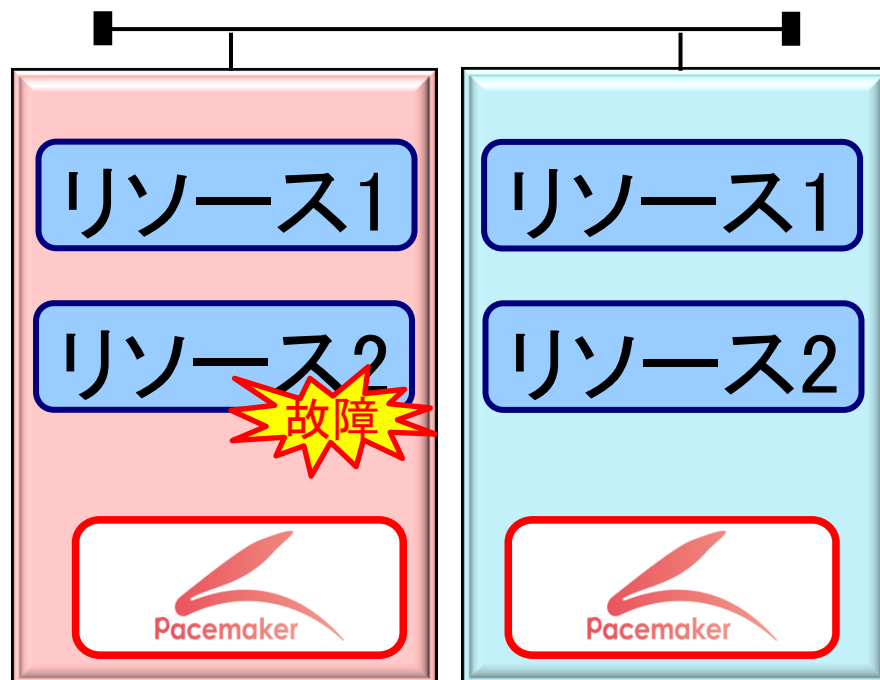
resource2が故障した場合はどうなるでしょうか？

「0」の場合



リソース2のみが
Standby側へF/Oします。

「INFINITY」の場合



リソース1,2ともに
Standby側へF/Oします。

「0」は「なるべく」、「INFINITY」は「絶対に」のイメージ



⑤

Linux-HA Japanについて

Linux-HA Japan URL

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

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



The screenshot shows the Linux-HA Japan Project website. At the top is the logo with the text "LINUX-HA JAPAN High-Availability Clustering on Linux". Below the logo is a navigation bar with links: HOME, メーリングリスト, ダウンロード&インストール, マニュアル, デスクトップテーマ・壁紙等, コミュニティ概要, その他, ニュース, イベント情報, 読み物, WEBラジオ. The main content area is titled "Linux-HA Japan プロジェクト" and includes social media links for Facebook, Twitter, and YouTube. Below this is a paragraph in Japanese describing the project as an open-source high-availability clustering solution for Linux. A table lists various resources: Linux-HA Japan 成果物ダウンロード, マニュアル, メーリングリスト, イベント情報, and 開発者向けサイト. At the bottom, there is a Twitter link and a note about the site's maintenance.

LINUX-HA JAPAN
High-Availability Clustering on Linux

HOME メーリングリスト ダウンロード&インストール マニュアル デスクトップテーマ・壁紙等 コミュニティ概要

その他 ニュース イベント情報 読み物 WEBラジオ

Linux-HA Japan プロジェクト

Linux上で高可用(HA)クラスタ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、クラスタリソースマネージャ、ク ラスタ通信レイヤ、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等を、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。

今は主に Pacemaker , Heartbeat , Corosync , DRBD等を扱っています。

Linux-HA Japan 成果物ダウンロード	RHEL/CentOS向けPacemaker RPM/パッケージ(yumのリポジトリ形式)や設定ファイル(crm)作成支援ツール、ディスク監視機能などをダウンロードできます。とりあえずRHELもしくはCentOS等のRHEL互換OSにインストールしてみたい場合はこちら。インストール後にとりあえず何か動かしてみたい場合はこちらを参考にしてみてください。
マニュアル	本家コミュニティ提供の公式マニュアルやLinux-HA Japan提供の翻訳マニュアル。マニュアル読んでもよくわからない場合は、過去のカンファレンスや勉強会等の発表資料も参考に。
メーリングリスト	インストール方法や設定方法等の質問はMLまで。 ※投稿するにはメールアドレスの登録が必要です。
イベント情報	カンファレンスへの出席や講演、勉強会開催情報、講演時のスライド公開など。
開発者向けサイト	Linux-HA Japan開発者向けサイトです。Linux-HA Japan独自開発機能のソースコードやバイナリのダウンロード等。

Twitter 公式ハッシュタグ [#linux_ha_jp](#)

本サイトに関するお問い合わせは、Linux-HA Japan メーリングリスト管理者
([linux-ha-japan-owner](#) アット [lists.sourceforge.jp](#)) までお願いいたします。

Pacemaker関連の最新情報を 日本語で発信

Pacemakerのダウンロードも こちらからどうぞ。 (インストールが楽なリポジトリパッケージ を公開しています。)

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



参考文献

- CRM-CLIの公式マニュアル(日本語)
 - http://linux-ha.sourceforge.jp/wp/wp-content/uploads/crm_cli.html
- 動かして理解するPacemaker ～CRM設定編～ その1
 - <http://linux-ha.sourceforge.jp/wp/archives/3786>
 - propertyとrsc_defaultsを解説
- 動かして理解するPacemaker ～CRM設定編～ その2
 - <http://linux-ha.sourceforge.jp/wp/archives/3855>
 - primitive, clone, groupを解説
- 動かして理解するPacemaker ～CRM設定編～ その3(予定)
 - <http://linux-ha.sourceforge.jp/wp/archives/????>
 - 制約を解説予定(11月中??)

ご清聴ありがとうございました。



Linux-HA Japan

検索

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



参考 Dummyとは

- ✓ DummyはれっきとしたPacemaker付属のRA
- ✓ 名前が示唆している通り、あくまでダミー
 - ✓ start : ある一時ファイルを作成する
 - ✓ monitor : 一時ファイルの存在確認
 - ✓ stop : 一時ファイルの削除
- ✓ 本講演のようなPacemakerの動作を確認したい場合のために用意されている。

Apache等のリソースを設定することなく手軽に使用できるので、とても便利です。

