試して覚えるPacemaker入門
『PG-REX運用』
(Pacemaker+PostgreSQLによるシェアードナッシング構成運用)

OSC2018 Osaka
2018/1/27
Linux-HA Japan
竹田　健二

はじめに：本日の話の流れ

- 「Pacemaker」とは？
- 「PG-REX」とは
- クラスタの状態確認
  - Pacemakerの監視コマンドやログなど、クラスタの状態を確認するために役立つ情報をご紹介します
- 故障パターンと復旧方法
  - 各種故障を擬似的に発生させる方法や、故障の特定方法、復旧方法などについてお話します
- コミュニティ紹介

まずはおさらい
「Pacemaker」とは？

Pacemaker(ぺーすめーかー)とはオープンソースで開発されているHAクラスタソフト

- Heartbeatの後継として開発されたソフトウェアであり、ソースはGitHubで管理
  - ClusterLabs：https://github.com/ClusterLabs
- バイナリファイルは以下より入手可能
  - Linux-HA Japanプロジェクト：http://linux-ha.osdn.jp/wp/

High Availability = 高可用性

サービス継続性

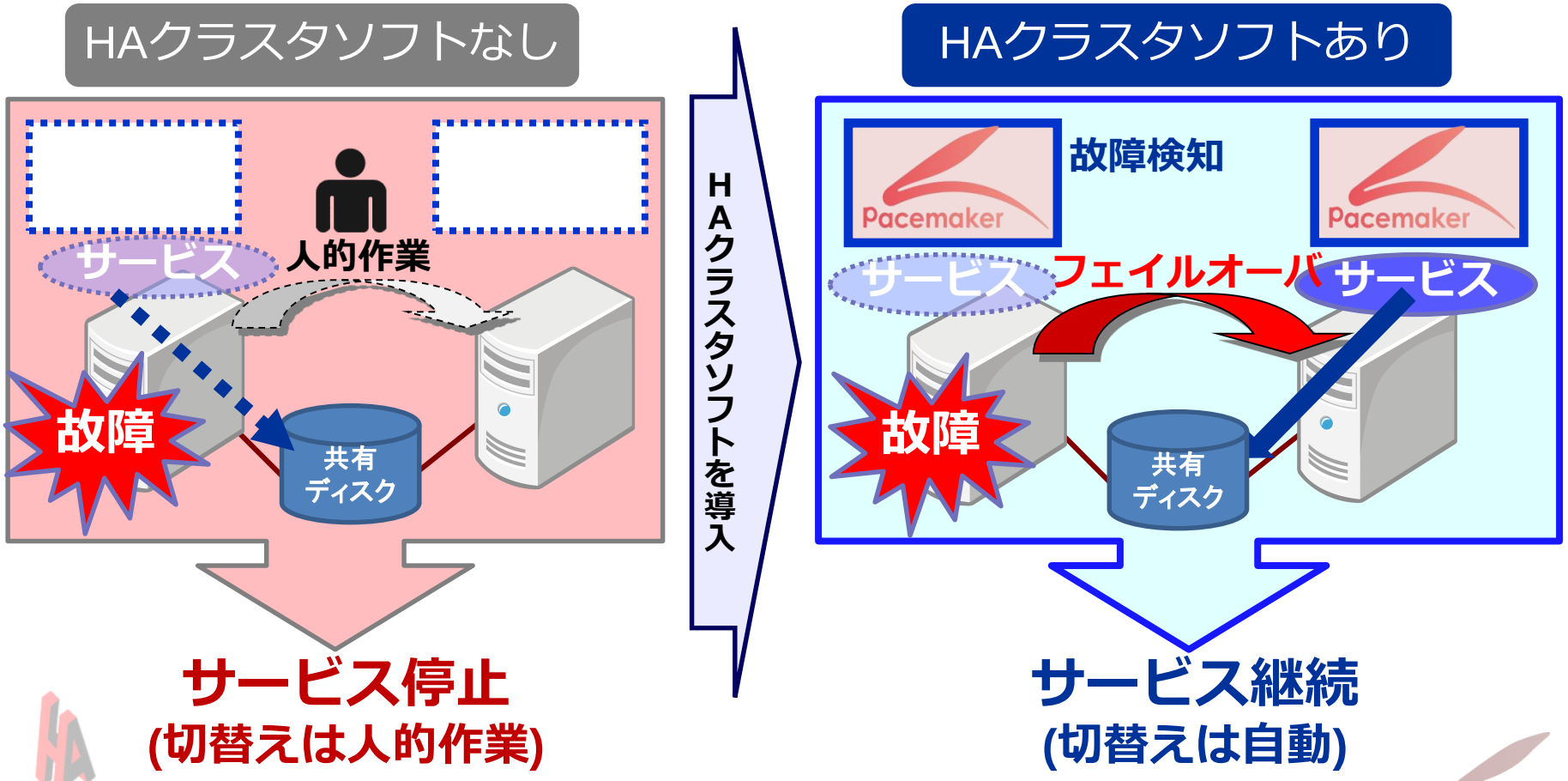
つまり

一台のコンピュータでは得られない高い
信頼性を得るため、複数のコンピュータを
結合してひとまとまりとする(クラスタ化)

ためのソフトウェア

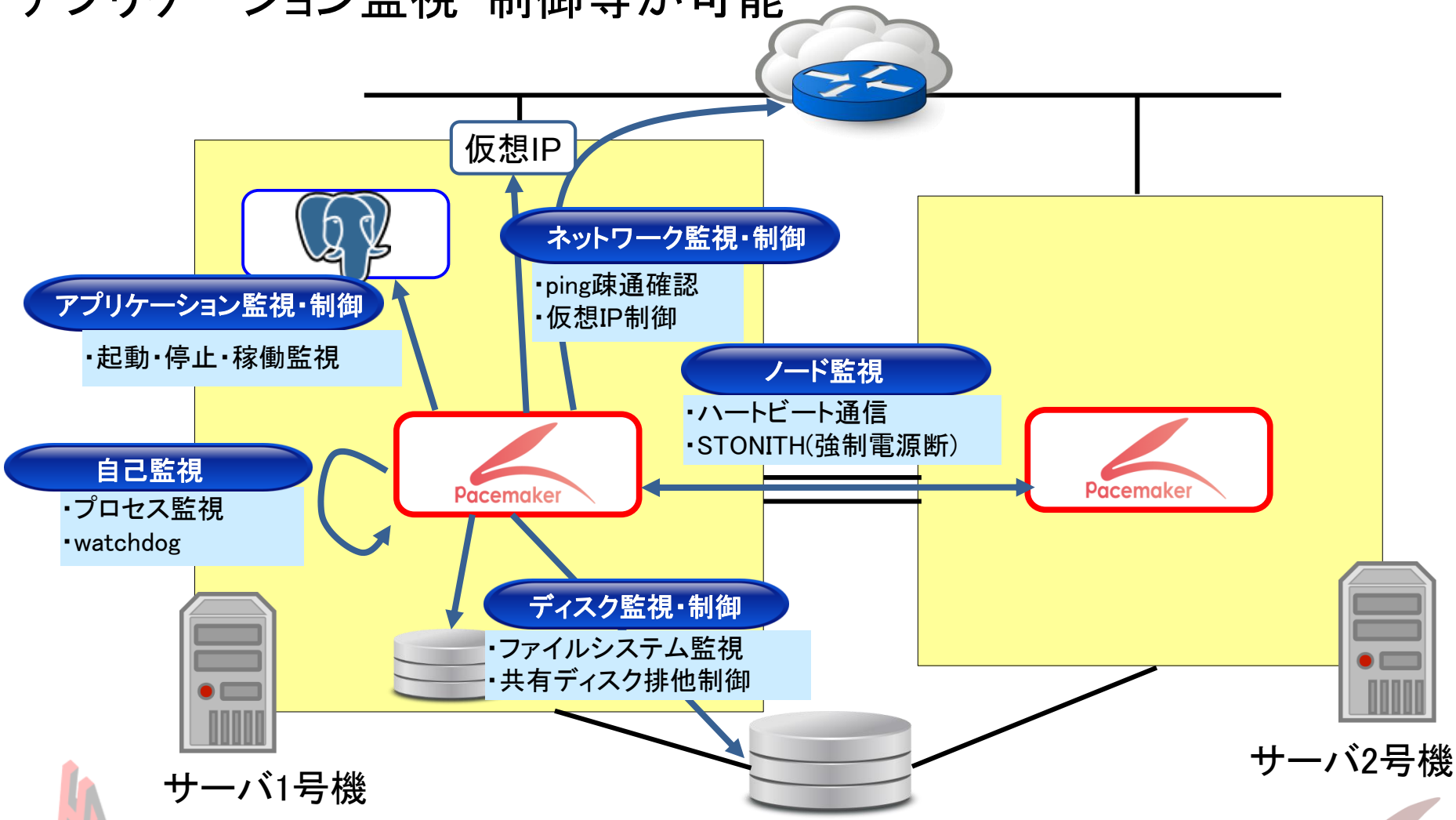
HAクラスタソフトは何故必要か？

HAクラスタソフトを導入することで、システムを監視し、故障が発生した
時にはそれを検知してサービスを自動で切り替えて継続することが可能になる
この仕組みは「**フェイルオーバ (F.O)**」と呼ばれる



Pacemakerで監視/制御ができるもの

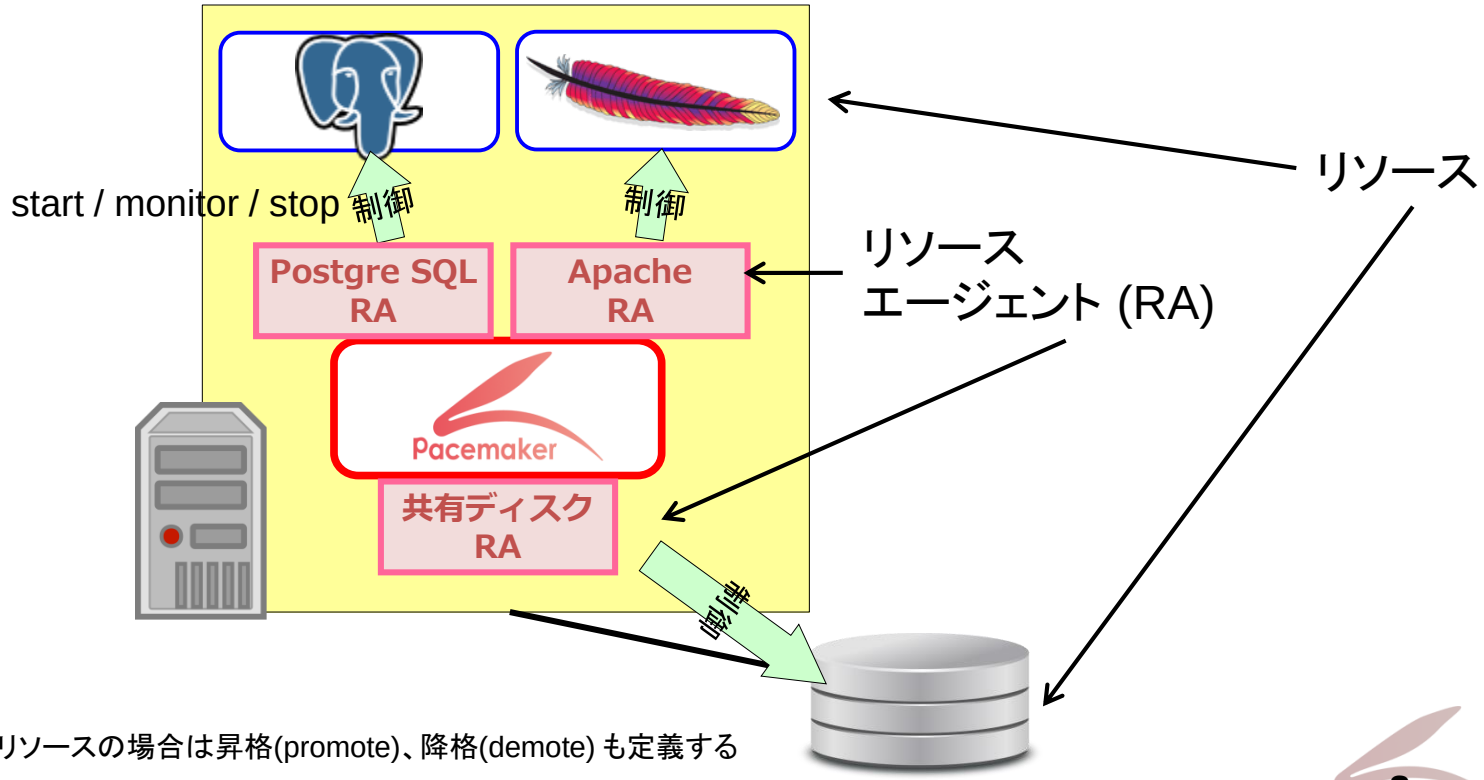
ノード監視、ネットワーク監視・制御、ディスク監視・制御、
アプリケーション監視・制御等が可能



リソースとリソースエージェント

✓ 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で制御可能としている
- リソースの 起動(start)、監視(monitor)、停止(stop) を行うメソッドを定義する※



※ Master/Slaveリソースの場合は昇格(promote)、降格(demote) も定義する

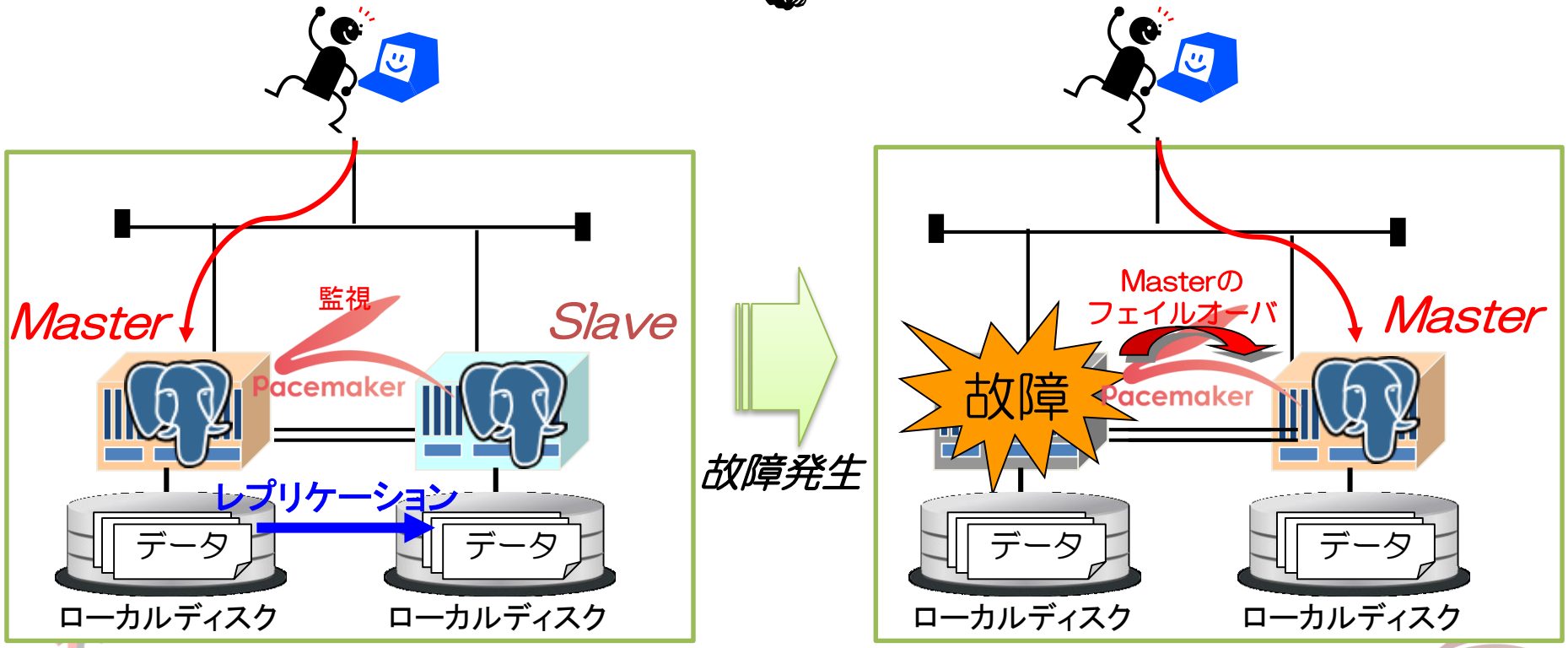
「PG-REX」とは

PG-REXとは

PostgreSQLレプリケーション機能＋Pacemakerにおける
シェアードナッシング構成モデルを「PG-REX（ぴーじーれっくす）」と呼ぶ

PostgreSQLのストリーミングレプリケーション機能を用いてデータを常に両系にコピー

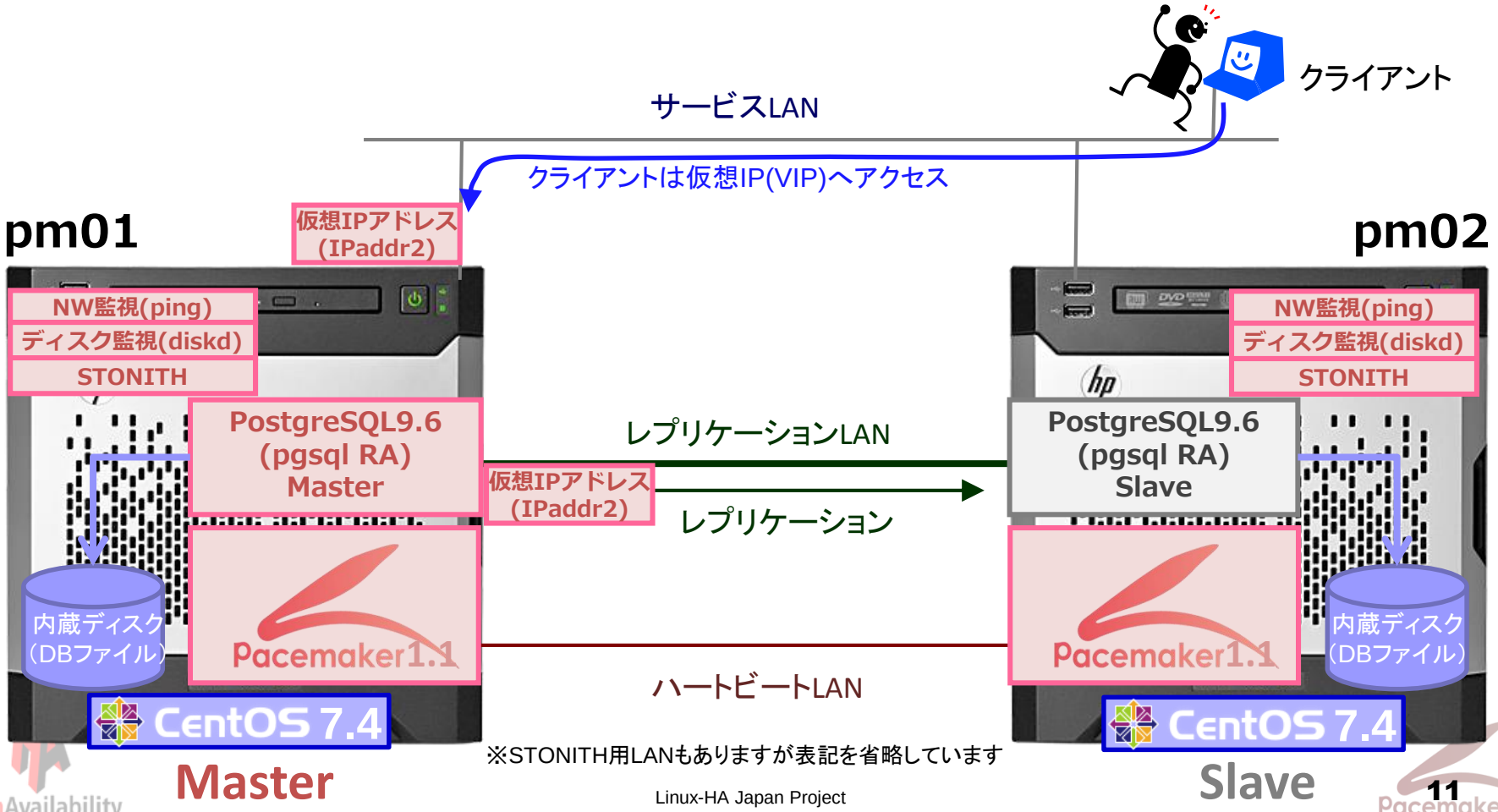
故障をPacemakerが監視・検知
SlaveをMasterに昇格させることで自動的にサービスを継続



本資料で使用する構成

本資料では以下の構成を例に運用方法について説明する

- ✓ PG-REXプロジェクトで配布されている環境定義書の構成とほぼ同等
  (差分は「vip-slave」を使用していない点。crm設定は本資料と併せて公開します)
- ✓ CentOS7.4／Pacemaker1.1.16-1.1／PostgreSQL9.6 を使用



構築手順概要

「PG-REX利用マニュアル」に従い、以下3ステップで構築

- 今回は運用がメインのため、概要レベルで・・・

構築手順

1. インストール
   - 両系にPacemaker1.1をインストール
   - 両系にPostgreSQL9.6、PG-REX運用補助ツールをインストール

2. Master構築・起動
   - Master側DB初期化
   - Master側Pacemakerを起動しリソースを設定（crm設定流し込み）
     - 「pg-rex_master_start」コマンドを使えば楽ちん！

3. Slave構築・起動
   - Slave側にMasterのPostgreSQL最新DBをコピー
   - Slave側Pacemakerを起動
     - 「pg-rex_slave_start」コマンドを使えば楽ちん！

【補足】設定・構築についての関連情報

✓マニュアル等の資料について

PG-REX利用マニュアルは「PG-REXプロジェクト」のページ(①)で公開されています
設定・構築や運用について詳しく書かれている資料です
また、PG-REXの構築について分かりやすく説明した資料(②)もありますので、
そちらも是非ご覧ください

①PG-REXプロジェクト
https://ja.osdn.net/projects/pg-rex/

②資料『試して覚えるPacemaker入門 PG-REX構築』
https://linux-ha.osdn.jp/wp/archives/4627

✓「PG-REX運用補助ツール」や「pg-rex_~~」コマンドとは

PG-REX運用補助ツールは複雑なPG-REX構成の運用を楽ちんにしてくれるツールです
「pg-rex_～～」コマンドはこの中に含まれています
ツールはPG-REXプロジェクトのページで「pg-rex_operation_tools」として公開しています
使い方はPG-REX利用マニュアルや②の資料をご覧ください

クラスタの状態確認

早速、故障パターンとそれに応じた対処方法を紹介します

・・・といいたいところですが、そもそもクラスタの状態を確認する方法がわからないと正常or故障の判断ができません！

なので・・・

クラスタの状態確認方法を少し詳しくご紹介

状態確認の上で重要な以下のトピックについて説明

- 監視コマンド(crm_monコマンド)
- ログ＆ログメッセージ制御機能

監視コマンド

- ✓ 監視コマンド(crm_monコマンド)

  Pacemakerに付属の、クラスタ状態をCUIベースの画面(以下、「監視画面」と呼ぶ)で
  リアルタイムに確認できるコマンド

- ✓ 具体的には以下の事柄が確認可能

✓ quorumを持っているか？ ✓ DCノードは誰か？ ✓ Pacemakerのバージョンは？	基本情報
✓ ノードがクラスタに参加している(online)かどうか？	ノード情報
✓ リソースがどのノードで起動(停止)しているか？	リソース情報
✓ ネットワーク監視は正常か？ ✓ ディスク監視は正常か？	属性値
✓ なにか故障は発生していないか？	故障情報

監視コマンド

✓監視画面

「crm_mon -rfA」で今回の構成を表示(正常時)した例

```
Stack: corosync
Current DC: pm01 (version 1.1.16-1.el7-94ff4df) - partition with quorum
Last updated: Fri Jan  5 10:50:11 2018
Last change: Fri Jan  5 10:50:01 2018 by root via crm_attribute on pm01

2 nodes configured
14 resources configured

Online: [ pm01 pm02 ]

Full list of resources:

 Resource Group: master-group
     vip-master     (ocf::heartbeat:IPaddr2):       Started pm01
     vip-rep        (ocf::heartbeat:IPaddr2):       Started pm01
 Resource Group: grpStonith1
     prmStonith1-1                  (stonith:external/stonith-helper):  Started pm02
     prmStonith1-2                  (stonith:external/ipmi):            Started pm02
 Resource Group: grpStonith2
     prmStonith2-1                  (stonith:external/stonith-helper):  Started pm01
     prmStonith2-2                  (stonith:external/ipmi):            Started pm01
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
 Clone Set: clnPing [prmPing]
     Started: [ pm01 pm02 ]
 Clone Set: clnDiskd1 [prmDiskd1]
     Started: [ pm01 pm02 ]
 Clone Set: clnDiskd2 [prmDiskd2]
     Started: [ pm01 pm02 ]
```

基本情報

pm01とpm02がオンライン

ノード情報

リソース情報

master-groupはpm01で起動
pgsqlはpm01がMaster
etc...

~右上に続く

~続き

```
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status                : normal
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline           : 000000000B000140
    + pgsql-status                    : PRI
    + ringnumber_0                    : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status                : normal
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 100
    + pgsql-data-status               : STREAMING|SYNC
    + pgsql-status                    : HS:sync
    + ringnumber_0                    : 192.168.11.32 is UP
```

属性値

ネットワーク／ディスク監視は正常　etc...

```
Migration Summary:
* Node pm01:
* Node pm02:
```

故障情報

故障は発生していない

・確認できる事柄(再掲)

✓ quorumを持っているか？ ✓ DCノードは誰か？ ✓ Pacemakerのバージョンは？	基本情報
✓ ノードがクラスタに参加している(online)かどうか？	ノード情報
✓ リソースがどのノードで起動(停止)しているか？	リソース情報
✓ ネットワーク監視は正常か？ ✓ ディスク監視は正常か？	属性値
✓ なにか故障は発生していないか？	故障情報

監視コマンド

✓監視画面

vip-rep※故障時(正常時との差分箇所は色文字で記載)の例

基本情報

```
Stack: corosync
Current DC: pm01 (version 1.1.16-1.el7-94ff4df) - partition with quorum
Last updated: Fri Jan  5 10:37:31 2018
Last change: Fri Jan  5 10:35:45 2018 by root via crm_attribute on pm01

2 nodes configured
14 resources configured
```

ノード情報

```
Online: [ pm01 pm02 ]
```

リソース情報

```
Full list of resources:

 Resource Group: master-group
     vip-master    (ocf::heartbeat:IPaddr2):          Started pm01
     vip-rep       (ocf::heartbeat:IPaddr2):          Started pm01
 Resource Group: grpStonith1
     prmStonith1-1                 (stonith:external/stonith-helper):  Started pm02
     prmStonith1-2                 (stonith:external/ipmi):            Started pm02
 Resource Group: grpStonith2
     prmStonith2-1                 (stonith:external/stonith-helper):  Started pm01
     prmStonith2-2                 (stonith:external/ipmi):            Started pm01
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
 Clone Set: clnPing [prmPing]
     Started: [ pm01 pm02 ]
 Clone Set: clnDiskd1 [prmDiskd1]
     Started: [ pm01 pm02 ]
 Clone Set: clnDiskd2 [prmDiskd2]
     Started: [ pm01 pm02 ]
```

~右上に続く

~続き

属性値

```
Node Attributes:
* Node pm01:
    + default_ping_set                 : 100
    + diskcheck_status                 : normal
    + diskcheck_status_internal        : normal
    + master-pgsql                     : 1000
    + pgsql-data-status                : LATEST
    + pgsql-master-baseline            : 0000000009003D80
    + pgsql-status                     : PRI
    + ringnumber_0                     : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                 : 100
    + diskcheck_status                 : normal
    + diskcheck_status_internal        : normal
    + master-pgsql                     : 100
    + pgsql-data-status                : STREAMING|SYNC
    + pgsql-status                     : HS:sync
    + ringnumber_0                     : 192.168.11.32 is UP
```

故障情報

```
Migration Summary:
* Node pm01:
   vip-rep: migration-threshold=0 fail-count=1 last-failure='Fri Jan  5
10:37:19 2018'
* Node pm02:

Failed Actions:
* vip-rep_monitor_10000 on pm01 'not running' (7): call=66,
status=complete, exitreason='none',
    last-rc-change='Fri Jan  5 10:37:19 2018', queued=0ms, exec=0ms
```

故障**あり**。vip-repに故障が発生した旨を表示

※vip-repはレプリケーション用途の仮想IP制御(IPaddr2)リソース

監視コマンド

✓監視画面

サービスLAN故障時(正常時との差分箇所は色文字で記載)の例

基本情報

```
Stack: corosync
Current DC: pm01 (version 1.1.16-1.el7-94ff4df) - partition with quorum
Last updated: Tue Jan  9 11:49:43 2018
Last change: Tue Jan  9 11:49:11 2018 by root via crm_attribute on pm02

2 nodes configured
12 resources configured
```

ノード情報

```
Online: [ pm01 pm02 ]
```

リソース情報

```
Full list of resources:

 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):       Started pm02
     vip-rep    (ocf::heartbeat:IPaddr2):       Started pm02
 Resource Group: grpStonith1
     prmStonith1-1      (stonith:external/stonith-helper):      Started pm02
     prmStonith1-2      (stonith:external/ipmi):                Started pm02
 Resource Group: grpStonith2
     prmStonith2-1      (stonith:external/stonith-helper):      Started pm01
     prmStonith2-2      (stonith:external/ipmi):                Started pm01
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm02 ]
     Stopped: [ pm01 ]
 Clone Set: clnPing [prmPing]
     Started: [ pm01 pm02 ]
 Clone Set: clnDiskd1 [prmDiskd1]
     Started: [ pm01 pm02 ]
```

~右上に続く

~続き

属性値

```
Node Attributes:
* Node pm01:
    + default_ping_set                : 0     : Connectivity is lost
    + diskcheck_status_internal       : normal
    + master-pgsql                    : -INFINITY
    + pgsql-data-status               : DISCONNECT
    + pgsql-status                    : STOP
    + ringnumber_0                    : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline           : 0000000016000440
    + pgsql-status                    : PRI
    + ringnumber_0                    : 192.168.11.32 is UP
```

故障情報

```
Migration Summary:
* Node pm01:
* Node pm02:
```

Q.故障情報には何も表示されていないけど・・・

A.故障かどうかの判断は故障情報だけでなく「属性値」など、他の情報表示も見て判断する必要があります！！

属性値とは

- 属性値(Attribute)とは、ノード毎の各リソースの状態を補完するPacemaker内部の値
  - Key-Valueの形式でRA毎に任意に定義可能
  - 現在の値はcrm_mon -Aコマンドで確認可能
  - 属性値はリアルタイムに更新される
- PG-REXの運用で特に重要なのは以下の2つ
  - 「pgsql-status」属性値
    - PostgreSQLの稼働状況を示す

値	意味
PRI	Masterとして稼働中
HS:sync	Slaveとして同期モードで稼働中
HS:async	Slaveとして非同期モードで稼働中
HS:alone	Slaveとして稼働しているが、Masterに接続できない

  - 「pgsql-data-status」属性値
    - PostgreSQLのデータの状態を示す

値	意味
LATEST	他にMasterはおらず、データは最新
STREAMING|SYNC	同期接続によりデータは最新
STREAMING|ASYNC	非同期接続によりデータは追いついていない可能性あり
DISCONNECT	Masterとの接続が切断し、データは古い

※上記属性値名の「pgsql」部分はPostgreSQLのリソース名(Pacemaker上でのPostgreSQL管理名でユーザが任意に設定可)により変化します
通常は同リソース名は「pgsql」とすることを推奨します

属性値とは

✓監視コマンドにおける属性値表示の確認

- 属性値がどのような意味をもつのかを運用者は把握しておく必要がある
- 属性値はあくまで状態を示すもの
  その値がクラスタ稼働状況的に問題がないかどうかは運用者が判断しないといけない

default_ping_set→NW監視
diskcheck_status→ディスク監視
etc..
運用者は属性値の意味を把握しておくこと

正常稼働時と比較して異常はないか？
値がどのような状態を示すのかを把握することも大事
「故障情報」には表示されていなくとも、「属性値」から故障を判断するケースもある

属性値

```
Node Attributes:
* Node pm01:
    + default_ping_set                  : 100
    + diskcheck_status                  : normal
    + diskcheck_status_internal         : normal
    + master-pgsql                      : 1000
    + pgsql-data-status                 : LATEST
    + pgsql-master-baseline             : 000000000B000140
    + pgsql-status                      : PRI
    + ringnumber_0                      : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                  : 100
    + diskcheck_status                  : normal
    + diskcheck_status_internal         : normal
    + master-pgsql                      : 100
    + pgsql-data-status                 : STREAMING|SYNC
    + pgsql-status                      : HS:sync
    + ringnumber_0                      : 192.168.11.32 is UP
```

ログ

✓ログ(ha-log)

- ha-logとは、Pacemakerが出力するログの名前
  - デフォルトでは/var/log/messagesに出力
    (syslog_facilityのデフォルトが「daemon」のため)

- クラスタの運用に役立つ情報がたくさん出力される
  - 監視コマンドのみでは確認できない故障状況を追うことが可能
  - Act側とSby側で出力内容が異なるので注意！
  - 故障解析には両系分のログがあるほうがよい

- その分、ログの量もかなりのもの
  - 解析時の手間を考慮すると、できればデフォルト設定を変更し
    /var/log/ha-log等に切り出した方がよい(設定については付録ページに掲載)
  - それでも、量が多く運用者視点では情報過多な面も・・・

なので・・・

『ログメッセージ制御機能(pm_logconv)』を
使うことをオススメします！

ログ

✓ログメッセージ制御機能(pm_logconv)

- 『pm_logconv』は、ha-logを変換して運用上必要なログだけを出力
- ha-logのフォーマットに変更があった場合も、pm_logconv のログ変換で吸収することで影響を受けにくい(監視ツール等の変更対応が不要)
- 開発はLinux-HA Japanが行っており、Linux-HA Japan制のPacemakerリポジトリパッケージをインストールすればデフォルトで使用可能

＜変換例：vip-master故障時のログ＞

- Pacemaker標準ログ(ha-log)

```
Jan  5 10:50:06 pm01 cib[10502]:    info: Reporting our current digest to pm01:
5557aa29e9ae50fcd0eb5396ce048388 for 0.14.2 (0x55fd9e99bfe0 0)
Jan  5 10:51:06 pm01 crmd[10507]:    info: Result of monitor operation for vip-master on
pm01: 7 (not running)
  :
Jan  5 10:51:07 pm01 crmd[10507]:  notice: Result of stop operation for vip-master on
pm01: 0 (ok)
```

322行

変換

運用上必要なログだけを出力

- ログメッセージ制御機能により変換されたログ(pm_logconv.out)

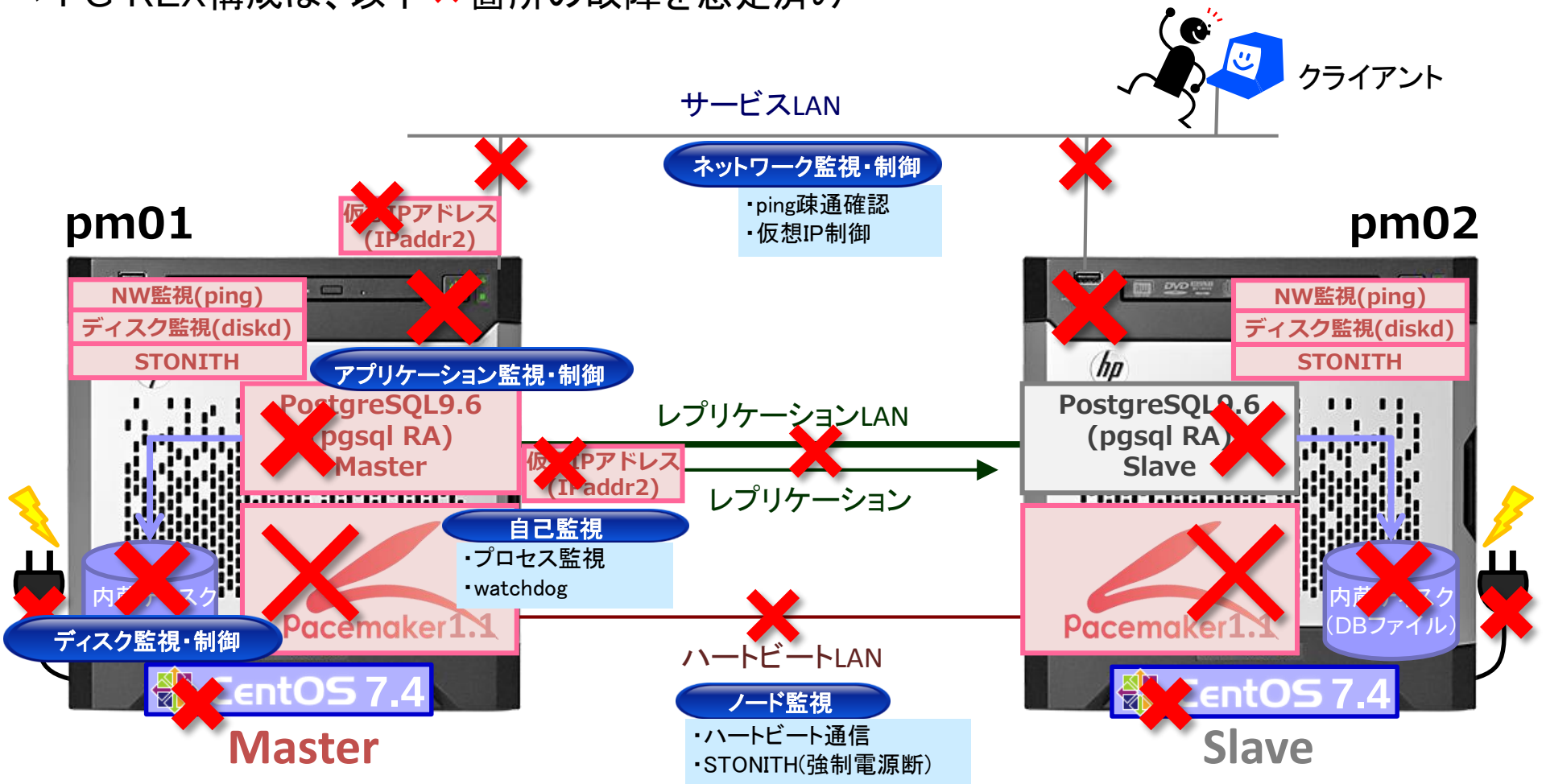
```
Jan 5 10:51:06 pm01    error: Resource vip-master does not work. (rc=7) not running
:
Jan 5 10:51:07 pm01    info: Resource vip-master stopped. (rc=0) ok
```

12行

故障パターンと復旧方法

Pacemakerによる監視/制御と故障パターン

✓Pacemakerでは様々な故障を検知して、サービスの継続性を高めることが可能
✓PG-REX構成は、以下 ✕ 箇所の故障を想定済み



※STONITH用LANもありますが表記を省略しています

故障毎のPacemakerの動作

故障パターン(前ページ×箇所)とPacemakerの動作を整理すると以下のようになる
（Master側のみ記載）

故障項目	故障内容	Pacemakerの動作	
リソース故障	vip-rep 故障	①	アプリケーション監視・制御
	vip-master 故障	②	
	PostgreSQL故障	②	
ネットワーク故障	レプリケーションLAN故障	①	ネットワーク監視・制御
	サービスLAN故障	②	
	ハートビートLAN故障	③	
ノード故障	カーネルハング	③	ノード監視
	電源停止	③	
Pacemakerプロセス故障	pacemakerdプロセス故障	①	自己監視
	corosyncプロセス故障	③	
	cibプロセス故障	③	
ディスク故障	内蔵ディスク故障	② or ③	ディスク監視・制御
リソース停止失敗	vip-rep	②	アプリケーション監視・制御
	vip-master	③	
	PostgreSQL(demote失敗)	③	
	PostgreSQL(stop失敗)	③	

①リソース／プロセス再起動
②通常F.O
③STONITH後F.O
詳しくは次ページで説明

故障時のPacemakerの動作(3パターン)

故障時のPacemakerの動作は、サービス影響や故障サーバ状態により、3つに分けられる

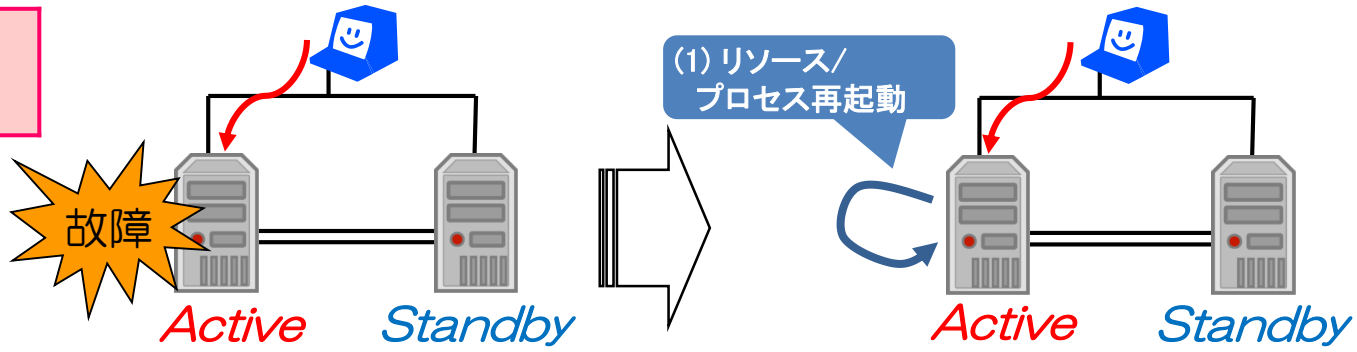
	Pacemakerの動作		
	① リソース/プロセス 再起動	② 通常 フェイルオーバ	③STONITH後 フェイルオーバ
動作概要	同じサーバ上でリソース/ プロセスをもう一度起動、または設定変更する ※フェイルオーバはしない	故障サーバの関連リソースを停止後、Standby(Slave)サーバでリソースを起動する	故障サーバの電源を強制的に断(STONITH)後、Standby(Slave)サーバでリソースを起動する
対処条件	**サービス継続に直接関係ない**リソース故障時の対処	**サービス継続に影響がある**故障時の対処	**故障サーバの状態が確認 できない**場合に二重起動を防ぐ対処
故障例	・レプリケーションLAN故障（共有ディスク無し構成）	・DBプロセス停止 ・サービスLAN故障 ・共有ディスクケーブル故障	・サーバ電源停止 ・Pacemakerプロセス故障 ・ハートビートLAN故障 ・リソース停止失敗

故障時のPacemakerの動作(3パターン)

＜動作イメージ＞

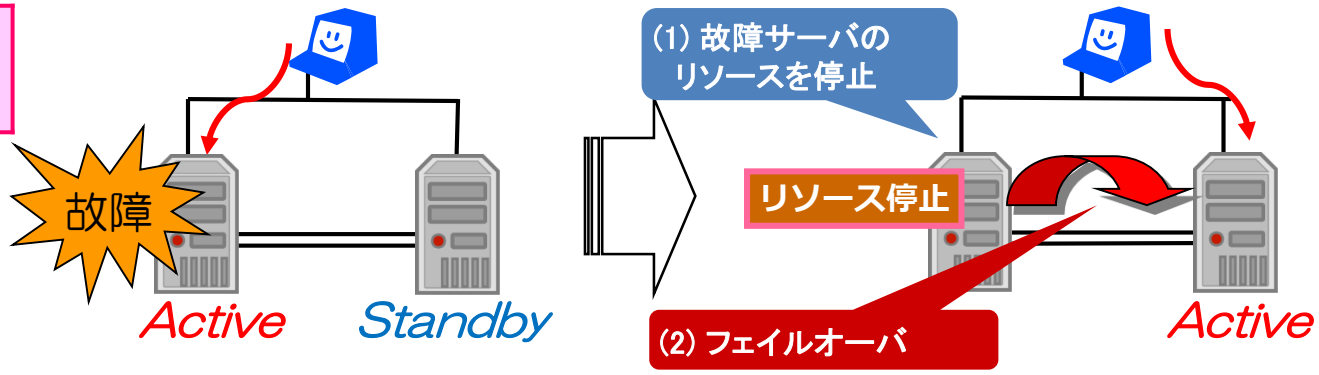
①リソース/プロセス再起動

※フェイルオーバはせずに、故障リソースのみ再起動する



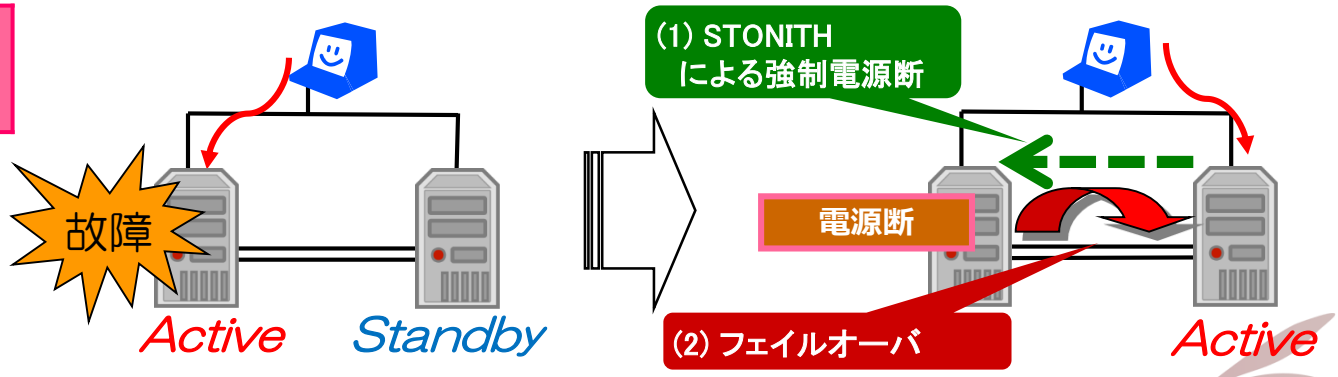
② 通常フェイルオーバ

※故障サーバのリソースを停止後に、フェイルオーバを行う（通常の切り替え動作）



③ STONITH後フェイルオーバ

※故障サーバのリソース停止不可や、故障サーバの状態確認不可の場合に、二重起動を防ぐため、強制電源断後にフェイルオーバを行う



故障毎のPacemakerの動作

故障パターン(前ページ×箇所)とPacemakerの動作を整理すると以下のようになる
（Master側のみ記載）

本編ではこれらについて紹介
(他のパターンは付録に掲載)

故障項目	故障内容	Pacem
リソース故障	vip-rep 故障	①
	vip-master 故障	②
	PostgreSQL故障	②
ネットワーク故障	レプリケーションLAN故障	①
	サービスLAN故障	②
	ハートビートLAN故障	③
ノード故障	カーネルハング	③
	電源停止	③
Pacemakerプロセス故障	pacemakerdプロセス故障	①
	corosyncプロセス故障	③
	cibプロセス故障	③
ディスク故障	内蔵ディスク故障	② or ③
リソース停止失敗	vip-rep	②
	vip-master	③
	PostgreSQL(demote失敗)	③
	PostgreSQL(stop失敗)	③

アプリケーション監視・制御
ネットワーク監視・制御
ノード監視
自己監視
ディスク監視・制御
アプリケーション監視・制御

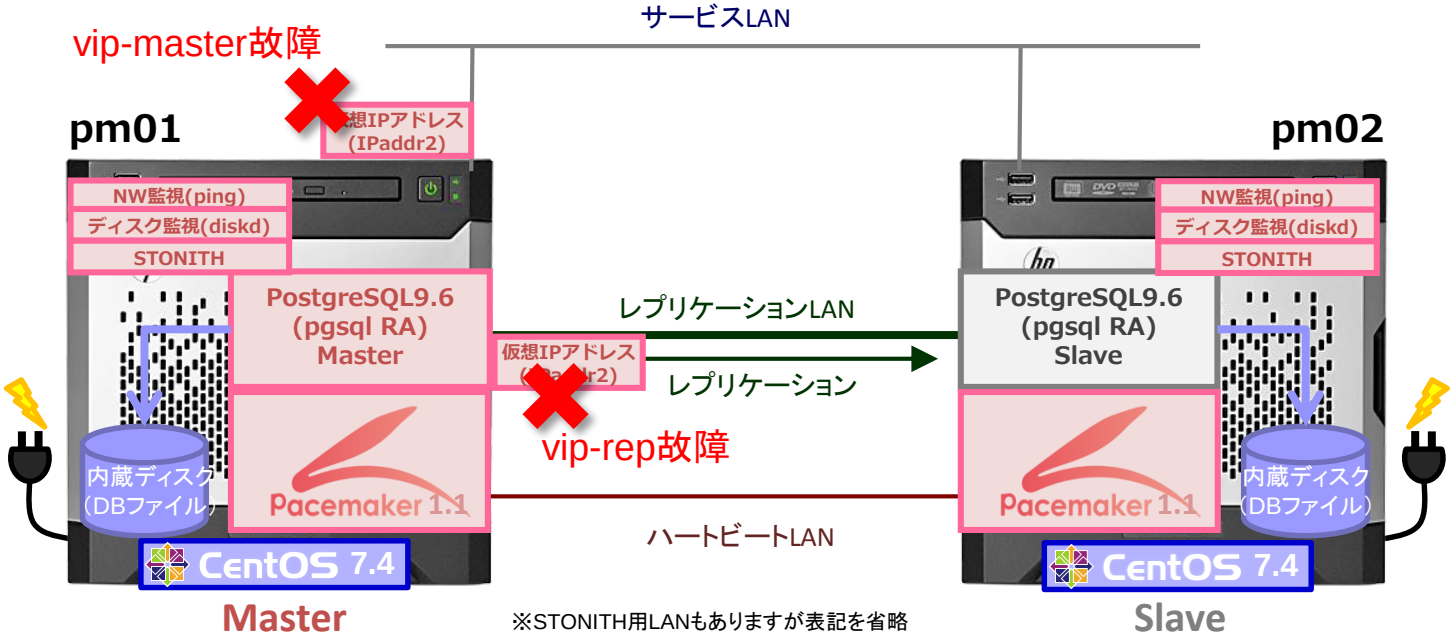
①リソース／プロセス再起動
②通常F.O
③STONITH後F.O
詳しくは次ページで説明

リソース故障

リソース故障について

大項目	小項目	Pacemakerの動作	故障再現方法(コマンド)	
リソース故障	vip-rep 故障	①	# ip addr del <vip-repｱﾄﾞﾚｽ> dev ethX	実例紹介
	vip-master 故障	②	# ip addr del <vip-masterｱﾄﾞﾚｽ> dev ethX	実例紹介
	PostgreSQL故障	②	# pg_ctl -m i stop	割愛(上と同様↑)

リソースの起動時および稼働中にRAが故障を検知するパターン
本項ではvip-repとvip-master故障時の動作(処理の流れ)と復旧方法について紹介
また、これらのリソースの故障においてPacemakerの動作が異なる(①or②)理由も補足



リソース(vip-rep)故障時の動作

✓ 故障動作例：vip-rep故障

リソース(vip-rep)故障時の動作

✓ vip-rep故障時の監視画面(正常時との差分箇所を色文字で記載)

故障前

```
~~
 Resource Group: master-group
     vip-master     (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep        (ocf::heartbeat:IPaddr2):    Started pm01
~~
Node Attributes:
* Node pm01:
    + default_ping_set                 : 100
    + diskcheck_status                 : normal
    + diskcheck_status_internal        : normal
    + master-pgsql                     : 1000
    + pgsql-data-status                : LATEST
    + pgsql-master-baseline            : 000000000B000140
    + pgsql-status                     : PRI
    + ringnumber_0                     : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                 : 100
    + diskcheck_status                 : normal
    + diskcheck_status_internal        : normal
    + master-pgsql                     : 100
    + pgsql-data-status                : STREAMING|SYNC
    + pgsql-status                     : HS:sync
    + ringnumber_0                     : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

故障後

```
~~
 Resource Group: master-group
     vip-master     (ocf::heartbeat:IPaddr2):        Started pm01
     vip-rep        (ocf::heartbeat:IPaddr2):        Started pm01
~~
Node Attributes:
* Node pm01:
    + default_ping_set                 : 100
    + diskcheck_status                 : normal
    + diskcheck_status_internal        : normal
    + master-pgsql                     : 1000
    + pgsql-data-status                : LATEST
    + pgsql-master-baseline            : 0000000009003D80
    + pgsql-status                     : PRI
    + ringnumber_0                     : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                 : 100
    + diskcheck_status                 : normal
    + diskcheck_status_internal        : normal
    + master-pgsql                     : 100
    + pgsql-data-status                : STREAMING|SYNC
    + pgsql-status                     : HS:sync
    + ringnumber_0                     : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
   vip-rep: migration-threshold=0 fail-count=1 last-failure='Fri Jan  5 10:37:19 2018'
* Node pm02:

Failed Actions:
* vip-rep_monitor_10000 on pm01 'not running' (7): call=66, status=complete,
exitreason='none',
    last-rc-change='Fri Jan  5 10:37:19 2018', queued=0ms, exec=0ms
```

vip-repは再起動されるため、リソースおよび属性値に変化はない

pm01のMigration summaryおよび"Failed actions"にvip-repで故障が発生した旨が表示される

リソース(vip-rep)故障時の動作

✓ vip-rep故障時のログ

pm_logconv.outより抜粋(Master側)

```
Jan 5 10:37:19 pm01   error: Resource vip-rep does not work. (rc=7) not running
```

vip-repの故障検知

```
Jan 5 10:37:20 pm01   info: Resource vip-rep tries to stop.
Jan 5 10:37:20 pm01   info: Resource vip-rep stopped. (rc=0) ok
Jan 5 10:37:20 pm01   info: Resource vip-rep tries to start.
Jan 5 10:37:20 pm01   info: Resource vip-rep started. (rc=0) ok
```

vip-repの復旧(再起動)

リソース(vip-rep)故障の復旧方法

✓ 手順(といっても以下の一つのみです)

1.フェイルカウントを削除

```
# crm_resource -C -r vip-rep -N pm01
```

故障リソース名↑　　↑ノード名

✓ crm_monに故障情報が表示されたままだと、
以降の故障時に、そのノードにはフェイルオーバ不可

- Pacemakerはそのノードを"異常"と思っている
- この状態を“フェイルカウント”が立っているという

✓ 故障の原因を取り除いたら、上記コマンドで“フェイルカウント”を削除
＝Pacemakerにもう異常はないという通知

- フェイルカウントが削除され、監視画面の表示が元に戻る

故障情報

```
~~
Migration Summary:
* Node pm01:
   vip-rep: migration-threshold=0 fail-count=1 last-failure='Fri Jan  5 10:37:19 2018'
* Node pm02:

Failed Actions:
* vip-rep_monitor_10000 on pm01 'not running' (7): call=66, status=complete, exitreason='none',
    last-rc-change='Fri Jan  5 10:37:19 2018', queued=0ms, exec=0ms
```

これらが消える

リソース(vip-master)故障時の動作

✓ 故障動作例：vip-master故障

処理の流れ

ノード1(Master)	ノード2(Slave)
正常稼働中・・・	正常稼働中・・・
1.vip-master削除(疑似故障)	
2.vip-master故障検知	
3.PostgreSQL demote	
4.vip-rep停止	
5.vip-master停止	
(PostgreSQL故障検知)	
6.PostgreSQL停止	
	7.PostgreSQL promote
	8.vip-master起動
	9.vip-rep起動

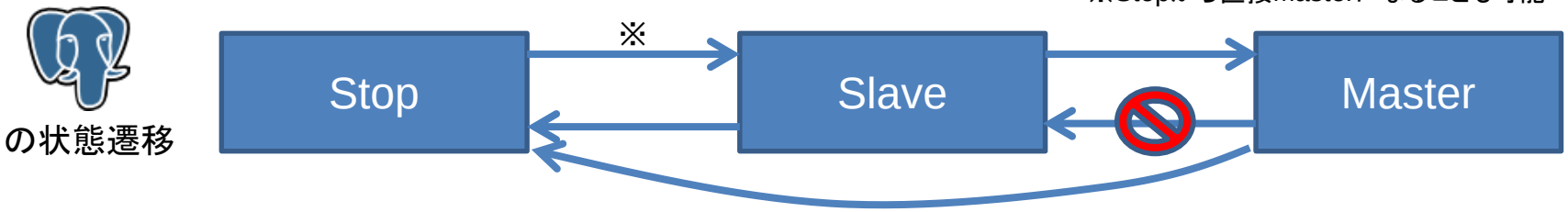
 # ip addr del <vip-masterｱﾄﾞﾚｽ> dev ethX

何故故障＆停止？？

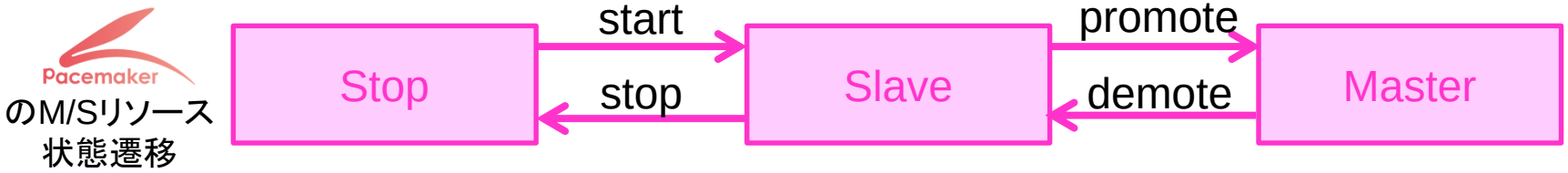
PostgreSQLが故障と判定される理由

PostgreSQLは仕様上、MasterをSlaveに降格(demote)という状態遷移ができないため、Masterは直接停止するしかない

※Stopから直接Masterになることも可能



一方、PacemakerはMaster/Slaveリソースを、「start」「promote」「demote」「stop」の4つの命令で、それぞれの状態への制御を試みる



そのため、PacemakerがMasterを「demote」すると、(Pacemakerの思惑とは異なり)PostgreSQLは停止してしまう
よって、残念ながらPG-REXでは、Master故障でのフェイルオーバ後は、旧Masterは停止し、PostgreSQLが片系で稼働する状態となる

また、「demote」→「stop」の間にpgsql RAによる監視(monitor)が実行されてしまうと、故障と判断され、監視画面に故障情報が表示されることになる

リソース(vip-master)故障時の動作

✓ vip-master故障時の監視画面(正常時との差分箇所を色文字で記載)

故障前

```
~~
 Resource Group: master-group
     vip-master    (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep       (ocf::heartbeat:IPaddr2):    Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status                : normal
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline           : 000000000B000140
    + pgsql-status                    : PRI
    + ringnumber_0                    : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status                : normal
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 100
    + pgsql-data-status               : STREAMING|SYNC
    + pgsql-status                    : HS:sync
    + ringnumber_0                    : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

故障後

```
~~
 Resource Group: master-group
     vip-master    (ocf::heartbeat:IPaddr2):    Started pm02
     vip-rep       (ocf::heartbeat:IPaddr2):    Started pm02
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm02 ]
     Stopped: [ pm01 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status                : normal
    + diskcheck_status_internal       : normal
    + master-pgsql                    : -INFINITY
    + pgsql-data-status               : DISCONNECT
    + pgsql-status                    : STOP
    + ringnumber_0                    : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status                : normal
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline           : 000000000D000060
    + pgsql-status                    : PRI
    + ringnumber_0                    : 192.168.11.32 is UP
~~
```

フェイルオーバし、pm02がMaster(リソースが起動)になる

~次ページに続く

リソース(vip-master)故障時の動作

✓ vip-master故障時の監視画面(正常時との差分箇所を色文字で記載)

故障前

```
~~
 Resource Group: master-group
     vip-master     (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep        (ocf::heartbeat:IPaddr2):    Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                 : 100
    + diskcheck_status                 : normal
    + diskcheck_status_internal        : normal
    + master-pgsql                     : 1000
    + pgsql-data-status                : LATEST
    + pgsql-master-baseline            : 000000000B000140
    + pgsql-status                     : PRI
    + ringnumber_0                     : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                 : 100
    + diskcheck_status                 : normal
    + diskcheck_status_internal        : normal
    + master-pgsql                     : 100
    + pgsql-data-status                : STREAMING|SYNC
    + pgsql-status                     : HS:sync
    + ringnumber_0                     : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

~前ページからの続き

故障後

```
~~
Migration Summary:
* Node pm01:
   vip-master: migration-threshold=1 fail-count=1 last-failure='Fri Jan  5 10:51:06
2018'
   pgsql: migration-threshold=1 fail-count=1 last-failure='Fri Jan  5 10:51:09 2018'
* Node pm02:

Failed Actions:
* vip-master_monitor_10000 on pm01 'not running' (7): call=64, status=complete,
exitreason='none',
    last-rc-change='Fri Jan  5 10:51:06 2018', queued=0ms, exec=0ms
* pgsql_monitor_10000 on pm01 'not running' (7): call=79, status=complete,
exitreason='none',
    last-rc-change='Fri Jan  5 10:51:09 2018', queued=0ms, exec=89ms
```

pm01のMigration summaryおよび"Failed actions"にvip-masterで故障が発生した旨が表示される

同時にPostgreSQLの故障も表示される(37ページで説明した通り、「demote」→「stop」の間にpgsql RAによるmonitorが実行されたため)

リソース(vip-master)故障時の動作

✓vip-master故障時のログ

pm_logconv.outより抜粋(Master側)

```
Jan 5 10:51:06 pm01   error: Resource vip-master does not work. (rc=7) not running

Jan 5 10:51:06 pm01   error: Start to fail-over.
Jan 5 10:51:06 pm01    info: Resource pgsql tries to demote.
～(略)～
Jan 5 10:51:06 pm01    info: Resource pgsql demoted. (rc=0) ok
～(略)～

Jan 5 10:51:07 pm01    info: Resource vip-rep tries to stop.
Jan 5 10:51:07 pm01    info: Resource vip-rep stopped. (rc=0) ok
Jan 5 10:51:07 pm01    info: Resource vip-master tries to stop.
Jan 5 10:51:07 pm01    info: Resource vip-master stopped. (rc=0) ok
～(略)～

Jan 5 10:51:09 pm01   error: Resource pgsql does not work. (rc=7) not running
Jan 5 10:51:09 pm01    info: Resource pgsql tries to stop.
Jan 5 10:51:09 pm01    info: Resource pgsql stopped. (rc=0) ok



Jan 5 10:51:09 pm01    info: Resource vip-master : Started on pm02
Jan 5 10:51:09 pm01    info: Resource vip-rep : Started on pm02
Jan 5 10:51:09 pm01    info: fail-over succeeded.
```

vip-masterの故障検知

PostgreSQLのdemote(降格)／停止

仮想IP(VIP)の停止

demoteによりPostgreSQLが停止してしまうため故障と判定される（フェイルオーバは継続する）

pm02で仮想IPの起動

リソース(vip-master)故障時の動作

✓ vip-master故障時のログ

pm_logconv.outより抜粋(Slave側)

```
Jan 5 10:51:07 pm02    info: Resource pgsql tries to promote.
～(略)～
Jan 5 10:51:08 pm02    info: Resource pgsql promoted. (rc=0) ok
```

PostgreSQLのpromote

```
Jan 5 10:51:08 pm02    info: Resource vip-master tries to start.
Jan 5 10:51:08 pm02    info: Resource vip-master started. (rc=0) ok
Jan 5 10:51:09 pm02    info: Resource vip-rep tries to start.
Jan 5 10:51:09 pm02    info: Resource vip-rep started. (rc=0) ok
```

仮想IP(VIP)の起動

リソース(vip-master)故障の復旧方法

✓ 手順

1.故障したノードのPacemakerを一旦停止する
→“フェイルカウント”はPacemakerを停止すると削除される

2.vip-masterが故障した原因を確認し、復旧する

3.故障したノードの起動禁止フラグ「/var/lib/pgsql/tmp/PGSQL.lock」を削除
→このファイルが存在するとPostgreSQLが起動しない(詳しい話は付録ページへ)

4.故障したノードをSlaveとして初期構築と同様に組み込む

```
# pg-rex_slave_start
```

5.(任意)故障したノードを再度Masterにしたい場合、手動でF.Oを発生させる

```
# pg-rex_switchover
```

リソース(vip-master)故障の復旧方法

✓ 手順

1.故障したノードのPacemakerを一旦停止する
→“フェイルカウント”はPacemakerを停止すると削除される

2.vip-masterが故障した原因を確認し、復旧する

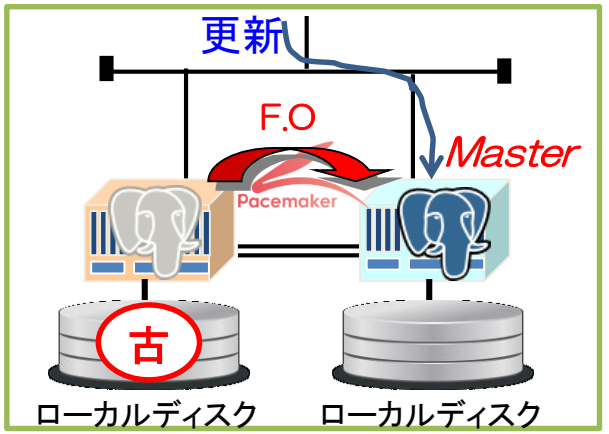
3.故障したノードの起動禁止フラグ「/var/lib/pgsql/tmp/PGSQL.lock」を削除
→このファイルが存在するとPostgreSQLが起動しない(詳しい話は付録ページへ)

4.故障したノードをSlaveとして初期構築

```
# pg-rex_slave_start
```

5.(任意)故障したノードを再度Masterにし

```
# pg-rex_switchover
```



データ差異がある状態で
クラスタに組み込むのはNG

起動禁止フラグ＝組み込み防止

リソース(vip-master)故障の復旧方法

✓ 手順

1.故障したノードのPacemakerを一旦停止する
→“フェイルカウント”はPacemakerを停止すると削除される

2.vip-masterが故障した原因を確認し、復旧する

3.故障したノードの起動禁止フラグ「/var/lib/pgsql/tmp/PGSQL.lock」を削除
→このファイルが存在するとPostgreSQLが起動しない(詳しい話は付録ページへ)

4.故障したノードをSlaveとして初期構築と同様に組み込む

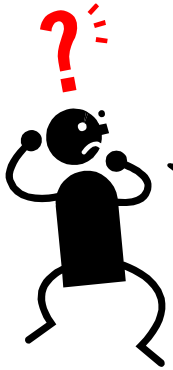
```
# pg-rex_slave_start
```

5.(任意)故障したノードを再度Masterにしたい場合、手動でF.Oを発生させる

```
# pg-rex_switchover
```

【補足】故障時の動作の違い

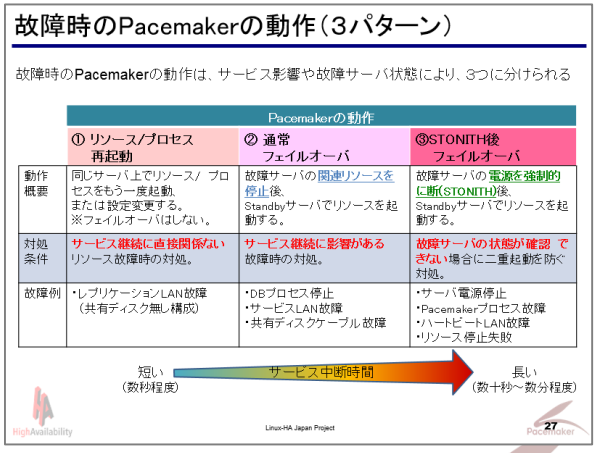
ところで・・・



Q.vip-repとvip-masterでなぜ故障時の動作がこんなにも異なるの？　仕様？？

A.もちろん仕様ではありません！**設定した内容に基づいた動作**です
本例では、以下のような要件を定めて設定をしています

このページにも記載した通り



- **vip-rep**の故障は直ちにサービスに影響はないため、リソースを再起動する
- **vip-master**の故障はサービス停止につながるため、vip-masterおよび関連リソースを正常稼働中のノードに直ちにフェイルオーバさせる

【補足】設定と故障時の動作

✓ 故障動作例：vip-rep故障

処理の流れ

ノード1(Master)	ノード2(Slave)
正常稼働中・・・	正常稼働中・・・
1.vip-rep削除(疑似故障)	
2.vip-rep故障検知	
3.vip-rep復旧(再起動)	

①vip-rep故障により自ノードで再起動
これは「migration-threshold=“0”」のため
本パラメータは何回故障したらF.Oするかを決めるもので、
0だと何度故障してもF.Oはせずリソースを再起動する
（他リソースは1に設定）

crm設定抜粋

```
～～
primitive vip-rep ocf:heartbeat:IPaddr2 ¥
        params ¥
                ip="192.168.2.30" ¥
                nic="eno3" ¥
                cidr_netmask="24" ¥
        meta ¥
                migration-threshold="0" ¥
        op start interval="0s" timeout="60s" on-fail="stop" ¥
        op monitor interval="10s" timeout="60s" on-fail="restart" ¥
        op stop interval=“0s” timeout=“60s” on-fail=“ignore”
～～
```

【補足】設定と故障時の動作

✓ 故障動作例：vip-master故障

ノード1(Master)

正常稼働中・・・

1.vip-master削除(疑似故障)

2.vip-master故障検知

3.PostgreSQL demote

4.vip-rep停止

5.vip-master停止

(PostgreSQL故障検知)

6.PostgreSQL停止

処理の流れ

①vip-master故障によりF.O(migration-threshold=“1”)
②groupリソース(master-group)のためvip-repも併せてF.O＆vip-rep→vip-masterの順序で停止
③ master-groupがF.Oするため、colocation制約に従い、PostgreSQLをdemote

crm設定抜粋

```
~~
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1“
~~
Group master-group    vip-master  vip-rep
~~
 primitive vip-master ocf:heartbeat:IPaddr2 ¥
                    params ¥
                                        ip="192.168.1.30" ¥
                                        nic="eno2" ¥
                                        cidr_netmask="24" ¥
                    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="fence“
~~
colocation rsc_colocation-master-group-msPostgresql-3 INFINITY:
master-group msPostgresql:Master
~~
```

【補足】設定と故障時の動作

✓ 故障動作例：vip-master故障

処理の流れ

ノード1(Master)	
正常稼働中・・・	
1.vip-master削除(疑似故障)	
2.vip-master故障検知	
3.PostgreSQL demote	
4.vip-rep停止	
5.vip-master停止	
(PostgreSQL故障検知)	
6.PostgreSQL停止	
	7.PostgreSQL promote
	8.vip-master起動
	9.vip-rep起動

④PostgreSQLをpromote(③の続き)。order制約によりmaster-groupが起動する前にpromoteされる
⑤ノード1からF.O(②の続き)。groupリソースのためvip-master→vip-repの順序で起動

crm設定抜粋

```
~~
group master-group    vip-master  vip-rep
~~
order rsc_order-msPostgresql-master-group-4 INFINITY:
msPostgresql:promote master-group:start symmetrical=false
~~
```

crm設定に登場するパラメータについて詳しく知りたい場合は、Linux-HA Japanのページの「読み物」カテゴリーにある、「動かして理解するPacemaker ～CRM設定編～」「Pacemaker で扱えるリソースの種類 (Primitive, Clone, Master/Slave, Group)」等の記事をご覧下さい
http://linux-ha.osdn.jp/wp/archives/category/doc

【補足】設定と故障時の動作

このように、設定を把握し「なぜそのような動作になるか」を
理解できていれば故障時にもあせらず対応ができる(はず)！

- 解析担当者に質問する際に事象の説明がし易い
- 故障時に、想定通りに動作(F.O等)しているかを確認できる
- 故障時に適切な対処方法を選択することができる

etc..

ネットワーク故障

ネットワーク故障について

大項目	小項目	Pacemakerの動作	故障再現方法(コマンド)※
ネットワーク故障	レプリケーションLAN故障	①	# iptables -A INPUT -i ethX -j DROP # iptables -A OUTPUT -o ethX -j DROP
	サービスLAN故障	②	# iptables -A INPUT -i ethX -j DROP # iptables -A OUTPUT -o ethX -j DROP
	ハートビートLAN故障	③	# iptables -A INPUT -i ethX -j DROP # iptables -A INPUT -i ethY -j DROP # iptables -A OUTPUT -o ethX -j DROP # iptables -A OUTPUT -o ethY -j DROP

実例紹介

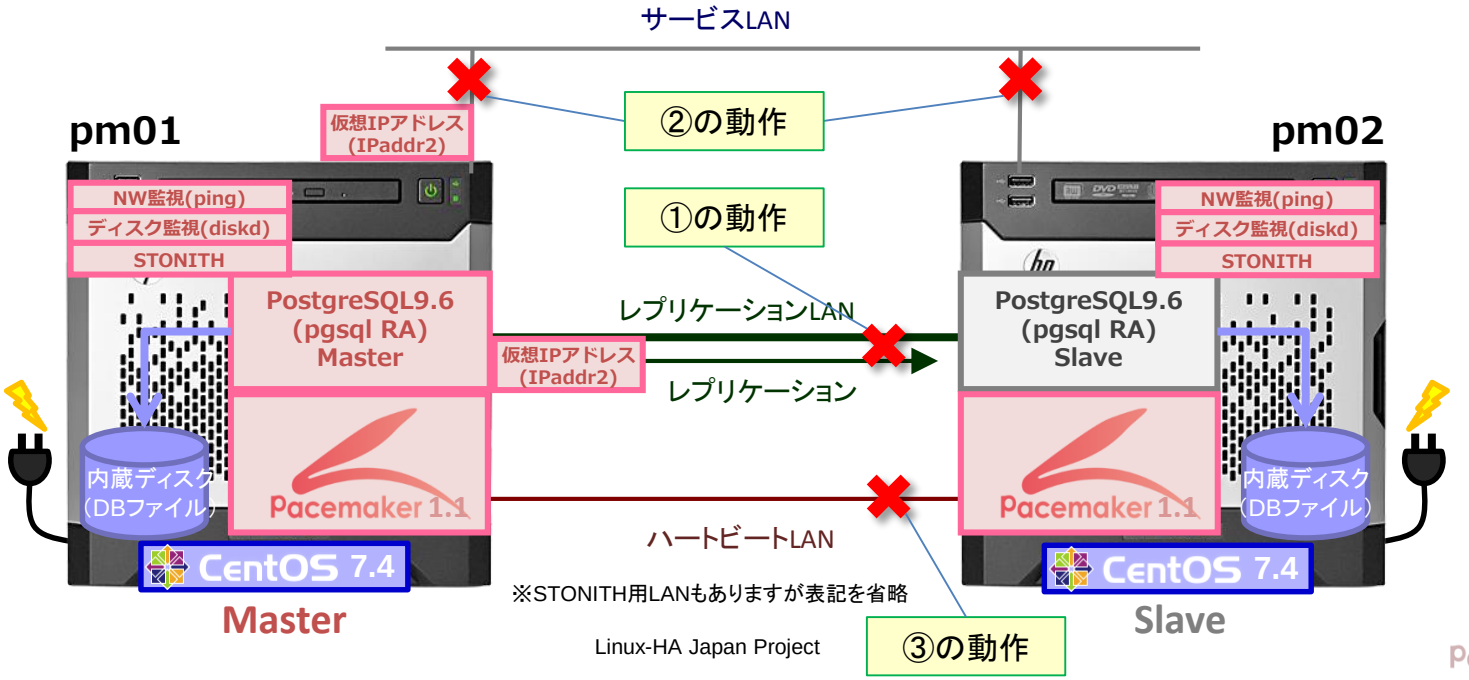
実例紹介

実例紹介

本編にて紹介

※INPUTとOUTPUTを連続して遮断します(「;」で区切って1行で実行するとよい)

NIC故障や断線等でネットワークが切れる故障
切れたネットワークの用途によってPacemakerの挙動が大きく異なる(下図参照)



ハートビートLAN故障時の動作

✓ 故障動作例：ハートビートLAN故障

処理の流れ

ノード1(Master)	ノード2(Slave)
正常稼働中・・・	正常稼働中・・・
1.ハートビートLAN切断(疑似故障)	
2.ハートビートLAN断検知	2.ハートビートLAN断検知
3.STONITH実行	3.対向ノードからのSTONITH実行待ち
	4.MasterからのSTONITHにより停止

```
# iptables -A INPUT -i ethX -j DROP; iptables -A INPUT -i ethY -j DROP;
iptables -A OUTPUT -o ethX -j DROP; iptables -A OUTPUT -o ethY -j DROP
```

- 「stonith-helper」(Linux-HA Japan が独自に提供するリソース)により、STONITHの「相撃ち」を防止

  自身がMasterの場合は対向ノードに即座にSTONITH実行、
  自身がSlaveの場合は一定時間STONITHを待つ
  詳しい内容については「試して覚えるPacemaker入門 排他制御機能編」をご覧下さい
  http://linux-ha.osdn.jp/wp/archives/4338

ハートビートLAN故障時の動作

✓ ハートビートLAN故障時の監視画面(正常時との差分箇所を色文字で記載)

故障前

```
Online: [ pm01 pm02 ]

~~
 Resource Group: master-group
     vip-master     (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep        (ocf::heartbeat:IPaddr2):    Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline           : 0000000019000250
    + pgsql-status                    : PRI
    + ringnumber_0                    : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status                : normal
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 100
    + pgsql-data-status               : STREAMING|SYNC
    + pgsql-status                    : HS:sync
    + ringnumber_0                    : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

故障後

```
Online: [ pm01 ]
OFFLINE: [ pm02 ]
~~
 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep    (ocf::heartbeat:IPaddr2):    Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Stopped: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline           : 0000000019000250
    + pgsql-status                    : PRI
    + ringnumber_0                    : 192.168.11.31 is UP

Migration Summary:
* Node pm01:
```

pm02がOFFLINE(クラスタ未参加)に

リソースはpm01で起動したまま

pm02の属性値表示は無くなる
※故障前と比較しやすいよう空白としているが、実際には空白はない

ハートビートLAN故障時の動作

✓ ハートビートLAN故障時のログ

pm_logconv.outより抜粋(Master側)　(Slave側のpm_logconv.outはSTONITHにより未出力)

```
Jan 9 13:18:59 pm01 warning: Node pm02 is lost
```

(ハートビートLAN断により)対向ノード(pm02)が離脱したと判断

```
Jan 9 13:19:00 pm01    info: Try to STONITH (reboot) pm02.
Jan 9 13:19:00 pm01    info: Try to execute STONITH device prmStonith2-1 on pm01 for reboot pm02.
Jan 9 13:19:04 pm01 warning: Failed to execute STONITH device prmStonith2-1 for pm02.
```

stonith-helper稼働※

```
Jan 9 13:19:04 pm01    info: Try to execute STONITH device prmStonith2-2 on pm01 for reboot pm02.
Jan 9 13:19:05 pm01    info: Succeeded to execute STONITH device prmStonith2-2 for pm02.
Jan 9 13:19:05 pm01    info: Succeeded to STONITH (reboot) pm02 by pm01.
```

pm01から対向ノード(pm02)へSTONITH実行

※stonith-helperは前述の「相撃ち」防止以外にもノードの停止判定を行う機能がある
本例において、stonith-helper(prmStonith2-1)はまず、
対向ノード(dead_check_targetパラメータに設定したIPアドレス)にpingを行う

これに対して応答が返ってきた場合は対向ノードが停止していないものとみなし
(「Failed to execute STONITH」はこれを示すログ)、対向ノードを停止させるため
次のプラグイン(本例ではprmStonith2-2すなわちipmi)に処理をエスカレーションする

ハートビートLAN故障時の動作

✓ ハートビートLAN故障時のログ

STONITHにより、故障時間付近はha-logからpm_logconv.outへの変換ログが出力されない場合もあるため※、必要に応じてha-logも参照するとよい

ha-logより抜粋(Slave側)

```
Jan  9 13:18:59 pm02 corosync[28630]: [TOTEM ] A new membership (192.168.11.32:244) was formed.
Members left: 3232238367
```

対向ノード(pm01)が離脱したと判断
(クラスタメンバが自ノードのみとなった)

```
～(略)～
Jan  9 13:19:00 pm02 pengine[28643]: warning: Scheduling Node pm01 for STONITH
～(略)～
```

対向ノード(pm01)にSTONITHを実行すると判断

```
Jan  9 13:19:00 pm02 stonith-ng[28640]:    info: Requesting that 'pm02' perform op 'pm01 reboot' with
'prmStonith1-1' for crmd.28644 (48s)
```

stonith-helper(prmStonith1-1)の稼働

※pm_logconvはha-logをリアルタイムに変換しているが、変換されたログがディスクに同期される前にSTONITHによりノードが停止してしまうとpm_logconv.outにログが出力されない

ハートビートLAN故障の復旧方法

✓ 手順

1.故障したハートビートLANを復旧する

2.故障したノードを再度Slaveとして初期構築と同様に組み込む

```
# pg-rex_slave_start
```

コミュニティ紹介

コミュニティ紹介

Linux-HA Japan URL

http://linux-ha.osdn.jp/

https://ja.osdn.net/projects/linux-ha/

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)

コミュニティ紹介

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.osdn.jp/
  の「メーリングリスト」をクリック

- MLアドレス

linux-ha-japan@lists.osdn.me

※スパム防止のために、登録者以外の投稿は許可制です

コミュニティ紹介

PG-REX URL

https://ja.osdn.net/projects/pg-rex/

PG-REXの構築手順書やpm_crmgen_env.xls、PG-REX運用補助ツールを提供しています

本セミナーでは説明できなかった詳細内容も上記手順書に詳しく書いてあるので是非読んでください！

ご清聴ありがとうございました

付録

crm_monコマンドのオプション

No	オプション	説明
1	--help (-?)	オプションを表示
2	--verbose (-V)	デバック情報を表示
3	--group-by-node (-n)	ノード単位のリソースグループを表示
4	--simple-status (-s)	一行表示のクラスタ状態を表示
5	--inactive (-r)	停止状態中リソースを含む全てのリソースを表示
6	--one-shot (-1)	クラスタ状態を一回だけモニタに表示
7	--failcounts (-f)	リソースの故障回数を表示
8	--show-node-attributes(-A)	ノード毎のインターコネクトLAN状態、ディスク監視、ネットワーク監視の状態などを表示
9	--neg-locations(-L)	実行不可制約を表示(Pacemaker-1.1.13以降で使用可能なオプション)

運用者観点では、大抵の場合は「crm –rfA」で事足りると思いますので暗記してしまいましょう！

ログ関連の設定

✓Pacemakerログのha-logへの切り出し手順

vi /etc/corosync/corosync.conf
→syslog_facilityを「local1」等、他で使用していないものに変更。

vi /etc/sysconfig/pacemaker
→「PCMK_logfacility=local1」という設定を追記。

vi /etc/rsyslog.conf
→色文字部分を追記

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

vi /etc/logrotate.d/syslog
→色文字部分を追記(ローテート設定)

```
/var/log/spooler
/var/log/ha-log
{
    missingok
```

rm –f /etc/logrotate.d/pacemaker
→インストール時に作成されるが、上記設定をした場合は使用しないため削除

最後にrsyslog、pacemaker等を再起動し、設定を反映

ログ関連の設定

✓pm_logconvの設定

 # cp -p /etc/pm_logconv.conf.sample /etc/pm_logconv.conf
→サンプルコンフィグをコピーする

 # vi /etc/pm_logconv.conf
→色文字部分を追記

```
[Settings]
#ha_log_path = /var/log/ha-log
#output_path = /var/log/pm_logconv.out
～(略)～
#act_rsc = prmExPostgreSQLDB, prmApPostgreSQLDB

# ネットワーク監視を行う場合の設定
attribute_ping = not_defined default_ping_set or default_ping_set lt 100
# 内蔵ディスク監視を行う場合の設定
attribute_diskd_inner = not_defined diskcheck_status_internal or diskcheck_status_internal eq ERROR
# フェイルオーバの発生と成否の判断基準となるリソースのIDを設定
act_rsc = vip-master, vip-rep
```

 # systemctl restart pm_logconv
→最後にpm_logconvを再起動し、設定を反映

ha-log、pm_logconvの設定を紹介しましたが、細かい点については
PG-REX利用マニュアルに記載がありますので、そちらも併せてご覧下さい

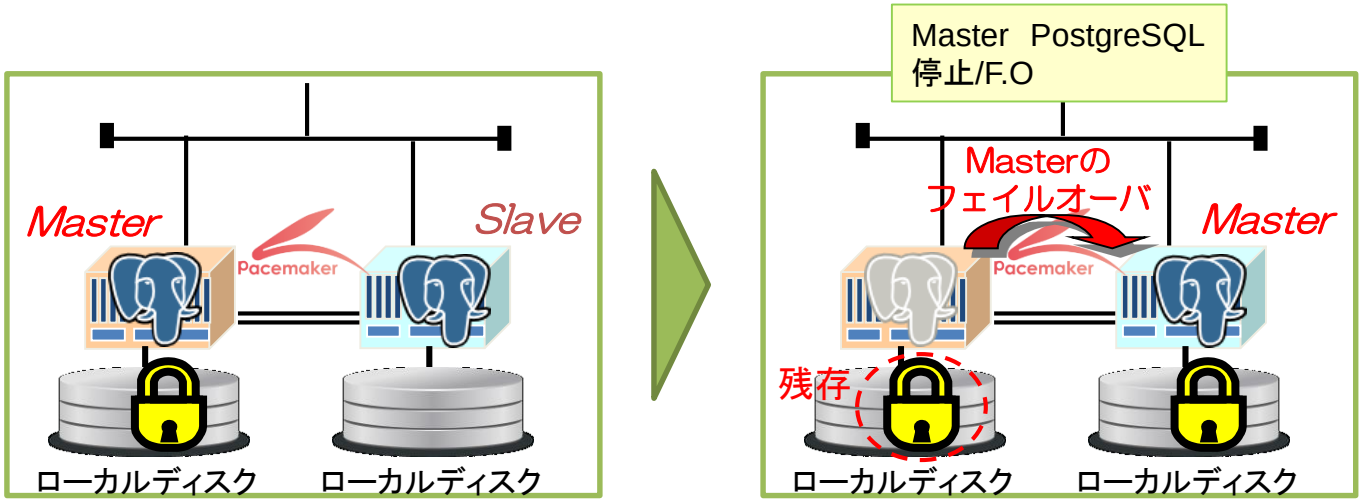
起動禁止フラグ

✓起動禁止フラグ(ロックファイル)とは

- pgsql RAによりPostgreSQLがMasterに昇格する際に作成される空ファイル(デフォルトで「/var/lib/pgsql/tmp/PGSQL.lock」に作成される)
- 他のSlaveがいない状態でのMaster正常停止時に削除される

ロックファイルの残存＝最新データに対して不整合がある可能性を示唆

ロックファイルが残存している場合はPostgreSQLを起動不可
そのため、復旧にあたっては本ファイルを削除したうえで
ベースバックアップを取得しなおすこと



本編では紹介しなかった故障パターン

ネットワーク故障

ネットワーク故障について

大項目	小項目	Pacemakerの動作	故障再現方法(コマンド)※
ネットワーク故障	レプリケーションLAN故障	①	# iptables -A INPUT -i ethX -j DROP # iptables -A OUTPUT -o ethX -j DROP
	サービスLAN故障	②	# iptables -A INPUT -i ethX -j DROP # iptables -A OUTPUT -o ethX -j DROP
	ハートビートLAN故障	③	# iptables -A INPUT -i ethX -j DROP # iptables -A INPUT -i ethY -j DROP # iptables -A OUTPUT -o ethX -j DROP # iptables -A OUTPUT -o ethY -j DROP

実例紹介

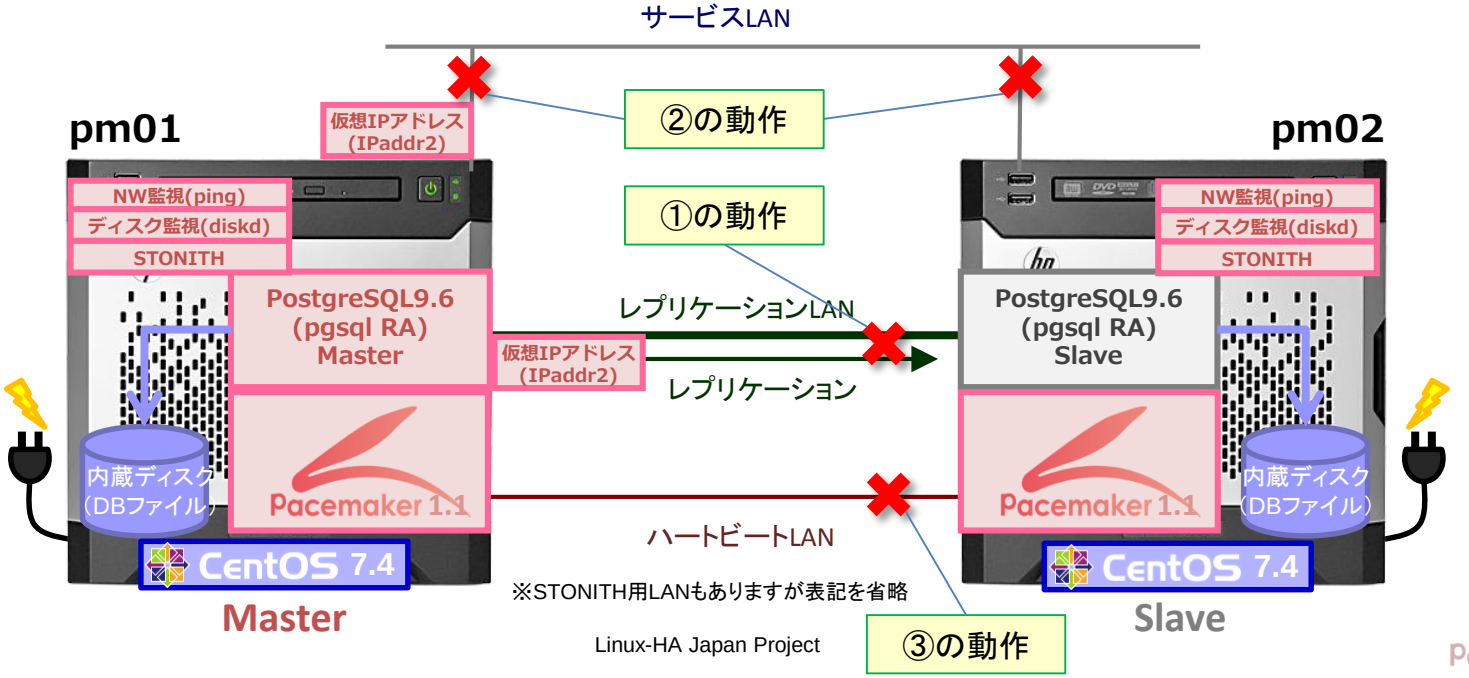
実例紹介

実例紹介

付録にて紹介

※INPUTとOUTPUTを連続して遮断します(「;」で区切って1行で実行するとよい)

NIC故障や断線等でネットワークが切れる故障
切れたネットワークの用途によってPacemakerの挙動が大きく異なる(下図参照)



レプリケーションLAN故障時の動作

✓ 故障動作例：レプリケーションLAN故障

処理の流れ

ノード1(Master)	ノード2(Slave)
正常稼働中・・・	正常稼働中・・・
1.レプリケーションLAN切断(疑似故障)	
2.Slave側がMasterになれないように属性値を変更(master-pgsql="-INFINITY").	
	3.Master側との接続が切れているため属性値を変更(pgsql-status="HS:alone").

 # iptables -A INPUT -i ethX -j DROP; iptables -A OUTPUT -o ethX -j DROP

- レプリケーションLAN切断に気づくのはMaster側のpgsql RAで、pg_stat_replicationビューを監視しており故障検知までwal_sender_timeoutに設定の時間がかかる
- この後、pgsql RAによりPostgreSQLのsynchronous_standby_namesを空に変更し、Slaveを同期モードから切離す処理を実施

レプリケーションLAN故障時の動作

✓レプリケーションLAN故障時の監視画面(正常時との差分箇所を色文字で記載)

故障前

```
Online: [ pm01 pm02 ]
~~
 Resource Group: master-group
     vip-master    (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep       (ocf::heartbeat:IPaddr2):    Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline       : 0000000012000410
    + pgsql-status                    : PRI
    + ringnumber_0                  : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                    : 100
    + pgsql-data-status               : STREAMING|SYNC
    + pgsql-status                    : HS:sync
    + ringnumber_0                  : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

故障後

```
Online: [ pm01 pm02 ]
~~
 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):      Started pm01
     vip-rep       (ocf::heartbeat:IPaddr2):      Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                   : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline        : 0000000012000410
    + pgsql-status                    : PRI
    + ringnumber_0                  : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                   : -INFINITY
    + pgsql-data-status               : DISCONNECT
    + pgsql-status                    : HS:alone
    + ringnumber_0                  : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

pm02のpgsql-*属性値が、故障を示す値になる(DISCONNECT、HS:alone)

レプリケーションLAN故障時の動作

✓レプリケーションLAN故障時のログ

pm_logconv.outより抜粋(Master側※)

```
Jan 9 11:19:21 pm01    info: Attribute "master-pgsql" is updated to "-INFINITY" at "pm02".
```

属性値を変更(master-pgsql="-INFINITY")

```
Jan 9 11:19:31 pm01    info: Attribute "pgsql-status" is updated to "HS:alone" at "pm02".
```

属性値を変更(pgsql-status="HS:alone")

※上記属性値はSlave側のログにも同様に出力される

レプリケーションLAN故障の復旧方法

✓ 手順

1.Slave側のPacemakerを一旦停止する

2.故障したレプリケーションLANを復旧する

3.故障したノードを再度Slaveとして初期構築と同様に組み込む

```
# pg-rex_slave_start
```

サービスLAN故障時の動作

✓ 故障動作例：サービスLAN故障

処理の流れ

ノード1(Master)	ノード2(Slave)
正常稼働中・・・	正常稼働中・・・
1.サービスLAN切断 (疑似故障)	
2.サービスLAN通信(ping)断検知	
3.ping失敗により属性値を変更	
(default_ping_setを元の値から-100)	
4.PostgreSQL demote	このあたりの流れは
5.vip-rep停止	vip-master故障等、
6.vip-master停止	他の故障ケースと同じ
7.PostgreSQL停止	ただし、今回はPostgreSQL
	故障は検知されていない
	8.PostgreSQL promote
	9.vip-master起動
	10.vip-rep起動

 # iptables -A INPUT -i ethX -j DROP; iptables -A OUTPUT -o ethX -j DROP

サービスLAN故障時の動作

✓サービスLAN故障時の監視画面(正常時との差分箇所を色文字で記載)

故障前

```
Online: [ pm01 pm02 ]
~~
 Resource Group: master-group
     vip-master    (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep       (ocf::heartbeat:IPaddr2):    Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline       : 0000000016000330
    + pgsql-status                    : PRI
    + ringnumber_0                  : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                    : 100
    + pgsql-data-status               : STREAMING|SYNC
    + pgsql-status                    : HS:sync
    + ringnumber_0                  : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

故障後

```
Online: [ pm01 pm02 ]
~~
 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):      Started pm02
     vip-rep       (ocf::heartbeat:IPaddr2):      Started pm02
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm02 ]
     Stopped: [ pm01 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                : 0       : Connectivity is lost
    + diskcheck_status_internal   : normal
    + master-pgsql                    : -INFINITY
    + pgsql-data-status               : DISCONNECT
    + pgsql-status                    : STOP
    + ringnumber_0                  : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline       : 0000000016000440
    + pgsql-status                    : PRI
    + ringnumber_0                  : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

リソースがpm02で起動/Masterになる

pm01の属性値default_ping_setが0になる

pm01のPostgreSQLの属性値は停止を示す値に

pm02のPostgreSQLの属性値はMasterであることを示す値に

サービスLAN故障時の動作

✓サービスLAN故障時のログ

pm_logconv.outより抜粋(Master側)

```
Jan 9 11:49:04 pm01   error: Network to 192.168.1.17 is unreachable.
```

サービスLAN通信断検知(復旧するまで継続して出力される)

```
Jan 9 11:49:04 pm01    info: Attribute "default_ping_set" is updated to "0" at "pm01".
```

属性値を変更(default_ping_setを元の値から-100)

```
Jan 9 11:49:09 pm01   error: Start to fail-over.
Jan 9 11:49:09 pm01    info: Resource pgsql tries to demote.
～(略)～
Jan 9 11:49:09 pm01    info: Resource pgsql demoted. (rc=0) ok
～(略)～
Jan 9 11:49:09 pm01    info: Resource vip-rep tries to stop.
Jan 9 11:49:09 pm01    info: Resource vip-rep stopped. (rc=0) ok
Jan 9 11:49:09 pm01    info: Resource vip-master tries to stop.
Jan 9 11:49:09 pm01    info: Resource vip-master stopped. (rc=0) ok
Jan 9 11:49:09 pm01    info: Resource pgsql tries to stop.
Jan 9 11:49:10 pm01    info: Resource pgsql stopped. (rc=0) ok
～(略)～
Jan 9 11:49:12 pm01    info: Resource vip-master : Started on pm02
Jan 9 11:49:12 pm01    info: Resource vip-rep : Started on pm02
Jan 9 11:49:12 pm01    info: fail-over succeeded.
```

F.Oの流れは他の故障(vip-master故障等)と同じ

サービスLAN故障時の動作

✓サービスLAN故障時のログ

pm_logconv.outより抜粋(Slave側)

```
Jan 9 11:49:10 pm02    info: Resource pgsql tries to promote.
～(略～)
Jan 9 11:49:11 pm02    info: Resource pgsql promoted. (rc=0) ok
Jan 9 11:49:12 pm02    info: Resource vip-master tries to start.
Jan 9 11:49:12 pm02    info: Resource vip-master started. (rc=0) ok
Jan 9 11:49:12 pm02    info: Resource vip-rep tries to start.
Jan 9 11:49:12 pm02    info: Resource vip-rep started. (rc=0) ok
```

promoteの流れは他の故障(vip-master故障等)と同じ

サービスLAN故障の復旧方法

✓ 手順

1.故障したノードのPacemakerを一旦停止する

2.故障したサービスLANを復旧する

3.故障したノードの起動禁止フラグ「/var/lib/pgsql/tmp/PGSQL.lock」を削除

4.故障したノードを再度Slaveとして初期構築と同様に組み込む

```
# pg-rex_slave_start
```

5.(任意)故障したノードを再度Masterにしたい場合、手動でF.Oを発生させる

```
# pg-rex_switchover
```

ノード故障

ノード故障について

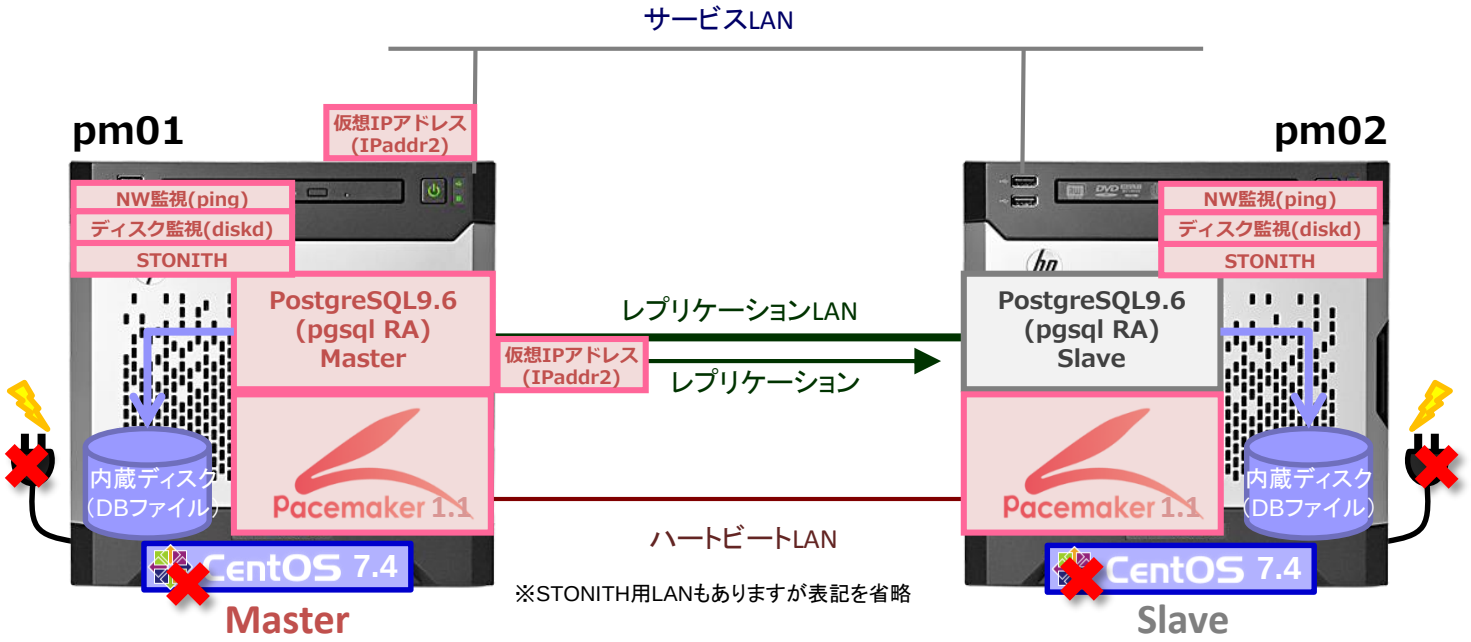
大項目	小項目	Pacemakerの動作	故障再現方法(コマンド)
ノード故障	カーネルハング	③	# echo c > /proc/sysrq-trigger
	電源停止	③	# poweroff –nf

実例紹介

割愛(上と同様↑)

カーネルの故障やコンセント抜け、電源ユニット故障等により、
ノード全体(OS)の動作が停止してしまうパターン

相手ノードの状態が急にわからなくなるため、リソースの2重起動を防ぐため
STONITHが発動する



ノード故障時の動作

✓ 故障動作例：ノード故障

処理の流れ

ノード1(Master)	ノード2(Slave)
正常稼働中・・・	正常稼働中・・・
1.カーネルハング(疑似故障)	
# echo c > /proc/sysrq-trigger	2.対向ノードの故障検知
	3.対向ノードからのSTONITH実行待ち
	4.STONITH実行
	5.PostgreSQL promote
	6.vip-master起動
	7.vip-rep起動

- 対向ノードを故障と検知するまでの時間はcorosync.confに設定の「token値+consensus値 [msec]」
  consensus値は未設定の場合token値の1.2倍に自動設定される

ノード故障時の動作

✓ノード故障時の監視画面(正常時との差分箇所を色文字で記載)

故障前

```
Online: [ pm01 pm02 ]

~~
 Resource Group: master-group
     vip-master    (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep       (ocf::heartbeat:IPaddr2):    Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline       : 0000000019000520
    + pgsql-status                    : PRI
    + ringnumber_0                  : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                    : 100
    + pgsql-data-status               : STREAMING|SYNC
    + pgsql-status                    : HS:sync
    + ringnumber_0                  : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

故障後

```
Online: [ pm02 ]
OFFLINE: [ pm01 ]
~~
 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):    Started pm02
     vip-rep       (ocf::heartbeat:IPaddr2):    Started pm02
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm02 ]
     Stopped: [ pm01 ]
~~
Node Attributes:

* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status_internal   : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline       : 000000001B000060
    + pgsql-status                    : PRI
    + ringnumber_0                  : 192.168.11.32 is UP

Migration Summary:
* Node pm02:
```

pm01がOFFLINE(クラスタ未参加)に

リソースがpm02で起動/Masterになる

pm01の属性値表示は無くなる
※故障前と比較しやすいよう空白としているが、実際には空白はない

pm02のpgsqlがMasterになる(pgsql-status属性値がPRIに)

ノード故障時の動作

✓ノード故障時のログ

pm_logconv.outより抜粋(Slave側)　※ノード故障では、ほとんどの場合故障した側のログは消失する

```
Jan 9 13:41:50 pm02    info: Unset DC node pm01.
Jan 9 13:41:50 pm02 warning: Node pm01 is lost
```

対向ノード(pm01)の故障検知

```
Jan 9 13:41:50 pm02    info: Set DC node to pm02.
Jan 9 13:41:51 pm02   error: Start to fail-over.
Jan 9 13:41:51 pm02    info: Try to STONITH (reboot) pm01.
Jan 9 13:41:51 pm02    info: Try to execute STONITH device prmStonith1-1 on pm02 for reboot pm01.
Jan 9 13:42:05 pm02 warning: Failed to execute STONITH device prmStonith1-1 for pm01.
```

stonith-helper稼働

```
Jan 9 13:42:05 pm02    info: Try to execute STONITH device prmStonith1-2 on pm02 for reboot pm01.
Jan 9 13:42:07 pm02    info: Succeeded to execute STONITH device prmStonith1-2 for pm01.
Jan 9 13:42:07 pm02    info: Succeeded to STONITH (reboot) pm01 by pm02.
```

pm02から対向ノード(pm01)へSTONITH実行

```
～(略)～
Jan 9 13:42:07 pm02    info: Resource pgsql tries to promote.
～(略)～
Jan 9 13:42:09 pm02    info: Resource pgsql promoted. (rc=0) ok
～(略)～
Jan 9 13:42:09 pm02    info: Resource vip-master : Started on pm02
Jan 9 13:42:09 pm02    info: Resource vip-rep : Started on pm02
Jan 9 13:42:09 pm02    info: fail-over succeeded.
```

promoteの流れは他の故障と同じ

ノード故障の復旧方法

✓ 手順

1.故障(STONITHで停止/再起動)したノードの起動禁止フラグ
「/var/lib/pgsql/tmp/PGSQL.lock」を削除

2.故障したノードをSlaveとして初期構築と同様に組み込む

```
# pg-rex_slave_start
```

3.(任意)故障したノードを再度Masterにしたい場合、手動でF.Oを発生させる

```
# pg-rex_switchover
```

Pacemakerプロセス故障

ノード故障について

大項目	小項目	Pacemakerの動作	故障再現方法(コマンド)	
Pacemakerプロセス故障	pacemakerdプロセス故障	①	# pkill -9 pacemakerd	実例紹介
	corosyncプロセス故障	③	# pkill -9 corosync	割愛
	cibプロセス故障	③	# pkill -9 cib	割愛(上と同様↑)

Pacemakerは以下に挙げる複数のプロセスが連携して動作している
「corosync」「pacemakerd」「cib」は代表的なプロセス
プロセス毎に故障した際の動作が異なる

Pacemaker-1.1関連プロセス一覧

- corosync
- cib
- stonithd
- lrmd
- attrd
- pengine
- crmd
- pacemakerd
- ifcheckd
- pm_logconv.py

故障時③の動作

故障時①の動作

「corosync」や「cib」の場合、発動するメカニズムは前述の「ノード故障」とほぼ同じ
(なので、故障時の動作は割愛)
「pacemakerd」の場合、特にF.O等は発生せず、プロセスが再起動する

プロセス故障時の動作

✓ 故障動作例：pacemakerd故障

- pacemakerdはPacemakerの関連プロセスを起動/監視する役割を担う
  仮にpacemakerdがダウンしてもPacemakerの稼働上は即座に影響はないため「再起動」するような設定としている(監視が途切れるが、再起動後に監視は再開される)

- 設定は「/etc/systemd/system/pacemaker.service」で行う

  設定箇所抜粋

```
Restart=on-failure
```

プロセス故障時の動作

✓ pacemakerd故障時の監視画面

故障前

```
Online: [ pm01 pm02 ]

~~
 Resource Group: master-group
     vip-master     (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep        (ocf::heartbeat:IPaddr2):    Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set              : 100
    + diskcheck_status_internal     : normal
    + master-pgsql                  : 1000
    + pgsql-data-status             : LATEST
    + pgsql-master-baseline         : 000000001E000288
    + pgsql-status                  : PRI
    + ringnumber_0                  : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set              : 100
    + diskcheck_status_internal     : normal
    + master-pgsql                  : 100
    + pgsql-data-status             : STREAMING|SYNC
    + pgsql-status                  : HS:sync
    + ringnumber_0                  : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

故障後

```
Online: [ pm01 pm02 ]

~~
 Resource Group: master-group
     vip-master     (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep        (ocf::heartbeat:IPaddr2):    Started pm01
~~
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
~~
Node Attributes:
* Node pm01:
    + default_ping_set              : 100
    + diskcheck_status_internal     : normal
    + master-pgsql                  : 1000
    + pgsql-data-status             : LATEST
    + pgsql-master-baseline         : 000000001E000288
    + pgsql-status                  : PRI
    + ringnumber_0                  : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set              : 100
    + diskcheck_status_internal     : normal
    + master-pgsql                  : 100
    + pgsql-data-status             : STREAMING|SYNC
    + pgsql-status                  : HS:sync
    + ringnumber_0                  : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

故障前からの変化はなし

プロセス故障時の動作

✓ pacemakerd故障時のログ

pm_logconv.outより抜粋(Master側)

```
Jan 9 15:39:49 pm01    info: Starting Pacemaker 1.1.16.
```

Pacemakerが再起動した旨が出力される

Pacemakerのログ(ha-logおよびpm_logconv.out)には、pacemakerdプロセスがダウンした旨を直接的に示すようなログは出力されないため一見して事象がわかりづらいが、/var/log/messagesにはもう少しわかりやすいログが出力される

messagesより抜粋(Master側)

```
Jan  9 15:39:49 pm01 systemd: pacemaker.service failed.
Jan  9 15:39:49 pm01 systemd: pacemaker.service holdoff time over, scheduling restart.
Jan  9 15:39:49 pm01 systemd: Started Pacemaker High Availability Cluster Manager.
Jan  9 15:39:49 pm01 systemd: Starting Pacemaker High Availability Cluster Manager...
```

このように、Pacemakerのログをメインとして、messagesや故障したアプリケーション(＝リソース)等のログも併せて参照したほうがより効率的に事象把握が行える

プロセス故障の復旧方法

✓ 手順

- 自動で再起動するため、特に復旧する必要はなし

- プロセス故障が頻発する場合は、真の原因（メモリ不足？、設定ミス？　etc..）を探すとよい
  - 難しい場合は、遠慮なくLinux-HA Japan MLへ質問してください！
    (監視画面の表示結果とログの添付もお願いします)

ディスク故障

ディスク故障について

大項目	小項目	Pacemakerの動作	故障再現方法(コマンド)
ディスク故障	内蔵ディスク故障	② or ③	ディスクを引っこ抜く！

ディスクはディスク監視機能(diskd)により監視される

OSのコマンドが配置された内蔵ディスクが故障した場合、リソース停止に使うコマンド(RA自体やpsql、umount、ipコマンド etc...)が使用できる場合と、そうでない場合があり
コマンドが使用できる場合②の動作、できない場合は③の動作となる

補足:
OSのディスクキャッシュにコマンド（のバイナリファイル）が存在する場合、ディスクが故障してもコマンドが実行できます。が、Linuxの場合キャッシュはメモリ残量に応じて削除されたりもするので、コマンドが使用できたり、できなかったりします。

②の動作の場合、前述の「サービスLAN故障」の動作とほぼ同じ、
③の動作の場合、後述の「リソース停止失敗」の動作とほぼ同じ、
となるため、いずれについても故障時の動作は割愛

リソース停止失敗

リソース停止失敗について

大項目	小項目	Pacemakerの動作	故障再現方法(コマンド)	
リソース停止失敗	vip-rep	②	RAのstopメソッドを exit $OCF_ERR_GENERICに書き換え	割愛
	vip-master	③	RAのstopメソッドを exit $OCF_ERR_GENERICに書き換え	割愛(下と同様↓)
	PostgreSQL(demote失敗)	③	RAのdemoteメソッドを exit $OCF_ERR_GENERICに書き換え	割愛(下と同様↓)
	PostgreSQL(stop失敗)	③	RAのstopメソッドを exit $OCF_ERR_GENERICに書き換え	実例紹介

スイッチオーバやフェイルオーバ等、リソースの停止を伴うオペレーション中に、
停止処理に失敗するケース
リソースの停止ができないと、クラスタとしては2重起動が懸念されるため、
リソースを確実に停止する手段としてSTONITHを発動させる(＝③の動作)

vip-repだけは②の動作になっているが、これはvip-repはSlave側のPostgreSQLだけがアクセスするアドレスで、フェイルオーバ後は使用されず停止に失敗しても問題が無いので、停止失敗しても無視するよう設定してあるため

```
primitive vip-rep ocf:heartbeat:IPaddr2_stop_error ¥
        params ip="192.168.2.30" nic="eno3" cidr_netmask="24" ¥
        meta migration-threshold="0" ¥
        op start interval="0s" timeout="60s" on-fail="stop" ¥
        op monitor interval="10s" timeout="60s" on-fail="restart" ¥
        op stop interval="0s" timeout="60s" on-fail="ignore"
```

on-failは当該オペレーション(start/stop/monitor)が失敗した場合にどうするかを決める設定
ignoreだと失敗は無視し動作を継続する
他リソースはfence(STONITH実行)に設定

リソース停止失敗時の動作

✓ 故障動作例：PostgreSQL停止失敗

処理の流れ

ノード1(Master)	ノード2(Slave)
正常稼働中・・・	正常稼働中・・・
1.PostgreSQL強制停止(疑似故障)	
2.PostgreSQL demote	
4.vip-rep停止	
5.vip-master停止	
6.PostgreSQL停止(停止失敗)	
	7.対向ノードからのSTONITH実行待ち
	8.STONITH実行
	9.PostgreSQL promote
	10.vip-master起動
	11.vip-rep起動

事前にpgsql RAのstopメソッドに
exit $OCF_ERR_GENERICを仕込む

```
# pg_ctl -m i stop
```

リソース停止失敗時の動作

✓ PostgreSQL停止失敗時の監視画面(正常時との差分箇所を色文字で記載)

故障前

```
Online: [ pm01 pm02 ]
～～
 Resource Group: master-group
     vip-master    (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep       (ocf::heartbeat:IPaddr2):    Started pm01
～～
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
～～
Node Attributes:
* Node pm01:
    + default_ping_set                : 100
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline           : 000000003A000288
    + pgsql-status                    : PRI
    + ringnumber_0                    : 192.168.11.31 is UP
* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status                : normal
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 100
    + pgsql-data-status               : STREAMING|SYNC
    + pgsql-status                    : HS:sync
    + ringnumber_0                    : 192.168.11.32 is UP

Migration Summary:
* Node pm01:
* Node pm02:
```

故障後

```
Online: [ pm02 ]
OFFLINE: [ pm01 ]
～～
 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):    Started pm02
     vip-rep    (ocf::heartbeat:IPaddr2):    Started pm02
～～
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ pm02 ]
     Stopped: [ pm01 ]
～～
Node Attributes:

* Node pm02:
    + default_ping_set                : 100
    + diskcheck_status_internal       : normal
    + master-pgsql                    : 1000
    + pgsql-data-status               : LATEST
    + pgsql-master-baseline           : 000000003A000398
    + pgsql-status                    : PRI
    + ringnumber_0                    : 192.168.11.32 is UP

Migration Summary:
* Node pm02:
```

pm01がOFFLINE(クラスタ未参加)に

リソースがpm02で起動/Masterになる

pm01の属性値表示は無くなる
※故障前と比較しやすいよう空白としているが、実際には空白はない

pm02のpgsqlがMasterになる(pgsql-status属性値がPRIに)

リソース停止失敗時の動作

✓PostgreSQL停止失敗時のログ

pm_logconv.outより抜粋(Slave側)　(Master側のpm_logconv.outはSTONITHにより未出力)

```
Jan 18 19:16:45 pm02    info: Try to execute STONITH device prmStonith1-1 on pm02 for reboot pm01.
Jan 18 19:16:49 pm02 warning: Failed to execute STONITH device prmStonith1-1 for pm01.
```

stonith-helper稼働

```
Jan 18 19:16:49 pm02    info: Try to execute STONITH device prmStonith1-2 on pm02 for reboot pm01.
Jan 18 19:16:52 pm02    info: Unset DC node pm01.
Jan 18 19:16:52 pm02 warning: Node pm01 is lost
Jan 18 19:16:52 pm02    info: Succeeded to execute STONITH device prmStonith1-2 for pm01.
Jan 18 19:16:52 pm02    info: Set DC node to pm02.
Jan 18 19:16:52 pm02    info: Succeeded to STONITH (reboot) pm01 by pm02.
```

pm02から対向ノード(pm01)へSTONITH実行

```
Jan 18 19:16:52 pm02   error: Start to fail-over.
Jan 18 19:16:52 pm02    info: Resource pgsql tries to promote.
～(略)～
Jan 18 19:16:54 pm02    info: Resource pgsql promoted. (rc=0) ok
Jan 18 19:16:54 pm02    info: Resource vip-master tries to start.
Jan 18 19:16:54 pm02    info: Resource vip-master started. (rc=0) ok
Jan 18 19:16:54 pm02    info: Resource vip-rep tries to start.
Jan 18 19:16:54 pm02    info: Resource vip-rep started. (rc=0) ok
Jan 18 19:16:54 pm02    info: Resource vip-master : Started on pm02
Jan 18 19:16:54 pm02    info: Resource vip-rep : Started on pm02
Jan 18 19:16:54 pm02    info: fail-over succeeded.
```

F.Oの流れは他の故障と同じ

リソース停止失敗時の動作

✓ PostgreSQL停止失敗時のログ

STONITHにより、故障時間付近はha-logからpm_logconv.outへの変換ログが出力されない場合もあるため、必要に応じてha-logも参照するとよい

ha-logより抜粋(Master側)

```
Jan 18 19:16:44 pm01 pgsql(pgsql)[2569]: INFO: PostgreSQL is down
～(略)～
```

PostgreSQL故障検知

```
Jan 18 19:16:44 pm01 crmd[21057]:  notice: Result of demote operation for pgsql on pm01: 0 (ok)
～(略)～
```

PostgreSQL demote

```
Jan 18 19:16:45 pm01 crmd[21057]:  notice: Result of stop operation for vip-rep on pm01: 0 (ok)
～(略)～
```

vip-rep停止

```
Jan 18 19:16:45 pm01 crmd[21057]:  notice: Result of stop operation for vip-master on pm01: 0 (ok)
～(略)～
```

vip-master停止

```
Jan 18 19:16:45 pm01 crmd[21057]:  notice: Result of stop operation for pgsql on pm01: 1 (unknown error)
```

PostgreSQL停止失敗

リソース(PostgreSQL)停止失敗の復旧方法

✓ 手順

1.故障(STONITHで停止/再起動)したノードの起動禁止フラグ
「/var/lib/pgsql/tmp/PGSQL.lock」を削除

2.故障したノードをSlaveとして初期構築と同様に組み込む

```
# pg-rex_slave_start
```

3.(任意)故障したノードを再度Masterにしたい場合、手動でF.Oを発生させる

```
# pg-rex_switchover
```