試して覚える Pacemaker-2.0入門 『構築・リソース設定』

2020/10/23 OSC2020 Online/Fall

Linux-HA Japan プロジェクト

三浦　貴紀

自己紹介

三浦　貴紀(みうら　たかのり)

OSCがオンライン開催になってから初参加です

普段はNTT OSSセンタという所で
高信頼技術をやってます。

目次

①Pacemakerの概要

②Pacemaker-2.0で変わったこと

③構築・リソース設定の紹介

目次

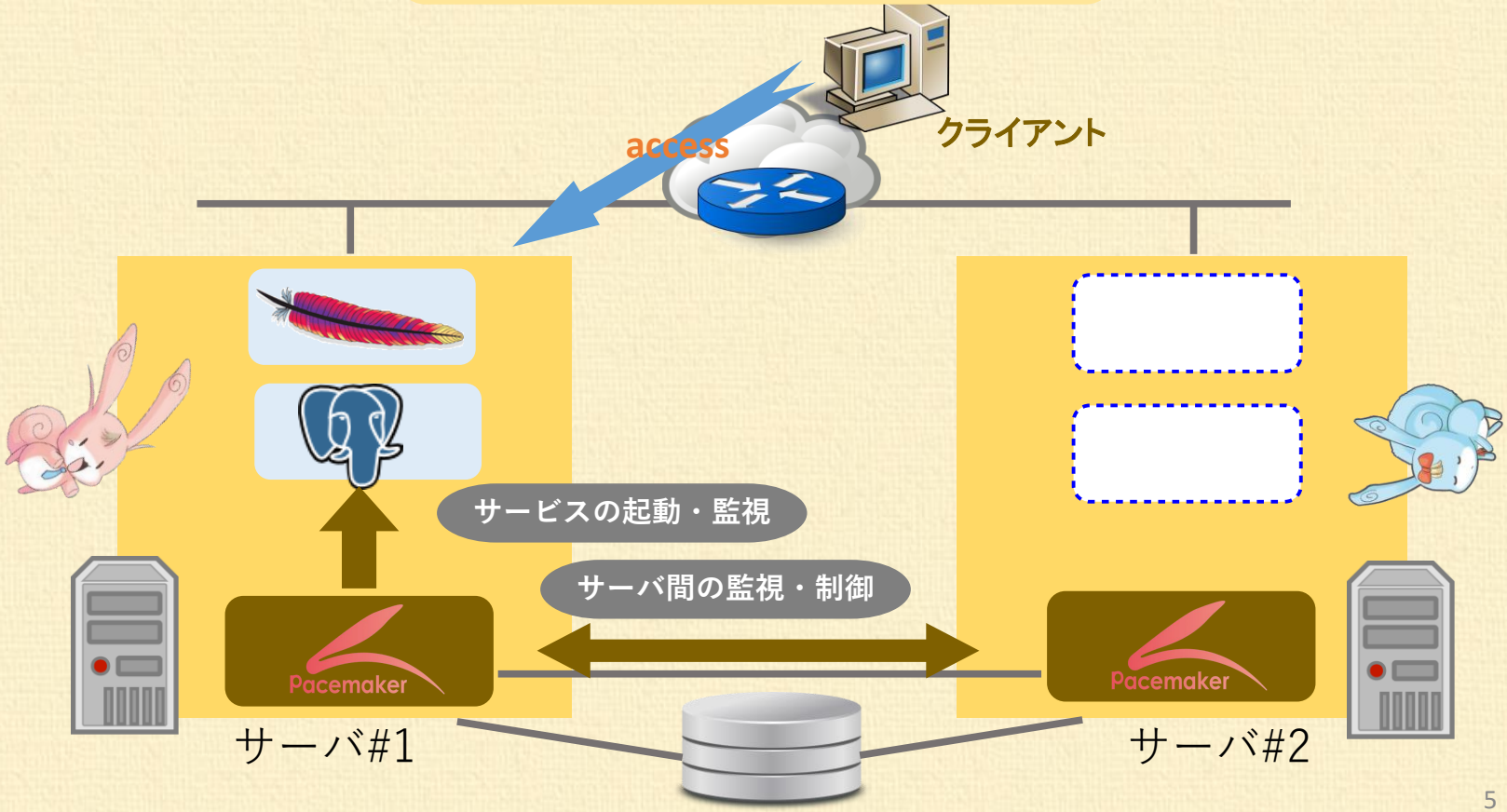
①Pacemakerの概要

②Pacemaker-2.0で変わったこと

③構築・リソース設定の紹介

Pacemakerの概要(1/2)

サーバ・アプリの故障監視

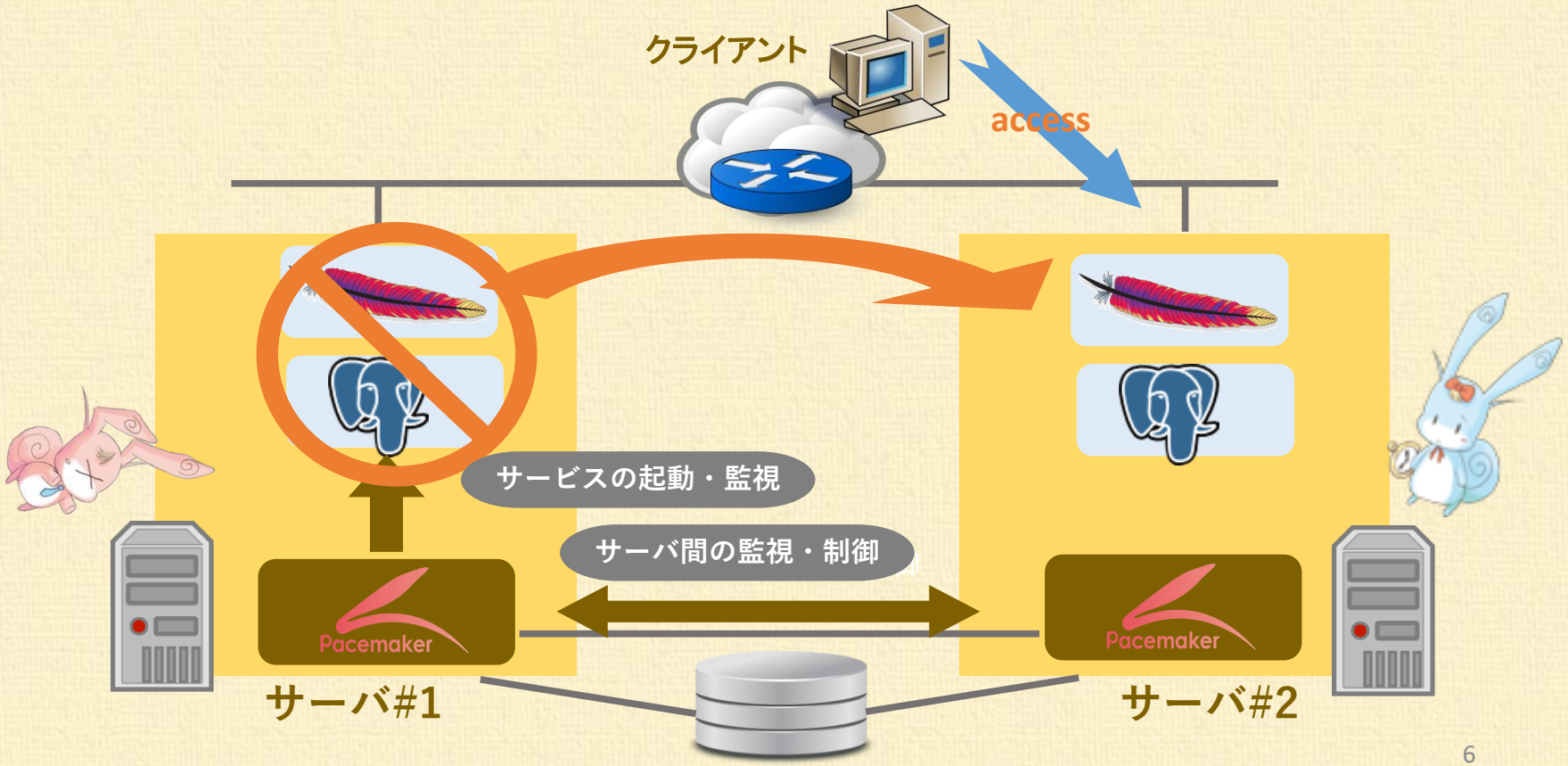


Pacemakerの概要(2/2)

故障検知時に自動的にフェイルオーバ

ダウンタイムの最小化

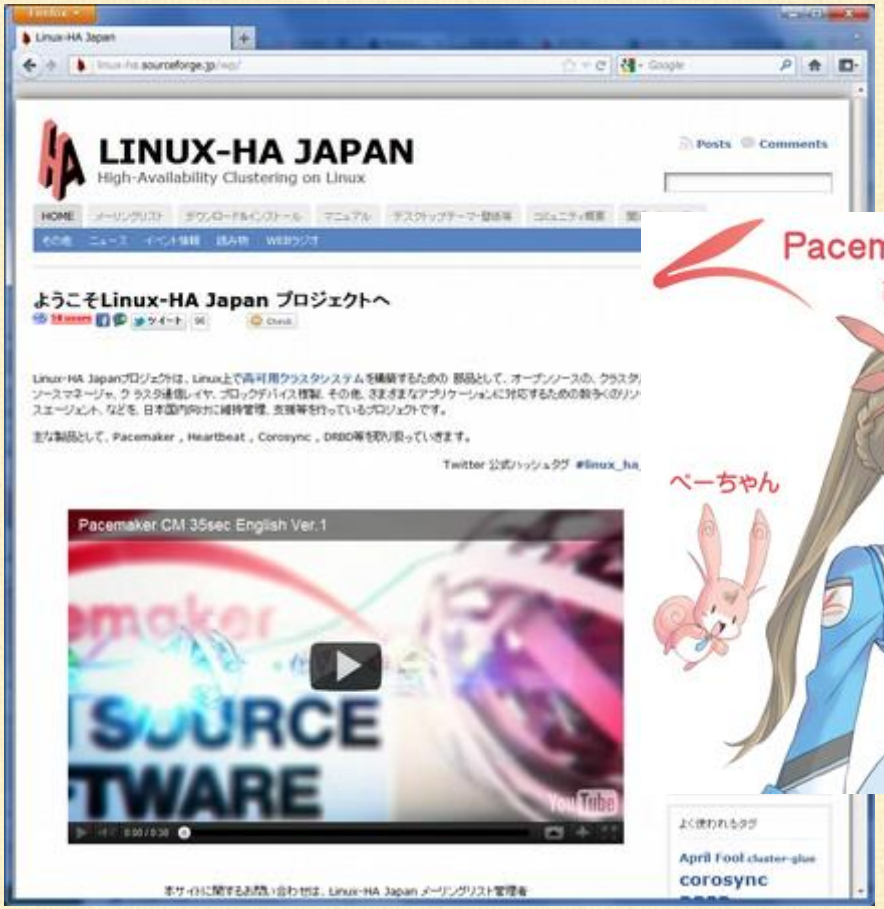
STONITH※によるデータの安全性確保



※ スプリットブレイン、F/Oにおけるリソース停止失敗時に対向ノードを強制電源断する機能

もっと知りたい方はこちらまで

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



目次

①Pacemakerの概要

②Pacemaker-2.0で変わったこと

③構築・リソース設定の紹介

Pacemaker-2.0では特にここが変わりました

✓パッケージ提供

これまで(Pacemaker-1.1)はLinux-HA Japanからパッケージを提供していましたが、Pacemaker-2.0からはRHEL/CentOS 8 High Availability Add-On(以下、HA Add-On)の利用を推奨します。

- 利用手順・設定もRed Hatの方針に近くなります

✓構築・リソース設定

Pacemaker-1.1ではクラスタ管理ツールにcrmの利用を推奨していましたが、Pacemaker-2.0からはpcsに変更します。

- 構築・リソース設定が割と変わります
- ただできるだけ今まで通りに使えるようLinux-HA Japanから役立つツールを提供しています！

✓STONITH必須

Pacemaker-2.0ではSTONITH有構成のみを推奨します。

- 共有ディスクを介した代替STONITH方式等も利用できます。

Pacemaker-2.0では特にここが変わりました

✓パッケージ提供

これまで(Pacemaker-1.1)はLinux-HA Japanからパッケージを提供していましたが、Pacemaker-2.0からはRHEL/CentOS 8 High Availability Add-On(以下、HA Add-On)の利用を推奨します。

- 利用手順・設定もRed Hatの方針に近くなります

✓構築・リソース設定

Pacemaker-1.1ではクラスタ管理ツールにcrmの利用を推奨していましたが、Pacemaker-2.0からはpcsに変更します。

- 構築・リソース設定が割と変わります
- ただできるだけ今まで通りに使えるようLinux-HA Japanから役立つツールを提供しています！

✓STONITH必須

Pacemaker-2.0ではSTONITH有構成のみを推奨

- 共有ディスクを介した代替STONITH方式等も利用

ここからは
ここを中心に話します

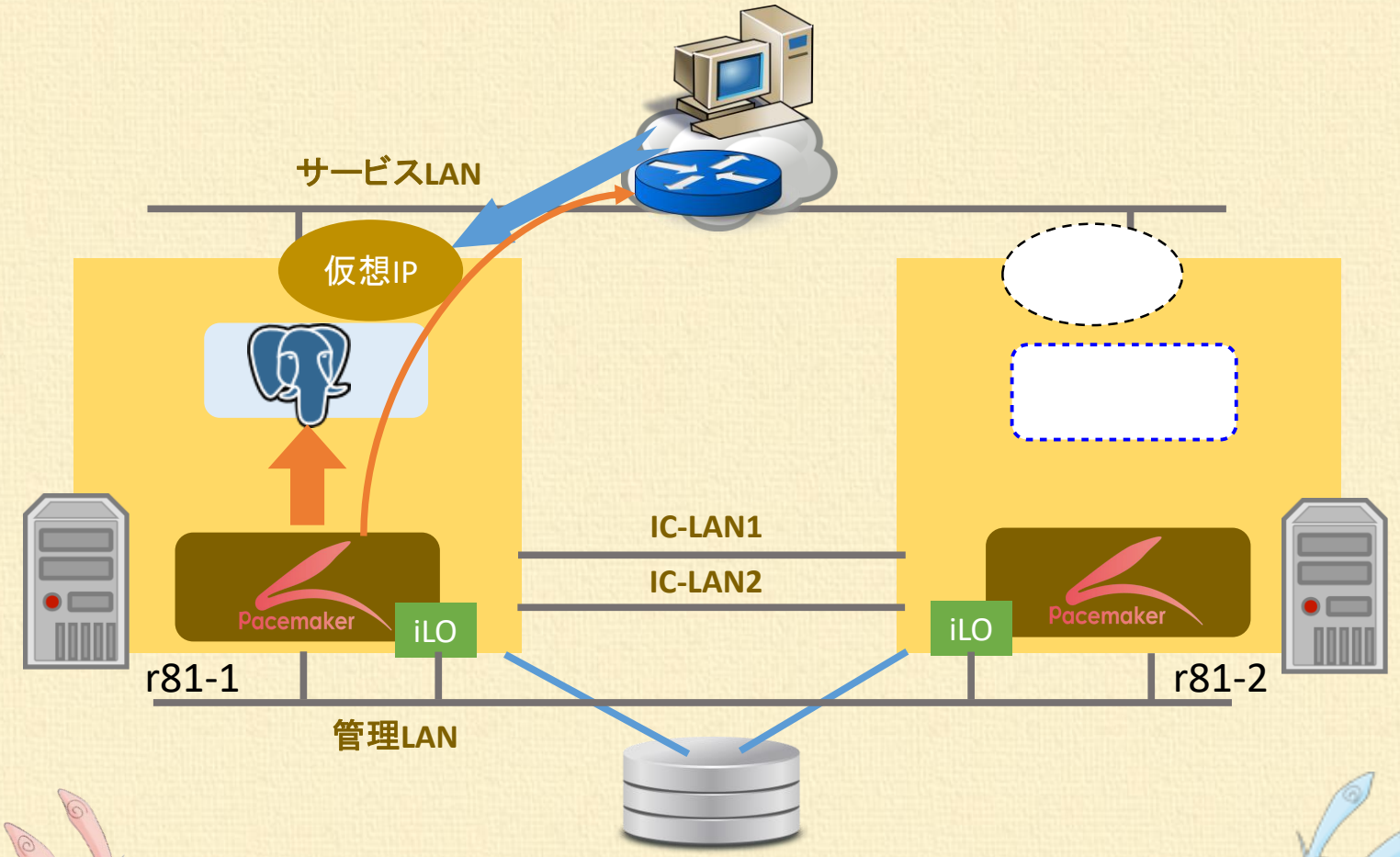
目次

①Pacemakerの概要

②Pacemaker-2.0で変わったこと

③構築・リソース設定の紹介

今日はこんな感じのクラスタを作る手順を紹介します



OSやPostgreSQLのインストールは済んでいるものとします

r81-1に故障が発生したらr81-2へフェイルオーバさせます

目次

①Pacemakerの概要

②Pacemaker-2.0で変わったこと

③構築・リソース設定の紹介

先ずは構築から

構築手順(1/2)

1. **インストール**　■両系で

```
# dnf install pcs pacemaker fence-agents-all –enablerepo=HighAvailability
```

2. **クラスタが利用するポートの許可**　■両系で

```
# firewall-cmd –permanent –add-service=high-availability
# firewall-cmd –add-service=high-availability
```

3. **pcsdサービスの起動**　■両系で

```
# systemctl start pcsd.service
# systemctl enable pcsd.service
```

4. **hacluster ユーザのパスワード設定**　■両系で

```
# passwd hacluster
Changing password for user hacluster.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

構築手順(2/2)

5.　クラスタノードの認証設定　★いずれか一方のノードで

```
# pcs host auth r81-1 addr= <r81-1の管理LAN> r81-2 addr= <r81-2の管理LAN>
Username: hacluster
Password:
r81-1: Authorized
r81-2: Authorized
```

6.　クラスタの作成　●5を実行したノードで

```
# pcs cluster setup ＜クラスタ名(任意)＞ r81-1 addr=<r81-1のIC-LAN1> addr=<r81-1のIC-LAN2> r81-2 addr=<r81-2のIC-LAN1> addr=<r81-2のIC-LAN2>
```

今まで” corosync-keygen -l”で作った認証ファイルを手動で各ノードへ配置していたのが「4. クラスタノードの認証設定」に変わりました

今まで”corosync.conf”を手動で作成していたのが「5. クラスタの作成」に代わりました

目次

①Pacemakerの概要

②Pacemaker-2.0で変わったこと

③構築・リソース設定の紹介

次はこっち

リソース設定の流れ(1/3)

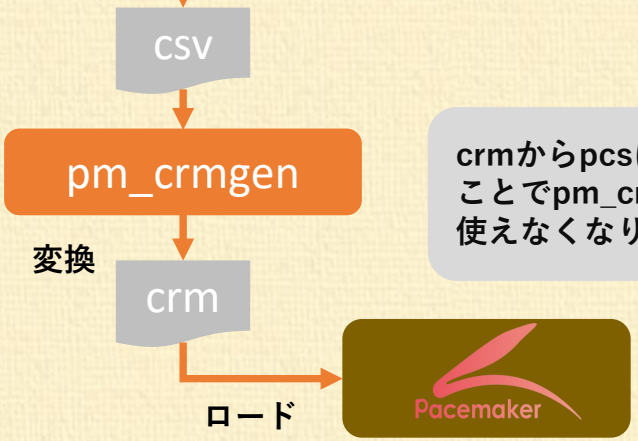
これまで(Pacemaker-1.1系)リソース設定はExcel形式の環境定義書を変換してPacemakerへロードしていました。

350行くらい

#表 4-1 クラスタ設定 … リソース構成

RESOURCES

#	resourceItem	resourceItem	resourceItem	id	
	リソース構成要素			リソースID	概要
	Group			grpPostgreSQLDB	グループ定義
		Primitive		prmExPostgreSQLDB	共有ディスク排他制御
		Primitive		prmFsPostgreSQLDB1	DBクラスタ領域
		Primitive		prmFsPostgreSQLDB2	WALログ領域
		Primitive		prmFsPostgreSQLDB3	アーカイブログ領域
		Primitive		prmIpPostgreSQLDB	DB仮想IP割当
		Primitive		prmApPostgreSQLDB	PostgreSQL制御
	Clone			clnPing	クローン定義
		Primitive		prmPing	ネットワーク監視



今まではLinux-HA Japanから出していたpm_crmgenがあったのでExcel形式で環境定義書を作れました

crmからpcsになったことでpm_crmgenが使えなくなります。

リソース設定の流れ(2/3)

Red Hatのお作法に従いpcsで構築する際には、これらのpcsコマンドを一から作成しなければなりません。

```
# pcs resource defaults resource-stickiness=200
# pcs resource defaults migration-threshold=1
# pcs resource create filesystem1 ocf:heartbeat:Filesystem device="/dev/mapper/mpatha2" directory="/dbfp/pgdata" fstype="xfs" force_unmount="safe" op start
timeout=60s on-fail=restart monitor timeout=60s interval=10s on-fail=restart stop timeout=60s on-fail=fence
# pcs resource create filesystem2 ocf:heartbeat:Filesystem device="/dev/mapper/mpatha3" directory="/dbfp/pgwal" fstype="xfs" force_unmount="safe" op start
timeout=60s on-fail=restart monitor timeout=60s interval=10s on-fail=restart stop timeout=60s on-fail=fence
# pcs resource create filesystem3 ocf:heartbeat:Filesystem device="/dev/mapper/mpatha4" directory="/dbfp/pgarch" fstype="xfs" force_unmount="safe" op start
timeout=60s on-fail=restart monitor timeout=60s interval=10s on-fail=restart stop timeout=60s on-fail=fence
# pcs resource create ipaddr ocf:heartbeat:IPaddr2 ip="192.168.1.87" nic="ens4" cidr_netmask="24" op start timeout=60s on-fail=restart monitor timeout=60s interval=10s
on-fail=restart stop timeout=60s on-fail=fence
# pcs resource create pgsql ocf:linuxhajp:pgsql pgctl="/usr/pgsql-11/bin/pg_ctl" psql="/usr/pgsql-11/bin/psql" pgdata="/dbfp/pgdata/data" pgdba="postgres"
pgport="5432" pgdb="template1" op start timeout=300s on-fail=restart monitor timeout=60s interval=10s on-fail=restart stop timeout=300s on-fail=fence
# pcs resource group add pgsql-group filesystem1 filesystem2 filesystem3 ipaddr pgsql
# pcs resource create ping ocf:pacemaker:ping name="ping-status" host_list="192.168.1.1" attempts="2" timeout="2" debug="true" op start timeout=60s on-fail=restart
monitor timeout=60s interval=10s on-fail=restart stop timeout=60s on-fail=fence
# pcs resource clone ping
# pcs stonith create fence1-ipmilan fence_ipmilan delay="10" pcmk_host_list="r81-1" ip="192.168.2.85" username="root" password="XXXXXX" lanplus="1"
ipmitool_path="/usr/bin/ipmitool_stub" op start timeout=60s on-fail=restart monitor timeout=60s interval=3600s on-fail=restart stop timeout=60s on-fail=ignore
# pcs stonith create fence2-ipmilan fence_ipmilan delay="0" pcmk_host_list="r81-2" ip="192.168.2.86" username="root" password="XXXXXX" lanplus="1"
ipmitool_path="/usr/bin/ipmitool_stub" op start timeout=60s on-fail=restart monitor timeout=60s interval=3600s on-fail=restart stop timeout=60s on-fail=ignore
# pcs constraint location pgsql-group prefers r81-1=200
# pcs constraint location pgsql-group prefers r81-2=100
# pcs constraint location fence1-ipmilan avoids r81-1
# pcs constraint location fence2-ipmilan avoids r81-2
# pcs constraint location pgsql-group rule score=-INFINITY ping-status lt 1 or not_defined ping-status
# pcs constraint colocation add pgsql-group with ping-clone score=INFINITY
# pcs constraint order ping-clone then pgsql-group symmetrical=false
```

正直しんどい！なので

リソース設定の流れ(3/3)

Excel形式の環境定義書からpcsコマンドを生成するpm_pcsgenというツールを作りました！作成したpcsコマンドからクラスター設定ファイル(cib)を作成するところも自動で実施してくれます。Red Hatのお作法に従いつつも簡単に構築ができます！

350行くらい

#表 4-1 クラスタ設定 … リソース構成

RESOURCES

#	resourceItem リソース構成要素	resourceItem	resourceItem	id リソースID	概要
	Group			grpPostgreSQLDB	グループ定義
		Primitive		prmExPostgreSQLDB	共有ディスク排他制御
		Primitive		prmFsPostgreSQLDB1	DBクラスタ領域
		Primitive		prmFsPostgreSQLDB2	WALログ領域
		Primitive		prmFsPostgreSQLDB3	アーカイブログ領域
		Primitive		prmIpPostgreSQLDB	DB仮想IP割当
		Primitive		prmApPostgreSQLDB	PostgreSQL制御
	Clone			clnPing	クローン定義
		Primitive		prmPing	ネットワーク監視

csv

pm_pcsgen

pcsコマンド

cib

ロード

Pacemaker

pcsはしんどいという多くの方々の声にお応えしました

これで移行も楽ちん

pm_extra_toolsについて(1/2)

pm_pcsgenは**pm_extra_tools**というパッケージに
詰めてここに置いてます

http://linux-ha.osdn.jp/wp/archives/4963

pm_extra_toolsについて(2/2)

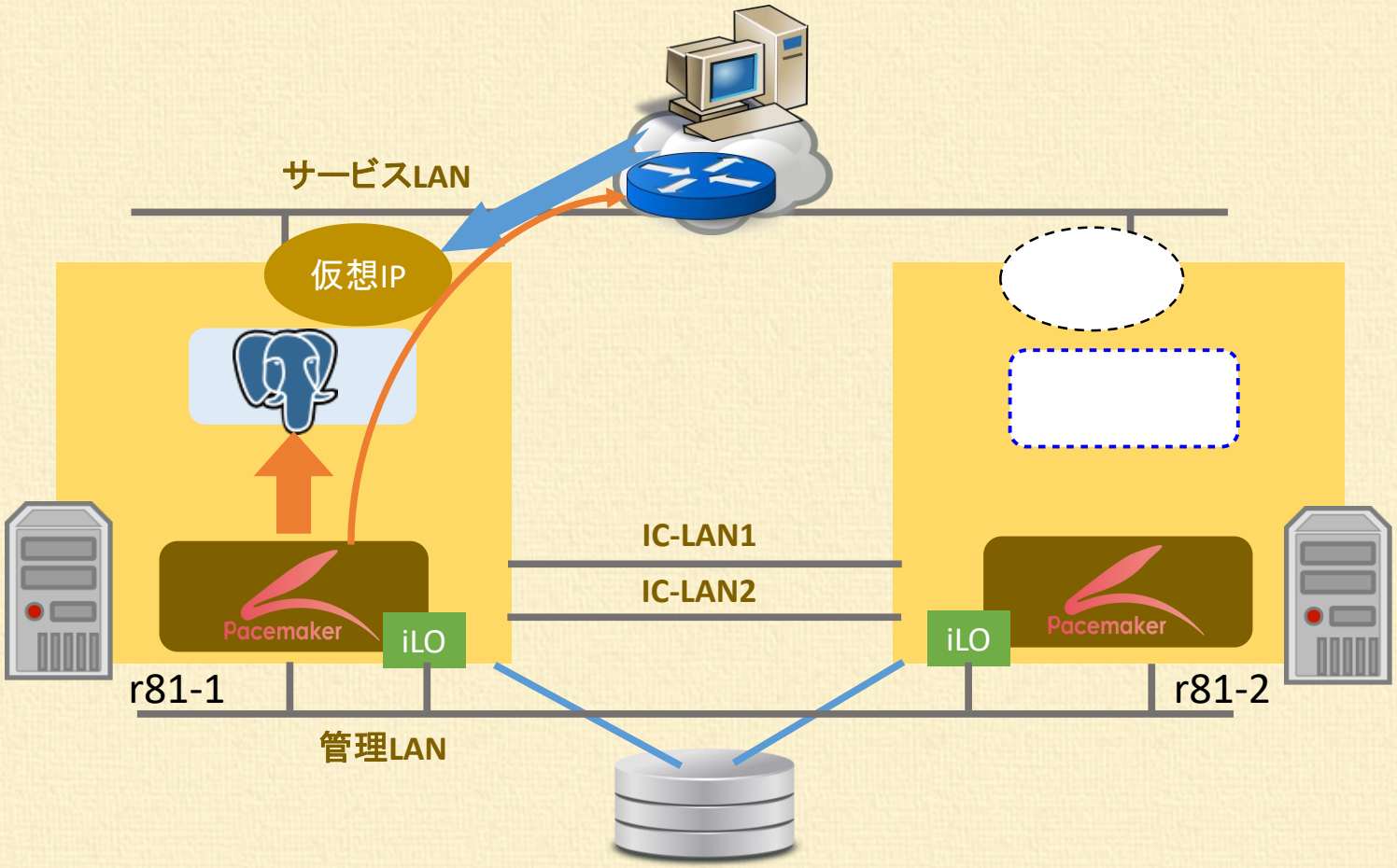
pm_extra_toolsにはpm_pcsgenの他にRA(リソースエージェントも入ってます)

1. **pgsql RA**
   HA Add-onにも含まれていますがそちらはPG-REXに対応していません。
   PG-REXやPostgreSQL(コミュニティ版)を考えているならこちらを使用してください。
2. **hulft RA**
   セゾン情報システムズ社のデータ連携ソフトウェアであるHULFT用のRAです。

ではリソース設定の話に戻ります

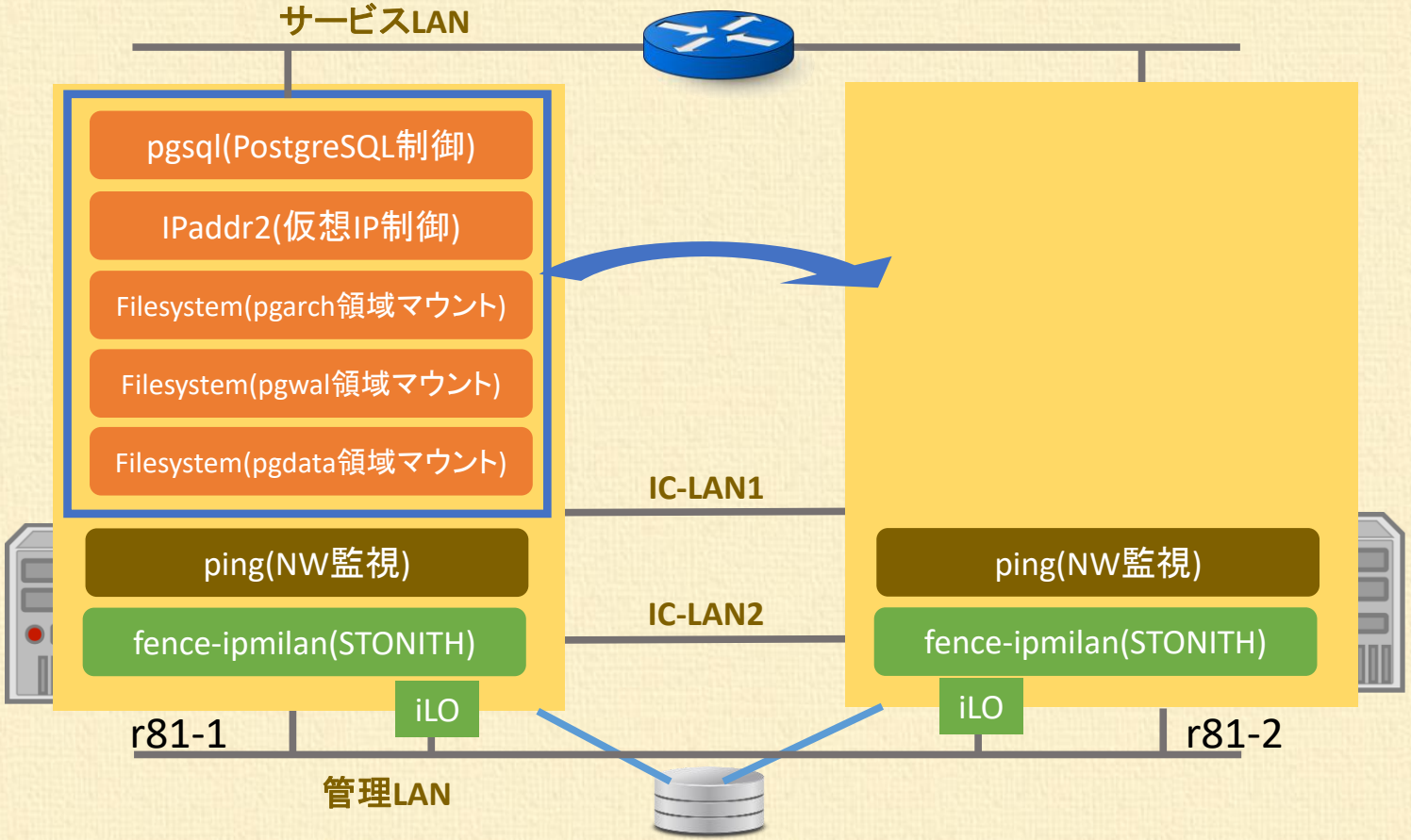
今まで通り、Excelでリソース設定ができると古参のユーザが安心できたところで、初回の方でもリソース設定ができるよう、ここからリソース設定の話に戻ります。

改めて今日はこんな感じのクラスタを作ります



リソース構成としてはこうします

- 共有ディスクマウント/仮想IP/PostgreSQLをどちらか一方(r81-1)で動作させます。これらは常にセットで動作させる必要があるため、グループ化して故障した際にはまとめてF/Oさせます。
- 両系でゲートウェイへのNW監視を行い、監視が失敗しているノードはActにさせません。
- IC-LANが全て切断された場合には、スプリットブレイン抑止のため、STONITHで対向ノードを電源断します。



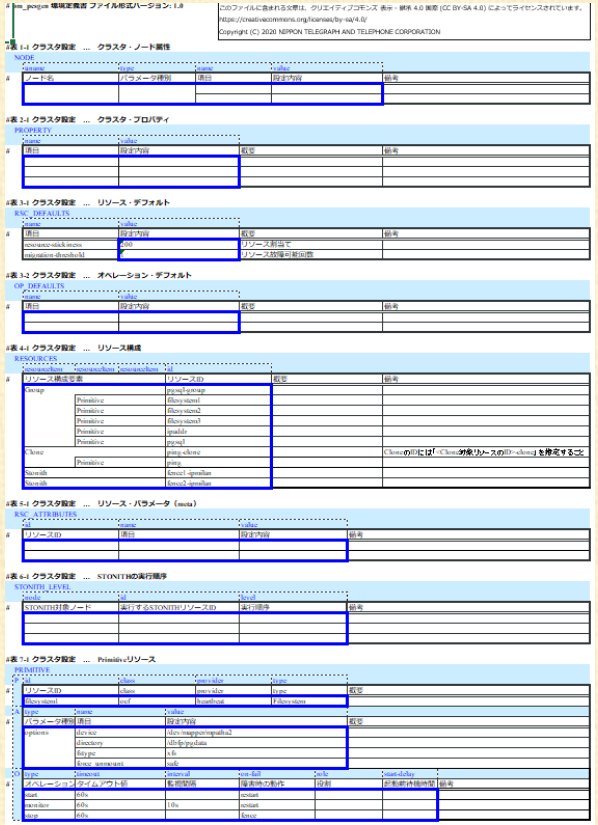
リソース設定~サンプル置き場

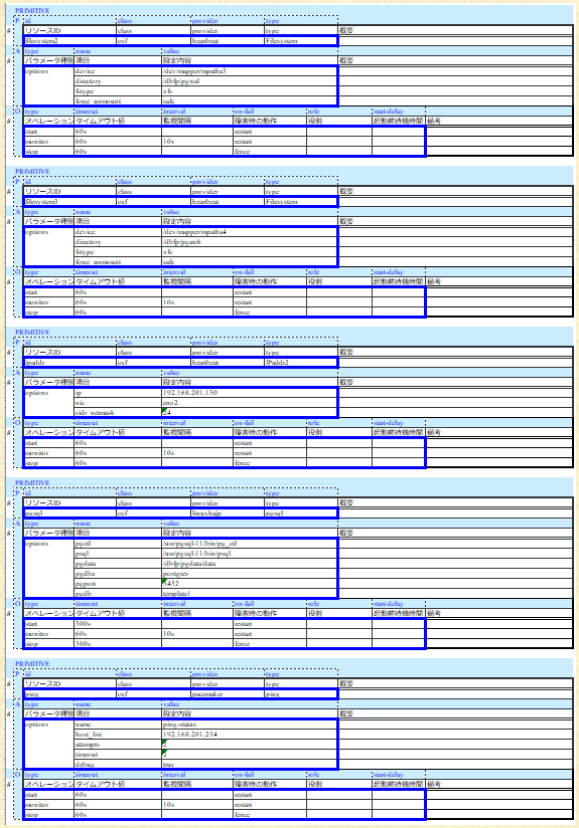
pm_extra_toolsをインストールすると下記にExcel形式の環境定義書のサンプルが配置されます。前頁のリソース構成を実現するリソース設定をこのサンプルを元に解説します。サンプルにはいくつか表がありますが、今回の構成に関わる表1~11を順に解説します。

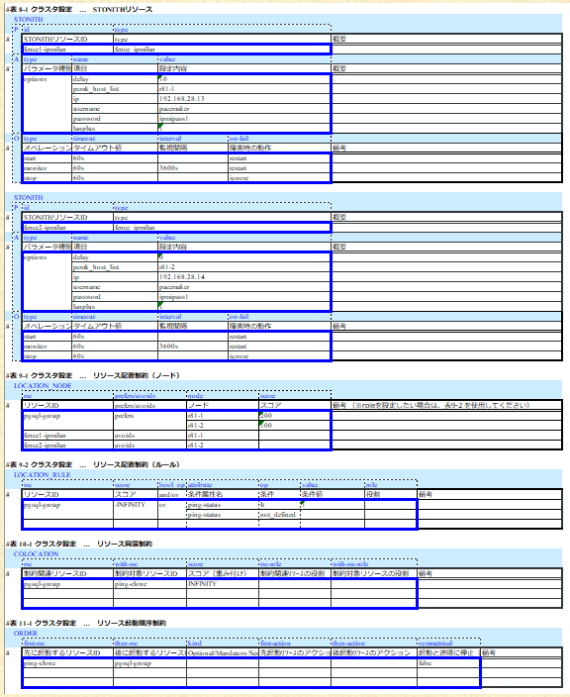
/usr/share/pm_extra_tools/pm_pcsgen_sample.xlsx

③構築・リソース設定の紹介

ちなみにサンプルの全体はこのぐらいあります
見えませんね💦







リソース設定解説(1/8)
Excelサンプル　表 1～3

ここにはクラスタ全体に関わる設定を記述します。
基本的には常にサンプル通りでよいです。

#表 1-1 クラスタ設定　...　クラスタ・ノード属性

NODE

	uname	type	name	value	
#	ノード名	パラメータ種別	項目	設定内容	備考

#表 2-1 クラスタ設定　...　クラスタ・プロパティ

PROPERTY

	name	value		
#	項目	設定内容	概要	備考

#表 3-1 クラスタ設定　...　リソース・デフォルト

RSC_DEFAULTS

	name	value		
#	項目	設定内容	概要	備考
	resource-stickiness	200	リソース割当て	
	migration-threshold	1	リソース故障可能回数	

#表 3-2 クラスタ設定　...　オペレーション・デフォルト

OP_DEFAULTS

	name	value		
#	項目	設定内容	概要	備考

今回は1回でも故障を検知したらF/Oさせます。「N回故障したらF/O」としたい場合は表3-1の”migration-threshold”にNを設定してください。

リソース設定解説(2/8)
Excelサンプル　表 4

設定するリソースの種別を定義します。

- 各リソースに任意の名前を付けリソースIDに指定します。リソースIDとRAの紐づけは表7で行います
- 基本的にリソースは全て“Primitive”として定義します。その上で“Group”に組み込んで1セットにしたり、”Clone”に組み込んで両系で動作させたりします。
- STONITHリソースは”Stonith”で定義します。

#表 4-1 クラスタ設定　…　リソース構成

RESOURCES

#	resourceItem	resourceItem	resourceItem	id	概要	備考
	リソース構成要素			リソースID		
	Group			pgsql-group		
		Primitive		filesystem1		
		Primitive		filesystem2		
		Primitive		filesystem3		
		Primitive		ipaddr		
		Primitive		pgsql		
	Clone			ping-clone		CloneのIDには「<Clone対象リソースのID>-clone」を指定すること
		Primitive		ping		
	Stonith			fence1-ipmilan		
	Stonith			fence2-ipmilan		

今回はpgsql-groupという名前のGroupにfilesystemからpgsqlまでのリソースを含めています。

備考にも書いていますがCloneの名前の付け方には注意です。Pacemaker-2.0で変わったところです。

リソース設定解説(3/8)
Excelサンプル　表 5～6

ここはもうサンプル通りです。すっ飛ばします。

#表 5-1 クラスタ設定　…　リソース・パラメータ（meta）

RSC_ATTRIBUTES

	id	name	value	
#	リソースID	項目	設定内容	備考

#表 6-1 クラスタ設定　…　STONITHの実行順序

STONITH_LEVEL

	node	id	level	
#	STONITH対象ノード	実行するSTONITHリソースID	実行順序	備考

Pacemaker-1.1の時はSTONITHリソースを複数使っていたので表6-1のSTONITHの実行順序を使っていました。2.0からは1種類になりましたので何も設定しません。

リソース設定解説(4/8)
Excelサンプル　表 7 – サービスリソース

各リソースの動作を制御するパラメータを指定します。

- 以下はpgsql RAでPostgreSQL 11を制御する場合の例です。サンプルには他の設定例も記載されています。
- 各リソースにどのようなパラメータが指定できるかを調べたい時はHA Add-onナレッジの以下が参考になります。
  - 10.2. リソース固有のパラメーターの表示

PRIMITIVE

	P	id	class	provider	type	
#		リソースID	class	provider	type	概要
		pgsql	ocf	linuxhajp	pgsql	

	A	type	name	value	
#		パラメータ種別	項目	設定内容	概要
		options	pgctl	/usr/pgsql-11/bin/pg_ctl	
			psql	/usr/pgsql-11/bin/psql	
			pgdata	/dbfp/pgdata/data	
			pgdba	postgres	
			pgport	5432	
			pgdb	template1	

	O	type	timeout	interval	on-fail	role	start-delay	
#		オペレーション	タイムアウト値	監視間隔	障害時の動作	役割	起動前待機時間	備考
		start	300s		restart			
		monitor	60s	10s	restart			
		stop	300s		fence			

providerにlinuxhajpを指定すると pm_extra_toolsに含まれるpgsql RAを使用することになります。

コミュニティ版の PostgreSQLやPG-REXを使う場合はpm_extra_tools版を使ってください

リソース設定解説(5/8)
Excelサンプル　表７－NW監視

ＮＷ監視のパラメータを指定します。

NW監視は常に両系で動作させ、サービスに必要なネットワーク経路の正常性を監視します。
host_listに監視先のIPを指定します。

	PRIMITIVE							
	P	id	class	provider	type			
#		リソースID	class	provider	type	概要		
		ping	ocf	pacemaker	ping			
	A	type	name	value				
#		パラメータ種別	項目	設定内容		概要		
		options	name	ping-status				
			host_list	192.168.1.1				
			attempts	2				
			timeout	2				
			debug	true				
	O	type	timeout	interval	on-fail	role	start-delay	
#		オペレーション	タイムアウト値	監視間隔	障害時の動作	役割	起動前待機時間	備考
		start	60s		restart			
		monitor	60s	10s	restart			
		stop	60s		fence			

通常、ゲートウェイのIPをhost_listに指定します。

リソース設定解説(6/8)
Excelサンプル　表 8

STONITHリソースに渡すパラメータを指定します。

- 今回使用するfence_ipmilanはiLO経由で強制電源断を行うリソースです。そのため指定するパラメータはiLOのIPやユーザIDです。
- 下記はr81-1の電源断を行うSTONITHリソースです。このリソースは常に対向のr81-2で動作させる必要があります。これは次の表9で設定します。

STONITH				
P	id	type		
#	STONITHリソースID	type		
	fence1-ipmilan	fence_ipmilan		
A	type	name	value	
#	パラメータ種別	項目	設定内容	
	options	delay	10	
		pcmk_host_list	r81-1	
		ip	192.168.2.85	
		username	root	
		password	nttossc	
		lanplus	1	
		ipmitool_path	/usr/bin/ipmitool_stub	
O	type	timeout	interval	on-fail
#	オペレーション	タイムアウト値	監視間隔	障害時の動作
	start	60s		restart
	monitor	60s	3600s	restart
	stop	60s		ignore

STONITH				
P	id	type		
#	STONITHリソースID	type		
	fence2-ipmilan	fence_ipmilan		
A	type	name	value	
#	パラメータ種別	項目	設定内容	
	options	delay	0	
		pcmk_host_list	r81-2	
		ip	192.168.2.86	
		username	root	
		password	nttossc	
		lanplus	1	
		ipmitool_path	/usr/bin/ipmitool_stub	
O	type	timeout	interval	on-fail
#	オペレーション	タイムアウト値	監視間隔	障害時の動作
	start	60s		restart
	monitor	60s	3600s	restart
	stop	60s		ignore

スプリットブレインになった際、相撃ちして両系が停止しないようr81-1をSTONITHする↑の設定ではdeleyというSTONITH発動の待機時間10秒を指定しています

r81-2側にはdelayに0秒を指定しています

リソース設定解説(7/8)
Excelサンプル　表 9

リソースを配置するノード/ルールを設定します。

- リソース配置制約(ノード)では下記の2つの制約を課しています。
  - pgsql-groupはr81-1で優先的に動作させる(スコアがr81-1の方が高い)
  - fence1_ipmilan、 fence2_ipmilanはそれぞれr81-1、 r81-2では動作させない(必ず対向をSTONITHさせる)

#表 9-1 クラスタ設定　...　リソース配置制約（ノード）

LOCATION_NODE

	rsc	prefers/avoids	node	score	
#	リソースID	prefers/avoids	ノード	スコア	備考　（※roleを設定したい場合は、表9-2 を使用してください）
	pgsql-group	prefers	r81-1	200	
			r81-2	100	
	fence1-ipmilan	avoids	r81-1		
	fence2-ipmilan	avoids	r81-2		

- リソース配置制約(ルール)ではpgsql-groupはNW監視が異常(ping-statusが1未満)もしくはNW監視が定義されていない環境では動作させないという制約を課しています。

#表 9-2 クラスタ設定　...　リソース配置制約（ルール）

LOCATION_RULE

	rsc	score	bool_op	attribute	op	value	role	
#	リソースID	スコア	and/or	条件属性名	条件	条件値	役割	備考
	pgsql-group	-INFINITY	or	ping-status	lt	1		
				ping-status	not_defined			

Pacemaker-1.1ではルール/ノードをまとめて一つの表にしていましたが、Pacemaker-2.0から分割しました

リソース設定解説(8/8)
Excelサンプル　表 10~11

リソース間の同居/起動順序制約を設定します。

- リソース同居制約ではpgsql-groupをping-cloneが動作しているノードで動作させるというリソース間の同居に関する制約を課しています。

#表 10-1 クラスタ設定　…　リソース同居制約

COLOCATION

	rsc	with-rsc	score	rsc-role	with-rsc-role	
#	制約関連リソースID	制約対象リソースID	スコア（重み付け）	制約関連ﾘｿｰｽの役割	制約対象リソースの役割	備考
	pgsql-group	ping-clone	INFINITY			

- リソース順序制約ではping-cloneが起動した後にpgsql-statusを起動させるというリソース間の起動順序に関する制約を課しています。

#表 11-1 クラスタ設定　…　リソース起動順序制約

ORDER

	first-rsc	then-rsc	kind	first-action	then-action	symmetrical	
#	先に起動するリソースID	後に起動するリソースID	Optional/Mandatory/Serial	先起動ﾘｿｰｽのアクション	後起動ﾘｿｰｽのアクション	起動と逆順に停止（y	備考
	ping-clone	pgsql-group				false	

サンプル解説はここまでです。
次のページではリソース設定を反映させる手順を紹介します

リソース設定反映手順

1. Excel環境定義書をCSVに変換し、いずれか一方のノードへ格納
   - ここでは”sample.csv”とします。
2. xmlファイルへの変換　●CSVを配置したノードで

```
# pm_pcsgen sample.csv
sample.xml (CIB), sample.sh (PCS) を出力しました。
```

3. Pacemakerの起動　★いずれか一方のノードで

```
# pcs cluster start --all
```

4. xmlファイルのロード　●CSVを配置したノードで

```
# pcs cluster cib-push sample.xml
```

〇〇.csvをpm_pcsgenで変換すると〇〇.xmlと〇〇.shが作成されます。

sample.sh(PCS)の中にはpcsでリソース設定を行う場合のコマンドが羅列されています。

構築・リソース設定が上手くいくと こんな感じになります♪

```
# pcs status –full
```

```
・・・省略・・・
Node List:
  * Online: [ r81-1 (1) r81-2 (2) ]

Full List of Resources:
  * Resource Group: pgsql-group:
    * filesystem1       (ocf::heartbeat:Filesystem): Started r81-1
    * filesystem2       (ocf::heartbeat:Filesystem): Started r81-1
    * filesystem3       (ocf::heartbeat:Filesystem): Started r81-1
    * ipaddr    (ocf::heartbeat:IPaddr2):       Started r81-1
    * pgsql     (ocf::linuxhajp:pgsql): Started r81-1
  * Clone Set: ping-clone [ping]:
    * ping      (ocf::pacemaker:ping):  Started r81-2
    * ping      (ocf::pacemaker:ping):  Started r81-1
    * Started: [ r81-1 r81-2 ]
  * fence1-ipmilan      (stonith:fence_ipmilan):        Started r81-2
  * fence2-ipmilan      (stonith:fence_ipmilan):        Started r81-1

Node Attributes:
  * Node: r81-1 (1):
    * ping-status                   : 1
  * Node: r81-2 (2):
    * ping-status
・・・省略・・・
```

Started ●●は●●で対象のリソースが起動していることを示しています

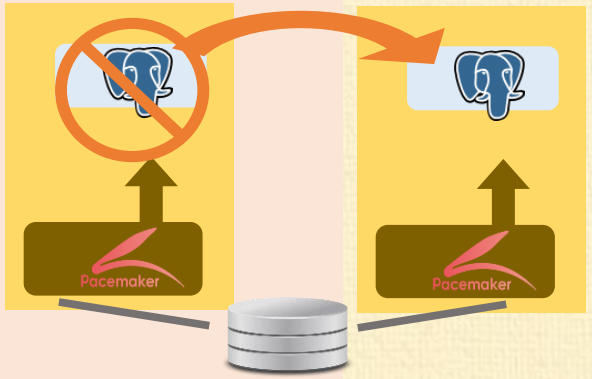
PostgreSQL故障が発生して F/Oするとこうなります

r81-2でリソースが起動していることが分かります

```
Node List:
  * Online: [ r81-1 (1) r81-2 (2) ]

Full List of Resources:
  * Resource Group: pgsql-group:
    * filesystem1       (ocf::heartbeat:Filesystem): Started r81-2
    * filesystem2       (ocf::heartbeat:Filesystem): Started r81-2
    * filesystem3       (ocf::heartbeat:Filesystem): Started r81-2
    * ipaddr    (ocf::heartbeat:IPaddr2):       Started r81-2
    * pgsql     (ocf::linuxhajp:pgsql): Started r81-2
  * Clone Set: ping-clone [ping]:
    * ping      (ocf::pacemaker:ping):  Started r81-2
    * ping      (ocf::pacemaker:ping):  Started r81-1
    * Started: [ r81-1 r81-2 ]
  * fence1-ipmilan      (stonith:fence_ipmilan):        Started r81-2
  * fence2-ipmilan      (stonith:fence_ipmilan):        Started r81-1

Node Attributes:
  * Node: r81-1 (1):
    * ping-status               : 1
  * Node: r81-2 (2):
    * ping-status               : 1
```



↓のように故障情報が残ります

```
Failed Resource Actions:
  * pgsql_monitor_10000 on r81-1 'not running' (7): call=48, status='complete',
exitreason='No process state file found', last-rc-change='2020-10-06 03:34:49 -04:00', queued=0ms, exec=0ms
```

という感じでPacemaker-1.1と ほとんど変わりません！

Pacemaker-1.1の環境定義書についての詳細は↓の付録Aで
解説してますので興味がある方は見てください

試して覚えるPacemaker入門 リソース設定編 ～ Pacemakerで ノードやサービスを手玉に取ろう！ ～

http://linux-ha.osdn.jp/wp/archives/4532

それでも分からないことがあればMLへ

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

今後もPacemakerを
よろしくお願いします