

Hyper-V上でPacemakerの 高可用クラスタを構築してみた

– Pacemaker構築入門 –

2026-10-2 OSC2026 Online/Fall

Linux-HA Japan プロジェクト

内田 智



自己紹介

内田 智（うちだ さとし）

- Pacemaker を触り始めたのは 2020 年ごろ
- 2023 年 8 月から NTT OSS センタへ在籍
- OSC は今回が初参加です

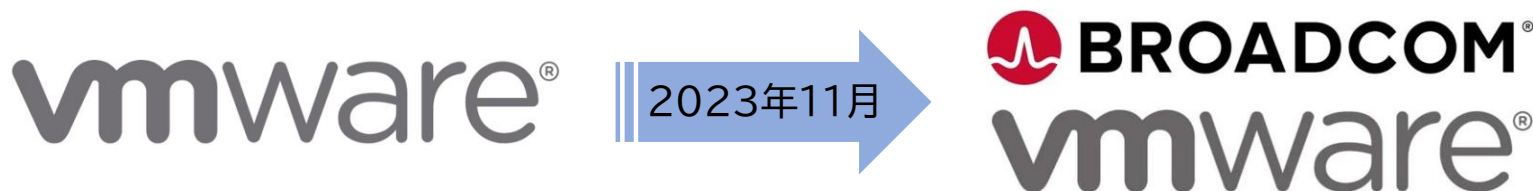
本日の内容

- 1.なんで今Hyper-V？
- 2.Pacemakerとは？
- 3.Pacemaker構築
- 4.Hyper-V特有の考慮点

なんで今Hyper-V ?

VMWareが高すぎる！

- 2023年11月にBroadcomがVMwareを買収



- その結果VMwareのライセンス費が高騰し、VMware環境の維持が困難に
 - CPU課金の永続ライセンスは廃止、Core課金のサブスク型へ
 - 新ライセンス体系では4つのグレードがあるが、会社の規模等によって購入できるグレードが決められている
 - NTTの場合は最上位グレードの **VMware Cloud Foundation (VCF)** しか買えない



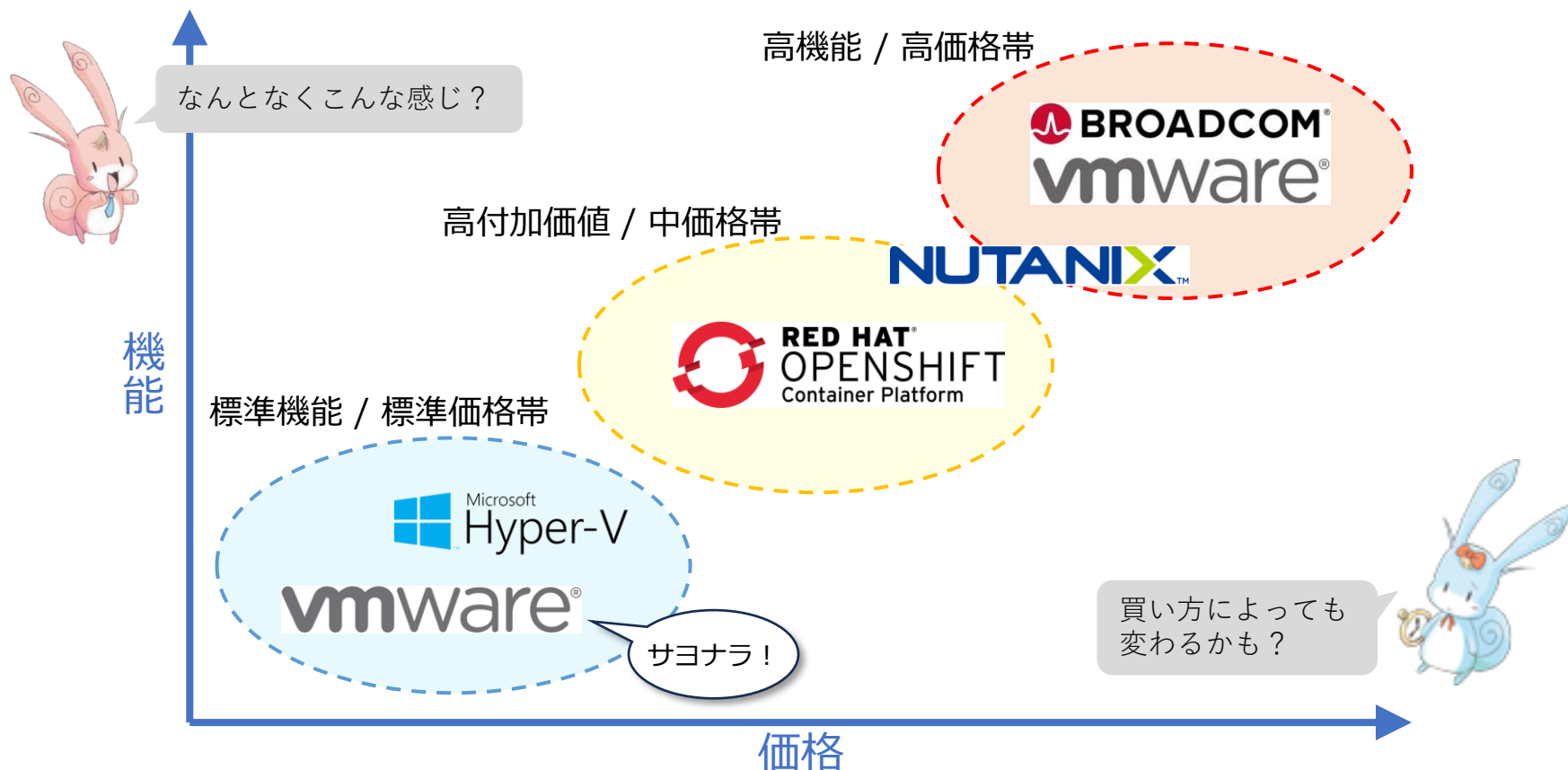
機能もりもりで一番
お高いやつだ...

168製品のバラ売りだったのが
まとめ売り4パターンに...
そんなにたくさんの機能は使
わないんだけどな～



お手頃な乗り替え先は？

- パブクラ環境は別途あるので、オンプレ環境で検討
- 標準的な機能があれば十分、なるべくお財布にやさしく...



Hyper-Vってどうなの？

- 仮想化ソフトの標準的な機能は備えていて価格もお手頃
- Windows向けのイメージだがRHELもサポートされている

けっこうよさそうかも？



- しかし、**RHEL HA Add-On** はサポートされない
 - Support Policies for RHEL High Availability Clusters - Microsoft Hyper-V Virtual Machines as Cluster Members
<https://access.redhat.com/articles/3131321>



RHELのPacemakerは **RHEL HA Add-On** で提供されてるよ

! ?



- とはいえ、Hyper-VだからといってPacemakerの動作に何か問題が出るようにも思えないが...？

ちょっと試しに動かしてみよう！

Pacemakerとは？

Pacemakerとは？

- オープンソースのHAクラスタソフトウェアです

High Availability = 高可用性

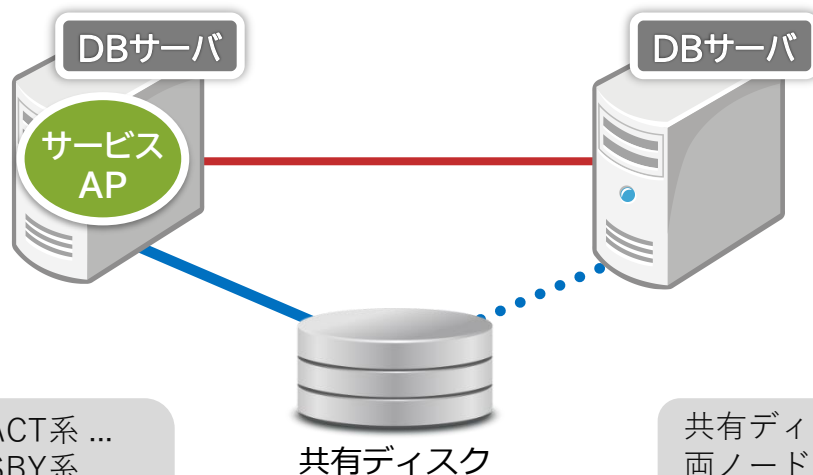
- システムサービスが停止する時間をできる限り短くすることが目的です
- ClusterLabsのコミュニティで管理されています
 - <https://github.com/ClusterLabs>

HAクラスタの概要

- サービス起動中のサーバに故障が発生した場合、他のサーバに処理を引き継ぐことでサービスを継続します
- サービスを提供する現用ノードと、予備機となる待機ノードで構成された、Active-Standby構成が一般的
- サーバ間でデータを共有する場合は共有ディスクを使用

現用ノード(Active)

待機ノード(Standby)



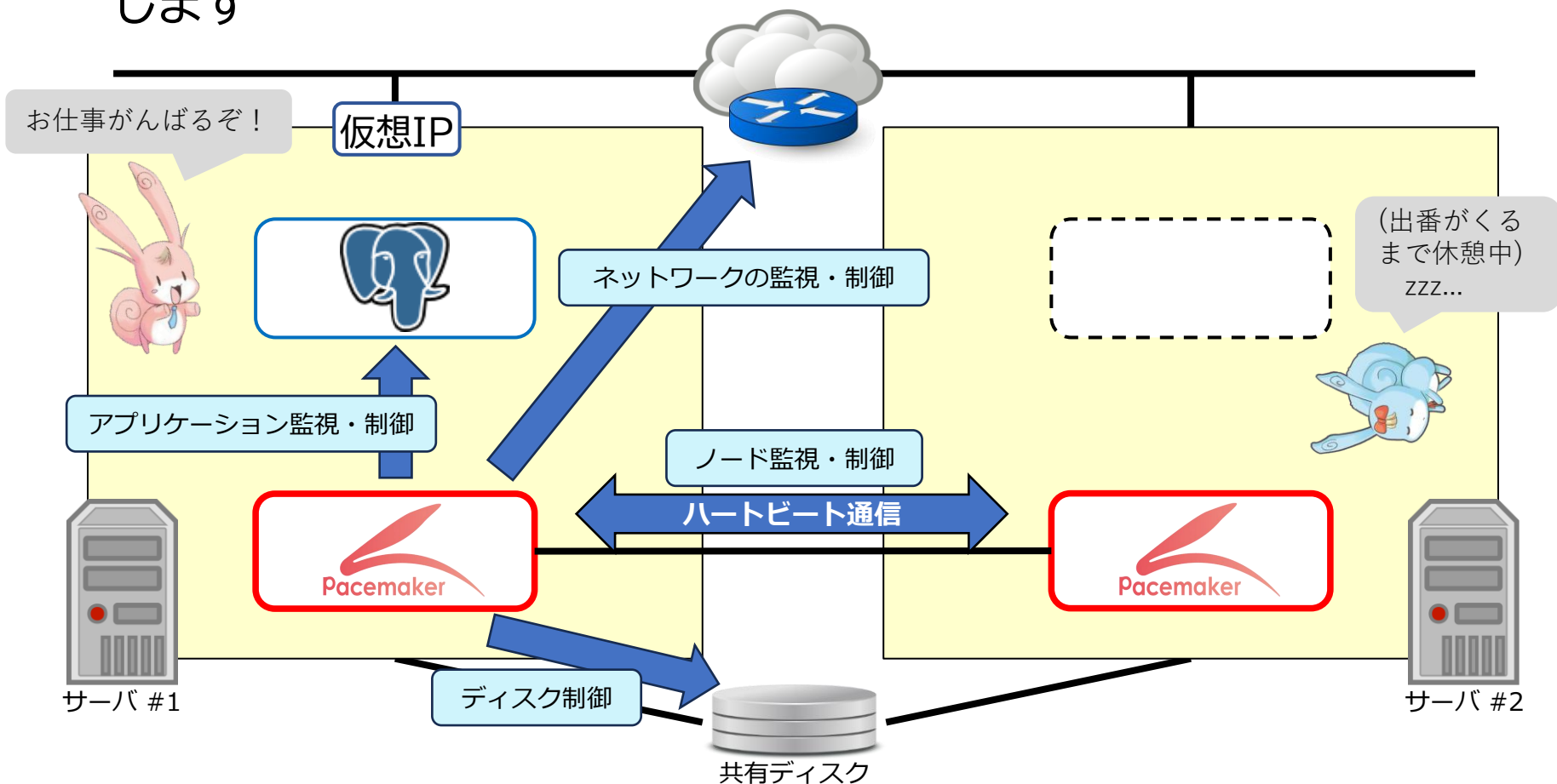
現用系、稼働系、ACT系 ...
待機系、予備系、SBY系 ...
などなどいろんな呼ばれ方が
あります

共有ディスクを使わずに、
両ノード間でデータを同期
する方式もあります



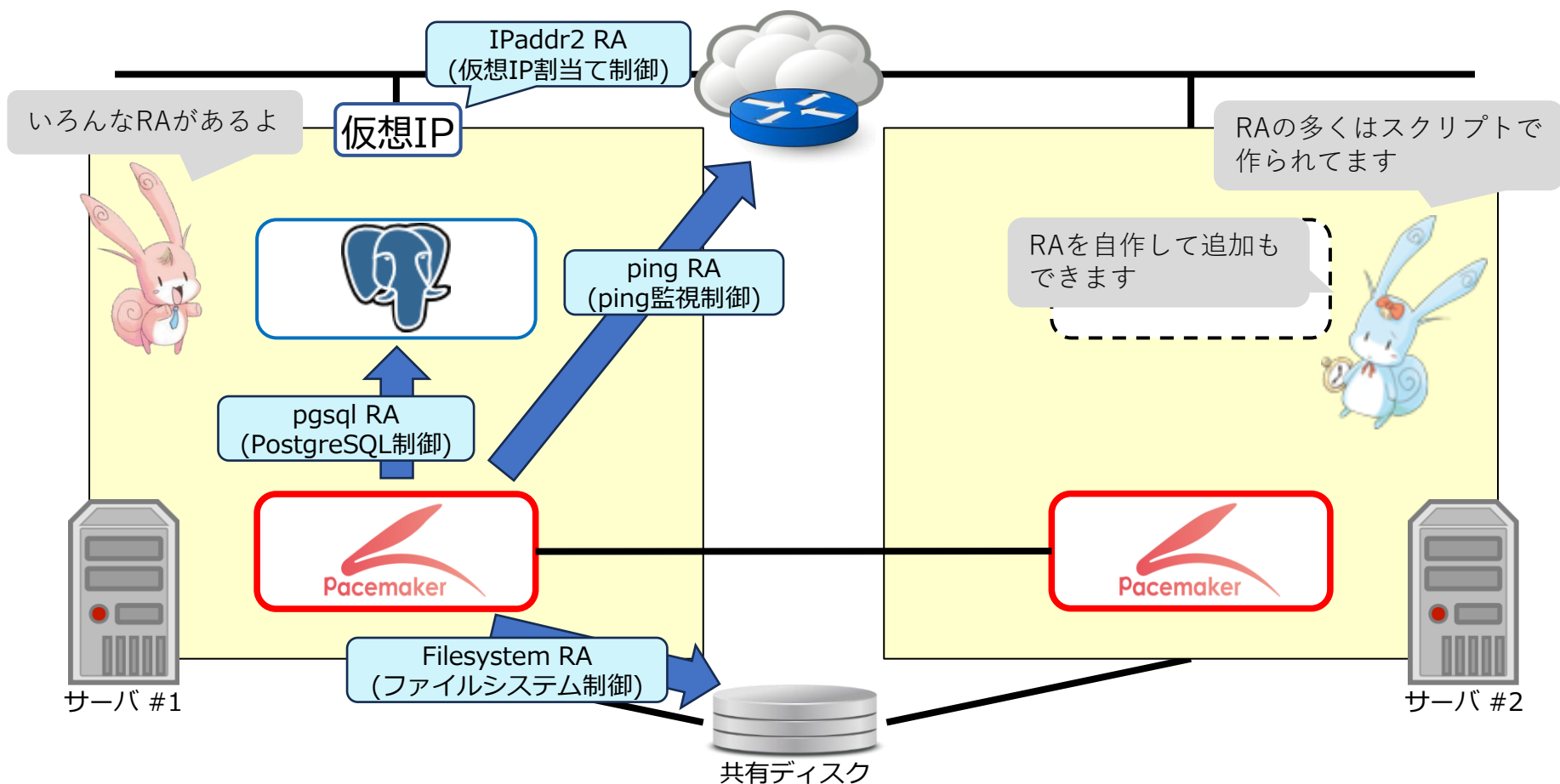
リソース制御・監視

- 様々なリソースを制御し、問題なく動いているか監視します
- 各リソースは現用ノードで起動し、待機ノードでは停止します
- 現用ノードと待機ノードはハートビート通信でお互いの状態を確認します



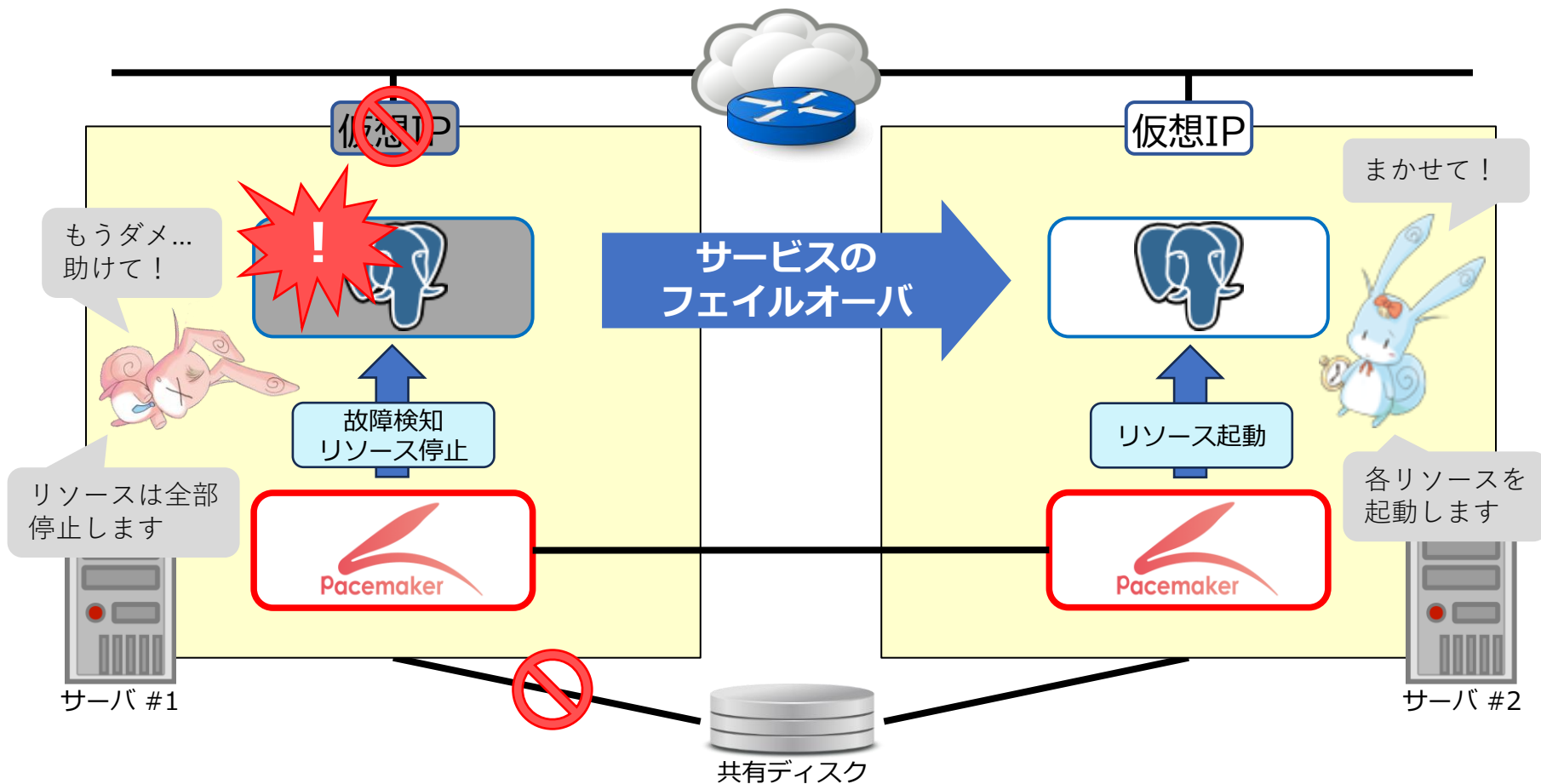
リソースエージェント

- 各種リソースを制御(起動・停止・監視)する実行ファイル
- 略して RA(Resource Agent) と呼ばれます



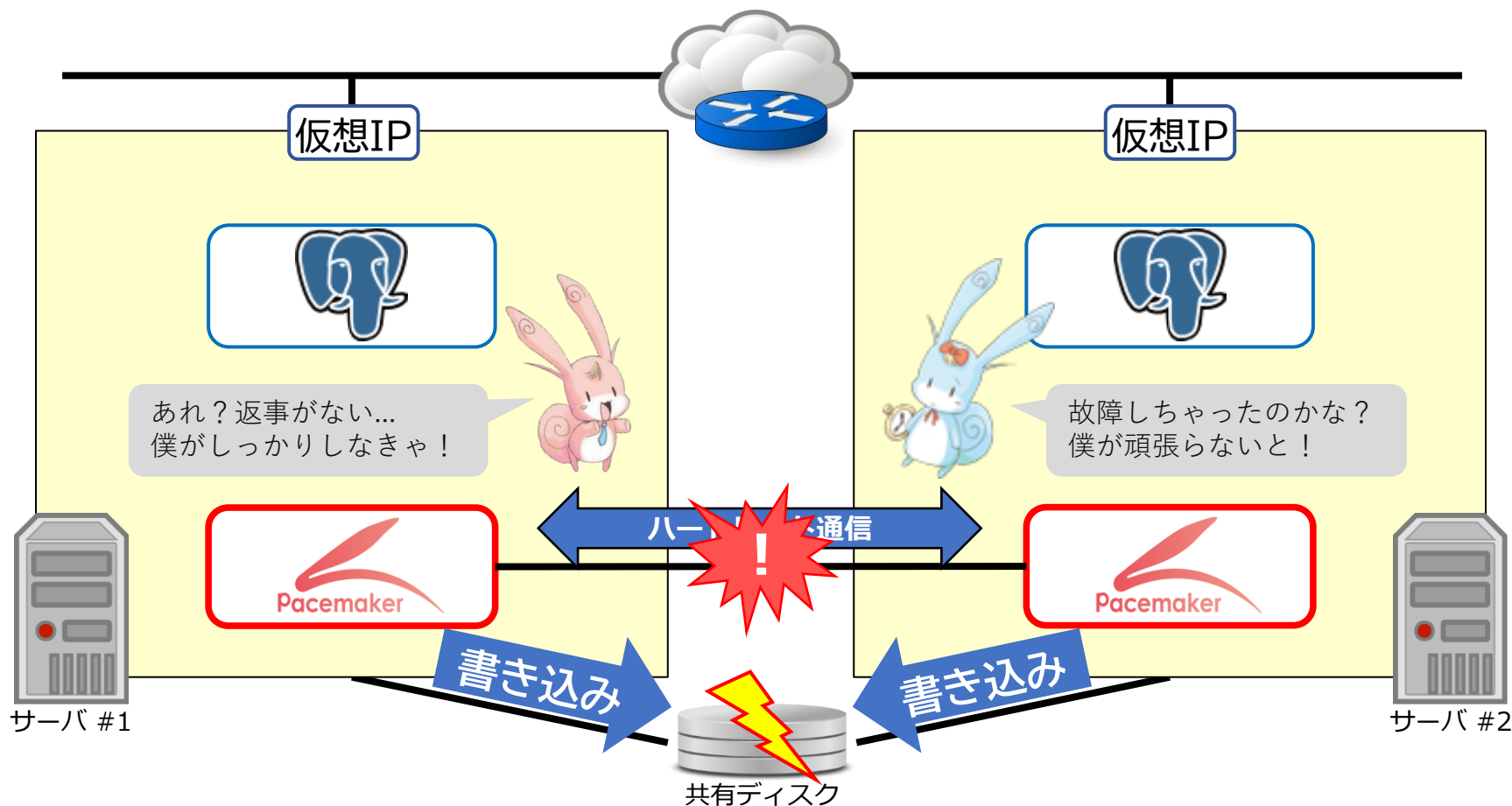
フェイルオーバー

- 故障を検知すると、自動的に待機ノードでリソースを起動し、サービスを継続します
- 略して **F/O**(**F**ail **O**ver) と呼ばれます



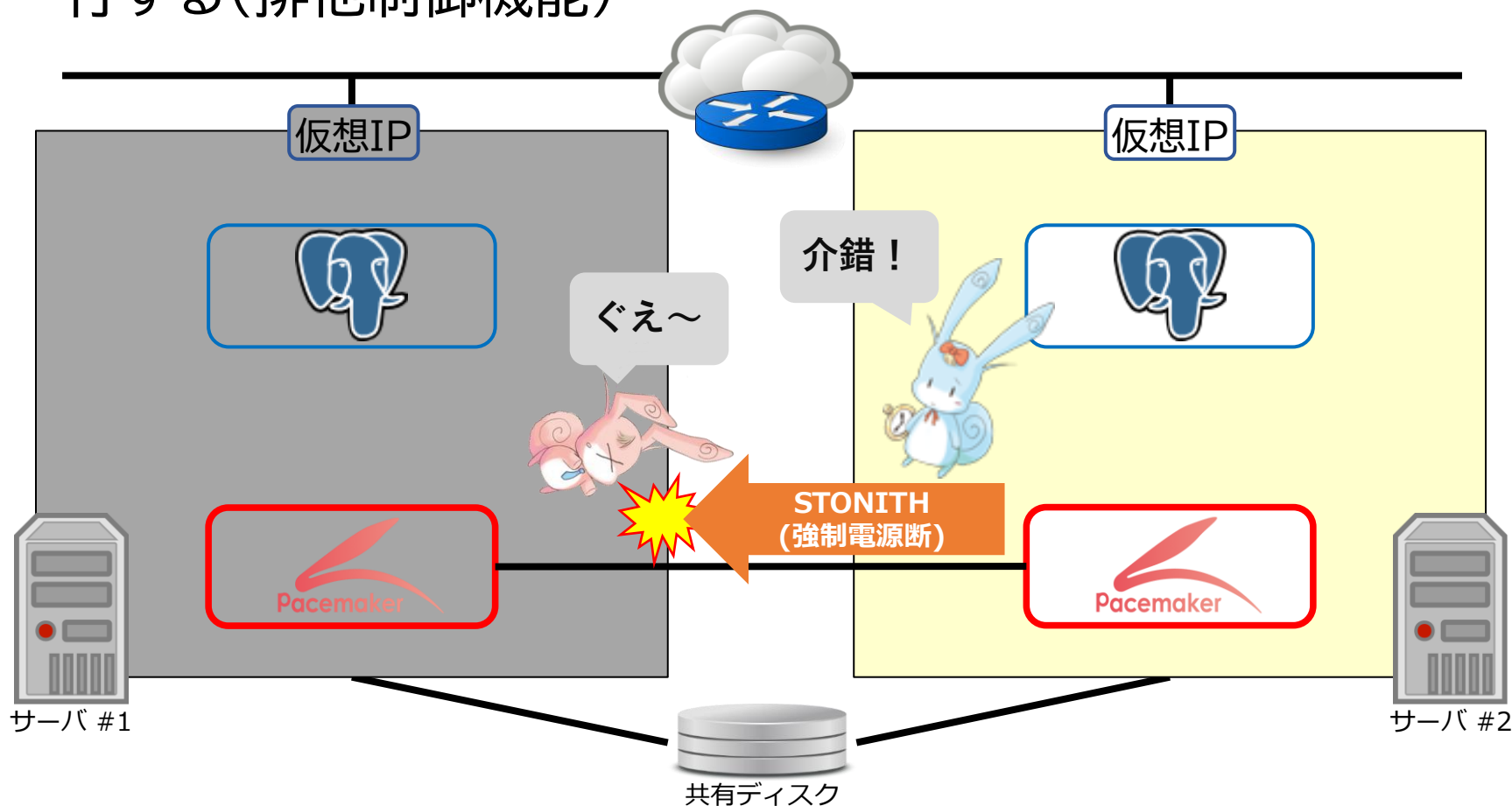
スプリットブレイン

- ハートビート通信途絶により両系でサービスが起動した状態
- 両系からのディスク書き込みによりデータが破損する



STONITH

- STONITH(Shoot The Other Node In The Head)
- スプリットブレインになる前に対向ノードの強制電源断を実行する(排他制御機能)



STONITH

- RHEL HA Add-On のPacemakerを利用する場合、**STONITH機能の利用は必須**です
- Red Hat社は、STONITH機能を使っていないクラスタはサポートしないと明言しています
 - Support Policies for RHEL High Availability Clusters - General Requirements for Fencing/STONITH
<https://access.redhat.com/articles/2881341>

フェンスエージェント

- STONITHを実行するモジュールのこと
- 環境に合わせた様々なフェンスエージェントがある
 - **fence_ipmilan**
物理環境向けのフェンスエージェント。
ハードウェア制御ボード(iLO等)を利用してSTONITHを実行。
 - **fence_vmware_rest**
VMware環境向けのフェンスエージェント。
vCenterの機能を利用してSTONITHを実行。
 - **fence_scsi**
共有ディスクがある環境向けのフェンスエージェント。
共有ディスクのSCSIリザーベーション機能を利用してSTONITHを実行。
 - etc...



他にも色々なフェンスエージェントがあるよ

STONITHのことをフェンシングと言ったりもします（隔離・防御のような意味合い）

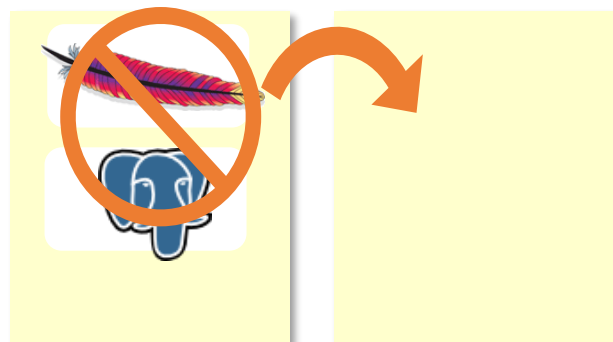
fence_ipmilan や fence_vmware_rest のように、基本的にはその環境向けの専用フェンスエージェントを使います

専用のフェンスエージェントがない環境では、共有ディスクさえあれば汎用的に使える fence_scsi 等を利用します

もし共有ディスクもなかったらフェンスエージェントを自作するしかありません...



(参考) さまざまなHAクラスタ構成



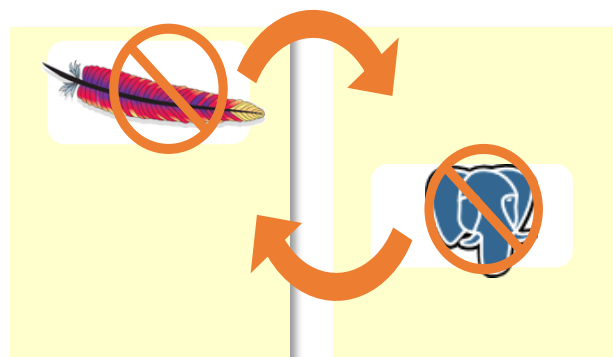
サーバ#1 (Act) サーバ#2 (Sby)

1+1構成



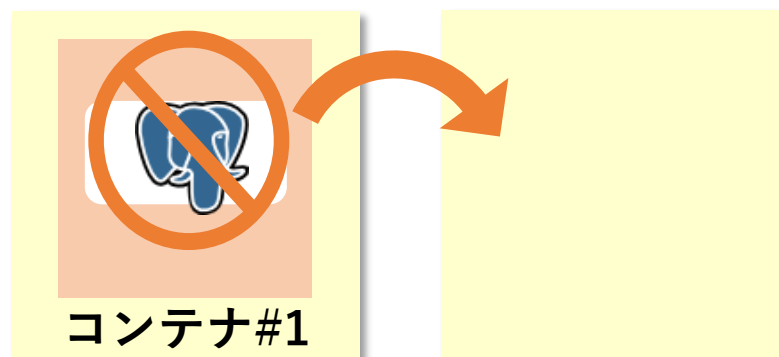
サーバ#1 (Act) サーバ#2 (Act) サーバ#3 (Sby)

N+1構成(N+M構成)



サーバ#1 (Act) サーバ#2 (Act)

1+1Cross構成



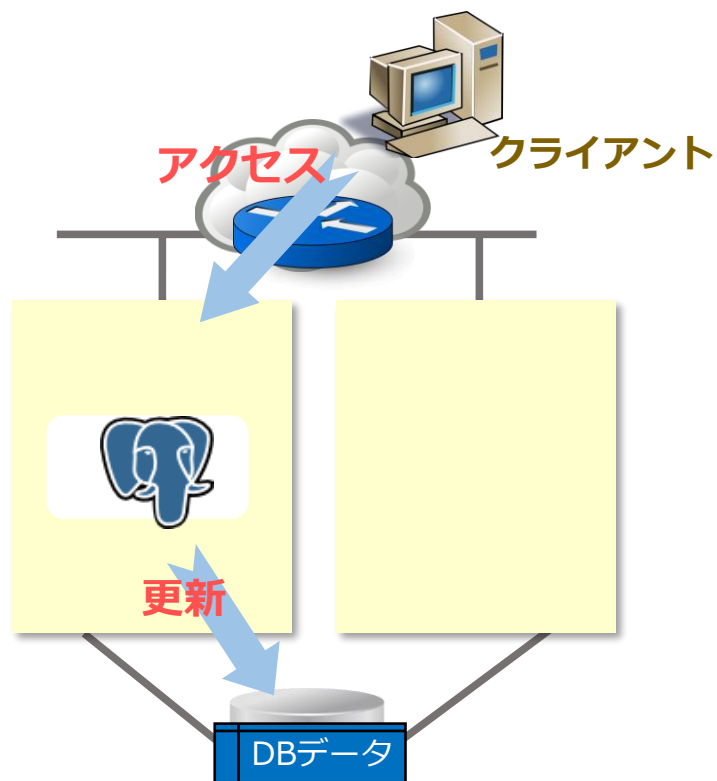
コンテナ#1

サーバ#1

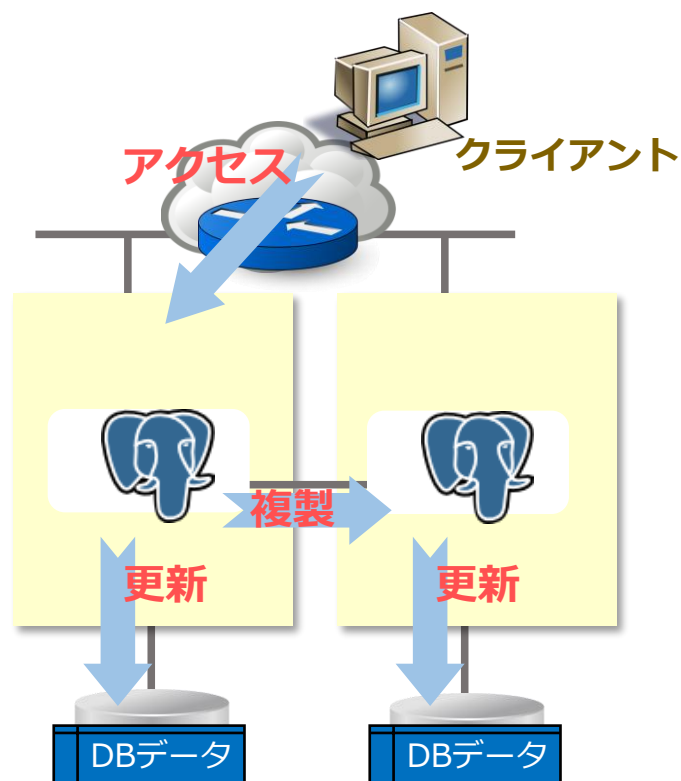
サーバ#2

コンテナの管理(bundle)

(参考) さまざまなHAクラスタ構成



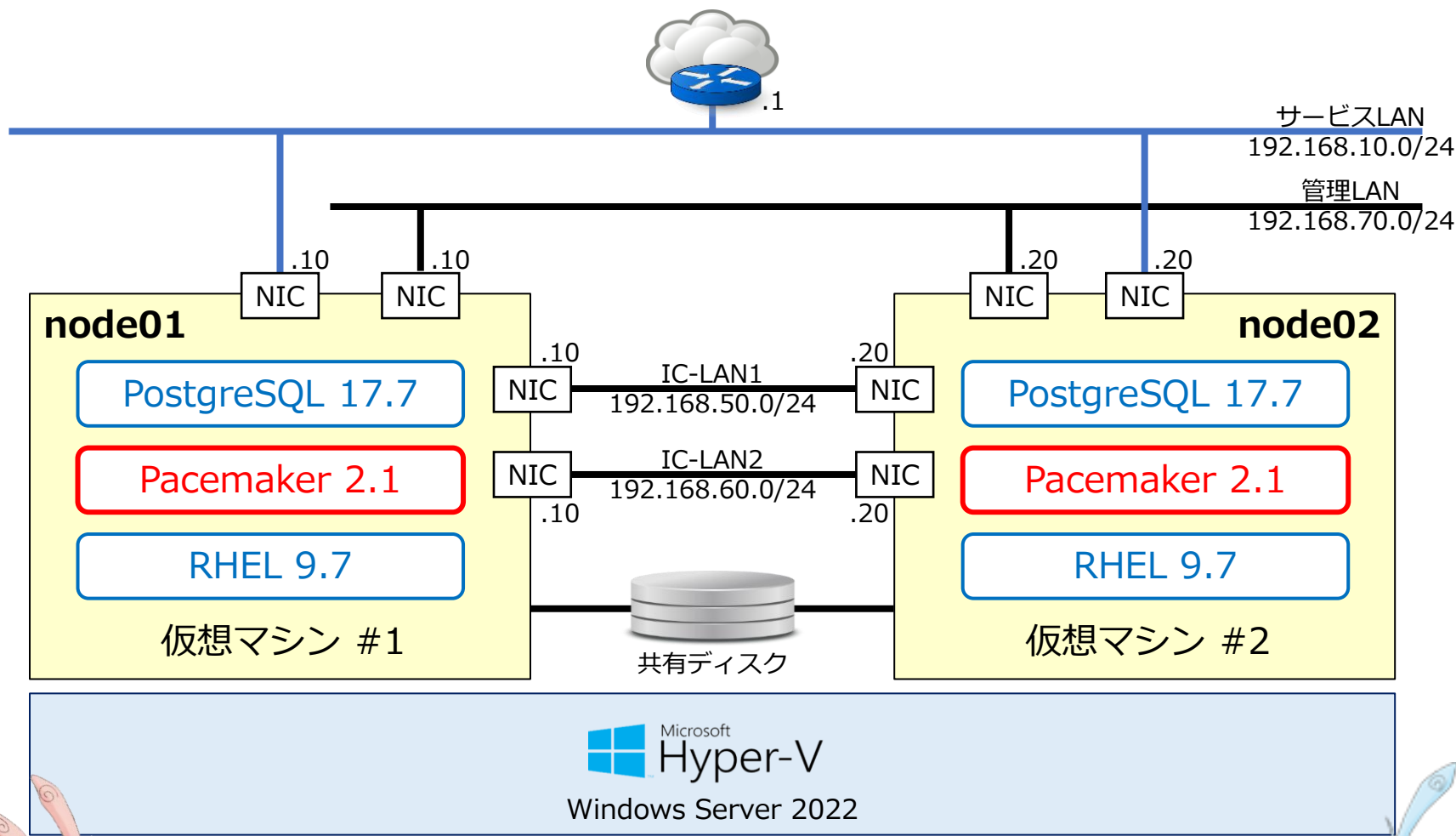
シェアードディスク構成



シェアードナッシング構成

Pacemaker構築

今回の環境構成



IC-LANはハートビート通信用のLANです

インターコネクトLAN (Inter **C**onnect **L**AN) を略してIC-LANですね

Pacemakerの設定

- Pacemaker 関連パッケージをインストール（両ノードで実施）

```
# dnf -y install pcs pacemaker fence-agents-all
```

- クラスタが利用するポートの通信許可（両ノードで実施）

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

- pcsd.service の起動および自動起動設定（両ノードで実施）

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



pcsd.service を介してPacemakerとCorosyncを管理します

Corosyncはハートビート通信の担い手です



Pacemakerの設定

- hacluster ユーザのパスワードを設定（両ノードで実施）

```
# passwd hacluster
ユーザー hacluster のパスワードを変更。
新しいパスワード:
新しいパスワードを再入力してください:
passwd: すべての認証トークンが正しく更新できました。
```

- クラスタノードの認証設定（いずれか片方のノードで実施）

```
# pcs host auth node01 addr=192.168.70.10 node02 addr=192.168.70.20
Username: hacluster
Password:
node01: Authorized
node02: Authorized
```

先ほど設定したパスワードを入力

Pacemakerの設定

- クラスタの作成（いずれか片方のノードで実施）

```
# pcs cluster setup DB_Cluster ¥  
node01 addr=192.168.50.10 addr=192.168.60.10 ¥  
node02 addr=192.168.50.20 addr=192.168.60.20
```

- 故障検知時のfail_fast動作を設定（両ノードで実施）

```
# vi /etc/sysconfig/pacemaker  
～略～  
PCMK_fail_fast=yes  
PCMK_panic_action=sync-reboot
```



このfail_fast動作は Linux-HA Japan の推奨設定です

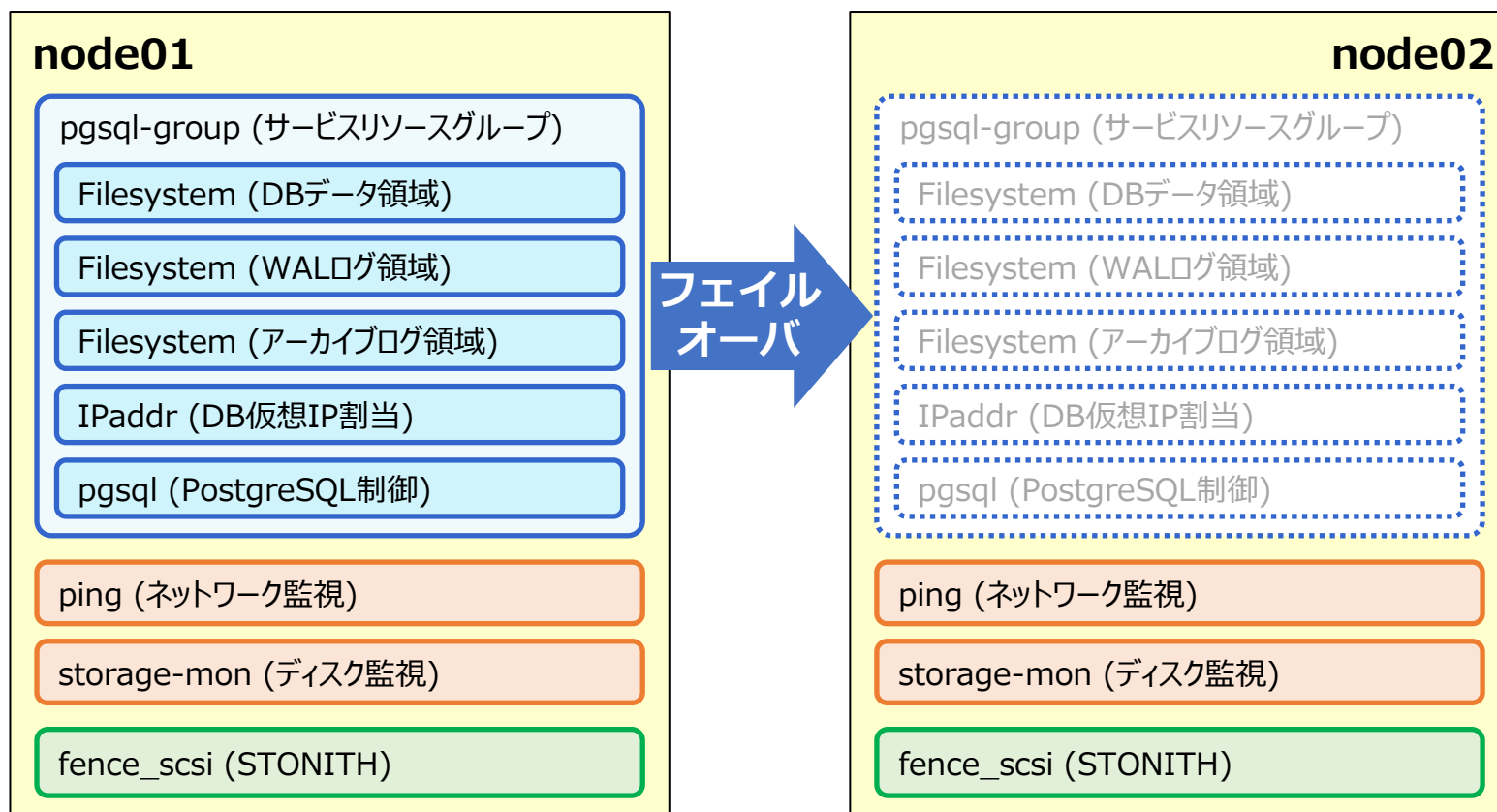
設定の意味はこんな感じ

- **PCMK_fail_fast=yes**
Pacemaker に異常が起きた際に再起動
- **PCMK_panic_action=sync-reboot**
メモリ上にある変更内容をディスクに書き込み後、再起動



今回のクラスタリソース構成

- Hyper-Vには専用のフェンスエージェントがありません
- なので fence_scsi を使用します



クラスタリソースの設定

- Red Hatのお作法に従いpcsでリソースを設定する場合、以下のようなpcsコマンドを一から作成しなければなりません

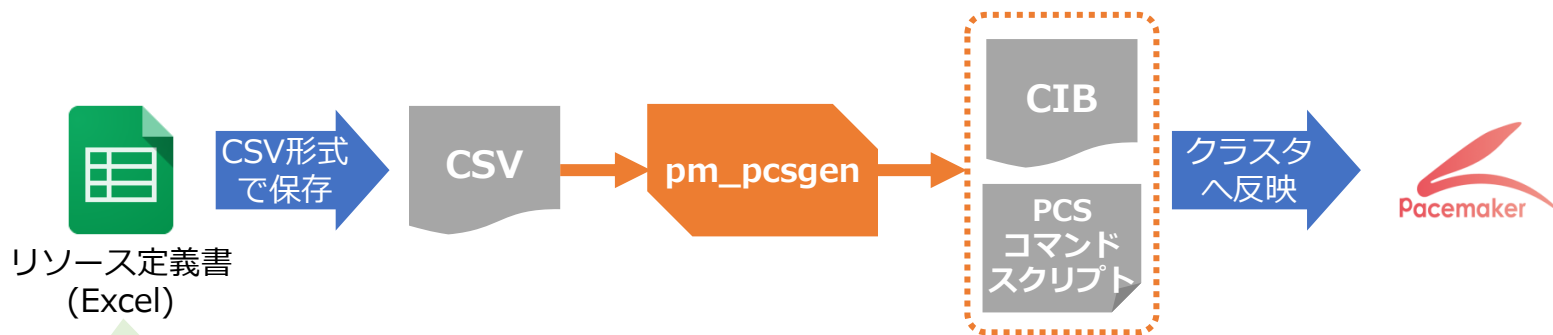
```
# pcs resource defaults resource-stickiness=200
# pcs resource defaults migration-threshold=1
# pcs resource create filesystem1 ocf:heartbeat:Filesystem device="/dev/mapper/mpatha2" directory="/dbfp/pgdata" fstype="xfs"
force_unmount="safe" op start timeout=60s on-fail=restart monitor timeout=60s 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 postgresql ocf:linuxhbj:postgresql 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 postgresql-group filesystem1 filesystem2 filesystem3 ipaddr postgresql
# 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 postgresql-group prefers r81-1=200
# pcs constraint location postgresql-group prefers r81-2=100
# pcs constraint location fence1-ipmilan avoids r81-1
# pcs constraint location fence2-ipmilan avoids r81-2
# pcs constraint location postgresql-group rule score=-INFINITY ping-status lt 1 or not_defined ping-status
# pcs constraint colocation add postgresql-group with ping-clone score=INFINITY
# pcs constraint order ping-clone then postgresql-group symmetrical=false
```

これはちょっとたいへんそう



クラスタリソースの設定

- Linux-HA Japan では、Excel形式のリソース定義書からpcsコマンドやクラスタ設定ファイル(CIB)を生成する pm_pcsген というツールを用意しています
 - pm_pcsген は pm_extra_tools というパッケージに含まれます
 - 以下のサイトから pm_extra_tools を入手可能です
https://github.com/linux-ha-japan/pm_extra_tools/releases



#表 4-1 クラスタ設定 ... リソース構成

RESOURCES			
#	リソース構成要素	リソースID	概要
Group	Primitive	pgsql-group	グループ定義
	Primitive	filesystem1	DBクラスタ領域
	Primitive	filesystem2	WALログ領域
	Primitive	filesystem3	アーカイブログ領域
	Primitive	ipaddr	DB仮想IP割当
Clone	Primitive	pgsql	PostgreSQL制御
	Primitive	ping-clone	クローン定義
Clone	Primitive	ping	ネットワーク監視
	Primitive	storage-mon-clone	クローン定義
Stonith	Primitive	storage-mon	ディスク監視
	Primitive	fence1-scsi	ディスク監視
Stonith	Primitive	fence2-scsi	ディスク監視
	Primitive	fence3-scsi	ディスク監視



Excelの方がクラスタ設定の全体像を把握しやすいね

クラストリソースの設定

- 今回作成したリソース定義書(Excel)の全体は以下くらいの分量です
- Hyper-Vだからといって特別に設定した箇所は(今のところ)ありません

[illegible][illegible][illegible]

クラスタリソースの設定

- クラスタ全体の設定

#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
	pe-input-series-max	0	正常状態遷移ファイルの履歴数	
	pe-warn-series-max	0	エラー発生状態遷移ファイルの履歴数	
	pe-error-series-max	0	警告発生状態遷移ファイルの履歴数	
	priority-fencing-delay	30s	fencing実行遅延時間	
	node-health-strategy	only-green	ノード正常性属性	
	fence-reaction	panic	自ノードのフェンシング通知受信時の動作	fence_scsi使用のため設定

#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	



「N回故障したらF/O」としたい場合は migration-threshold にNを設定します

今回は1回でも故障を検知したら F/Oさせます

STONITHの相打ちで両系が停止してしまうことを避けるため、priority-fencing-delay でSTONITHの実行タイミングをずらしします



クラスタリソースの設定

- リソース構成

#表 4-1 クラスタ設定 ...リソース構成

RESOURCES				
	resourceltem	resourceltem	resourceltem	id
#	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	
		Primitive	ping	
	Clone		storage-mon-clone	
		Primitive	storage-mon	
	Stonith		fence1-scsi	
	Stonith		fence2-scsi	



今回は pgsql-group という名前のリソースグループでサービスリソースをまとめています

NW監視とディスク監視は常に両ノードで実施するためクローンリソースとなっています

node01 と node02 それぞれ向けにSTONITHリソースが2つあります



クラスタリソースの設定

• PostgreSQL

#表 7-1-5 クラスタ設定 ... Primitiveリソース (id=pgsql)

PRIMITIVE						
#	P	id	class	provider	type	
		リソースID	class	provider	type	概要
		pgsql	ocf	linuxhajp	pgsql	PostgreSQL制御
#	A	type	name	value		
		パラメータ種別	項目	設定内容	概要	
	options	pgctl	/usr/pgsql-17/bin/pg_ctl			pg_ctlコマンド・パス
		psql	/usr/pgsql-17/bin/psql			psqlコマンド・パス
		pgdata	/dbfp/pgdata/data			DBクラスタ領域ディレクトリ名
		pgdba	postgres			DB管理ユーザ
		pgport	5432			通信ポート番号
		pgdb	template1			アクセス確認用DB名
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
	start	300s		restart		
	monitor	60s	10s	restart		
	stop	300s		fence		



コミュニティ版の PostgreSQL を使う場合は
pm_extra_tools版の pgsql RA を使ってください

provider に linuxhajp と指定すれば pm_extra_tools に
含まれる pgsql RA を使用できます



クラスタリソースの設定

- NW監視、ディスク監視

#表 7-2-1 クラスタ設定 ... Primitiveリソース (id=ping)

PRIMITIVE						
#	P					
	id	class	provider	type		
	リソースID	class	provider	type	概要	
	ping	ocf	pacemaker	ping	ネットワーク監視	
#	A					
	type	name	value			
	パラメータ種別	項目	設定内容	概要		
	options	name	ping-status	ネットワーク監視用の属性名		
		host_list	192.168.10.1	監視するネットワークのIPアドレス		
		attempts	2	リトライ回数		
		timeout	2	タイムアウト		
		debug	true	ping失敗時検知時のログ出力		
#	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
	start	60s		restart		
	monitor	60s	10s	restart		
	stop	60s		fence		

#表 7-3-1 クラスタ設定 ... Primitiveリソース (id=storage-mon)

PRIMITIVE						
#	P					
	id	class	provider	type		
	リソースID	class	provider	type	概要	
	storage-mon	ocf	heartbeat	storage-mon	ディスク監視	
#	A					
	type	name	value			
	パラメータ種別	項目	設定内容	概要		
	options	drives	/dev/internal_a /dev/external_b /dev/external_c /dev/external_d	監視ディスクデバイス名		
		io_timeout	25	ディスクI/Oのタイムアウト時間(秒)		
#	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
	start	120s		restart		
	monitor	120s	10s	restart		
	stop	120s		fence		

クラスタリソースの設定

• STONITH (fence_scsi)

#表 8-1-1 クラスタ設定 ... STONITHリソース (id=fence1-scsi)

STONITH				
#	P	id	type	
		STONITHリソースID	type	
		fence1-scsi	fence_scsi	
			fence_scsiプラグイン(STONITHプラグイン)	
#	A	type	name	value
		パラメータ種別	項目	設定内容
		options	pcmk_host_list	node01
			devices	/dev/external_b_/dev/external_c_/dev/external_d
			pcmk_off_timeout	47
			pcmk_on_timeout	47
			pcmk_reboot_action	off
			power_wait	44
#	O	type	timeout	interval
		オペレーション	タイムアウト値	監視間隔
		start	60s	restart
		monitor	60s	3600s
		stop	60s	ignore

#表 8-2-1 クラスタ設定 ... STONITHリソース (id=fence2-scsi)

STONITH				
#	P	id	type	
		STONITHリソースID	type	
		fence2-scsi	fence_scsi	
			fence_scsiプラグイン(STONITHプラグイン)	
#	A	type	name	value
		パラメータ種別	項目	設定内容
		options	pcmk_host_list	node02
			devices	/dev/external_b_/dev/external_c_/dev/external_d
			pcmk_off_timeout	47
			pcmk_on_timeout	47
			pcmk_reboot_action	off
			power_wait	44
#	O	type	timeout	interval
		オペレーション	タイムアウト値	監視間隔
		start	60s	restart
		monitor	60s	3600s
		stop	60s	ignore

クラスタリソースの設定

・リソース制約

#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
	pgsql-group	prefers	node01	200
			node02	100
	備考			
	ホストnode01を優先的にActiveにする			

#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
	pgsql-group	-INFINITY	or	ping-status	!=	1	
				ping-status	not_defined		
				#health-storage-mon	not_defined		
	備考						
	ネットワーク故障が発生したノードでは起動しない						
	ディスク故障が発生したノードでは起動しない						

#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
	pgsql-group	ping-clone	INFINITY		
	pgsql-group	storage-mon-clone	INFINITY		
	備考				

#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/n)
	ping-clone	pgsql-group				false
	storage-mon-clone	pgsql-group				false
	備考					

NW監視とディスク監視がOKな場合にサービスリソースを起動できるような制約となっています



クラスタリソースの設定

- pm_extra_tools をインストール（両ノードで実施）

```
# dnf install pm_extra_tools-1.7-1.el9.noarch.rpm
```

```
# ls /usr/share/pm_extra_tools/
```

リソース定義書のサンプル

```
pm_pcsgen.conf  pm_pcsgen.py  pm_pcsgen_sample.xlsx
```

- 作成したリソース定義書(Excel)をCSV形式で保存し、仮想マシンへ転送
- CSVファイルをCIBファイルへ変換（いずれか片方のノードで実施）

```
# pm_pcsgen sample.csv
```

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

CIBファイル

pcsコマンドスクリプト

クラスターソースの設定

- Pacemakerの起動（いずれか片方のノードで実施）

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

- CIBファイルをクラスタへ反映（いずれか片方のノードで実施）

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

クラスタリソースの設定

- クラスタの状態を確認（いずれか片方のノードで実施）

```
# pcs status --full
Cluster name: DB_Cluster
～略～
Node List:
  * Node node01 (1): online, feature set 3.19.6
  * Node node02 (2): online, feature set 3.19.6

Full List of Resources:
  * Resource Group: pgsql-group:
    * filesystem1 (ocf:heartbeat:Filesystem):      Started node01
    * filesystem2 (ocf:heartbeat:Filesystem):      Started node01
    * filesystem3 (ocf:heartbeat:Filesystem):      Started node01
    * ipaddr      (ocf:heartbeat:IPAddr2):         Started node01
    * pgsql       (ocf:linuxhajp:pgsql):           Started node01
  * Clone Set: ping-clone [ping]:
    * ping        (ocf:pacemaker:ping):            Started node02
    * ping        (ocf:pacemaker:ping):            Started node01
  * Clone Set: storage-mon-clone [storage-mon]:
    * storage-mon (ocf:heartbeat:storage-mon):     Started node02
    * storage-mon (ocf:heartbeat:storage-mon):     Started node01
  * fence1-scsi  (stonith:fence_scsi):             Started node01
  * fence2-scsi  (stonith:fence_scsi):             Started node02
～略～
```

「Started node01」となっていればそのリソースは node01 で起動しています



クラスタリソースの設定

- 故障が発生しフェイルオーバーすると...

```
# pcs status --full
Cluster name: DB_Cluster
～略～
Node List:
  * Node node01 (1): OFFLINE
  * Node node02 (2): online, feature set 3.19.6

Full List of Resources:
  * Resource Group: pgsql-group:
    * filesystem1 (ocf:heartbeat:Filesystem):      Started node02
    * filesystem2 (ocf:heartbeat:Filesystem):      Started node02
    * filesystem3 (ocf:heartbeat:Filesystem):      Started node02
    * ipaddr      (ocf:heartbeat:IPaddr2):         Started node02
    * pgsql       (ocf:linuxhajp:pgsql):           Started node02
  * Clone Set: ping-clone [ping]:
    * ping       (ocf:pacemaker:ping):             Started node02
    * ping       (ocf:pacemaker:ping):             Stopped
  * Clone Set: storage-mon-clone [storage-mon]:
    * storage-mon (ocf:heartbeat:storage-mon):     Started node02
    * storage-mon (ocf:heartbeat:storage-mon):     Stopped
  * fence1-scsi  (stonith:fence_scsi):             Started node02
  * fence2-scsi  (stonith:fence_scsi):             Started node02
～略～
```

サービスリソースが node02 で
起動しました

node01 はSTONITHされたので
「OFFLINE」になってます

クローンリソースも node01 で
は「Stopped」になってます



Hyper-V特有の考慮点

実際に試してみる

実際にいろいろな故障を発生させると...

- ちゃんとフェイルオーバーはする
- STONITHも機能している



特に問題なさそう？

めでたし！めでたし！



...と思いきや

あれ？ 何かがおかしい？

Hyper-Vホスト故障時の挙動がおかしい

□ 現用ノード側のHyper-Vホストを故障させる

➤ 想定していた動作

- 現用ノードはOS停止する
- 待機ノードから現用ノードへのSTONITHが実施される
- 待機ノードへフェイルオーバーサービスを継続する

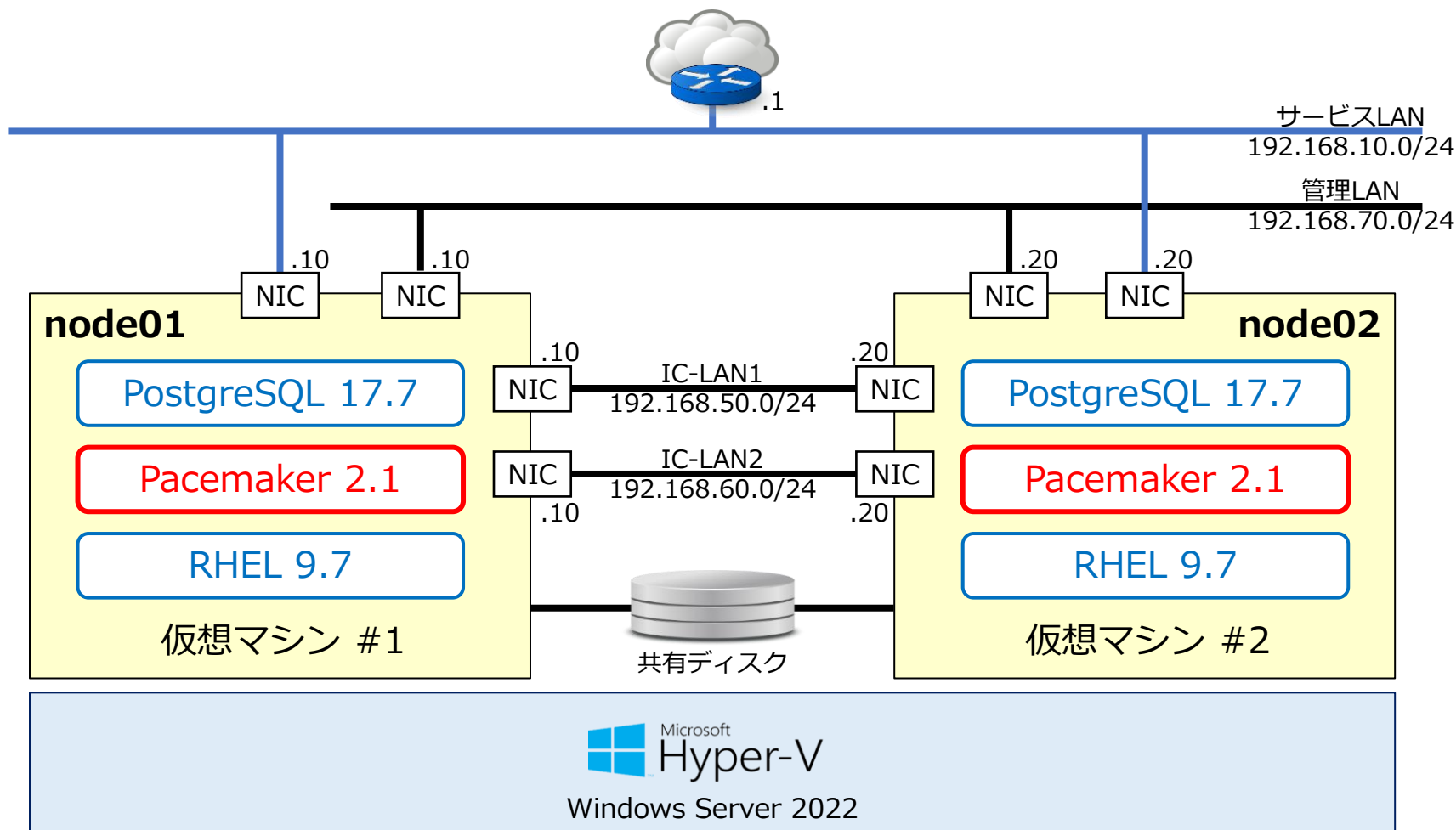
➤ 実際の動作

- 現用ノードはOS停止する
- 待機ノードにてディスク故障を検知
- 待機ノードでOS再起動が発生
- その結果、両系でサービス停止に...

なぜ待機ノードが??

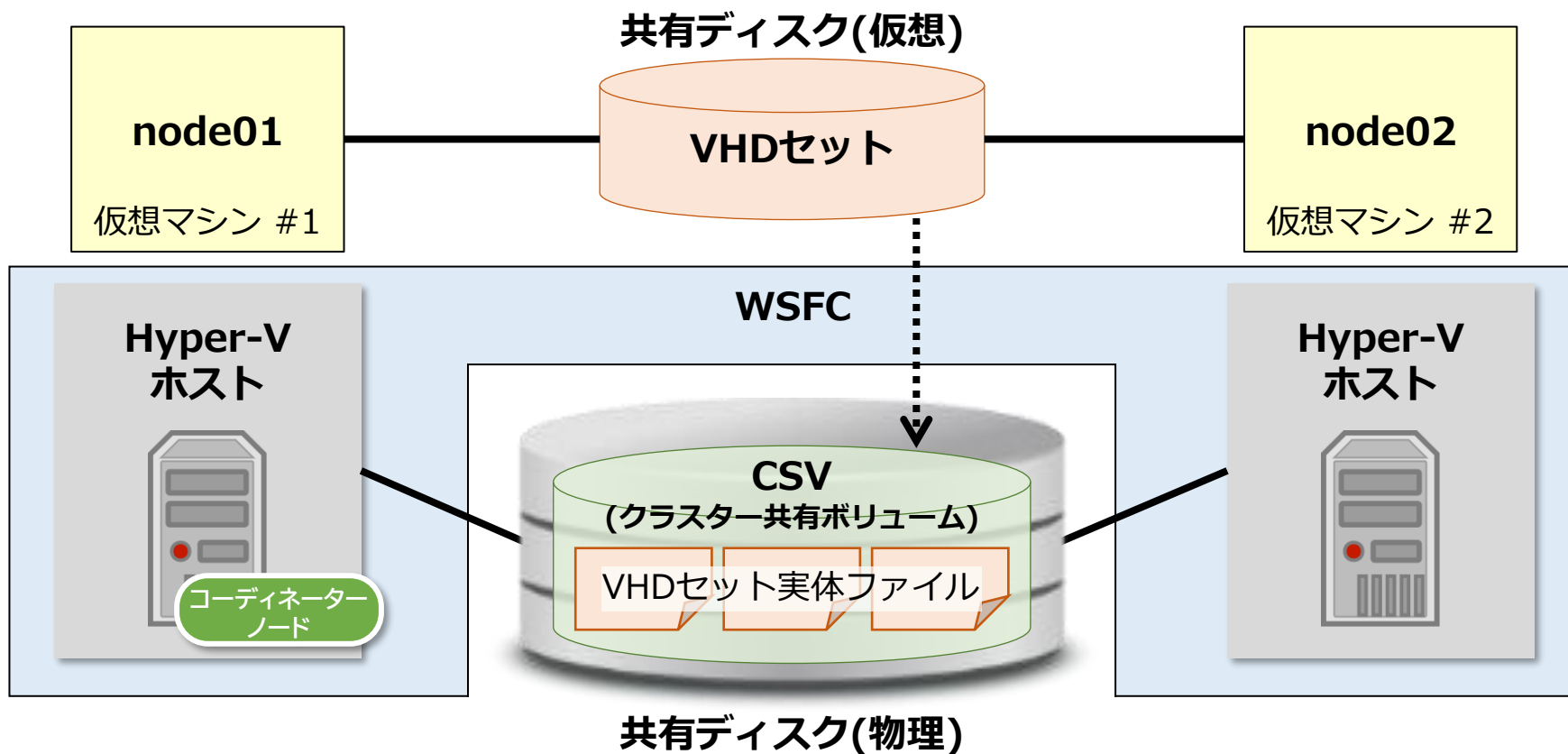


あらためて今回の環境構成



あらためて今回の環境構成

- Hyper-Vと共有ディスクまわりをもう少し詳しく



あらためて今回の環境構成

- WSFC (Windows Server Failover Clustering)
 - 複数台のHyper-Vホストをまとめて1台のホストのように扱う
 - 実際の仮想マシンはいずれか1台のHyper-Vホスト上で稼働
 - Hyper-Vホストが故障した場合、そこで稼働する仮想マシンは他のHyper-Vホストへ自動マイグレーション
- CSV (Cluster Shared Volume: クラスタ共有ボリューム)
 - 複数のHyper-Vホストから同時アクセス可能なディスク領域
 - VHDセットの実体ファイルなどが置かれる
- VHDセット (Virtual Hard Disk)
 - 仮想ディスクを複数の仮想マシンで共有できる
 - 仮想マシンの共有ディスクとして利用

コーディネーターノード

- CSVの物理ディスクリソースを管理するためのノード
- ファイルシステムのメタデータ（ファイル作成、削除、サイズ変更など）の更新処理を一手に引き受ける
- WSFCから1台のHyper-Vホストが選出される
- コーディネーターノードが故障した場合、別のHyper-Vホストが自動的にコーディネーターノードになる
- ただし、コーディネーターノードが再選出されるまでの間は、全ノードでCSVへのI/Oが遅延する

今回の問題

□ 待機ノードにてディスク故障を検知

- コーディネーターノードの故障により共有ディスクへのI/Oが遅延したことに起因
- I/O遅延がディスク監視(storage-mon)のI/Oタイムアウト閾値を超過したことでディスク故障が検知された

□ 待機ノードでOS再起動が発生

- コーディネーターノードの故障により共有ディスクへのI/Oが遅延したことに起因
- fence_scsiはSCSIリザーベーション機能を利用しており、共有ディスクに登録したSCSI予約キーを定期的にチェックしている
- I/O遅延によりSCSI予約キーのチェックがタイムアウトしたことで、状態異常とみなされwatchdogによりOS再起動された

じゃあどうする？

□ Pacemaker側でできる対策

- storage-monのI/Oタイムアウト閾値を延長することで、ディスク故障検知を回避する
 - storage-mon の io_timeout パラメータを延長
 - 故障発生～検知までの時間が伸びるため、F/O時間も延びる

#表 7-3-1 クラスタ設定 ... Primitiveリソース (id=storage-mon)

PRIMITIVE						
P	id	class	provider	type		
#	リソースID	class	provider	type	概要	
	storage-mon	ocf	heartbeat	storage-mon	ディスク監視	
A	type	name	value			
#	パラメータ種別	項目	設定内容	概要		
	options	drives	/dev/internal_a /dev/external_b /dev/external_c /dev/external_d			監視ディスクデバイス名
		io_timeout	25			ディスクI/Oのタイムアウト時間(秒)
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
	start	120s		restart		
	monitor	120s	10s	restart		
	stop	120s		fence		

じゃあどうする？

□ Pacemaker側でできる対策

- SCSI予約キーチェックのタイムアウト閾値を延長することで、watchdogによるOS再起動を回避する
 - watchdog の test-timeout パラメータを延長
 - STONITH所要時間が伸びるため、F/O時間も延びる



各パラメータをどれくらいに設定すればよいかは、環境ごとにコーディネーターノード故障時のI/O遅延時間を確認して決める必要があります

パラメータを延長するとF/O時間も延びるので、故障発生時のサービス停止期間が長くなる点にも注意が必要



じゃあどうする？

□ Hyper-V側でできる対策

- コーディネーターノード故障時のI/O遅延時間を短縮する
 - WSFCの "SameSubnetDelay" と "SameSubnetThreshold" のパラメータを短縮
 - Hyper-Vホストの故障発生～検知までの時間を短縮することで、I/O遅延時間の短縮を図る
 - やりすぎると故障でないのに故障と誤検知されてしまう
- 共有ディスクにCSVを使わずFCパススルーを利用する
 - CSVを使わないのでコーディネーターノード自体が存在しない
 - ただし、ライブマイグレーションができなくなる



いずれの方法も一長一短あるね

何をどこまで許容できるか、要件に応じた対応が必要ですね



ざっくりまとめると

- Hyper-V上でもPacemakerの高可用クラスタは動きます
- ただし、コーディネーターノードの故障に対応するためには、パラメータのチューニングやHyper-V自体の構成を考慮する必要があります
- Hyper-V上だと RHEL HA Add-On が Red Hat社のサポート対象外となる点にも注意

おわりに

Pacemaker をもっと知りたい方はこちらまで

<https://linux-ha-japan.github.io/>

技術的な質問や情報交換などがあれば、
ぜひ **Linux-HA Japan ディスカッション** をご活用ください！
<https://linux-ha-japan.github.io/discussions/>



本発表で詳しく紹介できなかったリソースの設定についても紹介されているので、
気になる方はぜひ見てみてください



今後も Linux-HA Japan を よろしくお願いします

