高可用性システムを実現するための Pacemaker最適設計

2017年9月9日

Linux-HA Japan プロジェクト
http://linux-ha.osdn.jp/
水谷 浩二

もくじ

- Pacemaker概要
- 高可用性システム実現プロセス
- HA方式検討／クラスタ設計のポイント
- テスト／運用監視の実施ポイント
- まとめ
- おまけ

Pacemakerとは

- オープンソースのHAクラスタソフトです。

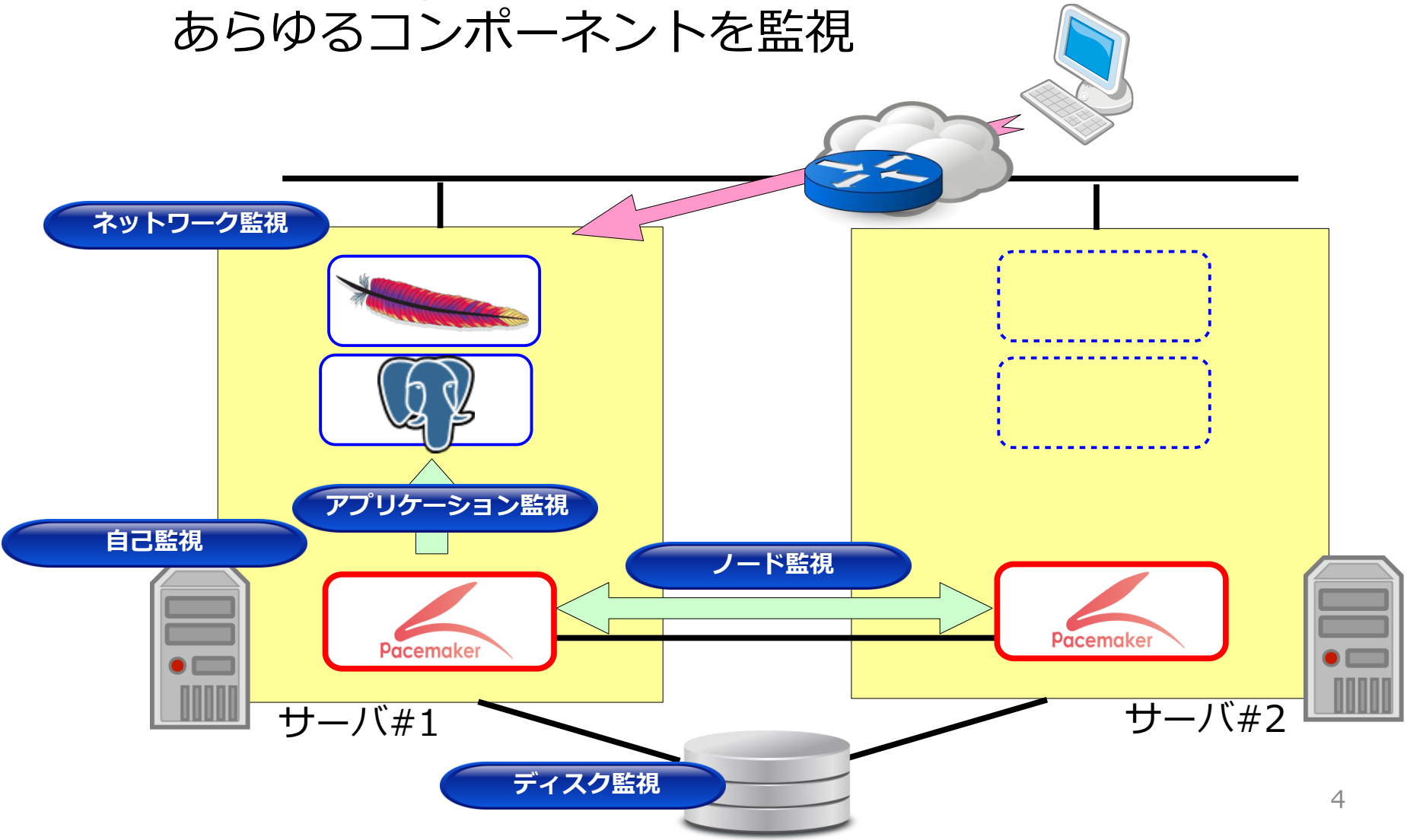
- High Availability = 高可用性
  つまり

  一台のコンピュータでは得られない、高い稼働率のシステムを実現するために、複数のコンピュータを結合(クラスタ化)し、ひとまとまりとする…

  ためのソフトウェアです。

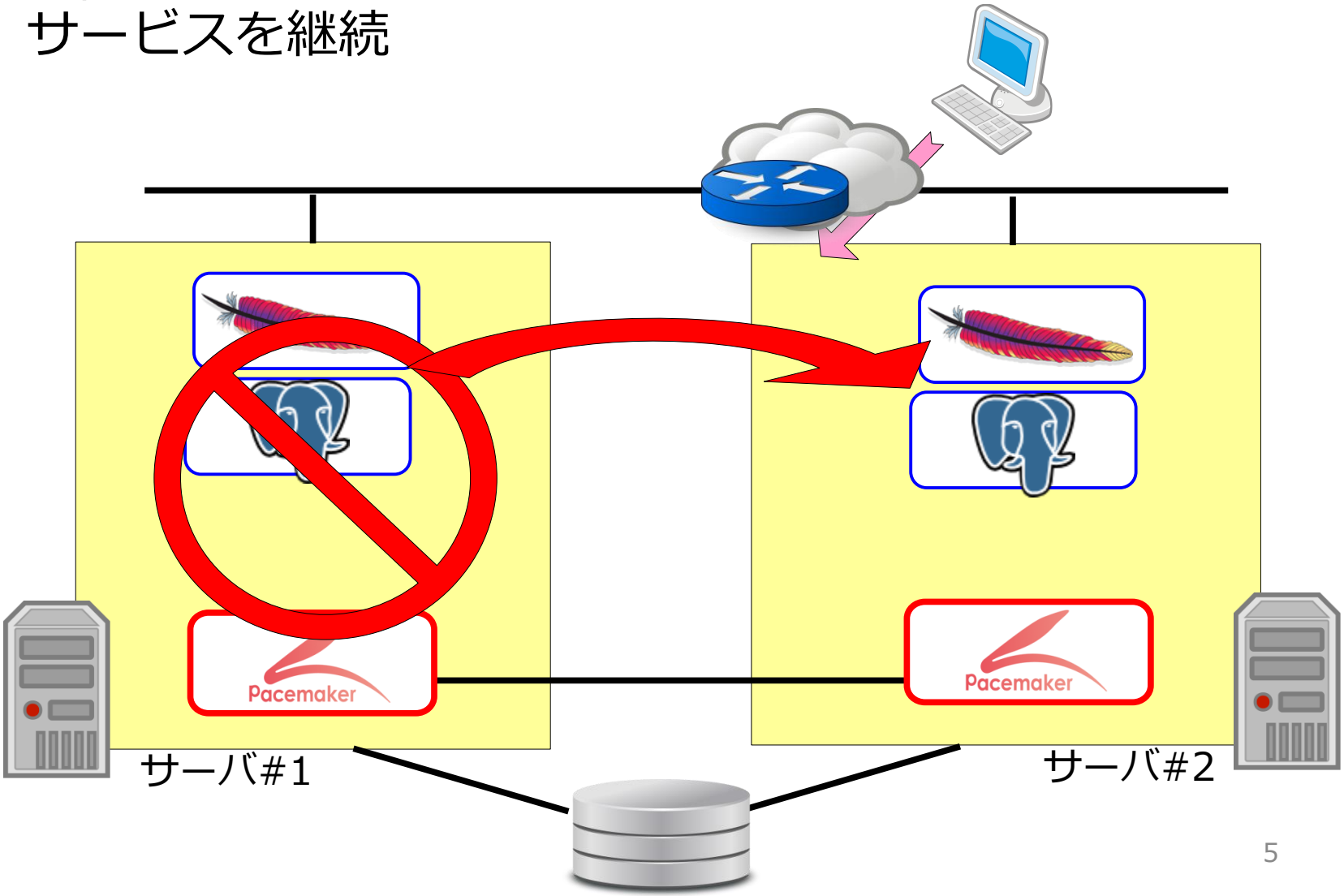
Pacemaker動作イメージ（1/2）

- サービス提供に必要な
  あらゆるコンポーネントを監視



Pacemaker動作イメージ（2/2）

- 故障が発生するとフェイルオーバして
  サービスを継続

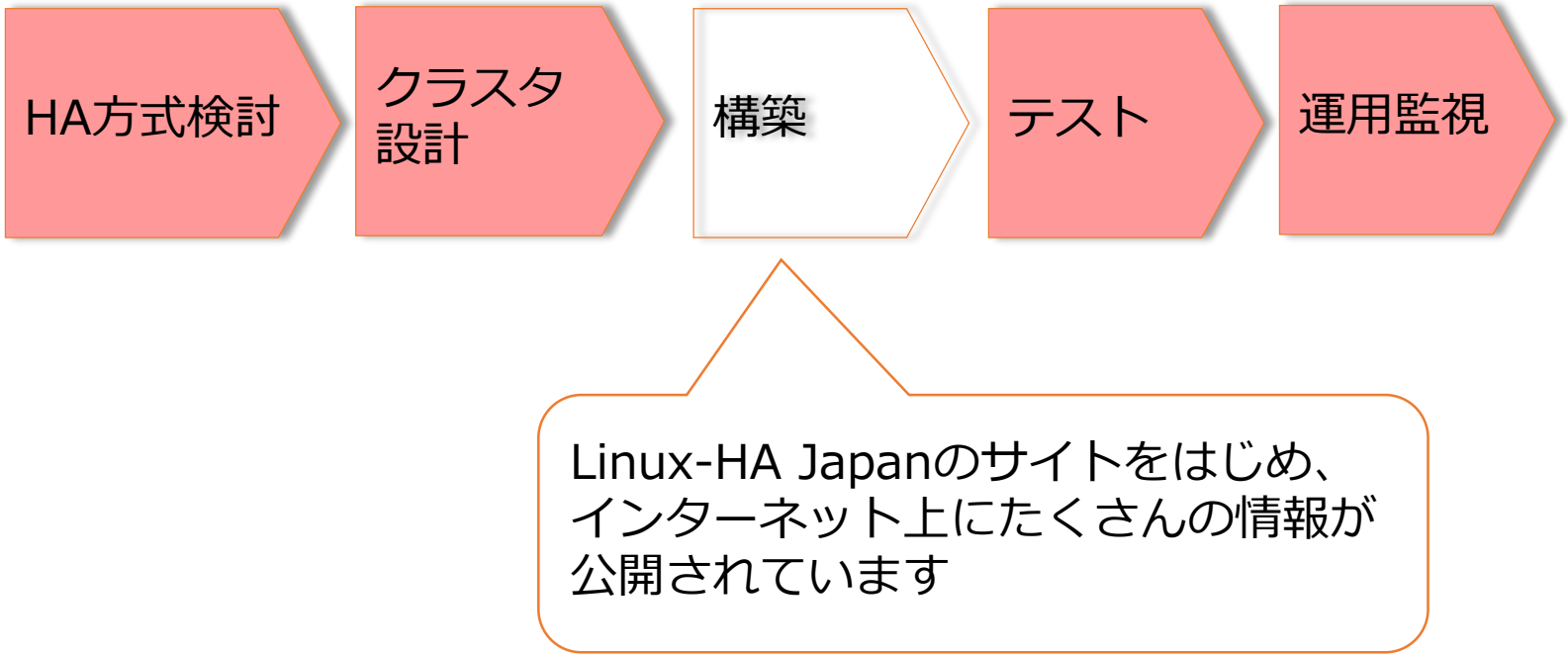


高可用性システム実現プロセス

- Pacemakerで高可用性システムを実現するには、次のプロセスを適切に実行する必要があります
  - Pacemakerに限った話ではありません。プロプラのHAクラスタソフトでも同じです
  - これを怠ると・・
    - 故障したときにフェイルオーバに失敗してしまった
    - 故障していないのにフェイルオーバしてしまった
    - フェイルオーバしたが、復旧手順がわからない

高可用性システム実現プロセス

- 本講演では、「構築」を除く各プロセスでの実施ポイントをお話します



✓ 本資料での「構築」は、Pacemakerの環境定義書や設定ファイルを作成し、Pacemakerをマシンにインストールしてサービスを起動させるまでの作業範囲を意味しています。

高可用性システム実現プロセス

Pacemaker

HA方式検討

クラスタ設計

構築

テスト

運用監視

HA方式の検討ポイント

- そもそもHAクラスタ構成とする必要はありますか？
  - HAクラスタ構成とするには、相応の設計や構築、運用コストが必要になります
  - この増加するコストと、サービス停止時間によるビジネスインパクトを比較
    - 影響が小さいサーバやミドルウェアは、シングル構成で割り切ることも視野に入れる
  - 基盤側サービスとして、例えば、vSphere HA 機能によりサーバ/OS故障の範囲に対して一定の可用性が実現されるような環境もあります
    - サーバ/OS故障の範囲では不足で、ミドルウェアなども含めた可用性の向上が必要か？

HA方式の検討ポイント

- Pacemakerを適用するサーバ／ミドルウェアを選別

  Pacemaker以外によって一定の可用性が確保される方式があります

  - HAクラスタソフト以外によるHA構成化

    （例）ロードバランサで振り分け先として監視されるApache

  - ミドルウェア自身やコンポーネントとの組合せでHAクラスタ機能を実現

    （例）Oracle RAC (Oracle Clusterware)、
    Tomcat + mod_jk / mod_proxy_balancer

  - クラウドサービスとしてHAクラスタ構成を提供

    （例）Amazon RDSサービス

  ➢ これらに該当しないものは、PacemakerによってHAクラスタ構成を実現します

高可用性システム実現プロセス

Pacemaker

HA方式検討

クラスタ設計

構築

テスト

運用監視

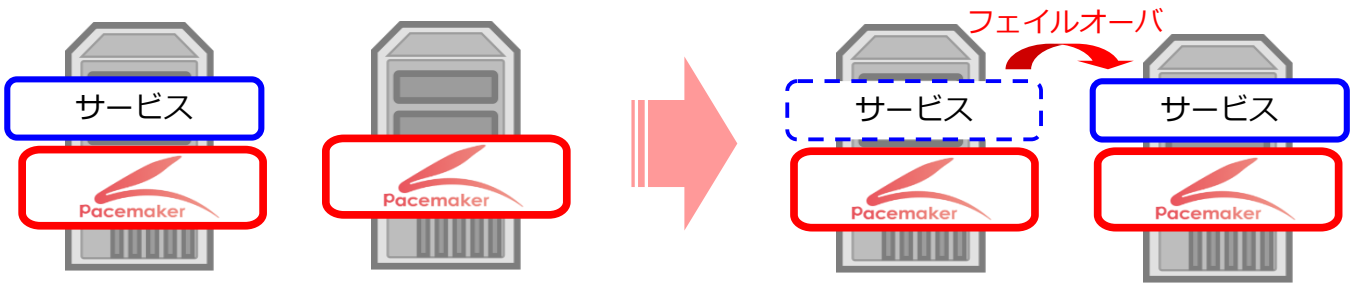
クラスタ設計のポイント

- ノード構成
- 監視・制御ソフトウェア
- データ共有方式
- 基盤環境

ノード構成

- Pacemakerでは、1+1 (Act-Sby) 構成をはじめ、1+1クロス (Act-Sbyたすきがけ) 構成、N+1構成、N+M構成といったすべてのノード構成パターンを実現できます

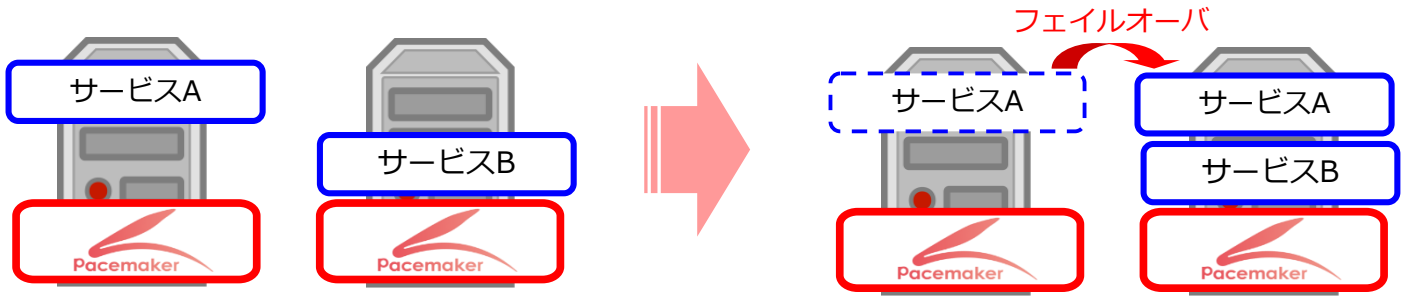
- 特に制約がなければ、1+1 (Act-Sby) 構成がお勧め
  - もっとも実績が多い
  - 設計と運用がシンプル（コストがもっとも小さく済む）
  - クラウド環境のようにノード（インスタンス）の利用コストが小さい環境では、この構成を選択するのが望ましい



ノード構成

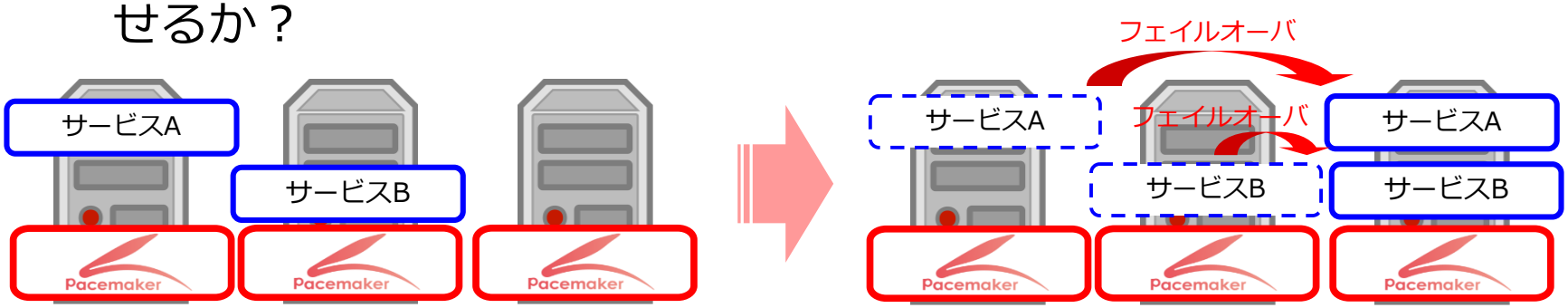
- ノード数を削減したい要望があるとき、1+1構成以外を検討します

- 1+1クロス構成は、フェイルオーバ時に1ノードで複数のサービスを提供することになるため、マシンリソースの設計に注意が必要

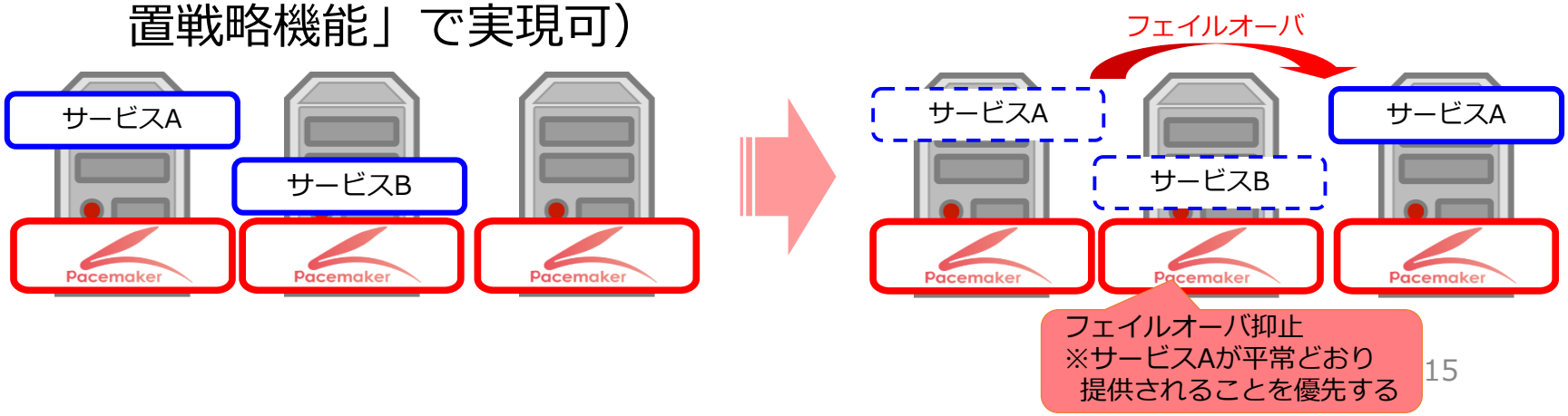


ノード構成

- N+1構成は、多重故障（2ノード以上の故障）時のフェイルオーバポリシーを検討しておく
  - 2ノード目が故障したとき、待機系で複数サービスを提供させるか？



  - 2ノード目はフェイルオーバを抑止するか？（「リソース配置戦略機能」で実現可）



監視・制御ソフトウェア

- まずは、Pacemakerで監視・制御したいソフトウェア（ミドルウェア）を洗い出します

- 次に、対応する監視・制御スクリプト（リソースエージェント (RA)）をPacemakerが提供しているかを確認します
  - 提供RAは、OSSを中心として、市場でよく利用されるプロプラ製品 (Oracle など) にも対応しており、比較的充実しています
  - RAの種類は、コミュニティサイト (GitHub) か、Pacemakerインストール環境で確認できます
    - https://github.com/ClusterLabs/resource-agents
    - /usr/lib/ocf/resource.d/heartbeat 配下
    - RAのファイル名 (pgsql, nfsserver) で、おおよそ管理対象ソフトウェアをイメージできます（が、正確には中身を参照する必要あり）
  - 開発が止まっているRAは、ソフトウェアのバージョンアップに追従できていなかったり、品質に課題がある場合あり
    - 事前の異常系を含んだ動作テストは必須

監視・制御ソフトウェア

- RAが未提供の場合や、RAの品質に課題が確認されたような場合、 RHEL7/CentOS7では「Systemd形式」で対応できるかを検討します(※)
  - Pacemakerが、次のコマンドに相当する処理を内部的に実行する
    - 監視: systemctl status <ユニット名>.service
    - 起動: systemctl start <ユニット名>.service
    - 停止: systemctl stop <ユニット名>.service

    ✓ 実際は、Pacemakerがsystemd D-Bus APIを直接利用して実行する

  - Pacemaker提供RAと比較して、監視レベルが劣る傾向あり
    - Systemd形式による監視は、プロセス監視レベルとなる
    - Pacemaker提供RAでは、ソフトウェアの内部動作までをチェックするものが多い（プロセスのハングアップのような故障も検知可能）
      - PostgreSQL用RA (pgsql): SQL実行
      - Apache用RA (apache): HTTP GET

(※) RHEL6/CentOS6では、LSB形式（initスクリプト）で対応できるかを検討します

監視・制御ソフトウェア

- PacemakerがRA未提供で、Systemd形式でも対応できないとき、独自にRAを作成することを検討します
  - ソフトウェアの仕様調査
    - プロセス構成の把握
      - 複数プロセスの場合、それぞれのプロセスが故障したときの動作仕様を把握する
    - 制御インターフェイスの洗い出し
  - RA設計・作成
    - Pacemakerが提供しているRAを流用するとよい
    - 異常系の考慮
      - どこまで異常系を考慮した設計となっているかによって、可用性が変わってくる
  - RA単体試験
    - Pacemakerと組み合わせる前に、事前に異常系を含めた試験を行う

✓ Act/Sby型のRAを対象とした説明です。Master/Slave型RAを個別のシステムで独自に作成することは、極めて高度なスキルと膨大な試験が必要であり、現実的ではありません。

監視・制御ソフトウェア

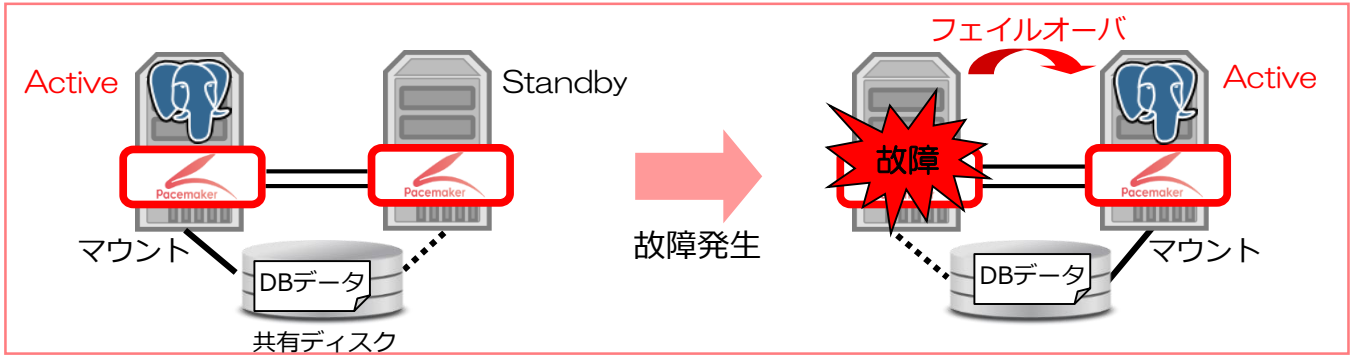
- サービス提供に直接影響しないミドルウェアなど、故障したときに、即座にフェイルオーバさせる必要がないソフトウェアがないかを識別し、アクションを検討します
  - （例）運用監視エージェント
    - 監視ができなくなるが、サービス提供には影響を与えない
  - フェイルオーバ以外のアクション種別
    - 監視を行わない
      - crm ファイルの op 制御部 monitor を設定しない
    - 監視するけど、同じノードで再起動を複数回試行する
      - migration-threshold で再起動の試行回数を指定

データ共有方式

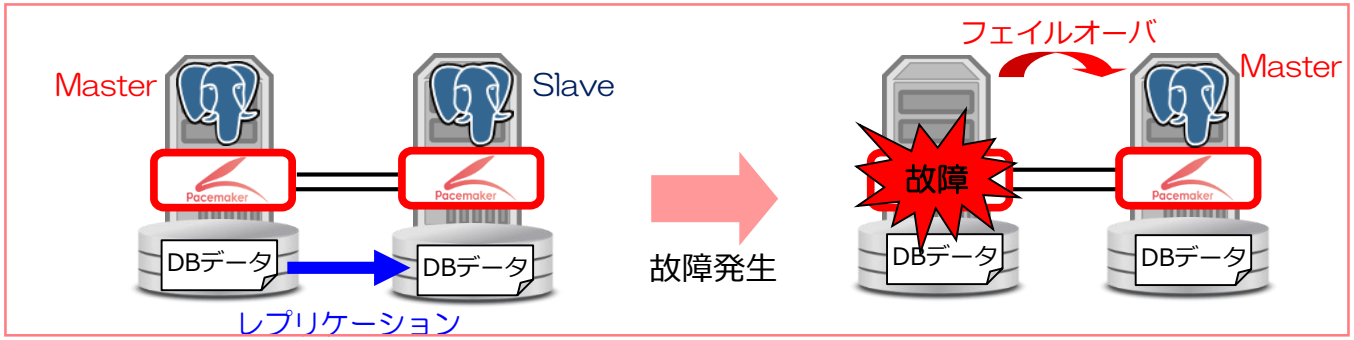
- フェイルオーバしたときに、データを引き継いでサービス提供する必要がある場合、データ共有方式を検討します
  - DBMSサーバ（PostgreSQL、MySQL、Oracle など）
  - NFSサーバ
  - メールサーバ
  - 独自アプリケーション...etc.

データ共有方式

- データ共有方式は、大きく次のとおり分類されます
  - 共有ディスク型
    - 共有するデータ容量が大規模の場合はこちらがお勧め
    - 実績多数、運用がシンプル



  - シェアードナッシング型
    - 共有ディスクが不要なため、設備コストは低くなる
    - 設計・運用コストが、共有ディスク型と比較して高くなる
    - 最近では、クラウド環境など、共有ディスクが利用できない基盤で選定される事例が多数



共有ディスク型

- FC接続
  - 実績多数で、最も信頼性が高い
  - 設備費用 (HBA、FCスイッチ) が高くなる傾向
  - 中～大規模システム向け
- iSCSI接続
  - FC接続より設備費用は低く抑えられる
  - 小～中規模システム向け
- NFS接続
  - スプリットブレイン対策機能 sfex が利用不可
    - ほかスプリットブレイン対策機能 (STONITH、VIPcheck) の利用が必須
  - DBMSデータ領域以外の用途で利用される（帳票ファイル、音声データの保存など）

シェアードナッシング型

- PostgreSQLであれば、PG-REXの利用がお勧め
  - 運用補助ツールが提供されており、運用コストが抑制される
  - 詳しくはPG-REXコミュニティサイトへ
    https://ja.osdn.net/projects/pg-rex/
  - 最新情報は、2017年7月OSC2017 Kyoto講演資料で細かく紹介されています
    http://linux-ha.osdn.jp/wp/archives/4627
    - 試して覚えるPacemaker入門 PG-REX(Pacemaker + PostgreSQLによるシェアードナッシングHA構成)構築(PDF)
- DBMSではない、通常のファイルを共有する場合はDRBDを利用

基盤環境

- 最近のシステムは、様々な基盤環境上に構築されます
  - オンプレミス
  - プライベートクラウド
    - 本資料では、別の組織がハードウェア～OSをサービスとして提供・管理する形態を意味しています
    - 社内システムを集約した共通基盤なども含みます
  - パブリッククラウド
  - （コンテナ）
- システム基盤の仕様によって、Pacemaker設計は様々な影響を受けます

✓ 基盤上に構築するゲストシステムへPacemakerを導入することを前提とした説明です（基盤環境の高可用性については、説明対象外です）。

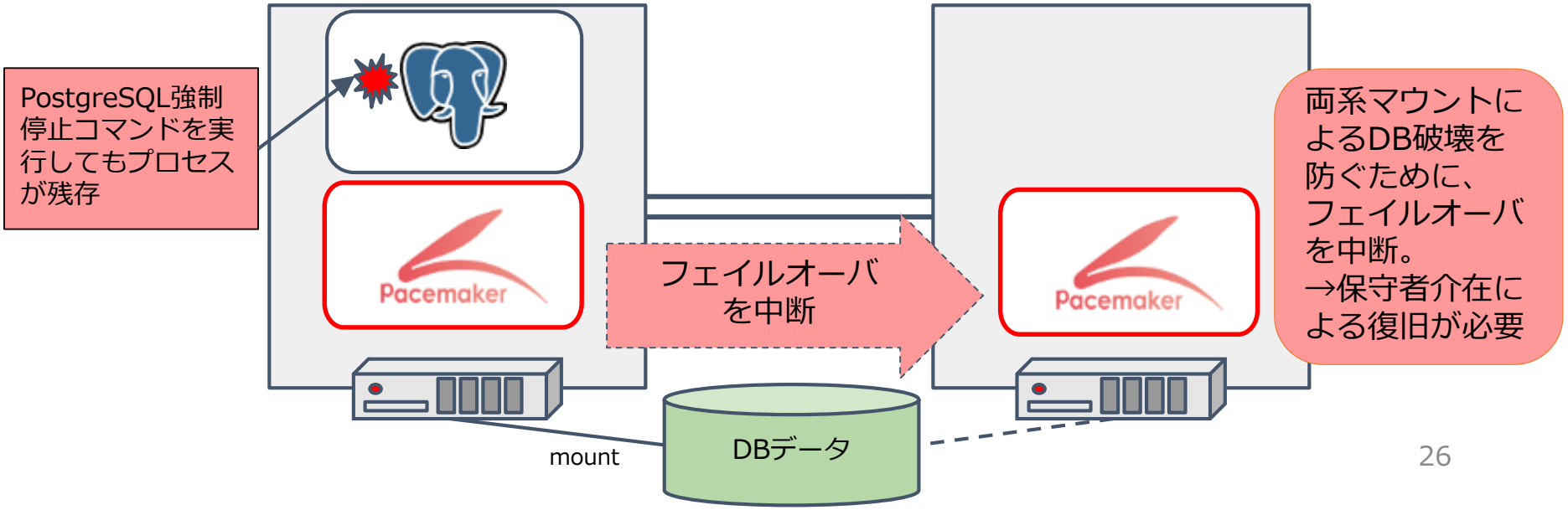
オンプレミス

- 制限が小さいことから、可用性を向上させるためにハードウェア／ネットワーク設計において、次のポイントを検討します
  - STONITH
  - ネットワーク構成

オンプレミス

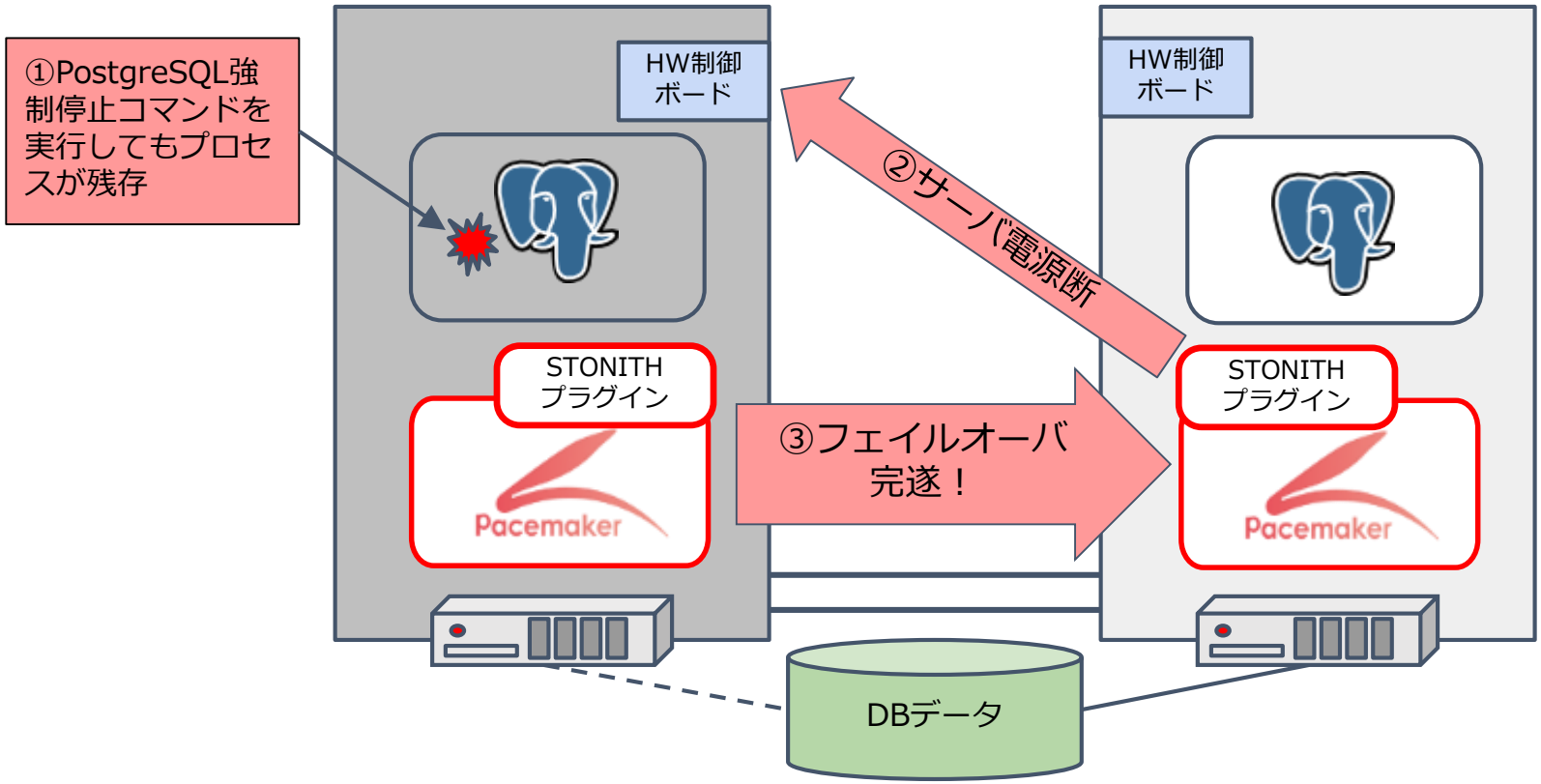
- STONITH
  - STONITH を利用しない場合、故障発生ノードが閉塞ができないとき (リソース停止処理に失敗したとき) に、保守者介在による復旧が必要となります
  - よって、STONITHを利用した方が可用性は高くなります

- STONITHがないとき



オンプレミス

- STONITHがあるとき



オンプレミス

- STONITH導入のポイント
  - IPMI対応のハードウェア制御ボードがあるサーバを選定
    - HP iLO、富士通 iRMC S2
    - だいたいのサーバには搭載されている
  - サーバ相互にハードウェア制御ボードを結線
    - 保守LANを活用することが多い

✓ 物理環境を前提とした説明です。仮想環境では、利用するハイパーバイザによりSTONITH導入のポイントが異なってきます（後ろの「プライベートクラウド」で一部説明）。

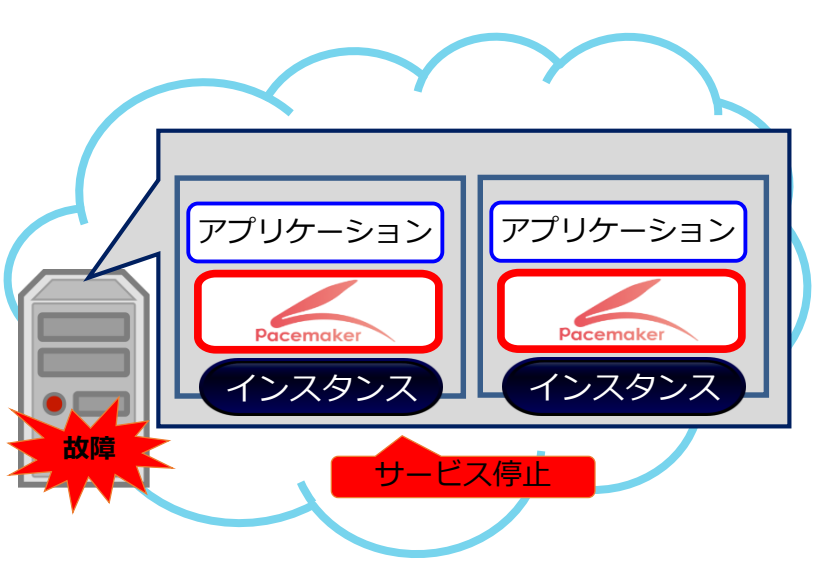
- ネットワーク
  - ハートビート通信LANの2重化
  - シェアードナッシング型ではレプリケーションLANの設置
  - 保守LANの設置

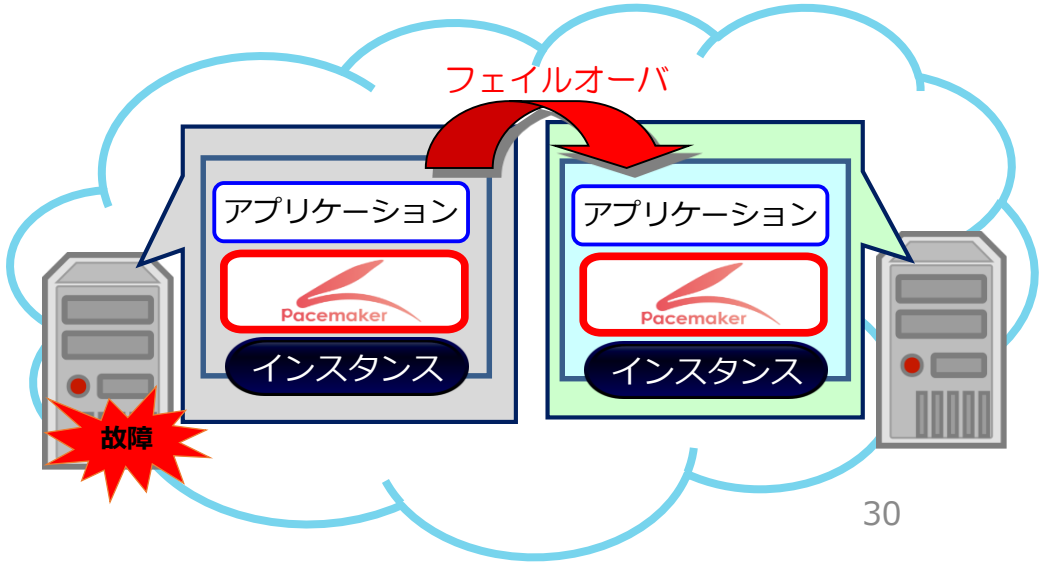
プライベートクラウド

- プライベートクラウド（社内システム共通基盤などを含む）では、次の観点で基盤仕様を確認します
  - インスタンス配置
  - 共有ディスク利用可否
  - ネットワーク設計
  - STONITH利用可否
  - 仮想IP制御

プライベートクラウド

- インスタンス配置
  - HAクラスタのノードとなる複数の仮想マシンを、それぞれ異なるホスト（物理サーバ）上で提供してもらえるか？
    - 同じ物理サーバ上で複数の仮想マシンをHAクラスタ構成とした場合、物理サーバ故障でサービス停止となってしまうため、可用性が下がる
  - 基盤メンテナンスで仮想マシンがマイグレーションされたようなケースでも、それぞれ別ホスト上に配置されることが保証されるか？





プライベートクラウド

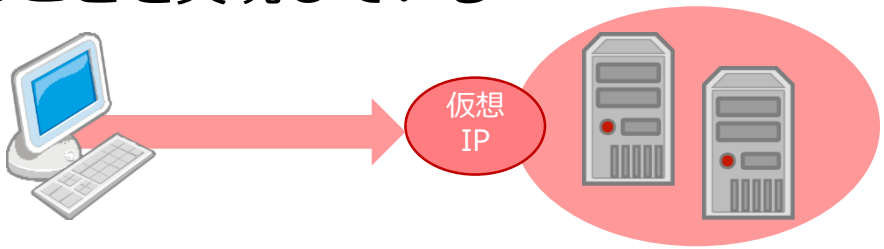
- 共有ディスク利用可否
  - 共有ディスクを利用できるサービスが提供されているか？
    - 未提供のときはシェアードナッシング型で検討する
- ネットワーク設計
  - サービスLAN以外に、次のような個別ネットワークを払いだしてもらえるか？
    - ハートビート通信LAN（2系統）
    - シェアードナッシング型ではレプリケーションLAN
  - 物理的なネットワークを論理的に分割し、個別のネットワークとして払いだされるような基盤においては、可用性の観点では、個別ネットワークを複数設置する必要性はない
    - 物理的なネットワークは、bondingなどにより基盤側で可用性が確保されていることがほとんど
    - ハートビート通信やレプリケーションなどが同じ物理ネットワークに重畳する形となるため、ネットワーク負荷に問題がないかの事前テストが重要になる

プライベートクラウド

- STONITH利用可否
  - 基盤の設計や運用ポリシーにより、利用可否が決まる
  - VMwareとKVMでのSTONITH方式（libvirtプラグイン）
    - VMware
      - virshコマンドにより、 vCenter経由で対向ホストのVMを強制停止させるイメージ
      - Pacemaker が動作するゲストOSから、vCenter へのシステム管理者権限を有するアカウントでのログインが許可される必要あり
    - KVM
      - virshコマンドにより、SSHトランスポートを使用して対向ホストに接続し、VMを強制停止させるイメージ
      - VMのホスト配置が固定されている必要あり
      - VMからホストマシンに対して、仮想マシンの操作権限を持ったユーザによるノンパスワードでsshログインできる必要あり
  - STONITHが利用できない場合は、別のスプリットブレイン対策機能を利用する
    - 共有ディスクがない場合は、VIPcheck機能を使う

プライベートクラウド

- 仮想IP制御
  - Pacemakerでは、IPaddr2 RAが仮想IPを制御することで、HAクラスタ複数ノードを、クライアントから透過的に1台に見せることを実現している



  - IPaddr2 RAの仮想IP制御方式イメージ
    - ipコマンドでノードに仮想IPを追加し、Gratuitous ARP (ARP REQUEST) を送信して、同じセグメントのネットワーク機器のARPテーブルを更新
  - 基盤担当へ、上記IPaddr2 RAの仮想IP制御方式を伝え、動作可否や、必要な事前作業がないかなどを確認しておく
    - 基盤のネットワーク仕様（例: 特殊なSDN製品を使っている）によっては、特別な対応が必要となる場合がある
    - 仮想IPによる通信を許可するために、事前にセキュリティ関連の対処が必要な場合がある

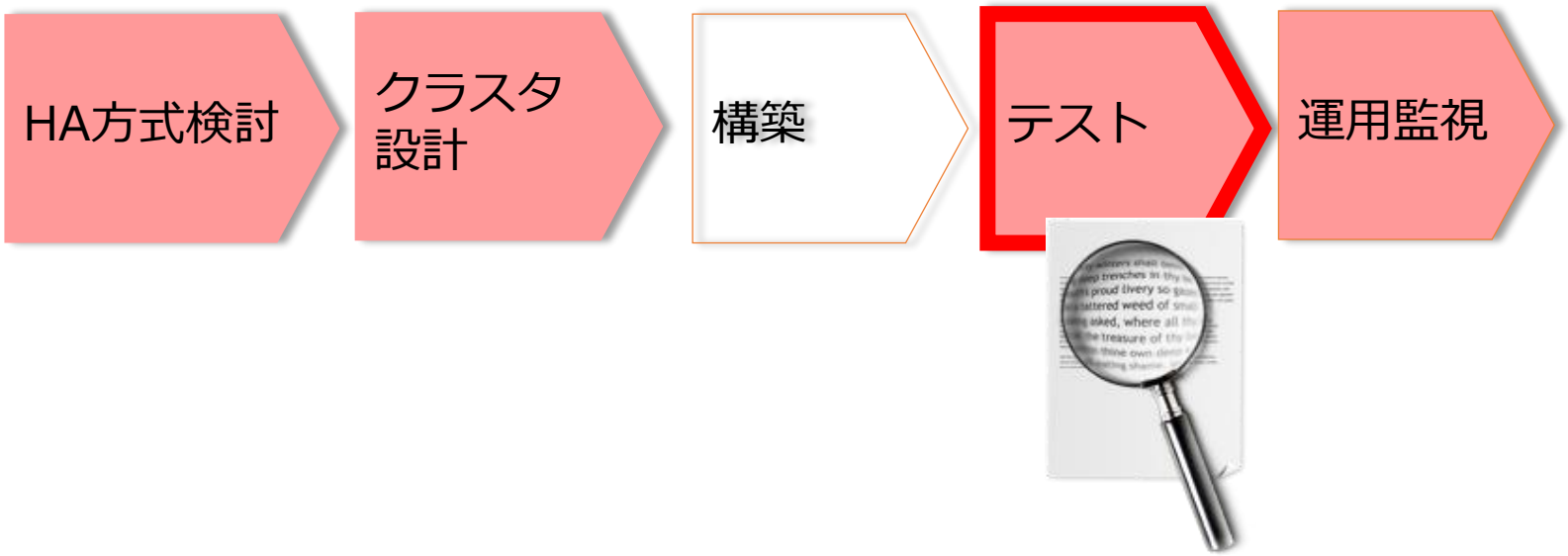
パブリッククラウド

- 「プライベートクラウド」と検討ポイントは同じ
- AWSに対する検討例
  - インスタンス配置
    - 異なるホストにインスタンスが配置されるよう、各インスタンスをそれぞれ異なるAvailability Zone に設定
  - 共有ディスク利用可否
    - シェアードナッシング型とするのが一般的
    - Amazon EBS は Availability Zone を跨いだ付け替えができない
  - ネットワーク設計
    - 個別ネットワークは、物理的に同じネットワークを論理的に分割して払い出される、複数設置しない設計もあり
  - STONITH利用可否
    - Pacemaker より Amazon EC2 用 STONITH プラグイン (ec2) が提供されている
  - 仮想IP制御
    - IPaddr2 RAでは実現不可
    - Availability Zoneを跨ぐため、Amazon VPC の RouteTable や、Amazon Route 53 の設定を監視・制御するRAの個別作成が必要(※)

(※) Amazon Route 53 の設定を監視・制御する aws-vpc-route53 RA がコミュニティで開発中
https://github.com/ClusterLabs/resource-agents/pull/1003

✓ 利用サービスや条件で異なる可能性があり、検討時はあらためてAmazon社へ確認してください

高可用性システム実現プロセス



テストの実施ポイント

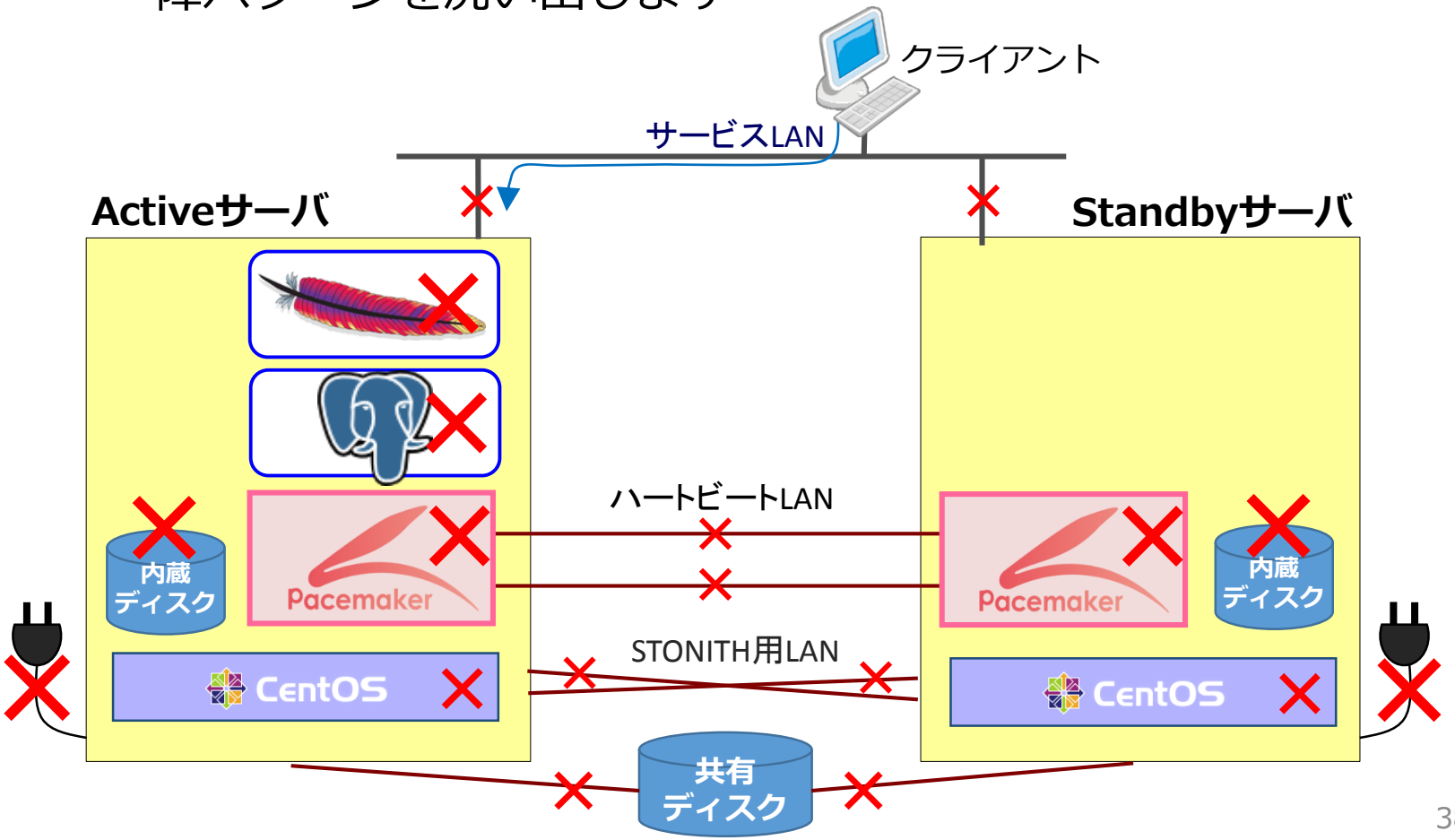
- 商用サービス提供中、メンテナンスや故障発生時に、適切に動作するかを試験します
- テスト内容
  - メンテナンス試験
  - フェイルオーバ試験
  - 業務負荷をかけた長時間安定動作試験
- テストでは、 crm_monコマンドによるPacemakerの状態確認だけでなく、クライアントからアクセスして、サービス継続性をあわせて確認することが推奨されます
  - クライアントからのアクセス確認により、仮想IPの切替えから各ソフトウェア（ミドルウェア）を通じての正常性を一括で確認できます

テストの実施ポイント

- メンテナンス試験
  - オペレータによるコマンド実行で、スイッチオーバ／スイッチバックが正しく動作することを確認します
    - crm_standby コマンド
    - crm_resource –M コマンド

テストの実施ポイント

- フェイルオーバ試験
  - サービス提供に必要なコンポーネント単位で、想定される故障パターンを洗い出します



テストの実施ポイント

- 洗い出したパターンで実際に故障させ、サービス継続性を確認します
  - ソフトウェア故障は、強制停止コマンドの実行やプロセスにSIGKILLを送信して発生させる
  - ハードウェア故障は、基盤環境で発生させることが困難な場合や、商用環境で本当に故障してしまうことを避けたいときには、擬似的に故障を発生させる
    - 内蔵ディスク故障は、擬似が困難なため商用環境では省略することがほとんど

テストの実施ポイント

Pacemaker

- ハードウェア擬似故障の発生方法
  - ネットワーク故障
    - iptables コマンドによるパケット・ドロップで擬似

```
# iptables -A INPUT -i eth1 –j DROP ; iptables –A OUTPUT –o eth1 -j DROP
```

    - ifdown/ifup など、ネットワーク構成をLinux OSのレイヤで変更させる方法は不適（実際のネットワークインターフェイス故障とは異なる振る舞いになってしまう）
  - ノード故障
    - reboot –f コマンドで擬似
    - shutdown –r/-h といったコマンドでは、Linux OSのシャットダウン処理が実行されるため、実際のノード故障（サーバ電源断など）とは異なる振る舞いとなってしまう

テストの実施ポイント

- 故障箇所を復旧し、クラスタを平常状態に戻します
  - フェイルオーバした後の復旧までを確認する
  - 復旧手順の確立ができる
    - 復旧手順は、商用サービス開始後に必要となるため、保守運用チームへ共有を
- 想定される業務負荷をかけた長時間安定動作試験
  - Pacemakerでは多くのタイムアウト・パラメータがある
  - すべてのパラメータをシステム個別に検討することは現実的ではないため、ひとまずコミュニティ推奨値で設定しておく
  - 想定される業務負荷をかけて長時間（例：24時間、48時間）運転し、フェイルオーバしないかを確認する
    - ここでタイムアウトなどが発生した場合、パラメータ・チューニングを検討する

高可用性システム実現プロセス

Pacemaker

HA方式検討 → クラスタ設計 → 構築 → テスト → 運用監視

運用監視の実施ポイント

- 運用監視を行わないと、いざ故障したときにフェイルオーバできない（サービス停止）となる可能性があります
  - いつの間にかフェイルオーバしていた
  - 待機系が故障していた
- 保守運用者が運用中、Pacemaker状態表示コマンド(crm_mon)やログを常時監視することは現実的ではありません
- そこで、次のような方式で運用監視します
  - 運用監視エージェントによるログ監視
  - Pacemaker SNMPトラップ通知機能

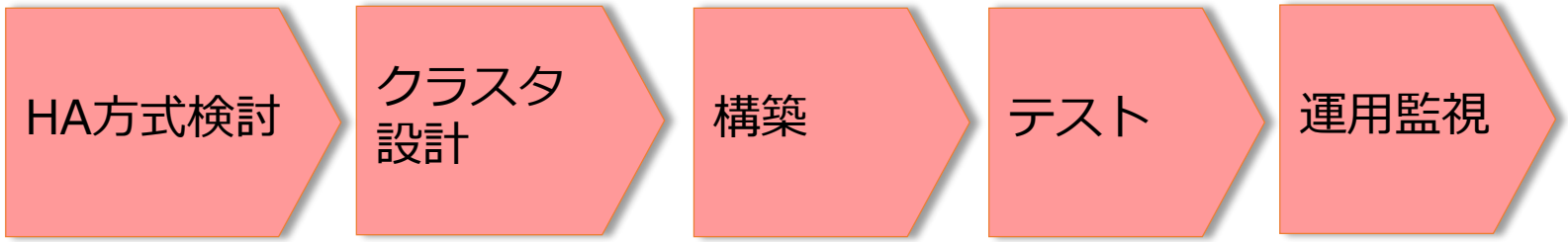
運用監視の実施ポイント

- 運用監視エージェントによるログ監視
  - /var/log/pm_logconv.out のメッセージを監視
    - メッセージ一覧などはコミュニティサイトで公開
      https://github.com/linux-ha-japan/pm_logconv-cs
    - 故障検知やフェイルオーバ発生などのメッセージが出力されたときに、運用監視エージェントが捕捉してマネージャへ通知する
  - 運用監視エージェントを導入するため、システム全体の監視が可能
    - サーバのリソース（CPU、メモリ、ディスク）使用量監視
    - /var/log/messagesや、ほかミドルウェアなどの ERROR / WARNING メッセージ
  - 中～大規模システム向け
- Pacemaker SNMPトラップ通知機能
  - 運用監視エージェントの導入が不要
  - 小～中規模システム向け

Linux-HA Japan プロジェクト デモ展示中！

まとめ

- 各プロセスの実施ポイントを押さえ、これらを適正に実施することで、高い可用性のシステムを実現しましょう



おまけ

- Pacemakerリリース状況（2017年8月末時点）
  - Linux-HA Japanプロジェクトでは、2017/06/09 に最新版リポジトリ・パッケージ「Pacemaker 1.1.16-1.1」をリリース
  - 本家コミュニティ（ClusterLabs）では、2017/07/07 に「Pacemaker1.1.17」がリリース

おまけ

- Pacemaker1.1.17のトピック
  - 新機能 “bundle”
    - Dockerコンテナの可用性を高める新しい管理機能
    - 内部的には、pacemaker_remote、docker RAおよびIPaddr2 RA といったコンポーネントが利用される
    - ステートフルコンテナの可用性向上での活用に期待
    - 詳しくはこちら https://wiki.clusterlabs.org/wiki/Bundle_Walk-Through
  - Linux-HA Japanプロジェクトでは、1.1.17対応リポジトリ パッケージのリリースに向けて活動中
    - PostgreSQL10を採用したPG-REX10への対応を検討中
- Pacemaker1.1.18以降のトピック
  - Pacemaker SNMPトラップ通知の機能強化
    - これまでのイベントに加えて、Pacemakerの属性値変更イベントでもSNMPトラップ通知ができるようになる予定
    - これまでより、さらに広範囲なイベント監視が可能となる