

# コンテナを止めるな!

PacemakerによるコンテナHAクラスタリングとKubernetesとの違いとは



2018年10月27日  
Linux-HA Japan プロジェクト  
<http://linux-ha.osdn.jp/>  
森 啓介

- 名前: 森 啓介 (Keisuke MORI)
  - twitter: @ksk\_ha
- Linux-HA Japanプロジェクト関連の活動
  - Pacemakerリポジトリパッケージのリリース
  - <http://linux-ha.osdn.jp/>
- ClusterLabs プロジェクトのコミット
  - Pacemaker、resource-agents などHAクラスタ関連の開発コミュニティ
  - <https://github.com/ClusterLabs/>
- 本業
  - 普段の業務: NTT OSSセンタ
    - NTTグループ内におけるPacemaker/Heartbeatの導入支援・サポート
    - バグ報告・パッチ作成などによるNTTから開発コミュニティへのフィードバック・貢献

- Pacemakerとは
- オーケストレーションツールによるコンテナHA
- Pacemaker bundle 機能によるコンテナHA
- 今後の動向

## ■ Pacemakerとは

- オーケストレーションツールによるコンテナHA
- Pacemaker bundle 機能によるコンテナHA
- 今後の動向

# Pacemaker ? なにそれおいしいの ?

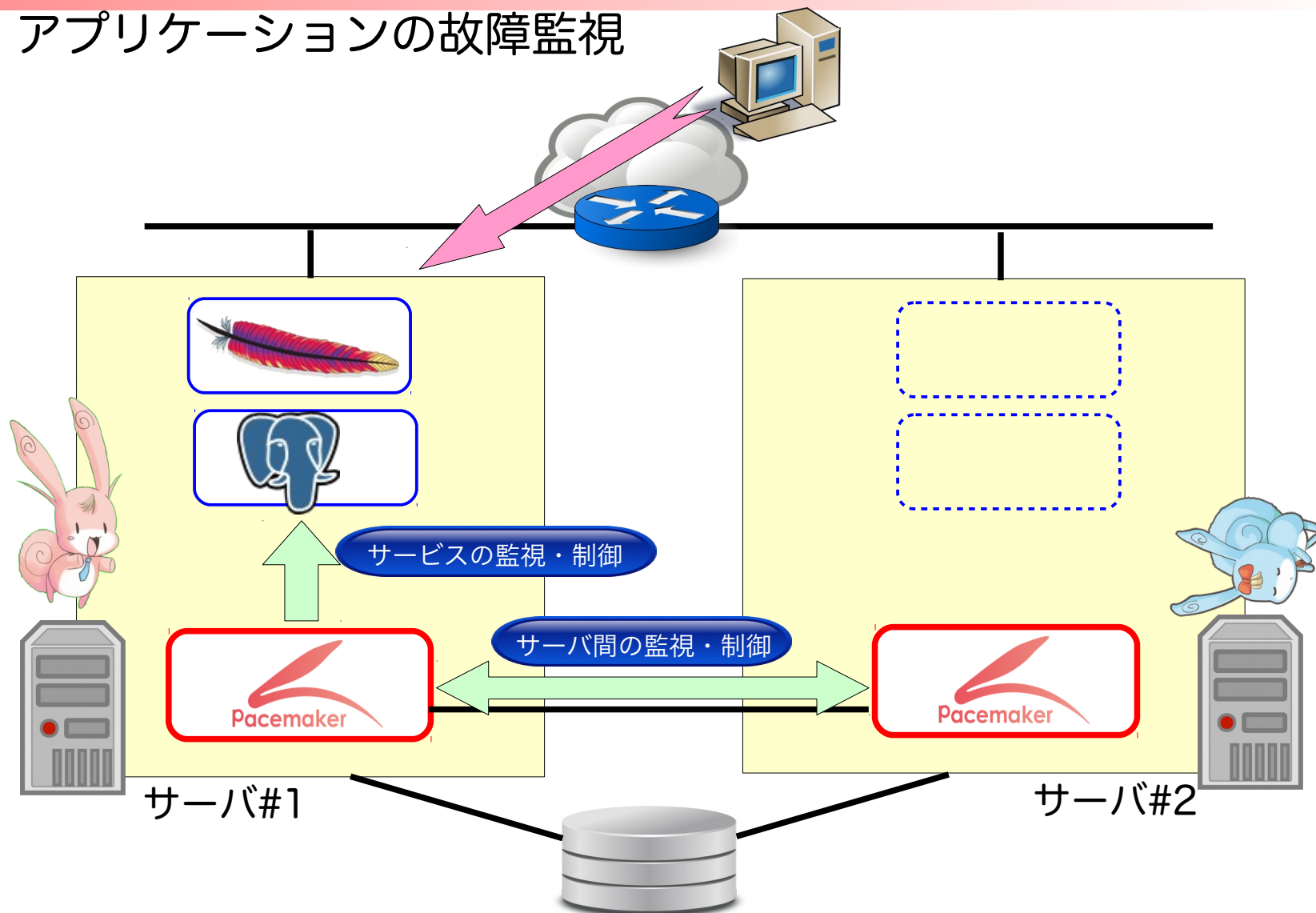
- Pacemakerとは、オープンソースのHAクラスタソフトウェアです。

**H**igh **A**vailability = 高可用性

サービスをできる限り  
「止めない！」こと

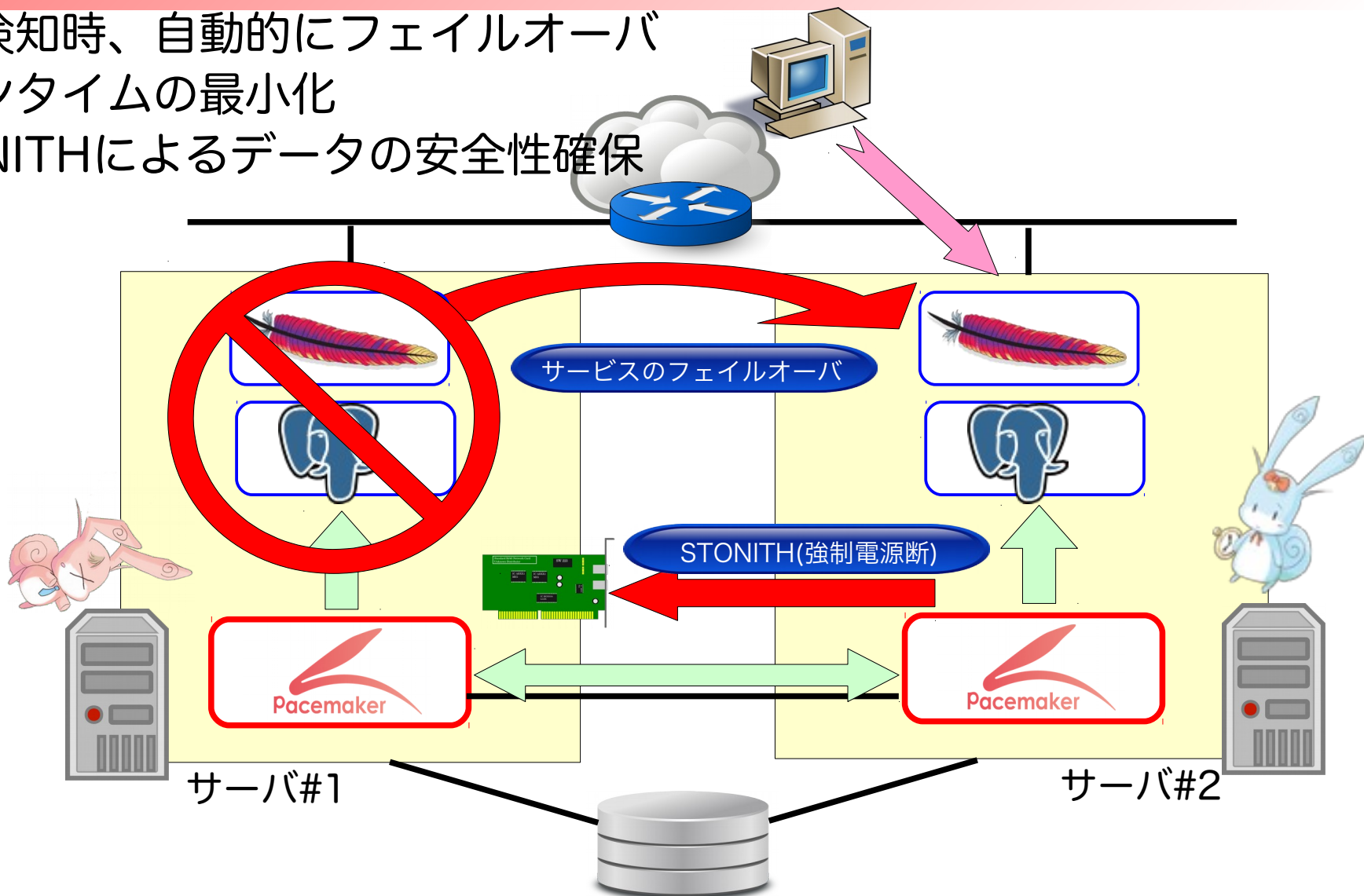
# Pacemakerの概要

## ■ サーバ・アプリケーションの故障監視



# Pacemakerの概要

- 故障検知時、自動的にフェイルオーバー
- ダウンタイムの最小化
- STONITHによるデータの安全性確保





# Pacemakerを詳しく知りたかったら…



## Pacemaker 応援キャラクター



Linux-HA Japan プロジェクト 2F 206教室にてデモ展示中！



## ■ Pacemakerとは

### ■ オーケストレーションツールによるコンテナHA

### ■ Pacemaker bundle 機能によるコンテナHA

### ■ 今後の動向

# コンテナが止まった！そのときどうなる？

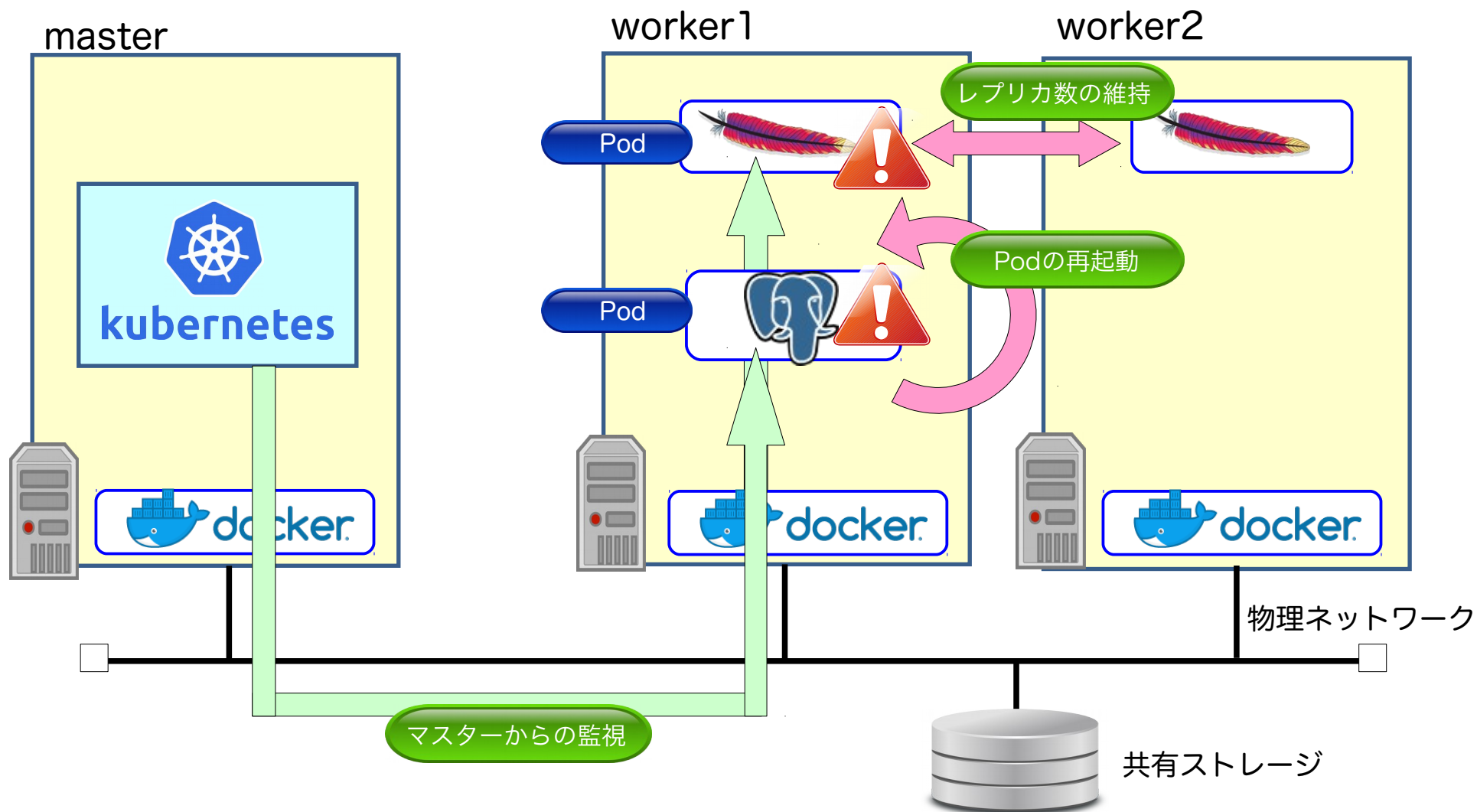


もう時代はコンテナ・クラウドでしょー  
Kubernetesがあるから大丈夫！

セルフヒーリング機能は確かに便利ね  
でもPodの種類にも気をつけて



# Kubernetes セルフヒーリング機能



## Statelessコンテナ

Deployment, ReplicaSet

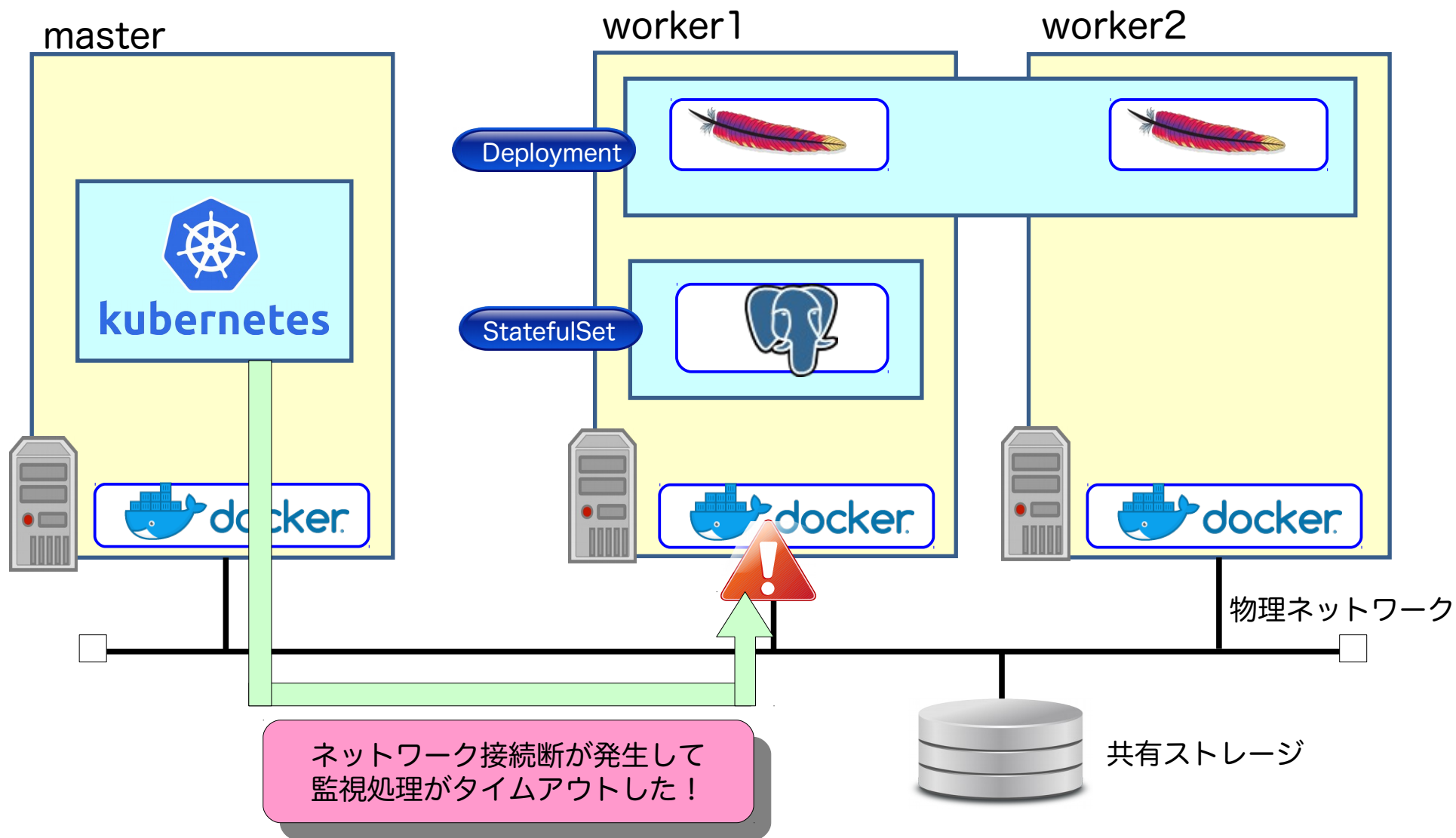
- 起動時は常に同じ状態
- データを保存する必要はない
- 必要なだけいくつでも起動すればよい
- Webサーバ、アプリケーションサーバなど

## Statefulコンテナ

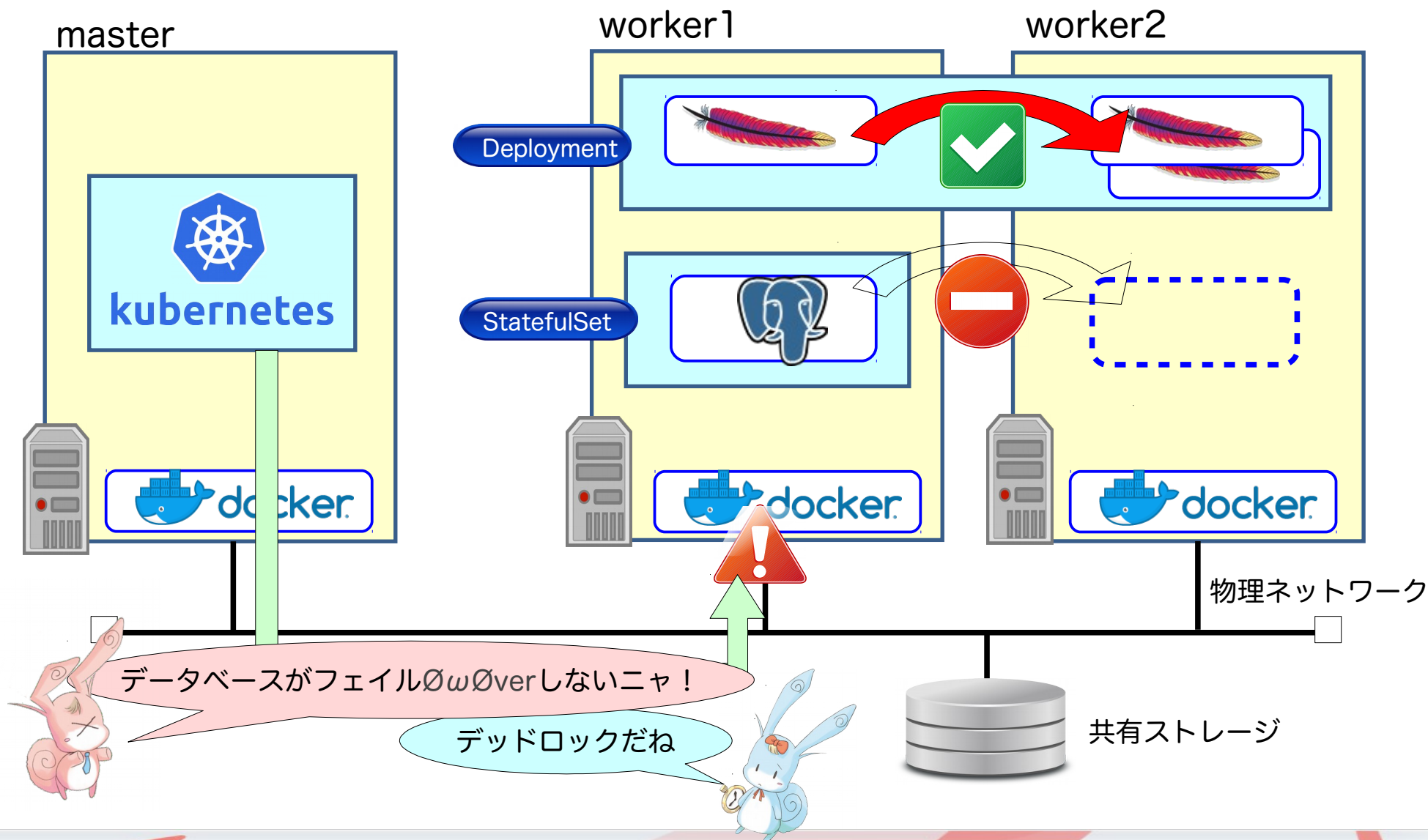
StatefulSet

- 起動時は「前回と同じ」データを持った状態で起動
- データの永続化と共有
  - 別の物理サーバでも同じデータ
- 個々の識別が必要
  - 再起動しても同じデータ
- データベースサーバ、ファイルサーバなど

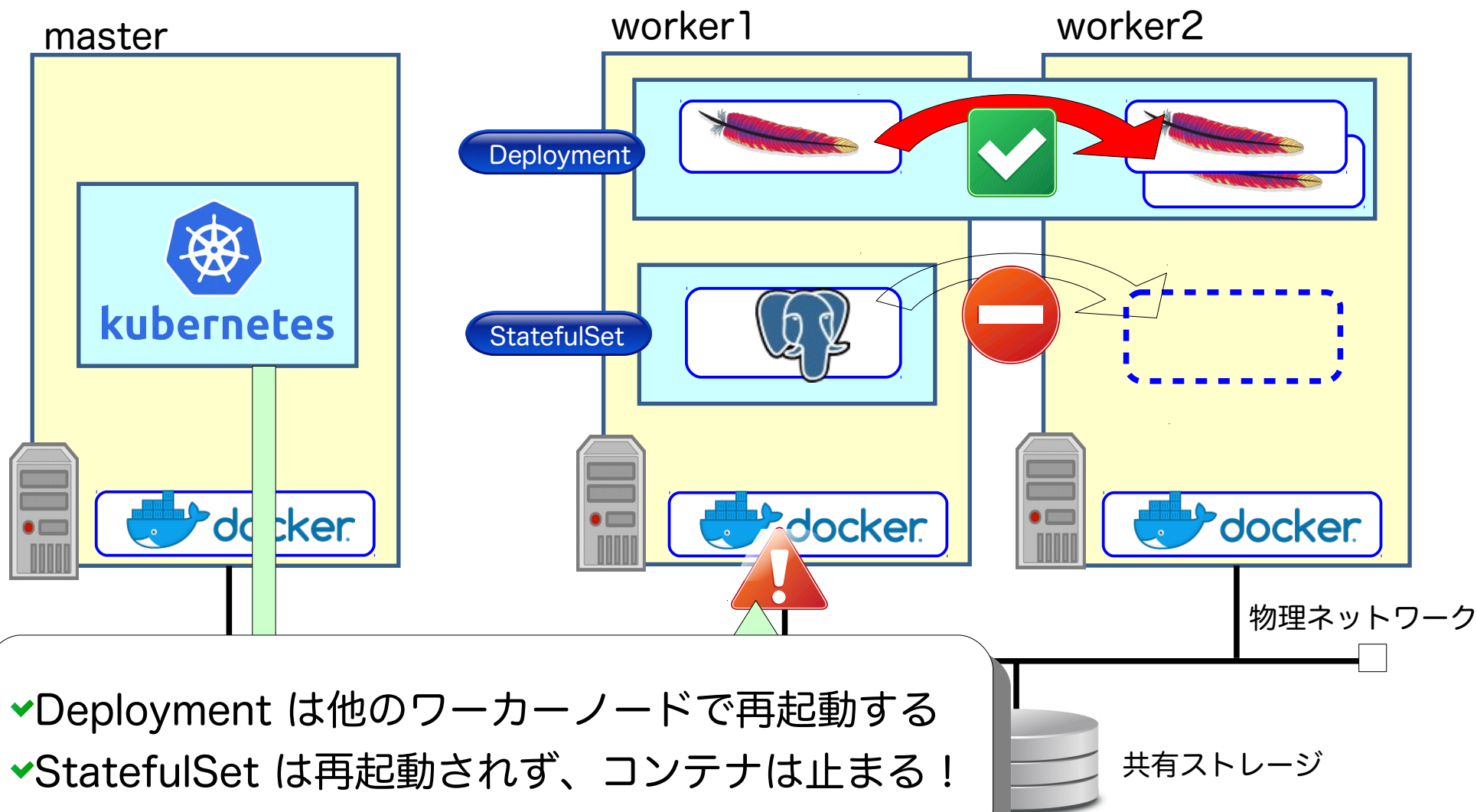
# Deployment vs. StatefulSet サービス継続性の違い



# Deployment vs. StatefulSet サービス継続性の違い



# Deployment vs. StatefulSet サービス継続性の違い





# デモ 1

## Kubernetes の実際の動作

# デモ 1: 初期状態(ノード)

```
demo — root@master:/osc2018tk-demo — ssh master — 88x16
[root@master osc2018tk-demo]# kubectl get nodes
NAME          STATUS    ROLES    AGE    VERSION
master        Ready     master   4d     v1.11.3
worker1       Ready     <none>    4d     v1.11.3
worker2       Ready     <none>    4d     v1.11.3
[root@master osc2018tk-demo]#
```

- 3 ノード構成
- 全てReady状態

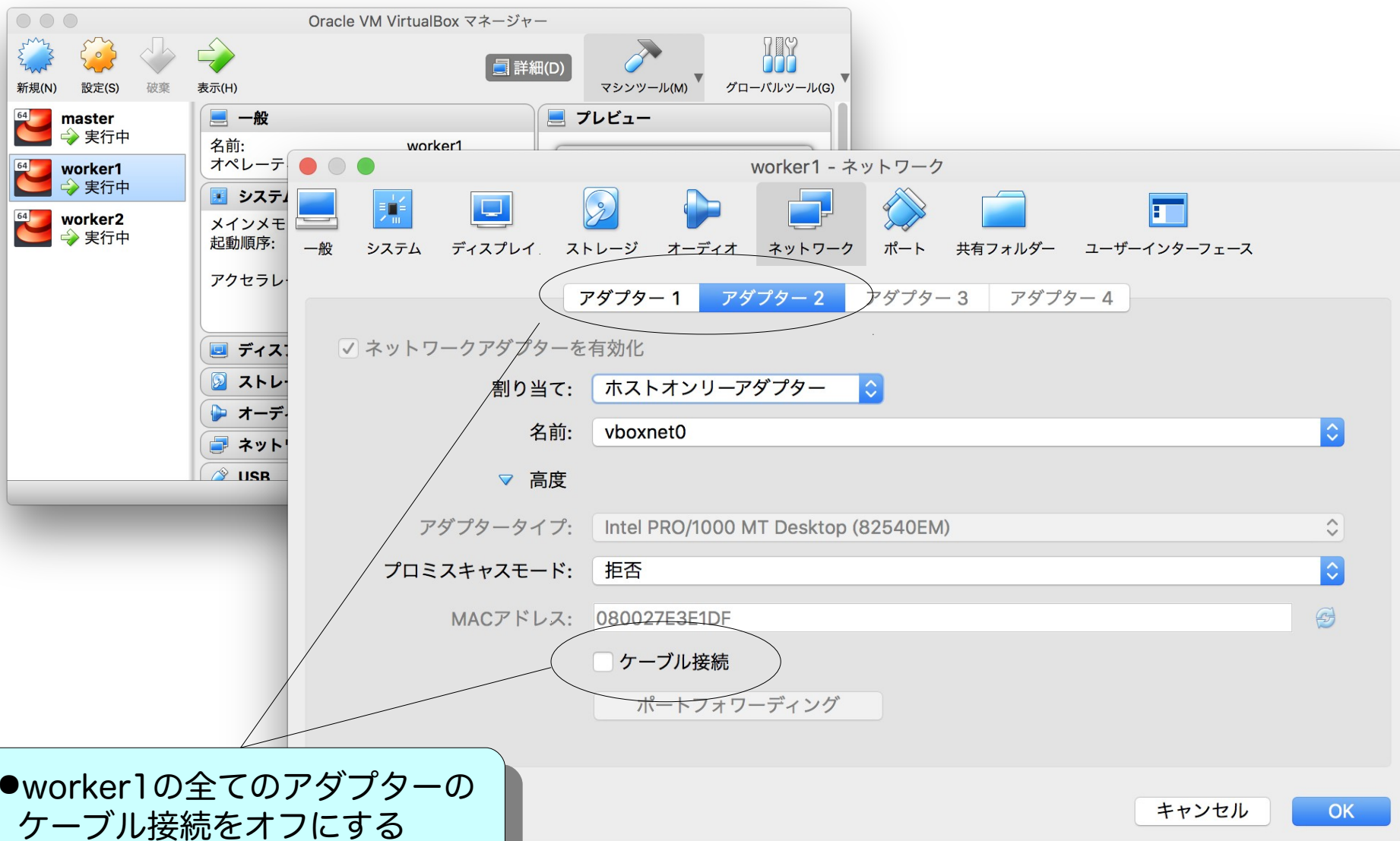
# デモ 1: 初期状態(Pod)

- httpdは両ノードで起動
- postgresはworker1で起動

```
demo — root@master:/osc2018tk-demo — ssh master — 88x16
[root@master osc2018tk-demo]# kubectl get pods -o wide
NAME                                READY    STATUS    RESTARTS   AGE    IP             NODE
httpd-5f9cf547cc-dfsqn              1/1     Running   0           8m     10.244.2.16    worker2
httpd-5f9cf547cc-pc7s2              1/1     Running   0           30s    10.244.1.22    worker1
postgres-0                          1/1     Running   0           46s    10.244.1.21    worker1
[root@master osc2018tk-demo]# kubectl get deployments
NAME      DESIRED   CURRENT   UP-TO-DATE   AVAILABLE   AGE
httpd     2         2         2            2           4d
[root@master osc2018tk-demo]# kubectl get statefulsets
NAME      DESIRED   CURRENT   AGE
postgres  1         1         4d
[root@master osc2018tk-demo]#
```

- httpdはDeployment
- postgresはStatefulSet

# デモ 1: ネットワーク切断の疑似故障



Oracle VM VirtualBox マネージャー

worker1 - ネットワーク

アダプター 1 アダプター 2 アダプター 3 アダプター 4

☒ ネットワークアダプターを有効化

割り当て: ホストオンリーアダプター

名前: vboxnet0

高度

アダプタータイプ: Intel PRO/1000 MT Desktop (82540EM)

プロミスカスモード: 拒否

MACアドレス: 080027E3E1DF

☐ ケーブル接続

ポートフォワーディング

キャンセル OK

●worker1の全てのアダプターのケーブル接続をオフにする

# デモ 1: ネットワーク切断後の状態

- worker1はNotReady状態となる  
(デフォルトでは40秒後)

demo — root@master:/osc2018tk-demo — ssh master — 88x16

Every 1.0s: kubectl get node; echo ; kubectl get pods -o wide Wed Oct 24 08:35:42 2018

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	4d	v1.11.3
worker1	NotReady	<none>	4d	v1.11.3
worker2	Ready	<none>	4d	v1.11.3

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
httpd-5f9cf547cc-chk6c	1/1	Running	0	7s	10.244.2.18	worker2
httpd-5f9cf547cc-dfsqn	1/1	Running	0	10m	10.244.2.16	worker2
httpd-5f9cf547cc-pc7s2	1/1	Unknown	0	2m	10.244.1.22	worker1
postgres-0	1/1	Unknown	0	2m	10.244.1.21	worker1

- worker1上のpodは Unknown 状態となる
- httpdはworker2で2つ起動する
- postgres はworker2では起動しない  
(デフォルトでは5分後)

# なぜフェイルオーバーしないのか？

## ■ コンテナの二重起動によるデータ破損を防ぐためです。

- コンテナ状態: Unknown → まだ動いてるかも？
- コンテナが「確実に停止」したことを確認できない限り再起動は危険

## ■ でもさあ…

### □ Q.でもネットワークが切れてたらデータも書き込めないから安全だよね？

- ネットワーク通信が復旧するとそこで二重書き込みが発生します。
  - ネットワークスイッチの一時的な故障などではよくあるシナリオ
- 監視用とストレージ用で別々のネットワークである構成もあります。
  - SAN経由、ストレージ専用ネットワーク(トラフィック、セキュリティ上の理由など)

### □ Q.物理サーバの電源が落ちたらさすがに再起動するよね？

- しません。
- 物理サーバの電源断とネットワーク通信断(通信経路の故障)は区別がつかないからです。
  - どちらも「監視タイムアウト(応答なし)」としか検知できない

- 一般的なデータベースサーバは StatefulSet とする必要がある。
  - データの永続化のため
  - インスタンスごとの識別のため
- StatefulSet は、物理サーバの故障に対しては自動的にサービス継続を行うことはできない。
  - Podの二重起動によるデータ破損を防ぐため



# コンテナが止まった！そのときどうなる？



もう時代はコンテナ・クラウドでしょー  
Kubernetesがあるから大丈夫！



セルフヒーリング機能は確かに便利ね  
でもPodの種類にも気をつけて

StatefulSetは物理サーバが故障  
したら止まっちゃうんだよ



大事なデータが壊れないように  
守っているんだね！

# 物理サーバではどうやってたの？



でも物理サーバのHAクラスタは  
フェイルオーバーできてたよね？なんで？

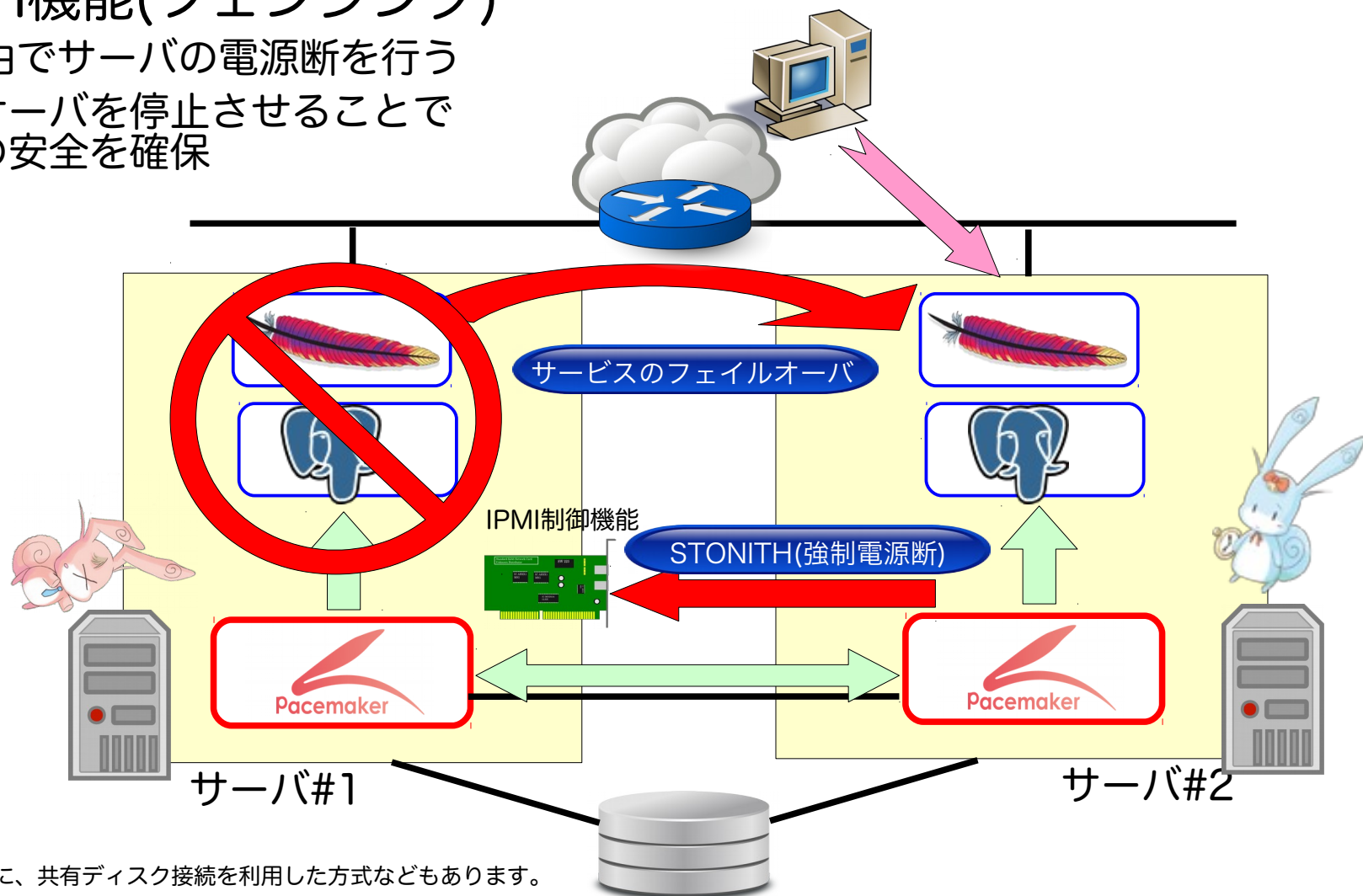


それは Pacemaker の  
STONITH機能のおかげね

# 物理サーバではどうやってたの？

## ■ STONITH機能(フェンシング)

- IPMI経由でサーバの電源断を行う
- 確実にサーバを停止させることでデータの安全を確保



※IPMIによる電源断以外に、共有ディスク接続を利用した方式などもあります。

# STONITH機能(フェンシング)とは?

- フェンスを立てて隔離すること



## ■ 狩りに出かけたハンターからの救急電話

- 「大変だ! 友達が木から落ちて死んでしまった!」
- 「落ち着いて。まずは彼が本当に死んでいるかどうか確かめて」
- 「バーン!(銃声)」
- 「OK、死んでる。次は?」

※出典 “World's funniest joke” ハートフォードシャー大学 リチャード・ワイズマン博士(2002年)

Make sure he is dead.

死んでるかどうかを確かめる

(気絶しているだけかもしれない獲物を)  
確実に死んだ状態にする

これってSTONITHじゃね?



## 故障モデル




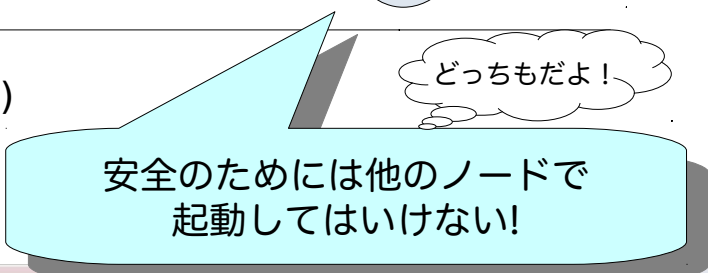
- サーバが壊れる状況のしんどさを段階的に切り分けたモデルのこと
- より楽な故障モデルで動かないプロトコルは、しんどい故障モデルでは「絶対に」動かない
  - Fail-Stop: 壊れたサーバはいずれ全部のサーバから故障として観測される。壊れていないサーバを壊れたと誤認する事や、壊れたサーバを壊れていないと誤認する事は発生しない。壊れたサーバは二度と復活しない。
  - Crash-Recover: サーバは壊れるかも知れないが復活する事もある。つまりいつまでも故障したと断言できない。



実際の故障例

他ノードで  
起動した場合の  
データの安全性

監視タイムアウト  
発生時の判断

故障なし			
Fail-Stop	<ul style="list-style-type: none"><li>●物理サーバの電源断</li><li>●カーネルクラッシュ</li></ul>	○安全	
Crash-Recovery	<ul style="list-style-type: none"><li>●ネットワーク通信断</li><li>●異常な高負荷</li></ul>	×危険	
Byzantine	<ul style="list-style-type: none"><li>●クラッキングなど(通常は対応は想定しない)</li></ul>		



実際の故障例	他ノードで 起動した場合の データの安全性	STONITH 実行後の判断
--------	-----------------------------	-------------------

故障なし		
Fail-Stop	<ul style="list-style-type: none"><li>●物理サーバの電源断</li><li>●カーネルクラッシュ</li></ul>	○安全
Crash-Recovery	<ul style="list-style-type: none"><li>●ネットワーク通信断</li><li>●異常な高負荷</li></ul>	×危険
Byzantine	<ul style="list-style-type: none"><li>●クラッキングなど(通常は対応は想定しない)</li></ul>	

STONITHの実行により  
確実に安全と言える  
状態に確定させる

# 物理サーバではどうやってたの？



でも物理サーバのHAクラスタは  
フェイルオーバーできてたよね？なんで？



それは Pacemaker の  
STONITH機能のおかげね



じゃあそれを使ってコンテナを  
止めないようにしちゃおう！



そこで Pacemaker bundle 機能の出番よ！

- Pacemakerとは
- オーケストレーションツールによるコンテナHA
- Pacemaker bundle 機能によるコンテナHA
- 今後の動向

## ■ Pacemaker でコンテナHAを実現する新機能

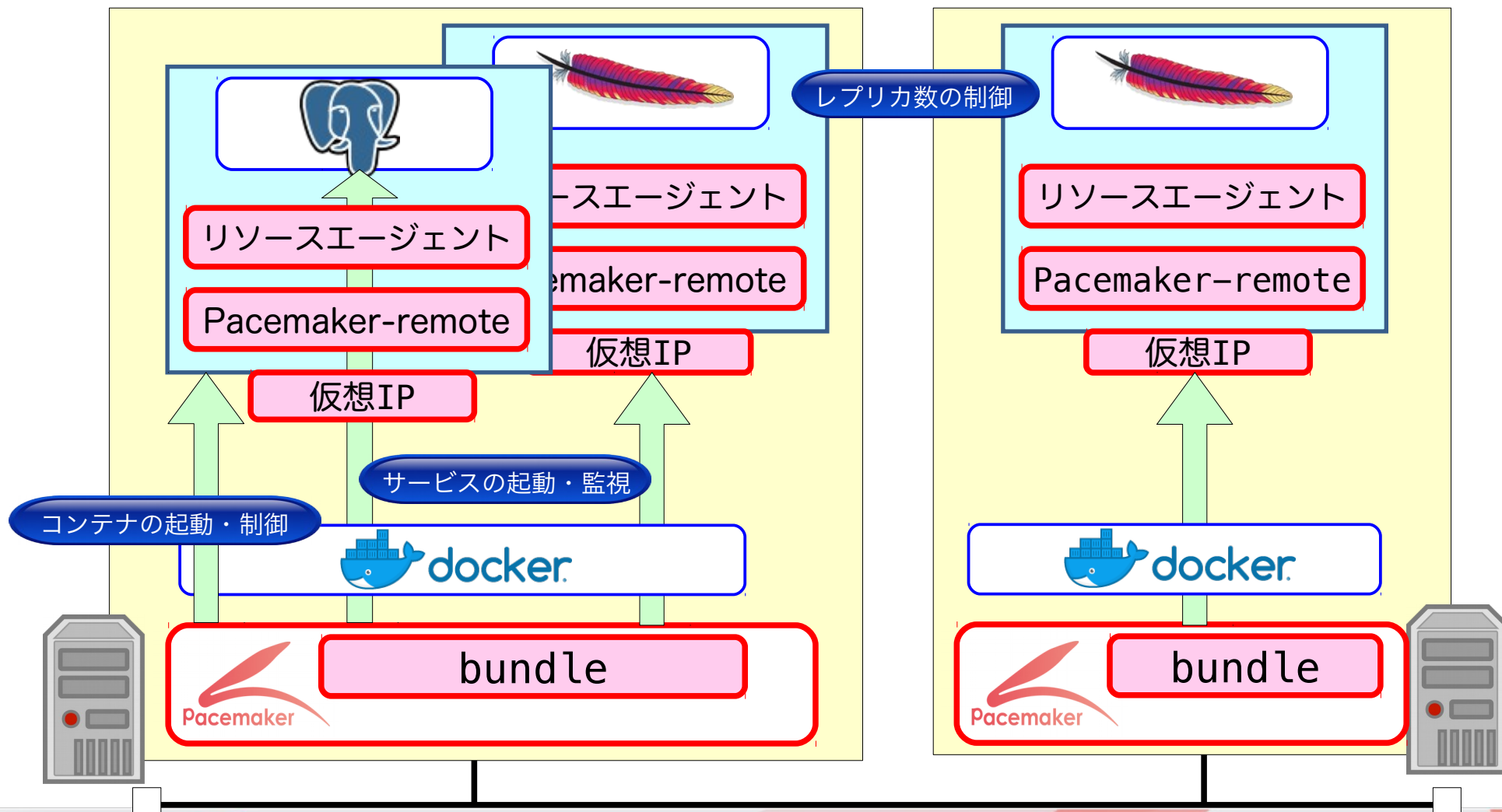
- コンテナ管理に必要な機能をまとめたPacemakerのリソース
- コンテナおよびコンテナ内で起動するアプリケーションの起動・終了・監視が可能
- STONITH機能や既存のリソースエージェント(RA)と合わせて利用可能

## ■ Pacemaker-1.1.17 以降で利用可能

- 対応コンテナランタイム

コンテナランタイム	対応Pacemakerバージョン
Docker	1.1.17以降
rkt	1.1.18以降
podman (CRI-O)	開発中(2.0.*以降予定)

# Pacemaker bundle のアーキテクチャ

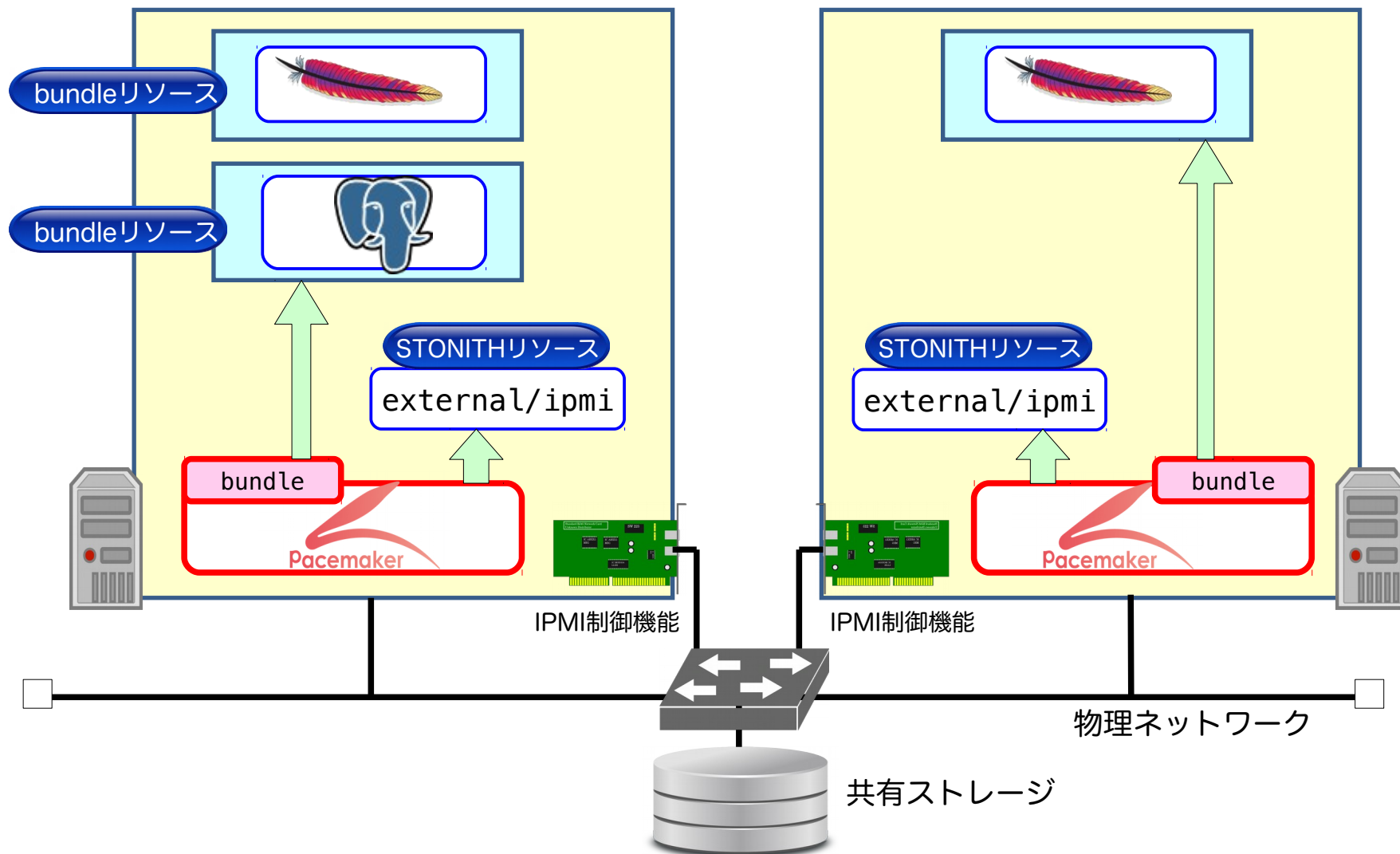


# bundle 設定例(cib.xml)

```
<bundle id="httpd-bundle">
  <docker image="pcmkttest:http" replicas="2" replicas-per-host="2" options="--log-driver=journald"/>
  <network ip-range-start="192.168.33.200" host-interface="eth1" host-netmask="24">
    <port-mapping id="httpd-port" port="80"/>
  </network>
  <storage>
    <storage-mapping id="httpd-root"
      source-dir-root="/var/local/containers"
      target-dir="/var/www/html"
      options="rw"/>
    <storage-mapping id="httpd-logs"
      source-dir-root="/var/log/pacemaker/bundles"
      target-dir="/etc/httpd/logs"
      options="rw"/>
  </storage>
  <primitive class="ocf" id="httpd" provider="heartbeat" type="apache">
    <operations>
      <op name="start" interval="0s" timeout="60s" on-fail="restart" id="httpd-start-0s"/>
      <op name="monitor" interval="10s" timeout="60s" on-fail="restart" id="httpd-monitor-10s"/>
      <op name="stop" interval="0s" timeout="60s" on-fail="block" id="httpd-stop-0s"/>
    </operations>
  </primitive>
</bundle>
```

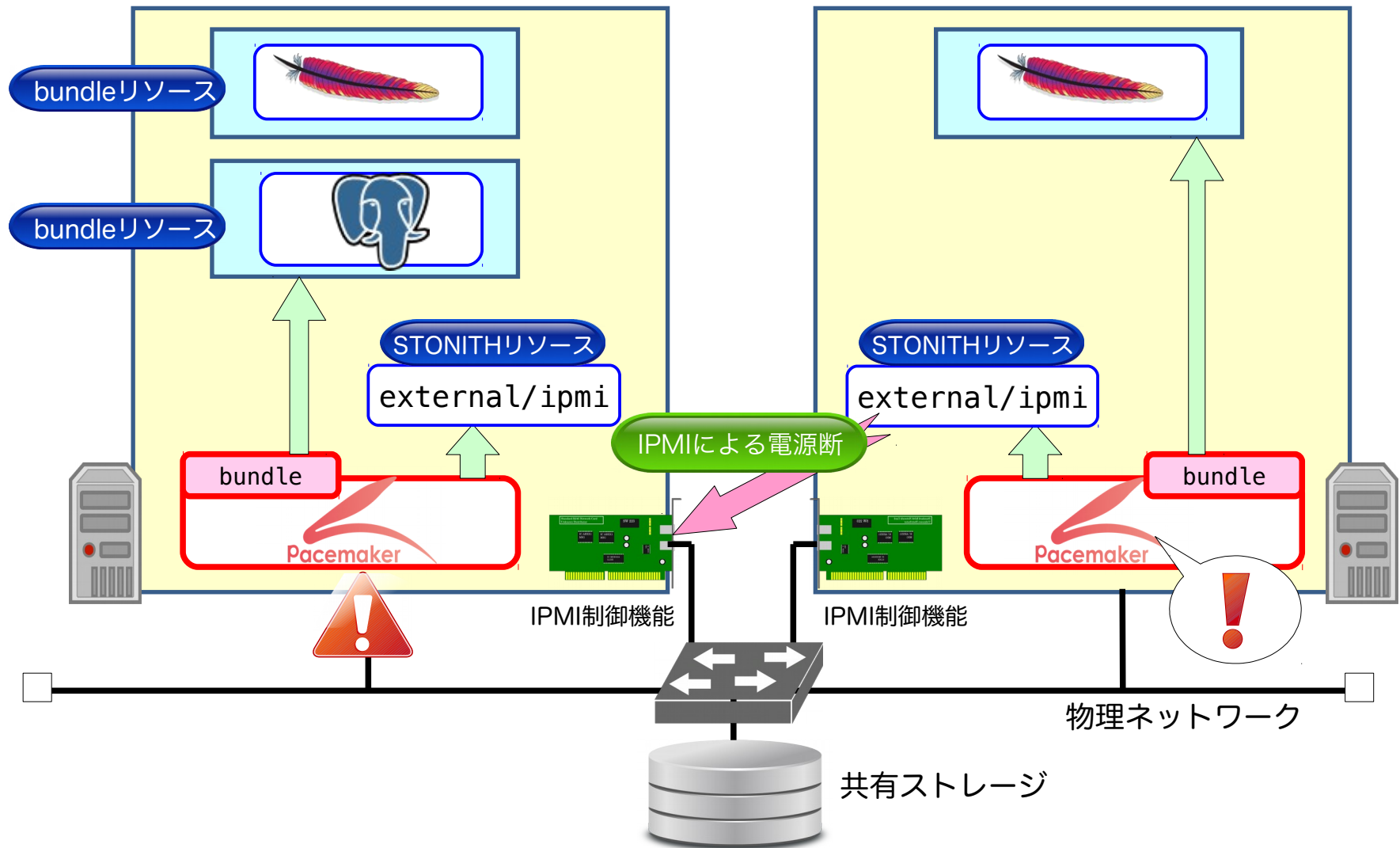
※ bundleに関する詳細については OSC 2018 Kyoto セミナー資料を参照  
<http://linux-ha.osdn.jp/wp/archives/4744>

# Pacemaker bundle とSTONITHによるコンテナHA

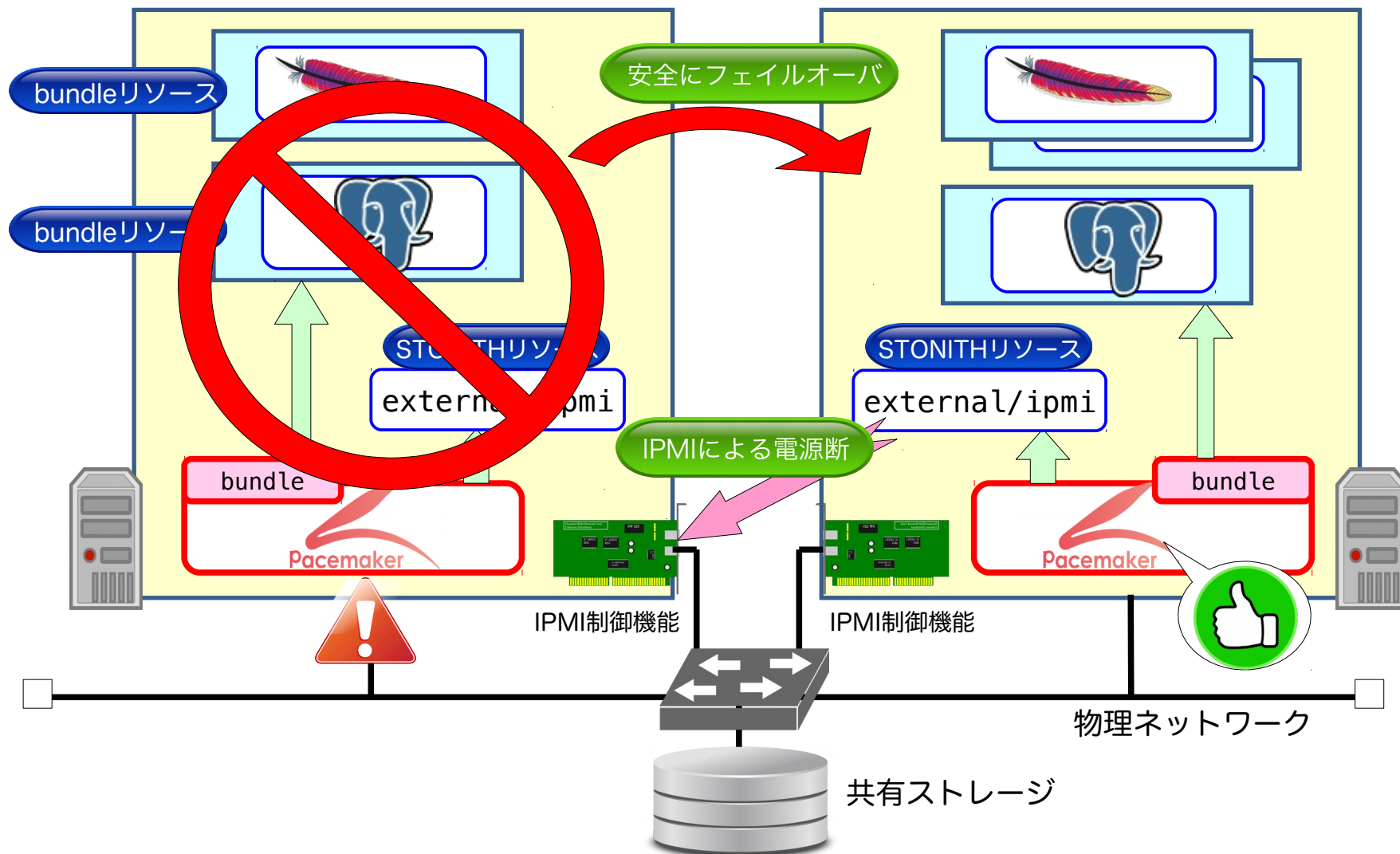




# Pacemaker bundle とSTONITHによるコンテナHA



# Pacemaker bundle とSTONITHによるコンテナHA



## ■ crm\_mon 出力例

```
demo — root@master:/osc2018tk-demo — ssh master — 88x16
Online: [ worker1 worker2 ]
GuestOnline: [ httpd-bundle-0@worker1 httpd-bundle-1@worker2 postgres-bundle-0@worker1 ]

Full list of resources:

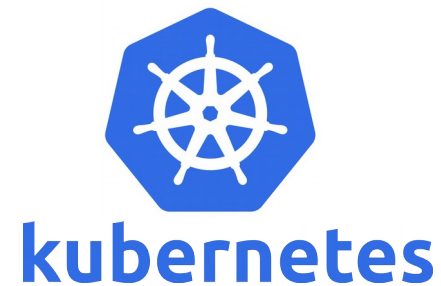
  Docker container: postgres-bundle [192.168.33.10:5000/pcmkdemo/postgres:0.1]
    postgres-bundle-0 (192.168.33.200) (ocf::heartbeat:pgsql): Started worker1
  Docker container set: httpd-bundle [192.168.33.10:5000/pcmkdemo/httpd:0.1] (unique)
    httpd-bundle-0 (192.168.33.210) (ocf::heartbeat:apache): Started worker1
    httpd-bundle-1 (192.168.33.211) (ocf::heartbeat:apache): Started worker2
  stonith-worker2 (stonith:external/ipmi): Started worker1
  stonith-worker1 (stonith:external/ipmi): Started worker2
```

STONITHリソース

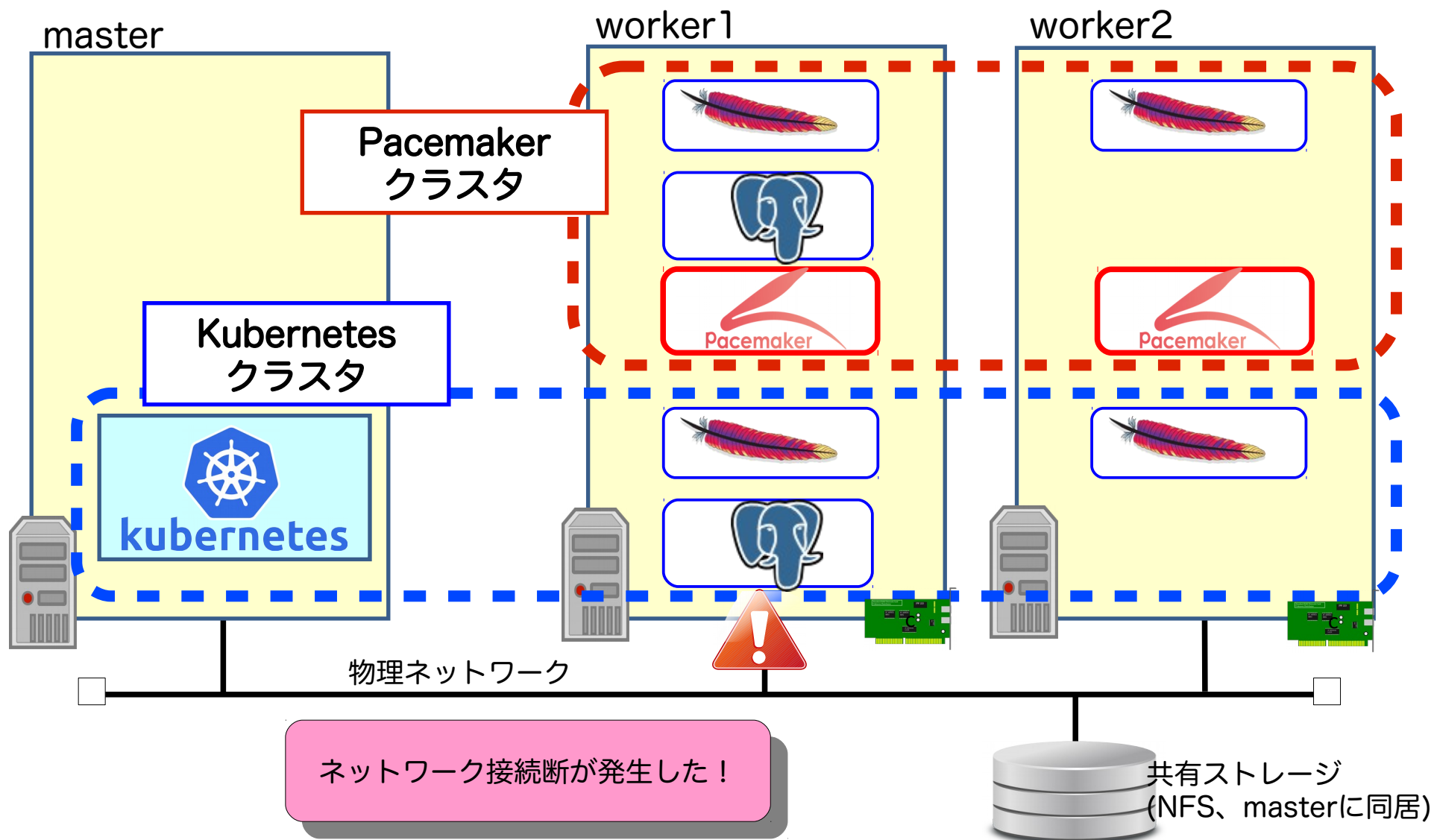
bundleリソース

デモ

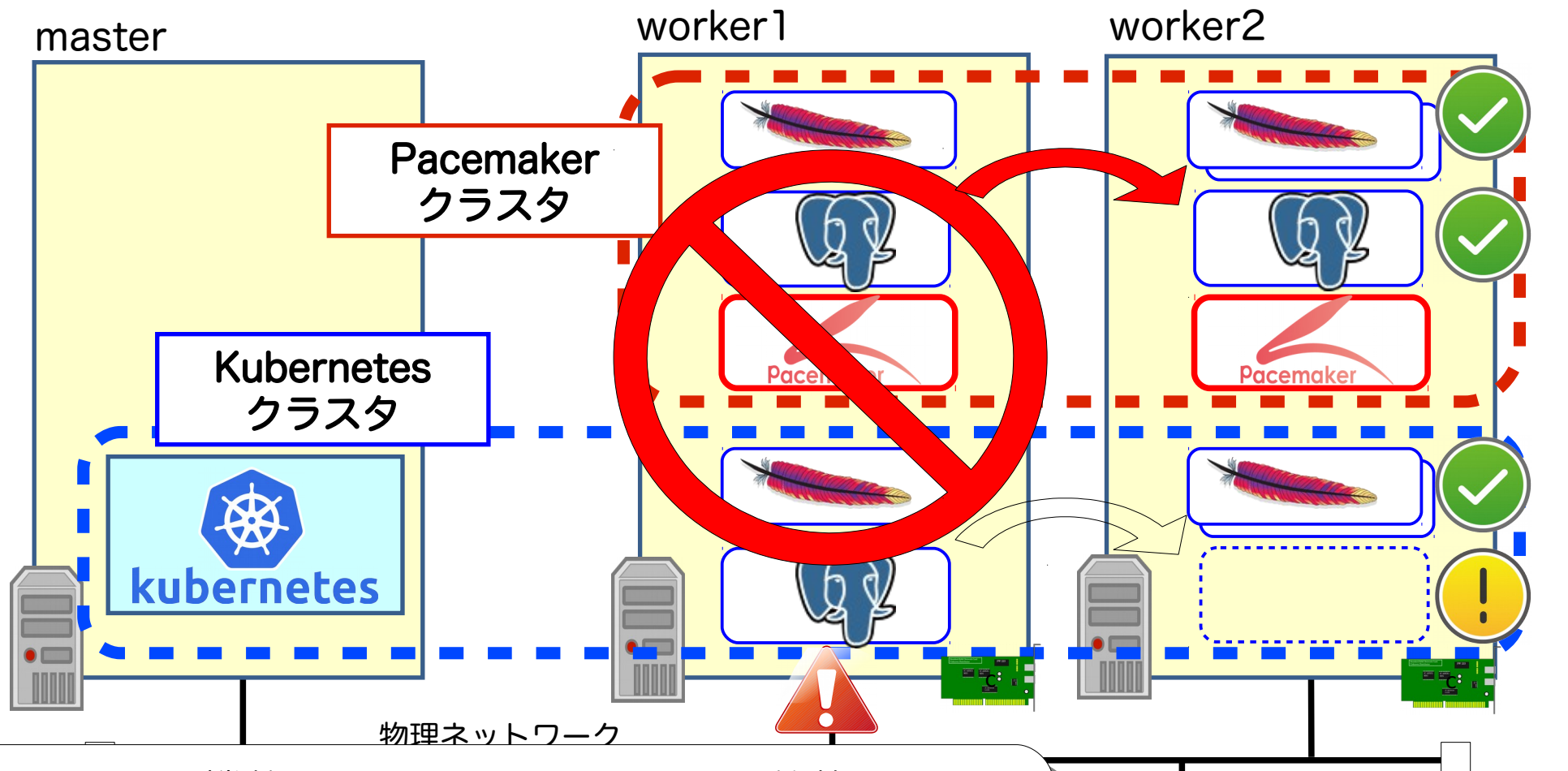
## Pacemaker bundle vs. Kubernetes



# Pacemaker bundle デモ環境構成



# Pacemaker bundle デモ実行結果(予定)



- ✓STONITH機能によりworker1は電源断状態となる
- ✓Pacemakerクラスタは全コンテナがフェイルオーバー
- ✓Kubernetesクラスタはデータベースが停止

共有ストレージ  
(NFS、masterに同居)

# デモ 2: 初期状態(Pacemaker)

- httpdは両ノードで起動
- postgresはworker1で起動

```
demo — root@master:/osc2018tk-demo — ssh master — 88x16
Online: [ worker1 worker2 ]
GuestOnline: [ httpd-bundle-0@worker1 httpd-bundle-1@worker2 postgres-bundle-0@worker1 ]

Full list of resources:

  Docker container: postgres-bundle [192.168.33.10:5000/pcmkdemo/postgres:0.1]
    postgres-bundle-0 (192.168.33.200) (ocf::heartbeat:pgsql): Started worker1
  Docker container set: httpd-bundle [192.168.33.10:5000/pcmkdemo/httpd:0.1] (unique)
    httpd-bundle-0 (192.168.33.210) (ocf::heartbeat:apache): Started worker1
    httpd-bundle-1 (192.168.33.211) (ocf::heartbeat:apache): Started worker2
stonith-worker2 (stonith:external/ipmi): Started worker1
stonith-worker1 (stonith:external/ipmi): Started worker2
```



# デモ 2: 初期状態(Kubernetes)

- httpdは両ノードで起動
- postgresはworker1で起動

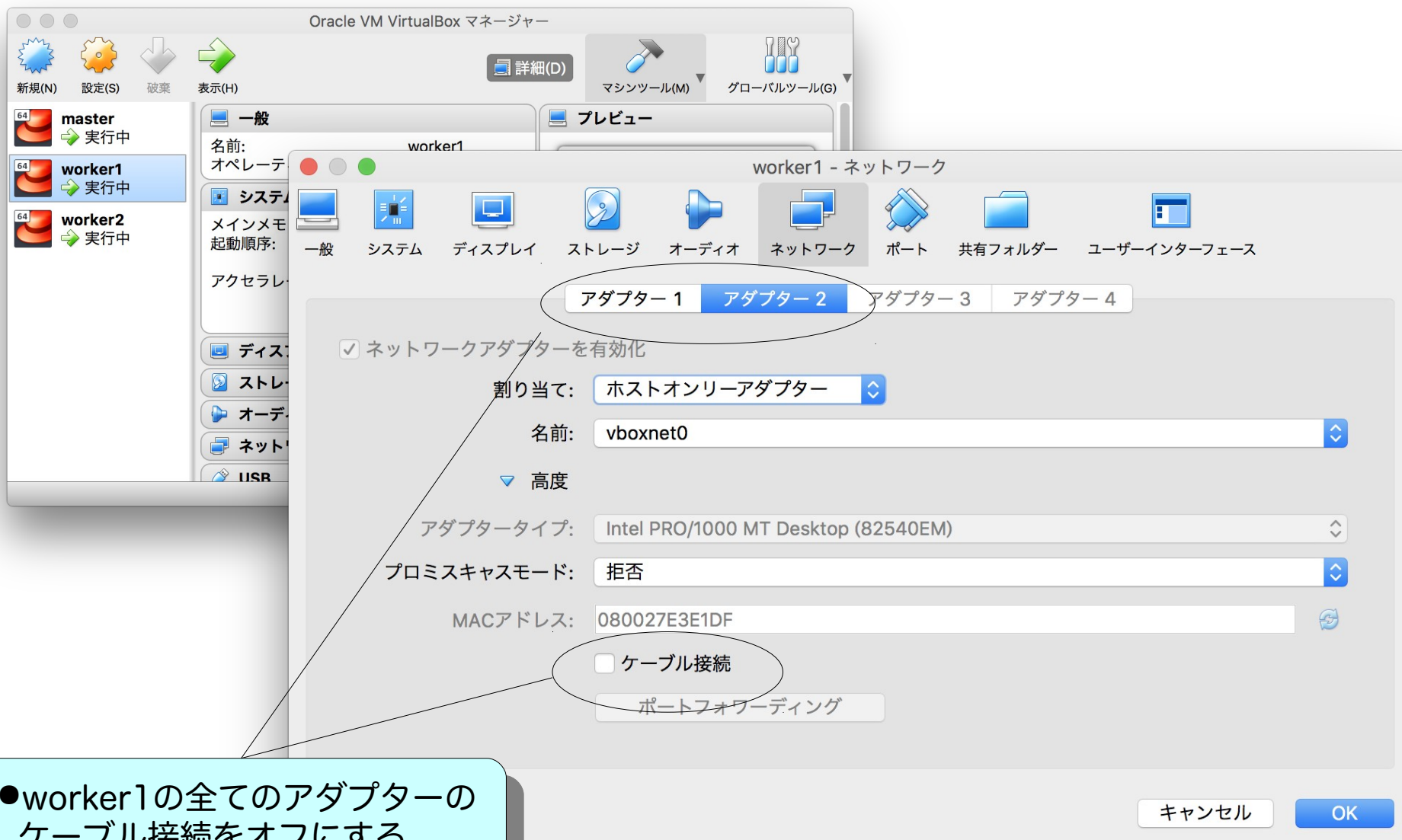
```
demo — root@master:/osc2018tk-demo — ssh master — 88x16
Every 1.0s: kubectl get node; echo ; kubectl get pods -o wide   Wed Oct 24 08:39:02 2018
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	master	4d	v1.11.3
worker1	Ready	<none>	4d	v1.11.3
worker2	Ready	<none>	4d	v1.11.3

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
httpd-5f9cf547cc-dfsqn	1/1	Running	0	13m	10.244.2.16	worker2
httpd-5f9cf547cc-tn9wd	1/1	Running	0	2m	10.244.1.24	worker1
postgres-0	1/1	Running	0	2m	10.244.1.23	worker1

# デモ 2: ネットワーク切断の疑似故障



Oracle VM VirtualBox マネージャー

worker1 - ネットワーク

アダプター 1 アダプター 2 アダプター 3 アダプター 4

☒ ネットワークアダプターを有効化

割り当て: ホストオンリーアダプター

名前: vboxnet0

高度

アダプタータイプ: Intel PRO/1000 MT Desktop (82540EM)

プロミスキャスモード: 拒否

MACアドレス: 080027E3E1DF

☐ ケーブル接続

ポートフォワーディング

キャンセル OK

●worker1の全てのアダプターのケーブル接続をオフにする

# デモ 2: ネットワーク切断後の状態(故障サーバ)



Oracle VM VirtualBox マネージャー

新規(N) 設定(S) 破棄 起動(T) 詳細(D) マシンツール(M) グローバルツール(G)

64 master → 実行中

64 **worker1** 電源オフ

64 worker2 → 実行中

**worker1**

名前: worker1  
オペレーティングシステム: Red Hat (64-bit)

システム  
メインメモリ: 2048 MB  
起動順序: フロッピー, 光学, ハードディスク  
アクセラレーション: VT-x/AMD-V, ネステッドページング, PAE/NX, KVM 準仮想化

ディスプレイ  
ストレージ  
オーディオ  
ネットワーク

●worker1はSTONITH機能により電源オフとなる

# デモ 2: マシン電源オフ後の状態(Pacemaker)

```
demo — root@master:/osc2018tk-demo — ssh master — 88x16

Online: [ worker2 ]
OFFLINE: [ worker1 ]
GuestOnline: [ httpd-bundle-0@worker2 httpd-bundle-1@worker2 postgres-bundle-0@worker2 ]

Full list of resources:

  Docker container: postgres-bundle [192.168.33.10:5000/pcmkdemo/postgres:0.1]
    postgres-bundle-0 (192.168.33.200) (ocf::heartbeat:pgsql): Started worker2
  Docker container set: httpd-bundle [192.168.33.10:5000/pcmkdemo/httpd:0.1] (unique)
    httpd-bundle-0 (192.168.33.210) (ocf::heartbeat:apache): Started worker2
    httpd-bundle-1 (192.168.33.211) (ocf::heartbeat:apache): Started worker2
  stonith-worker2 (stonith:external/ipmi): Stopped
  stonith-worker1 (stonith:external/ipmi): Started worker2
```

- httpdはworker2で2つ起動する
- postgresもworker2で再起動する

# デモ 2: マシン電源オフ後の状態(Kubernetes)

```
demo — root@master:/osc2018tk-demo — ssh master — 88x16

Every 1.0s: kubectl get node; echo ; kubectl get pods -o wide    Wed Oct 24 08:41:52 2018

NAME        STATUS    ROLES    AGE    VERSION
master      Ready     master   4d     v1.11.3
worker1     NotReady  <none>    4d     v1.11.3
worker2     Ready     <none>    4d     v1.11.3

NAME                                READY    STATUS    RESTARTS   AGE    IP             NODE
httpd-5f9cf547cc-9qgh2              1/1     Running   0          1m     10.244.2.19    worker2
httpd-5f9cf547cc-dfsqn              1/1     Running   0          16m    10.244.2.16    worker2
httpd-5f9cf547cc-tn9wd              1/1     Unknown   0          5m     10.244.1.24    worker1
postgres-0                          1/1     Unknown   0          5m     10.244.1.23    worker1
```

- worker1上のpodは Unknown 状態となる
- httpdはworker2で2つ起動する
- postgres はworker2では起動しない  
(ネットワーク切断時と同じ結果)

- Pacemakerとは
- オーケストレーションツールによるコンテナHA
- Pacemaker bundle 機能によるコンテナHA
- 今後の動向



## ■ Pacemaker によるコンテナHAが可能になった!

- とは言え…

## ■ bundle機能もまだまだ発展途上

- ユーザインタフェースの対応

- crmshは開発最新版のみ対応。pcs の方が対応が早い。

- ドキュメント・動作仕様の詳細

- コンテナイメージの要件・作り方、詳細設定や故障解析方法などの情報

- コンテナ技術自体の発展

- podman(CRI-O)対応、Dockerの今後は?

- 実績・サポート

- Red Hat 社 HA add-on では Technology Preview

- Red Hat OpenStack Platform 12以降のユースケースのみフルサポート

- <https://access.redhat.com/articles/3388681>

## ■ オーケストレーションツールに取って代わるものではない



## ■ オーケストレーションツールが適している利用シーン

- DevOps, CI/CD、頻繁なリリース、大規模なスケーリング
- 物理故障はマネージドサービスのSLA

## ■ Pacemaker bundle が適している利用シーン

- 物理環境と同様のHA運用が必要、かつコンテナのメリットが欲しい場合
  - アプリケーションのバージョンアップが容易
  - バージョン依存が異なる複数のアプリケーションの同居
  - コンテナと非コンテナアプリケーションの同居(性能上の理由など)
- 比較的小規模なコンテナHAクラスタ
- 信頼性を求められるオンプレミスシステム
  - 物理故障に対する運用責任・説明責任を求められるようなシステム

## ■ Pacemaker bundle の利用実績例

- Red Hat OpenStack Platform 12以降
  - コントローラノードの各種ミドルウェアのコンテナ化
  - HAProxy, RabbitMQ, Galera, redis

## ■ Pacemaker-1.1.19-1.1 リポジトリパッケージ 近日リリース予定!



- PostgreSQL 11以降対応など
- 本日の bundle のデモでも使用しました!
- なお、バージョン1.1.18-1.1のリリースはありません。
  - (一部の利用方法で懸念事項が存在したため。1.1.19-1.1では解消済みです)
- 1.1.\*系は今後も継続してメンテナンスしていきます。

## ■ Pacemaker-2.0以降は…

- RHEL 8 / CentOS 8 同梱版と親和性の高い方式を検討中。

## ■ 本日のデモ環境

- <https://github.com/kskmori/osc2018tk-demo>
- (README等準備中です)

## ■ Pacemaker bundle 関連

- Bundle Walk-Through (公式ドキュメント)
  - [https://wiki.clusterlabs.org/wiki/Bundle\\_Walk-Through](https://wiki.clusterlabs.org/wiki/Bundle_Walk-Through)
- Red Hat OSP12 での利用事例
  - [https://access.redhat.com/documentation/en-us/red\\_hat\\_opensstack\\_platform/12/html/understanding\\_red\\_hat\\_opensstack\\_platform\\_high\\_availability/pacemaker#pacemaker-services](https://access.redhat.com/documentation/en-us/red_hat_opensstack_platform/12/html/understanding_red_hat_opensstack_platform_high_availability/pacemaker#pacemaker-services)

## ■ STONITH関連

- 分散システムについて語らせてくれ
  - <https://www.slideshare.net/kumagi/ss-78765920>
- 世界で一番笑えるジョーク (wikipedia)
  - <https://ja.wikipedia.org/wiki/世界で一番笑えるジョーク>

## ■ Kubernetes 関連

### □ StatefulSets (公式ドキュメント)

- <https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/>

### □ StatefulSets の強制削除手順

- 二重起動時のデータ喪失のリスクについても記載あり

- <https://kubernetes.io/docs/tasks/run-application/force-delete-stateful-set-pod/>

### □ ノード停止時に StatefulSet が移動しないという issue

- 設計通りの動作であるとのコメントあり

- <https://github.com/kubernetes/kubernetes/issues/54368#issuecomment-339378164>

### □ Kubernetes に対する Fencing 機能の提案

- <https://github.com/kubernetes/community/blob/master/contributors/design-proposals/storage/pod-safety.md>

### □ Kubernetes のノード断検知のロジック、パラメタチューニング

- <https://fatalfailure.wordpress.com/2016/06/10/improving-kubernetes-reliability-quicker-detection-of-a-node-down/>

- Pacemakerをこれからもよろしくお願いします!

