増税に立ち向かえ、俺が”コンテナ”を”Pacemaker”と”DRBD”を使って無償で冗長化してやる

2019/11/24 OSC Tokyo/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)

強気なタイトルですが、強気なタイプではありません。

普段はNTT OSSセンタという所で
高信頼技術をやってます。

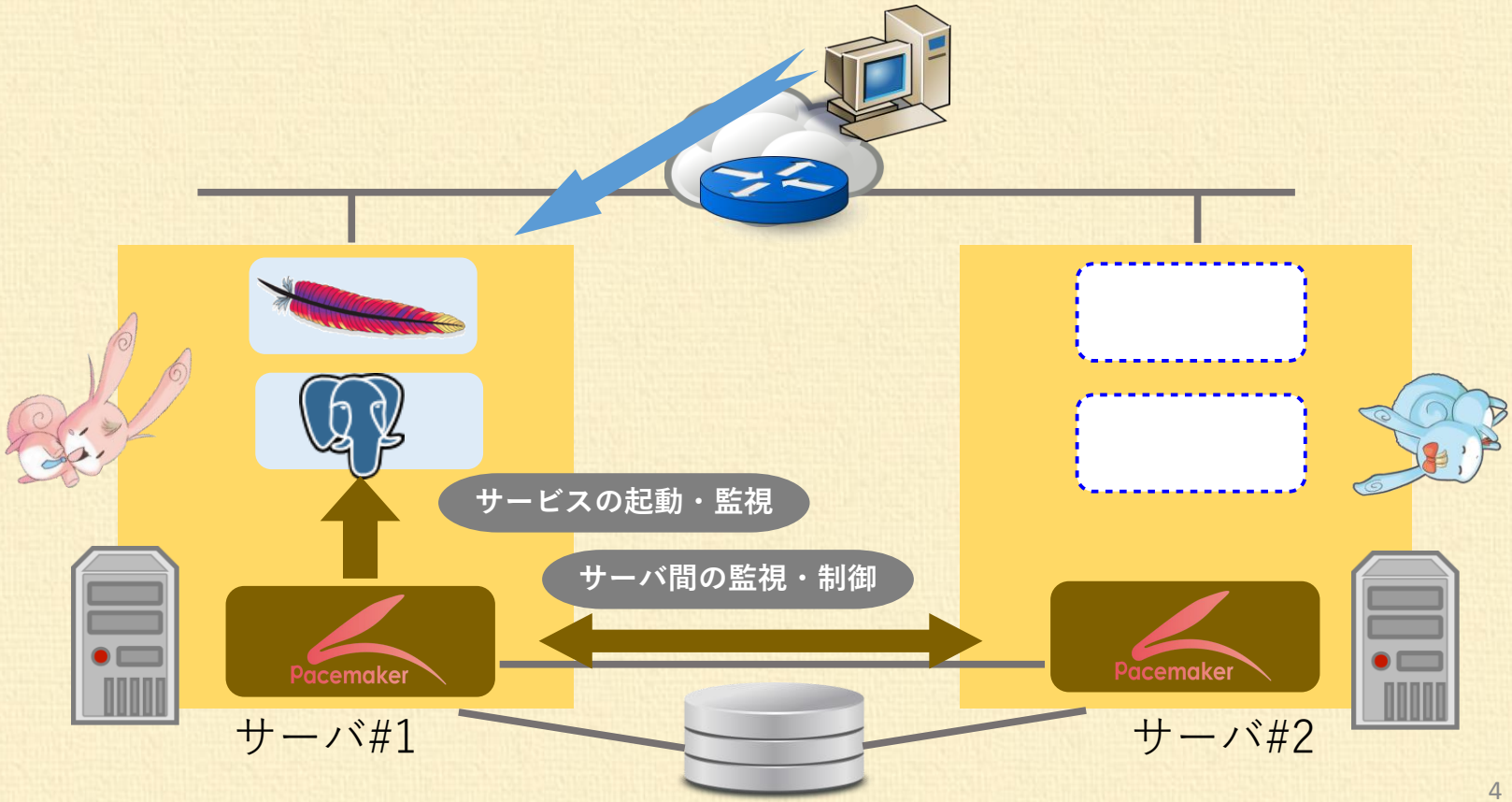
Pacemakerとは?

PacemakerはオープンソースのHAクラスタソフトウェア

High Availability → 高可用性

Pacemakerの概要(1/2)

サーバ・アプリの故障監視

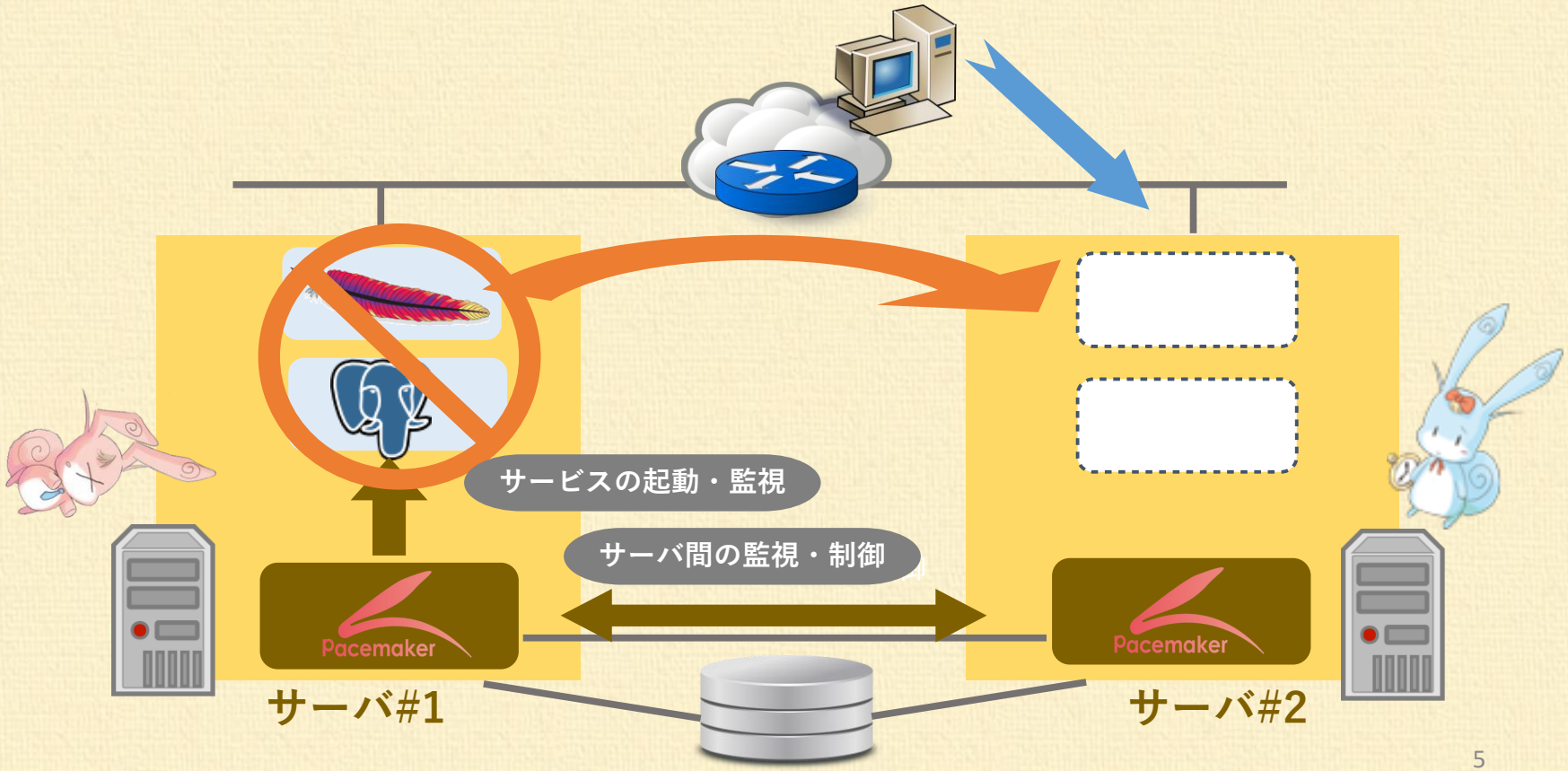


Pacemakerの概要(2/2)

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



Pacemakerの適用領域

サーバ

物理

仮想

クラウド

＋

コンテナ

Docker

今回はコンテナの話をします

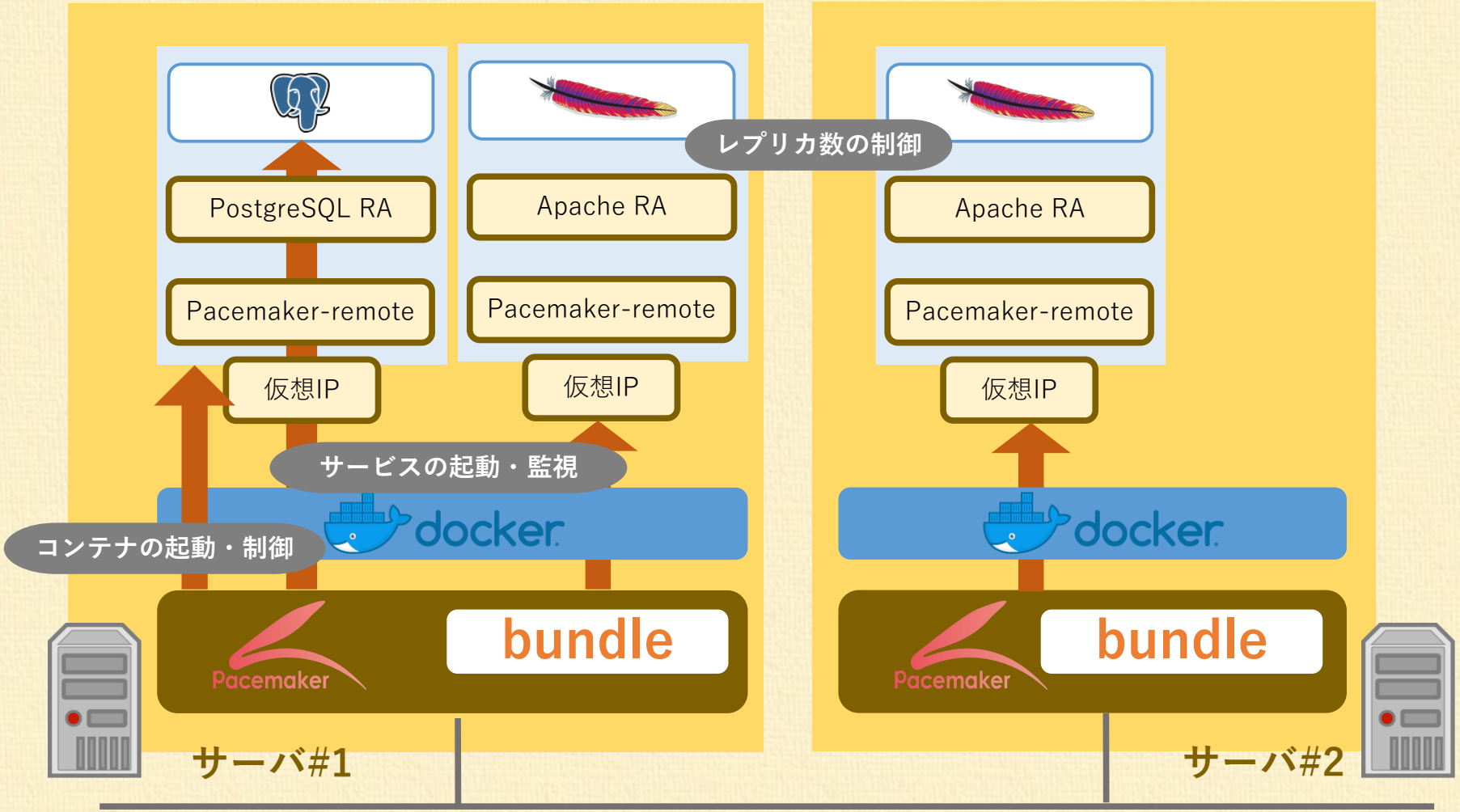
PacemakerでのコンテナHA

Pacemakerのbundle機能を用いてコンテナHAの実現が可能

bundleとは

- コンテナ管理に必要な機能をまとめたPacemakerのリソース
- コンテナとコンテナ内のアプリの起動・終了・監視が可能
- STONITH機能や既存のリソースエージェントと併せて利用可能
- Pacemaker-1.1.17以降で利用可能

bundleのアーキテクチャ



bundle設定例

```
<bundle id="httpd-bundle">
  <docker image="pcmktest: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>
```

コンテナ

ネットワーク

ストレージ

コンテナ内アプリ

コンテナの種類

ステートレスコンテナ

- データを保存する必要が無いコンテナ
- Webサーバ、APサーバ等が対象

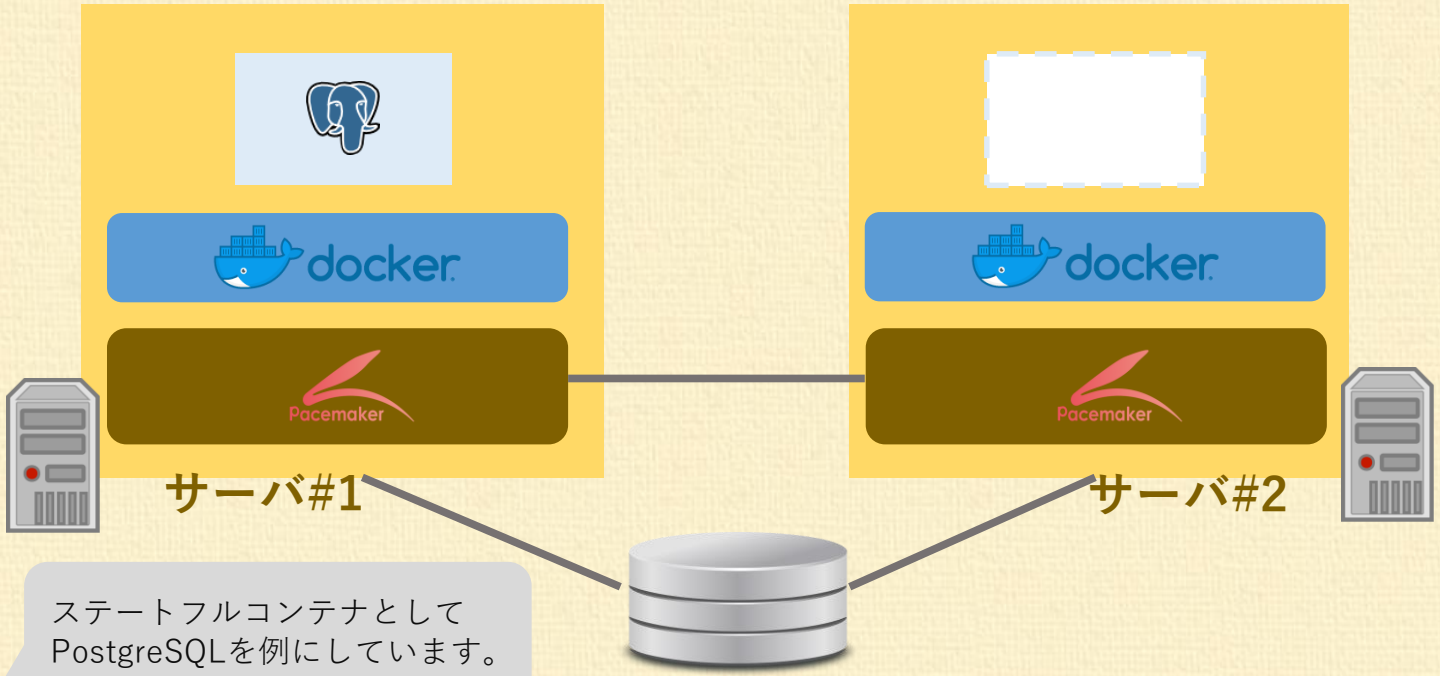
ステートフルコンテナ

- データを保存する必要が有るコンテナ
  - ➢ フェイルオーバ時にはSby側へのデータ引継ぎが必要
- データベースサーバ、ファイルサーバ等が対象

今回はステートフルコンテナに
着目します。

通常のステートフルコンテナのHA構成

コンテナのデータを共有ストレージに保存し
Sby側へのデータの引継ぎを行う



ステートフルコンテナとして
PostgreSQLを例にしています。

ここから本編
今回のモチベーション

共有ストレージの調達にはお金がかかる

だから共有ストレージを使わずに
ステートフルコンテナHAを実現したい

基本100万円以上します。

プライベートでは無理ですね。

「・・・・」

ここでDRBD

DRBDを用いることで共有ストレージを使わずに
Sby側へのデータ引継が可能

DRBDとは

- ネットワーク越しにレプリケーションを行うOSS
- ブロックデバイス単位でレプリケーションを行うためデータの形式に依存しない

加えてLINSTOR

LINSTORを用いることでコンテナへの
DRBD領域のマッピングが簡単に実施可能

LINSTORとは

- **DRBD9の管理ツール**
- **DRBD領域の追加や変更等がコマンドラインで実施可能**
- **DRBD9に利用可能**

LINSTORを入れると動的にDRBD領域が追加できます。

linstor-docker-volumeというプラグインを入れることでDRBD領域のdockerへのマッピングが簡単にできるようです。

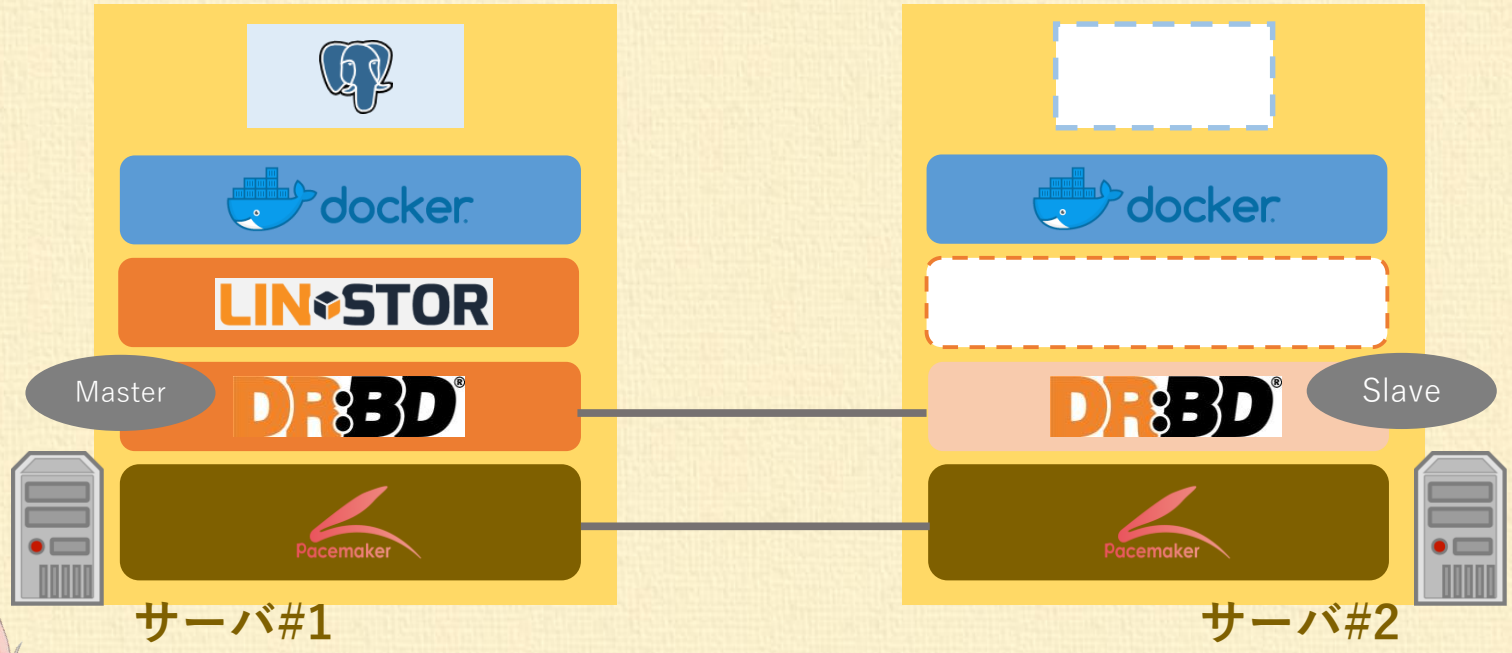
LINSTORを使うので今回使用するDRBDのバージョンは”9”となります。

今回のステートフルコンテナHA構成（共有ストレージ無にて）

DRBDでAct側からSby側へのデータを引継ぎ

LINSTORでDRBD領域をコンテナにマッピング

PacemakerでDRBD9/LINSTOR/コンテナを冗長化



この構成を目指します！！

ここからは以下の流れで構築した流れを紹介します。

PacemakerでDRBD9/LINSTOR/コンテナを冗長化

1. DRBD9の冗長化
2. LINSTORの冗長化
3. PostgreSQLコンテナの冗長化

LINSTORでDRBD領域をコンテナにマッピング

1. linstor-docker-volumeによるコンテナへのDRBDマッピング
2. bundleへのlinstor-docker-volumeの応用

前のスライドにあった「**DRBDでAct側からSby側へのデータを引継ぎ**」はDRBDを起動することで実現できるため割愛します。

PacemakerでDRBD9/LINSTOR/コンテナを冗長化

DRBD9の冗長化

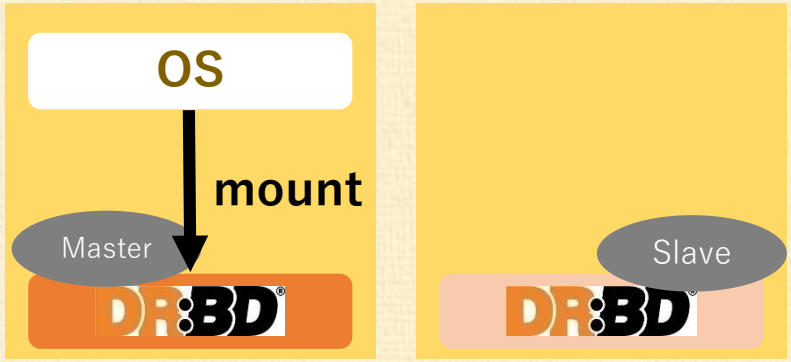
DRBD9の冗長化は2パターンの方式が可能

①DRBD RA方式

Act側がMasterに昇格するように Pacemaker(DRBD RA)側で制御

机上検討ではどちらでも良さそうでしたが、**今回は慣れているDRBD RA方式で実現します。**

②自動プロモーション方式



mountされる側がMasterに昇格するようにDRBD9側で制御

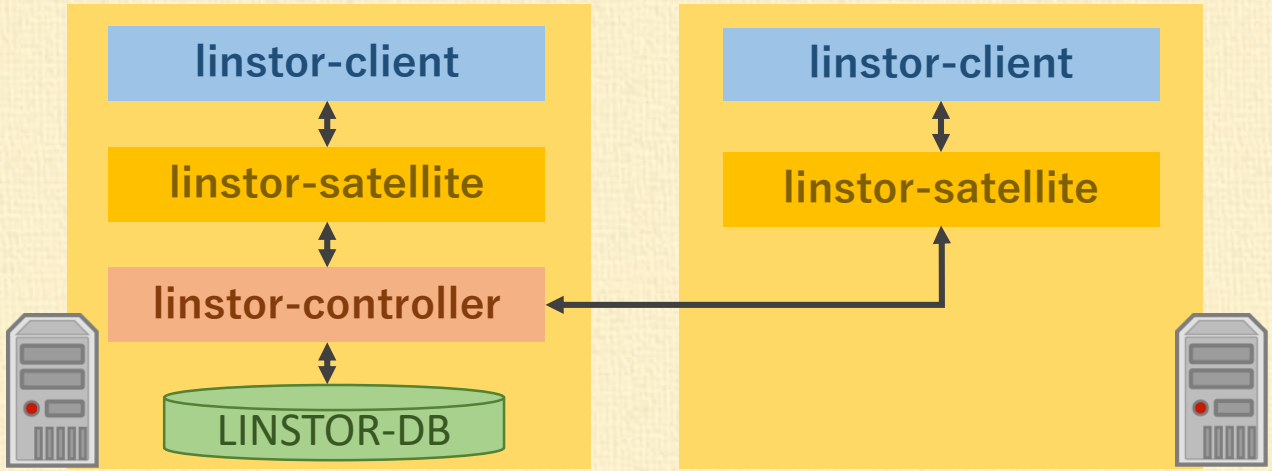
自動プロモーション機能はDRBD9から搭載された機能です。

PacemakerでDRBD9/LINSTOR/コンテナを冗長化

LINSTORの冗長化(1/4)

LINSTORの冗長化のポイントは3つ

1. linstor-controllerデーモンの冗長化
2. LINSTORの管理コマンド(linstor-client)の発行先を常にlinstor-controllerデーモンが動いているノードに向ける
3. linstor DBデータの冗長化



一口にlinstorといっても色々な要素が絡んでいます。

linstor DBにはDRBD9の管理情報が格納されています。

1ノードでのみ動作するlinstor-controllerとlinstor DBを冗長化する必要があります。

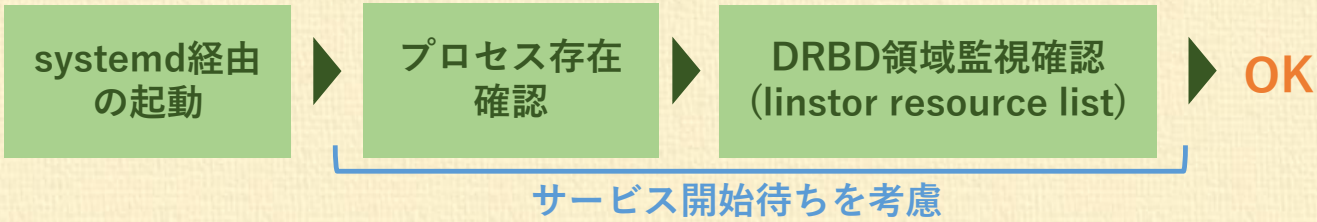
PacemakerでDRBD9/LINSTOR/コンテナを冗長化

LINSTORの冗長化(2/4)

1. linstor-controllerデーモンの冗長化
2. LINSTORの管理コマンド(linstor-client)の発行先を常にlinstor-controllerデーモンが動いているノードに向ける
3. linstor DBの冗長化

linstor-controllerの起動・監視・停止を制御するRAを作成

今回作ったRAの起動フロー



linstor-controllerはsystemd経由で制御できるのでそこを上手く使ってRAを作ります

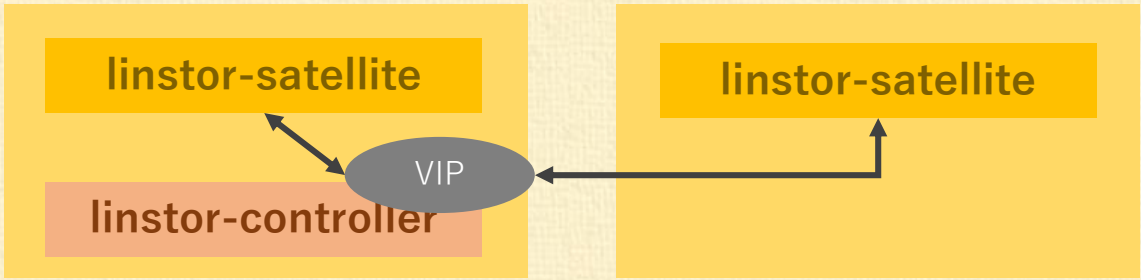
systemdで起動済となってもlinstor-controllerがサービスを提供できるようになるには10秒程度時間を要するため、そこを上手く制御できるようにします

PacemakerでDRBD9/LINSTOR/コンテナを冗長化

LINSTORの冗長化(3/4)

1. linstor-controllerデーモンの冗長化
2. **LINSTORの管理コマンド(linstor-client)の発行先を常にlinstor-controllerデーモンが動いているノードに向ける**
3. linstor DBの冗長化

① linstor-controller用VIPを追加し、Pacemakerで制御する



② linstorコマンド(linstor-client)の発行先を①のVIPに指定する

```
# cat /etc/linstor/linstor-client.conf
[global]
controllers=<VIP>
```

フェイルオーバする際にはlinstor-controllerとVIPは一緒に移動します

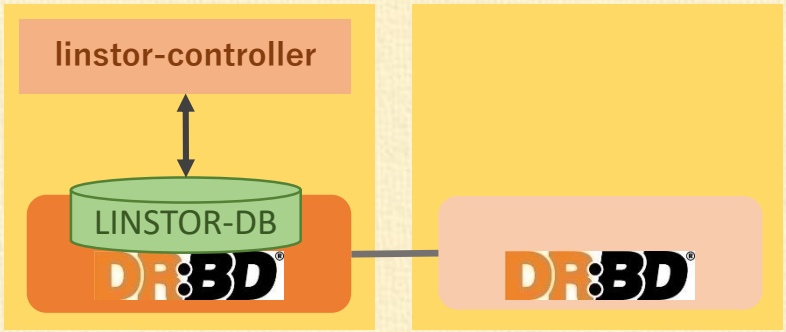
クライアントは常にVIPに繋げることになるため、linstor-controllerがどこで動いているが意識する必要がありません

PacemakerでDRBD9/LINSTOR/コンテナを冗長化

LINSTORの冗長化(4/4)

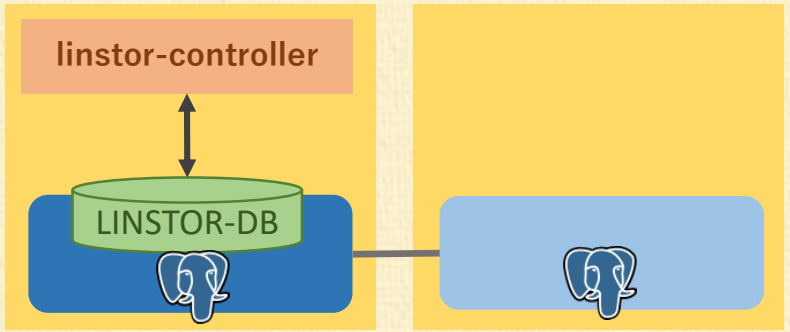
1. linstor-controllerデーモンの冗長化
2. LINSTORの管理コマンド(linstor-client)の発行先を常にlinstor-controllerデーモンが動いているノードに向ける
3. **linstor DBの冗長化**

①DRBDレプリケーション方式



DRBD上に配置し、DRBDのレプリケーションで冗長化する方式

②PostgreSQLレプリケーション方式(PG-REX)



PostgreSQL上に配置しPostgreSQLのレプリケーション機能で冗長化する方式

DRBDレプリケーション方式ではLINSTORを構築する前にDRBD領域が利用できるよう、LINSTOR管理外のDRBD設定が必要となります。

今回はDRBDレプリケーション方式にします。

PostgreSQLレプリケーション方式では全てのDRBD領域がLINSTOR管理となりますが、LINSTOR管理のためにPostgreSQLを両系に構築する必要があります。

PacemakerでDRBD9/LINSTOR/コンテナを冗長化

PostgreSQLコンテナの冗長化

PostgreSQLコンテナのbundle設定

```
<bundle id="postgres-bundle">
      <docker image="container-ha:postgresql12" replicas="1" replicas-per-host="1" options="-v data:/dbfp/pgdata -v wal:/dbfp/pgwal
-v arc:/dbfp/pgarch --volume-driver=linstor --log-driver=journald -e PCMK_logfacility=local1"/>
      <network ip-range-start="192.168.1.103" host-interface="ens4" host-netmask="24">
        <port-mapping id="postgres-port" port="5432"/>
      </network>
      <storage>
        <storage-mapping id="postgresql-log" source-dir="/var/log/pg_log" target-dir="/var/log/pg_log" options="rw"/>
        <storage-mapping id="postgresql-locale" source-dir="/usr/share/zoneinfo/Asia/Tokyo" target-dir="/etc/localtime" options="ro"/>
        <storage-mapping id="postgres-ra-log" source-dir="/dev/log" target-dir="/dev/log" options="rw"/>
      </storage>
      <primitive class="ocf" id="postgres" provider="heartbeat" type="pgsql">
        <meta_attributes id="postgres-meta_attributes">
          <nvpair name="migration-threshold" value="INFINITY" id="postgres-meta_attributes-migration-threshold"/>
        </meta_attributes>
        <instance_attributes id="postgres-attrs">
          <nvpair id="pgctl-attrs" name="pgctl" value="/usr/pgsql-12/bin/pg_ctl"/>
          <nvpair id="psql-attrs" name="psql" value="/usr/pgsql-12/bin/psql"/>
          <nvpair id="pgdata-attrs" name="pgdata" value="/dbfp/pgdata/data"/>
          <nvpair id="pgdba-attrs" name="pgdba" value="postgres"/>
          <nvpair id="pgport-attrs" name="pgport" value="5432"/>
          <nvpair id="pgdb-attrs" name="pgdb" value="template1"/>
        </instance_attributes>
        <operations>
          <op id="postgres-start" name="start" interval="0s" timeout="300s" on-fail="fence"/>
          <op id="postgres-monitor" name="monitor" interval="10s" timeout="60s" on-fail="fence"/>
          <op id="postgres-stop" name="stop" interval="0s" timeout="300s" on-fail="fence"/>
        </operations>
      </primitive>
    </bundle>
```

コンテナ

ネットワーク

ストレージ

Postgre SQL

LINSTORでDRBD領域をコンテナにマッピング

linstor-docker-volumeによるコンテナへのDRBDマッピング

「linstor-docker-volume」プラグインにより、
コンテナからLINSTORを通じてDRBD領域のマッピングが可能

青：コンテナ内のパス
緑：DRBD領域(DRBDリソース)

【使い方の例】

```
# docker run -ti --rm --name=postgres -v data:/dbfp/pgdata -v wal:/dbfp/pgwal -v arc:/dbfp/pgarch --volume-driver=linstor container-ha:postgresql12 /bin/bash
```

--volume-driverにlinstorを指定できるようになり、DRBD領域をコンテナにマッピングできるようになります

次のページでbundleに応用する方法を紹介します

linstor-docker-volumeが無い場合には、必要なDRBD領域をAct側にマウントしてからコンテナにマッピングしなければなりません

LINSTORでDRBD領域をコンテナにマッピング

bundleへのlinstor-docker-volumeの応用

bundleにてlinstorのvolume-driverを使用

青：コンテナ内のパス
緑：DRBD領域(DRBDリソース)

【bundle設定の抜粋】

```
<bundle id="postgres-bundle">
      <docker image="container-ha:postgresql12" replicas="1" replicas-per-host="1" options="-v data:/dbfp/pgdata -v wal:/dbfp/pgwal -v arc:/dbfp/pgarch --volume-driver=linstor --log-driver=journald -e PCMK_logfacility=local1"/>
      <network ip-range-start="192.168.1.103" host-interface="ens4" host-netmask="24">
</bundle>
```

コンテナ

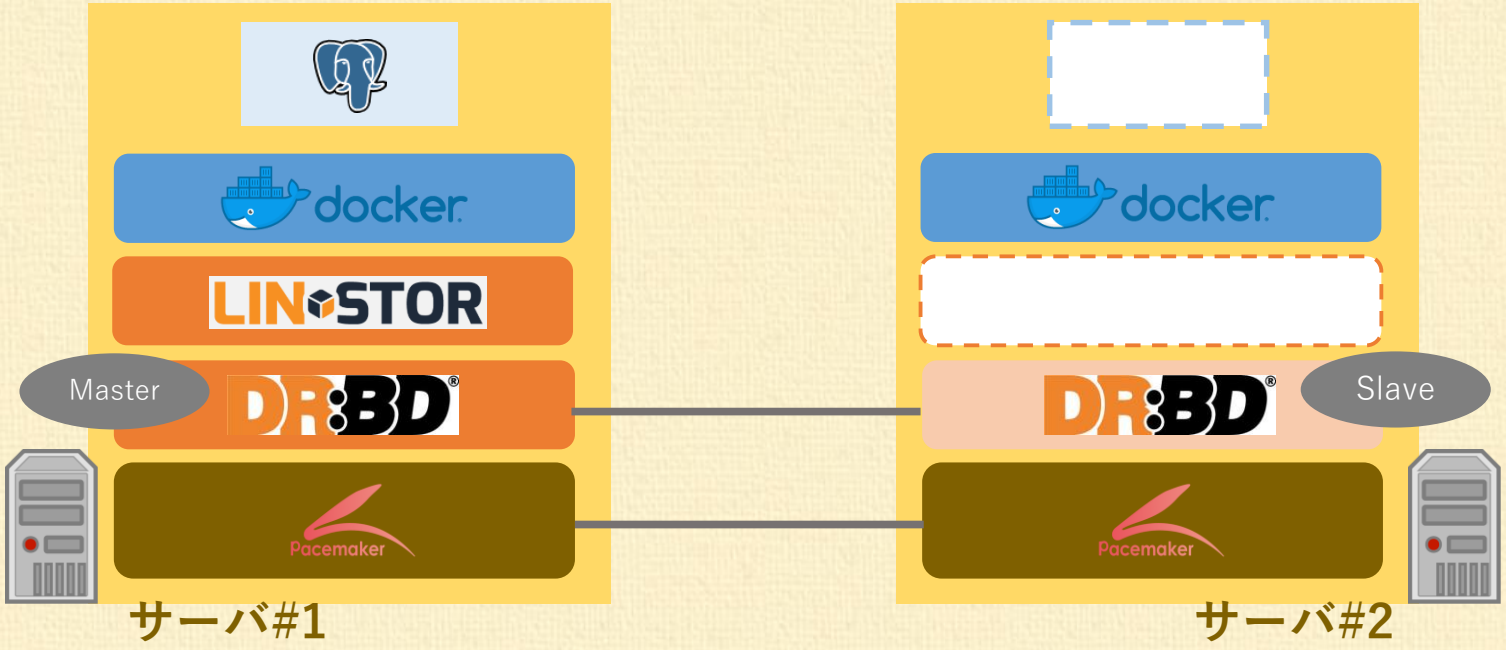
bundleでコンテナ起動時のオプションを指定することが可能です。

これで要件を全て満たすことができました

DRBDでAct側からSby側へのデータを引継ぎ

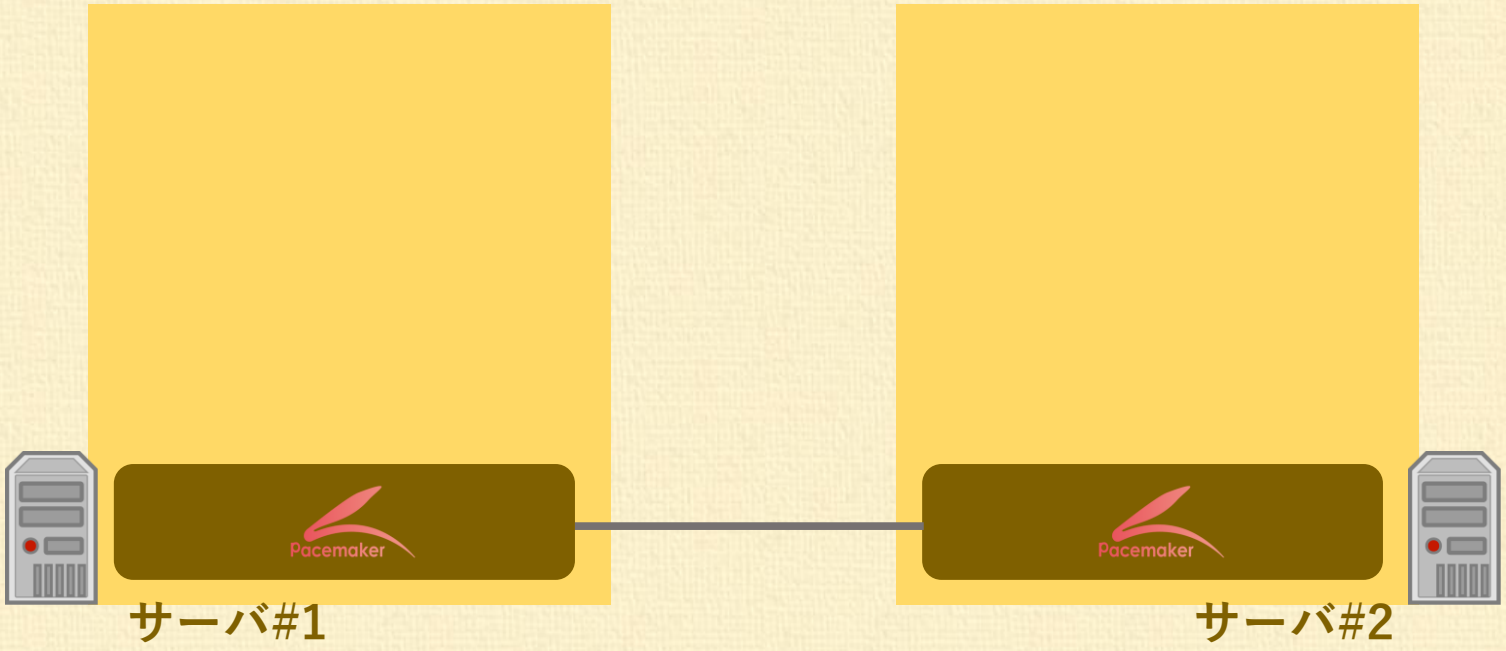
LINSTORでDRBD領域をコンテナにマッピング

PacemakerでDRBD9/LINSTOR/コンテナを冗長化



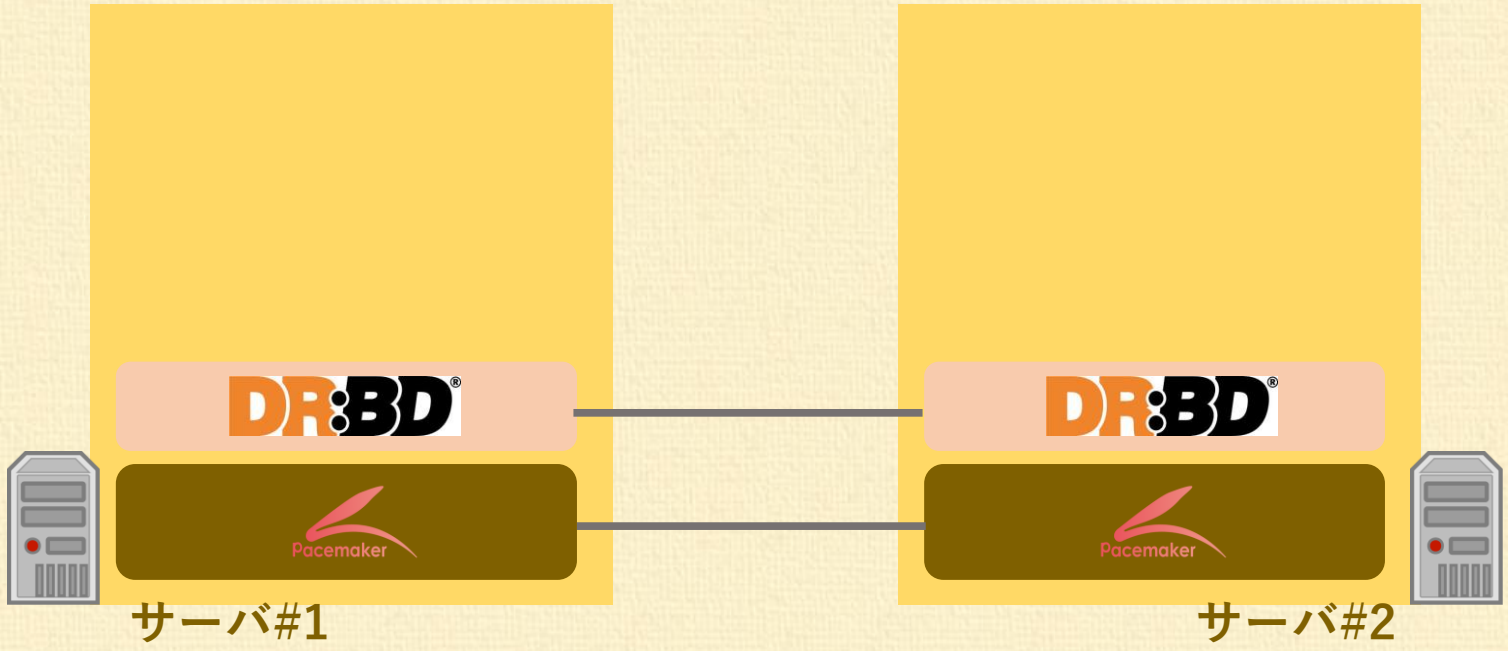
起動の流れ(1/5)

Pacemaker起動



起動の流れ(2/5)

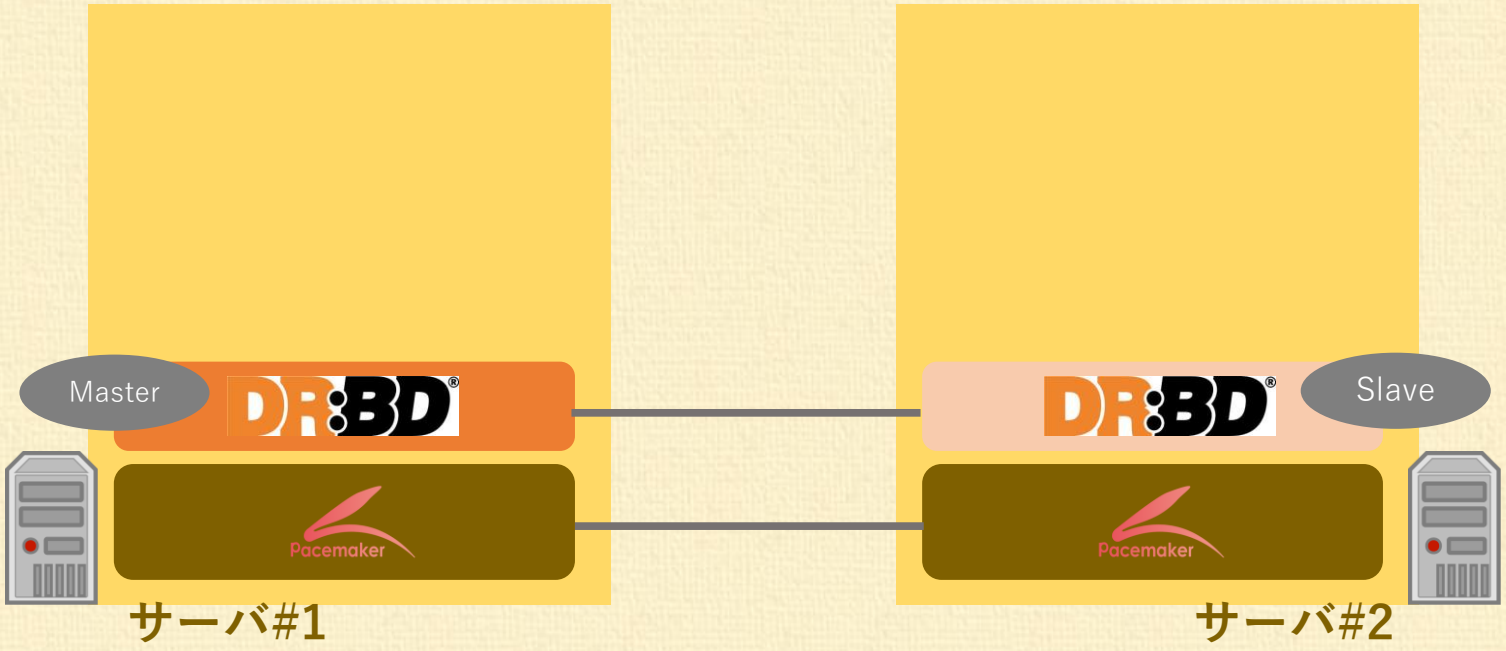
PacemakerによるDRBD起動



今回はDRBD RAを使ってDRBD9の制御をしています。

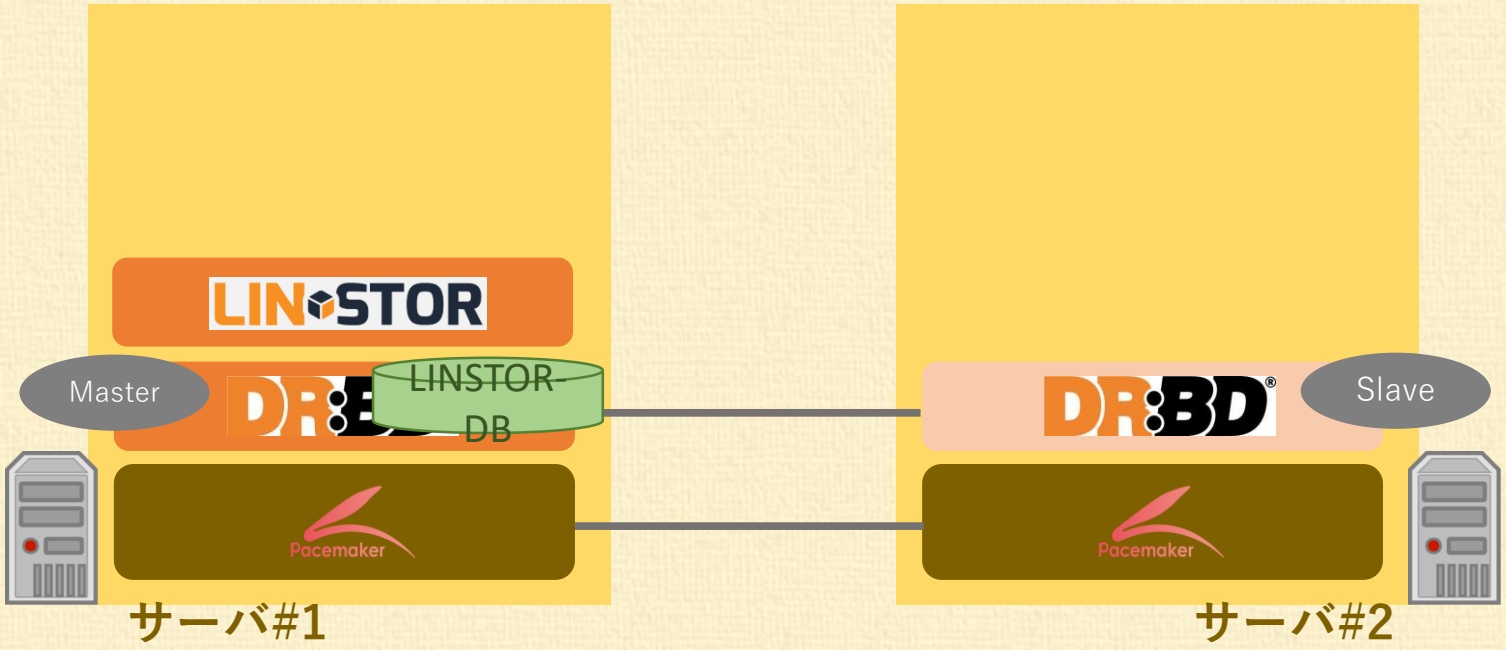
起動の流れ(3/5)

Pacemakerによるサーバ#1のDRBDのMaster昇格



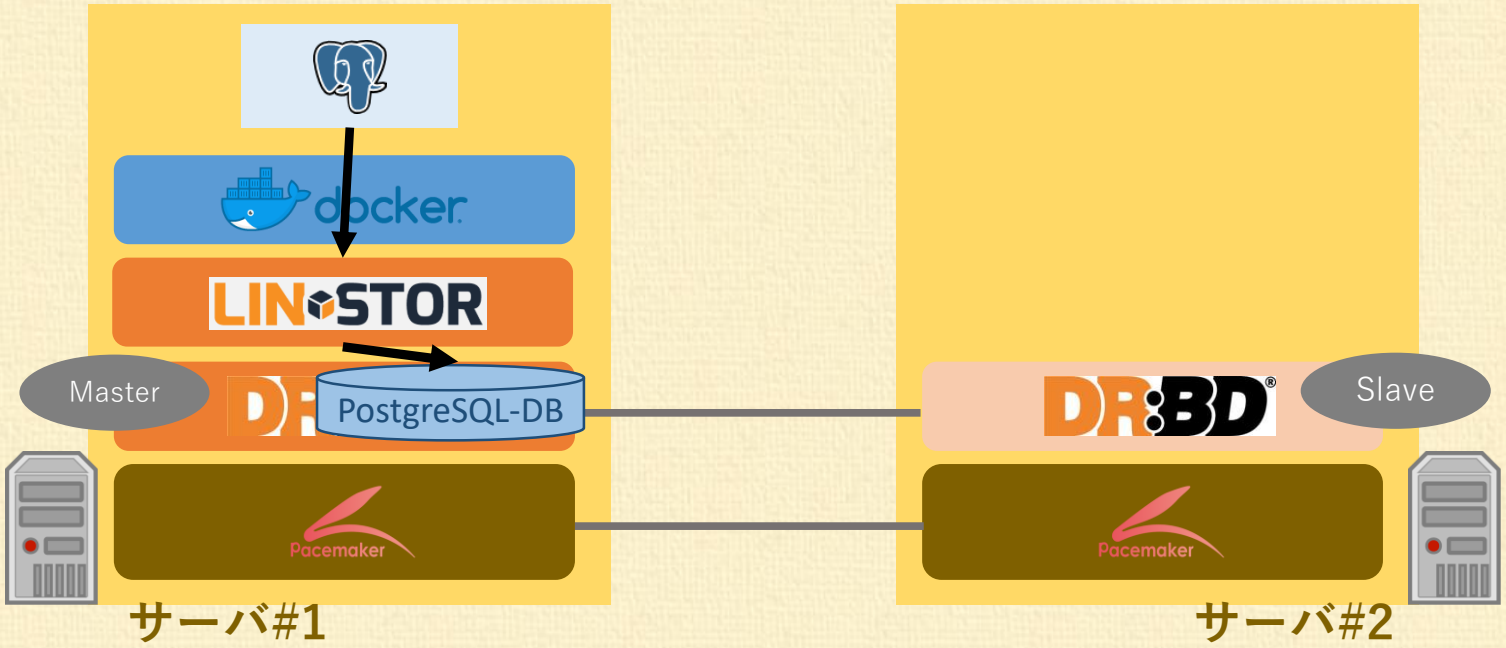
起動の流れ(4/5)

DRBD上のlinstor DBをマウントし、
LINSTOR(linstor-controller)を起動



起動の流れ(5/5)

linstorを通じてコンテナへDRBDのPostgreSQL DB領域を
マッピングしPostgreSQLを起動



Pacemaker上の見え方

```
Stack: corosync
Current DC: node_A (version 1.1.21-1.el7-f14e36f) - partition with quorum
Last updated: Tue Nov  5 20:33:53 2019
Last change: Tue Nov  5 20:33:13 2019 by hacluster via crmd on node A

3 nodes configured
18 resources configured

Online: [ node_B node_A ]
GuestOnline: [ postgres-bundle-0@node_A ]

Active resources:

 Resource Group: LINSTOR-group
     prmFsLINSTORDB     (ocf::heartbeat:Filesystem):    Started node_A
     prmLINSTOR-VIP     (ocf::heartbeat:IPaddr2):       Started node_A
     prmlinstor-controller      (ocf::osc:linstor-controller):  Started node_A
 Master/Slave Set: msDrbd [drbd]
     Masters: [ node_A ]
     Slaves: [ node_B]
 Docker container: postgres-bundle [container-ha:postgresql12]
   postgres-bundle-0 (192.168.1.103)   (ocf::heartbeat:pgsql): Started node_A
 Clone Set: clnDiskd1 [prmDiskd1]
     Started: [ node_Bnode_A ]
 Clone Set: clnDiskd2 [prmDiskd2]
     Started: [ node_Bnode_A ]
 Clone Set: clnPing [prmPing]
     Started: [ node_Bnode_A ]
```

コンテナ

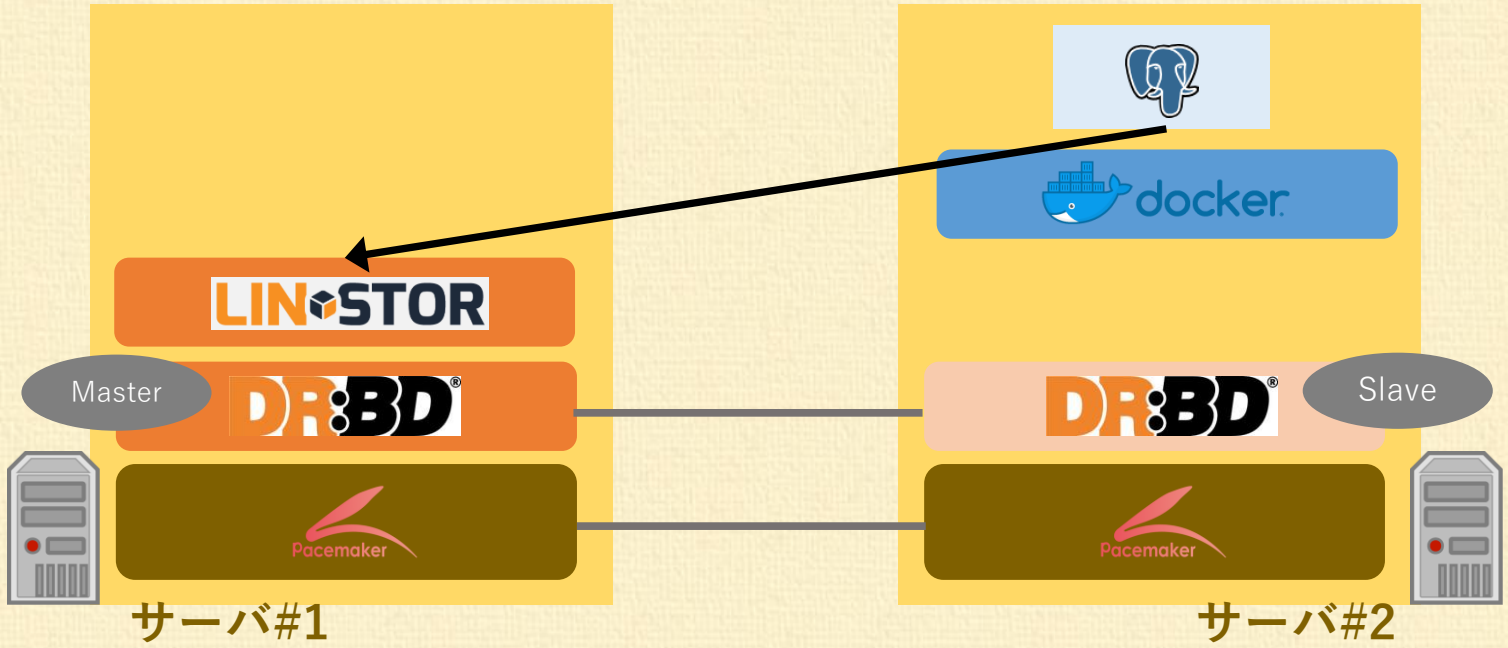
LINSTOR

DRBD

コンテナ内の PostgreSQL

ちなみにコンテナが
フェイルオーバすると？

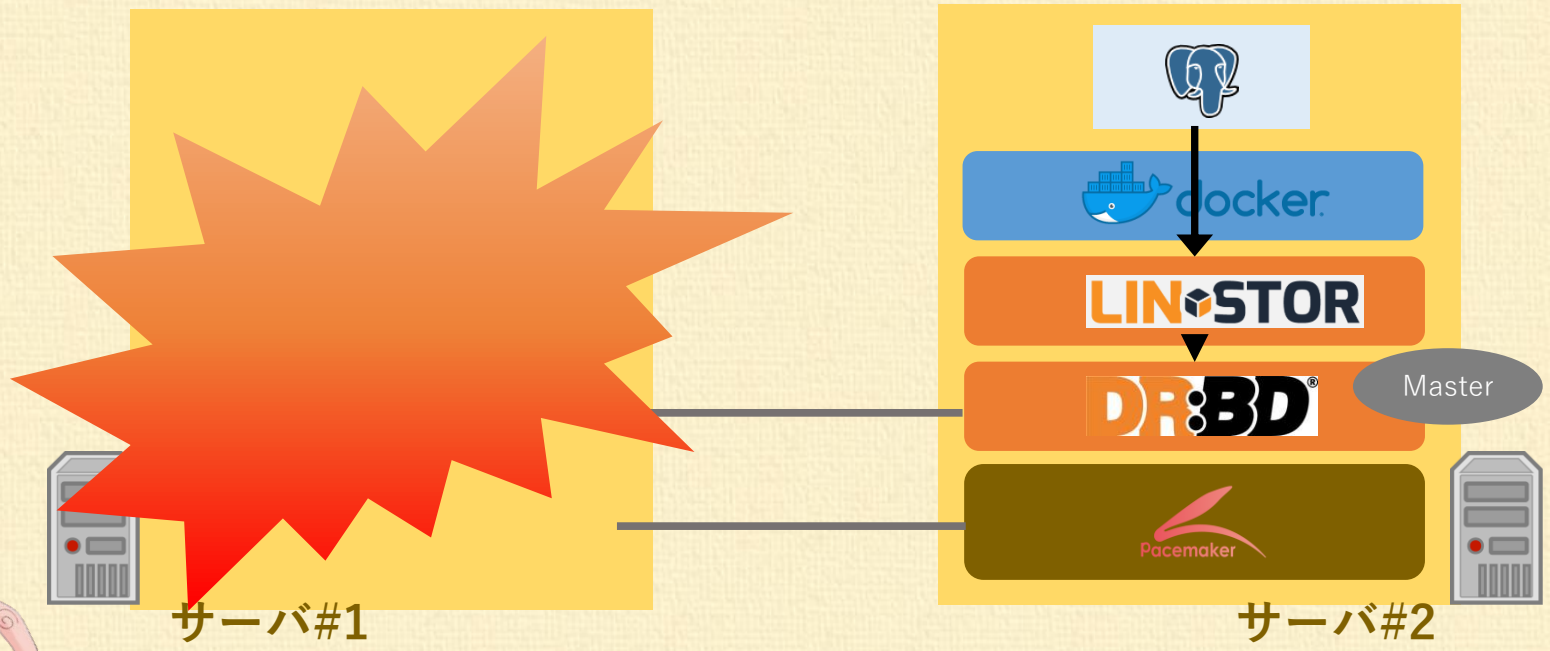
コンテナがフェイルオーバしてもコンテナとDRBDをマッピングすることが可能です。



どこからでもlinstorコマンドが実行できるようになった恩恵です！

サーバ#1が死んでも？

サーバ#1が死んでもLINSTORが冗長化できているのでサービス継続可能です。



linstor DB、linstor-controllerを冗長化できたことによる恩恵です！

ここまでで構築の話は終わりです。

最後にこの構成の長所を
アピールしたいと思います。

俺が作った構成の長所

めっちゃ安上がり！！

→共有ストレージなんていらない

Kubernetesみたいに5台もノードが必要とか言わない！！

→2台で事足りるし、小規模にイイね

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)

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、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

予告

開発コミュニティ(ClusterLabs)では現在
Pacemaker-2.0系の開発が進んでいます。
RHEL 8 のHA Add-Onでも採用されています。

Linux-HA Japanでは、RHEL 8/CentOS 8 以降OS
同梱のPacemakerをベースに、クラスタ界隈を
より盛り上げていこうと準備しています。

詳細は OSC2020 Tokyo/Spring にて！！！！

参考資料

■Pacemakerで学ぶHAクラスタ(OSC 2019 Tokyo/Spring)
http://linux-ha.osdn.jp/wp/wp-content/uploads/OSC2019_Tokyo-Spring.pdf

■コンテナを止めるな!(OSC 2018 Tokyo/Fall)
https://www.slideshare.net/ksk_ha/pacemakerhakubernetes-121194991

■PacemakerでDockerコンテナをクラスタリング(OSC 2017 Tokyo/Spring)
http://linux-ha.osdn.jp/wp/wp-content/uploads/OSC2017_Tokyo_Spring.pdf

ご清聴ありがとうございました