

---

# すぐに始められる！ 最新のPacemakerで始める 高可用クラスタ入門



---

オープンソースカンファレンス2014 Tokyo/Fall  
LinuxHA-Japan 飯田 雄介

---

# 自己紹介

---

## 名前

- 飯田 雄介 (いいだ ゆうすけ)

## 所属

- Linux-HA Japanプロジェクト
- 株式会社メトロシステムズ

## 主な活動

- Pacemaker本体の機能改善や、外部ツールの開発を行っています。
    - この仕事を始めて8年が経ち、現在までにコミュニティに提供したパッチが51件採用されました。
    - Linux-HA Japanからpm\_logconvやpm\_crmgenといったツールを提供しています。
-

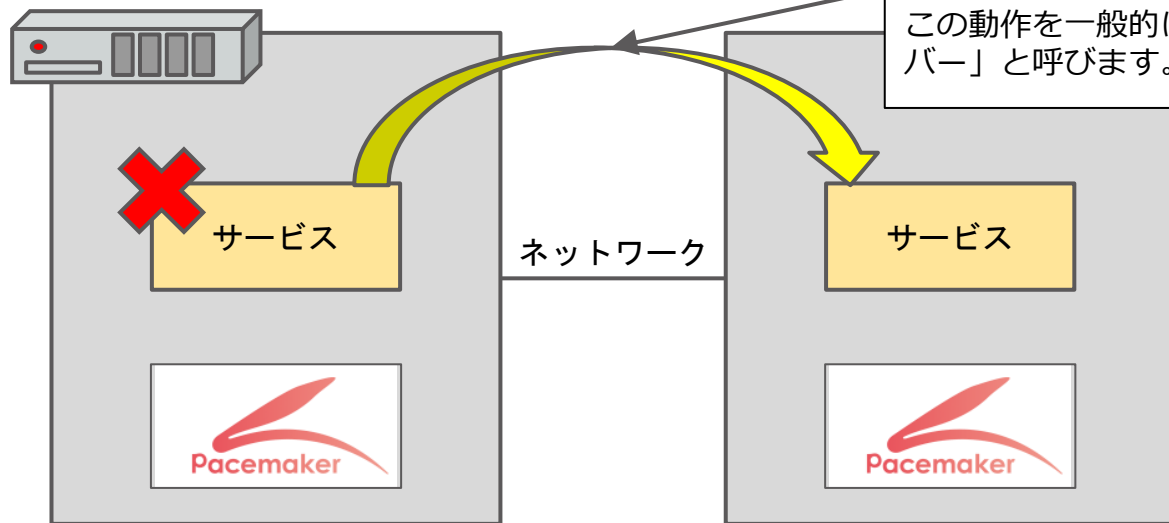
# 本日のおはなし

---

1. 最新Pacemaker-1.1.12での変更点を紹介
2. Pacemakerクラスタを構築してみよう！
3. フェイルオーバーの動きを体験してみよう！

# 高可用クラスタってなに？

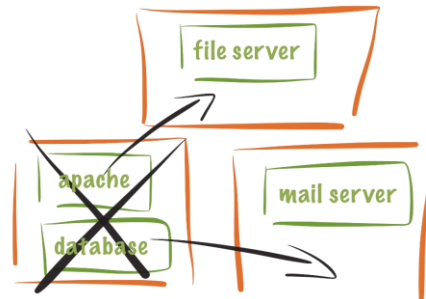
- 高可用クラスタとは複数のコンピュータをネットワークで繋いでサービスが故障しても継続して動き続けられるようにした仕組みのことです。



# Pacemakerとは？

---

- Pacemaker（ペーすめーカー）とはオープンソースで開発されている高可用クラスタソフトウェアのことです。
  - Pacemakerは、ハードウェアとソフトウェアの両方の障害を監視します。障害が発生した場合、自動的にサービスをフェイルオーバーさせてくれます。
  - Linux-HA Japanプロジェクトでは現在「Pacemaker-1.0.13-2.1」を公開しています。

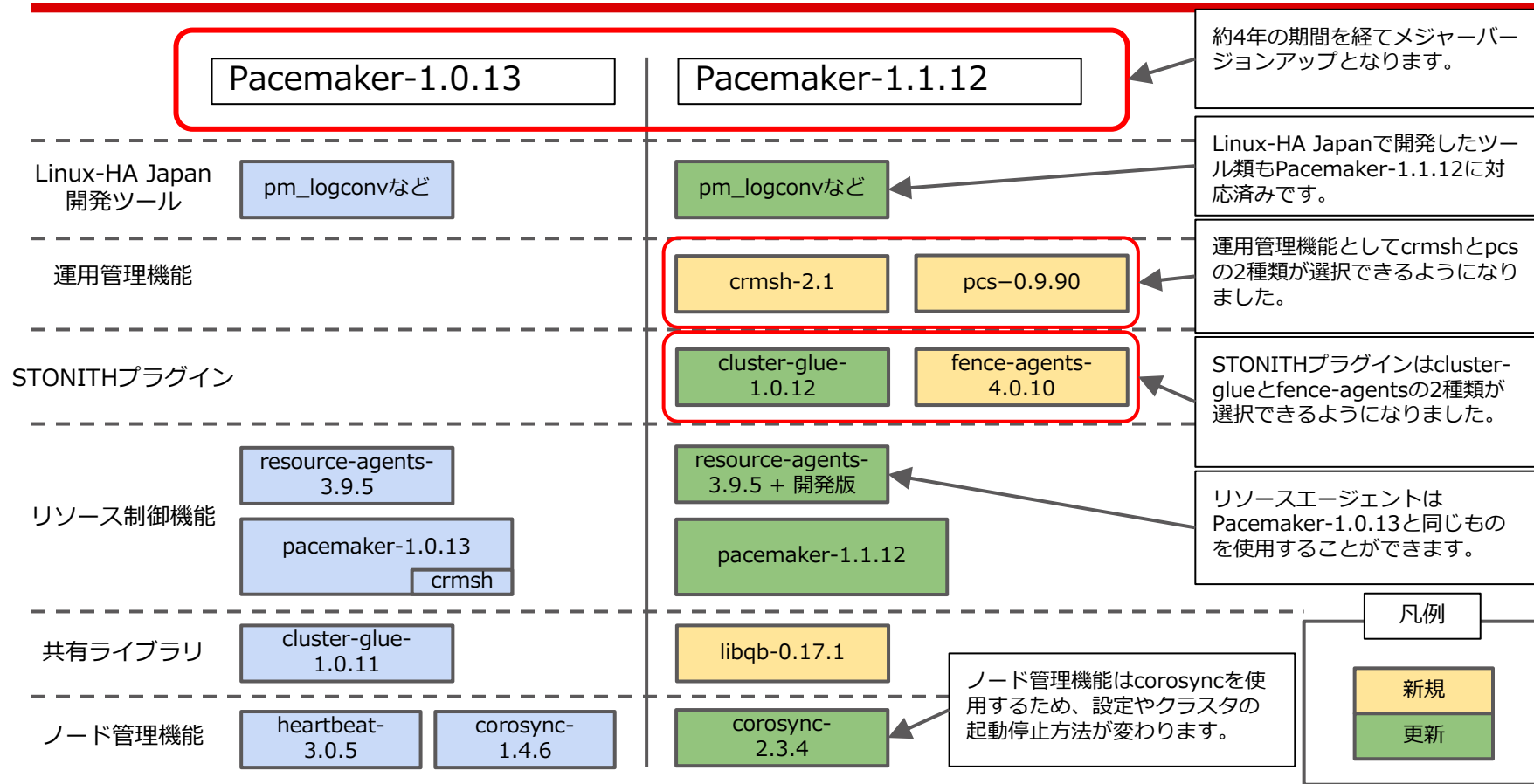


---

# 最新Pacemaker-1.1.12での 変更点を紹介

---

# コンポーネントが刷新されました！



# ノード管理機能にCorosyncを採用しました！

---

- Pacemaker-1.1.12からLinux-HA Japanではノード管理機能にCorosyncを採用しました。
- Corosyncを使用することによるメリット
  - Corosyncを使うとノード故障の検知速度が向上し、フェイルオーバー時間を短縮することができます。
  - Corosyncを使うとHeartbeatより大規模なクラスタ構成が組めるようになります。
    - Heartbeatでは6ノード、80リソース構成程度が限界だったが、Corosyncでは16ノード、200リソース程度までの動作実績があります。



---

**Pacemakerクラスタを構築してみよう！**

---

# デモ環境について（1）

---

- Pacemaker、Apache、PostgreSQL、Tracを使用したWEBサービスのActive/Standby構成を作ります。
    - デモ用の環境ですが、一般的なWEBサービスに必要なリソースはすべて組み込んであります。(仮想IPや共有ディスクも含め)
  - この環境では次に挙げる故障に対応できます。
    - リソース故障
    - ノード故障
    - ディスク故障（内蔵・共有ディスク）
    - ネットワーク故障（サービス・インターコネクトLAN）
-

# デモ環境について (2)

---

## ■ デモ環境はKVMで仮想マシンを2台使用しています。

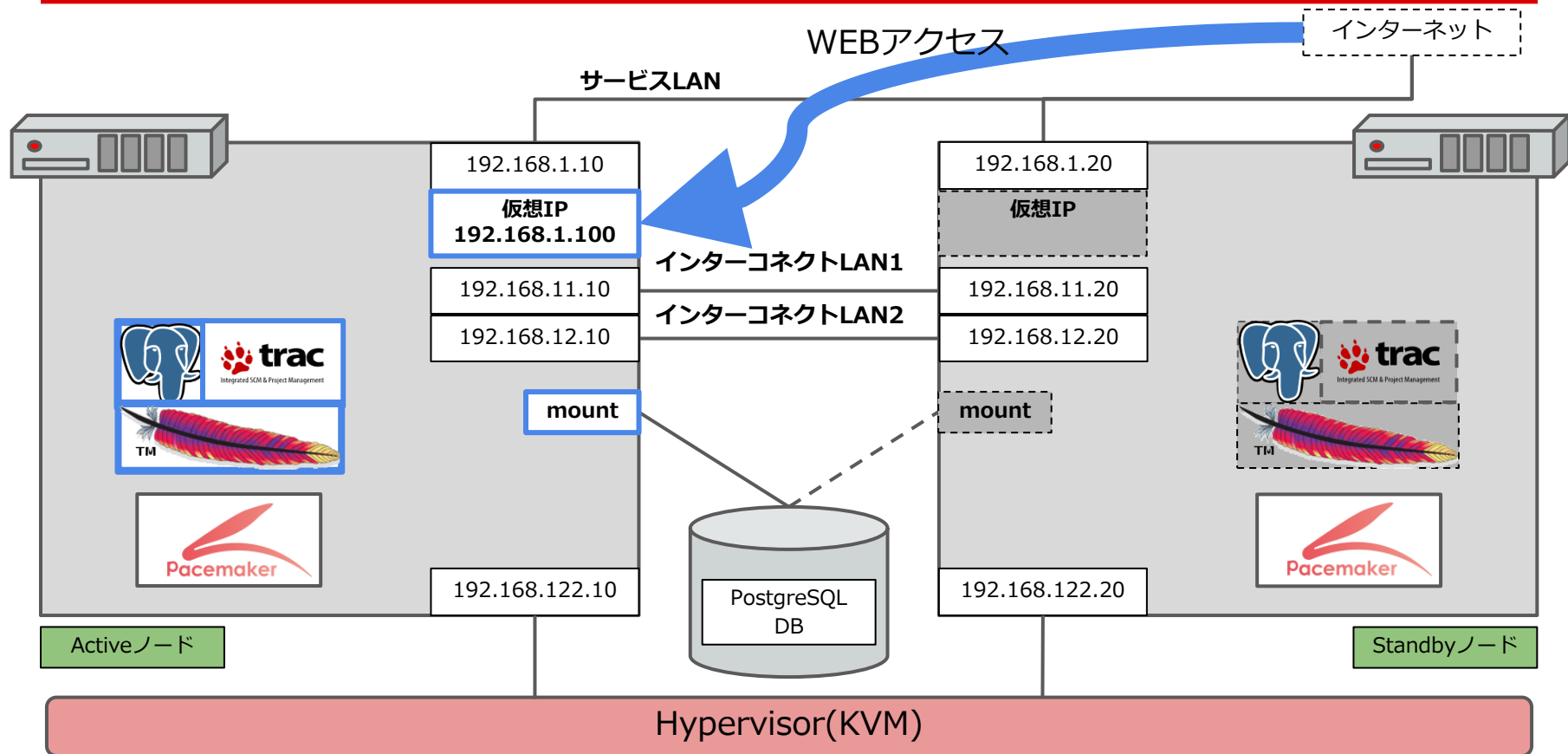
### □ ハードウェア

- CPU : 1コア、メモリ : 2GB、ディスク : 10GB

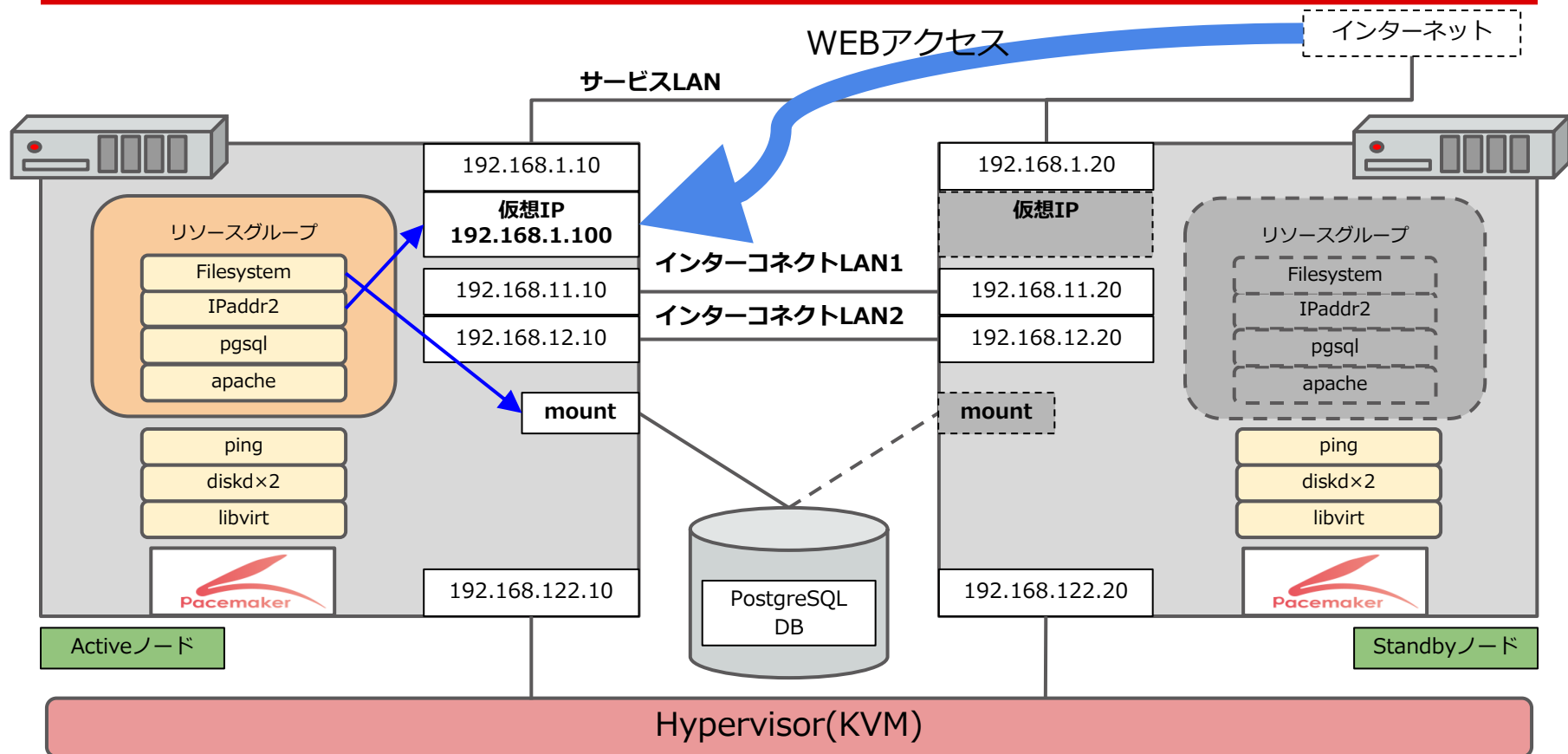
### □ ソフトウェア

- OS : CentOS-6.5-x86 64
  - ・ CentOSは「最小構成」を選択してインストールしたものを使用しています。
  - ・ baseリポジトリが参照できる状態になっています。
- PostgreSQL-9.3.5 (公式サイトから取得)
- Trac-1.0.1 (easy installコマンドを使ってインストール)
- httpd-2.2.15-31 (OS同梱版を使用)
- libvirt-client-0.10.2-29 (OS同梱版を使用)

# こんな環境を作ります



# Pacemakerのリソースに表すと、このようになります



# 前提条件

---

- 本日のデモでは説明を簡略化するため、以下のセットアップを予め行っています。
    - Apache、PostgreSQL、Tracはインストール済みで動作する状態となっています。
      - Tracは"http://192.168.1.100/osc2014"のアドレスでアクセスできるように構築してあります。
    - ホストマシン/仮想マシン間は鍵認証を使ってrootユーザがパスワードなしでログインできる状態にしてあります。
    - selinuxとiptablesは無効化しています。
  - 作業はrootユーザで行います。
-

# Pacemakerのインストール

---

1. Pacemakerパッケージの取得
2. Pacemakerのインストール

# Pacemakerパッケージの取得

---

- Linux-HA JapanのHPからPacemakerリポジトリパッケージを取得します。
  - (注)本パッケージはリリース候補版です。正式版は後日Linux-HA Japanから公開いたします。

```
# wget http://linux-ha.sourceforge.jp/wp/wp-content/uploads/pacemaker-repo-1.1.12-0.RC1.el6.x86_64.rpm
(snip)
2014-10-14 13:21:30 (115 KB/s) - `pacemaker-repo-1.1.12-0.RC1.el6.x86_64.rpm'
へ保存完了 [21185244/21185244]
```



# Pacemakerのインストール(1)

---

- 先ほど取得したRPMをインストールします。
  - Pacemakerのパッケージ群(RPM)がローカルディスクに配置されます。

```
# rpm -ivh pacemaker-repo-1.1.12-0.RC1.el6.x86_64.rpm
# ls /opt/linux-ha/pacemaker/rpm/
pacemaker-1.1.12-1.el6.x86_64.rpm      corosync-2.3.4-1.el6.x86_64.rpm
resource-agents-3.9.5-1.589.b6443.el6.x86_64.rpm
crmsh-2.1-1.el6.x86_64.rpm
(snip)
# ls /etc/yum.repos.d/pacemaker.repo
/etc/yum.repos.d/pacemaker.repo
```

# Pacemakerのインストール(2)

---

- yumコマンドでPacemakerをインストールします。
  - 依存関係のあるパッケージは自動的にインストールされます。

```
# yum install pacemaker-all -y  
(snip)  
Complete!
```

# Pacemakerクラスタを動かすための設定

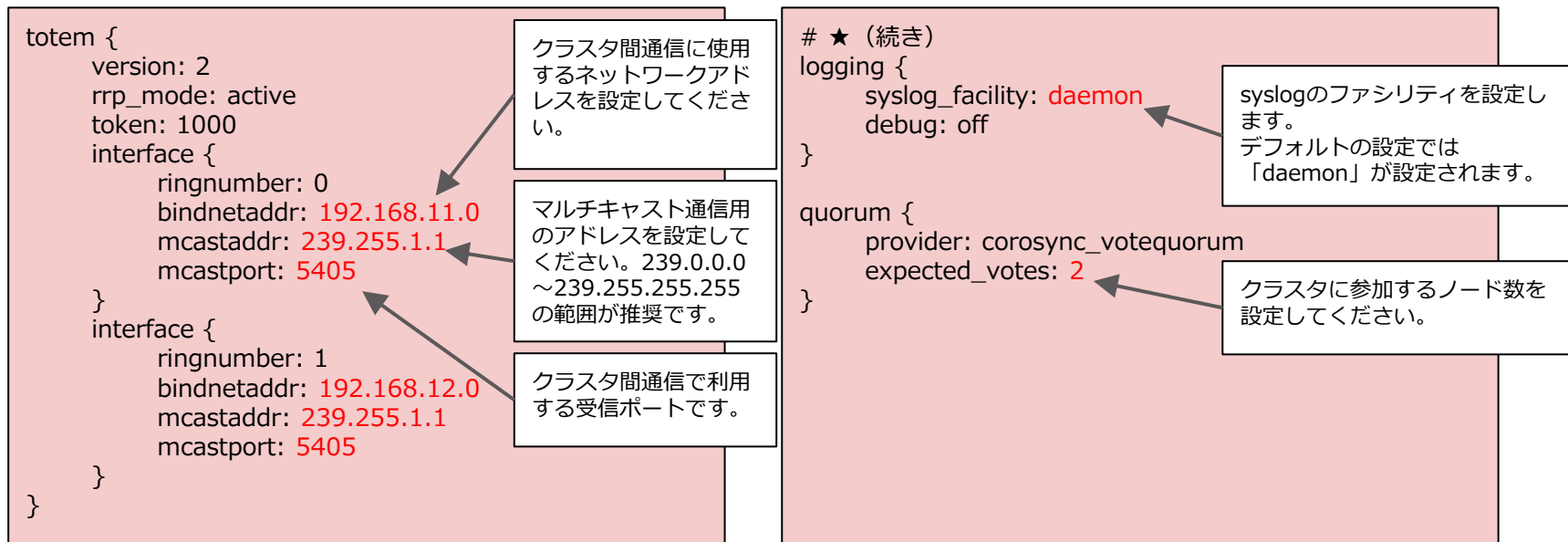
---

1. corosync.confの作成
2. 認証鍵ファイルの作成（corosync通信用）
3. /etc/sysconfig/pacemakerの設定
4. クラスタ起動スクリプトの修正

# corosync.confの作成

■ /etc/corosync/corosync.confを以下のように作成します。

- クラスタを組む全てのマシンに同じファイルを配置してください。
- 赤字の設定については自身の環境に合わせて適宜変更してください。



# 認証鍵ファイルの作成

---

- 以下のコマンドを実行してクラスタ間通信に使用する認証鍵ファイルを作成します。
  - 生成された認証鍵ファイルをクラスタを組む全てのマシンにコピーしてください。

```
# corosync-keygen -l  
# ls -la /etc/corosync/authkey  
-rw-r--r-- 1 root root 128  8月 20 16:56 14 /etc/corosync/authkey  
# scp -p /etc/corosync/authkey server02:/etc/corosync/authkey
```

# /etc/sysconfig/pacemakerの設定

- 本設定でPacemakerのプロセスが故障した時の振る舞いを指定できます。
  - 本設定を追加すると、Pacemakerのプロセスが故障したノードはhalt状態となり、他のノードからはノードに故障が発生したと判断されるようになります。

```
# vi /etc/sysconfig/pacemaker
```

```
(snip)
```

```
67 # Enable this for rebooting this machine at the time of process (subsystem)  
failure
```

```
68 export PCMK_fail_fast=yes
```

```
69
```

```
(snip)
```

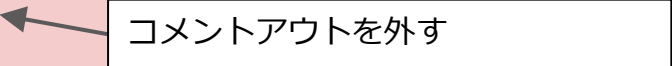
コメントアウトを外し、設定値を「yes」に

# クラスタ起動スクリプトの修正 (1)

---

- corosyncプロセスが故障した場合にcorosyncの watchdogを動作させるため、起動スクリプトの52行目を有効にします。

```
# vi /etc/init/pacemaker.combined.conf
(snip)
50
51  # if you use watchdog of corosync, uncomment the line below.
52  pidof corosync || false
53
54  pidof crmd || stop corosync
(snip)
```



コメントアウトを外す

# クラスタ起動スクリプトの修正 (2)

---

- クラスタ起動中にOSをshutdownした場合にクラスタを正常に停止させるため、起動スクリプトの5行目に設定を追加します。

```
# vi /etc/init/pacemaker.combined.conf
```

```
(snip)
```

```
3 # Starts Corosync cluster engine and Pacemaker cluster manager.
```

```
4
```

```
5 stop on runlevel [0123456]
```

この1行を追加

```
6 kill timeout 3600
```

```
7 respawn
```

```
(snip)
```



# クラスタを起動する

---

1. クラスタを起動する
2. クラスタの状態を確認する

# クラスタを起動する

---

- 以下のコマンドを実行してクラスタを起動します。

- Pacemaker-1.1.12からはUpstart経由(CentOS6)で起動します。

```
# initctl start pacemaker.combined  
pacemaker.combined start/running, process 25490
```

- クラスタ停止コマンドはこちら

```
# initctl stop pacemaker.combined  
pacemaker.combined stop/waiting
```

# クラスタの状態を確認する

---

- `crm_mon`※を実行してノードの状態が「Online」になっていることを確認します。

```
# crm_mon -fAD1  
Online: [ server01 server02 ]  
(snip)
```

2台のマシンの状態が  
Onlineになっているこ  
とを確認します。

※ : `crm_mon`はクラスタの状態を確認するためのコマンドです。

# クラスタにリソースを管理させる

---

1. リソース定義ファイルを作成する
2. リソース定義ファイルの簡単な解説
3. リソース定義ファイルをクラスタに読み込ませる
4. サービスが起動したことを確認してみよう

# リソース定義ファイルを作成する(1)

---

- 今回のデモ構成では以下のものをリソース化します。
  - サービスリソース
    - apache
    - pgsql
    - IPaddr2(仮想IPの管理)
    - Filesystem(mountの管理)
  - 監視リソース
    - ping (ネットワークを監視するリソース)
    - diskd (ディスクを監視するリソース)

# リソース定義ファイルを作成する(2)

---

- STONITH※リソース

- libvirt (KVMなどの仮想マシンを再起動させるリソース)

※：STONITHとは相手ノードの状態が確認できなくなった時に、そのノードの電源を停止させることで、リソースが完全に停止したことを保証させるための機能です。

詳細：<http://linux-ha.sourceforge.jp/wp/archives/3729>

---

# リソース定義ファイルの簡単な解説

```
primitive prmFS ocf:heartbeat:Filesystem ¥
params ¥
  fstype="ext4" ¥
  run_fsck="force" ¥
  device="/dev/vdb1" ¥
  options="barrier=0" ¥
  directory="/pgsqldb" ¥
op start interval="0s" timeout="60s" on-fail="restart" ¥
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
op stop interval="0s" timeout="60s" on-fail="fence"
```

```
primitive prmVIP ocf:heartbeat:IPAddr2 ¥
params ¥
  ip="192.168.1.100" ¥
  nic="eth3" ¥
  cidr_netmask="24" ¥
op start interval="0s" timeout="60s" on-fail="restart" ¥
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
op stop interval="0s" timeout="60s" on-fail="fence"
```

```
primitive prmDB ocf:heartbeat:pgsql ¥
params ¥
  pgctl="/usr/pgsql-9.3/bin/pg_ctl" ¥
  psql="/usr/pgsql-9.3/bin/psql" ¥
  pgdata="/pgsqldb/data" ¥
  start_opt="-p 5432" ¥
  pgdba="postgres" ¥
  pgport="5432" ¥
  pgdb="template1" ¥
op start interval="0s" timeout="300s" on-fail="restart" ¥
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
op stop interval="0s" timeout="300s" on-fail="fence"
```

```
primitive prmWEB ocf:heartbeat:apache ¥
op start interval="0s" timeout="300s" on-fail="restart" ¥
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
op stop interval="0s" timeout="300s" on-fail="fence"
```

(snip)

```
primitive prmVIP ocf:heartbeat:IPAddr2 ¥
params ¥
  ip="192.168.1.100" ¥
  nic="eth3" ¥
  cidr_netmask="24" ¥
op start interval="0s" timeout="60s" on-fail="restart" ¥
op monitor interval="10s" timeout="60s" on-fail="restart" ¥
op stop interval="0s" timeout="60s" on-fail="fence"
```

※赤字の設定値については環境に合わせて適宜変更してください。  
※詳細な定義については付録を参照してください

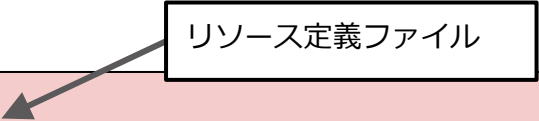
# リソース定義ファイルをクラスタに読み込ませる

---

- crmコマンド※を実行してクラスタにリソース定義ファイルを読み込ませます。

```
# crm configure load update osc2014.crm
```

リソース定義ファイル



※crmコマンドはPacemakerクラスタを操作する運用管理コマンドです。

---



# クラスタの状態を確認する

- crm\_monを実行して、リソースがActiveノード上で「Started」状態になったことを確認します。

```
# crm_mon -fAD1
Online: [ server01 server02 ]
prnStonith1      (stonith:external/libvirt): Started server02
prnStonith2      (stonith:external/libvirt): Started server01
Resource Group: grpTrac
  prnFS (ocf::heartbeat:Filesystem):
  prnVIP (ocf::heartbeat:IPaddr2):
  prnDB (ocf::heartbeat:pgsql):
  prnWEB(ocf::heartbeat:apache):
Clone Set: clnDiskd1 [prnDiskd1]
  Started: [ server01 server02 ]
Clone Set: clnDiskd2 [prnDiskd2]
  Started: [ server01 server02 ]
Clone Set: clnPing [prnPing]
  Started: [ server01 server02 ]
(snip)
```

Started server01  
Started server01  
Started server01  
Started server01

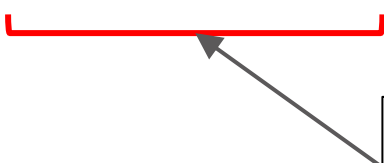
リソースがActiveノード上で「Started」状態になっていることを確認します。

# サービスが起動できたことを確認してみよう

---

- WEBブラウザを起動して、下記アドレスにアクセスします。Tracに接続できたら無事構築完了です。

- <http://192.168.1.100/osc2014>



このIPはリソース定義の  
IPaddr2で設定した仮想  
IPです。

---

**フェイルオーバーの動きを体験してみよう！**

---

# Pacemakerはどんな時にフェイルオーバーしてくれるの？

---

- 例えば、次に挙げるような状況になった時、リソースをフェイルオーバーしてくれます。
  - リソース故障
    - 例) httpdプロセスが故障により停止してしまった時
  - ノード故障
    - 例) 電源故障によりノードが停止してしまった時
  - ディスクやネットワークの故障

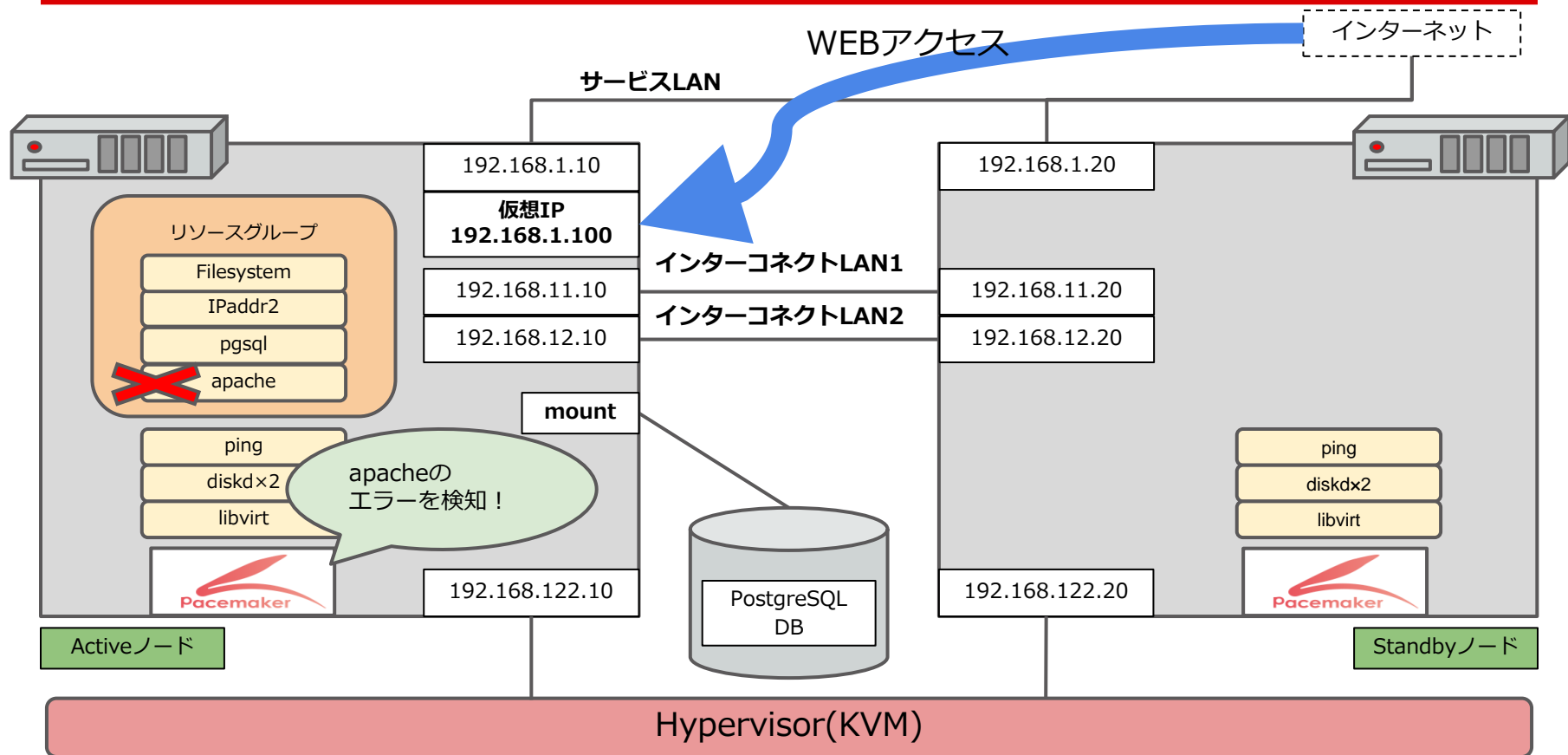
# リソース故障によるフェイルオーバーのデモ

---

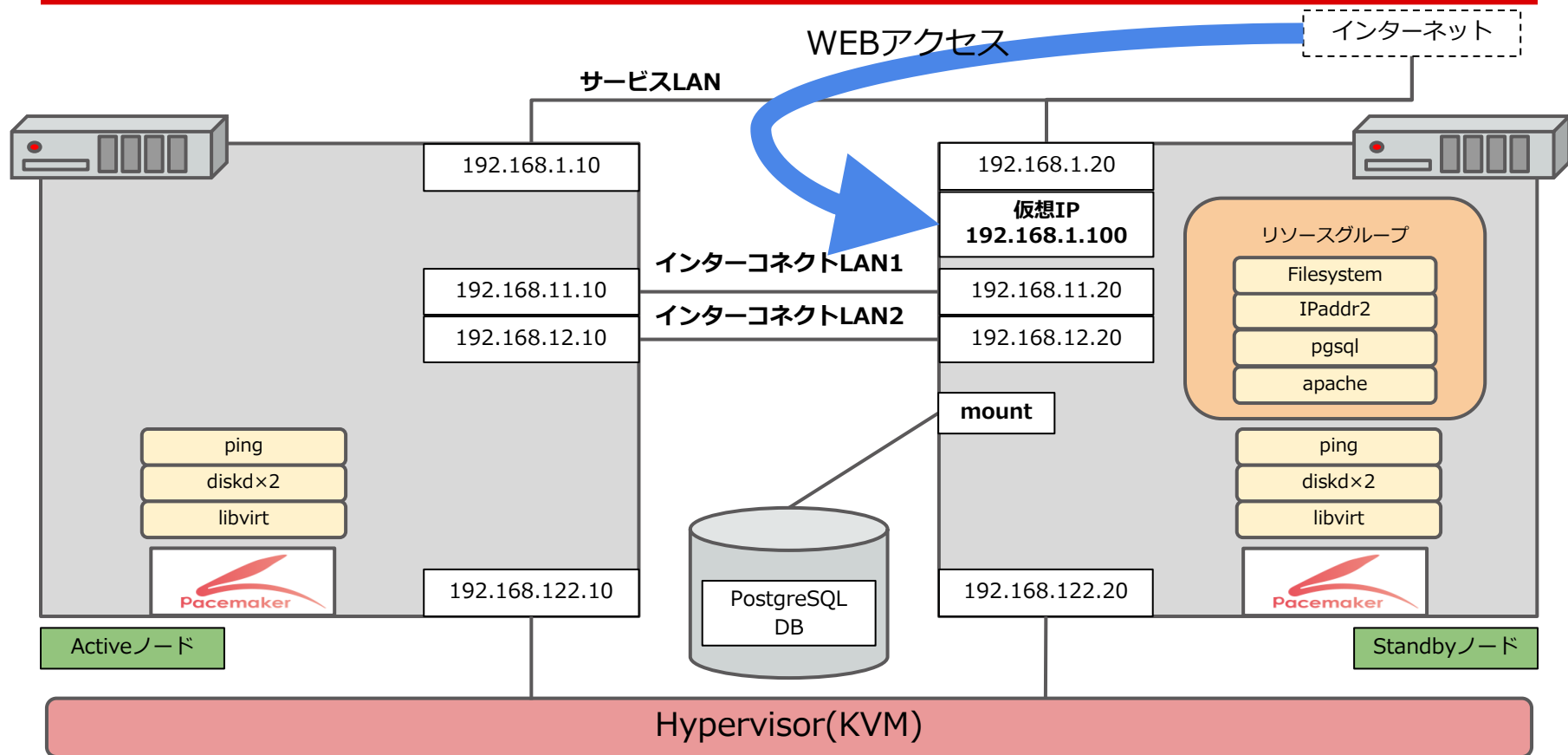
- 今回のデモではActiveノードでApache(httpd)プロセスをkillコマンドで強制停止させることで、フェイルオーバーを起こさせます。

```
# kill -9 <httpdの親プロセス>
```

# リソース故障によるフェイルオーバー (故障発生時)



# リソース故障によるフェイルオーバー (フェイルオーバー後)



# リソース故障発生後のクラスタ状態

```
# crm_mon -fAD1
```

```
Online: [ server01 server02 ]
```

```
Resource Group: grpTrac
```

```
  prmFS (ocf::heartbeat:Filesystem):
```

```
  prmVIP (ocf::heartbeat:IPaddr2):
```

```
  prmDB (ocf::heartbeat:pgsql):
```

```
  prmWEB(ocf::heartbeat:apache):
```

```
(snip)
```

```
Migration summary:
```

```
* Node server02:
```

```
* Node server01:
```

```
  prmWEB: migration-threshold=1 fail-count=1 last-failure='Tue Oct 7 03:04:48 2014'
```

```
Failed actions:
```

```
  prmWEB_monitor_10000 on server01 'not running' (7): call=57, status=complete, last-rc-change='Tue Oct 7 03:04:48 2014', queued=0ms, exec=0ms
```

リソースはフェイルオーバーされ、Standbyノード上で起動されます。

「Migration summary」に故障リソースの情報が表示されます。

「Failed actions」に故障発生時のオペレーション情報が表示されます。

Started server02  
Started server02  
Started server02  
Started server02

(注)本来の運用では故障原因を取り除き、fail-countをクリアするなどして故障発生前の状態に戻しますが、今回のデモでは時間の都合上復旧の説明・手順は省き、一旦クラスタを再起動させる手順を取ります。



# ノード故障によるフェイルオーバーのデモ

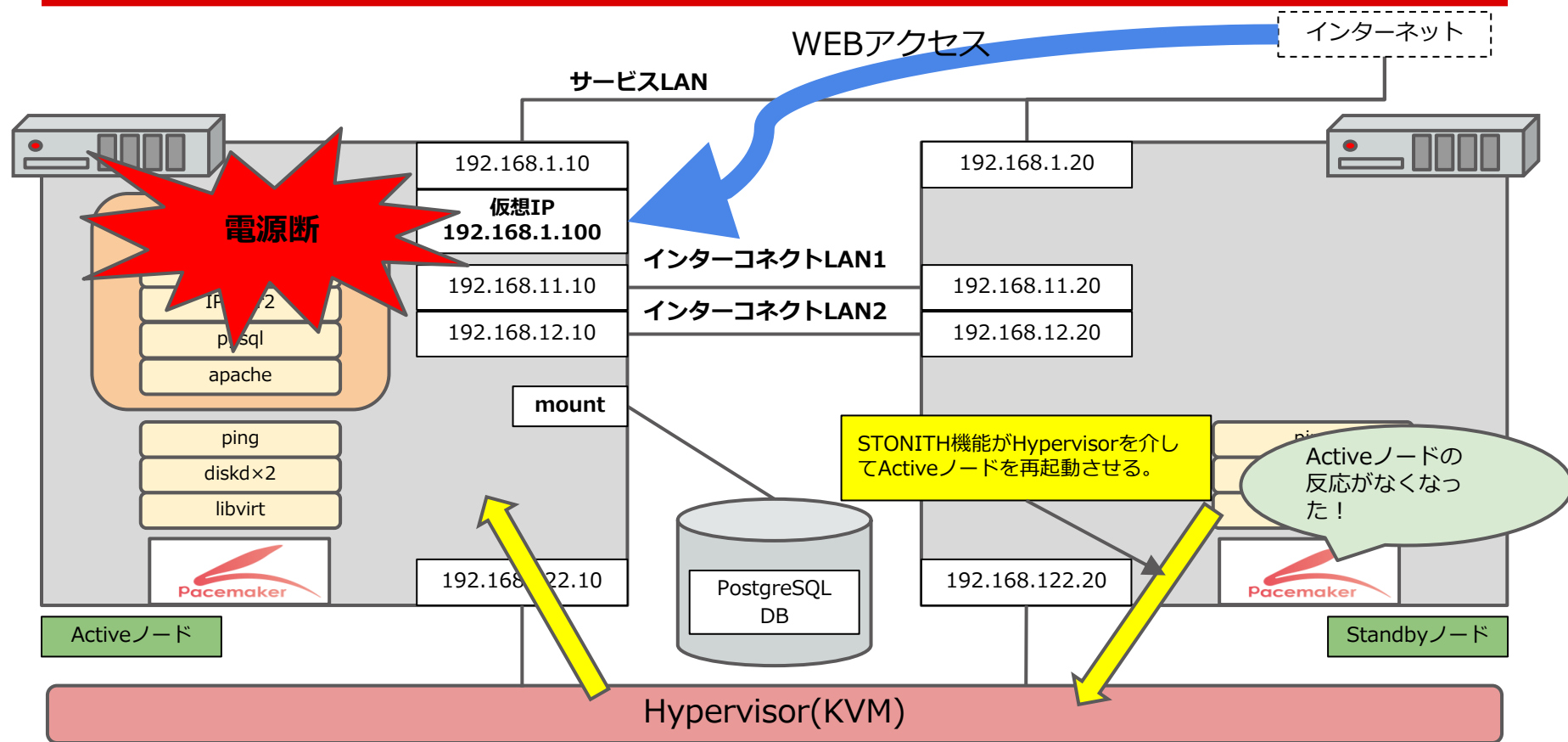
---

- 今回のデモではノードを仮想マシンで作っているので、ホストマシンから仮想マシンの強制停止コマンドを実行して、フェイルオーバーの動きを確認します。

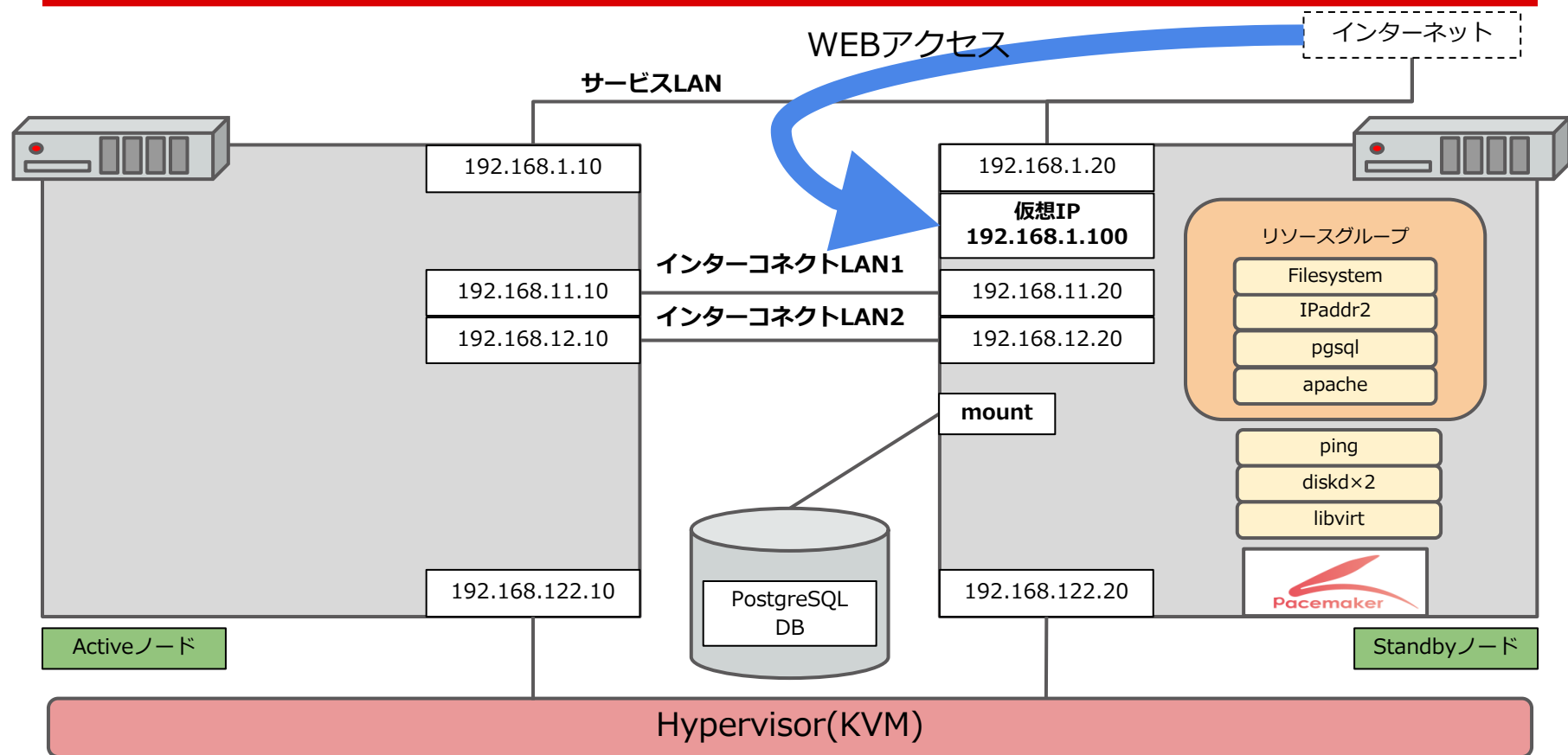
```
# virsh destroy server01
```

ドメイン server01 は強制停止されました

# ノード故障によるフェイルオーバー (故障発生時)



# ノード故障によるフェイルオーバー (フェイルオーバー後)



# ノード故障発生後のリソース状態

```
# crm_mon -fAD1
```

```
Online: [ server02 ]
```

```
OFFLINE: [ server01 ]
```

```
prnStonith1 (stonith:external/libvirt): Started server02
```

```
Resource Group: grpTrac
```

```
prnFS (ocf::heartbeat:Filesystem):
```

```
prnVIP (ocf::heartbeat:IPaddr2):
```

```
prnDB (ocf::heartbeat:pgsql):
```

```
prnWEB(ocf::heartbeat:apache):
```

```
Started server02
```

```
Started server02
```

```
Started server02
```

```
Started server02
```

```
Clone Set: clnDiskd1 [prnDiskd1]
```

```
Started: [ server02 ]
```

```
Stopped: [ server01 ]
```

```
Clone Set: clnDiskd2 [prnDiskd2]
```

```
Started: [ server02 ]
```

```
Stopped: [ server01 ]
```

```
Clone Set: clnPing [prnPing]
```

```
Started: [ server02 ]
```

```
Stopped: [ server01 ]
```

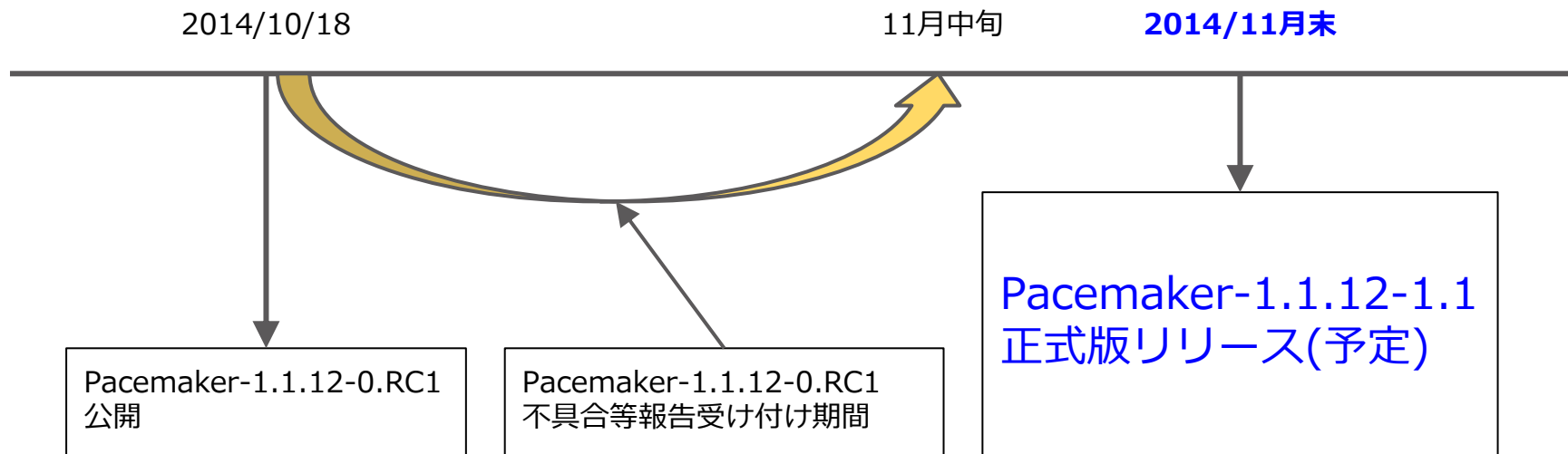
```
(snip)
```

電源が落ちたノードは  
OFFLINEとして認識されます。

リソースはフェイルオーバー  
され、Standbyノード上で起  
動されます。

# さいごに

## ■ 今後のリリーススケジュールについて



# ご清聴ありがとうございました。

---

## 【展示情報】

- 3F「306」教室にて「Linux-HA Japan Project」ブース展示中です！

## 【コミュニティ情報】

- Welcome to Linux-HA Japan
  - <http://linux-ha.sourceforge.jp/wp/>
- Linux-HA Japan日本語メーリングリスト
  - <http://linux-ha.sourceforge.jp/wp/ml>



Pacemaker-1.1.12  
をヨロシクね！



# 付録：リソース定義ファイル（1）

```
### Cluster Option ###
property no-quorum-policy="ignore" ¥
stonith-enabled="true" ¥
startup-fencing="false"

### Resource Defaults ###
rsc_defaults resource-stickiness="INFINITY" ¥
migration-threshold="1"

### Group Configuration ###
group grpTrac ¥
  prmFS ¥
  prmVIP ¥
  prmDB ¥
  prmWEB ¥

### Clone Configuration ###
clone clnPing ¥
  prmPing

clone clnDisk1 ¥
  prmDisk1

clone clnDisk2 ¥
  prmDisk2

### Master/Slave Configuration ###

### Fencing Topology ###
fencing_topology ¥
  server01: prmStonith1 ¥
  server02: prmStonith2
```

```
### Primitive Configuration ###
primitive prmFS ocf:heartbeat:Filesystem ¥
  params ¥
    fstype="ext4" ¥
    run_fsck="force" ¥
    device="/dev/vdb1" ¥
    options="barrier=0" ¥
    directory="/pgsqldb" ¥
    op start interval="0s" timeout="60s" on-fail="restart" ¥
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥
    op stop interval="0s" timeout="60s" on-fail="fence"

primitive prmVIP ocf:heartbeat:IPAddr2 ¥
  params ¥
    ip="192.168.1.100" ¥
    nic="eth3" ¥
    cidr_netmask="24" ¥
    op start interval="0s" timeout="60s" on-fail="restart" ¥
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥
    op stop interval="0s" timeout="60s" on-fail="fence"

primitive prmDB ocf:heartbeat:pgsql ¥
  params ¥
    pgctl="/usr/pgsql-9.3/bin/pg_ctl" ¥
    psql="/usr/pgsql-9.3/bin/psql" ¥
    pgdata="/pgsqldb/data" ¥
    start_opt="-p 5432" ¥
    pgdba="postgres" ¥
    pgport="5432" ¥
    pgdb="template1" ¥
    op start interval="0s" timeout="300s" on-fail="restart" ¥
    op monitor interval="10s" timeout="60s" on-fail="restart" ¥
    op stop interval="0s" timeout="300s" on-fail="fence"

primitive prmWEB ocf:heartbeat:apache ¥
  op start interval="0s" timeout="300s" on-fail="restart" ¥
  op monitor interval="10s" timeout="60s" on-fail="restart" ¥
  op stop interval="0s" timeout="300s" on-fail="fence"
```

# 付録：リソース定義ファイル（2）

```
primitive prmPing ocf:pacemaker:ping ¥
  params ¥
    name="default_ping_set" ¥
    host_list="192.168.1.1" ¥
    multiplier="100" ¥
    attempts="2" ¥
    timeout="2" ¥
    debug="true" ¥
  op start interval="0s" timeout="60s" on-fail="restart" ¥
  op monitor interval="10s" timeout="60s" on-fail="restart" ¥
  op stop interval="0s" timeout="60s" on-fail="ignore"

primitive prmDiskd1 ocf:pacemaker:diskd ¥
  params ¥
    name="diskcheck_status" ¥
    device="/dev/vdb" ¥
    options="-e -t 70" ¥
    interval="10" ¥
    dampen="2" ¥
  op start interval="0s" timeout="60s" on-fail="restart" ¥
  op monitor interval="10s" timeout="60s" on-fail="restart" ¥
  op stop interval="0s" timeout="60s" on-fail="ignore"

primitive prmDiskd2 ocf:pacemaker:diskd ¥
  params ¥
    name="diskcheck_status_internal" ¥
    device="/dev/vda" ¥
    options="-e" ¥
    interval="10" ¥
    dampen="2" ¥
  op start interval="0s" timeout="60s" on-fail="restart" ¥
  op monitor interval="10s" timeout="60s" on-fail="restart" ¥
  op stop interval="0s" timeout="60s" on-fail="ignore"
```

```
primitive prmStonith1 stonith:external/libvirt ¥
  params ¥
    pcmk_reboot_timeout="60s" ¥
    hostlist="vm13:server01" ¥
    hypervisor_uri="qemu+ssh://192.168.122.1/system" ¥
  op start interval="0s" timeout="60s" on-fail="restart" ¥
  op monitor interval="3600s" timeout="60s" on-fail="restart" ¥
  op stop interval="0s" timeout="60s" on-fail="ignore"

primitive prmStonith2 stonith:external/libvirt ¥
  params ¥
    pcmk_reboot_timeout="60s" ¥
    hostlist="vm14:server02" ¥
    hypervisor_uri="qemu+ssh://192.168.122.1/system" ¥
  op start interval="0s" timeout="60s" on-fail="restart" ¥
  op monitor interval="3600s" timeout="60s" on-fail="restart" ¥
  op stop interval="0s" timeout="60s" on-fail="ignore"

### Resource Location ###
location rsc_location-grpTrac-1 grpTrac ¥
  rule 200: #uname eq server01 ¥
  rule 100: #uname eq server02 ¥
  rule -INFINITY: not_defined default_ping_set or default_ping_set lt 100 ¥
  rule -INFINITY: not_defined diskcheck_status or diskcheck_status eq ERROR ¥
  rule -INFINITY: not_defined diskcheck_status_internal or diskcheck_status_internal eq ERROR
location rsc_location-prmStonith1 prmStonith1 ¥
  rule -INFINITY: #uname eq server01
location rsc_location-prmStonith2 prmStonith2 ¥
  rule -INFINITY: #uname eq server02

### Resource Colocation ###
colocation rsc_colocation-grpTrac-clnPing-1 INFINITY: grpTrac clnPing
colocation rsc_colocation-grpTrac-clnDiskd1-2 INFINITY: grpTrac clnDiskd1
colocation rsc_colocation-grpTrac-clnDiskd2-3 INFINITY: grpTrac clnDiskd2

### Resource Order ###
order rsc_order-clnPing-grpTrac-1 0: clnPing grpTrac symmetrical=false
order rsc_order-clnDiskd1-grpTrac-2 0: clnDiskd1 grpTrac symmetrical=false
order rsc_order-clnDiskd2-grpTrac-3 0: clnDiskd2 grpTrac symmetrical=false
```