PacemakerとHAProxyではじめる高可用ロードバランサ入門

2021/10/22 OSC2021 Online/Fall

Linux-HA Japan プロジェクト

高藤　巧

自己紹介

高藤　巧(たかふじ　たくみ)

セミナー講師としてOSCに参加するのは初めてです。

普段はNTT OSSセンタという所でOSSロードバランサをメインに扱っています。

お手柔らかにお願いします。

目次

◆Pacemakerとロードバランサの概要

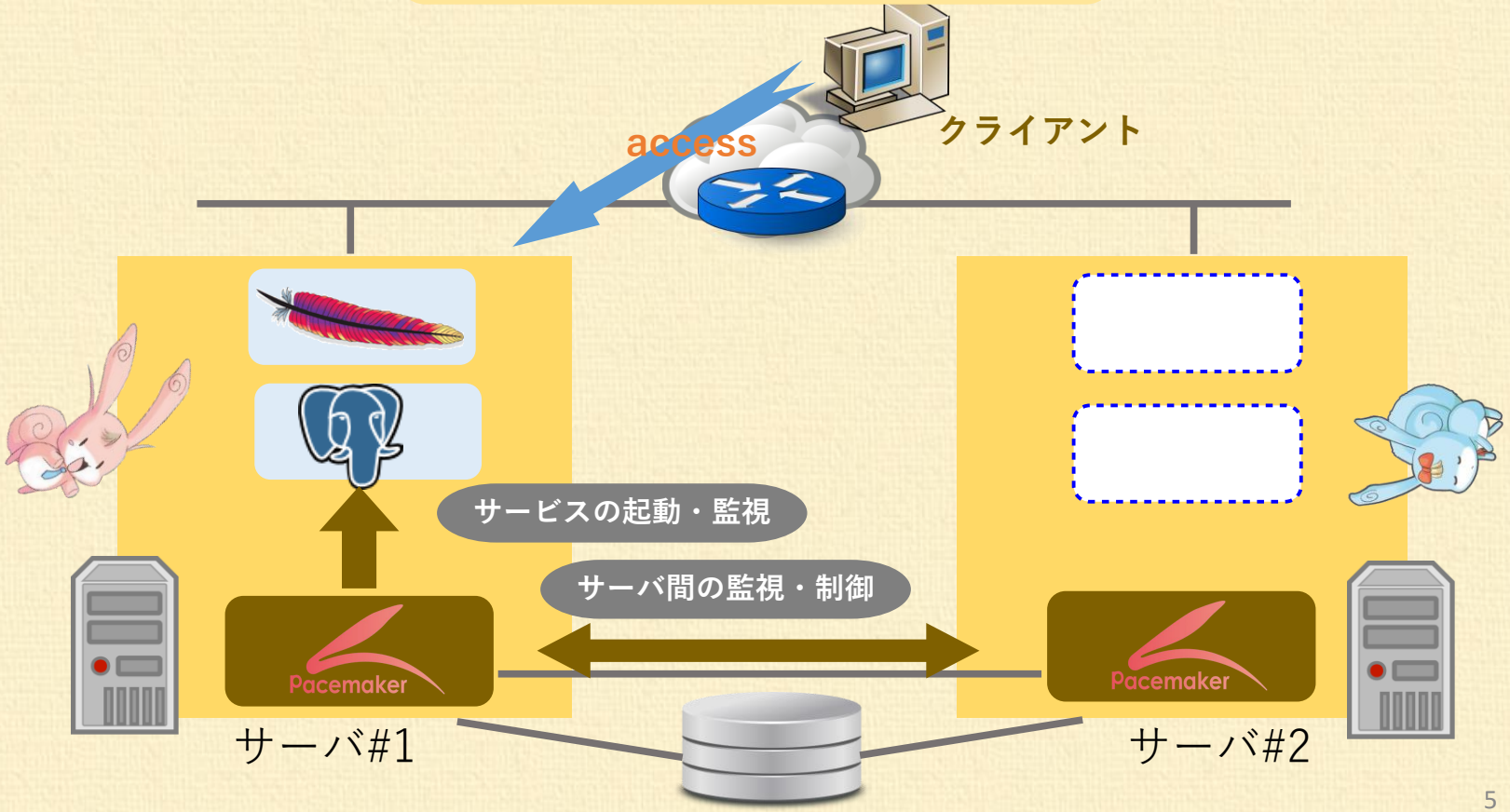
◆高可用ロードバランサ構築入門

◆おわりに

目次

Pacemakerの概要 [1/2]

サーバ・アプリの故障監視

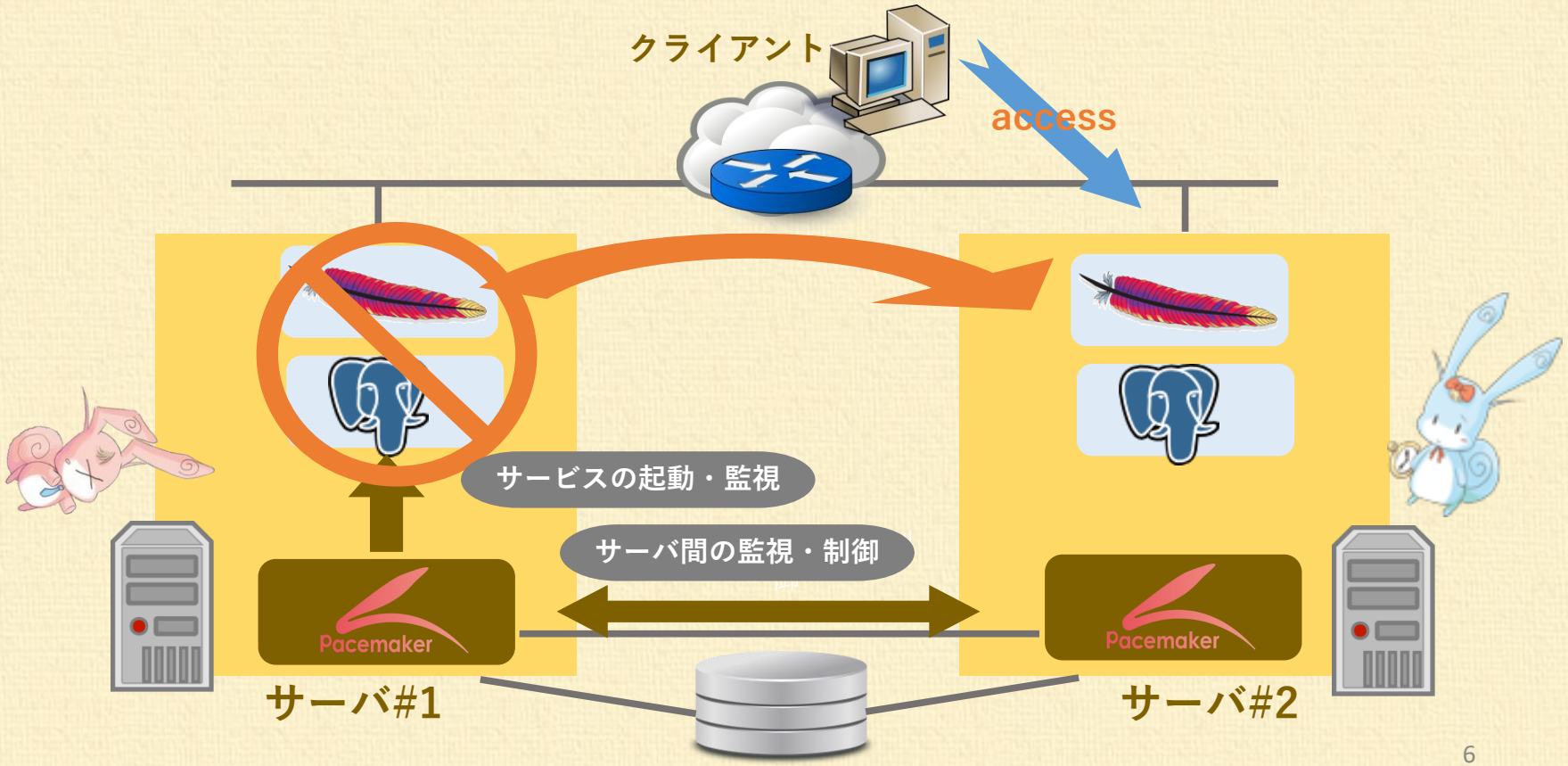


Pacemakerの概要 [2/2]

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



※ スプリットブレイン、F/Oにおけるリソース停止失敗時に対向ノードを強制電源断する機能

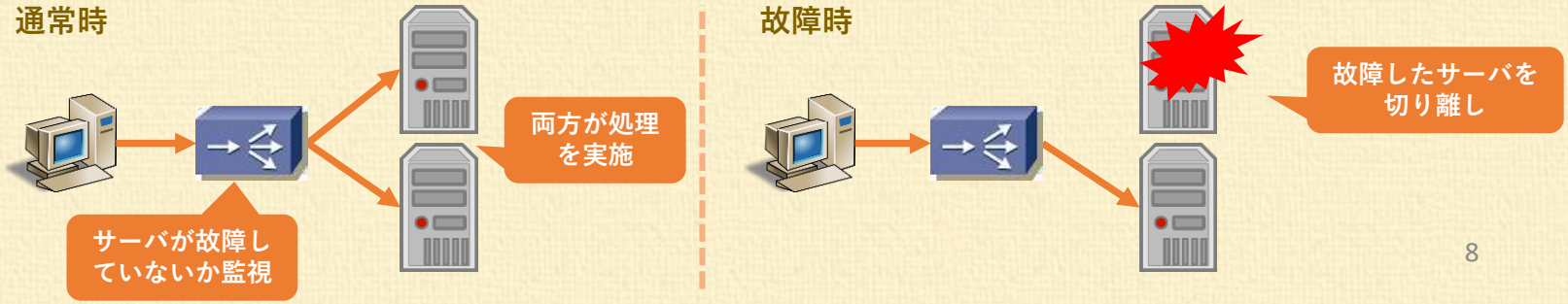
もっと知りたい方はこちらまで

http://linux-ha.osdn.jp/

高可用クラスタとロードバランサ(概要)

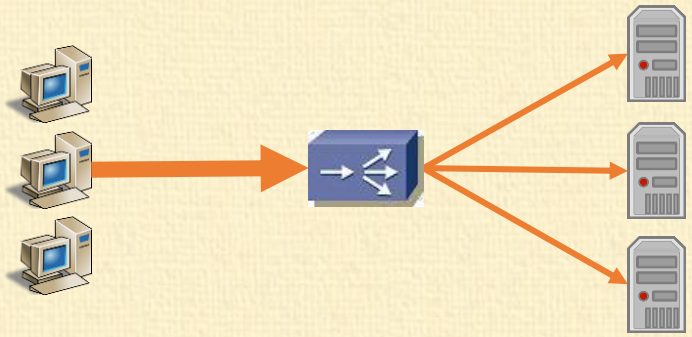
- 高可用クラスタ
  - Active/Standby
  - DBサーバなど同時に処理されたら困るものはこちら

- ロードバランサ
  - 両系Active
  - Webサーバなど同時に処理されても困らないものはこちら

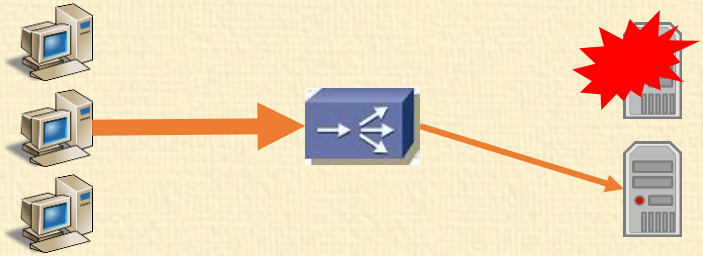


ロードバランサの主な機能

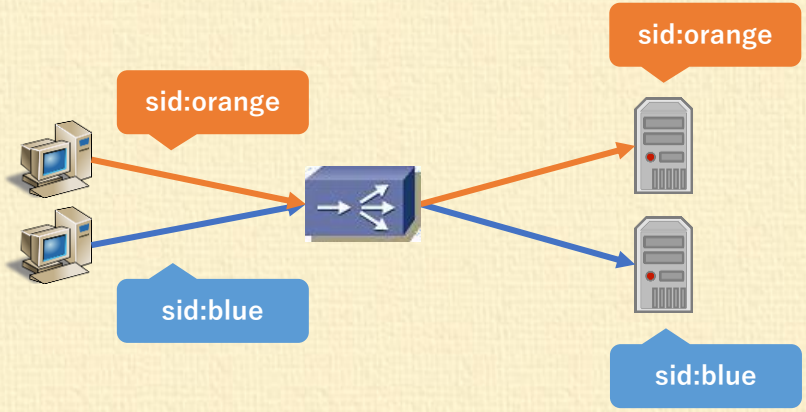
負荷分散（ロードバランス）



冗長化

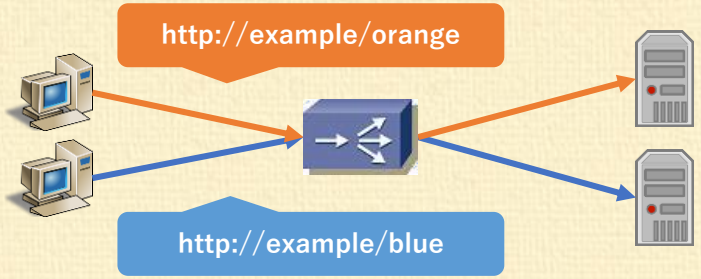


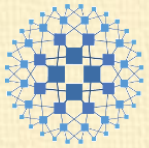
パーシステンス（セッション維持）



各種情報に基づく振り分け

（URL, パス, ドメイン, 接続元IP, SNI, 接続数, etc …）





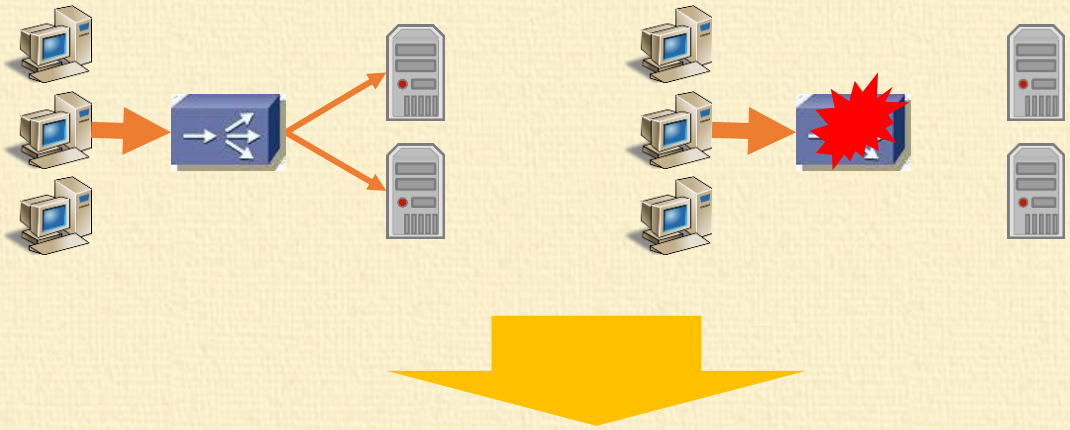
HAPROXY 概要

- 高機能なOSSロードバランサ
  - L4 / L7 ロードバランサとして利用可能
  - パーシステンス, SSL終端などロードバランサに必要な機能を具備
  - L4～L7の情報に基づく柔軟な振り分け、通信拒否、書き換え
  - etc …
- コミュニティ版は無料！勉強にも最適！！
  - ロードバランサのイメージ※
    - 高い
    - 高レイヤのネットワーク制御は機能が多くて難しそう…
    - 高い
    - 検証環境を用意するのが大変…
    - 高い
- コミュニティ版の情報はこちら (https://www.haproxy.org/)

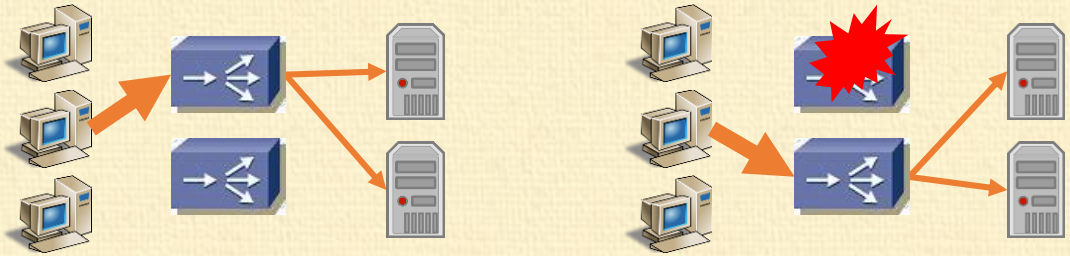
※発表者の個人的な見解です

高可用ロードバランサ

- ロードバランサが単一故障点になるのは可用性の観点から**NO**



- ロードバランサ自身も冗長化にすれば**OK!!**

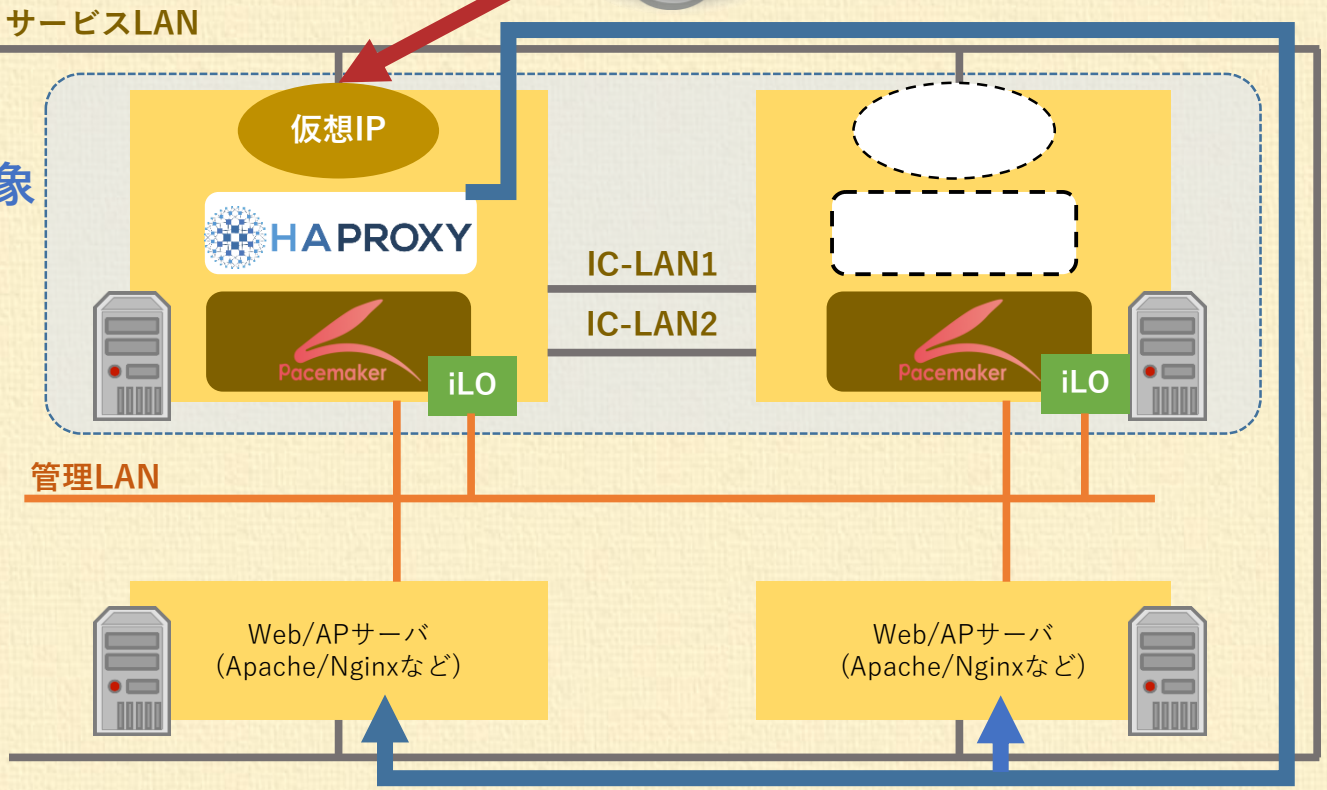


本日のテーマ

Q: 冗長化したロードバランサがどうしても必要なのですが、多くの課題（金銭面など）があり用意できません。どうしたらいいですか？

A: PacemakerとHAProxyで作りましょう!!

今回の対象



目次

前提

- OSインストールやネットワーク設定などのOS関連の設定は両系とも完了済みであるとし、以下を紹介します。
  1. HAProxy構築 (インストール、設定)
  2. Pacemaker構築 (インストール、設定)
  3. 動作確認

- Pacemakerのインストールや設定については下記のセミナー資料もご参照ください。

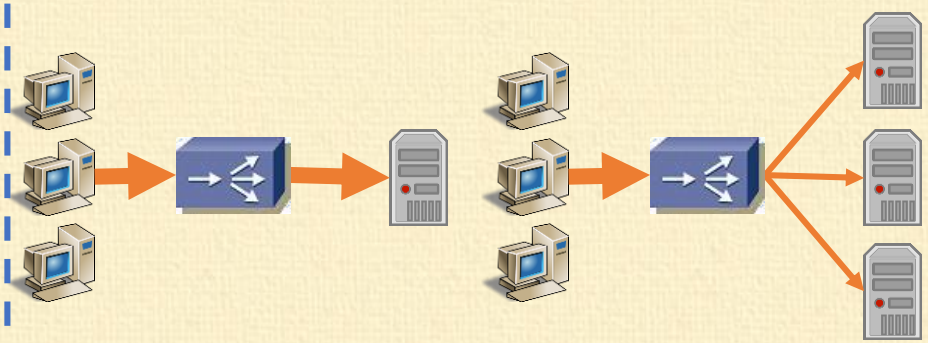
試して覚えるPacemaker-2.0入門『構築・リソース設定』
http://linux-ha.osdn.jp/wp/archives/4970

- ロードバランサの振り分け先のWeb/APサーバ (Apache, Nginx など) は用意済みであるものとします。

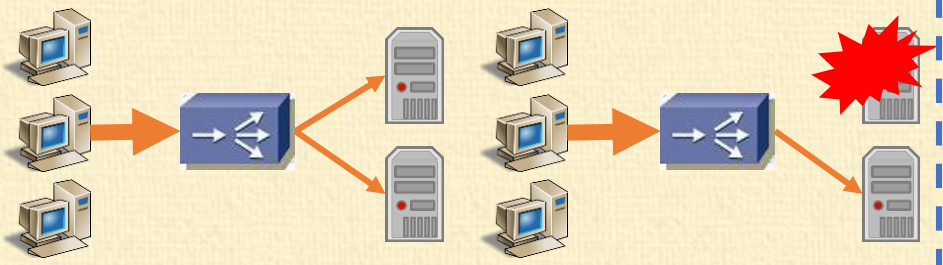
ロードバランサの主な機能（再掲）

今回の対象

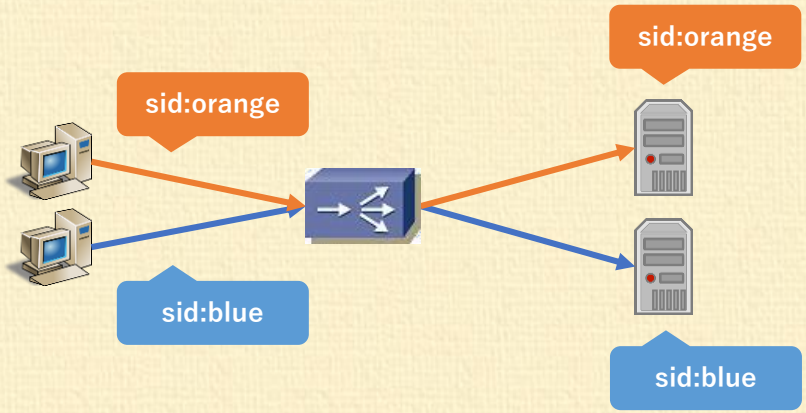
負荷分散（ロードバランス）



冗長化

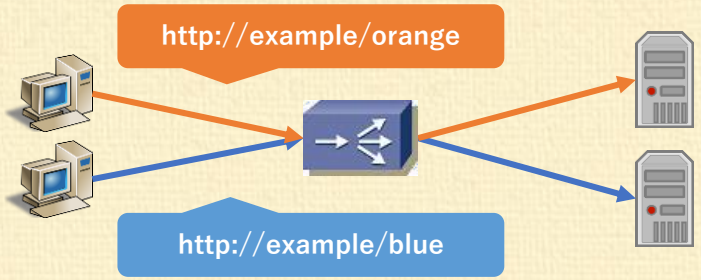


パーシステンス（セッション維持）



各種情報に基づく振り分け

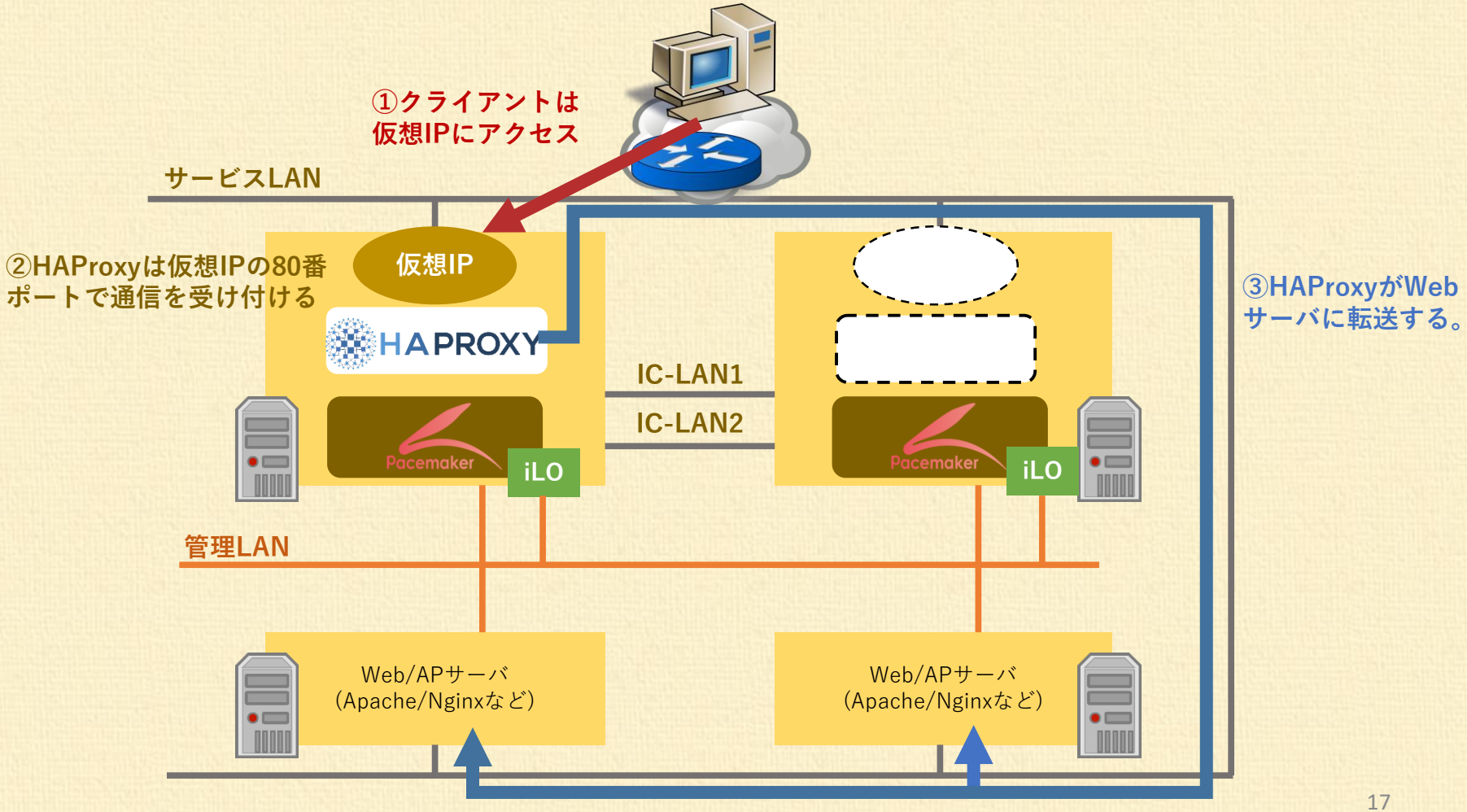
（URL, パス, ドメイン, 接続元IP, SNI, 接続数, etc …）



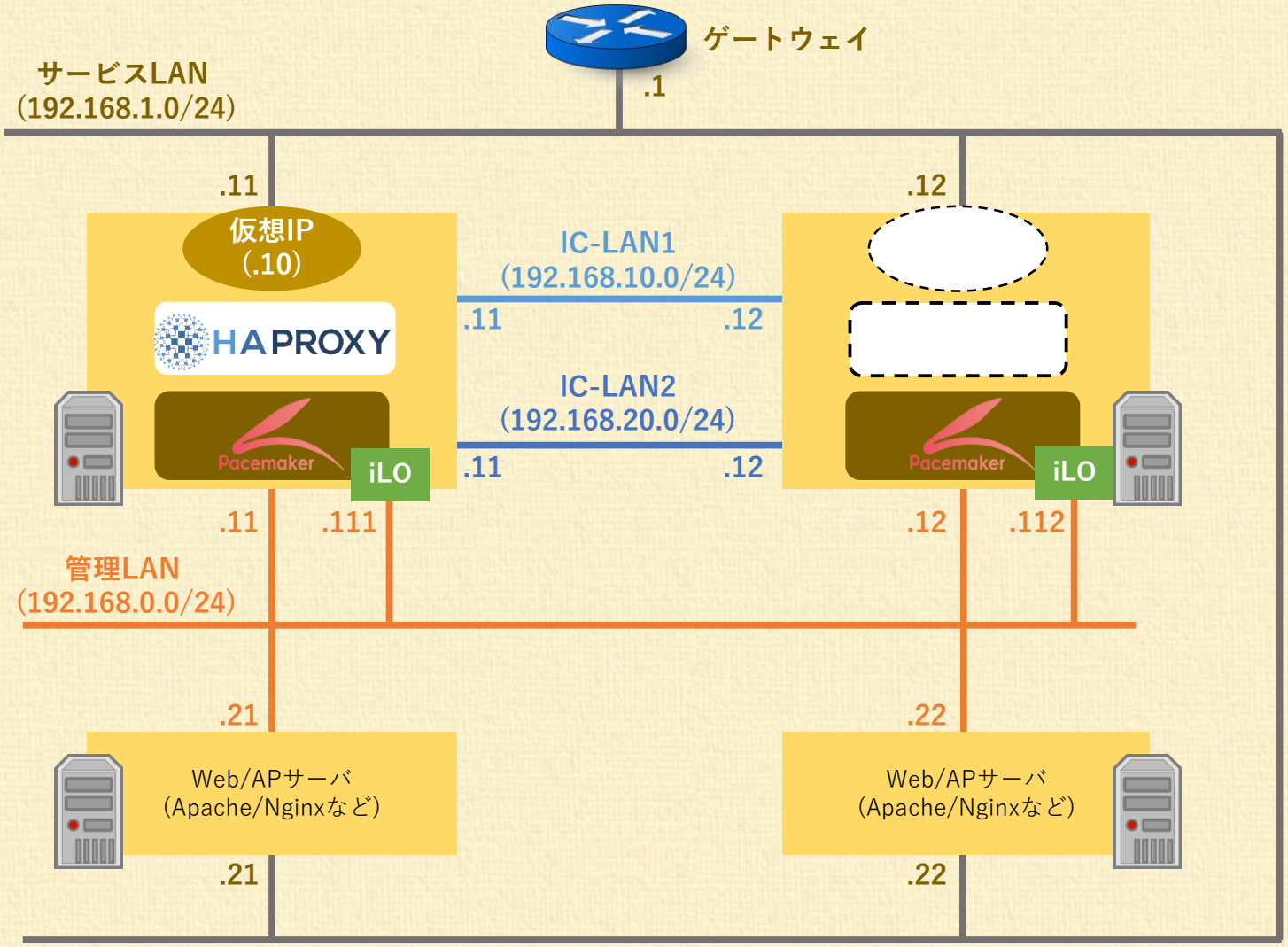
ソフトウェア構成

製品	パッケージ
Cent OS	CentOS-8.4.2105-x86_64
Pacemaker	CentOS HighAvailability 同梱版 (pacemaker-2.0.5-9.el8.x86_64)
pm_extra_tools	pm_extra_tools-1.3-1.el8.noarch
HAProxy	haproxy-1.8.27-2.el8.x86_64

ネットワーク構成 [1/2]

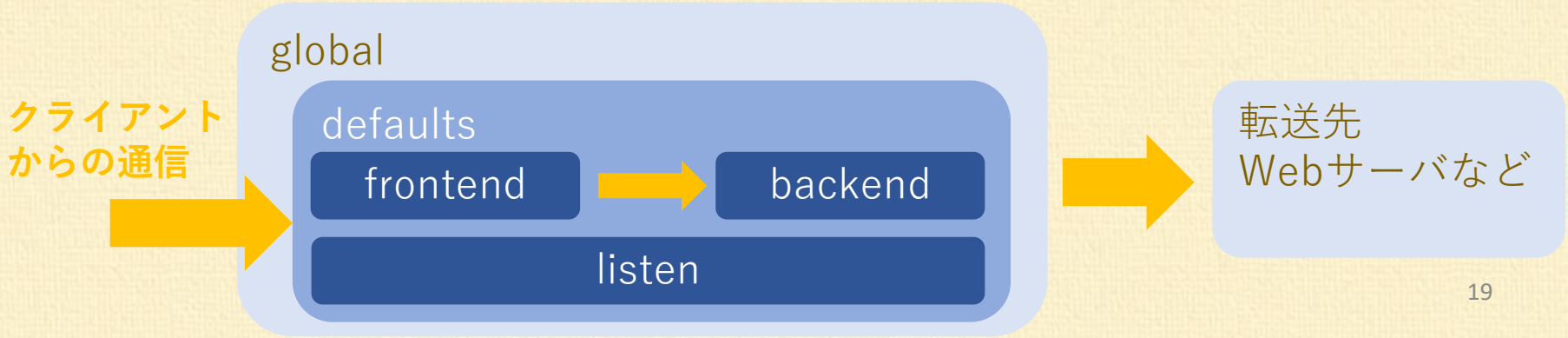

①クライアントは
仮想IPにアクセス
サービスLAN
②HAProxyは仮想IPの80番
ポートで通信を受け付ける
仮想IP
HAPROXY
IC-LAN1
IC-LAN2
Pacemaker
iLO
Pacemaker
iLO
③HAProxyがWeb
サーバに転送する。
管理LAN
Web/APサーバ
(Apache/Nginxなど)
Web/APサーバ
(Apache/Nginxなど)

ネットワーク構成 [2/2]


ゲートウェイ
.1
サービスLAN
(192.168.1.0/24)
.11
.12
仮想IP
(.10)
HAPROXY
IC-LAN1
(192.168.10.0/24)
.11
.12
IC-LAN2
(192.168.20.0/24)
.11
.12
Pacemaker
iLO
Pacemaker
iLO
.11
.111
.12
.112
管理LAN
(192.168.0.0/24)
.21
.22
Web/APサーバ
(Apache/Nginxなど)
Web/APサーバ
(Apache/Nginxなど)
.21
.22

HAProxy設定の前に

- HAProxyのコンフィグはいくつかのセクションで構成される
  - global:　HAProxy全体の動作に関わる設定
  - defaults:　frontend/backend/listen のデフォルト設定
  - frontend:　LBとして通信を受け付ける部分に関する設定
  - backend:　LBの転送先に関する設定
  - listen:　frontend + backend を同時に記載
  - その他は割愛します
    - (気になる人はコミュニティの設定マニュアルをチェック→ https://cbonte.github.io/haproxy-dconv/1.8/configuration.html)
- イメージ図



HAProxy構築手順 [1/3]

- HAProxyのインストール

```
# dnf install haproxy
```

- HAProxy設定変更

```
# mv /etc/haproxy/haproxy.cfg /etc/haproxy/backup_haproxy.cfg
# vi /etc/haproxy/haproxy.cfg

global
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     2010
    user        haproxy
    group       haproxy
    daemon
    stats socket /var/lib/haproxy/stats
    ssl-default-bind-ciphers PROFILE=SYSTEM
    ssl-default-server-ciphers PROFILE=SYSTEM
    userlist stats-auth
        group admin users admin
        user  admin insecure-password password
```

global の maxconn (最大同時接続数) は後述の listen/frontend の maxconn の合計値より大きい値を設定する。

管理画面アクセス用のユーザを作成する。

HAProxy構築手順 [2/3]

- HAProxy設定変更 (続き)

```
defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s

listen statspage
    bind 192.168.0.11:10080
    maxconn 10
    stats enable
    stats uri /statspage
    stats admin if TRUE
    acl AUTH http_auth(stats-auth)
    stats http-request auth unless AUTH
```

タイムアウト値はデフォルトコンフィグから変更していないが、実際は要件や振り分け先の設定に合わせて調整が必要になる。

maxconn は listen/fronend で個別に設定するため、defaultsから削除する。

管理画面設定
- 管理LANのIPアドレスをバインドする。
  (ポート番号は空いているポートを使用する)
- ユーザ認証を実施する。

HAProxy構築手順 [3/3]

- HAProxy設定変更 (続き)

```
frontend main
    bind 192.168.1.10:80
    maxconn 2000
    default_backend web

backend web
    balance roundrobin
    server web01 192.168.1.21:80 check maxconn 1000
    server web02 192.168.1.22:80 check maxconn 1000
```

Pacemaker で管理する仮想IPを設定する。

- 転送先のIPとポート番号を指定する。
- ヘルスチェック (check) を有効化する。
- maxconn は frontend に合わせて調整する。

Pacemaker構築手順
インストール～クラスタ作成 [1/2]

- Pacemakerのインストール

```
# dnf install pcs pacemaker fence-agents-all --enablerepo=ha
```

- pm_extra_tools(※)のインストール

```
# ls
pm_extra_tools-1.3-1.el8.noarch.rpm
# dnf install -y pm_extra_tools-1.3-1.el8.noarch.rpm
```

(※) pm_extra_tools はこちらからダウンロード
https://osdn.net/projects/linux-ha/releases/p16919

- pcsの起動と自動起動の有効化

```
# systemctl start pcsd.service
# systemctl enable pcsd.service
Created symlink /etc/systemd/system/multi-user.target.wants/pcsd.service →
 /usr/lib/systemd/system/pcsd.service.
```

Pacemaker構築手順
インストール～クラスタ作成 [2/2]

- hacluster ユーザのパスワード設定

```
# passwd hacluster
Changing password for user hacluster.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

- クラスタノードの認証設定 (片方のノードで実施)

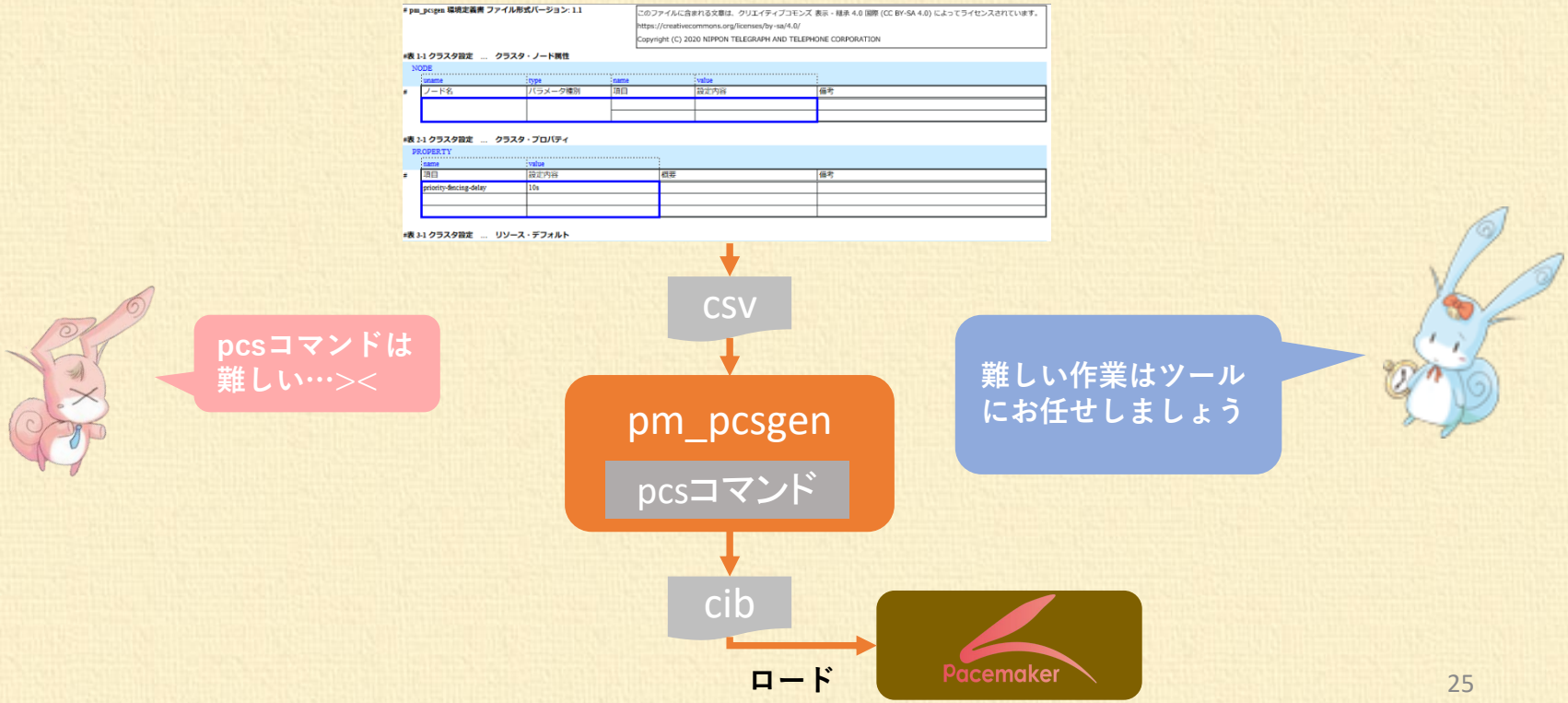
```
# pcs host auth node01 addr=192.168.0.11 node02 addr=192.168.0.12
Username: hacluster
Password:
node01: Authorized
node02: Authorized
```

- クラスタの作成 (片方のノードで実施)

```
# pcs cluster setup lbcluster node01 addr=192.168.10.11 addr=192.168.20.11 node02 addr=192.168.10.12
addr=192.168.20.12
```

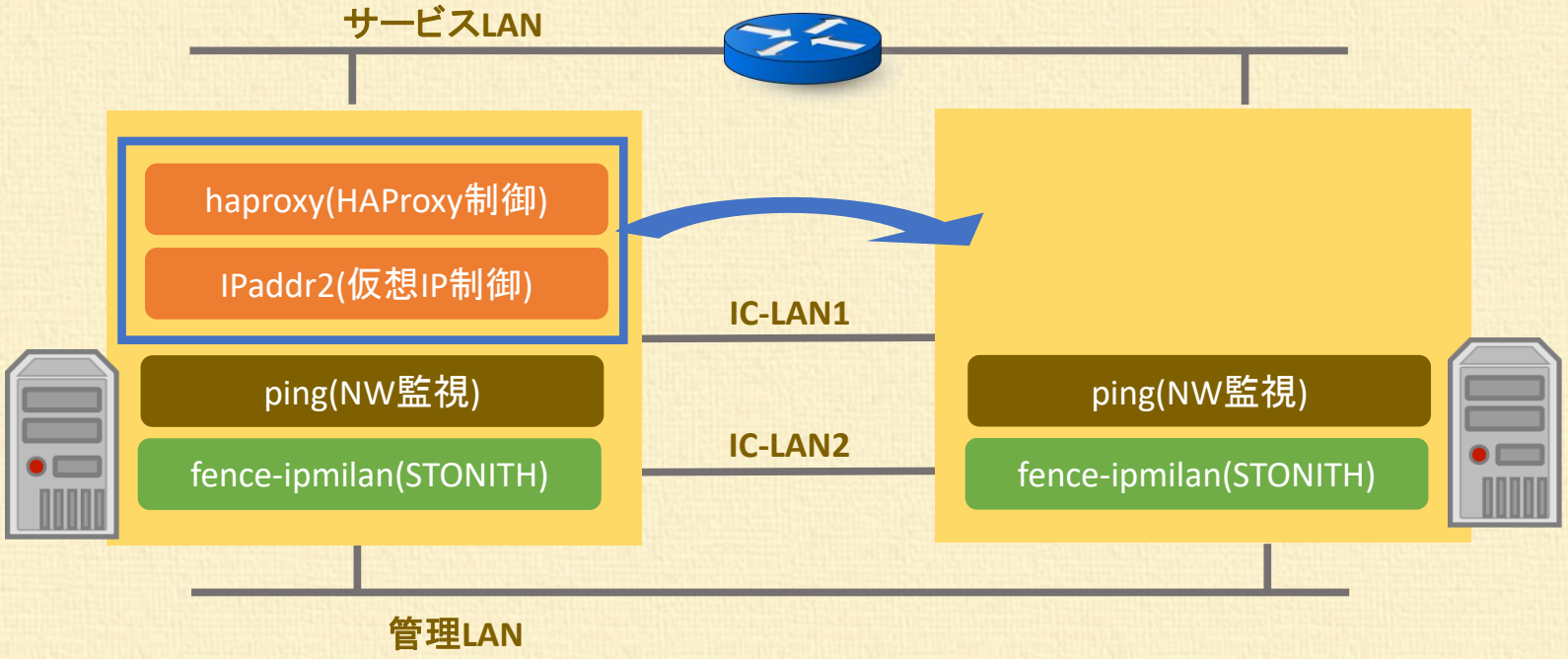
Pacemaker構築手順
リソース設定 [1/11]

- pm_extra_toolsから下記のファイルを取得
  /usr/share/pm_extra_tools/pm_pcsgen_sample.xlsx
- この後の作業イメージ



Pacemaker構築手順
リソース設定 [2/11]

- 今回のリソース構成
  - データを持たないのでリソースも少なめ



Pacemaker構築手順
リソース設定 [3/11]

- 表1～3

pm_pcsgen 環境定義書 ファイル形式バージョン: 1.1

#表 1-1 クラスタ設定　...　クラスタ・ノード属性

NODE

#	uname ノード名	type パラメータ種別	name 項目	value 設定内容	備考

#表 2-1 クラスタ設定　...　クラスタ・プロパティ

PROPERTY

#	name 項目	value 設定内容	概要	
	priority-fencing-delay	10s		

表1~3は基本的に変更不要

#表 3-1 クラスタ設定　...　リソース・デフォルト

RSC_DEFAULTS

#	name 項目	value 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	

#表 3-2 クラスタ設定　...　オペレーション・デフォルト

OP_DEFAULTS

#	name 項目	value 設定内容	概要	備考

Pacemaker構築手順
リソース設定 [4/11]

- 表4 (リソース構成)

#表 4-1 クラスタ設定　…　リソース構成

RESOURCES

#	resourceItem	resourceItem	resourceItem	id	概要	
	リソース構成要素			リソースID		
	Group			pgsql-group		
		Primitive		ipaddr		
		Primitive		haproxy		
	Clone			ping-clone		
		Primitive		ping		
	Stonith			fence1-ipmilan		
	Stonith			fence2-ipmilan		

HAProxyと仮想IPはグループ化

Pacemaker構築手順
リソース設定 [5/11]

- 表7 (仮想IPとHAProxy)

#表 7-1 クラスタ設定　…　Primitiveリソース

PRIMITIVE

P	id	class	provider	type	
#	リソースID	class	provider	type	概要
	ipaddr	ocf	heartbeat	IPaddr2	

A	type	name	value	
#	パラメータ種別	項目	設定内容	概要
	options	ip	192.168.1.10	
		nic	eno2	
		cidr_netmask	24	

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	備考
	start	60s		restart			
	monitor	60s	10s	restart			
	stop	60s		fence			

PRIMITIVE

P	id	class	provider	type	
#	リソースID	class	provider	type	概要
	haproxy	systemd		haproxy	

A	type	name	value	
#	パラメータ種別	項目	設定内容	概要
	options			

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	備考
	start	100s		restart			
	monitor	60s	10s	restart			
	stop	100s		fence			

HAProxyはsystemdリソースを使用

Pacemaker構築手順
リソース設定 [6/11]

- 表7 (ping監視)

	PRIMITIVE							
	P	id	class	provider	type			
#		リソースID	class	provider	type	概要		
		ping	ocf	pacemaker	ping			
	A	type	name	value				
#		パラメータ種別	項目	設定内容	概要			
		options	name	ping-status				
			host_list	192.168.1.1				
			attempts	2				
			timeout	2				
			debug	true				
	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	備考
		start	60s		restart			
		monitor	60s	10s	restart			
		stop	60s		fence			

Pacemaker構築手順
リソース設定 [7/11]

- 表8

#表 8-1 クラスタ設定　…　STONITHリソース

STONITH						
P	id		type			
#	STONITHリソースID		type			概要
	fence1-ipmilan		fence_ipmilan			
A	type	name		value		
#	パラメータ種別	項目		設定内容		概要
	options	pcmk_host_list		node01		
		ip		192.168.0.111		IPMIポートのIPアドレス
		username		pacemaker		
		password		ipmipass1		
		lanplus		1		
O	type	timeout		interval	on-fail	
#	オペレーション	タイムアウト値		監視間隔	障害時の動作	備考
	start	60s			restart	
	monitor	60s		3600s	restart	
	stop	60s			ignore	

STONITH						
P	id		type			
#	STONITHリソースID		type			概要
	fence2-ipmilan		fence_ipmilan			
A	type	name		value		
#	パラメータ種別	項目		設定内容		概要
	options	pcmk_host_list		node02		
		ip		192.168.0.112		IPMIポートのIPアドレス
		username		pacemaker		
		password		ipmipass1		
		lanplus		1		
O	type	timeout		interval	on-fail	
#	オペレーション	タイムアウト値		監視間隔	障害時の動作	備考
	start	60s			restart	
	monitor	60s		3600s	restart	
	stop	60s			ignore	

Pacemaker構築手順
リソース設定 [8/11]

- 表9

#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	備考 (※roleを設定したい場合は、表9-2 を使用してください)
	haproxy-group	prefers	node01	200	
			node02	100	
	fence1-ipmilan	avoids	node01		
	fence2-ipmilan	avoids	node02		

#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	備考
	haproxy-group	-INFINITY	or	ping-status	lt	1		
				ping-status	not_defined			

Pacemaker構築手順
リソース設定 [9/11]

- 表10～11

#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	備考
	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	
	haproxy-group	ping-clone	INFINITY			

#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	備考
	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(tru	
	ping-clone	haproxy-group				false	

Pacemaker構築手順
リソース設定 [10/11]

- Excel環境定義書をCSVに変換し、いずれか一方のノードへ格納

- pm_pcsgenでxmlファイルに変換 (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。
```

- クラスタを起動

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

- xmlファイルを反映

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

Pacemaker構築手順
リソース設定 [11/11]

- 完成

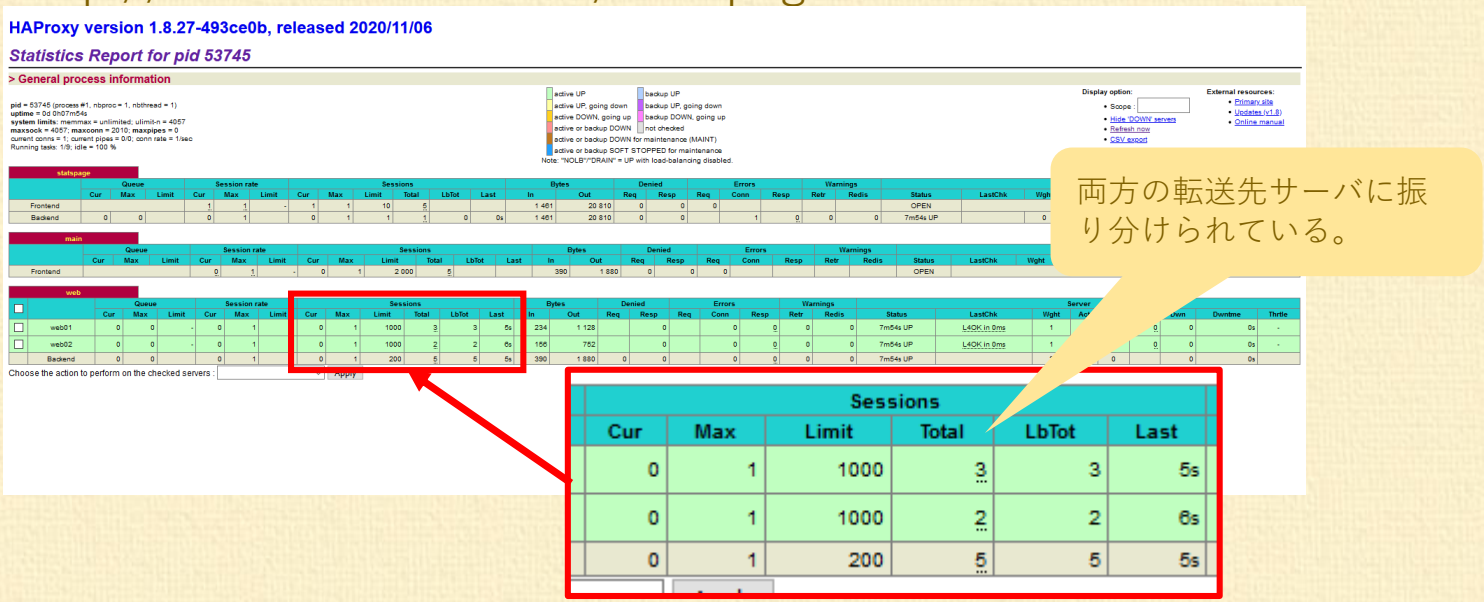
```
# pcs status --full
(省略)
Node List:
  * Online: [ node01 (1) node02 (2) ]

Full List of Resources:
  * Resource Group: haproxy-group:
    * ipaddr       (ocf::heartbeat:IPaddr2):        Started node01
    * haproxy      (systemd:haproxy):               Started node01
  * Clone Set: ping-clone [ping]:
    * ping         (ocf::pacemaker:ping):           Started node01
    * ping         (ocf::pacemaker:ping):           Started node02
  * fence1-ipmilan      (stonith:fence_ipmilan):      Started node02
  * fence2-ipmilan      (stonith:fence_ipmilan):      Started node01

Node Attributes:
  * Node: node01 (1):
    * ping-status                : 1
  * Node: node02 (2):
    * ping-status                : 1
(省略)
```

動作確認 [1/4]

- 仮想IPにHTTPでアクセス

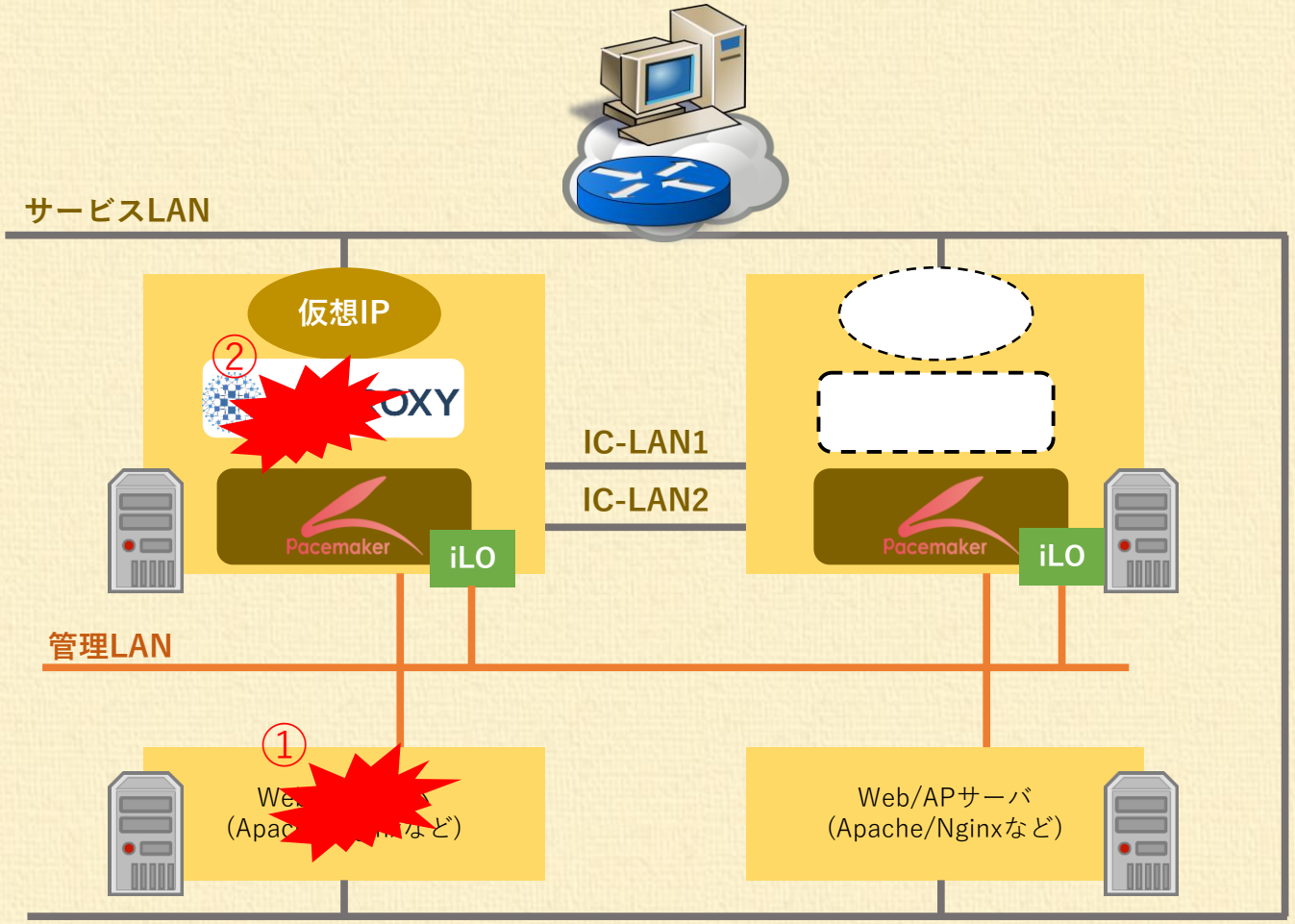
```
# curl http://192.168.1.10/ -I
HTTP/1.1 200
Content-Type: text/html;charset=UTF-8
Transfer-Encoding: chunked
Date: Mon, 04 Oct 2021 06:40:55 GMT
```

- HAProxyの管理画面をチェック
  - http://192.168.0.11:10080/statspage



動作確認 [2/4]

- 下図の①と②をそれぞれ故障させた場合の動作を確認する



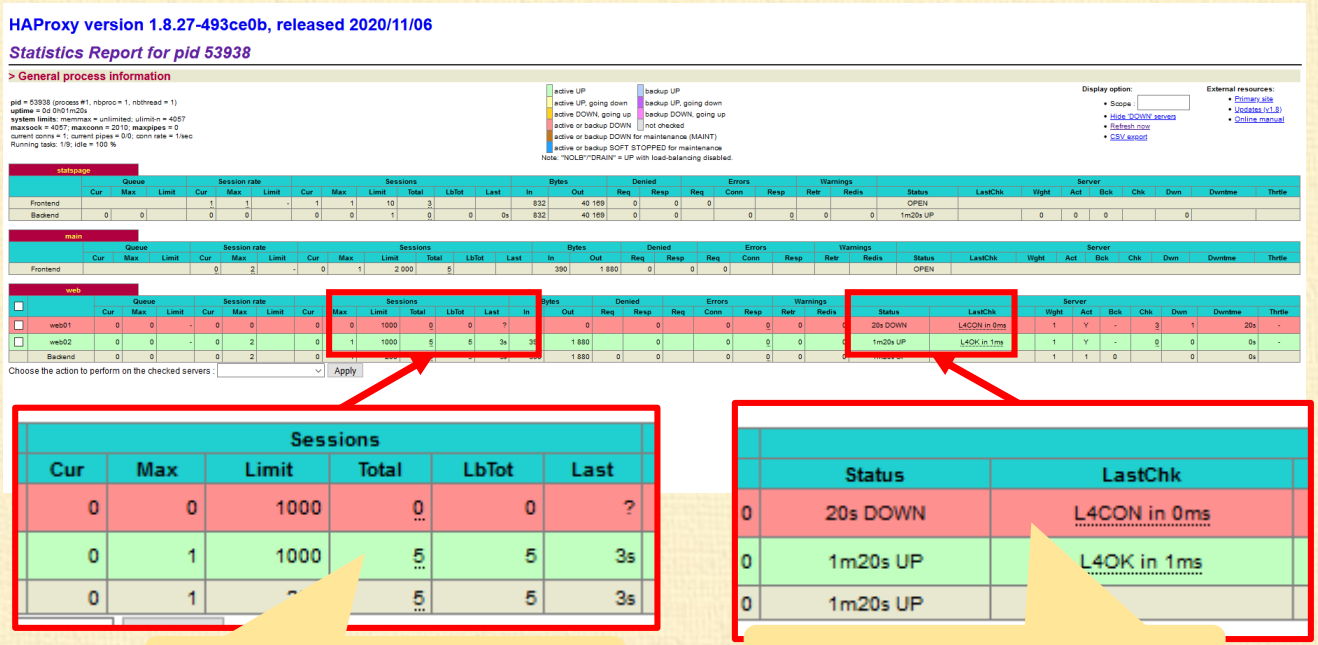
動作確認 [3/4]

①Web/APサーバ故障 (例: プロセス故障)

- Web/APサーバ上でプロセスをkillする (Apacheの例)

```
# killall -s 9 httpd
```

- HAProxyの管理画面をチェック



動作確認 [4/4]

②LB故障 (例: HAProxyプロセス故障)

- Active側のノードでHAProxyのプロセスをkillする

```
# killall -s 9 haproxy
(PacemakerでF/Oが完了するまで待機)
# pcs status --full
(省略)
Node List:
  * Online: [ node01 (1) node02 (2) ]

Full List of Resources:
  * Resource Group: haproxy-group:
    * ipaddr       (ocf::heartbeat:IPaddr2):       Started node02
    * haproxy      (systemd:haproxy):              Started node02
  * Clone Set: ping-clone [ping]:
    * ping         (ocf::pacemaker:ping):          Started node01
    * ping         (ocf::pacemaker:ping):          Started node02
  * fence1-ipmilan     (stonith:fence_ipmilan):      Started node02
  * fence2-ipmilan     (stonith:fence_ipmilan):      Started node01

(省略)

Migration Summary:
  * Node: node01 (1):
    * haproxy: migration-threshold=1 fail-count=1 last-failure='Mon Oct  4 16:02:01 2021'

Failed Resource Actions:
  * haproxy_monitor_10000 on node01 'not running' (7): call=34, status='complete', exitreason='', last-rc-
change='2021-10-04 16:02:01 +09:00', queued=0ms, exec=0ms
(省略)
```

フェイルオーバしてnode02側に

エラー原因が表示

目次

Pacemaker & HAProxy を利用した高可用ロードバランサ

- HAProxyの管理にはsystemdリソースを使用すればよい
  - どちらか片方でプロセスが起動する構成で基本的には問題なし※

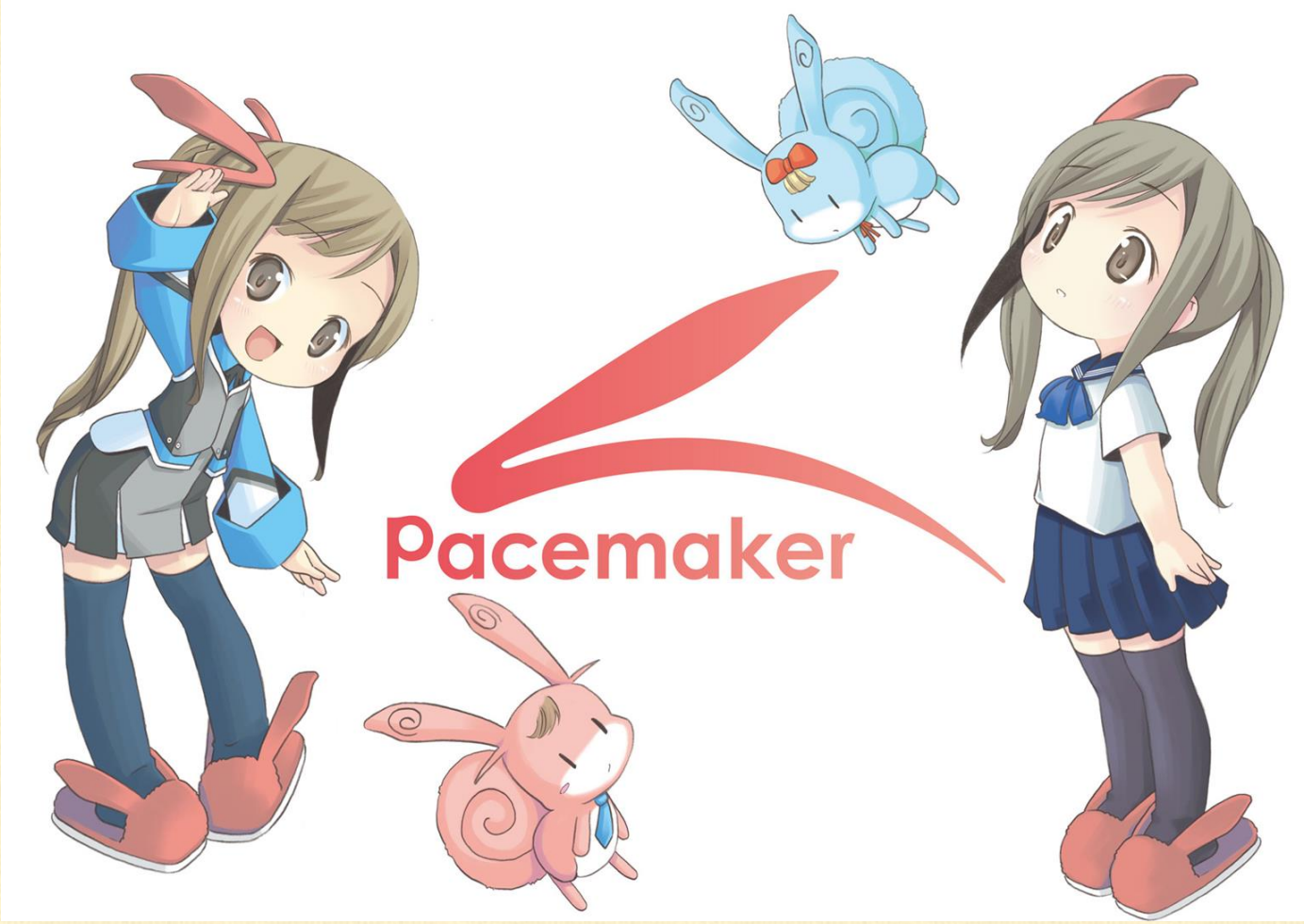
    ※セッション情報のレプリケーション等、一部機能を使用する場合を除く

- pm_extra_toolsのサンプルの大半の設定がそのまま流用可能なので構築もラクラク

それでも分からないことがあればMLへ

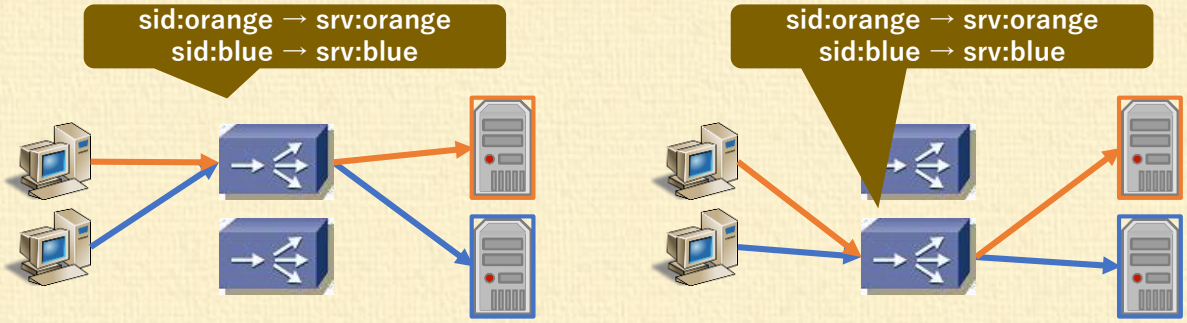
http://linux-ha.osdn.jp/wp/ml

今後もPacemakerを
よろしくお願いします



(参考) HAProxyの一部機能を使用する場合のクラスタ構成 [1/2]

- HAProxyにはパーシステンス (セッション維持) のテーブル情報をレプリケーションする機能があるが、この機能を利用するためには、両方のサーバ上でHAProxyが動作している必要があるため、単純にsystemdリソースを使用した場合には利用できない。



(参考) HAProxyの一部機能を使用する場合のクラスタ構成 [2/2]

- Keepalived (VRRP) などで仮想IPのみF/Oの対象にすることで条件は満たせる
  - Pacemakerと機能差分がかなり大きいため、どちらを採用するかはよくよく検討する必要あり