PostgreSQL 9 High Availability With Linux-HA

Agenda

- Introduction
- HA considerations
- PostgreSQL HA evolution
- Linux HA – components
- PostgreSQL streaming replication + Linux HA recipe
- DEMO!
- Summary

Who am I?

- Nikhil Sontakke
  - Responsible for the HA aspects of the Postgres-XL product
  - PostgreSQL community contributor
  - Co-organizer Indian PUG Meetup
  - Stints earlier at EnterpriseDB, StormDB, Translattice Inc.

HA - Definition

Postgres-XL

- What is High Availability (HA):
  - HA is a “concept”
  - A percentage of time that a given system is providing service since it has been deployed
  - For example: A system is 99% available if the downtime is 4 days in a year
  - Everyone craves for the five 9s (downtime of less than 5 minutes in a year – 99.999%)
  - HA is NOT designed for high performance
  - HA is NOT designed for high throughput (aka load balancing)

HA – Why does it matter?

- Why do we bother with HA:
  - Downtime is expensive
  - You miss out on earnings due to the downtime
  - You bother because your boss might complain ;)
  - Users might not return!

HA – wish our sysadmin is like this ;)

xkcd.com/705

PostgreSQL – HA evolution

- Log Shipping and Point In Time Recovery
  - PostgreSQL 8.1
  - Base backup of the database
  - Write Ahead Logs (WAL) sent to the standby

- Warm Standby
  - PostgreSQL 8.2
  - Continuously apply WAL on the standby

PostgreSQL – HA evolution (contd...)

- HA using Logical Replication
  - Slony (PG 7.3 onwards), Londiste, Bucardo
  - Trigger based replication systems
- HA using statement based replication
  - Pgpool-II (PG 6.4 onwards)
  - Intercept SQL queries and send to multiple servers

PostgreSQL – HA evolution (contd...)

- HA using Shared Storage
  - Sharing disk array between two servers
  - SAN environment needed (very expensive)

- HA using Block-Device replication
  - All changes to a filesystem residing on a block device are replicated to a block device on another system
  - DRBD pretty popular

PostgreSQL – HA latest...

- HA using Streaming Replication
  - Standby can be a HOT one to serve read only queries as well
  - Synchronous streaming available to have almost zero lag with the primary
- HA using Multi-master clusters
  - Postgres-XL coordinator and datanodes
- All solutions mentioned need an “external” HA infrastructure to manage failover

PostgreSQL – HA not in-built

- HA not in-built/in-core in PostgreSQL
- PostgreSQL provides the means, mechanisms and building blocks to get a HA system in place
- External monitoring and cluster management tools needed to come up with a “working” HA solution

PostgreSQL – Streaming Replication Scenario

Application

IP 1

SRHS

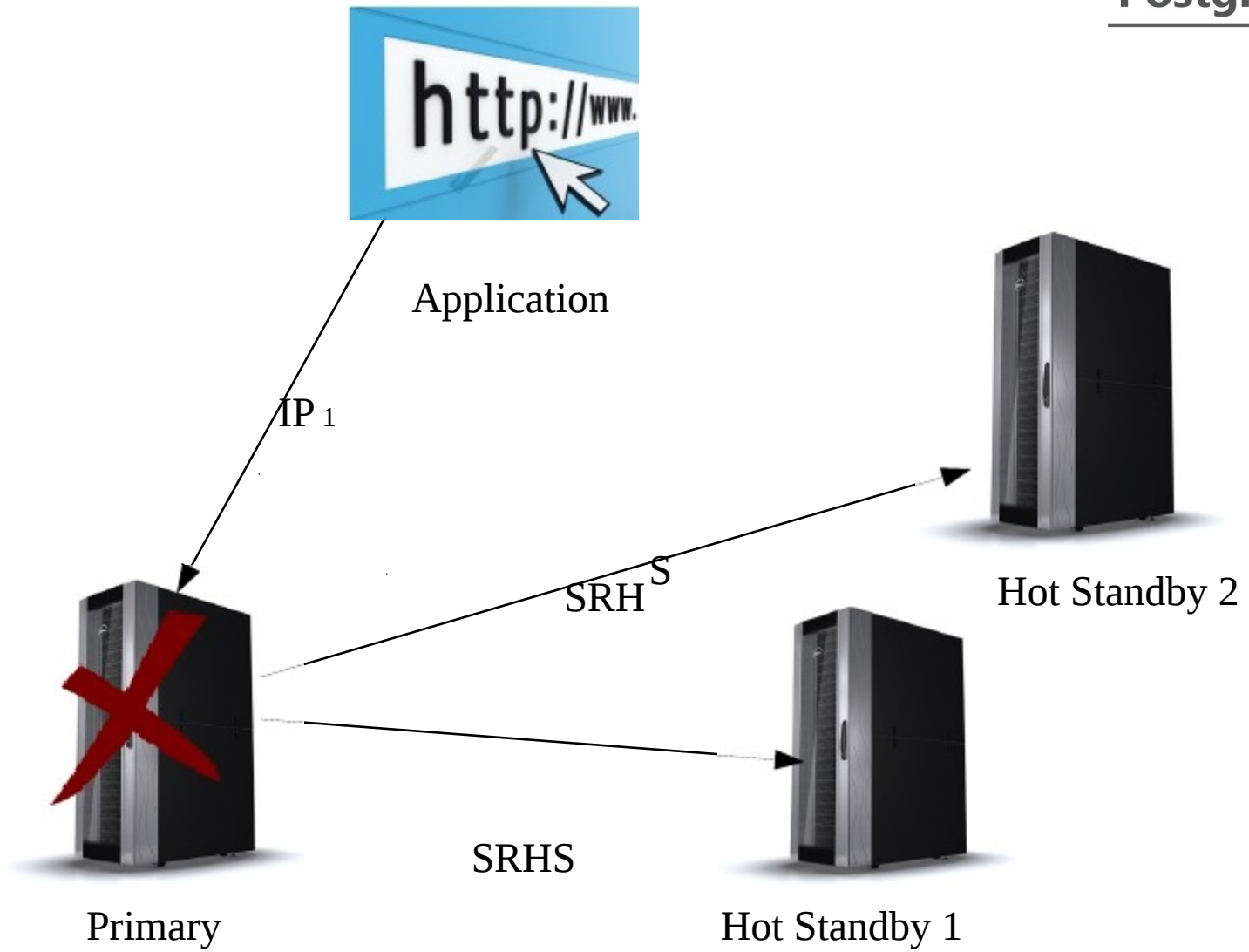
SRHS

Hot Standby 2

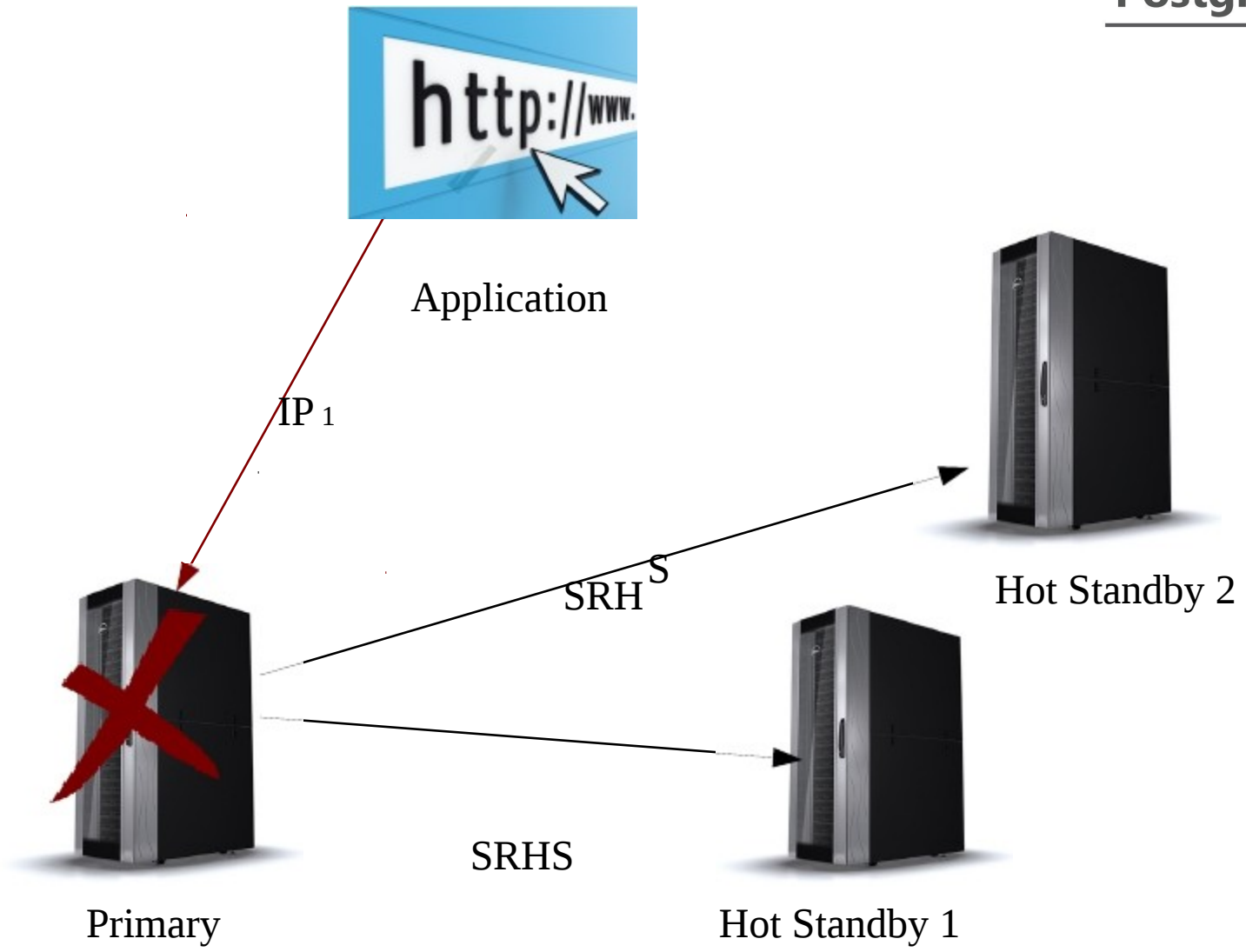
Primary

Hot Standby 1

PostgreSQL – Streaming Replication Scenario



PostgreSQL – Streaming Replication Scenario



PostgreSQL – Streaming Replication Scenario

Application

IP 1

Hot Standby 2

Primary

Promote Primary

PostgreSQL – Streaming Replication Scenario

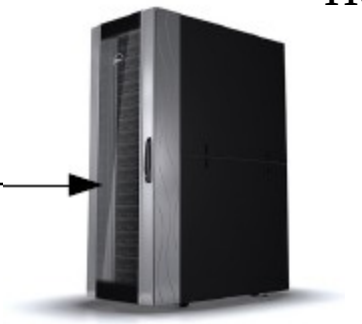
Application

IP 1

Hot Standby 2

Move IP1

Primary



Promote Primary

PostgreSQL – Streaming Replication Scenario

Application

IP 1

Hot Standby 2

Primary

Promote Primary

PostgreSQL – Streaming Replication Scenario – Bonus!

Application

IP 1

HS

SR

New Hot Standby 1

Primary

New Primary

PostgreSQL SR – HA requirements

- The Application should be able to connect to the database on a fixed IP address
- There should be a monitor running on the Primary and Standby nodes checking for running PG processes
- The monitor should first try to re-start PG if not running on the nodes configurable by a failure count
- In case if the node running the primary goes down for whatever reason exactly one of the Standby nodes should be promoted to Primary

PostgreSQL SR – HA requirements (contd)

- The IP address should move to the new node only after it has been promoted to be the new master
- It will be good to have the surviving standby connect to the new master and re-start the replication process
- Obviously all of the above should be done “automatically” without manual intervention via the clustering infrastructure :)

Introducing Linux-HA!

Postgres-XL

- The Linux-HA project is a high-availability clustering solution for Linux, FreeBSD, Solaris, etc.
- It has been around since quite a while (1999) and is increasingly gaining traction in Linux environments
- Suse Linux Enterprise Server (SLES) uses it as default clustering layer. RedHat also warming up to it in recent releases. Rpms available for Fedora, RHEL, Ubuntu etal

Linux-HA – Latest Version Components

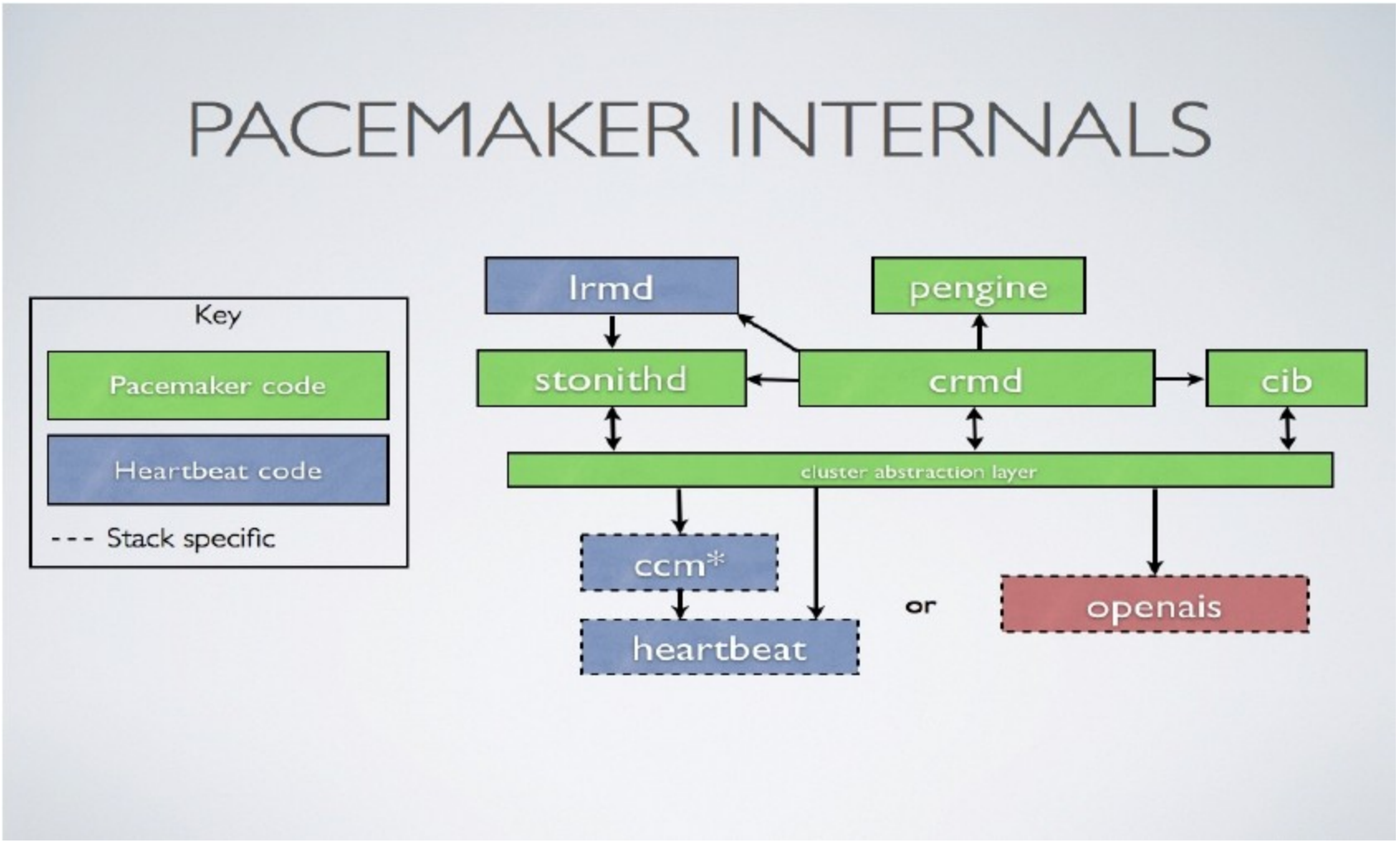
Postgres-XL

- Messaging Layer via Heartbeat/Corosync:
  - Node membership and notifications of nodes joining/leaving
  - Messaging between the nodes
  - A quorum system
- Cluster resource manager (crm) via Pacemaker:
  - Stores the configuration of the cluster
  - Uses the messaging layer to achieve maximum availability of your resources
  - Extensible: Anything that can be scripted can be managed by Pacemaker

Linux-HA – Latest Version Components

- Cluster Glue
  - Stuff that is neither cluster messaging (Corosync) nor CRM (Pacemaker)
  - Local node resource manager to interface with resource agents
  - STONITH daemon to provide fencing
- Resource Agents
  - Agent to manage a cluster resource
  - Support operations like start, stop, monitor, promote, demote etc.
  - Readymade agents available to manage resources like Apache, PostgreSQL, drbd etc

Linux-HA – The BIG picture (Whoops!)



Linux-HA – PostgreSQL resource agent

- The latest PostgreSQL resource agent is available at:

  https://raw.github.com/ClusterLabs/resource-agents/master/heartbeat/pgsql

- It follows the OCF (Open Cluster Framework) specifications

- The latest version is a Master/Slave resource agent supporting streaming replication (added by Takatoshi Matsuo)

Linux-HA – Warning! Eyes will HURT ;) :E

https://i.chzbgr.com/maxW500/6591864832/hC8B27BD6/

Linux-HA – Planning

- Create data directory on one node
- Setup the postgresql.conf, pg_hba.conf configuration files for replication
  - wal_level = hot_standby
  - max_wal_senders, wal_keep_segments
  - hot_standby = on, etc..
- Do a basebackup onto the other node
- No need to create recovery.conf file for the Standby. The RA creates it itself
- Check http://clusterlabs.org/wiki/PgSQL_Replicated_Cluster for inspiration

Linux-HA – Resource definitions

- The Linux HA configuration can be specified using the crm cli
  - crm configure edit (as root)
- Define a master public IP resource to which applications will connect to:

```
primitive vip-master ocf:heartbeat:IPaddr2 \
    params ip="192.168.168.108" nic="eth0"
    cidr_netmask="24" \
    op start interval="0s" timeout="60s" on-fail="stop" \
    op monitor interval="10s" timeout="60s" on-fail="restart" \
    op stop interval="0s" timeout="60s" on-fail="block"
```

Linux-HA – Resource definitions (contd...)

- Define a replication IP resource to which slaves will connect to:

```
primitive vip-rep ocf:heartbeat:IPaddr2 \
    params ip="192.168.168.109" nic="eth0"
    cidr_netmask="24" \
    op start interval="0s" timeout="60s" on-fail="stop" \
    op monitor interval="10s" timeout="60s" on-fail="restart" \
    op stop interval="0s" timeout="60s" on-fail="block"
```

- You can create an additional IP resource to allow reads to be queried from Standby nodes as well

Linux-HA – Resource definitions (contd...)

- The IP used for replication will shift along with the master IP whenever a standby is promoted.
- This allows other existing standbys to re-connect on this replication IP to the new Master.
- We use a “group” to keep them together:

```
group master-group vip-master vip-rep
```

Linux-HA – Resource definitions (contd...)

Postgres-XL

- Define the resource to control the PostgreSQL servers on the node:

```
primitive pgsql ocf:heartbeat:pgsql \
    params repuser="stormdb" pgdba="stormdb" pgport="5472"
    pgctl="/opt/PostgreSQL/bin/pg_ctl" psql="/opt/PostgreSQL/bin/psql"
    pgdata="/data/PostgreSQL/data/" start_opt="-p 5472"
    rep_mode="sync" node_list="stormtest1 stormtest3"
    master_ip="192.168.168.109" stop_escalate="0" \
    op start interval="0s" timeout="60s" on-fail="restart" \
    op monitor interval="7s" timeout="60s" on-fail="restart" \
    op monitor interval="2s" role="Master" timeout="60s" on-
    fail="restart" \
    op promote interval="0s" timeout="60s" on-fail="restart" \
    op demote interval="0s" timeout="60s" on-fail="stop" \
    op stop interval="0s" timeout="60s" on-fail="block" \
    op notify interval="0s" timeout="60s"
```

Linux-HA – Resource definitions (contd...)

- Create a master/slave configuration using the just specified pgsql resource

```
ms msPostgresql pgsql \
        meta \
        master-max="1" \
        master-node-max="1" \
        clone-max="2" \
        clone-node-max="1" \
        notify="true"
```

Linux-HA – Resource definitions (contd...)

- The “group” of the IP resources should always co-locate with the Master. Specify that

```
colocation rsc_colocation-1 \
    inf: master-group  msPostgresql:Master
```

- The IP addresses should be started ONLY after a MASTER has been chosen properly. We specify the same via resource ordering:

```
order rsc_order-1 0: msPostgresql:promote master-group:start symmetrical=false
```

Linux-HA – Resource definitions (contd...)

- Done!!
- Save the configuration by quitting the 'crm configure edit' window
- Check that there are no syntax or other errors while quitting
- Now take a deep breath, wipe off the sweat of your brow and invoke the command to start the cluster:

```
crm resource start msPostgresql
```

Linux-HA - Results

- Check if the HA cluster is up and running properly by issuing “crm_mon -1 -Afr”

```
 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):       Started stormtest1
     vip-rep    (ocf::heartbeat:IPaddr2):       Started stormtest1
 Master/Slave Set: msPostgresql [pgsql]
     Masters: [ stormtest1 ]
     Slaves: [ stormtest3 ]

Node Attributes:
* Node stormtest1:
    + master-pgsql:0                    : 1000
    + pgsql-data-status                 : LATEST
    + pgsql-master-baseline             : 0000000003001248
    + pgsql-status                      : PRI
* Node stormtest3:
    + master-pgsql:1                    : -INFINITY
    + pgsql-data-status                 : STREAMING|SYNC
    + pgsql-status                      : HS:sync
```

Linux-HA – Test!!

- Test, Test, TEST!
- Pull out network cables
- Power off nodes
- Use iptables to cause networking split brains

Linux-HA – Test Failover

Postgres-XL

- Stop the “corosync” service on one node. Check on the other node “crm_mon -Afr -1”:

```
Resource Group: master-group
    vip-master (ocf::heartbeat:IPaddr2):       Started stormtest3
    vip-rep    (ocf::heartbeat:IPaddr2):       Started stormtest3
Master/Slave Set: msPostgresql [pgsql]
    Masters: [ stormtest3 ]
    Stopped: [ pgsql:0 ]

Node Attributes:
* Node stormtest3:
    + master-pgsql:1                   : 1000
    + pgsql-data-status                : LATEST
    + pgsql-master-baseline            : 0000000030013B8
    + pgsql-status                     : PRI
```


It works!

PostgreSQL 9.x + Linux-HA == WIN!

- PostgreSQL 9.x provides the super cool streaming replication feature
- Linux HA has all the bells and whistles to provide a comprehensive HA infrastructure
- This gives you a full blown HA solution in place using purely awesome Open Source components
- Sure brings you closer to the 99.999% desired availability!

DEMO!!

- Two Virtual Machines: Ubuntu 14.04
- Corosync version : 2.3.3
- Pacemaker version : 1.1.10
- Resource Agents : 1:3.9.3
- PostgreSQL version: 9.3.4
- All components above installed using apt-get

Further reading

- http://www.linux-ha.org (Linux HA homepage)
- http://clusterlabs.org/ (for Pacemaker)
- http://corosync.github.io/corosync/ (Corosync)
- http://www.linux-ha.org/wiki/Resource_Agents (various supported resource agents)
- http://clusterlabs.org/wiki/PgSQL_Replicated_Cluster

Questions?

Questions?!

Thanks,
Nikhil Sontakke
nikkhils@gmail.com