Reference Architectures 2017

Red Hat CloudForms 4.1x - Implementing a Highly Available Virtual Management Database

Daniel Trieu Brett Thurber

Reference Architectures 2017 Red Hat CloudForms 4.1x - Implementing a Highly Available Virtual Management Database

Daniel Trieu

Brett Thurber
refarch-feedback@redhat.com

Legal Notice

Abstract

The purpose of this document is to provide guidelines and considerations for deploying Red Hat CloudForms 4.1x in a highly available configuration using pacemaker.

Table of Contents

COMMENTS AND FEEDBACK

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architecture. Although we review our papers internally, sometimes issues or typographical errors are encountered. Feedback allows us to not only improve the quality of the papers we produce, but allows the reader to provide their thoughts on potential improvements and topic expansion to the papers. Feedback on the papers can be provided by emailing refarch-feedback@redhat.com. Please refer to the title within the email.

CHAPTER 1. EXECUTIVE SUMMARY

Application high availability is a top subject for most enterprise IT environments. Downtime incurs unplanned expenses and creates frustration for both consumers and providers.

This reference architecture focuses on implementing a highly available configuration for Red Hat CloudForms.[1] by configuring the CloudForms database for a replicated setup with one master and one hot standby server.

A hot standby configuration provides the following:

- Software components are installed and available on both primary and secondary nodes. The software components on the secondary system are up but will not process data or requests
- Data is mirrored in near real time and both systems will have identical data. Data replication is typically done through the software's capabilities and generally provides a recovery time of a few seconds

For the reference environment the following technologies are used:

- NFS storage is provided by NetApp filers using SnapMirror technologies
- Virtual IP(VIP) along with DNS round-robin and load balancing provided by F5 Networks
- Clustering technologies provided by Red Hat

The targeted use case includes:

- Configuring a highly available Red Hat CloudForms implementation within a single datacenter, using Red Hat cluster services to provide a highly available database for each region

Note

Disclaimer: Third party products are not supported by Red Hat and are supported by the respective vendors. Custom scripts and other information shared are for informational/educational purposes. Refer to the **Red Hat Production Support Scope of Coverage**.[2] for additional information.

---

[1] https://www.redhat.com/en/technologies/management/cloudforms

[2] https://access.redhat.com/support/offerings/production/soc

CHAPTER 2. ENVIRONMENT

For the reference environment, the following diagram illustrates the high availability configuration deployed.

2.1. CLOUDFORMS HA CLUSTER

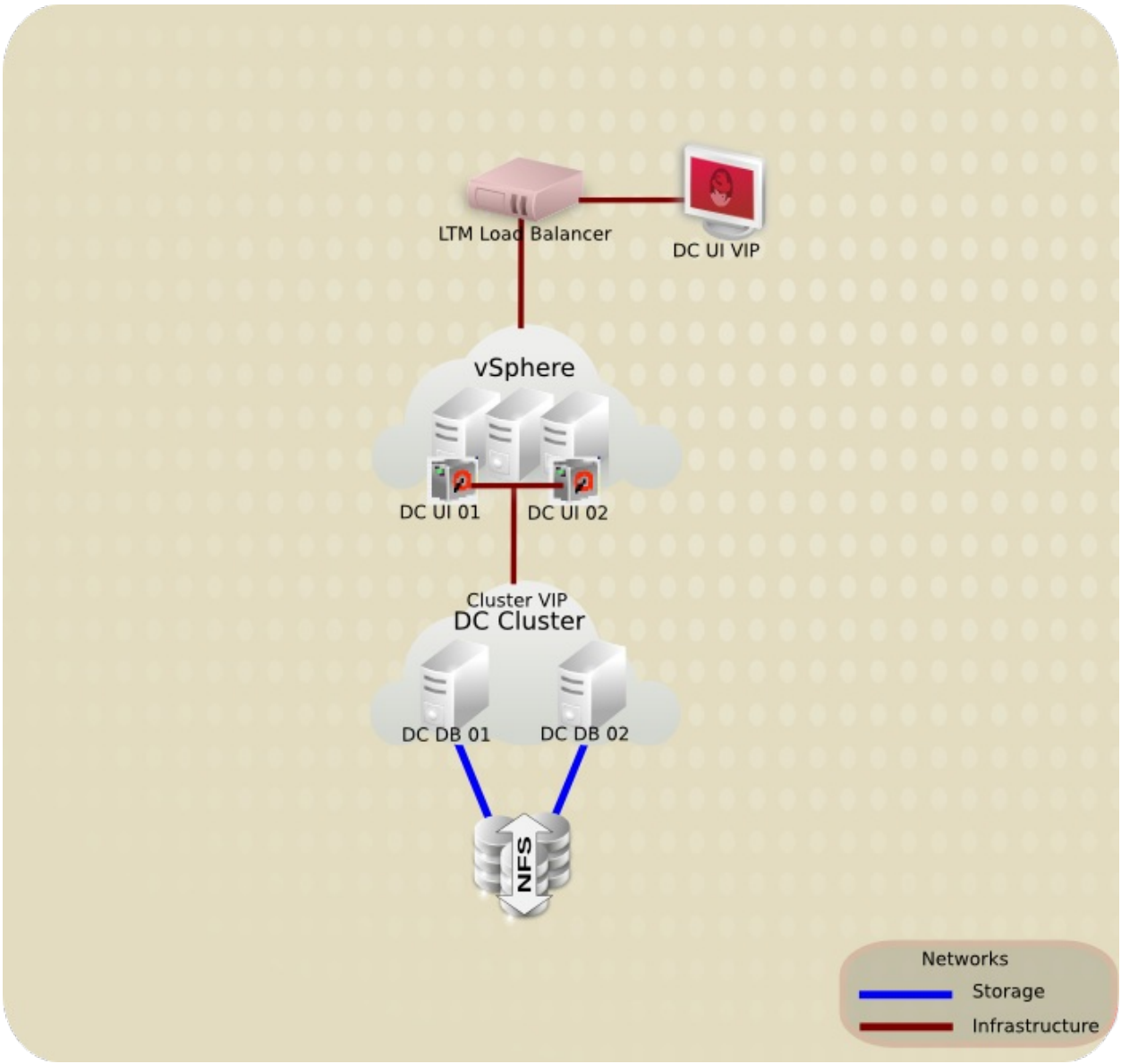

Figure 2.1.1: CloudForms HA Cluster

2.2. PRODUCTION NETWORK INFORMATION

Network details for both the database and user interface systems are listed below.

2.2.1. Database Systems

The following lists network details for the database systems used in the regional and global configurations.

Function	Hostname/IP	Storage Network
DB 01	cf-db1.example.com10.19.11.102	cf-nfs1.example.com10.18.11.106
DB 02	cf-db2.example.com10.19.11.103	cf-nfs2.example.com10.18.11.107

Table 2.3.1-1: Host Database Systems

Function	Hostname/IP	Comment
DB	cf-db.example.com10.19.11.51	DNS-based, IP address changes during failover

Table 2.3.1-2: Database Virtual IP

2.2.2. Red Hat CloudForms User Interface Appliances

The following lists network details for the CloudForms appliance systems used in the regional and global configurations.

Function	Hostname	IP
DC1 UI 01	cf-ui1.example.com	10.19.11.10
DC1 UI 02	cf-ui2.example.com	10.19.11.11

Table 2.3.2-1: User Interface CloudForms Appliances

Function	Hostname	Comment
UI	cf-ui.example.com10.19.11.50	DNS-based, IP address changes during failover

Table 2.3.2-2: User Interface Virtual IP’s

CHAPTER 3. POSTGRESQL CONFIGURATION

The following section lists details for the postgres and cluster configuration.

3.1. DATABASE SERVER SETUP

For the reference environment, PostgreSQL 9.4 is used for the shared external database for the Red Hat CloudForms appliances, running Red Hat Enterprise Linux 7.3, with one regional database servers.

Create two virtual machines, each with eight virtual CPUs, sixteen GBs of RAM and two disks.

The first disk is assigned for the operating system and is 40 GB in size.

The second disk holds the database files, is backed by NFS, and has the following size:

Region	Database Disk Size
Production, Global and Region	500 GB

Table 3.1-1: Red Hat CloudForms Database Configuration

Note

The database size relates to the number of managed VMs and hosts.[3]

3.1.1. Write Ahead Logs Archive

To allow the standby database to replay transactions that the primary has no longer available in its *pg_xlog* directory, the database is configured to archive the Write Ahead Logs.

The required sizes for the NFS volumes used for this depend on the desired retention time for Write Ahead Logs and the data change rate for each database. For the reference environment the following sizes are used:

Region	WAL Archive Size
Production, Global and Region	500 GB

Table 3.1.1-1: WAL Storage Configuration

3.1.1.1. Production Database for Global and Region

For the production database, a NFS volume that is writable by both database systems is used. The top-level directory needs to be owned by the **postgres** user and group and the permissions should be set so that only the owner has any access.

Create a directory in the NFS volume that is owned by the **postgres** user:

```
#mount <FILER_HOSTNAME:FILER_PATH> /mnt

#mkdir /mnt/wal-archive

#chown postgres:postgres /mnt/wal-archive

#umount /mnt
```

On the database systems, add the following line in */etc/fstab* :

```
<FILER_HOSTNAME:FILER_PATH>/wal-archive / /var/opt/rh/rh-
postgresql94/lib/pgsql/wal-archive nfs defaults 0 0
```

On the database systems, run the following commands to mount the NFS volumes:

```
#mkdir /var/opt/rh/rh-postgresql94/lib/pgsql/wal-archive

#mount -a

#systemctl enable nfs-utils
```

3.2. DATABASE REPLICATION SETUP

Once this basic setup is done, apply the following configuration steps only on the first database system.

3.2.1. Creating a New Database

For a new region, create an empty primary database with the following steps.

- appliance_console
- Create Internal Database
- Create v2_key
- Create New Partition
- Create New Region
- Initialize

Edit */var/opt/rh/rh-postgresql94/lib/pgsql/data/postgresql. conf* __ and change the following parameters:

```
* shared_buffers = 1GB (¼ of total allocated physical memory)
```

Disable the *evmserverd* because we do not want postgres to be managed by CloudForms, but by pacemaker. That is configured later.

```
#systemctl stop evmserverd

#systemctl disable evmserverd
```

On the secondary database, also create a new database with the same region number, but know that the database will be wiped later on.

- appliance_console
- Create Internal Database
- Fetch v2_key
- Create New Partition
- Create "New" Region
- Initialize

Edit */var/opt/rh/rh-postgresql94/lib/pgsql/data/postgresql.conf* and change the following parameters:

```
* shared_buffers = 1GB (¼ of total allocated physical memory)
```

Disable the *evmserverd* because we do not want postgres to be managed by CloudForms, but by *pacemaker* . That is configured later.

```
#systemctl stop evmserverd

#systemctl disable evmserverd
```

Also shutdown the secondary because we will be replacing its data directory with a basebackup.

```
#systemctl stop $APPLIANCE_PG_SERVICE
```

3.2.2. Common Database Configuration

Add the following lines to */var/opt/rh/rh-postgresql94/lib/pgsql/data/pg_hba.conf* to allow the CloudForms appliances to connect to the external database, to enable the **pg_basebackup** command locally, and to allow replication connections from the other cluster node.

Change the IP address range for the *cloudforms* line to only include the CloudForms appliances from the region (UI and workers).

Change the IP address range for the replicator line to only include the two database cluster nodes.

Instead of specifying a range, add multiple lines and put *IP_ADDRESS/MASK* in the second-to-last column.

```
local replication postgres peer

host replication replicator 10.19.11.0/21 md5

host vmdb_production cloudforms 10.19.11.0/21 md5
```

Edit *postgresql.conf* and make the following changes that are required for replication

```
archive_mode = on
archive_command = 'cp %p /var/opt/rh/rh-postgresql94/lib/pgsql_ /wal-
archive/%f'
```

Create the replication user on the first database.

```
#su - postgres

#createuser -P --replication replicator
```

Again, write down the password given to createuser. This is needed to configure the cluster database resource.

From the second database, take a database backup using the **pg_basebackup** command. This will do three things:

1. Test the tcp stream connectivity from secondary to primary
2. Copy over a full primary database snapshot
3. Create a wal-archive marker from which the synchronization can start

```
#su - postgres

#rm -rf /var/opt/rh/rh-postgresql94/lib/pgsql/data/*

#pg_basebackup -h primarydb.example.com -D \ /var/opt/rh/rh-
postgresql94/lib/pgsql/data/ -X stream -P -U \ replicator

<prompt for password>
```

At this point, you can test the configuration by starting the primary and then the secondary.

```
#systemctl start $APPLIANCE_PG_SERVICE
```

The primary should be doing the following:

- Writing files to: */var/opt/rh/rh-postgresql94/lib/pgsql/wal-archive*
- Running a process with a description like the following:

```
postgres: wal sender process postgres 127.0.0.1(44663) streaming
0/2000000
```

The secondary should be doing the following:

- Running a process with a description like the following:

```
postgres: wal receiver process   streaming 0/2000000
```

Once this is confirmed, shutdown the secondary, then shutdown the primary.

```
#systemctl stop $APPLIANCE_PG_SERVICE
```

[3] https://access.redhat.com/documentation/en-us/red_hat_cloudforms/4.2/html/deployment_planning_guide/

CHAPTER 4. DATABASE CLUSTER SETUP

Cluster services are used to automatically manage the primary and standby databases. For the reference environment, the Red Hat Enterprise Linux High Availability Add-On is used. Refer to **Red Hat Enterprise Linux 7: High Availability Add-on Reference**.[4] for additional information.

4.1. SETUP

Before setting up the cluster, stop the database on both nodes and turn on the appropriate firewalls:

```
#systemctl stop $APPLIANCE_PG_SERVICE

#firewall-cmd --permanent --add-service=high-availability
```

Set the password for the *'hacluster'* user which is created when pacemaker is installed. Be sure the hacluster password is the same on both systems.

```
#passwd hacluster
```

Enable the **pcsd** daemon:

```
#systemctl enable pcsd

#systemctl start pcsd
```

Install the **json_pure** gem which is required by the pcs command:

```
#gem install json_pure
```

Run the following commands on each database system. Pick a cluster name that is unique.

```
#NODE1=NODE1_HOSTNAME

#NODE2=NODE2_HOSTNAME

#CLUSTER_NAME=CLUSTER_NAME
```

Run this command to setup authorization to **pcsd** on both nodes. It will prompt for the password:

```
#pcs cluster auth $NODE1 $NODE2 -u hacluster
```

On **NODE1** (primary):

```
#pcs cluster setup --local --name $CLUSTER_NAME $NODE1 $NODE2

#pcs cluster setup --start --enable --name $CLUSTER_NAME $NODE1 $NODE2
```

Verify the cluster configuration with the following commands:

```
#pcs status
```

Next, configure fencing to allow a surviving cluster node to forcibly remove a non-responsive node from the cluster.

Note

For the reference environment VMware components are used as the virtualization provider. If using Red Hat technologies, the following specific VMware configuration is not used. Refer to **Red Hat Enterprise Linux 7: High Availability Add-on Administration**.[5] for additional information on fencing configuration and setup.

Check that both database systems can access the SOAP API of the vCenter they are running on. The vCenter user needs to be able to power virtual machines on and off.

```
#fence_vmware_soap -o list -a VCENTER_HOSTNAME -l VCENTER_USERNAME -p
VCENTER_PASSWORD -z
```

Add vCenter as a fence device to the cluster. This only needs to be run on the first database system:

```
#pcs stonith create VCENTER_NAME_stonith fence_vmware_soap
ipaddr="VCENTER_HOSTNAME" ssl="1" login='VCENTER_USERNAME'
passwd='VCENTER_PASSWORD' pcmk_host_list="$NODE1,$NODE2"
pcmk_host_check="static-list"
```

Note

For virtualization environments it is recommended to configure **`pcmk_host_map`**. Refer to the following knowledge base article for configuration:
https://access.redhat.com/solutions/701463

On both database systems, verify that fencing works. The following command reboots **NODE2** when run on **NODE1** .

```
#pcs stonith fence $NODE2
```

When it has finished rebooting, fence **NODE1** from **NODE2** :

```
#pcs stonith fence $NODE1
```

After each of the systems comes back, verify that the following command shows both cluster nodes as online, and the **`fence_vmware_soap`** agent as started:

```
#pcs status
```

Add the virtual IP address that the CloudForms appliances will use to connect to the primary database.

```
#pcs resource create pgvip ocf:heartbeat:IPaddr2 ip=VIP
cidr_netmask=NETWORK_PREFIX `iflabel=pgvip meta target-role=Started`
```

Next, create the resource for the cluster to run the PostgreSQL database. Due to using a newer

PostgreSQL version from the *Red Hat Software Collections* channel, create two custom scripts on both database systems:

/usr/local/bin/pg_ctl:

```
#cat >/usr/local/bin/pg_ctl <<'EOF'

#!/bin/bash

scl enable rh-postgresql94 -- pg_ctl "$@"

EOF
```

/usr/local/bin/psql:

```
#cat >/usr/local/bin/psql <<'EOF'

#!/bin/bash

scl enable rh-postgresql94 -- psql "$@"

EOF
```

```
#chmod 755 /usr/local/bin/{pg_ctl,psql}
```

On the **NODE1** , create the resource for the cluster to manage the PostgreSQL service. Replace *REPLICATOR_PASSWORD* and *REPLICATION_VIP* with the actual values.

```
#pcs resource create postgresql pgsql pgctl=/usr/local/bin/pg_ctl
pgdata=/var/opt/rh/rh-postgresql94/lib/pgsql/data/
psql=/usr/local/bin/psql config=/var/opt/rh/rh-
postgresql94/lib/pgsql/data/postgresql.conf rep_mode=async
repuser=replicator primary_conninfo_opt="password=REPLICATOR_PASSWORD"
node_list="$NODE1 $NODE2" restore_command='cp /var/opt/rh/rh-
postgresql94/lib/pgsql/wal-archive/%f "%p" master_ip=REPLICATION_VIP
tmpdir=/var/opt/rh/rh-postgresql94/lib/pgsql/tmp
check_wal_receiver=false restart_on_promote=true op start timeout="60s"
interval="0s" on-fail="restart" op monitor timeout="60s" interval="4s"
on-fail="restart" op monitor timeout="60s" interval="3s" on-
fail="restart" role="Master" op promote timeout="60s" interval="0s" on-
fail="restart" op demote timeout="60s" interval="0s" on-fail="stop" op
stop timeout="60s" interval="0s" op notify timeout="60s" interval="0s"

#pcs resource master postgresql-ms postgresql master-max=1 master-node-
max=1 clone-max=2 clone-node-max=1 notify=true
```

Set low timeout values for the monitor operations to speed up the master/slave negotiation when the resource is started.

Configure the cluster to keep the replication and service IP address on the same cluster node as the primary database.

```
#pcs constraint colocation add pgvip with Master postgresql-ms INFINITY

#pcs constraint order promote postgresql-ms then start pgvip
```

```
symmetrical=false score=INFINITY

#pcs constraint order demote postgresql-ms then stop pgvip
symmetrical=false score=0
```

Replace the postgresql resource agent file to support the pglogical failover. This new resource agent will run the syncsubs method every 10 minutes as defined in the resource configuration. Syncsubs reads from the pglogical tables for the subscription information and writes it to a target location which is either a local file or remote-by-ssh location. When a failure occurs, the subscription information is already available to the new master and it will read in and recreate the subscription.

```
#cp pgsql /usr/lib/ocf/resource.d/
```

Verify PostgreSQL is started:

```
#pcs status
```

This should show that the database is running as master on one node, and as slave on the other node. If it shows that the database is stopped on both nodes and provides "failed actions" details, run the following to help diagnose the issue:

```
#pcs resource debug-start postgresql
```

One more thing to do is to ensure that old WAL files are deleted. This can be done with the **`archive_cleanup_command`** option of the pgsql cluster resource script, however there is desire to keep WAL files longer than they are needed for replication and to be able to reconstruct the database from an older backup.

To do this, create a script in */etc/cron.daily* on both nodes. The **`find`** command at the end of the script is run as the **postgres** user as in production, the *wal-archive* directory is an NFS mount and not readable by the **root** user.

/etc/cron.daily/pgsql-replication:

```
#!/bin/bash

# Delete archived WAL files after N days. This assumes that we take a
full backup more often than that.

set -eu

# Exit if we are on the standby node. We only need to run the delete
command once, and the primary node is the one who # writes the files.

if [ -e /var/opt/rh/rh-postgresql94/lib/pgsql/data/recovery.conf ];
then

exit 0

fi

# The number of days after which archived WAL files will be deleted.

MAX_AGE=3

su -c "find /var/opt/rh/rh-postgresql94/lib/pgsql/wal-archive -maxdepth
```

```
1 -mtime +$MAX_AGE -type f -delete" postgres

EOF
```

```
chmod +x /etc/cron.daily/pgsql-replication
```

4.1.1. Operations

To stop all cluster resources on one node:

```
#pcs cluster standby $HOSTNAME
```

If that node is the primary database, the remaining node should automatically switch from standby to primary.

To re-enable the node to host cluster resources:

```
#pcs cluster unstandby $HOSTNAME
```

The pgsql cluster resource agent uses a lock file to track clean shutdowns of the primary/standby combo and will refuse to make a node the primary if the lock file is there.

This lock file is only deleted when the standby is stopped first, and then the primary is stopped. If the primary is stopped first, or is fenced, and the standby takes over, the lock file is not deleted.

In reference environment, this precaution is not necessary because the *Write Ahead Logs* are archived. If PostgreSQL remains in a “Stopped” state on a node and the "Failed actions" list shows "unknown error", execute to help diagnose the issue:

```
#pcs resource debug-start postgresql
```

If the node displays:

```
ERROR: My data may be inconsistent.
```

You have to remove */var/opt/rh/rh-postgresql94/lib/pgsql/tmp/PGSQL.lock* file to force start.

…delete */var/opt/rh/rh-postgresql94/lib/pgsql/tmp/PGSQL.lock* and run:

```
#pcs resource cleanup postgresql
```

---

[4] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/High_Availability_Add-On_Reference/index.html

[5] https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/High_Availability_Add-On_Administration/s1-fenceconfig-HAAA.html

CHAPTER 5. CLOUDFORMS 4.1 CLUSTER RESOURCE CONFIGURATION

With the release of CloudForms 4.1, the underlying database replication technology was changed from using **rubyrep**.[6] to **pglogical**.[7]. This technology change provides better performance as it takes advantage of the logical decoding concept versus using triggers and external process for replicating changes.

This change however poses some differences in the High Availability configuration. To implement HA using **pglogical** , the following steps are performed:

- Postgres DB HA has already been configured and failover has been tested. *postgresql.log* should show no errors; refer to **Section 3: PostgreSQL Configuration**
- **pglogical** replication has been setup and working properly from the current primary database
- Update **corosync** priority and timeout
- Create a *pgsql* resource agent for pacemaker
- Update the **postgresql-ms** cluster resource to support **pglogical**

Note

With the release of CloudForms 4.2, **repmgr** becomes the default HA technology and this document's architecture using pacemaker is deprecated.

Post cluster deployment, prior to validating, ensure that **pglogical** is configured on the database servers. Refer to **CloudForms 4.1: General Configuration Guide**.[8].

5.1. PGLOGICAL REPLICATION VALIDATION

Before continuing, we need to confirm that **pglogical** replication is working properly. Validation for **pglogical** replication is performed on the CloudForms appliance for the Global and Regional databases by performing the following:

Global

Monitor the state of the subscription in global by watching its sync status on the database. The sync status can be i, d, or r. A successful sync will list all tables with a status of "r" for "ready".

```
$ psql -d vmdb_production

psql> select sync_relname, sync_status from \

pglogical.local_sync_status;
```

Region

Check in the regional database and confirm that the subscription is active.

```
$ psql -d vmdb_production
```

```
psql> select slot_name, plugin, database, active from
pg_replication_slots;

                 slot_name                    |      plugin      |
database        |   active
pgl_vmdb_production_region_10_region_1b30d4cc | pglogical_output |
vmdb_production | t

(1 row)
```

Check progress in region by finding out how many bytes of wal archives still need to be replicated. Also you can check how much wal archives currently still exist.

```
irb> MiqPglogical.new.replication_lag
PostgreSQLAdapter#log_after_checkout, connection_pool: size: 1,
connections: 1, in use: 1, waiting_in_queue: 0 =>
[{"lag_bytes"=>"1698248",
"application_name"=>"region_10_subscription"}]

irb> MiqPglogical.new.replication_wal_retained =>
[{"retained_bytes"=>"13170184",
"slot_name"=>"pgl_vmdb_production_region_10_region_1b30d4cc"}]
```

5.2. COROSYNC UPDATES

Update the totem token timeout to 5000 milliseconds or 5 seconds. Default setting is 1000 milliseconds or 1 second.

Update */etc/corosync/corosync.conf* :

```
 totem {

 ...

 token: 5000

 }
```

Restart corosync:

```
# systemctl restart corosync
```

5.3. PACEMAKER RESOURCE AGENTS

Several changes are required for pacemaker to support **`pglogical`** . A new cluster resource is created and the existing **`postgresql-ms`** resource is modified to include **`pglogical`** options for Postgres.

5.3.1. `pgsql` Resource Agent

Update the *pgsql* cluster resource agent and place it in */var/lib/ocf/resource.d/_heartbeat/* .

Note

Refer to **Appendix E: Configuration Scripts** , *pgsql* for the cluster resource agent example. Optionally create this executable as a convenience method to watch the **`pglogical`** failover actions. If opting out, comment out the calls to this file in pgsql resource agent file.

```
$ cat > /usr/local/bin/pg_logical_failover << 'EOF'

#!/bin/bash

DT=`date`

echo "${DT} $1" >> /tmp/pg_logical_failover.log

EOF
```

```
$ chmod 755 /usr/local/bin/pg_logical_failover
```

5.3.2. `postgresql-ms` Resource Agent Update

Update the resource agent configurations to account for the new parameters and operations.

- **pglogical_user:** is the postgres database user that owns vmdb database
- **pglogical_db:** is the name of the vmdb database that contains the replication slots information
- **pglogical_shared_dir:** is a shared read/write location available to all nodes. If this parameter is not provided, it is assumed that a passwordless scp is available to write a subscriptions.dat file to each of the secondary nodes in the tmp directory
- **syncsubs:** is a new operation defined that will pull subscription data from the primary and write to a location available to the secondary nodes. If the secondary node is promoted to primary, it will read this subscription data and ensure that the slot is created in the same manner as before

```
$ pcs resource update postgresql-ms pglogical_user="root"

$ pcs resource update postgresql-ms pglogical_db="vmdb_production"

$ pcs resource update postgresql-ms \
pglogical_shared_dir="/var/opt/rh/rh-postgresql94/lib/pgsql/wal-
archive/"

$ pcs resource op add postgresql-ms syncsubs timeout=5s interval=600s
role="Master"
```

5.3.2.1. Under the Hood (informational only)

The handling of **`pglogical`** during a failover is entirely within the resource agent definition. The primary database is responsible for constantly providing all the secondary databases the information necessary to keep up the replication slots info. That is the purpose of the syncsubs operation. The interval setting is set to 600s which means this operation will update the secondary databases every

10 minutes.

The *syncsub* operation runs on the Master (primary) database and writes it to the designated tmp directory locally using the **`write_subs`** method.

```
write_subs `() {
        runasowner "pg_logical_failover master_writesubs"
        GET_SUBS_SQL="select slot_name from pg_replication_slots
where plugin =
        'pglogical_output'"
        `su $OCF_RESKEY_pgdba -c "cd $OCF_RESKEY_pgdata;
$OCF_RESKEY_psql $psql_options -U
$OCF_RESKEY_pglogical_user -d
         $OCF_RESKEY_pglogical_db -Atc \"${GET_SUBS_SQL}\" >
${OCF_RESKEY_tmpdir}/subscriptions.dat"

       if [ $? != 0 ];
       then
          ocf_exit_reason "Failed to select slot_name from
${OCF_RESKEY_pglogical_db}."
           return ${OCF_FAILED_MASTER}
       fi

       ocf_log info "Completed pg_logical_failover write subs script."
       return $OCF_RUNNING_MASTER
 }
```

The *syncsub* operation then calls the **`xfer_subs`** method to make a decision how and where to deliver the subscriptions data. The data needs to immediately be on all secondary nodes to be ready for promotion. The two methods to transfer are write to a locally available shared filesystem (like wal-archives) or scp with password-less keys.

```
xfer_subs() {
      runasowner "pg_logical_failover master_xfersubs
${OCF_RESKEY_node_list}"
      for standby in ${OCF_RESKEY_node_list};
      do
         current_host=`hostname`
         if [ ${current_host} != ${standby} ];
      then
        if [ -n "$OCF_RESKEY_pglogical_shared_dir" ];
        then
          runasowner "cp -f ${OCF_RESKEY_tmpdir}/subscriptions.da
${OCF_RESKEY_pglogical_shared_dir}"
      else
          runasowner "scp ${OCF_RESKEY_tmpdir}/subscriptions.dat
${OCF_RESKEY_pgdba}@${standby}:${OCF_RESKEY_tmpdir}"
      fi
    fi

    if [ $? != 0 ];
    then
      ocf_exit_reason "Failed to write subscriptions.dat on
${OCF_RESKEY_pgdba}@${standby} or${OCF_RESKEY_pglogical_shared_dir}."
      return ${OCF_FAILED_MASTER}
    fi
```

```
  done

  ocf_log info "Completed pg_logical_failover transfer subs script."
  return $OCF_RUNNING_MASTER
 }
```

Assuming that the subscription file is in place, if a node suddenly finds itself promoted to the primary, the promotion operation will call **read_subs** after it has switched to non-recovery mode. This **read_subs** method will create a pg logical replication slot of the same name as before if it doesn't already exist.

```
read_subs() {
      # read in the pg_logical subscription info
      runasowner "pg_logical_failover read_subs"

      if [ -n "$OCF_RESKEY_pglogical_shared_dir" ];
      then
        SLOT_NAME=$(
<${OCF_RESKEY_pglogical_shared_dir}/subscriptions.dat )
        PUT_SUBS_SQL="select * from
pg_create_logical_replication_slot('${SLOT_NAME}','pglogical_output')"

      else
        SLOT_NAME=$( <${OCF_RESKEY_tmpdir}/subscriptions.dat )
        PUT_SUBS_SQL="select * from
pg_create_logical_replication_slot('${SLOT_NAME}','pglogical_output')"
      fi

      if [ -n "${SLOT_NAME}" ];
      then
        `su $OCF_RESKEY_pgdba -c "cd $OCF_RESKEY_pgdata;
$OCF_RESKEY_psql $psql_options -U $OCF_RESKEY_pglogical_user -d
        $OCF_RESKEY_pglogical_db -Atc \"${PUT_SUBS_SQL}\" "`
      else
        ocf_exit_reason "Failed to read subscriptions.dat."
        return ${OCF_FAILED_MASTER}
      fi

      if [ $? != 0 ];
      then
        ocf_exit_reason "Failed to insert slot_name in #
{OCF_RESKEY_pglogical_db}."
        return ${OCF_FAILED_MASTER}
      fi

       ocf_log info "Completed pg_logical_failover read subs script."
 }
```

[6] http://www.rubyrep.org/

[7] https://2ndquadrant.com/en/resources/pglogical/

[8] https://access.redhat.com/documentation/en-us/red_hat_cloudforms/4.1/html/general_configuration/configuration#database-operations

CHAPTER 6. CLOUDFORMS 4.2

The following section covers high availability options when using CloudForms 4.2 to include both native PostgreSQL HA and more traditional HA using pacemaker.

6.1. BUILT-IN HIGH AVAILABILITY

With the release of CloudForms 4.2, there is a built-in database high availability option that simplifies the HA deployment and configuration for those interested in a basic configuration.

Refer to **CloudForms 4.2: Configuring High Availability**.[9] for configuration and setup.

6.2. PACEMAKER HIGH *AVAILABILITY

The process to configure CloudForms 4.2 with high availability using **`pacemaker`** is no different verses CloudForms 4.1.

CloudForms 4.2 does include an updated version of the PostgreSQL database, 9.4 to 9.5. This changes the path for the database:

Cloudforms 4.1:

/var/opt/rh/rh-postgresql94/lib/pgsql/

CloudForms 4.2:

/var/opt/rh/rh-postgresql95/lib/pgsql/

Bear this in mind when configuring CloudForms 4.2 high availability with **`pacemaker`** if following this reference architecture.

---

[9] https://access.redhat.com/documentation/en-us/red_hat_cloudforms/4.2/html/configuring_high_availability/

CHAPTER 7. CONCLUSION

Whether building fault tolerant hardware environments, utilizing software failover capabilities, or realizing the benefits of virtualized environments, high availability implementations allow IT organizations to avoid unnecessary downtime and expense for critical applications.

This reference architecture focused on implementing a highly available configuration for Red Hat CloudForms by configuring the CloudForms database for a replicated setup with one primary and one hot standby server.

The following use case is successfully demonstrated:

- Configuring a highly available Red Hat CloudForms implementation within a single datacenter, using Red Hat cluster services, to provide a highly available goblal and regional database.

By following the steps documented in this reference architecture, IT professionals can implement a highly available cloud management platform using Red Hat CloudForms.

APPENDIX A. REVISION HISTORY

Revision 1.0	March 2017	Brett Thurber and Daniel Trieu
Initial Release		

APPENDIX B. CONTRIBUTORS

Contributor	Title	Contribution
Daniel Trieu	Architect	Content, Review
Brett Thurber	Engineering Manager	Content, Review
John Ruemker	Principal Software Maintenance Engineer	Content, Review
Nick Carboni	Software Engineer	Content, Review
Andrew Beekhof	Senior Principal Software Engineer	Review
Brad Ascar	Senior Principal Product Marketing Manager	Review

APPENDIX C. TROUBLESHOOTING

The following section covers database and cluster troubleshooting steps.

C.1. CLUSTER CONFIGURATION

To check the attributes of a cluster resource, execute:

```
#pcs resource show RESOURCE`
```

Attribute values can be changed using:

```
#pcs resource update RESOURCE_NAME ATTR_NAME=ATTR_VALUE
```

If a resource fails to start and **`pcs status`** shows an error for the resource, run the following command to start it on the local node and get more details about the error:

```
#pcs resource debug-start RESOURCE
```

Note

Use caution if/when executing **`debug-restart`** . It is advised to disable the resource first to prevent conflicts, possible corruption, or resource failures. Consult Red Hat Support as needed.

To stop and start a cluster resource, execute:

```
#pcs resource disable RESOURCE

#pcs resource enable RESOURCE
```

While working on the configuration, the resource may fail to start so often that the cluster disables it permanently. This can be checked with:

```
#pcs resource failcount show postgresql
```

If the failcounts are shown as *INFINITY* , you can reset them with:

```
#pcs resource cleanup postgresql
```

C.2. REPLICATION IN A CLUSTER ENVIRONMENT

The cluster resource agent script automatically determines which of the two nodes should be the primary and which should be the standby node. The current status can be viewed with:

```
#crm_mon -Afr -1
```

If the primary and standby are both active, the output should appear as:

```
Node Attributes:
* Node cf-db1.example.com:

+ master-postgresql                : 1000
+ postgresql-data-status           : LATEST
+ postgresql-master-baseline       : 0000000010000080
+ postgresql-status                : PRI
+ postgresql-xlog-loc              : 0000000010000080

* Node cf-db2.example.com:

+ master-postgresql                : 100
+ postgresql-data-status           : STREAMING|ASYNC
+ postgresql-status                : HS:async
```

In this case, **cf-db1** is the primary, and **cf-db2** is the standby server, with streaming asynchronous replication.

If the standby lost the connection to the primary for too long and requires its database to be restored from a backup done on the primary, the output will appear as:

```
Node Attributes:
* Node cf-db1.example.com:

+ master-postgresql                : -INFINITY
+ postgresql-data-status           : DISCONNECT
+ postgresql-status                : HS:alone

* Node cf-db2.example.com:

+ master-postgresql                : 1000
+ postgresql-data-status           : LATEST
+ postgresql-master-baseline       : 0000000011000080
+ postgresql-status                : PRI
+ postgresql-xlog-loc              : 0000000011000080
```

Here, **cf-db2** is the primary, and **cf-db1** is unable to start because its database is out-of-date.

This can be caused by connection problems. Check the firewalls for both database systems, and check that *pg_hba.conf* has the same content on both systems.

If a problem is found and fixed, disable and enable the **`postgresql`** resource, run **`tail -f`** */var/log/messages* and some time after enabling the resource, one database system becomes the primary and the other one the standby.

C.3. RESTORING THE STANDBY DATABASE FROM A BACKUP

If the standby is still unable to start after checking the firewall, PostgreSQL access permissions and the NFS mount for archived Write Ahead Logs, take a backup of the primary and restore it on the standby database.

To do this, run the following commands on the standby cluster node:

```
#pcs cluster standby $HOSTNAME
```

```
#su - postgres

$rm -rf /tmp/pgbackup

$mkdir /tmp/pgbackup

$scl enable rh-postgresql94 -- pg_basebackup -h REPLICATION_VIP -U \
replicator -D /tmp/pgbackup -x

$rm -rf /var/opt/rh/rh-postgresql94/lib/pgsql/data/*

$mv /tmp/pgbackup/* /var/opt/rh/rh-postgresql94/lib/pgsql/data

$chown -R postgres:postgres \ /var/opt/rh/rh-
postgresql94/lib/pgsql/data

#pcs cluster unstandby $HOSTNAME
```

C.4. SIMULATING A NODE FAILURE

To test fencing and automating failover, trigger a kernel panic by running the command below. Before doing this, ensure access to the system console and power control.

```
#echo c >/proc/sysrq-trigger
```

Watching */var/log/messages* on the surviving node, the crashed node is fenced, and the surviving node becomes the primary database (if it was not already).

The crashed node should boot after the power off/power on cycle, automatically join the cluster and start the database as standby. If it was the primary before, *PGSQL.lock* needs to be removed as described above.

C.5. RED HAT CLOUDFORMS UI FAILOVER

To simulate a UI failure by stopping the Web server on one of the UI appliances, run the following command:

```
#service httpd stop
```

When done testing, start the Web server again with:

```
#service httpd start
```

To verify which CFME appliance serves requests, check:
/var/www/miq/vmdb/log/apache/ssl_access.log .

APPENDIX D. MAINTENANCE

While PostgreSQL and Red Hat CloudForms do most database maintenance tasks automatically, there are some additional cleanup operations that Red Hat recommends to run periodically for large databases.

Custom scripts can be created to re-index.[10] tables that frequently change and clean up tables from which CloudForms frequently deletes rows.

---

[10] https://www.postgresql.org/docs/9.4/static/sql-reindex.html

APPENDIX E. CONFIGURATION SCRIPTS

db_cluster_setup.sh

```
#!/bin/bash

source config

##REQUIREMENTS
##hacluster user has password set
##Run as root, Run on DB2 first, then DB1

if [ `whoami` != 'root' ]; then
  echo "db_cluster_setup needs to run as root"
  exit 1
fi

if [ `hostname` == $DB1NAME ]; then
  echo "Have you have run this on $DB2NAME already? [confirm|deny] :
"read response

  case $response in
  'confirm')
     echo "Continuing..."
     ;;
  'deny')
     echo "Please run this on $DB2NAME first. Exiting..."
     exit 0
     ;;
  *)
     echo "Unknown response $response ..."
     exit 2
     ;;
  esac
fi

#Open firewall

firewall-cmd --permanent --add-service=high-availability
firewall-cmd --add-service=high-availability

#Shutdown disable evmserverd and postgres
systemctl -q enable pcsd
systemctl -q start pcsd
systemctl -q stop evmserverd
systemctl -q disable evmserverd
systemctl -q stop rh-postgresql94-postgresql
systemctl -q disable rh-postgresql94-postgresql
sleep 5

#Create pg_ctl, psql commands with scl
cat > /usr/local/bin/pg_ctl << 'EOF'
#!/bin/bash

scl enable rh-postgresql94 -- pg_ctl "$@"
```

```
EOF

cat > /usr/local/bin/psql << 'EOF'
#!/bin/bash

scl enable rh-postgresql94 -- psql "$@"
EOF
chmod 755 /usr/local/bin/{pg_ctl,psql}

#Create pg_logical_failover
cat > /usr/local/bin/pg_logical_failover << 'EOF'
#!/bin/bash

DT=`date`

echo "${DT} $1" >> /tmp/pg_logical_failover.log

EOF
chmod 755 /usr/local/bin/pg_logical_failover

#Overwrite pgsql RA
mv -f /usr/lib/ocf/resource.d/heartbeat/pgsql ~/pgsql.orig
mv -f pgsql /usr/lib/ocf/resource.d/heartbeat/
chmod 755 /usr/lib/ocf/resource.d/heartbeat/pgsql

#Create tmp directory for PGSQL.lock file
mkdir -p /var/opt/rh/rh-postgresql94/lib/pgsql/tmp
chown postgres:postgres /var/opt/rh/rh-postgresql94/lib/pgsql/tmp
chmod 700 /var/opt/rh/rh-postgresql94/lib/pgsql/tmp

#Setup cluster and start cluster services
#Only on DB1
if [ `hostname` == $DB1NAME ]; then
  pcs cluster auth $DB1NAME $DB2NAME -u hacluster -p $HACLUSTER_PW
  pcs cluster setup --name $REGIONNAME $DB1NAME $DB2NAME --token 5000
  pcs cluster start --all
  pcs cluster enable --all
  sleep 5

  pcs property set no-quorum-policy=ignore

  #If stonith isn't available, set stonith-enabled=false, otherwise
true
  pcs property set stonith-enabled=false

   #pcs stonith create vcenter_stonith fence_vmware_soap
pcmk_host_check="static-list" pcmk_host_list="${DB1NAME},${DB2NAME}"
pcmk_host_map="${DB1NAME}:${DB1VM};${DB2NAME}:${DB1VM}" ipaddr=$VC_IP
login=$VC_LOGIN passwd=$VC_PASSWORD  action='reboot' ssl=1
ssl_insecure=1

  pcs resource create pgvip ocf:heartbeat:IPaddr2
   ip=$VIP_EXT_IP
   cidr_netmask=$VIP_EXT_NM

  pcs resource create pgrep ocf:heartbeat:IPaddr2
```

```
    ip=$REP_EXT_IP
    cidr_netmask=$REP_EXT_NM

  pcs resource group add pggroup pgvip pgrep

  pcs resource create pgsqld pgsql
    pgctl="/usr/local/bin/pg_ctl"
    psql="/usr/local/bin/psql"
    pgdata="/var/opt/rh/rh-postgresql94/lib/pgsql/data"
    tmpdir="/var/opt/rh/rh-postgresql94/lib/pgsql/tmp"
    config="/var/opt/rh/rh-postgresql94/lib/pgsql/data/postgresql.conf"
    logfile="/var/opt/rh/rh-postgresql94/lib/pgsql/data/pg_log/pgsql-
ra.log"
    rep_mode="async"
    repuser="replicator"
    primary_conninfo_opt=""
    node_list="$DB1NAME $DB2NAME"
    restore_command="cp /var/opt/rh/rh-postgresql94/lib/pgsql/wal-
archive/%f %p"
    master_ip=$REP_EXT_IP
    check_wal_receiver=false
    restart_on_promote=true
    pglogical_user="root"
    pglogical_db="vmdb_production"
    pglogical_shared_dir="/var/opt/rh/rh-postgresql94/lib/pgsql/
    wal-archive/"
    op start timeout=120s on-fail="restart"
    op monitor timeout=30s interval=30s on-fail="restart"
    op monitor timeout=30s interval=29s on-fail="restart"
    role="Master"
    op promote timeout=120s on-fail="restart"
    op demote timeout=120s on-fail="stop"
    op stop timeout=120s on-fail="block"
    op syncsubs timeout=5s interval=30s role="Master"
    op notify timeout=90s

  pcs resource master postgresql-ms pgsqld
    master-max=1
    master-node-max=1
    clone-max=2
    clone-node-max=1
    notify=true

  pcs constraint colocation add pggroup with Master postgresql-ms
INFINITY
  pcs constraint order promote postgresql-ms then start pggroup
symmetrical=false score=INFINITY
  pcs constraint order demote postgresql-ms then stop pggroup
  symmetrical=false score=0

fi
```

pgsql (cluster resource)

```
#!/bin/sh
#
```

```
#Description:  Manages a PostgreSQL Server as an OCF High-Availability
#              resource
#
#Authors:      Serge Dubrouski (sergeyfd@gmail.com) -- original RA
#               Florian Haas (florian@linbit.com) -- makeover
#               Takatoshi MATSUO (matsuo.tak@gmail.com) -- support
replication
#               Daniel Trieu (dtrieu@redhat.com) -- support cf
pglogical
#
#Copyright:    2006-2012 Serge Dubrouski <sergeyfd@gmail.com>
#                        and other Linux-HA contributors
#License:      GNU General Public License (GPL)
#
##############################################################################
########
#Initialization:

: ${OCF_FUNCTIONS_DIR=${OCF_ROOT}/lib/heartbeat}
. ${OCF_FUNCTIONS_DIR}/ocf-shellfuncs

#
#Get PostgreSQL Configuration parameter
#
get_pgsql_param() {
    local param_name

    param_name=$1
    perl_code="if (/^\s*$param_name[\s=]+\s*(.*)$/) {
       \$dir=\$1;
       \$dir =~ s/\s*\#.*//;
       \$dir =~ s/^'(\S*)'/\$1/;
       print \$dir;}"

    perl -ne "$perl_code" < $OCF_RESKEY_config
 }

#Defaults
OCF_RESKEY_pgctl_default=/usr/bin/pg_ctl
OCF_RESKEY_psql_default=/usr/bin/psql
OCF_RESKEY_pgdata_default=/var/lib/pgsql/data
OCF_RESKEY_pgdba_default=postgres
OCF_RESKEY_pghost_default=""
OCF_RESKEY_pgport_default=5432
OCF_RESKEY_start_opt_default=""
OCF_RESKEY_pgdb_default=template1
OCF_RESKEY_logfile_default=/dev/null
OCF_RESKEY_stop_escalate_default=30
OCF_RESKEY_monitor_user_default=""
OCF_RESKEY_monitor_password_default=""
OCF_RESKEY_monitor_sql_default="select now();"
OCF_RESKEY_check_wal_receiver_default="false"
#Defaults for replication
OCF_RESKEY_rep_mode_default=none
OCF_RESKEY_node_list_default=""
OCF_RESKEY_restore_command_default=""
```

```
OCF_RESKEY_archive_cleanup_command_default=""
OCF_RESKEY_recovery_end_command_default=""
OCF_RESKEY_master_ip_default=""
OCF_RESKEY_repuser_default="postgres"
OCF_RESKEY_primary_conninfo_opt_default=""
OCF_RESKEY_restart_on_promote_default="false"
OCF_RESKEY_tmpdir_default="/var/lib/pgsql/tmp"
OCF_RESKEY_xlog_check_count_default="3"
OCF_RESKEY_crm_attr_timeout_default="5"
OCF_RESKEY_stop_escalate_in_slave_default=30
OCF_RESKEY_pglogical_user_default=""
OCF_RESKEY_pglogical_db_default=""
OCF_RESKEY_pglogical_shared_dir_default=""

: ${OCF_RESKEY_pgctl=${OCF_RESKEY_pgctl_default}}
: ${OCF_RESKEY_psql=${OCF_RESKEY_psql_default}}
: ${OCF_RESKEY_pgdata=${OCF_RESKEY_pgdata_default}}
: ${OCF_RESKEY_pgdba=${OCF_RESKEY_pgdba_default}}
: ${OCF_RESKEY_pghost=${OCF_RESKEY_pghost_default}}
: ${OCF_RESKEY_pgport=${OCF_RESKEY_pgport_default}}
: ${OCF_RESKEY_config=${OCF_RESKEY_pgdata}/postgresql.conf}
: ${OCF_RESKEY_start_opt=${OCF_RESKEY_start_opt_default}}
: ${OCF_RESKEY_pgdb=${OCF_RESKEY_pgdb_default}}
: ${OCF_RESKEY_logfile=${OCF_RESKEY_logfile_default}}
: ${OCF_RESKEY_stop_escalate=${OCF_RESKEY_stop_escalate_default}}
: ${OCF_RESKEY_monitor_user=${OCF_RESKEY_monitor_user_default}}
: ${OCF_RESKEY_monitor_password=${OCF_RESKEY_monitor_password_default}}
: ${OCF_RESKEY_monitor_sql=${OCF_RESKEY_monitor_sql_default}}
:
${OCF_RESKEY_check_wal_receiver=${OCF_RESKEY_check_wal_receiver_default
}}

#for replication
: ${OCF_RESKEY_rep_mode=${OCF_RESKEY_rep_mode_default}}
: ${OCF_RESKEY_node_list=${OCF_RESKEY_node_list_default}}
: ${OCF_RESKEY_restore_command=${OCF_RESKEY_restore_command_default}}
:
${OCF_RESKEY_archive_cleanup_command=${OCF_RESKEY_archive_cleanup_comma
nd_default}}
:
${OCF_RESKEY_recovery_end_command=${OCF_RESKEY_recovery_end_command_def
ault}}
: ${OCF_RESKEY_master_ip=${OCF_RESKEY_master_ip_default}}
: ${OCF_RESKEY_repuser=${OCF_RESKEY_repuser_default}}
:
${OCF_RESKEY_primary_conninfo_opt=${OCF_RESKEY_primary_conninfo_opt_def
ault}}
:
${OCF_RESKEY_restart_on_promote=${OCF_RESKEY_restart_on_promote_default
}}
: ${OCF_RESKEY_tmpdir=${OCF_RESKEY_tmpdir_default}}
: ${OCF_RESKEY_xlog_check_count=${OCF_RESKEY_xlog_check_count_default}}
: ${OCF_RESKEY_crm_attr_timeout=${OCF_RESKEY_crm_attr_timeout_default}}
:
${OCF_RESKEY_stop_escalate_in_slave=${OCF_RESKEY_stop_escalate_in_slave
_default}}
```

```
: ${OCF_RESKEY_pglogical_user=${OCF_RESKEY_pglogical_user_default}}
: ${OCF_RESKEY_pglogical_db=${OCF_RESKEY_pglogical_db_default}}
:
${OCF_RESKEY_pglogical_shared_dir=${OCF_RESKEY_pglogical_shared_dir_def
ault}}

usage() {
    cat <<EOF
        usage: $0 start|stop|status|monitor|promote|demote|notify|meta-
data|syncsubs|validate-all|methods

        $0 manages a PostgreSQL Server as an HA resource.

        The 'start' operation starts the PostgreSQL server.
        The 'stop' operation stops the PostgreSQL server.
        The 'status' operation reports whether the PostgreSQL is up.
        The 'monitor' operation reports whether the PostgreSQL is
running.
        The 'promote' operation promotes the PostgreSQL server.
        The 'demote' operation demotes the PostgreSQL server.
        The 'syncsubs' operation synchronizes Pglogical subscriptions.
        The 'validate-all' operation reports whether the parameters are
valid.
        The 'methods' operation reports on the methods $0 supports.
EOF
  return $OCF_ERR_ARGS
 }

meta_data() {
    cat <<EOF
<?xml version="1.0"?>
<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">
<resource-agent name="pgsql">
<version>1.0</version>

<longdesc lang="en">
Resource script for PostgreSQL. It manages a PostgreSQL as an HA
resource.
</longdesc>
<shortdesc lang="en">Manages a PostgreSQL database instance</shortdesc>

<parameters>
<parameter name="pgctl" unique="0" required="0">
<longdesc lang="en">
Path to pg_ctl command.
</longdesc>
<shortdesc lang="en">pgctl</shortdesc>
<content type="string" default="${OCF_RESKEY_pgctl_default}" />
</parameter>

<parameter name="start_opt" unique="0" required="0">
<longdesc lang="en">
Start options (-o start_opt in pg_ctl). "-i -p 5432" for example.
</longdesc>
<shortdesc lang="en">start_opt</shortdesc>
<content type="string" default="${OCF_RESKEY_start_opt_default}" />
```

```
</parameter>
<parameter name="ctl_opt" unique="0" required="0">
<longdesc lang="en">
Additional pg_ctl options (-w, -W etc..).
</longdesc>
<shortdesc lang="en">ctl_opt</shortdesc>
<content type="string" default="${OCF_RESKEY_ctl_opt_default}" />
</parameter>

<parameter name="psql" unique="0" required="0">
<longdesc lang="en">
Path to psql command.
</longdesc>
<shortdesc lang="en">psql</shortdesc>
<content type="string" default="${OCF_RESKEY_psql_default}" />
</parameter>

<parameter name="pgdata" unique="0" required="0">
<longdesc lang="en">
Path to PostgreSQL data directory.
</longdesc>
<shortdesc lang="en">pgdata</shortdesc>
<content type="string" default="${OCF_RESKEY_pgdata_default}" />
</parameter>

<parameter name="pgdba" unique="0" required="0">
<longdesc lang="en">
User that owns PostgreSQL.
</longdesc>
<shortdesc lang="en">pgdba</shortdesc>
<content type="string" default="${OCF_RESKEY_pgdba_default}" />
</parameter>

<parameter name="pghost" unique="0" required="0">
<longdesc lang="en">
Hostname/IP address where PostgreSQL is listening
</longdesc>
<shortdesc lang="en">pghost</shortdesc>
<content type="string" default="${OCF_RESKEY_pghost_default}" />
</parameter>

<parameter name="pgport" unique="0" required="0">
<longdesc lang="en">
Port where PostgreSQL is listening
</longdesc>
<shortdesc lang="en">pgport</shortdesc>
<content type="integer" default="${OCF_RESKEY_pgport_default}" />
</parameter>

<parameter name="monitor_user" unique="0" required="0">
<longdesc lang="en">
PostgreSQL user that pgsql RA will user for monitor operations. If it's
not set
pgdba user will be used.
</longdesc>
```

```
<shortdesc lang="en">monitor_user</shortdesc>
<content type="string" default="${OCF_RESKEY_monitor_user_default}" />
</parameter>

<parameter name="monitor_password" unique="0" required="0">
<longdesc lang="en">
Password for monitor user.
</longdesc>
<shortdesc lang="en">monitor_password</shortdesc>
<content type="string" default="${OCF_RESKEY_monitor_password_default}"
/>
</parameter>

<parameter name="monitor_sql" unique="0" required="0">
<longdesc lang="en">
SQL script that will be used for monitor operations.
</longdesc>
<shortdesc lang="en">monitor_sql</shortdesc>
<content type="string" default="${OCF_RESKEY_monitor_sql_default}" />
</parameter>

<parameter name="config" unique="0" required="0">
<longdesc lang="en">
Path to the PostgreSQL configuration file for the instance.
</longdesc>
<shortdesc lang="en">Configuration file</shortdesc>
<content type="string" default="${OCF_RESKEY_pgdata}/postgresql.conf"
/>
</parameter>

<parameter name="pgdb" unique="0" required="0">
<longdesc lang="en">
Database that will be used for monitoring.
</longdesc>
<shortdesc lang="en">pgdb</shortdesc>
<content type="string" default="${OCF_RESKEY_pgdb_default}" />
</parameter>

<parameter name="logfile" unique="0" required="0">
<longdesc lang="en">
Path to PostgreSQL server log output file.
</longdesc>
<shortdesc lang="en">logfile</shortdesc>
<content type="string" default="${OCF_RESKEY_logfile_default}" />
</parameter>

<parameter name="socketdir" unique="0" required="0">
<longdesc lang="en">
Unix socket directory for PostgreSQL
</longdesc>
<shortdesc lang="en">socketdir</shortdesc>
<content type="string" default="" />
</parameter>

<parameter name="stop_escalate" unique="0" required="0">
<longdesc lang="en">
```

```
Number of shutdown retries (using -m fast) before resorting to -m
immediate
</longdesc>
<shortdesc lang="en">stop escalation</shortdesc>
<content type="integer" default="${OCF_RESKEY_stop_escalate_default}"
/>
</parameter>

<parameter name="pglogical_user" unique="0" required="0">
<longdesc lang="en">
Postgres user that runs the pglogical service.
</longdesc>
<shortdesc lang="en">Pglogical user</shortdesc>
<content type="string" default="${OCF_RESKEY_pglogical_user_default}"
/>
</parameter>

<parameter name="pglogical_db" unique="0" required="0">
<longdesc lang="en">
Postgres db that contains the pglogical tables.
</longdesc>
<shortdesc lang="en">Pglogical database</shortdesc>
<content type="string" default="${OCF_RESKEY_pglogical_db_default}" />
</parameter>

<parameter name="pglogical_shared_dir" unique="0" required="0">
<longdesc lang="en">
Postgres shared directory that is available to all nodes.
</longdesc>
<shortdesc lang="en">Pglogical shared directory</shortdesc>
<content type="string"
default="${OCF_RESKEY_pglogical_shared_dir_default}" />
</parameter>

<parameter name="rep_mode" unique="0" required="0">
<longdesc lang="en">
Replication mode may be set to "async" or "sync" or "slave".
They require PostgreSQL 9.1 or later.
Once set, "async" and "sync" require node_list, master_ip, and
restore_command parameters,as well as configuring PostgreSQL
for replication (in postgresql.conf and pg_hba.conf).

"slave" means that RA only makes recovery.conf before starting
to connect to primary which is running somewhere.
It dosen't need master/slave setting.
It requires master_ip restore_command parameters.
</longdesc>
<shortdesc lang="en">rep_mode</shortdesc>
<content type="string" default="${OCF_RESKEY_rep_mode_default}" />
</parameter>

<parameter name="node_list" unique="0" required="0">
<longdesc lang="en">
All node names. Please separate each node name with a space.
This is required for replication.
</longdesc>
```

```
<shortdesc lang="en">node list</shortdesc>
<content type="string" default="${OCF_RESKEY_node_list_default}" />
</parameter>

<parameter name="restore_command" unique="0" required="0">
<longdesc lang="en">
restore_command for recovery.conf.
This is required for replication.
</longdesc>
<shortdesc lang="en">restore_command</shortdesc>
<content type="string" default="${OCF_RESKEY_restore_command_default}"
/>
</parameter>

<parameter name="archive_cleanup_command" unique="0" required="0">
<longdesc lang="en">
archive_cleanup_command for recovery.conf.
This is used for replication and is optional.
</longdesc>
<shortdesc lang="en">archive_cleanup_command</shortdesc>
<content type="string"
default="${OCF_RESKEY_archive_cleanup_command_default}" />
</parameter>

<parameter name="recovery_end_command" unique="0" required="0">
<longdesc lang="en">
recovery_end_command for recovery.conf.
This is used for replication and is optional.
</longdesc>
<shortdesc lang="en">recovery_end_command</shortdesc>
<content type="string"
default="${OCF_RESKEY_recovery_end_command_default}" />
</parameter>

<parameter name="master_ip" unique="0" required="0">
<longdesc lang="en">
Master's floating IP address to be connected from hot standby.
This parameter is used for "primary_conninfo" in recovery.conf.
This is required for replication.
</longdesc>
<shortdesc lang="en">master ip</shortdesc>
<content type="string" default="${OCF_RESKEY_master_ip_default}" />
</parameter>

<parameter name="repuser" unique="0" required="0">
<longdesc lang="en">
User used to connect to the master server.
This parameter is used for "primary_conninfo" in recovery.conf.
This is required for replication.
</longdesc>
<shortdesc lang="en">repuser</shortdesc>
<content type="string" default="${OCF_RESKEY_repuser_default}" />
</parameter>

<parameter name="primary_conninfo_opt" unique="0" required="0">
<longdesc lang="en">
```

```
primary_conninfo options of recovery.conf except host, port, user and
application_name.
This is optional for replication.
</longdesc>
<shortdesc lang="en">primary_conninfo_opt</shortdesc>
<content type="string"
default="${OCF_RESKEY_primary_conninfo_opt_default}" />
</parameter>

<parameter name="restart_on_promote" unique="0" required="0">
<longdesc lang="en">
If this is true, RA deletes recovery.conf and restarts PostgreSQL
on promote to keep Timeline ID. It probably makes fail-over slower.
It's recommended to set on-fail of promote up as fence.
This is optional for replication.
</longdesc>
<shortdesc lang="en">restart_on_promote</shortdesc>
<content type="boolean"
default="${OCF_RESKEY_restart_on_promote_default}" />
</parameter>

<parameter name="tmpdir" unique="0" required="0">
<longdesc lang="en">
Path to temporary directory.
This is optional for replication.
</longdesc>
<shortdesc lang="en">tmpdir</shortdesc>
<content type="string" default="${OCF_RESKEY_tmpdir_default}" />
</parameter>

<parameter name="xlog_check_count" unique="0" required="0">
<longdesc lang="en">
Number of checks of xlog on monitor before promote.
This is optional for replication.
</longdesc>
<shortdesc lang="en">xlog check count</shortdesc>
<content type="integer" default="${OCF_RESKEY_check_count_default}" />
</parameter>

<parameter name="crm_attr_timeout" unique="0" required="0">
<longdesc lang="en">
The timeout of crm_attribute forever update command.
Default value is 5 seconds.
This is optional for replication.
</longdesc>
<shortdesc lang="en">The timeout of crm_attribute forever update
command.</shortdesc>
<content type="integer"
default="${OCF_RESKEY_crm_attr_timeout_default}" />
</parameter>

<parameter name="stop_escalate_in_slave" unique="0" required="0">
<longdesc lang="en">
Number of shutdown retries (using -m fast) before resorting to -m
immediate
in slave state.
```

```
This is optional for replication.
</longdesc>
<shortdesc lang="en">stop escalation_in_slave</shortdesc>
<content type="integer"
default="${OCF_RESKEY_stop_escalate_in_slave_default}" />
</parameter>

<parameter name="check_wal_receiver" unique="0" required="0">
<longdesc lang="en">
If this is true, RA checks wal_receiver process on monitor
and notifies its status using "(resource name)-receiver-status"
attribute.
It's useful for checking whether PostgreSQL (hot standby) connects to
primary.
The attribute shows status as "normal" or "ERROR".
</longdesc>
<shortdesc lang="en">check_wal_receiver</shortdesc>
<content type="boolean"
default="${OCF_RESKEY_check_wal_receiver_default}" />
</parameter>
</parameters>

<actions>
<action name="start" timeout="120" />
<action name="stop" timeout="120" />
<action name="status" timeout="60" />
<action name="monitor" depth="0" timeout="30" interval="30"/>
<action name="monitor" depth="0" timeout="30" interval="29"
role="Master" />
<action name="promote" timeout="120" />
<action name="demote" timeout="120" />
<action name="notify"   timeout="90" />
<action name="syncsubs" timeout="5" interval="600" role="Master" />
<action name="meta-data" timeout="5" />
<action name="validate-all" timeout="5" />
<action name="methods" timeout="5" />
</actions>
</resource-agent>
EOF
 }


#
#Run the given command in the Resource owner environment...
#
runasowner() {
    local quietrun=""
    local loglevel="-err"
    local var

    for var in 1 2
    do
        case "$1" in
            "-q")
                quietrun="-q"
                shift 1;;
```

```
            "warn"|"err")
                loglevel="-$1"
                shift 1;;
            *)
                ;;
        esac
    done

    ocf_run $quietrun $loglevel su $OCF_RESKEY_pgdba -c "cd
$OCF_RESKEY_pgdata; $*"
 }

#
#Shell escape
#
escape_string() {
    echo "$*" | sed -e "s/'/'\\\\''/g"
 }


#
#methods: What methods/operations do we support?
#

pgsql_methods() {
    cat <<EOF
    start
    stop
    status
    monitor
    promote
    demote
    notify
    methods
    syncsubs
    meta-data
    validate-all
EOF
 }


#pgsql_real_start: Starts PostgreSQL
pgsql_real_start() {
    local pgctl_options
    local postgres_options
    local rc

    if pgsql_status; then
        ocf_log info "PostgreSQL is already running. PID=`cat
$PIDFILE`"
        if is_replication; then
            return $OCF_ERR_GENERIC
        else
            return $OCF_SUCCESS
        fi
    fi
```

```
    #Remove postmaster.pid if it exists
    rm -f $PIDFILE

    #Remove backup_label if it exists
    if [ -f $BACKUPLABEL ] && ! is_replication; then
        ocf_log info "Removing $BACKUPLABEL. The previous backup might
have failed."
        rm -f $BACKUPLABEL
    fi

    #Check if we need to create a log file
    if ! check_log_file $OCF_RESKEY_logfile
    then
        ocf_exit_reason "PostgreSQL can't write to the log file:
$OCF_RESKEY_logfile"
        return $OCF_ERR_PERM
    fi

    #Check socket directory
    if [ -n "$OCF_RESKEY_socketdir" ]
    then
        check_socket_dir
    fi

    if [ "$OCF_RESKEY_rep_mode" = "slave" ]; then
        rm -f $RECOVERY_CONF
        make_recovery_conf || return $OCF_ERR_GENERIC
    fi

    #Set options passed to pg_ctl
    pgctl_options="$OCF_RESKEY_ctl_opt -D $OCF_RESKEY_pgdata -l
$OCF_RESKEY_logfile"

    #Set options passed to the PostgreSQL server process
    postgres_options="-c config_file=${OCF_RESKEY_config}"

    if [ -n "$OCF_RESKEY_pghost" ]; then
        postgres_options="$postgres_options -h $OCF_RESKEY_pghost"
    fi
    if [ -n "$OCF_RESKEY_start_opt" ]; then
        postgres_options="$postgres_options $OCF_RESKEY_start_opt"
    fi

    #Tack pass-through options onto pg_ctl options
    pgctl_options="$pgctl_options -o '$postgres_options'"

    #Invoke pg_ctl
    runasowner "unset PGUSER; unset PGPASSWORD; $OCF_RESKEY_pgctl
$pgctl_options start"

    if [ $? -eq 0 ]; then
        # Probably started.....
        ocf_log info "PostgreSQL start command sent."
    else
        ocf_exit_reason "Can't start PostgreSQL."
```

```
        return $OCF_ERR_GENERIC
    fi

    while :
    do
        pgsql_real_monitor warn
        rc=$?
        if [ $rc -eq $OCF_SUCCESS -o $rc -eq $OCF_RUNNING_MASTER ];
then
            break;
        fi
        sleep 1
        ocf_log debug "PostgreSQL still hasn't started yet.
Waiting..."
    done

    ocf_log info "PostgreSQL is started."
    return $rc
 }

pgsql_replication_start() {
    local rc

    #initializing for replication
    change_pgsql_status "$NODENAME" "STOP"
    delete_master_baseline
    $CRM_MASTER -v $CAN_NOT_PROMOTE
    rm -f ${XLOG_NOTE_FILE}.* $REP_MODE_CONF $RECOVERY_CONF
    if ! make_recovery_conf || ! delete_xlog_location || !
set_async_mode_all; then
        return $OCF_ERR_GENERIC
    fi

    if [ -f $PGSQL_LOCK ]; then
        ocf_exit_reason "My data may be inconsistent. You have to
remove $PGSQL_LOCK file to force start."
        return $OCF_ERR_GENERIC
    fi

    #start
    pgsql_real_start
    if [ $? -ne $OCF_SUCCESS ]; then
        return $OCF_ERR_GENERIC
    fi
    change_pgsql_status "$NODENAME" "HS:alone"
    return $OCF_SUCCESS
 }

#pgsql_start: pgsql_real_start() wrapper for replication
pgsql_start() {
    if ! is_replication; then
        pgsql_real_start
        return $?
    else
        pgsql_replication_start
        return $?
```

```
    fi
 }

#write_subs
write_subs() {
     runasowner "pg_logical_failover master_writesubs"
     GET_SUBS_SQL="select slot_name from pg_replication_slots where
plugin = 'pglogical_output'"
     `su $OCF_RESKEY_pgdba -c "cd $OCF_RESKEY_pgdata; \
      $OCF_RESKEY_psql $psql_options -U $OCF_RESKEY_pglogical_user -d
$OCF_RESKEY_pglogical_db \
      -Atc \"${GET_SUBS_SQL}\" >
${OCF_RESKEY_tmpdir}/subscriptions.dat"`


     if [ $? != 0 ];
     then
       ocf_exit_reason "Failed to select slot_name from
${OCF_RESKEY_pglogical_db}."
       return ${OCF_FAILED_MASTER}
     fi

     ocf_log info "Completed pg_logical_failover write subs script."
     return $OCF_RUNNING_MASTER
 }

#xfer_subs
xfer_subs() {
     runasowner "pg_logical_failover master_xfersubs
${OCF_RESKEY_node_list}"
     for standby in ${OCF_RESKEY_node_list};
     do
        current_host=`hostname`
        if [ ${current_host} != ${standby} ];
        then
          if [ -n "$OCF_RESKEY_pglogical_shared_dir" ];
          then
            runasowner "cp -f ${OCF_RESKEY_tmpdir}/subscriptions.dat
${OCF_RESKEY_pglogical_shared_dir}"
          else
            runasowner "scp ${OCF_RESKEY_tmpdir}/subscriptions.dat
${OCF_RESKEY_pgdba}@${standby}:${OCF_RESKEY_tmpdir}"
          fi
        fi

        if [ $? != 0 ];
        then
          ocf_exit_reason "Failed to write subscriptions.dat on
${OCF_RESKEY_pgdba}@${standby} or ${OCF_RESKEY_pglogical_shared_dir}."
          return ${OCF_FAILED_MASTER}
        fi
     done

     ocf_log info "Completed pg_logical_failover transfer subs script."
     return $OCF_RUNNING_MASTER
 }
```

```
#pgsql_syncsubs: Synchronize subscriptions to standbys
pgsql_syncsubs() {
     local rc

     # write out the pg_logical subscription info
     write_subs

     # transfer the pg_logical subscription info
     xfer_subs

     return $?
 }

#read_subs
read_subs() {
     # read in the pg_logical subscription info
     runasowner "pg_logical_failover read_subs"

    if [ -n "$OCF_RESKEY_pglogical_shared_dir" ];
    then
      SLOT_NAME=$(
<${OCF_RESKEY_pglogical_shared_dir}/subscriptions.dat )
      PUT_SUBS_SQL="select * from
pg_create_logical_replication_slot('${SLOT_NAME}', 'pglogical_output')"
    else
      SLOT_NAME=$( <${OCF_RESKEY_tmpdir}/subscriptions.dat )
      PUT_SUBS_SQL="select * from
pg_create_logical_replication_slot('${SLOT_NAME}', 'pglogical_output')"
    fi

    if [ -n "${SLOT_NAME}" ];
    then
     `su $OCF_RESKEY_pgdba -c "cd $OCF_RESKEY_pgdata; \
      $OCF_RESKEY_psql $psql_options -U $OCF_RESKEY_pglogical_user -d
$OCF_RESKEY_pglogical_db \
      -Atc \"${PUT_SUBS_SQL}\" "`
    else
      ocf_exit_reason "Failed to read subscriptions.dat."
      return ${OCF_FAILED_MASTER}
    fi

    if [ $? != 0 ];
    then
      ocf_exit_reason "Failed to insert slot_name in #
{OCF_RESKEY_pglogical_db}."
      return ${OCF_FAILED_MASTER}
    fi

     ocf_log info "Completed pg_logical_failover read subs script."
 }

#pgsql_promote: Promote PostgreSQL
pgsql_promote() {
    local target
    local rc
```

```
    if ! is_replication; then
        ocf_exit_reason "Not in a replication mode."
        return $OCF_ERR_CONFIGURED
    fi
    rm -f ${XLOG_NOTE_FILE}.*

    for target in $NODE_LIST; do
        [ "$target" = "$NODENAME" ] && continue
        change_data_status "$target" "DISCONNECT"
        change_master_score "$target" "$CAN_NOT_PROMOTE"
    done

    ocf_log info "Creating $PGSQL_LOCK."
    touch $PGSQL_LOCK
    show_master_baseline

    if ocf_is_true ${OCF_RESKEY_restart_on_promote}; then
        ocf_log info "Restarting PostgreSQL instead of promote."
        #stop : this function returns $OCF_SUCCESS only.
        pgsql_real_stop slave
        rm -f $RECOVERY_CONF
        pgsql_real_start
        rc=$?
        if [ $rc -ne $OCF_RUNNING_MASTER ]; then
            ocf_exit_reason "Can't start PostgreSQL as primary on
promote."
            if [ $rc -ne $OCF_SUCCESS ]; then
                change_pgsql_status "$NODENAME" "STOP"
            fi
            return $OCF_ERR_GENERIC
        fi
    else
        runasowner "$OCF_RESKEY_pgctl -D $OCF_RESKEY_pgdata promote"
        if [ $? -eq 0 ]; then
            ocf_log info "PostgreSQL promote command sent."
        else
            ocf_exit_reason "Can't promote PostgreSQL."
            return $OCF_ERR_GENERIC
        fi

        while :
        do
            pgsql_real_monitor warn
            rc=$?
            if [ $rc -eq $OCF_RUNNING_MASTER ]; then
                break;
            elif [ $rc -eq $OCF_ERR_GENERIC ]; then
                ocf_exit_reason "Can't promote PostgreSQL."
                return $rc
            fi
            sleep 1
            ocf_log debug "PostgreSQL still hasn't promoted yet.
Waiting..."
        done
        ocf_log info "PostgreSQL is promoted."
```

```
    fi

    #After database is promoted, check if there are subscriptions to
recover
    read_subs

    change_data_status "$NODENAME" "LATEST"
    $CRM_MASTER -v $PROMOTE_ME
    change_pgsql_status "$NODENAME" "PRI"
    return $OCF_SUCCESS
 }

#pgsql_demote: Demote PostgreSQL
pgsql_demote() {
    local rc

    if ! is_replication; then
        ocf_exit_reason "Not in a replication mode."
        return $OCF_ERR_CONFIGURED
    fi

    $CRM_MASTER -v $CAN_NOT_PROMOTE
    delete_master_baseline

    if ! pgsql_status; then
        ocf_log info "PostgreSQL is already stopped on demote."
    else
        ocf_log info "Stopping PostgreSQL on demote."
        pgsql_real_stop master
        rc=$?
        if [ "$rc" -ne "$OCF_SUCCESS" ]; then
            change_pgsql_status "$NODENAME" "UNKNOWN"
            return $rc
        fi
    fi
    change_pgsql_status "$NODENAME" "STOP"
    return $OCF_SUCCESS
 }

#pgsql_real_stop: Stop PostgreSQL
pgsql_real_stop() {
    local rc
    local count
    local stop_escalate

    if ocf_is_true ${OCF_RESKEY_check_wal_receiver}; then
        attrd_updater -n "$PGSQL_WAL_RECEIVER_STATUS_ATTR" -D -q
    fi

    if ! pgsql_status
    then
        #Already stopped
        return $OCF_SUCCESS
    fi

    stop_escalate=$OCF_RESKEY_stop_escalate
```

```
    if [ "$1" = "slave" ]; then
        stop_escalate="$OCF_RESKEY_stop_escalate_in_slave"
    fi

    # Stop PostgreSQL, do not wait for clients to disconnect
    if [ $stop_escalate -gt 0 ]; then
            runasowner "$OCF_RESKEY_pgctl -D $OCF_RESKEY_pgdata stop -
m fast"
    fi

    # stop waiting
    count=0
    while [ $count -lt $stop_escalate ]
    do
        if ! pgsql_status
        then
            #PostgreSQL stopped
            break;
        fi
        count=`expr $count + 1`
        sleep 1
    done

    if pgsql_status
    then
        #PostgreSQL is still up. Use another shutdown mode.
        ocf_log info "PostgreSQL failed to stop after
${OCF_RESKEY_stop_escalate}s using -m fast. Trying -m immediate..."
        runasowner "$OCF_RESKEY_pgctl -D $OCF_RESKEY_pgdata stop -m
immediate"
    fi

    while :
    do
        pgsql_real_monitor
        rc=$?
        if [ $rc -eq $OCF_NOT_RUNNING ]; then
            # An unnecessary debug log is prevented.
            break;
        fi
        sleep 1
        ocf_log debug "PostgreSQL still hasn't stopped yet.
Waiting..."
    done

    # Remove postmaster.pid if it exists
    rm -f $PIDFILE

    if  [ "$1" = "master" -a "$OCF_RESKEY_CRM_meta_notify_slave_uname"
= " " ]; then
        ocf_log info "Removing $PGSQL_LOCK."
        rm -f $PGSQL_LOCK
    fi
    return $OCF_SUCCESS
 }
```

```
pgsql_replication_stop() {
    local rc

    $CRM_MASTER -v $CAN_NOT_PROMOTE
    delete_xlog_location

    if ! pgsql_status
    then
        ocf_log info "PostgreSQL is already stopped."
        change_pgsql_status "$NODENAME" "STOP"
        return $OCF_SUCCESS
    fi

    pgsql_real_stop slave
    rc=$?
    if [ $rc -ne $OCF_SUCCESS ]; then
        change_pgsql_status "$NODENAME" "UNKNOWN"
        return $rc
    fi

    change_pgsql_status "$NODENAME" "STOP"
    set_async_mode_all
    delete_master_baseline
    return $OCF_SUCCESS
 }

#pgsql_stop: pgsql_real_stop() wrapper for replication
pgsql_stop() {
    if ! is_replication; then
        pgsql_real_stop
        return $?
    else
        pgsql_replication_stop
        return $?
    fi
 }

#
# pgsql_status: is PostgreSQL up?
#

pgsql_status() {
     if [ -f $PIDFILE ]
     then
         PID=`head -n 1 $PIDFILE`
         runasowner "kill -s 0 $PID >/dev/null 2>&1"
         return $?
     fi

     # No PID file
     false
 }

pgsql_wal_receiver_status() {
    local PID
    local receiver_parent_pids
```

```
    PID=`head -n 1 $PIDFILE`
    receiver_parent_pids=`ps -ef | tr -s " " | grep "[w]al receiver
process" | cut -d " " -f 3`
    if echo "$receiver_parent_pids" | grep -q -w "$PID" ; then
        attrd_updater -n "$PGSQL_WAL_RECEIVER_STATUS_ATTR" -v "normal"
-q
        return 0
    fi
    attrd_updater -n "$PGSQL_WAL_RECEIVER_STATUS_ATTR" -v "ERROR" -q
    ocf_log warn "wal receiver process is not running"
    return 1
 }

#
#pgsql_real_monitor
#

pgsql_real_monitor() {
    local loglevel
    local rc
    local output

    #Set the log level of the error message
    loglevel=${1:-err}

    if ! pgsql_status
    then
        ocf_log info "PostgreSQL is down"
        return $OCF_NOT_RUNNING
    fi

    if ocf_is_true ${OCF_RESKEY_check_wal_receiver}; then
        pgsql_wal_receiver_status
    fi

    if is_replication; then
        #Check replication state
        output=`su $OCF_RESKEY_pgdba -c "cd $OCF_RESKEY_pgdata; \
                $OCF_RESKEY_psql $psql_options -U $OCF_RESKEY_pgdba \
                -Atc \"${CHECK_MS_SQL}\""`
        rc=$?
        if [ $rc -ne  0 ]; then
            report_psql_error $rc $loglevel
            return $OCF_ERR_GENERIC
        fi

        case "$output" in
            f)  ocf_log debug "PostgreSQL is running as a primary."
                if [ "$OCF_RESKEY_monitor_sql" =
"$OCF_RESKEY_monitor_sql_default" ]; then
                    return $OCF_RUNNING_MASTER
                fi
                ;;

            t)  ocf_log debug "PostgreSQL is running as a hot
```

```
standby."
                return $OCF_SUCCESS;;

            *)  ocf_exit_reason "$CHECK_MS_SQL output is $output"
                return $OCF_ERR_GENERIC;;
        esac
    fi

    OCF_RESKEY_monitor_sql=`escape_string "$OCF_RESKEY_monitor_sql"`
    runasowner -q $loglevel "$OCF_RESKEY_psql $psql_options \
                  -c '$OCF_RESKEY_monitor_sql'"
    rc=$?
    if [ $rc -ne  0 ]; then
        report_psql_error $rc $loglevel
        return $OCF_ERR_GENERIC
    fi

    if is_replication; then
        return $OCF_RUNNING_MASTER
    fi
    return $OCF_SUCCESS
 }

pgsql_replication_monitor() {
    local rc

    rc=$1
    if [ $rc -ne $OCF_SUCCESS -a $rc -ne "$OCF_RUNNING_MASTER" ]; then
        return $rc
    fi
    #If I am Master
    if [ $rc -eq $OCF_RUNNING_MASTER ]; then
        change_data_status "$NODENAME" "LATEST"
        change_pgsql_status "$NODENAME" "PRI"
        control_slave_status || return $OCF_ERR_GENERIC
        return $rc
    fi

    #I can't get master node name from
$OCF_RESKEY_CRM_meta_notify_master_uname on monitor,
    #so I will get master node name using crm_mon -n
    crm_mon -n1 | tr -d "\t" | tr -d " " | grep -q "^${RESOURCE_NAME}
[(:].*[):]Master"
    if [ $? -ne 0 ] ; then
        # If I am Slave and Master is not exist
        ocf_log info "Master does not exist."
        change_pgsql_status "$NODENAME" "HS:alone"
        have_master_right
        if [ $? -eq 0 ]; then
            rm -f ${XLOG_NOTE_FILE}.*
        fi
    else
        output=`$CRM_ATTR_FOREVER -N "$NODENAME" \
                -n "$PGSQL_DATA_STATUS_ATTR" -G -q`
        if [ "$output" = "DISCONNECT" ]; then
            change_pgsql_status "$NODENAME" "HS:alone"
```

```
        fi
    fi
    return $rc
 }

#pgsql_monitor: pgsql_real_monitor() wrapper for replication
pgsql_monitor() {
    local rc

    pgsql_real_monitor
    rc=$?
    if ! is_replication; then
        return $rc
    else
        pgsql_replication_monitor $rc
        return $?
    fi
 }

#pgsql_post_demote
pgsql_post_demote() {
    DEMOTE_NODE=`echo $OCF_RESKEY_CRM_meta_notify_demote_uname | sed
"s/ /\n/g" | head -1 | tr '[A-Z]' '[a-z]'`
    ocf_log debug "post-demote called. Demote uname is $DEMOTE_NODE"
    if [ "$DEMOTE_NODE" != "$NODENAME" ]; then
        if ! echo $OCF_RESKEY_CRM_meta_notify_master_uname | tr '[A-Z]'
'[a-z]' | grep $NODENAME; then
            show_master_baseline
            change_pgsql_status "$NODENAME" "HS:alone"
        fi
    fi
    return $OCF_SUCCESS
 }

pgsql_pre_promote() {
    local master_baseline
    local my_master_baseline
    local cmp_location
    local number_of_nodes

    # If my data is newer than new master's one, I fail my resource.
    PROMOTE_NODE=`echo $OCF_RESKEY_CRM_meta_notify_promote_uname | \
                  sed "s/ /\n/g" | head -1 | tr '[A-Z]' '[a-z]'`
    number_of_nodes=`echo $NODE_LIST | wc -w`
    if [ $number_of_nodes -ge 3 -a \
         "$OCF_RESKEY_rep_mode" = "sync" -a \
         "$PROMOTE_NODE" != "$NODENAME" ]; then
        master_baseline=`$CRM_ATTR_REBOOT -N "$PROMOTE_NODE" -n \
                         "$PGSQL_MASTER_BASELINE" -G -q 2>/dev/null`
        if [ $? -eq 0 ]; then
            my_master_baseline=`$CRM_ATTR_REBOOT -N "$NODENAME" -n \
                                "$PGSQL_MASTER_BASELINE" -G -q
2>/dev/null`
            # get older location
            cmp_location=`printf
"$master_baseline\n$my_master_baseline\n" |\
```

```
                            sort | head -1`
            if [ "$cmp_location" != "$my_master_baseline" ]; then
                ocf_exit_reason "My data is newer than new master's
one. New master's location : $master_baseline"
                $CRM_FAILCOUNT -r $OCF_RESOURCE_INSTANCE -U $NODENAME
-v INFINITY
                return $OCF_ERR_GENERIC
            fi
        fi
    fi
    return $OCF_SUCCESS
 }

pgsql_notify() {
    local type="${OCF_RESKEY_CRM_meta_notify_type}"
    local op="${OCF_RESKEY_CRM_meta_notify_operation}"
    local rc

    if ! is_replication; then
        return $OCF_SUCCESS
    fi

    ocf_log debug "notify: ${type} for ${op}"
    case $type in
        pre)
            case $op in
                promote)
                    pgsql_pre_promote
                    return $?
                    ;;
            esac
            ;;
        post)
            case $op in
                promote)
                    delete_xlog_location
                    PROMOTE_NODE=`echo
$OCF_RESKEY_CRM_meta_notify_promote_uname | \
                                    sed "s/ /\n/g" | head -1 | tr '[A-
Z]' '[a-z]'`
                    if [ "$PROMOTE_NODE" != "$NODENAME" ]; then
                        delete_master_baseline
                    fi
                    return $OCF_SUCCESS
                    ;;
                demote)
                    pgsql_post_demote
                    return $?
                    ;;
                start|stop)
                    MASTER_NODE=`echo
$OCF_RESKEY_CRM_meta_notify_master_uname | \
                                    sed "s/ /\n/g" | head -1 | tr '[A-
Z]' '[a-z]'`
                    if [ "$NODENAME " = "$MASTER_NODE" ]; then
                        control_slave_status
```

```
                    fi
                    return $OCF_SUCCESS
                    ;;
            esac
            ;;
    esac
    return $OCF_SUCCESS
 }

control_slave_status() {
    local rc
    local data_status
    local target
    local all_data_status
    local tmp_data_status
    local node_name
    local number_of_nodes

    all_data_status=`su $OCF_RESKEY_pgdba -c "cd $OCF_RESKEY_pgdata; \
                     $OCF_RESKEY_psql $psql_options -U
$OCF_RESKEY_pgdba \
                     -Atc \"${CHECK_REPLICATION_STATE_SQL}\""`
    rc=$?
    if [ $rc -eq 0 ]; then
        if [ -n "$all_data_status" ]; then
            all_data_status=`echo $all_data_status | sed "s/\n/ /g"`
        fi
    else
        report_psql_error $rc warn
        return 1
    fi

    number_of_nodes=`echo $NODE_LIST | wc -w`
    for target in $NODE_LIST; do
        if [ "$target" = "$NODENAME" ]; then
            continue
        fi

        data_status="DISCONNECT"
        if [ -n "$all_data_status" ]; then
            for tmp_data_status in $all_data_status; do
                node_name=`echo $tmp_data_status | cut -d "|" -f 1`
                state=`echo $tmp_data_status | cut -d "|" -f 2`
                sync_state=`echo $tmp_data_status | cut -d "|" -f 3`
                ocf_log debug "node=$node_name, state=$state,
sync_state=$sync_state"
                if [ "$node_name" = "$target" ];then
                    data_status="$state|$sync_state"
                    break
                fi
            done
        fi

        case "$data_status" in
            "STREAMING|SYNC")
                change_data_status "$target" "$data_status"
```

```
                change_master_score "$target" "$CAN_PROMOTE"
                change_pgsql_status "$target" "HS:sync"
                ;;
            "STREAMING|ASYNC")
                change_data_status "$target" "$data_status"
                if [ "$OCF_RESKEY_rep_mode" = "sync" ]; then
                    change_master_score "$target" "$CAN_NOT_PROMOTE"
                    if ! is_sync_mode "$target"; then
                        set_sync_mode "$target"
                    fi
                else
                    if [ $number_of_nodes -le 2 ]; then
                        change_master_score "$target" "$CAN_PROMOTE"
                    else
                        # I can't determine which slave's data is
newest in async mode.
                        change_master_score "$target"
"$CAN_NOT_PROMOTE"
                    fi
                fi
                change_pgsql_status "$target" "HS:async"
                ;;
            "STREAMING|POTENTIAL")
                change_data_status "$target" "$data_status"
                change_master_score "$target" "$CAN_NOT_PROMOTE"
                change_pgsql_status "$target" "HS:potential"
                ;;
            "DISCONNECT")
                change_data_status "$target" "$data_status"
                change_master_score "$target" "$CAN_NOT_PROMOTE"
                if [ "$OCF_RESKEY_rep_mode" = "sync" ] && \
                   is_sync_mode "$target"; then
                    set_async_mode "$target"
                fi
                ;;
            *)
                change_data_status "$target" "$data_status"
                change_master_score "$target" "$CAN_NOT_PROMOTE"
                if [ "$OCF_RESKEY_rep_mode" = "sync" ] && \
                    is_sync_mode "$target"; then
                    set_async_mode "$target"
                fi
                change_pgsql_status "$target" "HS:connected"
                ;;
        esac
    done
    return 0
 }

have_master_right() {
    local old
    local new
    local output
    local data_status
    local node
    local mylocation
```

```
    local count
    local newestXlog
    local oldfile
    local newfile

    ocf_log debug "Checking if I have a master right."

    data_status=`$CRM_ATTR_FOREVER -N "$NODENAME" -n \
                 "$PGSQL_DATA_STATUS_ATTR" -G -q`
    if [ "$OCF_RESKEY_rep_mode" = "sync" ]; then
        if [ -n "$data_status" -a "$data_status" != "STREAMING|SYNC" -
a \
             "$data_status" != "LATEST" ]; then
            ocf_log warn "My data is out-of-date. status=$data_status"
            return 1
        fi
    else
        if [ -n "$data_status" -a "$data_status" != "STREAMING|SYNC" -
a \
             "$data_status" != "STREAMING|ASYNC" -a \
             "$data_status" != "LATEST" ]; then
            ocf_log warn "My data is out-of-date. status=$data_status"
            return 1
        fi
    fi
    ocf_log info "My data status=$data_status."

    show_xlog_location
    if [ $? -ne 0 ]; then
        ocf_exit_reason "Failed to show my xlog location."
        exit $OCF_ERR_GENERIC
    fi

    old=0
    for count in `seq $OCF_RESKEY_xlog_check_count`; do
       if [ -f ${XLOG_NOTE_FILE}.$count ]; then
           old=$count
           continue
       fi
       break
    done
    new=`expr $old + 1`

    #get xlog locations of all nodes
    for node in ${NODE_LIST}; do
        output=`$CRM_ATTR_REBOOT -N "$node" -n \
                "$PGSQL_XLOG_LOC_NAME" -G -q 2>/dev/null`
        if [ $? -ne 0 ]; then
            ocf_log warn "Can't get $node xlog location."
            continue
        else
            ocf_log info "$node xlog location : $output"
            echo "$node $output" >> ${XLOG_NOTE_FILE}.${new}
            if [ "$node" = "$NODENAME" ]; then
                mylocation=$output
            fi
```

```
        fi
    done

    oldfile=`cat ${XLOG_NOTE_FILE}.${old} 2>/dev/null`
    newfile=`cat ${XLOG_NOTE_FILE}.${new} 2>/dev/null`
    if [ "$oldfile" != "$newfile" ]; then
        #reset counter
        rm -f ${XLOG_NOTE_FILE}.*
        printf "$newfile\n" > ${XLOG_NOTE_FILE}.0
        return 1
    fi

    if [ "$new" -ge "$OCF_RESKEY_xlog_check_count" ]; then
        newestXlog=`printf "$newfile\n" | sort -t " " -k 2,3 -r | \
                    head -1 | cut -d " " -f 2`
        if [ "$newestXlog" = "$mylocation" ]; then
            ocf_log info "I have a master right."
            $CRM_MASTER -v $PROMOTE_ME
            return 0
        fi
        change_data_status "$NODENAME" "DISCONNECT"
        ocf_log info "I don't have correct master data."
        #reset counter
        rm -f ${XLOG_NOTE_FILE}.*
        printf "$newfile\n" > ${XLOG_NOTE_FILE}.0
    fi

    return 1
 }

is_replication() {
    if [ "$OCF_RESKEY_rep_mode" != "none" -a "$OCF_RESKEY_rep_mode" !=
"slave" ]; then
        return 0
    fi
    return 1
 }

get_my_location() {
    local rc
    local output
    local replay_loc
    local receive_loc
    local output1
    local output2
    local log1
    local log2
    local newer_location

    output=`su $OCF_RESKEY_pgdba -c "cd $OCF_RESKEY_pgdata; \
            $OCF_RESKEY_psql $psql_options -U $OCF_RESKEY_pgdba \
            -Atc \"${CHECK_XLOG_LOC_SQL}\""`
    rc=$?
    if [ $rc -ne 0 ]; then
        report_psql_error $rc warn
        ocf_log err "Can't get my xlog location."
```

```
        return 1
    fi
    replay_loc=`echo $output | cut -d "|" -f 1`
    receive_loc=`echo $output | cut -d "|" -f 2`

    output1=`echo "$replay_loc" | cut -d "/" -f 1`
    output2=`echo "$replay_loc" | cut -d "/" -f 2`
    log1=`printf "%08s\n" $output1 | sed "s/ /0/g"`
    log2=`printf "%08s\n" $output2 | sed "s/ /0/g"`
    replay_loc="${log1}${log2}"

    output1=`echo "$receive_loc" | cut -d "/" -f 1`
    output2=`echo "$receive_loc" | cut -d "/" -f 2`
    log1=`printf "%08s\n" $output1 | sed "s/ /0/g"`
    log2=`printf "%08s\n" $output2 | sed "s/ /0/g"`
    receive_loc="${log1}${log2}"

    newer_location=`printf "$replay_loc\n$receive_loc" | sort -r | head
-1`
    echo "$newer_location"
    return 0
 }

show_xlog_location() {
    local location

    location=`get_my_location` || return 1
    $CRM_ATTR_REBOOT -N "$NODENAME" -n "$PGSQL_XLOG_LOC_NAME" -v
"$location"
 }

delete_xlog_location() {
    $CRM_ATTR_REBOOT -N "$NODENAME" -n "$PGSQL_XLOG_LOC_NAME" -D
 }

show_master_baseline() {
    local rc
    local location

    runasowner -q err "$OCF_RESKEY_psql $psql_options \
                       -U $OCF_RESKEY_pgdba -c 'CHECKPOINT'"
    rc=$?
    if [ $rc -ne 0 ]; then
        report_psql_error $rc warn
    fi
    location=`get_my_location`
    ocf_log info "My master baseline : $location."
    $CRM_ATTR_REBOOT -N "$NODENAME" -n "$PGSQL_MASTER_BASELINE" -v
"$location"
 }

delete_master_baseline() {
    $CRM_ATTR_REBOOT -N "$NODENAME" -n "$PGSQL_MASTER_BASELINE" -D
 }

set_async_mode_all() {
```

```
    [ "$OCF_RESKEY_rep_mode" = "sync" ] || return 0
    ocf_log info "Set all nodes into async mode."
    runasowner -q err "echo \"synchronous_standby_names = ''\" >
\"$REP_MODE_CONF\""
    if [ $? -ne 0 ]; then
        ocf_exit_reason "Can't set all nodes into async mode."
        return 1
    fi
    return 0
 }

set_async_mode() {
    local sync_node_in_conf

    sync_node_in_conf=`cat $REP_MODE_CONF | cut -d "'" -f 2`
    if [ -n "$sync_node_in_conf" ]; then
        ocf_log info "Setup $1 into async mode."
        sync_node_in_conf=`echo $sync_node_in_conf | sed "s/$1//g" |\
                           sed "s/^,//g" | sed "s/,,/,/g" | sed
"s/,$//g"`
        echo "synchronous_standby_names = '$sync_node_in_conf'" >
"$REP_MODE_CONF"
    else
        ocf_log info "$1 is already in async mode."
        return 0
    fi

    ocf_log info "All synced nodes : \"$sync_node_in_conf\""
    reload_conf
 }

set_sync_mode() {
    local sync_node_in_conf

    sync_node_in_conf=`cat $REP_MODE_CONF | cut -d "'" -f 2`
    if [ -n "$sync_node_in_conf" ]; then
        ocf_log info "Setup $1 into sync mode."
        echo "synchronous_standby_names = '$sync_node_in_conf,$1'" >
"$REP_MODE_CONF"
    else
        ocf_log info "Setup $1 into sync mode."
        echo "synchronous_standby_names = '$1'" > "$REP_MODE_CONF"
    fi

    sync_node_in_conf=`cat $REP_MODE_CONF | cut -d "'" -f 2`
    ocf_log info "All synced nodes : \"$sync_node_in_conf\""
    reload_conf
 }

is_sync_mode() {
    cat $REP_MODE_CONF |  grep -q -e "[,' ]$1[,' ]"
 }

reload_conf() {
    #Invoke pg_ctl
    runasowner "$OCF_RESKEY_pgctl -D $OCF_RESKEY_pgdata reload"
```

```
    if [ $? -eq 0 ]; then
        ocf_log info "Reload configuration file."
    else
        ocf_exit_reason "Can't reload configuration file."
        return 1
    fi

    return 0
 }

user_recovery_conf() {
    #put archive_cleanup_command and recovery_end_command only when
defined by user
    if [ -n "$OCF_RESKEY_archive_cleanup_command" ]; then
        echo "archive_cleanup_command =
'${OCF_RESKEY_archive_cleanup_command}'"
    fi
    if [ -n "$OCF_RESKEY_recovery_end_command" ]; then
        echo "recovery_end_command =
'${OCF_RESKEY_recovery_end_command}'"
    fi
 }

make_recovery_conf() {
    runasowner "touch $RECOVERY_CONF"
    if [ $? -ne 0 ]; then
        ocf_exit_reason "Can't create recovery.conf."
        return 1
    fi

cat > $RECOVERY_CONF <<END
standby_mode = 'on'
primary_conninfo = 'host=${OCF_RESKEY_master_ip}
port=${OCF_RESKEY_pgport} user=${OCF_RESKEY_repuser}
application_name=${NODENAME} ${OCF_RESKEY_primary_conninfo_opt}'
restore_command = '${OCF_RESKEY_restore_command}'
recovery_target_timeline = 'latest'
END

    user_recovery_conf >> $RECOVERY_CONF
    ocf_log debug "Created recovery.conf. host=${OCF_RESKEY_master_ip},
user=${OCF_RESKEY_repuser}"
    return 0
 }

#change pgsql-status.
#arg1:node, arg2: value
change_pgsql_status() {
    local output

    if ! is_node_online $1; then
        return 0
    fi

    output=`$CRM_ATTR_REBOOT -N "$1" -n "$PGSQL_STATUS_ATTR" -G -q
2>/dev/null`
```

```
    if [ "$output" != "$2" ]; then
        #If slave's disk is broken, RA cannot read PID file
        #and misjudges the PostgreSQL as down while it is running.
        #It causes overwriting of pgsql-status by Master because
replication is still connected.
        if [ "$output" = "STOP" -o "$output" = "UNKNOWN" ]; then
            if [ "$1" != "$NODENAME" ]; then
                ocf_log warn "Changing $PGSQL_STATUS_ATTR on $1 :
$output->$2 by $NODENAME is prohibited."
                return 0
            fi
        fi
        ocf_log info "Changing $PGSQL_STATUS_ATTR on $1 : $output->$2."
        $CRM_ATTR_REBOOT -N "$1" -n "$PGSQL_STATUS_ATTR" -v "$2"
        if [ $? -ne 0 ]; then
            ocf_log err "Can't change $PGSQL_STATUS_ATTR."
            return 1
        fi
    fi
    return 0
 }

#change pgsql-data-status.
#arg1:node, arg2: value
change_data_status() {
    local output

    if ! node_exist $1; then
        return 0
    fi

    while :
    do
        output=`$CRM_ATTR_FOREVER -N "$1" -n "$PGSQL_DATA_STATUS_ATTR"
-G -q 2>/dev/null`
        if [ "$output" != "$2" ]; then
            ocf_log info "Changing $PGSQL_DATA_STATUS_ATTR on $1 :
$output->$2."
            exec_func_with_timeout "$CRM_ATTR_FOREVER" "-N $1 -n \
                                    $PGSQL_DATA_STATUS_ATTR -v
\"$2\"" \
                                    $OCF_RESKEY_crm_attr_timeout
            if [ $? -ne 0 ]; then
                ocf_log err "Can't change $PGSQL_DATA_STATUS_ATTR."
                return 1
            fi
        else
            break
        fi
    done
    return 0
 }

#set master-score
#arg1:node, arg2: score, arg3: resoure
set_master_score() {
```

```
    local current_score

    current_score=`$CRM_ATTR_REBOOT -N "$1" -n "master-$3" -G -q
2>/dev/null`
    if [ -n "$current_score" -a "$current_score" != "$2" ]; then
        ocf_log info "Changing $3 master score on $1 : $current_score-
>$2."
        $CRM_ATTR_REBOOT -N "$target" -n "master-$3" -v "$2"
        if [ $? -ne 0 ]; then
            ocf_log err "Can't change master score."
            return 1
        fi
    fi
    return 0
 }

#change master-score
#arg1:node, arg2: score
change_master_score() {
    local instance

    if ! is_node_online $1; then
        return 0
    fi

    if echo $OCF_RESOURCE_INSTANCE | grep -q ":"; then
        # If Pacemaker version is 1.0.x
        instance=0
        while :
        do
            if [ "$instance" -ge "$OCF_RESKEY_CRM_meta_clone_max" ];
then
                break
            fi
            if [ "${RESOURCE_NAME}:${instance}" =
"$OCF_RESOURCE_INSTANCE" ]; then
                instance=`expr $instance + 1`
                continue
            fi
            set_master_score $1 $2 "${RESOURCE_NAME}:${instance}" ||
return 1
            instance=`expr $instance + 1`
        done
    else
        #If globally-unique=false and Pacemaker version is 1.1.8 or
higher
        #Master/Slave resource has no instance number
        set_master_score $1 $2 ${RESOURCE_NAME} || return 1
    fi
    return 0
 }

report_psql_error()
 {
    local rc
    local loglevel
```

```
    rc=$1
    loglevel=${2:-err}

    ocf_log $loglevel "PostgreSQL $OCF_RESKEY_pgdb isn't running"
    if [ $rc -eq 1 ]; then
        ocf_exit_reason "Fatal error (out of memory, file not found,
etc.) occurred while executing the psql command."
    elif [ $rc -eq 2 ]; then
        ocf_log $loglevel "Connection error (connection to the server
went bad and the session was not interactive) occurred while executing
the psql command."
    elif [ $rc -eq 3 ]; then
        ocf_exit_reason "Script error (the variable ON_ERROR_STOP was
set) occurred while executing the psql command."
    fi
 }

#
#timeout management function
#arg1 : command
#arg2 : command's args
#arg3 : timeout(s)
#
exec_func_with_timeout() {
    local func_pid
    local count
    local rc

    $1 `eval echo $2` &
    func_pid=$!
    count=0
    while kill -s 0 $func_pid >/dev/null 2>&1; do
        sleep 1
        count=`expr $count + 1`
        if [ $count -ge $3 ]; then
            ocf_log debug "Execute $1 time out."
            kill -s 9 $func_pid >/dev/null 2>&1
            return 0
        fi
    done
    wait $func_pid
 }

is_node_online() {
    crm_mon -1 -n | tr '[A-Z]' '[a-z]' | grep -e "^node $1 " -e "^node
$1:" | grep -q -v "offline"
 }

node_exist() {
    crm_mon -1 -n | tr '[A-Z]' '[a-z]' | grep -q "^node $1"
 }

check_binary2() {
    if ! have_binary "$1"; then
        ocf_exit_reason "Setup problem: couldn't find command: $1"
```

```
        return 1
    fi
    return 0
 }

check_config() {
    local rc=0

    if [ ! -f "$1" ]; then
        if ocf_is_probe; then
           ocf_log info "Configuration file is $1 not readable during
probe."
           rc=1
        else
           ocf_exit_reason "Configuration file $1 doesn't exist"
           rc=2
        fi
    fi

    return $rc
 }

#Validate most critical parameters
pgsql_validate_all() {
    local version
    local check_config_rc
    local rep_mode_string

    if ! check_binary2 "$OCF_RESKEY_pgctl" ||
       ! check_binary2 "$OCF_RESKEY_psql"; then
        return $OCF_ERR_INSTALLED
    fi

    check_config "$OCF_RESKEY_config"
    check_config_rc=$?
    [ $check_config_rc -eq 2 ] && return $OCF_ERR_INSTALLED
    [ $check_config_rc -eq 0 ] && :
${OCF_RESKEY_socketdir=`get_pgsql_param unix_socket_directory`}

    getent passwd $OCF_RESKEY_pgdba >/dev/null 2>&1
    if [ ! $? -eq 0 ]; then
        ocf_exit_reason "User $OCF_RESKEY_pgdba doesn't exist";
        return $OCF_ERR_INSTALLED;
    fi

    if ocf_is_probe; then
        ocf_log info "Don't check $OCF_RESKEY_pgdata during probe"
    else
        if ! runasowner "test -w $OCF_RESKEY_pgdata"; then
            ocf_exit_reason "Directory $OCF_RESKEY_pgdata is not
writable by $OCF_RESKEY_pgdba"
            return $OCF_ERR_PERM;
        fi
    fi

    if [ -n "$OCF_RESKEY_monitor_user" -a ! -n
```

```
"$OCF_RESKEY_monitor_password" ]
    then
        ocf_exit_reason "monitor password can't be empty"
        return $OCF_ERR_CONFIGURED
    fi

    if [ ! -n "$OCF_RESKEY_monitor_user" -a -n
"$OCF_RESKEY_monitor_password" ]
    then
        ocf_exit_reason "monitor_user has to be set if monitor_password
is set"
        return $OCF_ERR_CONFIGURED
    fi

    if is_replication || [ "$OCF_RESKEY_rep_mode" = "slave" ]; then
        version=`cat $OCF_RESKEY_pgdata/PG_VERSION`
        if [ `printf "$version\n9.1" | sort -n | head -1` != "9.1" ];
then
            ocf_exit_reason "Replication mode needs PostgreSQL 9.1 or
higher."
            return $OCF_ERR_INSTALLED
        fi
        if [ ! -n "$OCF_RESKEY_master_ip" ]; then
            ocf_exit_reason "master_ip can't be empty."
            return $OCF_ERR_CONFIGURED
        fi
    fi

    if is_replication; then
        if ! ocf_is_ms; then
            ocf_exit_reason "Replication(rep_mode=async or sync)
requires Master/Slave configuration."
            return $OCF_ERR_CONFIGURED
        fi
        if [ ! "$OCF_RESKEY_rep_mode" = "sync" -a !
"$OCF_RESKEY_rep_mode" = "async" ]; then
            ocf_exit_reason "Invalid rep_mode : $OCF_RESKEY_rep_mode"
            return $OCF_ERR_CONFIGURED
        fi
        if [ ! -n "$NODE_LIST" ]; then
            ocf_exit_reason "node_list can't be empty."
            return $OCF_ERR_CONFIGURED
        fi
        if [ $check_config_rc -eq 0 ]; then
            rep_mode_string="include '$REP_MODE_CONF' # added by pgsql
RA"
            if [ "$OCF_RESKEY_rep_mode" = "sync" ]; then
                if ! grep -q "$rep_mode_string" $OCF_RESKEY_config;
then
                    ocf_log info "adding include directive into
$OCF_RESKEY_config"
                    echo "$rep_mode_string" >> $OCF_RESKEY_config
                fi
            else
                if grep -q "$rep_mode_string" $OCF_RESKEY_config; then
                    ocf_log info "deleting include directive from
```

```
$OCF_RESKEY_config"
                    sed -i "/${rep_mode_string//\//\\/}/d"
$OCF_RESKEY_config
                fi
            fi
        fi
        if ! mkdir -p $OCF_RESKEY_tmpdir || ! chown $OCF_RESKEY_pgdba
$OCF_RESKEY_tmpdir || ! chmod 700 $OCF_RESKEY_tmpdir; then
            ocf_exit_reason "Can't create directory $OCF_RESKEY_tmpdir
or it is not readable by $OCF_RESKEY_pgdba"
            return $OCF_ERR_PERM
        fi
    fi

    if [ "$OCF_RESKEY_rep_mode" = "slave" ]; then
        if ocf_is_ms; then
            ocf_exit_reason "Replication(rep_mode=slave) does not
support Master/Slave configuration."
            return $OCF_ERR_CONFIGURED
        fi
    fi

    return $OCF_SUCCESS
 }


#
#Check if we need to create a log file
#

check_log_file() {
    if [ ! -f "$1" ]
    then
        touch $1 > /dev/null 2>&1
        chown $OCF_RESKEY_pgdba:`getent passwd $OCF_RESKEY_pgdba | cut
-d ":" -f 4` $1
    fi

    #Check if $OCF_RESKEY_pgdba can write to the log file
    if ! runasowner "test -w $1"
    then
        return 1
    fi

    return 0
 }

#
#Check socket directory
#
check_socket_dir() {
    if [ ! -d "$OCF_RESKEY_socketdir" ]; then
        if ! mkdir "$OCF_RESKEY_socketdir"; then
            ocf_exit_reason "Can't create directory
$OCF_RESKEY_socketdir"
            exit $OCF_ERR_PERM
```

```
        fi

        if ! chown $OCF_RESKEY_pgdba:`getent passwd \
             $OCF_RESKEY_pgdba | cut -d ":" -f 4`
"$OCF_RESKEY_socketdir"
        then
            ocf_exit_reason "Can't change ownership for
$OCF_RESKEY_socketdir"
            exit $OCF_ERR_PERM
        fi

        if ! chmod 2775 "$OCF_RESKEY_socketdir"; then
            ocf_exit_reason "Can't change permissions for
$OCF_RESKEY_socketdir"
            exit $OCF_ERR_PERM
        fi
    else
        if ! runasowner "touch $OCF_RESKEY_socketdir/test.$$"; then
            ocf_exit_reason "$OCF_RESKEY_pgdba can't create files in
$OCF_RESKEY_socketdir"
            exit $OCF_ERR_PERM
        fi
        rm $OCF_RESKEY_socketdir/test.$$
    fi
 }

#
#'main' starts here...
#


if [ $# -ne 1 ]
then
    usage
    exit $OCF_ERR_GENERIC
fi


PIDFILE=${OCF_RESKEY_pgdata}/postmaster.pid
BACKUPLABEL=${OCF_RESKEY_pgdata}/backup_label
RESOURCE_NAME=`echo $OCF_RESOURCE_INSTANCE | cut -d ":" -f 1`
PGSQL_WAL_RECEIVER_STATUS_ATTR="${RESOURCE_NAME}-receiver-status"
RECOVERY_CONF=${OCF_RESKEY_pgdata}/recovery.conf
NODENAME=$(ocf_local_nodename | tr '[A-Z]' '[a-z]')

if is_replication; then
    REP_MODE_CONF=${OCF_RESKEY_tmpdir}/rep_mode.conf
    PGSQL_LOCK=${OCF_RESKEY_tmpdir}/PGSQL.lock
    XLOG_NOTE_FILE=${OCF_RESKEY_tmpdir}/xlog_note

    CRM_MASTER="${HA_SBIN_DIR}/crm_master -l reboot"
    CRM_ATTR_REBOOT="${HA_SBIN_DIR}/crm_attribute -l reboot"
    CRM_ATTR_FOREVER="${HA_SBIN_DIR}/crm_attribute -l forever"
    CRM_FAILCOUNT="${HA_SBIN_DIR}/crm_failcount"

    CAN_NOT_PROMOTE="-INFINITY"
```

```
    CAN_PROMOTE="100"
    PROMOTE_ME="1000"

    CHECK_MS_SQL="select pg_is_in_recovery()"
    CHECK_XLOG_LOC_SQL="select
pg_last_xlog_replay_location(),pg_last_xlog_receive_location()"
    CHECK_REPLICATION_STATE_SQL="select
application_name,upper(state),upper(sync_state) from
pg_stat_replication"

    PGSQL_STATUS_ATTR="${RESOURCE_NAME}-status"
    PGSQL_DATA_STATUS_ATTR="${RESOURCE_NAME}-data-status"
    PGSQL_XLOG_LOC_NAME="${RESOURCE_NAME}-xlog-loc"
    PGSQL_MASTER_BASELINE="${RESOURCE_NAME}-master-baseline"

    NODE_LIST=`echo $OCF_RESKEY_node_list | tr '[A-Z]' '[a-z]'`
fi

case "$1" in
    methods)    pgsql_methods
                exit $?;;

    meta-data)  meta_data
                exit $OCF_SUCCESS;;
esac

pgsql_validate_all
rc=$?

[ "$1" = "validate-all" ] && exit $rc

if [ $rc -ne 0 ]
then
    case "$1" in
        stop)    if is_replication; then
                    change_pgsql_status "$NODENAME" "UNKNOWN"
                 fi
                 exit $OCF_SUCCESS;;
        monitor) exit $OCF_NOT_RUNNING;;
        status)  exit $OCF_NOT_RUNNING;;
        *)       exit $rc;;
    esac
fi

US=`id -u -n`

if [ $US != root -a $US != $OCF_RESKEY_pgdba ]
then
    ocf_exit_reason "$0 must be run as root or $OCF_RESKEY_pgdba"
    exit $OCF_ERR_GENERIC
fi

#make psql command options
if [ -n "$OCF_RESKEY_monitor_user" ]; then
    PGUSER=$OCF_RESKEY_monitor_user; export PGUSER
    PGPASSWORD=$OCF_RESKEY_monitor_password; export PGPASSWORD
```

```
    psql_options="-p $OCF_RESKEY_pgport $OCF_RESKEY_pgdb"
else
    psql_options="-p $OCF_RESKEY_pgport -U $OCF_RESKEY_pgdba
$OCF_RESKEY_pgdb"
fi

if [ -n "$OCF_RESKEY_pghost" ]; then
   psql_options="$psql_options -h $OCF_RESKEY_pghost"
else
   if [ -n "$OCF_RESKEY_socketdir" ]; then
       psql_options="$psql_options -h $OCF_RESKEY_socketdir"
   fi
fi

#What kind of method was invoked?
case "$1" in
    status)     if pgsql_status
                then
                    ocf_log info "PostgreSQL is up"
                    exit $OCF_SUCCESS
                else
                    ocf_log info "PostgreSQL is down"
                    exit $OCF_NOT_RUNNING
                fi;;

    monitor)    pgsql_monitor
                exit $?;;

    start)      pgsql_start
                exit $?;;

    promote)    pgsql_promote
                exit $?;;

    demote)     pgsql_demote
                exit $?;;

    notify)     pgsql_notify
                exit $?;;

    stop)       pgsql_stop
                exit $?;;

    syncsubs)   pgsql_syncsubs
                exit $?;;

    *)
                exit $OCF_ERR_UNIMPLEMENTED;;
esac
```

postgres config

```
#The pcs library installs a user called hacluster
#You will first need to update its password then put it here.
export HACLUSTER_PW=hacluster
```

```
#The first database, or the currently active. Fully qualified hostname.
export DB1NAME=$DB1NAME


#The second database, or the  currently passive. Full qualified
hostname.
export DB2NAME=$DB2NAME


#The custom cluster name. Preferably the region's name.
export REGIONNAME=$REGIONNAME


#VIP netmask in CIDR form
export VIP_EXT_NM=16


#VIP ipaddress in quad octet
export VIP_EXT_IP=10.19.11.51


#REP netmask in CIDR form
export REP_EXT_NM=16

#REP ipaddress in quad octet
export REP_EXT_IP=10.19.11.52

#vCenter ipaddress and credentials where fencing will occur.
#ie, where both database VMs are hosted
export VC_IP=10.19.11.40
export VC_LOGIN='svc-cloudforms'
export VC_PASSWORD='<REDACTED>'

#The nodes needs to be mapped to their vCenter names in order for the
correct VM to be fenced. Put the vCenter VM names here.
export DB1VM=
export DB2VM=
```

APPENDIX F. REVISION HISTORY

Revision 1.0-0	**2017-03**	**DT**