Red Hat Reference Architecture Series

# Red Hat CloudForms:

## Implementing a Highly Available Virtual Management Database

**Version 1.2**

**October 2015**

redhat.

100 East Davie Street
Raleigh NC 27601 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588
Research Triangle Park NC 27709 USA



The GPG fingerprint of the security@redhat.com key is:
CA 20 86 86 2B D6 9D FC 65 F6 EC C4 21 91 80 CD DB 42 A6 0E

## Comments and Feedback

In the spirit of open source, we invite anyone to provide feedback and comments on any reference architectures. 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.

## Staying In Touch

Join us on some of the popular social media sites where we keep our audience informed on new reference architectures as well as offer related information on things we find interesting.

**Like us on Facebook:**

https://www.facebook.com/rhrefarch

**Follow us on Twitter:**

https://twitter.com/RedHatRefArch

**Plus us on Google+:**

https://plus.google.com/u/0/b/114152126783830728030/

# Table of Contents

# 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 cases include:

- Configuring a highly available CloudForms implementation across two data centers with a regional database for each data center, using Red Hat cluster services to provide a highly available database for each region
- Configuring a highly available CloudForms implementation across two data centers with a single global database mirroring data between the database servers

**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 http://www.redhat.com/en/technologies/cloud-computing/cloudforms
2 https://access.redhat.com/support/offerings/production/soc

# 2 Environment

For the reference environment, the following diagrams illustrate the high availability configurations deployed.

## *2.1 Regional Database Cluster*

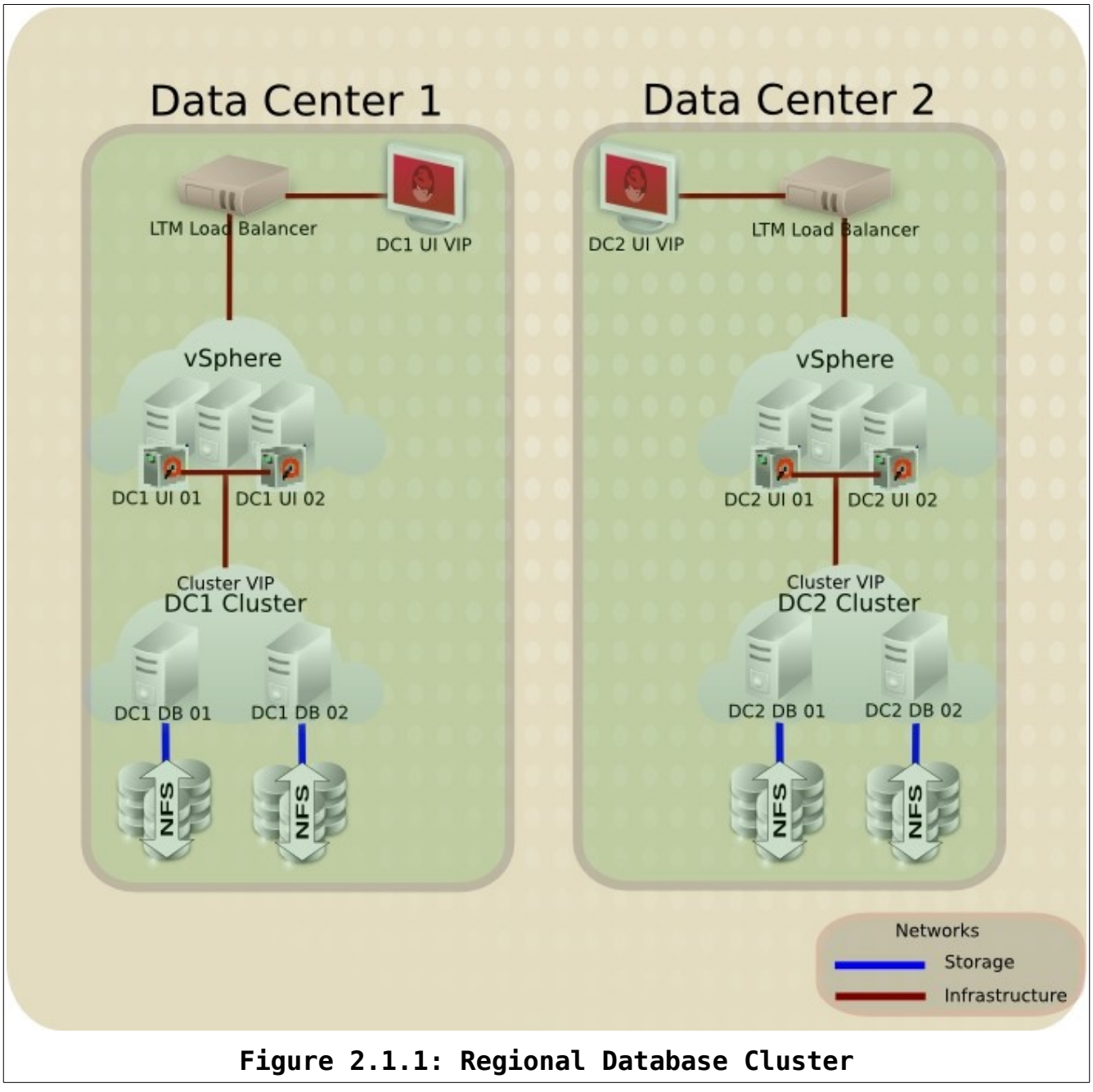


**Figure 2.1.1: Regional Database Cluster**

## 2.2 Global Database Cluster

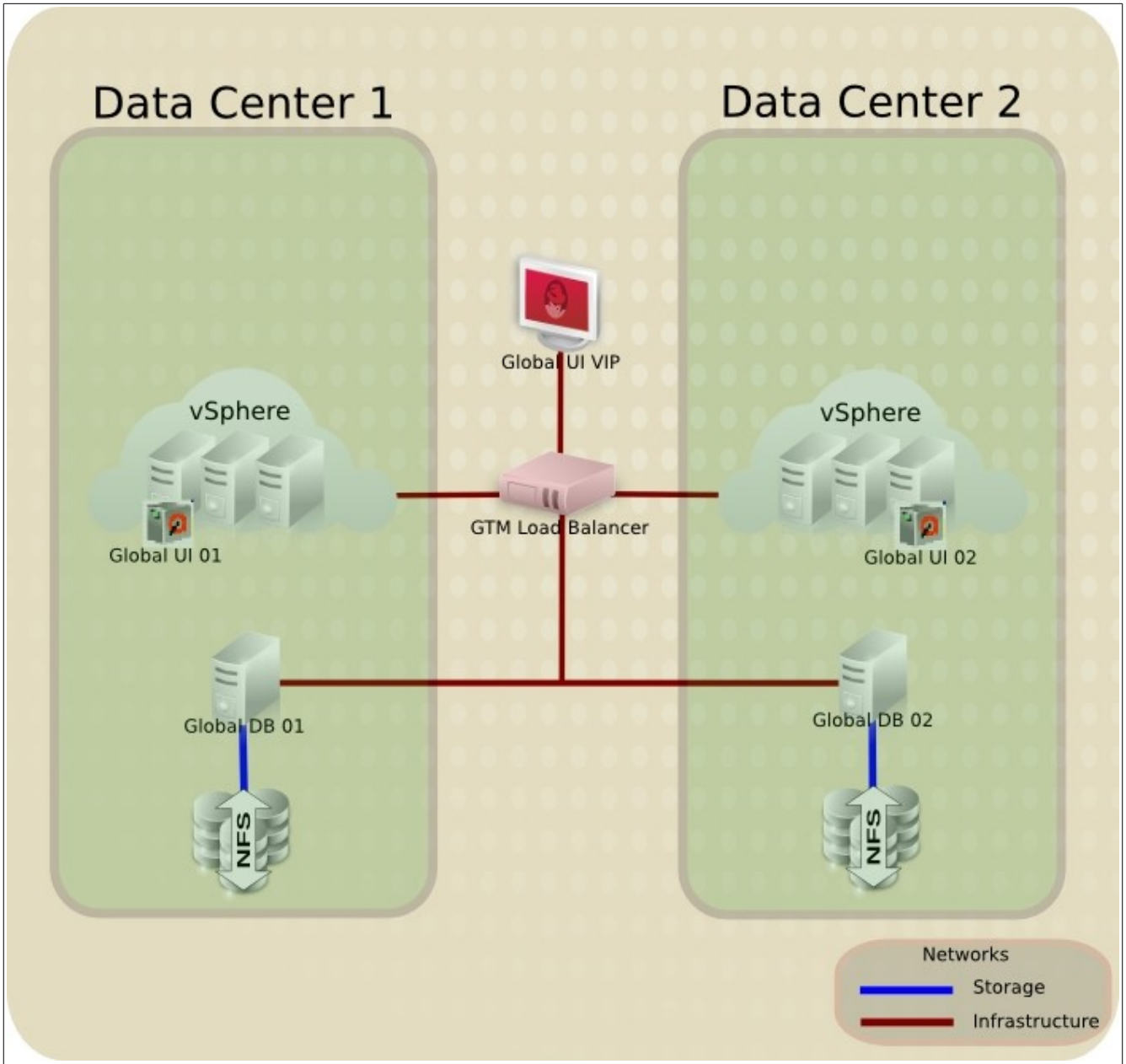


**Figure 2.2.1: Global Database Cluster**

## 2.3 Production Network Information

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

### 2.3.1 Database Systems

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

| Function | Hostname/IP | Storage Network |
|---|---|---|
| Global DB 01 | cf-global-db1.example.com<br>10.19.10.102 | cf-global-nfs1.example.com<br>10.18.10.106 |
| Global DB 02 | cf-global-db2.example.com<br>10.16.10.102 | cf-global-nfs2.example.com<br>10.17.10.106 |
| DC1 DB 01 | cf-dc1-db1.example.com<br>10.19.11.13 | cf-dc1-nfs1.example.com<br>10.18.11.20 |
| DC1 DB 02 | cf-dc1-db2.example.com<br>10.16.11.12 | cf-dc1-nfs2.example.com<br>10.17.11.19 |
| DC2 DB 01 | cf-dc2-db1.example.com<br>10.19.12.12 | cf-dc2-nfs1.example.com<br>10.18.12.19 |
| DC2 DB 02 | cf-dc2-db2.example.com<br>10.16.12.13 | cf-dc2-nfs2.example.com<br>10.17.12.20 |

***Table 2.3.1-1: Host Database Systems***

| Function | Hostname/IP | Comment |
|---|---|---|
| Global DB | cf-global-db.example.com<br>10.19.10.51 or 10.16.10.51 | DNS-based, IP address changes during failover |
| DC1 Region DB | cf-dc1-db.example.com<br>10.19.11.14 | managed by OS cluster |
| DC2 Region DB | cf-dc2-db.example.com<br>10.16.12.14 | managed by OS cluster |

***Table 2.3.1-2: Database Virtual IP's***

## 2.3.2 CloudForms User Interface Appliances

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

| Function | Hostname | IP |
|---|---|---|
| Global UI 01 | cf-global-ui1.example.com | 10.19.10.101 |
| Global UI 02 | cf-global-ui2.example.com | 10.16.10.101 |
| DC1 UI 01 | cf-dc1-ui1.example.com | 10.19.11.10 |
| DC1 UI 02 | cf-dc1-ui2.example.com | 10.19.11.11 |
| DC2 UI 01 | cf-dc2-ui1.example.com | 10.16.12.10 |
| DC2 UI 02 | cf-dc2-ui2.example.com | 10.16.12.11 |

***Table 2.3.2-1: User Interface CloudForms Appliances***

| Function | Hostname | Comment |
|---|---|---|
| Global UI | cf-global-ui.example.com<br>10.19.10.50 or 10.16.10.50 | DNS-based, IP address changes during failover |
| DC1 UI | cf-dc1-ui.example.com<br>10.19.11.52 | managed by Local Traffic Manager[3] |
| DC2 UI | cf-dc2-ui.example.com<br>10.16.12.52 | managed by Local Traffic Manager |

***Table 2.3.2-2: User Interface Virtual IP's***

3 https://f5.com/products/modules/local-traffic-manager

# 3 PostgreSQL Configuration

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

## 3.1 Database Server Setup

For the reference environment, PostgreSQL 9.2 is used for the shared external database for the CloudForms appliances, running Red Hat Enterprise Linux 6.5, with two database servers per region.

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

The first disk is assigned for the operating system and is 30 GB in size.

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

| Region | Database Disk Size |
|---|---|
| Production, Global | 825 GB |
| Production, DC1 | 700 GB |
| Production, DC2 | 400 GB |

***Table 3.1-1: CloudForms Regional Database Configuration***

**Note:** The database size relates to the number of managed VMs and hosts[4].

Below are the steps to set up the two database servers in an primary/standby configuration. These steps are performed on both systems.

- Check that the system clock is set correctly by running date, and configure NTP if necessary.
- Edit */etc/rhsm/rhsm.conf* and set the *proxy_hostname* and *proxy_port* parameters if the system needs to use a proxy to access the Internet.

Register the system with Red Hat, using the following commands:

```
#subscription-manager register
#subscription-manager list --available
#subscription-manager attach --pool=<POOL_ID with base RHEL, software
collections, and HA>
#subscription-manager repos --disable='*'
#subscription-manager repos --enable=rhel-6-server-rpms
#subscription-manager repos --enable=rhel-server-rhscl-6-rpms
#subscription-manager repos --enable=rhel-ha-for-rhel-6-server-rpms
```

**Note:** Entitlements for "Red Hat Software Collections" and the "High Availability Add On" are required.

4 https://access.redhat.com/documentation/en-US/Red_Hat_CloudForms/3.2/html/Deployment_Planning_Guide/sect-Planning.html#Database_Sizing_Assistant

Update the system, install PostgreSQL 9.2, and the cluster packages:

```
#yum update -y

#yum install -y postgresql92-postgresql-server /
postgresql92-postgresql-contrib pacemaker pcs cman
```

Configure the second disk to hold the database files:

```
#parted -s /dev/sdb mklabel msdos
#parted -s -a optimal /dev/sdb mkpart primary 0% 100%
#parted -s /dev/sdb set 1 lvm on

#pvcreate /dev/sdb1
#vgcreate vg_data /dev/sdb1
#lvcreate -n lv_pgsql_data -l 100%FREE vg_data
#mkfs.ext4 /dev/vg_data/lv_pgsql_data
```

Add the following line in */etc/fstab*:

```
/dev/vg_data/lv_pgsql_data /opt/rh/postgresql92/root/var/lib/pgsql/data /
ext4  defaults 1 2
```

Run the following commands to initialize the database directory:

```
#mount -a
#chown postgres:postgres /opt/rh/postgresql92/root/var/lib/pgsql/data
#chmod 700 /opt/rh/postgresql92/root/var/lib/pgsql/data
```

For the cluster and the database to work, the following lines need to be added in */etc/sysconfig/iptables* before the "-A INPUT -j REJECT" line:

```
-A INPUT -p udp -m state --state NEW -m udp --dport 5404 -j ACCEPT
-A INPUT -p udp -m state --state NEW -m udp --dport 5405 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 16851 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 21064 -j ACCEPT
-A INPUT -p tcp -m state --state NEW -m tcp --dport 5432 -j ACCEPT
-A INPUT -p igmp -j ACCEPT
```

> **Note:** The rules can be made more restrictive by specifying source and destination addresses. For more information, see the **Red Hat Enterprise Linux 6: Cluster Administration**[5] guide.

Activate the changes:

```
#service iptables restart
```

5 https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Cluster_Administration/s2-iptables_firewall-CA.html

## 3.1.1 Write Ahead Logs Archive

To allow the stand-by 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 | 25 GB |
| Production, DC1 and DC2 | 50 GB |

***Table 3.1.1-1: WAL Storage Configuration***

### 3.1.1.1 Common Setup

The NFS filers must be accessed through the storage network. To achieve this, a static route needs to be added.

Assuming that `eth1` is on the storage network, create */etc/sysconfig/network-scripts/route-eth1* with the following line:

```
<FILER_IP/FILER_CIDR_PREFIX> via <STORAGE_NETWORK_GATEWAY> dev eth1
```

Example:

```
10.19.11.0/21 via 10.18.11.1 dev eth1
```

Add the new route without rebooting by running:

```
ip route add 10.19.11.0/21 via 10.18.11.1 dev eth1
```

### 3.1.1.2 DC1 and DC2 Regions

For the DC1 and DC2 regional databases, 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, *UID 26* and *GID 26*, 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 DC1 and DC2 database systems, add the following line in */etc/fstab*:

```
<FILER_HOSTNAME:FILER_PATH>/wal-archive /
/opt/rh/postgresql92/root/var/lib/pgsql/wal-archive nfs defaults 0 0
```

On the DC1 and DC2 database systems, run the following commands to mount the NFS volumes:

```
#mkdir /opt/rh/postgresql92/root/var/lib/pgsql/wal-archive
```

```
#mount -a
#chkconfig netfs on
```

> **Note:** SELinux is disabled. Additional steps are necessary to make the *data* and *wal-archive* directories writable by the PostgreSQL database process if enabled.

### 3.1.1.3 Global Region

In the global region, the first database system has a writable NFS volume, and the second system has a read-only NFS volume that is a snap mirror of the other side's writable NFS volume.

On the first global database system, add the following line in */etc/fstab:*

```
<WRITABLE_FILER:WRITABLE_PATH>/wal-archive /
/opt/rh/postgresql92/root/var/lib/pgsql/wal-archive-write nfs defaults 0 0
```

Run the following commands to mount the NFS volume on the first system:

```
#mkdir /opt/rh/postgresql92/root/var/lib/pgsql/wal-archive-write
#mount -a
#chkconfig netfs on
```

On the second system, add the following line in /etc/fstab.

```
<READONLY_FILER:READONLY_PATH>/wal-archive /
/opt/rh/postgresql92/root/var/lib/pgsql/wal-archive-read nfs defaults 0 0
```

Run the following commands to mount the NFS volume on the second system:

```
#mkdir /opt/rh/postgresql92/root/var/lib/pgsql/wal-archive-read
#mount -a
#chkconfig netfs on
```

> **Note:** Depending on how the SnapMirror[6] is configured, it might take some time for the *wal-archive* directory to become available on the read-only side. Without this directory, the mount command will fail.

On the second system, run the following commands to enable archiving of WAL files when this system runs as primary. The files are not read by the other system, so they can be placed in a local directory.

```
#cd /opt/rh/postgresql92/root/var/lib/pgsql
#mkdir data/wal-archive-write
#chown postgres:postgres data/wal-archive-write
#ln -s data/wal-archive-write .
```

6 https://library.netapp.com/ecmdocs/ECMP1368826/html/GUID-97AFCAB0-CEEC-4816-9273-6D7948E6690C.html

## *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 database with the following steps. Otherwise, go to the next section.

```
#rmdir /opt/rh/postgresql92/root/var/lib/pgsql/data/lost+found
#service postgresql92-postgresql initdb
```

> **Note:** PostgreSQL's `initdb` command fails if the data directory is not empty

Copy
*/var/www/miq/system/COPY/opt/rh/postgresql92/root/var/lib/pgsql/data/EVM_V5_Dedicated_Appliance_postgresql.conf*

...from one of the CloudForms appliances to:
*/opt/rh/postgresql92/root/var/lib/pgsql/data/postgresql.conf*

...on the database system.

Make the following changes to configure logging and to facilitate the distribution of logs to a central logging server:

- logging_collector = on
- log_filename = '*postgresql.log*'
- Ensure the following directives are commented out or deleted:
  - #log_truncate_on_rotation = on
  - #log_rotation_age = 1d

Setup logrotate to rotate and compress logs. Create */etc/logrotate.d/postgres* with:

*/etc/logrotate.d/postgres*

```
/opt/rh/postgresql92/root/var/lib/pgsql/data/pg_log/postgresql.log {
  daily
  rotate 10
  copytruncate
  delaycompress
  compress
  notifempty
  missingok
  create 640 postgres postgres
}
```

## 3.2.2 Copying an Existing CloudForms Database

If CloudForms has been set up with the embedded database, the steps below can be used to move the data to an external database.

Run the following commands on the CloudForms database appliance:

```
#service evmserverd stop
#service postgresql92-postgresql stop
#chkconfig postgresql92-postgresql off
#cd /opt/rh/postgresql92/root/var/lib/pgsql
#tar cfz /tmp/pgbackup.tar.gz data
#scp /tmp/pgbackup.tar.gz ea@NODE1:/tmp
```

On the first cluster database system, run these commands:

```
#cd /opt/rh/postgresql92/root/var/lib/pgsql
#rmdir data/lost+found
#tar xf /tmp/pgbackup.tar.gz
```

Edit */opt/rh/postgresql92/root/var/lib/pgsql/data/postgresql.conf* and change the following parameters:

- shared_buffers = 1GB
- log_filename = 'postgresql-%a.log'
- log_truncate_on_rotation = on
- log_rotation_age = 1d

The username used to access the *vmdb_production* database is **root**, with the same password as the default password of the **admin** user in the Web UI.

## 3.2.3 Common Database Configuration

Add the following lines to */opt/rh/postgresql92/root/var/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:

- wal_level = hot_standby
- hot_standby = on
- archive_mode = on
- archive_command = 'cp %p /opt/rh/postgresql92/root/var/lib/pgsql/wal-archive/%f'
- max_wal_senders = 2

On the global database system, use the directory name *wal-archive-write* instead of *wal-archive* in the *archive_command* parameter.

Start PostgreSQL:

```
#service postgresql92-postgresql start
```

If creating a new database, run the following commands to create the user and database that CloudForms will use. Write down the password given to the `createuser` command. This is required when setting up the CloudForms appliances. Skip this step if the database is copied from a CloudForms database appliance.

```
#su - postgres
#scl enable postgresql92 bash
#createuser -Pd cloudforms
#createdb -E UTF8 -l en_US.UTF-8 -T template0 -O cloudforms vmdb_production
```

Create the replication user, take a database backup, and copy it to the second database system. Again, write down the password given to `createuser`. This is needed to configure the cluster database resource.

```
#su - postgres
#scl enable postgresql92 bash
#mkdir /tmp/pgbackup
#createuser -P --replication replicator
#pg_basebackup -D /tmp/pgbackup -x
#cd /tmp/pgbackup
#tar cfz /tmp/pgbackup.tar.gz .
#scp /tmp/pgbackup.tar.gz ea@NODE2:/tmp
```

On the second database system, restore the database from the backup:

```
#rm -rf /opt/rh/postgresql92/root/var/lib/pgsql/data/*
#cd /opt/rh/postgresql92/root/var/lib/pgsql/data
#tar xf /tmp/pgbackup.tar.gz
```

# 4 Regional Database Cluster

Cluster services are used to automatically manage the primary and standby databases for the DC1 and DC2 regions. For the reference environment. the Red Hat High Availability add-on for Red hat Enterprise Linux is used. Refer to **Red Hat Enterprise Linux 6 Configuring the Red Hat High Availability Add-On with Pacemaker**[7] for additional information.

Skip to the next section for the global region.

## *4.1 Setup*

Before setting up the cluster, stop the database on **node 1**:

```
#service postgresql92-postgresql stop
```

> **Note:** Verify `postgresql92-postgresql` is not running on **node 2**.

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
```

On **NODE1**:

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

On **NODE2**:

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

Start the cluster services on both nodes:

```
#service cman start
```

> **Note:** The command should show *OK* for all service components. If it fails at the *Waiting for quorum* step, check that the clusternode name attributes in */etc/cluster/cluster.conf* are correct, that the names either resolve via DNS or are defined in */etc/hosts*, and that the firewall allows cluster traffic to pass.

Start the pacemaker service and configure cman and pacemaker to start at boot time:

```
#service pacemaker start
#chkconfig cman on
#chkconfig pacemaker on
```

Verify the cluster configuration with the following commands:

```
#cman_tool nodes -a
#pcs status
```

7 https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html-single/Configuring_the_Red_Hat_High_Availability_Add-On_with_Pacemaker/index.html#s2-configurestartnodes-HAAR

**Note:** Regarding two node clusters and quorum policy in Red Hat Enterprise Linux, refer to the following Red Hat Knowledge Base article: https://access.redhat.com/solutions/645843

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 6: Configuring the Red Hat High Availability Add-On with Pacemaker**[8] 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 action="reboot" \
ipaddr="VCENTER_HOSTNAME" ssl="1" login='VCENTER_USERNAME' \
passwd='VCENTER_PASSWORD' pcmk_host_list="$NODE1,$NODE2" \
pcmk_host_check="static-list"
```

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
```

**Note:** The fencing agent may show a status of "stopped" after the cluster has been running for a while. This is due to timeouts in the fencing status check and can be ignored.

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

8 https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Configuring_the_Red_Hat_High_Availability_Add-On_with_Pacemaker/ch-fencing-HAAR.html

custom scripts on both database systems:
*/usr/local/bin/pg_ctl:*

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

scl enable postgresql92 -- pg_ctl "$@"
EOF
```

*/usr/local/bin/psql:*

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

scl enable postgresql92 -- psql "$@"
EOF
```

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

On the first system, 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=/opt/rh/postgresql92/root/var/lib/pgsql/data \
psql=/usr/local/bin/psql \
config=/opt/rh/postgresql92/root/var/lib/pgsql/data/postgresql.conf \
rep_mode=async repuser=replicator \
primary_conninfo_opt="password=REPLICATOR_PASSWORD" node_list="$NODE1 \
$NODE2" restore_command='cp \
/opt/rh/postgresql92/root/var/lib/pgsql/wal-archive/%f "%p"' \
master_ip=REPLICATION_VIP \
tmpdir=/opt/rh/postgresql92/root/var/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
```

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 /opt/rh/postgresql92/root/var/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 /opt/rh/postgresql92/root/var/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
/opt/rh/postgresql92/root/var/lib/pgsql/tmp/PGSQL.lock file to force start.
```

...delete */opt/rh/postgresql92/root/var/lib/pgsql/tmp/PGSQL.lock* and run:

```
#pcs resource cleanup postgresql
```

# 5 Global Database Configuration and Failover

The following section describes the setup and configuration for a shared global database and may contain steps that require customization for each environment. As a result, some items may be discussed in design only.

## 5.1.1 Design Considerations

Because the database systems for the global CloudForms region are in two different data centers, setting up a cluster would meet the criteria for stretch clusters as defined in **Support for Red Hat Enterprise Linux Cluster and High Availability Stretch Architectures**[9].

This is decided against for the following reasons:

Stretch clusters have two important limitations:

- No automatic failover occurs in case of a data center failure, or a network failure between the two data centers
- All cluster nodes must be on the same logical network, and routing between the two data centers may not be supported

Instead of a cluster, a Global Traffic Manager[10] (GTM) is used along with custom scripts.

There are some properties of the global databases that simplify the failover setup:

- The GTM sends all new database connections to the first system as long as the GTM can connect to port 5432 on that system
- All CloudForms appliances automatically re-open database connections in case of connection failures, and retry indefinitely
- When a data center goes down completely, there will be no write access to the global database from that data center
- When a data center loses its outside network connectivity completely, no new provisioning requests can be scheduled in that data center; consequently, a limited amount of provisioning data would be lost if appliances continue to write to the now isolated global database instance
- Database replication does not rely on shared storage that both database systems would write to simultaneously; because of this, a split brain situation does not completely destroy the database

There also are some items that increase complexity:

- If the first database system is down and comes back, new database connections are sent to the first system immediately
- While CloudForms gracefully handles database connection failures, it does not retry

9 https://access.redhat.com/articles/27136
10 https://f5.com/products/modules/global-traffic-manager

SQL queries when it encounters SQL errors; connections must therefore not be sent to the standby database
- The GTM only checks if something is listening on TCP port 5432, it does not check if the database actually works
- When a data center loses its outside network connectivity completely, database changes can still be triggered by provisioning requests that were queued just before the network outage, and by scheduled actions

Given these parameters, two PostgreSQL instances are setup. One in *Data Center 1* that will be the primary in normal operations and one in *Data Center 2* that will be the standby in normal operations along with custom scripts for automatic failover and scripts with documentation for manual failover where necessary. Refer to **Appendix E: Global Failover Scripts** for example scripts used in failover operations between datacenters.

> **Note:** Custom scripts are **not supported** by Red Hat Global Support Services. The commands called by these scripts are if they are part of a package shipped by Red Hat.

## 5.1.2 Setup

To start setting up the two global database systems, follow the same steps as for the regional databases, up to but not including the cluster setup. Do not install the `pacemaker`, `cman`, or `pcs` packages on these nodes.

Below, the global database system in *Data Center 1* is referred to as **node 1**, the global database system in *Data Center 2* is referred to as **node 2**.

Perform the following steps on both nodes.

Create */var/lib/pgsql/.pgpass* with the following content. If the database is copied from a CloudForms appliance, the *CLOUDFORMS_SQL_USER* is *root* and the password the default password. If the database was created from scratch, the username is *cloudforms* and the password is the same as set with the `createuser` command.

```
*:*:*:CLOUDFORMS_SQL_USER:SECRET
*:*:*:replicator:SECRET
```

Change the access permissions, otherwise the file will be ignored:

```
#chown postgres:postgres /var/lib/pgsql/.pgpass
#chmod 600 /var/lib/pgsql/.pgpass
```

Edit */etc/sysconfig/pgsql-replication* and set the following parameters:

```
PRIMARY_HOST: FQDN of node 1
STANDBY_HOST: FQDN of node 2
PGSQL_SQL_USER: same as CLOUDFORMS_USER above
PGSQL_SSH_USER: <a user that can be used to copy files from node 2 to node 1
via SSH; only needs write access to /tmp>
```

On **node 2**, edit */etc/sysconfig/iptables* and remove the line for port 5432. Access to this port

is controlled by a custom script.
On **node 2** only, run the following command. This will take a backup of the primary database, copy it to the standby, and start the standby.

```
#pgsql-standby-initialize
```

It also configures **node 2** to automatically start the database during the next reboot.

## 5.1.3 Operations

The following section places focus on the status, failover, and log collections for the nodes.

### 5.1.3.1 Status Check

To verify that streaming replication works, run this command on both nodes:

```
#ps -ef | grep "postgres: wal"
```

Output on the current primary:

```
#postgres: wal sender process replicator 10.6.1.252(44267) streaming
2/1DD61450
```

Output on the current standby:

```
#postgres: wal receiver process streaming 2/1DD614
```

The hexadecimal numbers at the end indicate which database change was transmitted last and should be roughly the same when the command is run at the same time on both systems.

If the processes do not exist, check the PostgreSQL log in:
*/opt/rh/postgresql92/root/var/lib/pgsql/data/pg_log*.

...messages related in the log:

```
LOG: connection received: host=10.159.0.6 port=49128
LOG: incomplete startup packet
```

...and are from the Global Traffic Manager verifying it can connect to port 5432.

### 5.1.3.2 Automatic Start at Boot on Node 1

On **node 1**, the `postgresql92-postgresql` system service is not configured to start at boot. If **node 2** is running as primary, **node 1** should not be listening on port 5432 and make the GTM send traffic to it. However, the desire is to start the database automatically if **node 1** is rebooted or comes back after a power outage, and **node 2** is only standby. To achieve this, a custom package adds a line to call */usr/sbin/pgsql-primary-check-start* in */etc/rc.local*.

The *pgsql-primary-check-start* script checks if it is on **node 1** and stops if it isn't. Otherwise, it tries to connect to the database on **node 2**, doing a simple query in the `vmdb_production` database. If this succeeds, the script knows that **node 2** is running as primary, because only then is port 5432 open, and stops. Otherwise, it starts the database on **node 1**.
*/etc/sysconfig/iptables* is configured to always allow incoming connections to port 5432.

### 5.1.3.3 Automatic Start at Boot on Node 2

On **node 2**, the `postgresql92-postgresql` system service is configured to start at boot, but port 5432 is blocked by the host-based firewall (not listed in */etc/sysconfig/iptables*). A custom package adds a line to call */usr/sbin/pgsql-standby-check-access* in */etc/rc.local*.

The *pgsql-standby-check-access* script checks if it is on **node 2** and stops if it isn't. Otherwise, it checks if the local database is running as primary and adds an iptables rule to open port 5432 if it is.

### 5.1.3.4 Automatic Failover

*/etc/cron.d/pgsql-replication* calls the script */usr/sbin/pgsql-standby-check-promote* every minute. This script does nothing when run on **node 1**. On **node 2**, it first checks if the local database is running in standby mode. If not, it stops. Otherwise, it tries to connect to the database on **node 1**, doing a simple query in the `vmdb_production` database. If that fails, a counter is incremented. If the counter has not yet reached the MAX_PRIMARY_FAILCOUNT value defined in */etc/sysconfig/pgsql-replication*, the script stops. Otherwise, **node 2** promotes itself to primary and opens the host-based firewall to allow incoming connections on port 5432.

If the GTM is unable to connect to port 5432 on **node 1**, it directs traffic to **node 2**.

The *pgsql-standby-check-promote* script logs it's output to */var/log/pgsql-standby-check-promote.log*.

To test failover, run this command on **node 1**:

```
#service postgresql92-postgresql stop
```

After a short time, */var/log/pgsql-standby-check-promote.log* on **node 2** should contain messages like:

```
psql: could not connect to server: Connection refused
 Is the server running on host "<FQDN of host>" (10.x.x.x) and accepting
 TCP/IP connections on port 5432?
Jul 24 11:11:01 INFO: <FQDN of host> is down
Jul 24 11:11:01 INFO: 2 failures reached, promoting myself to primary
server promoting
```

### 5.1.3.5 Failback from Node 2 to Node 1

After the problem on **node 1** has been fixed, the database needs to be copied from **node 2** to make **node 1** the primary again.

This is a two-step process. First, run the following command on **node 2**:

```
#pgsql-standby-release
```

This aborts all currently running database transactions, blocks port 5432, backs up the database, stops the database, and copies the backup to **node 1**, using the PGSQL_SSH_USER defined in */etc/sysconfig/pgsql-replication*.

Next, run the following command on **node 1**:

```
#pgsql-primary-initialize
```

The script first tries to connect to the database on **node 2**. If it detects that the database is running on **node 2**, the script exits. Otherwise, it overwrites the local database with the backup from **node 2** and starts the database.

### 5.1.3.6 Initializing the Standby Database

The script */usr/sbin/pgsql-standby-initialize* can be run manually on **node 2** to overwrite the local database with a backup from **node 1**, and start **node 2** in standby mode. It also enables the `postgresql92-postgresql` system service to be started automatically at boot, and blocks access to port 5432.

Before running the script, check that **node 1** is the primary and **node 2** is not (look for the `wal` processes as described above).
Stop the database on **node 2**:

```
#service postgresql92-postgresql stop
```

...and then run:

```
#pgsql-standby-initialize
```

### 5.1.3.7 Log Rotation

*/etc/logrotate.d/pgsql-replication* configures weekly log rotation for the files created by the custom scripts. Old log files are deleted after four weeks.

*/etc/cron.daily/pgsql-replication* deletes files from the WAL archive that are older than seven days. If the available space on */opt/rh/postgresql92/root/var/lib/pgsql/wal-archive-write* becomes too low, the MAX_AGE variable in this script has to be set to a lower value.

## *5.2 UI Failover Setup*

For each of the global, DC1, and DC2 regions, there are two database servers and two CloudForms appliances.

Almost all data shared between the UI instances (CloudForms appliances) are kept in the database. Outside access to the UIs is done through local and global traffic managers.

The most important information that is not stored in the database, by default, are current user session data. With the default configuration, a user would have to login again each time the load balancer sends them to a different UI.

To change this, access each UI appliance through its real hostname (not through the load balancer).
Go to *Configure* > *Configuration* and click on *Advanced* on the right side. In the text box in the *Server* section, look for `session_store: cache` and change this to `session_store: sql`. Click the *Save* button and restart the appliance via the appliance console.
All UI appliances need the *Database Operations* role to perform backups of the (external)

database.

The UI appliances in DC1 and DC2 require the *Database Synchronization* role, and require custom configuration to exclude some tables from the replication into the global database.

Go to *Configure* > *Configuration* > *Advanced* on DC1 and DC2 UI appliances.

> **Note**: The *Advanced* tab is only available when you connect directly to an appliance, not through the load balancer.

Under *replication_worker* > *replication* > *exclude_tables*, add the following lines:

```
- dialog_fields
- dialog_groups
- dialog_tabs
- dialogs

- miq_shortcuts

- services
- service_resources
- service_templates
- service_template_catalogs
```

## 5.3 Replication from DC1/DC2 to the Global Region

Some of the data from the DC1 and DC2 regions is copied to the database of the global region to allow users to see VMs from both DC1 and DC2 in one place. This requires merging of the two regional databases and to omit tables with large amounts of data that are not needed in the global region.

This replication is done on the application level. CloudForms uses rubyrep[11] to do the actual replication.

Replication is asynchronous. Whenever a database table changes in DC1 or DC2, an entry is added to the rubyrep table. These entries are periodically processed by the rubyrep worker. The number of items in this table are the "backlog" and should stay close to 0.

To check the current backlog for a region, log in to the regional UI appliance and navigate to *Configure* > *Configuration* > *Diagnostics* > *Region* > *Replication*. This will indicate if replication is active and how large the backlog is.

11 http://www.rubyrep.org/

In *evm.log*, replication can be monitored by looking for entries like:

```
MIQ(ReplicationWorker) Replication Status: Current Backlog=[0], Added=[6],
Deleted=[7]
```

Because of the high-latency connection between DC2 and DC1, customizing the rubyrep configuration to exclude additional tables from the replication is necessary. This has to be done separately for each UI appliance in the DC1 and DC2 regions. It is not possible to do this through the load balancer as the Web UI has to be accessed directly.

The tables are configured via *Configure* > *Configuration* > *Settings* > *Server Name* > *Advanced*. The *Advanced* tab is only shown for the current server.

In the edit field, under *exclude_tables*, the following tables are added:

- advanced_settings
- dialog_fields
- dialog_groups
- dialog_tabs
- dialogs
- services
- service_resources
- service_templates
- service_template_catalogs

*Dialogs* and *services* don't need to be replicated because they are not modified in the DC1 and DC2 regions. The *advanced_settings* table has the vCenter advanced settings for ESX hosts and virtual machines. Its content is not required in the global region and would take a long time to replicate from the DC2 UI appliances to the global appliances located in DC1.

# 6 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 master and one hot standby server.

The following use cases were successfully demonstrated:

- Configuring a highly available CloudForms implementation across two data centers with a regional database for each data center, using Red Hat cluster services to provide a highly available database for each region
- Configuring a highly available CloudForms implementation across two data centers with a single global database mirroring data between the database servers

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

# Appendix A: Revision History

| | | |
|---|---|---|
| Revision 1.0 | Monday, 10 August, 2015 | Brett Thurber |
| Initial Release | | |
| Revision 1.1 | Friday, 11 September, 2015 | Brett Thurber |
| Updates to section 4 | | |
| Revision 1.2 | Thursday, 1 October, 2015 | Brett Thurber |
| Added global failover scripts | | |

# Appendix B: Contributors

| Contributor | Title | Contribution |
|---|---|---|
| Carsten Clasohm | Principal Consultant | Content, Review |
| Brett Thurber | Principal Software Engineer | Content, Review |
| Brian Hamrick | Principal Product Manager | Content, Review |
| Jerome Marc | Solution Architect | Review |
| John Ruemker | Principal Software Maintenance Engineer | Content, Review |
| Andrew Beekhof | Principal Software Engineer | Review |
| Nenad Peric | Software Engineer | Review |

# Appendix C: Troubleshooting

The following section covers database and cluster troubleshooting steps.

## C.1 PostgreSQL Replication

The PostgreSQL log files can be found in */opt/rh/postgresql92/root/var/lib/pgsql/data/pg_log*. After starting the standby database, log entries present indicate status:

```
redo starts at 1/79022228
consistent recovery state reached at 1/79039E68
database system is ready to accept read only connections
```

Check the process list:

```
#ps -ef | grep "postgres: wal"
```

...the primary node displays:

```
#postgres: wal sender process replicator 10.6.1.252(44039) streaming
1/DE0EE408
```

...the standby node displays::

```
#postgres: wal receiver process streaming 1/DE0F8728
```

Errors like the one below mean that the WAL archive is not set up correctly. The primary already has moved WAL segment 76 from its *pg_xlog* directory to the archive, but the standby is not able to read it for some reason. Check the `restore_command` in *data/recovery.conf*, and check the content of the directory used by it on both the primary and standby.

```
#FATAL: requested WAL segment 000000010000000100000076 has already been
removed
```

To verify replication, connect to the local database on each of the two nodes:

```
#scl enable postgresql92 bash
#psql -h $HOSTNAME -U root vmdb_production
```

On the primary, create a table for testing with

```
create table t (ts timestamp);
```

On both systems, try to insert a row into the table:

```
insert into t values (current_timestamp);
```

This should succeed on the primary, and fail on the standby with:

```
#ERROR: cannot execute INSERT in a read-only transaction
```

Do a select on both systems. With streaming replication, the results should be the same on both systems:

```
select * from t;
```

When testing is complete, drop the table:

```
drop table t;
```

# C.2 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
```

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.3 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-dc1-db1.example.com:
   + master-postgresql                : 1000
   + postgresql-data-status           : LATEST
   + postgresql-master-baseline       : 0000000010000080
   + postgresql-status                : PRI
   + postgresql-xlog-loc              : 0000000010000080
* Node cf-dc1-db2.example.com:
   + master-postgresql                : 100
   + postgresql-data-status           : STREAMING|ASYNC
   + postgresql-status                : HS:async
```

In this case, **cf-dc1-db1** is the primary, and **cf-dc1-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-dc2-db1.example.com:
   + master-postgresql                : -INFINITY
   + postgresql-data-status           : DISCONNECT
   + postgresql-status                : HS:alone
* Node cf-dc2-db2.example.com:
   + master-postgresql                : 1000
   + postgresql-data-status           : LATEST
   + postgresql-master-baseline       : 0000000011000080
   + postgresql-status                : PRI
   + postgresql-xlog-loc              : 0000000011000080
```

Here, cf-dc2-db2 is the primary, and cf-dc2-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.4 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 postgresql92 -- pg_basebackup -h REPLICATION_VIP -U replicator \
-D /tmp/pgbackup -x
#rm -rf /opt/rh/postgresql92/root/var/lib/pgsql/data/*
#mv /tmp/pgbackup/* /opt/rh/postgresql92/root/var/lib/pgsql/data
#chown -R postgres:postgres /opt/rh/postgresql92/root/var/lib/pgsql/data

#pcs cluster unstandby $HOSTNAME
```

# C.5 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.6 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 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[12] tables that frequently change and clean up tables from which CloudForms frequently deletes rows.

# Appendix E: Global Failover Scripts

*pgsql-primary-archive-wal*

```
#!/bin/bash

# Archive a Write Ahead Log (WAL) file from the pg_xlog directory.

# Because we don't need the archive on the standby host, we simply
# discard the WAL files there.

# This script is NOT SUPPORTED by Red Hat Global Support Services.

set -eu

WAL_PATH="$1"
WAL_FILENAME="$2"

. /usr/libexec/pgsql-replication.sh

if [ "$HOSTNAME" != "$PRIMARY_HOST" ]; then
    log INFO "Not on $PRIMARY_HOST, exiting"
    exit 0
fi

cp "$WAL_PATH" "$WAL_ARCHIVE_WRITE/$WAL_FILENAME"
```

*pgsql-primary-check-start*

```
#!/bin/bash

# Check if we can connect to the database on the second node.
# It's either up in standby mode, up in primary mode, or down.
#
# We only start the local database if the secondary node is not in
# primary mode.
#
# This is made easier by the fact that the second node blocks
# connections to port 5432 if it is in standby mode.

# This script is NOT SUPPORTED by Red Hat Global Support Services.
```

12 http://www.postgresql.org/docs/9.4/static/sql-reindex.html

```
set -eu

. /usr/libexec/pgsql-replication.sh

exec >>/var/log/pgsql-primary-check-start.log 2>&1

if [ "$HOSTNAME" != "$PRIMARY_HOST" ]; then
    log INFO "Not on $PRIMARY_HOST, exiting"
    exit 1
fi

if [ ! -e /var/lib/pgsql/.pgpass ]; then
    log ERROR "Please create /var/lib/pgsql/.pgpass"
    exit 1
fi

if /sbin/service postgresql92-postgresql status >/dev/null; then
    log WARNING "PostgreSQL is already running locally, exiting"
    exit 1
fi

if su -l -c "scl enable postgresql92 -- psql -h $STANDBY_HOST -U
$PGSQL_SQL_USER -w -c 'select 1' vmdb_production" postgres >/dev/null; then
    log INFO "$STANDBY_HOST is running as primary, not starting local
database"
else
    log INFO "Cannot connect to $STANDBY_HOST, starting local database"

    /sbin/service postgresql92-postgresql start
fi
```

*pgsql-primary-initialize*

```
#!/bin/bash

# Using a backup that LVDC has copied here, make this server
# the primary again.

# This script is NOT SUPPORTED by Red Hat Global Support Services.

set -eu

. /usr/libexec/pgsql-replication.sh

if ! [ -e /tmp/pgbackup.tar.gz ]; then
    echo "ERROR: Cannot find database backup /tmp/pgbackup.tar.gz"
    echo "       Run pgsql-standby-release on $STANDBY_HOST first"
    exit 1
fi

if su -l -c "scl enable postgresql92 -- psql -h $STANDBY_HOST -U
$PGSQL_SQL_USER -w -c 'select 1' vmdb_production" postgres >&/dev/null; then
  echo "ERROR: $STANDBY_HOST is still running, not starting local database"
```

```
  exit 1
fi

if service postgresql92-postgresql status >/dev/null; then
  echo "Stopping database"
  service postgresql92-postgresql stop
fi

echo "Overwriting database files with backup from standby"

# We are starting from scratch, and can remove all WAL files
# that were previously archived.
rm -f $WAL_ARCHIVE_WRITE/*

# Same for the other database files.
rm -rf $PGSQL_HOME/data/*

cd $PGSQL_HOME/data
tar xfz /tmp/pgbackup.tar.gz

echo "Starting database"
service postgresql92-postgresql start
```

*pgsql-standby-check-access*

```
#!/bin/bash

# Check if this is the standby acting as primary. If so,
# open outside access to port 5432

# This script is NOT SUPPORTED by Red Hat Global Support Services.

set -eu

. /usr/libexec/pgsql-replication.sh

exec >>/var/log/pgsql-standby-check-access.log 2>&1

if [ "$HOSTNAME" != "$STANDBY_HOST" ]; then
    log INFO "Not on $STANDBY_HOST, exiting"
    exit 0
fi

if ! [ -e /opt/rh/postgresql92/root/var/lib/pgsql/data/recovery.conf ]; then
    if ! iptables -L -n | grep -q 5432; then
      log INFO "Opening port 5432"
      iptables_open
    else
      log WARNING "Port 5432 already open"
    fi
else
    log INFO "Running as standby, not opening port 5432"
fi
```

*pgsql-standby-check-promote*

```
#!/bin/bash

# Periodically check if the primary database is reachable. If not,
# promote this node to be the primary database.

# This script is NOT SUPPORTED by Red Hat Global Support Services.

set -eu

. /usr/libexec/pgsql-replication.sh

exec >>/var/log/pgsql-standby-check-promote.log 2>&1

if [ "$HOSTNAME" != "$STANDBY_HOST" ]; then
    log INFO "Not on $STANDBY_HOST, exiting"
    exit 0
fi

if [ ! -e /var/lib/pgsql/.pgpass ]; then
    log ERROR "Please create /var/lib/pgsql/.pgpass"
    exit 1
fi

if ! /sbin/service postgresql92-postgresql status >/dev/null; then
    log WARNING "PostgreSQL is not running locally, exiting."
    exit 1
fi

# Nothing to do if we are the primary already.
if ! [ -e $PGSQL_HOME/data/recovery.conf ]; then
    log INFO "I'm already the primary, nothing to do."
    exit 0
fi

# Check if we can connect to the database on the first node.
# It's either up in primary mode, or down.
#
# If it's down, we become the new primary and allow outside
# connections.
#
# To prevent short network outages from causing a failover,
# we only fail over after two consecutive failures.

if ! su -l -c "scl enable postgresql92 -- psql -h $PRIMARY_HOST -U
$PGSQL_SQL_USER -w -c 'select 1' vmdb_production" postgres >/dev/null; then
    log INFO "$PRIMARY_HOST is down"

    PRIMARY_FAILCOUNT=0
    if [ -e $PGSQL_HOME/primary-failcount ]; then
      . $PGSQL_HOME/primary-failcount
    fi

    PRIMARY_FAILCOUNT=$(( PRIMARY_FAILCOUNT + 1 ))
```

```
    if [ $PRIMARY_FAILCOUNT -lt $MAX_PRIMARY_FAILCOUNT ]; then
      log INFO "Current failure count $PRIMARY_FAILCOUNT less than threshold
$MAX_PRIMARY_FAILCOUNT, waiting"

      echo "PRIMARY_FAILCOUNT=$PRIMARY_FAILCOUNT" \
          >$PGSQL_HOME/primary-failcount
    else
      log INFO "$MAX_PRIMARY_FAILCOUNT failures reached, promoting myself to
primary"
      su -l -c  "scl enable postgresql92 -- pg_ctl -D $PGSQL_HOME/data
promote" postgres

      iptables_open

      rm $PGSQL_HOME/primary-failcount
    fi
else
    log DEBUG "Primary $PRIMARY_HOST is up, nothing to do."

    if [ -e $PGSQL_HOME/primary-failcount ]; then
      log INFO "Resetting failure count to 0"
      rm $PGSQL_HOME/primary-failcount
    fi
fi
```

*pgsql-standby-initialize*

```
#!/bin/bash

# Overwrite the local database with a copy from the primary and
# start the local database in standby mode.

# This script is NOT SUPPORTED by Red Hat Global Support Services.

set -eu

. /usr/libexec/pgsql-replication.sh

if [ "$HOSTNAME" != "$STANDBY_HOST" ]; then
    log INFO "Not on $STANDBY_HOST, exiting"
    exit 0
fi

if service postgresql92-postgresql status >/dev/null; then
    echo "The local database is still running." >&2
    echo "Please verify that $PRIMARY_HOST is" >&2
    echo "the current primary, and then stop the local database." >&2
    exit 1
fi

if ! su -l -c "scl enable postgresql92 -- psql -h $PRIMARY_HOST -U
$PGSQL_SQL_USER -w -c 'select 1' vmdb_production" postgres >/dev/null; then
    echo "ERROR: Cannot connect to $PRIMARY_HOST" >&2
    exit 1
```

```
fi

echo "Overwriting database files with backup from primary"

# We are starting from scratch, and can remove all WAL files
# that were previously archived.
rm -f $WAL_ARCHIVE_WRITE/*

# Same for the other database files.
rm -rf $PGSQL_HOME/data/*

su -l -c \
  "scl enable postgresql92 -- pg_basebackup -h $PRIMARY_HOST -U replicator
-w -D $PGSQL_HOME/data -x" \
  postgres

cat >$PGSQL_HOME/data/recovery.conf <<EOF
standby_mode = 'on'
primary_conninfo = 'host=$PRIMARY_HOST user=replicator'
restore_command = 'cp $WAL_ARCHIVE_READ/%f "%p"'
recovery_target_timeline = 'latest'
EOF

chown postgres:postgres $PGSQL_HOME/data/recovery.conf

if iptables -L -n | grep -q 5432; then
    echo "Closing port 5432"
    iptables_close
fi

echo "Starting and enabling database"
service postgresql92-postgresql start
chkconfig postgresql92-postgresql on
```

*pgsql-standby-release*

```
#!/bin/bash

# Take a backup of the database and stop it. This is the first step
# in making QDC the primary again after LVDC has been running
# as primary.

# This script is NOT SUPPORTED by Red Hat Global Support Services.

set -eu

. /usr/libexec/pgsql-replication.sh

rm -rf $BACKUP_DIR
mkdir $BACKUP_DIR
chown postgres:postgres $BACKUP_DIR

# Close all current transactions and prevent new connections from coming in.
# Otherwise, some data would be missing from the backup that we are going to
```

```
# copy to QDC.
if iptables -L -n | grep -q 5432; then
  echo "Blocking incoming connections on port 5432"
  iptables_close
fi

echo "Restarting PostgreSQL to close open transactions"
service postgresql92-postgresql restart

echo "Creating database backup"
su -l -c \
  "scl enable postgresql92 -- pg_basebackup -w -D $BACKUP_DIR -x" \
  postgres

echo "Shutting down and disabling database"
service postgresql92-postgresql stop
chkconfig postgresql92-postgresql off

cd /tmp/pgbackup
tar cfz /tmp/pgbackup.tar.gz .
cd ..
rm -rf /tmp/pgbackup

echo "Copying database to ${PGSQL_SSH_USER}@${PRIMARY_HOST}"
scp /tmp/pgbackup.tar.gz ${PGSQL_SSH_USER}@${PRIMARY_HOST}:/tmp
rm /tmp/pgbackup.tar.gz
echo "Now, run pgsql-primary-initialize on $PRIMARY_HOST"
```