Geographic Redundancy Configuration

Database Migration Utilities

The database migration utilities can be used to migrate a customer from Active/Standby Geographic Redundancy (GR) environment to Active/Active Geographic Redundancy environment. Currently, the migration utilities support doing remote database lookup based on NetworkId (i.e. MSISDN, IMSI, and so on). The user needs to split the SPR and balance databases from Active/Standby GR model i.e. one for each site in Active/Active GR model.

The workflow for splitting the databases is as follows:

- Dump the mongoDB data from active site of active/standby system using `mongodump` command.
- Run the Split Script, on page 2 on SPR and balance database files collected using `mongodump` command.

- Restore the mongo database for each site with `mongorestore` command using files collected from running Split Script, on page 2.

After the database splitting is done, you can audit the data by running the Audit Script, on page 4 on each set of site-specific database files separately.

The Split Script, on page 2 is a python script to split SPR and balance database into two site specific parts. The file `split.csv` is the input file which should have the Network Id regex strings for each site. The Audit Script, on page 4 is a tool to do auditing on the split database files to check for any missing/orphaned records.

To extract the database migration utility, execute the following command:

```
tar -zxvf /mnt/iso/app/install/xxx.tar.gz -C /tmp/release-train-directory
```

where, *xxx* is the release train version.

This command will extract release train into `/tmp/release-train-directory`.

Split Script

The split script first splits the SPR database into two site-specific SPR databases based on the *network_id_key* field. Then it loops through the balance database to check which site each balance record correlates to based on the *subscriberId* field and puts the balance record into one of two site-specific balance databases. If there is no match, then it is considered as `Orphaned balance record` and added to `nositebal.json`.

Here are the usage details of the split script:

Usage

```
python split.py split.csv > output.txt
```

Prerequisite

The prerequisite to run the script is `python-pymongo` module. To install python-pymongo on CPS VMs, run the command `yum install python-pymongo`.

System Requirements

- RAM: Minimum 1 GB of free memory. The script is memory-intensive and it needs at least 1 GB of RAM to work smoothly.
- vCPUs: Minimum 4 vCPUs. The script is CPU intensive.
- Persistent Storage: Free storage is required which should be at least as much as the Active/Standby database file sizes. SSD storage type is preferred (for faster runtimes) but not required.

Input Files

- The command line argument `split.csv` is a CSV file that will have network ID regex strings listed per site. The format of each line is site-name, one or more comma-separated regex strings. The regex format is python regex.

  Here is an example of a `split.csv` file where the networkId regex strings are in the MSISDN Prefix format (i.e. "Starts With" type in Policy Builder configuration).

  site1,5699[0-9],5697[0-9]*,5695[0-9]*,5693[0-9]*,5691[0-9]**

*site2,569[86420][0-9]**

Here is another example where the networkId strings are in the suffix format (i.e. "Ends With" type in Policy Builder configuration).

site1,^.[0-4]$*

site2,^.[5-9]$*

Important Since this is a CSV file, using "," in regex strings would result in unexpected behavior, so avoid using "," in regex strings.

- The script looks for the file `subscriber.bson` and one or more `account.bson` files in current directory. The `account.bson` files could be in nested folders to support a sharded balance database. The balance database could be compressed or uncompressed (the script does not look into the compressed fields).

Output Files

- `site1-balance-mgmt_account.bson`
- `site1_spr_subscriber.bson`
- `site2-balance-mgmt_account.bson`
- `site2_spr_subscriber.bson`

In addition there will be following error/debug output files:

- `errorbal.json`
- `errorspr.json`
- `nositebal.json`
- `nositespr.json`

Here is the output from a sample run of the split script.

```
$ time python split.py split.csv > output.txt
real8m44.015s
user8m0.236s
sys0m35.270s

$ more output.txt
Found the following subscriber file
./spr/spr/subscriber.bson
Found the following balance files
./balance_mgmt/balance_mgmt/account.bson
./balance_mgmt_1/balance_mgmt_1/account.bson
./balance_mgmt_2/balance_mgmt_2/account.bson
./balance_mgmt_3/balance_mgmt_3/account.bson
./balance_mgmt_4/balance_mgmt_4/account.bson
./balance_mgmt_5/balance_mgmt_5/account.bson
Site1 regex strings: 5699[0-9]*|5697[0-9]*|5695[0-9]*|5693[0-9]*|5691[0-9]*
Site2 regex strings: 569[86420][0-9]*

Started processing subscriber file
```

….

….

<snip>

Audit Script

The audit script first goes through the balance database and retrieves a list of IDs. Then it loops through each record in SPR database and tries to match the *network_id_key* or *_id* with the ID list from balance database. If there is no match, they are tagged with the counter for `Subscribers missing balance records`.

Here are the usage details for the audit script:

Usage

```
python audit.py > output.txt
```

Prerequisite

The prerequisite to run the script is `python-pymongo` module. To install python-pymongo on CPS VMs, run the command `yum install python-pymongo`.

System Requirements

- RAM: Minimum 1 GB of free memory. The script is memory-intensive and it needs at least 1 GB of RAM to work smoothly.
- vCPUs: Minimum 4 vCPUs. The script is CPU intensive.

Input Files

The script looks for the file `subscriber.bson` and one or more `account.bson` files in current directory. The `account.bson` files could be in nested folders to support a sharded balance database. The balance database could be compressed or uncompressed (the script does not look into the compressed fields).

Output Files

`sprbalmissing.bson`

Sample console output from script before splitting SPR and balance databases.

```
Total subscriber exceptions: 0
Total subscriber errors: 0
Total subscriber empty records: 1
Total subscriber records: 6743644
Total subscriber matched records: 6733102
Total subscriber missing records: 10541
```

After running the script on site-specific databases after the split, the user gets the following:

Site1:

```
Total subscriber exceptions: 0
Total subscriber errors: 0
Total subscriber empty records: 1
Total subscriber records: 4137817
```

```
Total subscriber matched records: 4131978
Total subscriber missing records: 5839
```

Site2:

```
Total subscriber exceptions: 0
Total subscriber errors: 0
Total subscriber empty records: 1
Total subscriber records: 2605826
Total subscriber matched records: 2601124
Total subscriber missing records: 4702
```

Recovery Procedures

This section covers the following recovery cases in Geographic Redundancy environment:

- Site recovery after entire site fails.
- Individual virtual machines recovery.
- Databases and replica set members recovery.

Site Recovery Procedures

Manual Recovery

When a site fails, it is assumed that other (secondary or standby) site is now operational and has become primary.

Here are the steps to recover the failed site manually:

Step 1 Confirm that the databases are in primary/secondary state on running site.

Step 2 Reset the member priorities of failed site so that when the site recovers, these members do not become primary.

a) Log on to current primary member and reset the priorities by executing the following commands:

Note To modify priorities, you must update the `members` array in the replica configuration object. The array index begins with 0. The array index value is different than the value of the replica set member's `members[n]._id` field in the array.

```
ssh <current primary replica set member>
mongo --port <port>
conf=rs.conf()
###here, note the output and note array index value of members for which we want to reset the
priorities.
#Assuming that array index value of members of failed members are 1 and 2,
conf.members[1].priority=2
conf.members[2].priority=3
conf.members[3].priority=5
conf.members[4].priority=4
rs.reconfig(conf)
#Ensure the changed priorities are reflected.
exit
```

Step 3 Re-configure gateways to make sure that no traffic is sent to failed site during recovery stage.

Step 4 Power on the VMs in the following sequence:

a) Cluster Manager
b) pcrfclients

Stop the following two scripts using monit on both pcrfclients to avoid automatic switchover of databases or automatic stopping of load balancer processes:

```
mon_db_for_call_model
```

```
mon_db_for_lb_failover
```

c) Session managers
d) Policy Server (QNS)
e) Load balancers

Step 5 Synchronize the timestamps between the sites for all VMs by executing the following command from pcrfclient01 of current secondary (recovering) site:

```
/var/qps/bin/support/sync_times.sh gr
```

Important The script should be executed only when policy director (lbs) time has been synced (NTP).

Step 6 Confirm that the databases on the failed site completely recovers and become secondary. If they do not become secondary, refer to Database Replica Members Recovery Procedures, on page 8.

Step 7 After the databases are confirmed as recovered and are secondary, reset these database's priorities using `set_priority.sh` script from Cluster Manager so that they become primary.

Step 8 If possible, run sample calls and test if recovered site is fully functional or not.

Step 9 Reconfigure the gateways to send the traffic to recovered site.

Step 10 Start `mon_db_for_call_model` and `mon_db_for_lb_failover` scripts on both pcrfclients.

Step 11 Monitor the recovered site for stability.

Automatic Recovery

CPS allows you to automatically recover a failed site.

In a scenario where a member fails to recover automatically, use the procedures described in Manual Recovery, on page 5.

For VMware

For VMware (CSV based installations), execute `automated_site_recovery.py` script on a failed site. The script recovers the failed replica members that are in RECOVERING or FATAL state. The script is located on Cluster Manager at `/var/qps/bin/support/gr_mon/automated_site_recovery.py`. The script starts the QNS processes on the Load Balancer VMs, resets the priorities of the replica set, and starts the DB monitor script. However, the script does not alter the state of the VIPs.

If you provide the replica set name as an input parameter, the script recovers the failed member of that replica set.

```
python /var/qps/bin/support/gr_mon/automated_site_recovery.py --setname <setname>
```

For example, `python /var/qps/bin/support/gr_mon/automated_site_recovery.py --setname set01`

If you do not provide any input parameter to the script, the script searches for all replica members from all sets and determines if any of the replica members are in RECOVERING or FATAL state. If yes, the script recovers the members of that replica set.

You can also execute the script with `-force`. The `-force` option recovers the replica members that are in RECOVERING, FATAL and STARTUP/STARTUP2 state as well. The script starts the QNS processes on the Load Balancer VMs in the course of recovering the DB Member. However, the script does not alter the state of the VIPs. The `-force` option must only be used when any database replica member does not come out of STARTUP/STARTUP2 state automatically.

For example: `python /var/qps/bin/support/gr_mon/automated_site_recovery.py --setname set01 --force`

During recovering of a failed site, if some replica set members do not recover, then such errors are logged in the log file located at `/var/log/broadhop/scripts/automated_site_recovery.log`.

For Open Stack

The following APIs are used to trigger a recovery script for a failed site.

The logs are located at `/var/log/orchestration-api-server.log` on the Cluster Manager VM.

/api/site/recover/start

This API is used to trigger a recovery script for a failed site. The API must only be used during a planned maintenance phase. Cluster and database processes may get reset during this process and traffic is affected. This API must only be used when the cluster is in a failed state.

- **Endpoint and Resource:** http://<*Cluster Manager IP*>:8458/api/site/recover/start

  Note If HTTPS is enabled, the Endpoint and Resource URL changes from HTTP to HTTPS. For more information, see chapter Installation in *CPS Installation Guide for OpenStack*.

- **Header:** Content-Type: application/json
- **Method:** POST
- **Payload:** YAML with force and setName fields

  force: true/false

  setName: All replica sets or specific replica set
- **Response:** 200 OK: success; 400 Bad Request: The input parameters are malformed or invalid.
- **Example:**

```
"force": "true "
"setName": "set01"
```

/api/system

This API is used to view the status of a recovery process.

- **Endpoint and Resource:** http://<*Cluster Manager IP*>:8458/api/system

Note If HTTPS is enabled, the Endpoint and Resource URL changes from HTTP to HTTPS. For more information, see chapter Installation in *CPS Installation Guide for OpenStack*.

- **Header:** Content-Type: application/json
- **Method:** GET
- **Payload:** No payload
- **Response:** 200 OK: success; 500: Script config not found
- **Example:**

  Recovery is currently underway

  ```
  "state":"recovering"
  ```

  or

  Problem with the recovery

  ```
  "state":"error_recovering"
  ```

Individual Virtual Machines Recovery

During recovery, the CPS VMs should come UP automatically on reboot without intervention. However, there are scenarios when the VMs will not recover, may be they are unstable or have become corrupt.

The following options exist to recover these CPS VMs:

- Reinitialize a VM — If a VM is stable but configurations are modified or have become corrupt and one is willing to restore the VM (reset all configurations, else, the configurations can be corrected manually). In that case, execute `/etc/init.d/vm-init-client` from the VM. Note that if the IP addresses are changed, this command would not recover the same.
- Redeploy a VM — If current VM is not recoverable, the operator may run the command `deploy.sh <vm-name>` from the cluster manager. This command will recreate the VM with latest saved configurations.

Database Replica Members Recovery Procedures

CPS database replica members can be recovered automatically or manually.

Automatic Recovery

A replica member holds a copy of the operations log (oplog) that is usually in synchronization with the oplog of the primary member. When the failed member recovers, it starts syncing from previous instance of oplog and recovers. If the member is session_cache whose data is on `/tmpfs` and if it is recovering from a reboot, the data and oplog has been lost. Therefore, the member resynchronizes all the data from primary's data files first, and then from the primary's oplog.

Verification: Execute `diagnostics.sh` and verify REPLICA STATE and LAST SYNC status.

- If REPLICA STATE does not come up as SECONDARY, and stuck into RECOVERING state for longer duration, then follow Manual Recovery, on page 11. (Refer Verification Step 1, on page 9)
- Also, if REPLICA STATE comes up as SECONDARY but does not catch up with PRIMARY and also you can see that replica lag is increasing (in LAST SYNC). (Refer Verification Step 2, on page 9)

Verification Step 1

Execute `diagnostics.sh` script to verify if all the members are healthy.

```
diagnostics.sh --get_replica_status <sitename>

CPS Diagnostics GR Multi-Node Environment
--------------------------
Checking replica sets...
|-------------------------------------------------------------------------------------------------------|
| Mongo:3.2.10                          MONGODB REPLICA-SETS STATUS INFORMATION OF site1
           Date : 2017-02-20 16:35:30 |
|-------------------------------------------------------------------------------------------------------|
|  SET NAME  -  PORT  : IP ADDRESS       - REPLICA STATE -           HOST NAME
 -  HEALTH  -   LAST SYNC   - PRIORITY |
|-------------------------------------------------------------------------------------------------------|
| BALANCE:set10
                                        |
|  Member-1  -  27718 : 172.20.18.54     -  ARBITER      -  arbiter-site3
 - ON-LINE  -   --------    -    0     |
|  Member-2  -  27718 : 172.20.17.40     -  RECOVERING   -  sessionmgr01-site1
 - ON-LINE  -   10 sec      -    2     |
|  Member-3  -  27718 : 172.20.17.38     -  RECOVERING   -  sessionmgr02-site1
 - ON-LINE  -   10 sec      -    3     |
|  Member-4  -  27718 : 172.20.19.29     -  PRIMARY      -  sessionmgr01-site2
 - ON-LINE  -    --------   -    5     |
|  Member-5  -  27718 : 172.20.19.27     -  SECONDARY    -  sessionmgr02-site2
 - ON-LINE  -    0 sec      -    4     |
|-------------------------------------------------------------------------------------------------------|
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Verification Step 2

1. Execute `diagnostics.sh` script to verify if all the members are healthy.

```
diagnostics.sh --get_replica_status <sitename>

CPS Diagnostics GR Multi-Node Environment
--------------------------
Checking replica sets...
|-------------------------------------------------------------------------------------------------------|
| Mongo:3.2.10                          MONGODB REPLICA-SETS STATUS INFORMATION OF site1
              Date : 2017-02-20 16:35:30 |
|-------------------------------------------------------------------------------------------------------|
|  SET NAME  -  PORT  : IP ADDRESS       - REPLICA STATE -           HOST NAME
    -  HEALTH  -   LAST SYNC   - PRIORITY |
|-------------------------------------------------------------------------------------------------------|
```

```
| BALANCE:set10                                                                                                    |
|  Member-1  -  27718 : 172.20.18.54     -  ARBITER      -  arbiter-site3                                          |
    - ON-LINE  -   --------    -    0     |
|  Member-2  -  27718 : 172.20.17.40     -  SECONDARY    -  sessionmgr01-site1                                     |
    - ON-LINE  -    10 sec     -    2     |
|  Member-3  -  27718 : 172.20.17.38     -  SECONDARY    -  sessionmgr02-site1                                     |
    - ON-LINE  -    10 sec     -    3     |
|  Member-4  -  27718 : 172.20.19.29     -  PRIMARY      -  sessionmgr01-site2                                     |
    - ON-LINE  -    --------   -    5     |
|  Member-5  -  27718 : 172.20.19.27     -  SECONDARY    -  sessionmgr02-site2                                     |
    - ON-LINE  -    0 sec      -    4     |
|-------------------------------------------------------------------------------------------------------------------|
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

2. Execute the following command from mongo CLI (PRIMARY member) to verify if replica lag is increasing:

```
mongo sessionmgr01-site2:27718

set10:PRIMARY> rs.printSlaveReplicationInfo()
```

If it is observed that the lag is increasing, then run `rs.status` to check for any exception. Refer to Manual Recovery, on page 11 to fix this issue.

```
mongo ––host sessionmgr01-site2 port 27718

set10:PRIMARY> rs.status()
{
"set" : "set10",
"date" : ISODate("2017-02-15T07:21:35.367Z"),
"myState" : 2,
"term" : NumberLong(-1),
"heartbeatIntervalMillis" : NumberLong(2000),
"members" : [
{
"_id" : 0,
"name" : "arbiter-site3:27718",
"health" : 1,
"state" : 7,
"stateStr" : "ARBITER",
"uptime" : 28,
"lastHeartbeat" : ISODate("2017-02-15T07:21:34.612Z"),
"lastHeartbeatRecv" : ISODate("2017-02-15T07:21:34.615Z"),
"pingMs" : NumberLong(150),
"configVersion" : 2566261
},
{
"_id" : 1,
"name" : "sessionmgr01-site1:27718",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
"uptime" : 28,
```

```
"optime" : Timestamp(1487066061, 491),
"optimeDate" : ISODate("2017-02-14T09:54:21Z"),
"lastHeartbeat" : ISODate("2017-02-15T07:21:34.813Z"),
"lastHeartbeatRecv" : ISODate("2017-02-15T07:21:34.164Z"),
"pingMs" : NumberLong(0),
"lastHeartbeatMessage" : "could not find member to sync from",
"configVersion" : 2566261
},
{
"_id" : 2,
"name" : "sessionmgr02-site1:27718",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
"uptime" : 30,
"optime" : Timestamp(1487066061, 491),
"optimeDate" : ISODate("2017-02-14T09:54:21Z"),
"infoMessage" : "could not find member to sync from",
"configVersion" : 2566261,
"self" : true
},
{
"_id" : 3,
"name" : "sessionmgr01-site2:27718",
"health" : 1,
"state" : 1,
"stateStr" : "PRIMARY",
"uptime" : 28,
"optime" : Timestamp(1487145333, 99),
"optimeDate" : ISODate("2017-02-15T07:55:33Z"),
"lastHeartbeat" : ISODate("2017-02-15T07:21:34.612Z"),
"lastHeartbeatRecv" : ISODate("2017-02-15T07:21:34.603Z"),
"pingMs" : NumberLong(150),
"electionTime" : Timestamp(1487066067, 1),
"electionDate" : ISODate("2017-02-14T09:54:27Z"),
"configVersion" : 2566261
},
{
"_id" : 4,
"name" : "sessionmgr02-site2:27718",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
"uptime" : 28,
"optime" : Timestamp(1487145333, 95),
"optimeDate" : ISODate("2017-02-15T07:55:33Z"),
"lastHeartbeat" : ISODate("2017-02-15T07:21:34.613Z"),
"lastHeartbeatRecv" : ISODate("2017-02-15T07:21:34.599Z"),
"pingMs" : NumberLong(150),
"syncingTo" : "sessionmgr01-site2:27718",
"configVersion" : 2566261
}
],
"ok" : 1
}
```


In a scenario where a member fails to recover automatically, the operator should use procedures described in Manual Recovery, on page 11.

Manual Recovery

Before performing recovery steps, refer to the following guidelines:

- Perform the recovery steps in a maintenance window on any production system. These recovery steps require restarts of the application.
- If a failure impacts the system for a long period (for example, data center, power or hardware failure) the database instance must be resynchronized manually as the oplog will have rolled over. Full resynchronizations of the database are considered events that operation teams should execute during maintenance windows with personnel monitoring the status of the platform.
- In Geo Redundancy, replica sets are used for different databases. The replication happens based on oplog. The oplog is a data structure that mongo maintains internally at Primary where the data operations logs are maintained. The secondaries fetch from the oplog entries from the primary asynchronously and apply those operations on themselves to achieve synchronization. If a secondary goes down for a long time, due to the limited size of oplog, there is a chance that some of the logs in oplog will be overwritten by new entries. In that case, when secondary comes up, it is unable to synchronize with the primary as it does not see a timestamp from where it had gone down.

Therefore, manual resynchronization is required which is termed as initial-sync in MongoDB. In this scenario, mongod process is stopped on concerned secondary, all the data in data directory is deleted and mongod process is started. The secondary Session Manager first copies all the data from primary and then copies the oplog.

Note These procedures are only for manually recovering the databases. Based on the system status (all VMs down, traffic on other site, LBs down or up, all session cache down or only one down etc.), execution of some pre- and post- tests may be required. For example, if only one session manager is to be recovered, and we have primary database and traffic on current site, we must not reset the database priorities.

Similarly, if all of the CPS databases and load balancers are down, monit processes corresponding to `mon_db_for_lb_failover` and `mon_db_for_call_model` scripts should be stopped. These scripts monitor load balancers and if LB processes or LB itself are down, they make local instances of databases secondary. Also, if local databases are secondary, these scripts shut down load balancer process. All these scripts refer to corresponding configurations in `/etc/broadhop`. Also, post recovery, user can run some sample calls on recovered site to make sure that system is stable, and then finally migrate traffic back to original Primary.

This following sections provide the detailed steps to recover a MongoDB when the database replica set member does not recover by itself:

Recovery Using Repair Option

The repair option can be used when few members have not recovered due to VM reboot or abrupt VM shutdown or some other problem.

Step 1 Execute the diagnostics script (on pcrfclient01/02) to know which replica set or respective member has failed.

For Site1:

```
#diagnostics.sh --get_replica_status site1
```

For Site2:

```
#diagnostics.sh --get_replica_status site2
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Step 2 Log onto session manager VM and check if mongod process is running or not.

```
#ps -ef | grep 27720
```

Note Port number can be different.

Step 3 If process is running, then shut down the process and try to repair the database.

a) To stop the process.

```
/usr/bin/systemctl stop sessionmgr-<port#>
```

b) To repair database.

```
/usr/bin/systemctl repair sessionmgr-<port#>
```

Sometimes the repair process takes time to recover. Check the mongo log to find the status:

```
#tailf /var/log/mongodb-<port#>.log
```

c) If the repair process is completed successfully, then start mongo process.

```
/usr/bin/systemctl start sessionmgr-<port#>
```

Step 4 Execute the diagnostics script again (on pcrfclient01/02) to know if replica set member has recovered.

For Site1:

```
#diagnostics.sh --get_replica_status site1
```

For Site2:

```
#diagnostics.sh --get_replica_status site2
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Step 5 To recover other failed members, follow the recovery steps from Step 1, on page 12 to Step 4, on page 13.

If the secondary member is still in RECOVERING state, refer Recovery Using Remove/Add Members Option, on page 13.

Recovery Using Remove/Add Members Option

Remove Specific Members

Caution Before removing the particular member from the replica set, make sure that you have identified correct member.

Sometimes a member lags behind significantly due to failure or network issues, and is unable to resync. In such cases, remove that member from replica set and add it again so that it can resync from start and come up.

Step 1 Log in to Cluster Manager.

Step 2 Execute the diagnostic script to know which replica set or respective member needs to be removed.

For Site1:

```
#diagnostics.sh --get_replica_status site1
```

For Site2:

```
#diagnostics.sh --get_replica_status site2
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Step 3 Execute `build_set.sh` with the port option to remove particular member from replica set. Script prompts to enter *<VM>:<port>* where member resides.

```
cd /var/qps/bin/support/mongo/
```

For session database:

```
#./build_set.sh --session --remove-members
```

For SPR database:

```
#./build_set.sh --spr --remove-members
```

Step 4 Execute the diagnostic script again to verify if that particular member is removed.

For Site1:

```
#diagnostics.sh --get_replica_status site1
```

For Site2:

```
#diagnostics.sh --get_replica_status site2
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Add Members

To add the earlier removed members to replica set, perform the following steps:

Step 1 Log in to Cluster Manager.

Step 2 Execute the diagnostic script to know which replica set member is not in configuration or failed member.

For Site1:

```
diagnostics.sh --get_replica_status site1
```

For Site2:

```
diagnostics.sh --get_replica_status site2
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Step 3 Update `/etc/broadhop/mongoConfig.cfg` file. Execute `build_etc.sh` to accept the changes done in `/etc/broadhop/mongoConfig.cfg` file and wait for AIDO server to create the replica-set.

```
cd /var/qps/bin/support/mongo/
```

To verify the replica-set has been created, run the following command for session database:

```
#./build_set.sh --session
```

OR

```
diagnostics.sh --get_replica_status
```

To verify whether the replica-set are created, run the following command for SPR database:

```
#./build_set.sh --spr
```

OR

```
diagnostics.sh --get_replica_status
```

Step 4 Set priority using `set_priority.sh` command. The following are example commands:

```
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db session
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db spr
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db admin
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db balance
```

Step 5 Execute the diagnostic script from Cluster Manager to know if member(s) are added successfully into the replica set.

For Site1:

```
#diagnostics.sh --get_replica_status site1
```

For Site2:

```
#diagnostics.sh --get_replica_status site2
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Recovery for High TPS

When the HA/GR setup is running with high TPS and if replica members are having high latency between them then some replica members can go to RECOVERING state and will not recover from that state unless some commands are executed to recover those replica members. We can use the manual/automated recovery procedure to recover the replica members which are in RECOVERING state.

Automated Recovery

There can be three different scenarios of setup possible for recovery of replica members:

1. Case 1: Two members of replica set are in RECOVERING state
2. Case 2: With all replica members except primary are in RECOVERING state
3. Case 3: Some replica members are in RECOVERING state

Note Automation script recovers only those replica members which are in RECOVERING state.

Step 1 Before executing automated recovery script (`high_tps_db_recovery.sh` <*replica_setname*>), go to current primary member (site-1) and reset the priorities by executing the following commands:

Note To modify priorities, you must update the `members` array in the replica configuration object. The array index begins with 0. The array index value is different than the value of the replica set member's `members[n]._id` field in the array.

```
ssh <primary member>
mongo --port <port>
conf=rs.conf()
conf.members[1].priority=2
conf.members[2].priority=3
conf.members[3].priority=5
conf.members[4].priority=4
rs.reconfig(conf)
exit
```

Step 2 Execute the following command script to recover the member:

`high_tps_db_recovery.sh` <*replica_setname*>

For Example:

```
high_tps_db_recovery.sh SPR-SET1
```

Step 3 Execute `diagnostics.sh` command to check whether the RECOVERING member has recovered.

```
diagnostics.sh --get_replica_status
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

After the replica set member is recovered, the state will change to SECONDARY and all the process logs are stored in a log file.

Note If you are unable to recover the replica set member from RECOVERING state using automated recovery script, refer to Manual Recovery, on page 17.

Manual Recovery

Step 1 Before recovery, on all concerned replica-set VMs, perform the following steps:

a) Edit `sshd_config` file.

```
vi /etc/ssh/sshd_config
```

b) Add the following entry at the end of `sshd_config` file. The below value (130) should be based on number of files that we have under secondary's data directory. It should be close to the number of files there.

```
MaxStartups 130
```

c) Restart sshd service by executing the following command:

```
service sshd restart
```

d) Execute `diagnostics.sh --get_replica_status` command to know which members are down.

Based on the status of system and subject to above note, check if you need to reset member priorities. For example, if site-1 is Primary and site-2 is Secondary and if site-1 has gone down, we need to login to new Primary and reset the replica members priorities in such a way that when site-1 comes UP again, it would not become Primary automatically.

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

For this purpose, perform the following steps:

1. Go to current primary member (site-1) and reset the priorities by executing the following commands:

Note To modify priorities, you must update the `members` array in the replica configuration object. The array index begins with 0. The array index value is different than the value of the replica set member's `members[n]._id` field in the array.

```
ssh <primary member>
mongo --port <port>
conf=rs.conf()
conf.members[1].priority=2
conf.members[2].priority=3
```

```
conf.members[3].priority=5
conf.members[4].priority=4
rs.reconfig(conf)
exit
```

2. Also on the failed replica set site there are chances that monitoring scripts would have stopped the load balancers, and shifted all the databases primaries to other site. Stop the monitoring on the failed site pcrfclient01 and pcrfclient02 both by executing the following commands:

   ```
   monit stop mon_db_for_lb_failover
   monit stop mon_db_for_callmodel
   ```

   At this point the operator should maintain two consoles: one for executing commands to recover the secondary member and another to manage the source secondary. The source Secondary is the Secondary that is nearest in terms of latency from the recovering Secondary.

   Also, note down the port and data directory for these members. Typically these are the same, and only the host name will be different.

Step 2 Recover the member:

a) Go to recovering Secondary and execute the following commands:

```
ssh <recovering Secondary>
ps -eaf | grep mongo
/usr/bin/systemctl stop sessionmgr-<port>
cd <member data directory>
\rm -rf *
cp /var/qps/bin/support/gr_mon/fastcopy.sh
```

b) Go to nearest working available secondary and execute the following commands:

```
ssh <source Secondary> mongo --port <mongo port>
#lock this secondary from writes
db.fsyncLock()
exit
ps -eaf | grep mongo
cd <data directory>
tar -cvf _tmp.tar _tmp
```

Any errors can be ignored.

```
tar -cvf rollback.tar rollback
```

c) Go to recovering Secondary and execute the following commands:

```
cd <data directory>
./fastcopy.sh <nearest working secondary>
<secondary data directory>
ps -eaf | grep scp | wc -l
```

d) After the count is one, start secondary by executing the following commands:

```
tar -xvf _tmp.tar
tar -xvf rollback.tar
/usr/bin/systemctl start sessionmgr-<port>
```

e) Go to nearest secondary from where you are recovering and execute the following commands:

```
mongo --port <port>
db.fsyncUnlock()
db.printSlaveReplicationInfo()
```

Exit the database by executing `exit` command.

Monitor the lag for some time (close to 3 to 4 minutes). Initially the lag will be small, later it will increase and then decrease. This is because secondary is catching up with primary oplog and also calculating the lag. As the secondary has just restarted, it takes sometime to calculate real lag, but MongoDB shows intermittent values, hence, we see the lag initially increasing. On the other hand, the secondary is also catching up on synchronization and eventually the lag would reduce to one second or less. The member should become secondary soon.

On similar lines, recover another secondary, preferably from the just recovered one if it is closest in terms of ping connectivity.

Once all the secondaries are recovered, we need to reset the priorities and then restart the stopped load balancers.

f) Connect to primary of concerned replica set:

```
ssh <primary member>
mongo --port <port>
conf=rs.conf()
```

Based on the output, carefully identify the correct members and their ids:

Note To modify priorities, you must update the `members` array in the replica configuration object. The array index begins with 0. The array index value is different than the value of the replica set member's `members[n]._id` field in the array.

```
conf.members[1].priority=5
conf.members[2].priority=4
conf.members[3].priority=3
conf.members[3].priority=2
rs.reconfig(conf)
exit
```

Step 3 Log in to lb01 and lb02 and start monit for all qns processes.

Step 4 Check the status of qns processes on each of load balancers VMs by executing the following command:

```
monit status qnsXX
```

Step 5 Now login to pcrfclient01 and 02 and start the monit scripts that we had stopped earlier.

```
monit start mon_db_for_lb_failover
monit start mon_db_for_callmodel
```

Now reset the sshd connection on all virtual machines. Comment out the **MaxStartup** line in `/etc/ssh/sshd_conf` file and restart sshd using `service sshd restart` command.

Rebuild Replica Set

There could be a situation when all replica set members go into recovery mode and none of members are either in primary or secondary state.

Step 1 Stop CPS processes.

Step 2 Log in to Cluster Manager.

Step 3 Execute the diagnostic script to know which replica set (all members) have failed.

For Site1:

```
#diagnostics.sh --get_replica_status site1
```

For Site2:

```
#diagnostics.sh --get_replica_status site2
```

The output displays which replica set members of replica set set01 for session data are in bad shape.

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Step 4 Build session replica sets. Add the member information in `/etc/broadhop/mongoConfig.cfg` file and run `build_etc.sh` script and wait for AIDO server to create the replica-set.

To verify the replica-set has been created, run the following command:

```
# ./build_set.sh --session
```

OR

```
diagnostics.sh --get_replica_status
```

Step 5 Set priority using `set_priority.sh` command. The following are example commands:

```
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db session
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db spr
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db admin
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db balance
```

Step 6 To recover other failed set, follow the recovery steps from Step 1, on page 20 toStep 4, on page 20.

Step 7 Restart CPS.

```
restartall.sh
```

Add New Members to the Replica Set

Step 1 Log in to Cluster Manager.

Step 2 Execute the diagnostic script to know which replica-set member is not in configuration or failed member.

For Site1:

```
#diagnostics.sh --get_replica_status site1
```

For Site2:

```
#diagnostics.sh --get_replica_status site2
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Step 3 Update the member information in the `mongoConfig.cfg` file.

Example: The following is the example for adding two members in balance replica-set.

```
cd /etc/broadhop
vi mongoConfig.cfg
```

Before Update	**After Update**
[BALANCE-SET1] SETNAME=set03	[BALANCE-SET1] SETNAME=set03
ARBITER1=pcrfclient01-prim-site-1:37718	ARBITER1=pcrfclient01-prim-site-1:37718
ARBITER_DATA_PATH=/data/sessions.3	ARBITER_DATA_PATH=/data/sessions.3
PRIMARY-MEMBERS	PRIMARY-MEMBERS
MEMBER1=sessionmgr01-site-1:27718	MEMBER1=sessionmgr01-site-1:27718
SECONDARY-MEMBERS	MEMBER2=sessionmgr02-site-1:27718
MEMBER1=sessionmgr01-site-2:27718	SECONDARY-MEMBERS
DATA_PATH=/data/sessions.3	MEMBER1=sessionmgr01-site-2:27718
[BALANCE-SET1-END]	MEMBER2=sessionmgr02-site-2:27718
	DATA_PATH=/data/sessions.3
	[BALANCE-SET1-END]

Run `build_etc.sh` to accept the changes done in `/etc/broadhop/mongoConfig.cfg` file and wait for AIDO server to add the new replica-set

```
cd /var/qps/bin/support/mongo/
```

Example: To verify the replica-set members has been added to balance database, run the following command:

```
./build_set.sh --balance
```

OR

```
diagnostics.sh --get_replica_status
```

Step 4 Execute the diagnostic script to know if member/s are added successfully into the replica-set.

For Site1:

```
#diagnostics.sh --get_replica_status site1
```

For Site2:

```
#diagnostics.sh --get_replica_status site2
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Additional Session Replication Set on GR Active/Active Site

The objective of this section is to add one additional session replication set in GR Active/Active site.

The steps mentioned in this section needs to be executed from primary site Cluster Manager.

Note The steps in this section assume that the engineer performing the steps has SSH and VMware vCenter access to the production PCRF System. No impact to traffic is foreseen during the implementation of the steps although there might be a slight impact on response time during rebalance CLI.

Before you begin

You should run all the sanity checks prior and after executing the steps in this section.

Step 1 Run `diagnostics.sh` to verify that the system is in healthy state.

Step 2 Log in to primary Cluster Manager using SSH.

Step 3 Take the backup of `/etc/broadhop/mongoConfig.cfg` file.

```
cp /etc/broadhop/mongoConfig.cfg /etc/broadhop/mongoConfig.cfg.date.BACKUP
```

Step 4 Take the backup of admin database from Cluster Manager.

```
[root@cm-a ~]# mkdir admin
[root@cm-a ~]# cd admin
[root@cm-a admin]# mongodump -h sessionmgr01 --port 27721
connected to: sessionmgr01:27721
2016-09-23T16:31:13.962-0300 all dbs
** Truncated output **
```

Step 5 Edit `/etc/broadhop/mongoConfig.cfg` file using `vi` editor. Find the section for session replication set. Add the new session replication set members.

Note Server name and ports are specific to each customer deployment. Make sure that new session replication set has unique values.

Session set number must be incremented.

Make sure the port used in MEMBER1, MEMBER2, MEMBER3, MEMBER4, and so on are same.

```
#SITE1_START
[SESSION-SET2]
SETNAME=set10
OPLOG_SIZE=1024
```

```
ARBITER1=pcrfclient01a:27727
ARBITER_DATA_PATH=/var/data/sessions.1/set10
MEMBER1=sessionmgr01a:27727
MEMBER2=sessionmgr02a:27727
MEMBER3=sessionmgr01b:27727
MEMBER4=sessionmgr02b:27727
DATA_PATH=/var/data/sessions.1/set10
[SESSION-SET2-END]

#SITE2_START
[SESSION-SET5]
SETNAME=set11
OPLOG_SIZE=1024
ARBITER1=pcrfclient01b:47727
ARBITER_DATA_PATH=/var/data/sessions.1/set11
MEMBER1=sessionmgr01b:37727
MEMBER2=sessionmgr02b:37727
MEMBER3=sessionmgr01a:37727
MEMBER4=sessionmgr02a:37727
DATA_PATH=/var/data/sessions.1/set11
[SESSION-SET5-END]
```

Run `build_etc.sh` to accept the changes done in `mongoConfig.cfg` file and wait for AIDO server to create the additonal replica-set.

Note Verify that the `/etc/hosts` file on the both sites is correctly configured with alias.

Site1 `/etc/hosts` file should have the following content:

```
x.x.x.a sessionmgr01 sessionmgr01a
x.x.x.b sessionmgr02 sessionmgr02a
y.y.y.a psessionmgr01 sessionmgr01b
y.y.y.b psessionmgr02 sessionmgr02b
```

Site2 `/etc/hosts` file should have the following content:

```
y.y.y.a sessionmgr01 sessionmgr01b
y.y.y.b sessionmgr02 sessionmgr02b
x.x.x.a psessionmgr01 sessionmgr01a
x.x.x.b psessionmgr02 sessionmgr02a
```

Step 6 SSH to Cluster-A/Site1 Cluster Manager. Add the new session replication set information in `/etc/broadhop/mongoConfig.cfg` file. Run `build_etc.sh` to accept the changes and create new session replication set from Cluster Manager.

To verify the replica-set has been created, run the following command:

```
build_set.sh --session
```

OR

```
diagnostics.sh --get_replica_status
```

Step 7 Set priority using `set_priority.sh` command. The following are example commands:

```
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db session
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db spr
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db admin
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db balance
```

Step 8 Add shard to Cluster-B/Site2. Add the new session replication set information in `/etc/broadhop/mongoConfig.cfg` file. Run `build_etc.sh` to accept the changes and create new session replication set from Cluster Manager.

To verify the replica-set has been created, run the following command:

```
build_set.sh --session
```

OR

```
diagnostics.sh --get_replica_status
```

Step 9 Set priority using `set_priority.sh` command. The following are example commands:

```
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db session
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db spr
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db admin
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db balance
```

Step 10 Copy `mongoConfig.cfg` file to all the nodes using `copytoall.sh` from Cluster Manager.

```
copytoall.sh /etc/broadhop/mongoConfig.cfg /etc/broadhop/mongoConfig.cfg


Copying '/var/qps/config/mobile/etc/broadhop/mongoConfig.cfg'
to '/etc/broadhop/mongoConfig.cfg' on all VMs
lb01
mongoConfig.cfg

      100% 4659     4.6KB/s   00:00
lb02
mongoConfig.cfg

      100% 4659     4.6KB/s   00:00
sessionmgr01
** Truncated output **
```

Step 11 Transfer the modified `mongoConfig.cfg` file to Site2 (Cluster-B).

```
scp /etc/broadhop/mongoConfig.cfg cm-b:/etc/broadhop/mongoConfig.cfg


root@cm-b's password:
mongoConfig.cfg

      100% 4659     100% 4659     4.6KB/s   00:00
```

Step 12 SSH Cluster-B (Cluster Manager). Run `build_etc.sh` to make sure modified `mongoConfig.cfg` file is restored after the reboot.

```
/var/qps/install/current/scripts/build/build_etc.sh


Building /etc/broadhop...
Copying to /var/qps/images/etc.tar.gz...
Creating MD5 Checksum...
```

Step 13 Copy `mongoConfig.cfg` file from Cluster-B (Cluster Manager) to all the nodes using `copytoall.sh` from Cluster Manager.

```
copytoall.sh /etc/broadhop/mongoConfig.cfg /etc/broadhop/mongoConfig.cfg
```

```
Copying '/var/qps/config/mobile/etc/broadhop/mongoConfig.cfg'
to '/etc/broadhop/mongoConfig.cfg' on all VMs
lb01
mongoConfig.cfg

      100% 4659     100% 4659     4.6KB/s   00:00
lb02
mongoConfig.cfg

      100% 4659     100% 4659     4.6KB/s   00:00
** Truncated output **
```

Step 14 (Applicable for HA and Active/Standby GR only) Adding shards default option. Login to OSGi mode and add the shards as follows:

```
telnet qns01 9091
```

```
Trying XXX.XXX.XXX.XXX...
Connected to qns01.
Escape character is '^]'.


addshard seed1[,seed2] port db-index siteid [backup]

osgi> addshard sessionmgr01,sessionmgr02 27727 1 Site1

osgi> addshard sessionmgr01,sessionmgr02 27727 2 Site1

osgi> addshard sessionmgr01,sessionmgr02 27727 3 Site1

osgi> addshard sessionmgr01,sessionmgr02 27727 4 Site1

osgi> addshard sessionmgr01,sessionmgr02 37727 1 Site1

osgi> addshard sessionmgr01,sessionmgr02 37727 2 Site1

osgi> addshard sessionmgr01,sessionmgr02 37727 3 Site1

osgi> addshard sessionmgr01,sessionmgr02 37727 4 Site1

osgi> rebalance

osgi> migrate
```

```
Migrate ...
All versions up to date - migration starting
```

Step 15 Verify that the sessions have been created in the newly created replication set and are balanced.

```
session_cache_ops.sh --count site2

session_cache_ops.sh --count site1
```

Sample output:

```
Session cache operation script
Thu Jul 28 16:55:21 EDT 2016
-----------------------------------------------------
Session Replica-set SESSION-SET4
-----------------------------------------------------
Session Database          : Session Count
-----------------------------------------------------
 session_cache            : 1765
 session_cache_2          : 1777
```

```
 session_cache_3          : 1755
 session_cache_4          : 1750
-------------------------------------------------------
 No of Sessions in SET4   : 7047
-------------------------------------------------------

-------------------------------------------------------
Session Replica-set SESSION-SET5
-------------------------------------------------------
Session Database          : Session Count
-------------------------------------------------------
 session_cache            : 1772
 session_cache_2          : 1811
 session_cache_3          : 1738
 session_cache_4          : 1714
-------------------------------------------------------
 No of Sessions in SET5   : 7035
-------------------------------------------------------
```

Step 16 (Applicable for Active/Active GR only) Add shards with Site option. Login to OSGi mode and add the shards as follows:

Note This process is adding shards for Active/Active GR setup which has Site options enabled. To enable Geo-HA feature, refer to Active/Active Geo HA - Multi-Session Cache Port Support, on page 66.

```
telnet qns01 9091
Trying XXX.XXX.XXX.XXX...
Connected to qns01.
Escape character is '^]'.
```

Run `listsitelookup` if you are unsure about the site names. Similar information can be obtained from `/etc/broadhop/qns.conf` file (`-DGeoSiteName=Site1`).

```
osgi> listsitelookup
                  Id        PrimarySiteId       SecondarySiteId    LookupValues
                   1            Site1               Site2       pcef-gx-1.cisco.com
                   1            Site1               Site2       pcef-gy-1.cisco.com
                   2            Site2               Site1       pcef2-gx-1.cisco.com
                   2            Site2               Site1       pcef2-gy-1.cisco.com
```

Note Do not run `addshard` command on multiple sites in parallel. Wait for the command to finish on one site and then proceed to second site.

Adding shard to Site1. Run the following command from the qns of Site1:

```
osgi> addshard sessionmgr01,sessionmgr02 27727 1 Site1

osgi> addshard sessionmgr01,sessionmgr02 27727 2 Site1

osgi> addshard sessionmgr01,sessionmgr02 27727 3 Site1

osgi> addshard sessionmgr01,sessionmgr02 27727 4 Site1
```

Adding shards to Site2. Run the following command from the qns of Site2:

```
osgi> addshard sessionmgr01,sessionmgr02 37727 1 Site2

osgi> addshard sessionmgr01,sessionmgr02 37727 2 Site2

osgi> addshard sessionmgr01,sessionmgr02 37727 3 Site2
```

```
osgi> addshard sessionmgr01,sessionmgr02 37727 4 Site2
```

Run `osgi> rebalance Site1` command from Site1 qns.

Run `osgi> rebalance Site2` command from Site2 qns.

Run the following command from the Site1 qns:

```
osgi> migrate Site1
Migrate ...
All versions up to date - migration starting
```

Run the following command from the Site2 qns:

```
osgi> migrate Site2
Migrate ...
All versions up to date - migration starting
```

Step 17 Verify that the sessions have been created in the newly created replication set and are balanced.

```
session_cache_ops.sh --count site2

session_cache_ops.sh --count site1
```

Sample output:

```
Session cache operation script
Thu Jul 28 16:55:21 EDT 2016
-----------------------------------------------------
Session Replica-set SESSION-SET4
-----------------------------------------------------
Session Database          : Session Count
-----------------------------------------------------
 session_cache            : 1765
 session_cache_2          : 1777
 session_cache_3          : 1755
 session_cache_4          : 1750
-----------------------------------------------------
 No of Sessions in SET4   : 7047
-----------------------------------------------------

-----------------------------------------------------
Session Replica-set SESSION-SET5
-----------------------------------------------------
Session Database          : Session Count
-----------------------------------------------------
 session_cache            : 1772
 session_cache_2          : 1811
 session_cache_3          : 1738
 session_cache_4          : 1714
-----------------------------------------------------
 No of Sessions in SET5   : 7035
-----------------------------------------------------
```

Step 18 Secondary Key Ring Configuration: This step only applies If you are adding additional session replication set to a new session manager server. Assuming that existing setup has the secondary key rings configured for existing session Replication servers.

Refer to the section *Secondary Key Ring Configuration* in *CPS Installation Guide for VMware*.

Step 19 Configure session replication set priority from Cluster Manager.

```
cd /var/qps/bin/support/mongo/; ./set_priority.sh --db session
```

Step 20 Verify whether the replica set status and priority is set correctly by running the following command from Cluster Manager:

```
diagnostics.sh --get_replica_status
```

```
|-----------------------------------------------------------------------------------------------------------------|
| SESSION:set10
                  |
|  Member-1  -  27727 : 192.168.116.33   -  ARBITER      -  pcrfclient01a    - ON-LINE  -   --------
    -    0     |
|  Member-2  -  27727 : 192.168.116.71   -  PRIMARY      -  sessionmgr01a    - ON-LINE  -   --------
    -    5     |
|  Member-3  -  27727 : 192.168.116.24   -  SECONDARY    -  sessionmgr02a    - ON-LINE  -    0 sec
      -    4     |
|  Member-4  -  27727 : 192.168.116.70   -  SECONDARY    -  sessionmgr01b    - ON-LINE  -    0 sec
      -    3     |
|  Member-5  -  27727 : 192.168.116.39   -  SECONDARY    -  sessionmgr02b    - ON-LINE  -    0 sec
      -    2     |
|-----------------------------------------------------------------------------------------------------------------|
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Step 21 Run `diagnostics.sh` to verify whether the priority for new replication set has been configured or not.

Step 22 Add session geo tag in MongoDBs. Repeat these steps for both session replication sets.

For more information, refer to Session Query Restricted to Local Site during Failover, on page 61 for more details.

Site1 running log: This procedure only applies if customer have local site tagging enabled.

Note To modify priorities, you must update the `members` array in the replica configuration object. The array index begins with 0. The array index value is different than the value of the replica set member's `members[n]._id` field in the array.

```
mongo sessionmgr01:27727
MongoDB shell version: 2.6.3
connecting to: sessionmgr01:27727/test

set10:PRIMARY> conf = rs.conf();
{
 "_id" : "set10",
 "version" : 2,
 "members" : [
  {
   "_id" : 0,
   "host" : "pcrfclient01a:27727",
   "arbiterOnly" : true
  },
  {
   "_id" : 1,
   "host" : "sessionmgr01a:27727",
   "priority" : 5
  },
  {
   "_id" : 2,
```

```
    "host" : "sessionmgr02a:27727",
    "priority" : 4
   },
   {
    "_id" : 3,
    "host" : "sessionmgr01b:27727",
    "priority" : 3
   },
   {
    "_id" : 4,
    "host" : "sessionmgr02b:27727",
    "priority" : 2
   }
  ],
  "settings" : {
   "heartbeatTimeoutSecs" : 1
  }
}
set10:PRIMARY> conf.members[1].tags = { "sessionLocalGeoSiteTag": "Site1" }
{ "sessionLocalGeoSiteTag" : "Site1" }
set10:PRIMARY> conf.members[2].tags = { "sessionLocalGeoSiteTag": "Site1"}
{ "sessionLocalGeoSiteTag" : "Site1" }
set10:PRIMARY> conf.members[3].tags = { "sessionLocalGeoSiteTag": "Site2"}
{ "sessionLocalGeoSiteTag" : "Site2" }
set10:PRIMARY> conf.members[4].tags = { "sessionLocalGeoSiteTag": "Site2"}
{ "sessionLocalGeoSiteTag" : "Site2" }
set10:PRIMARY> rs.reconfig(conf);
{ "ok" : 1 }
set10:PRIMARY> rs.conf();
{
  "_id" : "set10",
  "version" : 3,
  "members" : [
   {
    "_id" : 0,
    "host" : "pcrfclient01a:27727",
    "arbiterOnly" : true
   },
   {
    "_id" : 1,
    "host" : "sessionmgr01a:27727",
    "priority" : 5,
    "tags" : {
     "sessionLocalGeoSiteTag" : "Site1"
    }
   },
   {
    "_id" : 2,
    "host" : "sessionmgr02a:27727",
    "priority" : 4,
    "tags" : {
     "sessionLocalGeoSiteTag" : "Site1"
    }
   },
   {
    "_id" : 3,
    "host" : "sessionmgr01b:27727",
    "priority" : 3,
    "tags" : {
     "sessionLocalGeoSiteTag" : "Site2"
    }
   },
   {
    "_id" : 4,
```

```
   "host" : "sessionmgr02b:27727",
   "priority" : 2,
   "tags" : {
    "sessionLocalGeoSiteTag" : "Site2"
   }
  }
 ],
 "settings" : {
  "heartbeatTimeoutSecs" : 1
 }
}
set10:PRIMARY>
```

Site2 TAG configuration:

Note To modify priorities, you must update the `members` array in the replica configuration object. The array index begins with 0. The array index value is different than the value of the replica set member's `members[n]._id` field in the array.

```
mongo sessionmgr01b:37727
MongoDB shell version: 2.6.3
connecting to: sessionmgr01b:37727/test
set11:PRIMARY> conf = rs.conf();
{
 "_id" : "set11",
 "version" : 2,
 "members" : [
  {
   "_id" : 0,
   "host" : "pcrfclient01b:47727",
   "arbiterOnly" : true
  },
  {
   "_id" : 1,
   "host" : "sessionmgr01b:37727",
   "priority" : 5
  },
  {
   "_id" : 2,
   "host" : "sessionmgr02b:37727",
   "priority" : 4
  },
  {
   "_id" : 3,
   "host" : "sessionmgr01a:37727",
   "priority" : 3
  },
  {
   "_id" : 4,
   "host" : "sessionmgr02a:37727",
   "priority" : 2
  }
 ],
 "settings" : {
  "heartbeatTimeoutSecs" : 1
 }
}
set11:PRIMARY> conf.members[1].tags = { "sessionLocalGeoSiteTag": "Site2"}
{ "sessionLocalGeoSiteTag" : "Site2" }
set11:PRIMARY> conf.members[2].tags = { "sessionLocalGeoSiteTag": "Site2"}
{ "sessionLocalGeoSiteTag" : "Site2" }
set11:PRIMARY> conf.members[3].tags = { "sessionLocalGeoSiteTag": "Site1"}
```

```
{ "sessionLocalGeoSiteTag" : "Site1" }
set11:PRIMARY> conf.members[4].tags = { "sessionLocalGeoSiteTag": "Site1"}
{ "sessionLocalGeoSiteTag" : "Site1" }
set11:PRIMARY> rs.reconfig(conf);
{ "ok" : 1 }
set11:PRIMARY> rs.conf();
{
 "_id" : "set11",
 "version" : 3,
 "members" : [
  {
   "_id" : 0,
   "host" : "pcrfclient01b:47727",
   "arbiterOnly" : true
  },
  {
   "_id" : 1,
   "host" : "sessionmgr01b:37727",
   "priority" : 5,
   "tags" : {
    "sessionLocalGeoSiteTag" : "Site2"
   }
  },
  {
   "_id" : 2,
   "host" : "sessionmgr02b:37727",
   "priority" : 4,
   "tags" : {
    "sessionLocalGeoSiteTag" : "Site2"
   }
  },
  {
   "_id" : 3,
   "host" : "sessionmgr01a:37727",
   "priority" : 3,
   "tags" : {
    "sessionLocalGeoSiteTag" : "Site1"
   }
  },
  {
   "_id" : 4,
   "host" : "sessionmgr02a:37727",
   "priority" : 2,
   "tags" : {
    "sessionLocalGeoSiteTag" : "Site1"
   }
  }
 ],
 "settings" : {
  "heartbeatTimeoutSecs" : 1
 }
}
set11:PRIMARY>
```

Step 23 Run `diagnostics.sh` to verify that the system is in healthy state.

Rollback Additional Session Replication Set

Caution Removing session replication set from running Production system can impact in session loss hence it is not recommended. But if there are no other options due to any circumstances, follow these instructions.

Step 1 Run `diagnostics.sh` from OAM (pcrfclient) or Cluster Manager to verify the system is in healthy state.

Step 2 Restore ADMIN database from the backup. Restore sharding database only.

```
mongorestore --drop --objcheck --host sessionmgr01 --port 27721 --db sharding sharding
```

Step 3 Run `restartall.sh` to restart the system.

Note If it is a GR site, run `restartall.sh` on both sites before proceeding to the next step.

Step 4 Drop the newly created session replication set. In this example, remove set15.

```
build_set.sh --session --remove-replica-set --setname set15
```

Step 5 Verify sharding errors are not reported by qns nodes. Login to pcrfclient01 of Site-1 and Site-2.

```
tailf /var/log/broadhop/consolidated-qns.log
```

Ignore the following errors:

```
2016-10-05 11:45:01,446 [pool-3-thread-1] WARN  c.b.c.m.dao.impl.ShardInterface.? - Unexpected error
java.lang.NullPointerException: null
at
com.broadhop.cache.mongodb.dao.impl.ShardInterface$MonitorShards.monitorSessionTypeStatisticsCounter(ShardInterface.java:496)
 ~[com.broadhop.policy.geoha.cache_8.1.1.r090988.jar:na]
at com.broadhop.cache.mongodb.dao.impl.ShardInterface$MonitorShards.run(ShardInterface.java:407)
~[com.broadhop.policy.geoha.cache_8.1.1.r090988.jar:na]
at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511) [na:1.8.0_45]
at java.util.concurrent.FutureTask.runAndReset(FutureTask.java:308) [na:1.8.0_45]
at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$301(ScheduledThreadPoolExecutor.java:180)
 [na:1.8.0_45]
at
java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:294)
 [na:1.8.0_45]
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142) [na:1.8.0_45]
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617) [na:1.8.0_45]
at java.lang.Thread.run(Thread.java:745) [na:1.8.0_45]
```

Step 6 Run `diagnostics.sh` from OAM (pcrfclient) or Cluster Manager to verify the system is in healthy state.

Step 7 Edit `mongoConfig.cfg` file and remove the entries related to set15 from Site-1 (Cluster-A).

Step 8 Copy `mongoConfig.cfg` file to all the nodes using `copytoall.sh` script from Cluster Manager.

```
copytoall.sh /etc/broadhop/mongoConfig.cfg /etc/broadhop/mongoConfig.cfg
```

```
Copying '/var/qps/config/mobile/etc/broadhop/mongoConfig.cfg' to '/etc/broadhop/mongoConfig.cfg'
on all VMs
lb01
mongoConfig.cfg
100%  4659  4.6KB/s  00:00
lb02
mongoConfig.cfg
100%  4659  4.6KB/s  00:00
```

```
sessionmgr01
<output truncated>
```

Step 9 Transfer the modified `mongoConfig.cfg` file to Site2 (Cluster-B).

```
scp /etc/broadhop/mongoConfig.cfg cm-b:/etc/broadhop/mongoConfig.cfg
root@cm-b's password:
mongoConfig.cfg
100%  4659  4.6KB/s  00:00
[root@cm-a ~]#
```

Step 10 SSH to Cluster-B Cluster Manager. Run `build_etc.sh` to make sure modified `mongoConfig.cfg` file is restored after reboot.

```
/var/qps/install/current/scripts/build/build_etc.sh
Building /etc/broadhop...
Copying to /var/qps/images/etc.tar.gz...
Creating MD5 Checksum...
```

Step 11 Copy `mongoConfig.cfg` file from Cluster-B Cluster Manager to all the nodes using `copytoall.sh` from Cluster Manager.

```
copytoall.sh /etc/broadhop/mongoConfig.cfg /etc/broadhop/mongoConfig.cfg
```

```
Copying '/var/qps/config/mobile/etc/broadhop/mongoConfig.cfg' to '/etc/broadhop/mongoConfig.cfg'
on all VMs
lb01
mongoConfig.cfg
100%  4659  4.6KB/s  00:00
lb02
mongoConfig.cfg
100%  4659  4.6KB/s  00:00
<output truncated>
```

Network Latency Tuning Parameters

In GR, if the network latency between two sites is more than the threshold value, `-DringSocketTimeOut`, `-DshardPingerTimeoutMs` and `-balancePingerTimeoutMs` parameters need to be configured with the appropriate values.

To get the values that must be configured in `-DringSocketTimeOut`, `-DshardPingerTimeoutMs` and `-balancePingerTimeoutMs`, check the latency using ping command for the sessionmgr which hosts the shard.

Example:

If the network latency between two sites is 150 ms, the value must be configured as 50 + 150 (network latency in ms) = 200 ms.

The parameters need to be added in `/etc/broadhop/qns.conf` on both sites:

```
-DringSocketTimeOut=200
-DshardPingerTimeoutMs=200
-DbalancePingerTimeoutMs=200
```

If the parameters are not configured, default values will be considered:

```
-DringSocketTimeOut=50
```

```
-DshardPingerTimeoutMs=75
-DbalancePingerTimeoutMs=75
```

In addition to the above parameters, the following parameters need to updated if network latency is found to be 150 ms. These values need to be increased so that QNS processes can come up and get connected to Mongo Database on Session Managers without having timeouts.

```
-Dmongo.client.thread.maxWaitTime=1700
 -Dmongo.client.thread.maxWaitTime.balance=1700
 -Dmongo.client.thread.maxWaitTime.remoteBalance=1700
 -Dmongo.client.thread.maxWaitTime.remoteSpr=1700
 -Dmongo.client.thread.maxWaitTime.cdrrep=1700
 -Dmongo.client.thread.maxWaitTime.cdr=1700
```

Remote SPR Lookup based on IMSI/MSISDN Prefix

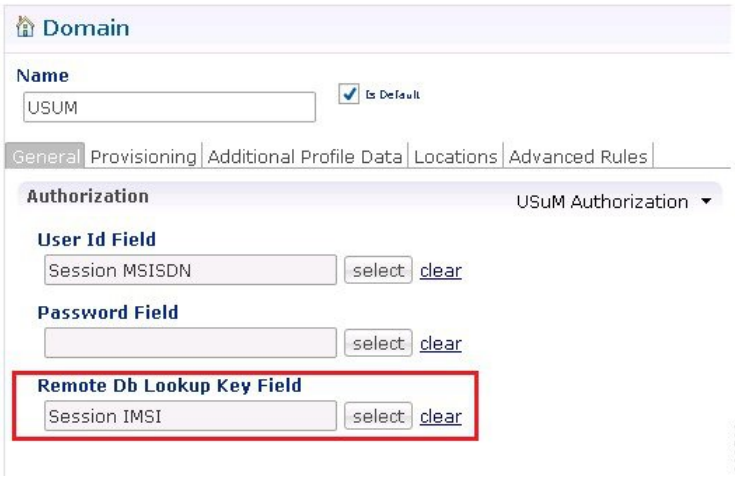
Prerequisites

Policy Builder configuration on both the sites should be the same.

Configuration

Step 1 Configure the Lookup key field in Policy Builder under 'Domain'. It can be IMSI, MSISDN, Session User Name, and so on. An example configuration is given below:

Figure 1: Remote Db Lookup Key



Step 2 Configure remote databases in Policy Builder under USuM Configuration.

Consider there are two sites: Site1 (Primary) and Site2 (Secondary). In Policy Builder there will be two clusters for Site1 and Site2 in case of Active/Active model.

Under 'Cluster-Site2', create USuM Configuration and add remote databases to be accessed when Site1 is not available.

Here is an example configuration:

Figure 2: Remote Database Configuration

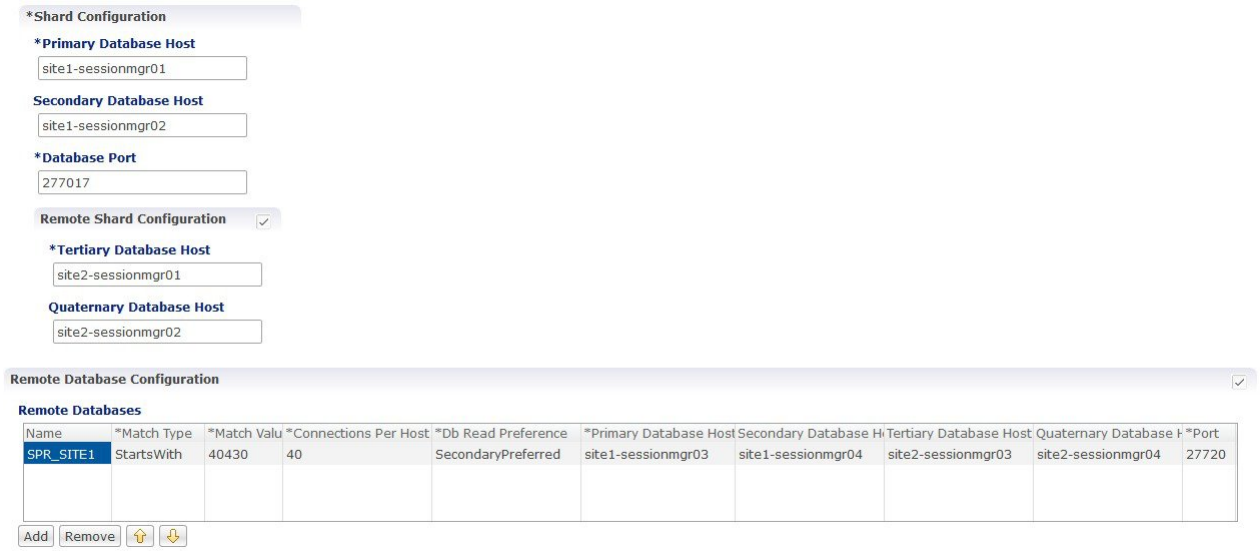


Table 1: Remote Database Configuration Parameters

Parameter	Description
Name	Unique name to identify the remote database. **Note** This is needed to see the correct sites's subscriber in Control Center, when multiple SPR is configured.
Match Type	Pattern match type. It can be Starts With, Ends With, Equals, Regex match type.
Match Value	Key/regex to be used in pattern matching.
Connection per host	Number of connections that can be created per host.
Db Read Preference	Database read preferences.
Primary/Secondary/Tertiary/Quaternary Database Host, Port	Connection parameter to access database. This should be accessible from Site2 irrespective of Site1 is UP or DOWN. **Important** The host names must exactly be the same host name used when the corresponding replica-set is created in Mongo. Only the data holding members need to be configured (and not the arbiters).

For more information on Remote Database Configuration parameters, refer to the *CPS Mobile Configuration Guide* for this release.

Remote Balance Lookup based on IMSI/MSISDN Prefix

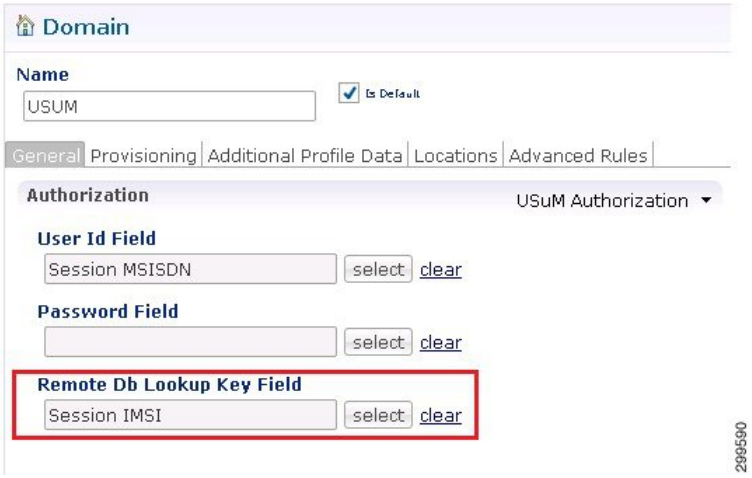
Prerequisites

Policy Builder configuration on both the sites should be the same.

Configuration

Step 1 Configure the Lookup key field in policy builder under **Domain**. It can be IMSI, MSISDN, Session User name and so on. An example configuration is given:

Figure 3: Lookup Key



Step 2 Configure remote databases in policy builder under **Balance Configuration**.

Consider there are two sites: Site1 (Primary) and Site2 (Secondary). So in Policy Builder there will be two clusters for Site1 and Site2.

Under 'Cluster-Site2', create **Balance Configuration** and add remote databases to be accessed when Site1 is not available.

An example configuration is given:

Figure 4: Example Configuration

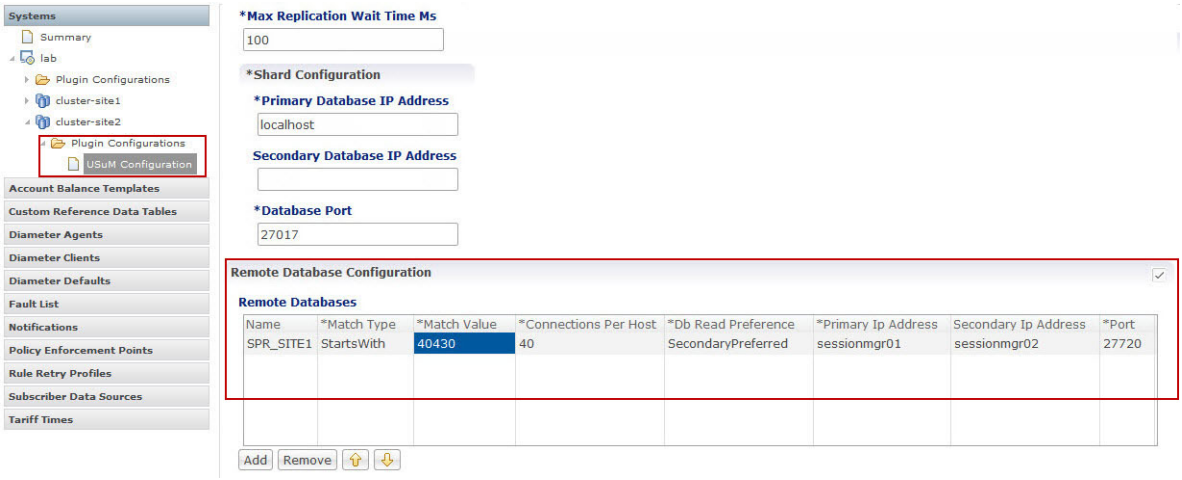


Table 2: Remote Database Configuration Parameters

Parameter	Description
Name	Unique name to identify the remote database.
Match Type	Match type to be matched for the remote database to be selected for lookup.
Connection per host	Number of connections that can be created per host.
Db Read Preference	Database read preferences.
Primary Ip Address, Secondary Ip Address, Port	Connection parameter to access database. This should be accessible from Site2 irrespective of Site1 is UP or DOWN.

For more information on **Balance Configuration** parameters, refer to the *CPS Mobile Configuration Guide* for this release.

SPR Provisioning

CPS supports multiple SPR and multiple balance databases in different ways based on deployments. SPR provisioning can be done either based on end point/listen port or using API router configuration.

SPR Location Identification based on End Point/Listen Port

Prerequisites

Policy Builder configuration on both the sites should be the same.

Configuration

Consider there are two sites: Site1 (Primary) and Site2 (Secondary).

Add new entry on Site2 in haproxy.cfg (`/etc/haproxy/haproxy.cfg`) file listening on port 8081 (or any other free port can be used) with custom header “RemoteSprDbName”. Same configuration to be done on both load balancers.

```
listen pcrf_a_proxy lbvip01:8081
        mode http
        reqadd RemoteSprDbName:\ SPR_SITE1
        balance roundrobin
        option httpclose
        option abortonclose
        option httpchk GET /ua/soap/KeepAlive
        server qns01_A qns01:8080 check inter 30s
        server qns02_A qns02:8080 check inter 30s
        server qns03_A qns03:8080 check inter 30s
        server qns04_A qns04:8080 check inter 30s
# If there are more qns add all entries here
```

Where,

RemoteSprDbName is the custom header.

SPR_SITE1 is the remote database name configured in Step 2, on page 34.

API Router Configuration

The following are the three use cases where a user must configure API router:

- Multiple SPR/Multiple Balance
- Common SPR/Multiple Balance (Hash Based)
- Common SPR Database and Multiple Balance Database based on SPR AVP

Use Cases

Multiple SPR/Multiple Balance

Use Case

This will be useful when there are multiple active sites and each has their own separate SPR and balance database.

Logic

- When API router receives the request it will extract the Network Id (MSISDN) from the request.
- It will iterate through the API router criteria table configured in **Api Router Configuration** in Policy Builder and find SPR, Balance database name by network Id.
- API router will make unified API call adding SPR, balance database name in http header.

- SPR and balance module will use this SPR, balance database name and make queries to appropriate databases.

Common SPR/Multiple Balance (Hash Based)

Use Case

This will be useful when there is common SPR across multiple sites, and latency between site is less. Also this will evenly distribute the data across all balance databases.

Logic

- When API router receives the create subscriber request:
  - It first generates hash value using network Id (in range 0 to n-1, where n is from qns.conf parameter -DmaxHash=2)
  - It will add generated hash value in SPR AVP (with code _balanceKeyHash).
  - For example, if maxHash is 2 and generated hash value is 1, then SPR AVP will be _balanceKeyHash=1
- When API router receives the request other than create subscriber.
  - It will query subscriber and get hash value from subscriber AVP (_balanceKeyHash).
- Once hash value is available, it will iterate through the API router criteria table configured in **Api Router Configuration** in Policy Builder and find balance database name by hash.
- API router will make unified API call adding balance database name in http header.
- Balance module will use this balance database name and make query to appropriate balance database.

Common SPR Database and Multiple Balance Database based on SPR AVP

Use Case

This will be useful when there is common SPR across multiple sites, but each has separate balance in each site. We can add region AVP in each subscriber to read from local database.

Logic

- When API router receives the request:
  - It will query subscriber and get subscriber AVP from subscriber. AVP name is configurable.
  - Once AVP value is available, it will iterate through the API router criteria table configured in **Api Router Configuration** in Policy Builder and find balance database name by AVP.
  - API router will make unified API call adding balance database name in http header.
- Balance module will use this balance database name and make query to appropriate balance database.

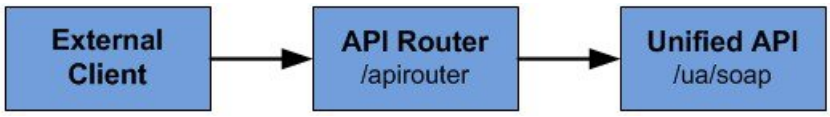
HTTP Endpoint

By default, API router is exposed on `/apirouter` and the Unified API endpoint is exposed on `/ua/soap`. The default URLs are as follows:

Table 3: Default URLs

Setup	Unified API	API Router
AIO	http://lbvip01:8080/ua/soap	http://lbvip01:8080/apirouter
HA	https://lbvip01:8443/ua/soap	https://lbvip01:8443/apirouter

Figure 5: API Router Configuration



Some customer may have configured the URL at multiple places and do not want to change URL. To change the endpoint so that the API router uses `/ua/soap`, add the following parameters in `/etc/broadhop/qns.conf`. This makes API router act as unified API.

```
-DapirouterContextPath=/ua/soap
-Dapi.ua.context.path=/ua/backend
-Dua.context.path=/ua/backend
```

Note The `api.ua.context.path` and `ua.context.path` parameters must be the same.

Figure 6: Unified API Router Configuration

New URLs are as follows:

Table 4: New URLs

Setup	Unified API	API Router
AIO	http://lbvip01:8080/ua/backend	http://lbvip01:8080/ua/soap
HA	https://lbvip01:8443/ua/backend	https://lbvip01:8443/ua/soap

Configuration

By default, API router configuration feature is not installed. To install this feature, put the following entries in feature files.

Table 5: Feature File Changes

Feature File	Feature Entry
/etc/broadhop/pcrf/feature	com.broadhop.apirouter.service.feature
/etc/broadhop/pb/feature	com.broadhop.client.feature.apirouter

Note Change in feature file requires to run `buildall.sh`, `reinit.sh` from Cluster Manager for the features to get installed.

Policy Builder Configuration

Domain

1. Remote Db lookup key field:

   This retriever fetches the value that can be used in patten match to get remote SPR, balance database name.

   This field retriever can be:

   - MSISDN retriever, IMSI retriever or whatever is networkid in subscriber to support multiple SPR, multiple balance.
   - NetworkIdHash retriever to support common SPR, multiple balance based on hashing.
   - Subscriber retriever AVP to support common SPR, multiple balance based on subscriberAVP.

API Router Configuration

1. Enable Multi-Credential Management: This check box is used to add the support for the multi-credential management for the API router to handle the following Unified API requests: CreateSubscriber, DeleteSubscriber, GetSubscriber, CreateBalance, DeleteBalance, ChangeCredentialUsername.

Note AddCredential, AddCredentials, DeleteCredential, and DeleteCredentials API requests will be added in the future to complete the multi-credential handling.

2. Enable Backup Cache Lookup For Primary Key: This check box is used to turn on/off the step to iterate over the know databases to lookup the subscriber primary key if the secondary key cache does not find the record.

Note A CreateSubscriberRequest does not look into the cache or iterate over the databases because it inserts a new record in the database and must build the cache entries from the request.

3. Filter Type: Type of filter to be used.

- NetworkId — To configure multiple SPR, multiple balance.
- NetworkIdHash — To configure common SPR, multiple balance based on hashing.
- SubscriberAVP — To configure common SPR, multiple balance based on susbcriberAVP.

  AVP Name can be changed by adding flag -DbalanceKeyAvpName=avpName. Refer to Configurable Flags, on page 43.

4. Router Criteria: The following is the list of criteria to consider for pattern matching.

Table 6: Router Criteria

Criteria	Description
Match Type	Pattern match type. It can be Starts With, Ends With, Equals, Regex match type.
Match Value	Key/regex to be used in pattern matching.
Remote SPR DB name	If criteria match, use this database name as SPR database name. This database name should match with the remote database name configured in **USuMConfiguration > Remote databases > Name**.
Remote Balance DB name	If criteria match, use this database name as Balance database name. This database name should match with the remote database name configured in **BalanceConfiguration > Remote databases > Name**.

Balance Configuration (remote databases)

The following parameters can be configured under Remote Database for Balance Configuration.

Table 7: Remote Database Parameters

Parameter	Description
Name	Balance database name.
Match Type	Pattern match type. It can be Starts With, Ends With, Equals, Regex match type.
Match Value	Key/regex to be used in pattern matching.
Connection per host	Balance database mongo connection per host.
Db Read preference	Balance database read preference.
Primary Ip Address	Balance database primary member IP address.
Secondary Ip Address	Balance database secondary member IP address.
Port	Balance database primary member IP address.

Parameter	Description
Following fields needs to be configured if hot standby for balance database is needed	
Backup DB Host	Backup balance database primary member IP address.
Backup DB Secondary Host	Backup balance database secondary member IP address.
Backup DB Port	Backup balance database primary member IP address.

USuM Configuration (remote databases)

The following parameters can be configured under Remote Database for USuM Configuration.

Important The host names must exactly be the same host name used when the corresponding replica-set is created in Mongo. Only the data holding members need to be configured (and not the arbiters).

Table 8: Remote Database Parameters

Parameter	Description
Name	SPR database name.
Match Type	Pattern match type. It can be Starts With, Ends With, Equals, Regex match type.
Match Value	Key/regex to be used in pattern matching.
Connection Per host	SPR database mongo connection per host.
Db Read preference	SPR database read preference.
Primary Database Host	SPR database primary member host name.
Secondary Database Host	SPR database secondary member host name.
Tertiary Database Host	SPR database tertiary member host name.
Quaternary Database Host	SPR database quaternary member host name.
Port	SPR database primary member port number.

Configurable Flags

The following flags are configurable in `/etc/broadhop/qns.conf` file:

Table 9: Configurable Flags

Flag	Description	Default Value
balanceKeyAvpName	Subscriber AVP name to be used for subscriber based multiple balance databases.	_balanceKey
balanceKeyHashAvpName	Internal AVP name being used for hash based multiple balance databases.	_balanceKeyHash
maxHash	Maximum value of hash to generate.	No default value
ua.context.path	Unified API context path.	/ua/soap
apirouterContextPath	API router context path.	/apirouter

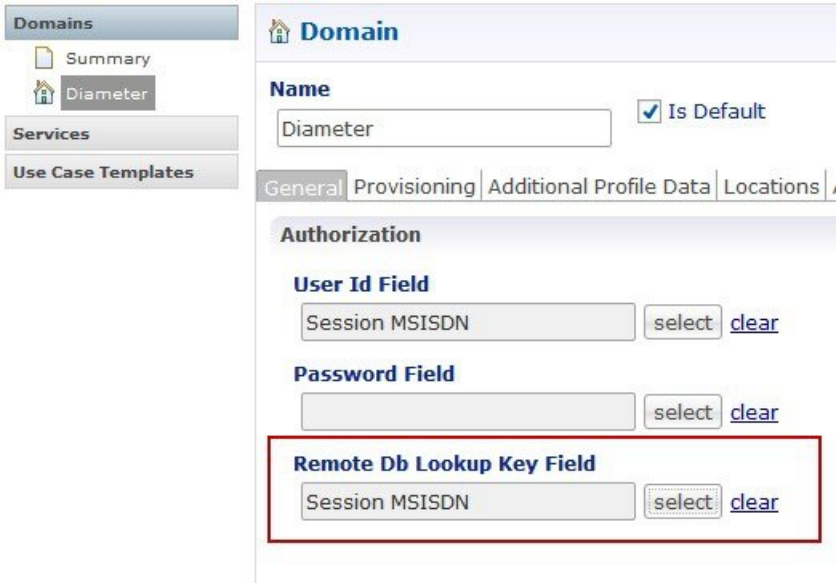
Configuration Examples

Multiple SPR/Multiple Balance

Domain Configuration

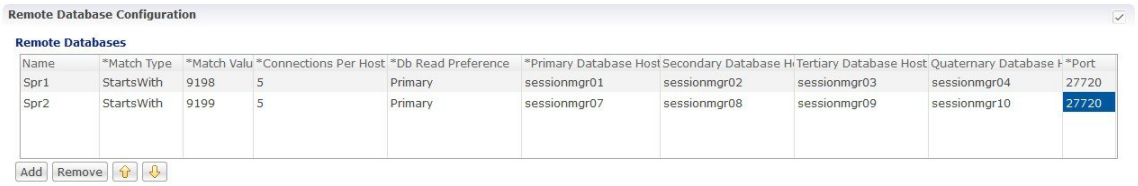
The Remote Db Lookup Key Field can be MSISDN, IMSI, and so on which is used as network ID in SPR.

Figure 7: Domain Configuration



USuM Configuration

Figure 8: USuM Configuration

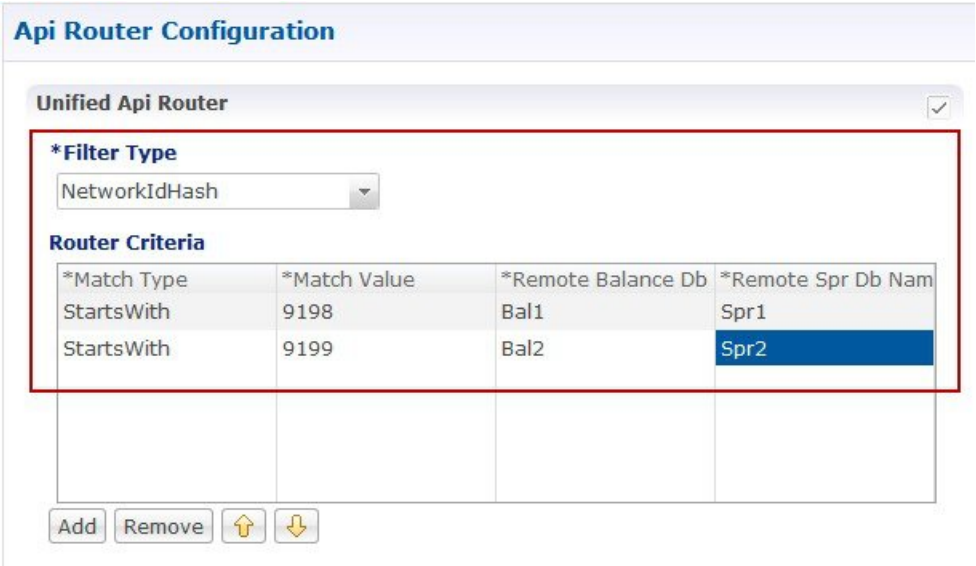


Balance Configuration

Figure 9: Balance Configuration

API Router Configuration

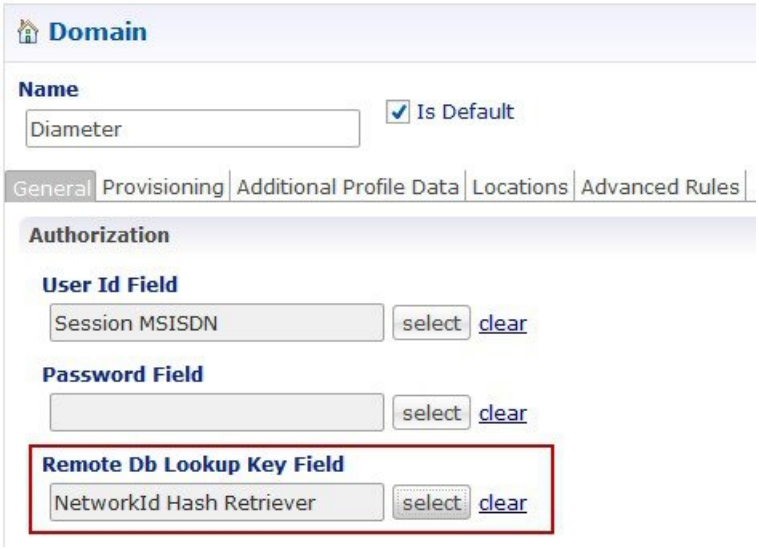
Figure 10: API Router Configuration



Common SPR/Multiple Balance (Hash Based)

Domain Configuration

Figure 11: Domain Configuration

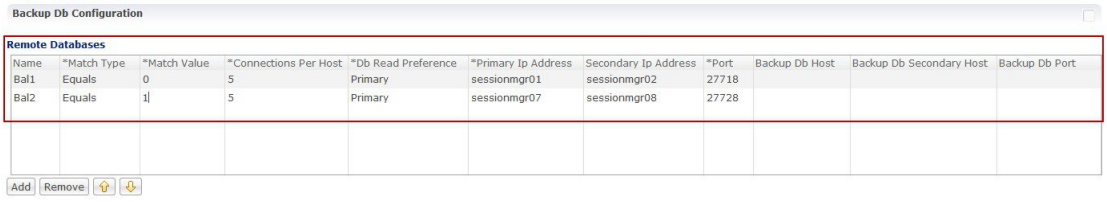


USuM Configuration

Figure 12: USuM Configuration

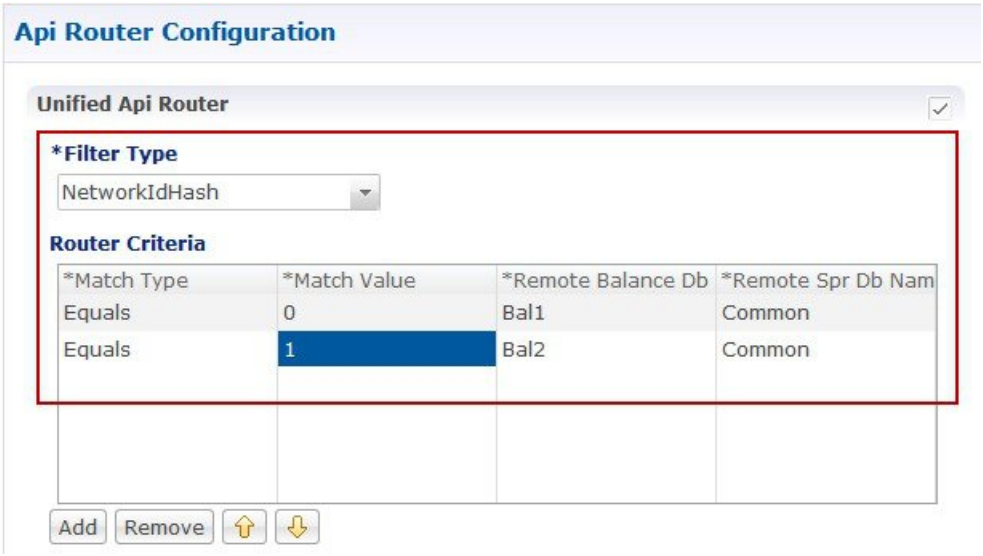
Balance Configuration

Figure 13: Balance Configuration



API Router Configuration

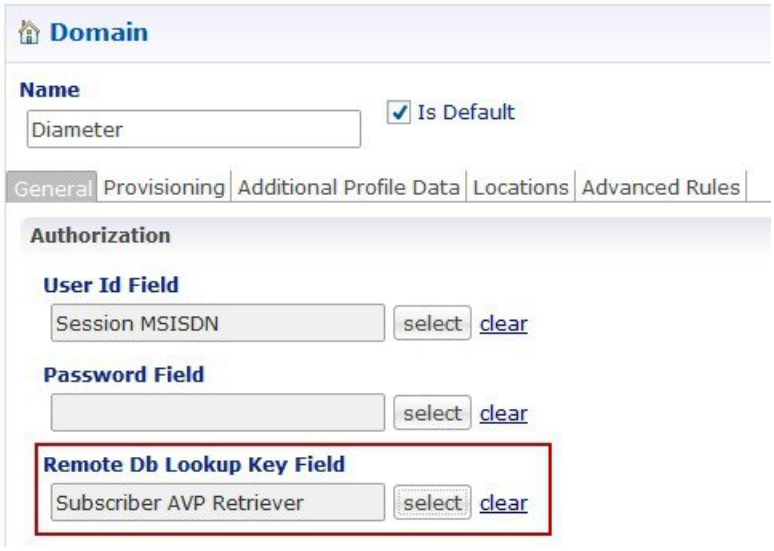
Figure 14: API Router Configuration



Common SPR Database and Multiple Balance Database based on SPR AVP

Domain Configuration

Figure 15: Domain Configuration

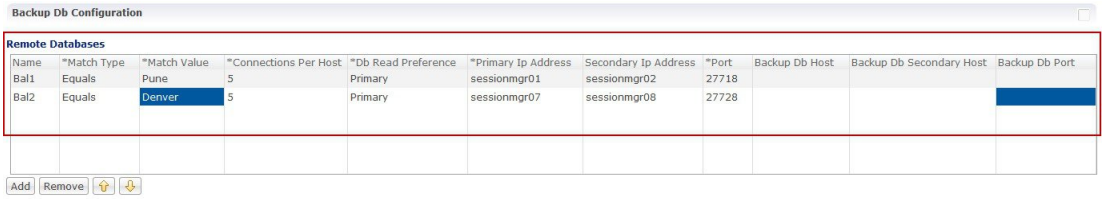


USuM Configuration

Figure 16: USuM Configuration

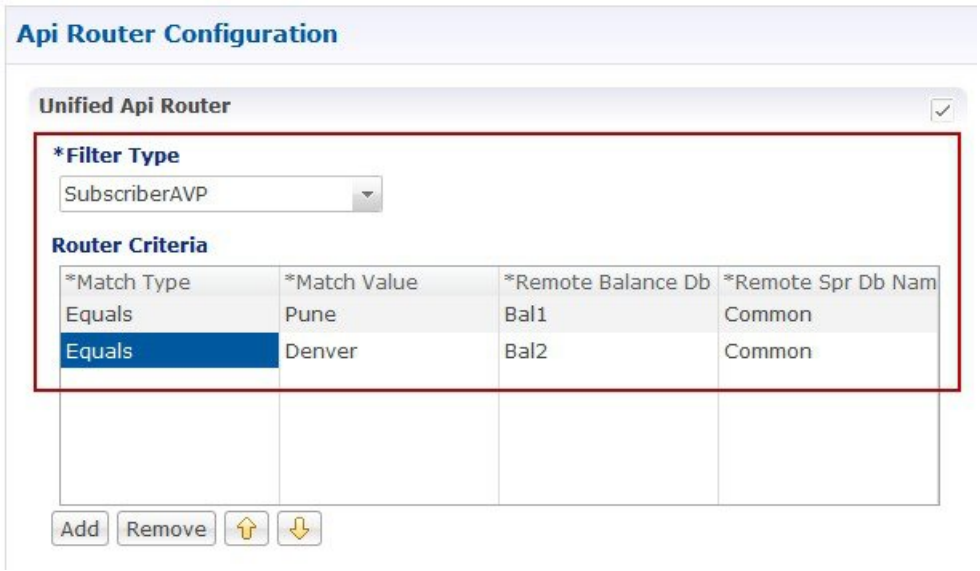
Balance Configuration

Figure 17: Balance Configuration



API Router Configuration

Figure 18: API Router Configuration



Rebalance

For hash-based balance and common SPR, rebalance is supported. It means old balance data can be rebalanced to new balance databases without need of re-provisioning.

Rebalance can be done by executing the following OSGi commands:

- `rebalanceByHash` — Rebalance with same balance shards (shards here means internal balance shards).
- `rebalanceByHash` *[oldShardCount] [newShardCount]* — Rebalance to change (increase/decrease) balance shards.

Rebalance with same balance shard

This is applicable only for hash based balance databases. To add new database to existing database, perform the following steps:

Step 1 Log in to Control Center and note down few subscriber which has balance.

Step 2 Change Policy Builder configuration: API Router, Balance, Domain, so on and publish the modified configuration.

Step 3 Add parameter `maxHash` in `qns.conf` file.

a) Value depends on number of databases.

For example, if there are two balance databases configured in Policy Builder, set value to 2.

```
-DmaxHash=2
```

Step 4 Add context path parameter `ua.context.path` and `apirouterContextPath` in `qns.conf` file. This is needed for Control Center to call via API router.

```
-DapirouterContextPath=/ua/soap
```

```
-Dua.context.path=/ua/backEnd
```

Step 5 Execute `copytoall.sh` and restart Policy Server (QNS) processes.

Step 6 Login to OSGi console on qns01.

```
telnet qns01 9091
```

Step 7 Execute `rebalanceByHash` command.

```
rebalanceByHash
```

Step 8 Log in to Control Center and verify subscriber still has balance noted in Step 1, on page 49.

Rebalance to Change Number of Balance Shards

This is applicable only for hash-based balance databases.

To increase the number of balance shards, perform the following steps:

1. Login to Control Center and note down *com.cisco.balance.dbs* few subscribers who have balance.
2. In the qns.conf file, add or edit *com.cisco.balance.dbs*.
    1. Value will be new shard number.

       Example — If you are increasing balance shards from 4 to 8, value should be set to 8.

       ```
       -Dcom.cisco.balance.dbs=8
       ```

3. Run `copytoall.sh` and restart qns processes.
4. Login to OSGi console on qns01.

   ```
   telnet qns01 9091
   ```

5. Run `rebalanceByHash` command.

   ```
   rebalanceByHash <old shard number> <new shard number>
   ```

   Example — If you are increasing balance shards from 4 to 8, old shard number is 4 and new shard number is 8.

   ```
   rebalanceByHash 4 8
   ```

6. Login to Control Center and verify subscriber still has balance noted in Step 1.

To decrease the number of balance shards, perform the following steps:

1. Login to Control Center and note down few subscribers who have balance.
2. Login to OSGi console on qns01.

   ```
   telnet qns01 9091
   ```

3. Run `rebalanceByHash` command.

   ```
   rebalanceByHash <old shard number> <new shard number>
   ```

   Example — If you are decreasing balance shards from 6 to 4, old shard number is 6 and new shard number is 4.

   ```
   rebalanceByHash 6 4
   ```

4. In the qns.conf file, add or edit *com.cisco.balance.dbs*.
   1. Value will be new shard number

      Example — If you are decreasing balance shards from 6 to 4, value should be set to 4.

      ```
      -Dcom.cisco.balance.dbs=4
      ```

5. Run `copytoall.sh` and restart qns processes.
6. Login to Control Center and verify subscriber still has balance noted in Step1.

Configurations to Handle Traffic Switchover

When Policy Server (QNS) is Down

The following configurations are needed to enable `qns_hb.sh` script. This script stops Policy Server (QNS) processes from lb01/lb02 when all Policy Server (QNS) are down (that is, qns01,02..n).

Note To understand traffic switchover, refer to Load Balancer VIP Outage.

Step 1 To enable script, add the following configuration in `/var/qps/config/deploy/csv/Configuration.csv` file:

```
mon_qns_lb,true,
```

For more information on how to add the parameter in `Configuration.csv` file, refer to *CPS Installation Guide for VMware* for 9.1.0 and later releases.

Note Any database that is not replicated across sites should not be configured for monitoring by `mon_db_for_callmodel.sh` script.

Step 2 To disable script, add the following configuration in `/var/qps/config/deploy/csv/Configuration.csv` file or remove `mon_qns_lb` tag from this CSV file:

```
mon_qns_lb,,
```

Step 3 Import CSV to JSON by executing the following command:

```
/var/qps/install/current/scripts/import/import_deploy.sh
```

Step 4 Execute the following command to validate the imported data:

```
cd /var/qps/install/current/scripts/deployer/support/

python jvalidate.py
```

Note The above script validates the parameters in the Excel/csv file against the ESX servers to make sure ESX server can support the configuration and deploy VMs.

Step 5 Reinitiate lb01 and lb02 by executing the following command:

```
/etc/init.d/vm-init
```

For more information on configuring GR features, refer to *CPS Mobile Configuration Guide*.

When Replicated (inter-site) Database is not Primary on a Site

Note To understand traffic switchover, refer to Load Balancer VIP Outage.

Step 1 Add the list of databases that needs to be monitored in `mon_db_for_callmodel.conf` file (`/etc/broadhop/mon_db_for_callmodel.conf`) in Cluster Manager.

Important Contact your Cisco Technical Representative for more details.

Add the following content in the configuration file (`mon_db_for_callmodel.conf`):

Note The following is an example and needs to be changed based on your requirement:

```
#this file contains set names that are available in mongoConfig.cfg. Add set names one below other.
#Refer to README in the scripts folder.
SESSION-SET1
```

```
SESSION-SET2
BALANCE-SET1
SPR-SET1
```

Step 2 To enable switch-off of UAPI traffic when replicated (inter-site) configured databases are not primary on this site, add `STOP_UAPI=1` in `/etc/broadhop/mon_db_for_callmodel.conf` file.

Note To disable switch-off of UAPI traffic when replicated (inter-site) configured databases are not primary on this site, add `STOP_UAPI=0` (if this parameter is not defined,) in `/etc/broadhop/mon_db_for_callmodel.conf` file.

When we recover GR site, we have to manually start UAPI interface (if it is disabled) by executing the following command as a root user on lb01 and lb02:

```
echo "enable frontend https-api" | socat stdio /tmp/haproxy
```

Step 3 To configure the percentage value for session replica sets to be monitored from the configured session replica set list, set the `PERCENTAGE_SESS_DB_FAILURE` parameter in the `/etc/broadhop/mon_db_for_callmodel.conf` configuration file.

Use an integer from **`1`** to **`100`**.

Note `PERCENTAGE_SESS_DB_FAILURE` parameter should be configured such that it results in integer value and not float for the number of sets user expects to be down to trigger the fail over. For example, if there are 4 session replica-sets and user wants failover to be triggered only when two or more replica-sets are down (>= 2 condition), user should configure this value as 50 (%). User at times wrongly interprets this parameter as "more than given number of replica-sets". For example in above case, user intends to have a failover if more than one replica-set (so, this is equivalent to "> 1" condition) are down and configures a value which is more than 25 (for example, 45) which does not work. The expected value should meet the condition of ">= desired number of replica-sets to be down for failover trigger".

Step 4 Rebuild etc directory on cluster by executing the following command:

```
/var/qps/install/current/scripts/build/build_etc.sh
```

When Virtual IP (VIP) is Down

You can configure CPS to monitor VIPs. If any of the configured VIPs goes down, the configured databases from the local site are made secondary. However, if the databases at a site go down, the VIP is not shut down.

For VMware

Specify the configuration in `/etc/broadhop/mon_db_for_lb_failover.conf`.

Note The `/etc/broadhop/mon_db_for_lb_failover.conf` file contains the configuration for VIPs along with the database set names.

For OpenStack

- During fresh install, apply the configuration using `http://<cluman-ip>:8458/api/system/config/action/`.

- Once system is deployed, use PATCH API
  `http://<cluman-ip>:8458/api/system/config/application-config` to apply the configuration that enables monitoring of VIPs.

The new configuration `vipMonitorForLb` with the following format is created under applicationConfig:

```
vipMonitorForLb:
 vipName:
 - lbvip01
 - lbvip02
```

Where, vipName is the array of VIPs to be monitored.

In case of any issues, check the API log file `/var/log/orchestration-api-server.log` and the `/var/log/broadhop/scripts` directory (after system configuration) for any errors.

Configuring Session Database Percentage Failure

You can configure the percentage value of session replica sets to be monitored from the configured session replica set list.

For VMware

Specify the configuration in `/etc/broadhop/mon_db_for_callmodel.conf` by modifying the `PERCENTAGE_SESS_DB_FAILURE` parameter. Use an integer from **`1`** to **`100`**.

For example, PERCENTAGE_SESS_DB_FAILURE=50

Note `PERCENTAGE_SESS_DB_FAILURE` parameter should be configured such that it results in integer value and not float for the number of sets user expects to be down to trigger the fail over. For example, if there are 4 session replica-sets and user wants failover to be triggered only when two or more replica-sets are down (>= 2 condition), user should configure this value as 50 (%). User at times wrongly interprets this parameter as "more than given number of replica-sets". For example in above case, user intends to have a failover if more than one replica-set (so, this is equivalent to "> 1" condition) are down and configures a value which is more than 25 (for example, 45) which does not work. The expected value should meet the condition of ">= desired number of replica-sets to be down for failover trigger".

For OpenStack

- During fresh install, apply the configuration using
  `http://<cluman-ip>:8458/api/system/config/action`.
- Once system is deployed, use PATCH API
  `http://<cluman-ip>:8458/api/system/config/application-config` to apply the configuration.

The new configuration `dbMonitorForQns` with the following format is created under applicationConfig:

```
dbMonitorForQns:
stopUapi: "false"
percentageSessDBFailure: 50
setName:
- SESSION-SET1
- SESSION-SET2
- SESSION-SET3
- SESSION-SET4
```

In case of any issues, check the API log file `/var/log/broadhop/scripts/mon_db_for_callmodel_$DATE.log`.

Remote Databases Tuning Parameters

If remote databases are configured for balance or SPR, respective mongo connection parameters are required to be added in `/etc/broadhop/qns.conf` file.

Remote SPR

```
-DdbSocketTimeout.remoteSpr=1200
-DdbConnectTimeout.remoteSpr=600
-Dmongo.connections.per.host.remoteSpr=10
-Dmongo.threads.allowed.to.wait.for.connection.remoteSpr=10
-Dmongo.client.thread.maxWaitTime.remoteSpr=1200
```

Remote Balance

```
-DdbSocketTimeout.remoteBalance=1200
-DdbConnectTimeout.remoteBalance=600
-Dmongo.connections.per.host.remoteBalance=10
-Dmongo.threads.allowed.to.wait.for.connection.remoteBalance=10
-Dmongo.client.thread.maxWaitTime.remoteBalance=1200
```

SPR Query from Standby Restricted to Local Site only (Geo Aware Query)

Step 1 Add new entry for Site1 and Site 2 in `/etc/broadhop/pcrf/qns.conf` file on pcrfclient01.

```
-DsprLocalGeoSiteTag=Site1 ====> in Site1 qns.conf file

-DsprLocalGeoSiteTag=Site2 ====> in Site2 qns.conf file
```

Step 2 Execute `syncconfig.sh`. To reflect the above change, CPS needs to be restarted.

Step 3 Add tag to SPR MongoDBs.

a) Run `diagnostics.sh` command on pcrfclient01 and find SPR database primary member and port number.

```
diagnostics.sh --get_replica_status
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

b) Login to SPR database using the primary replica set hostname and port number.

For example, `mongo --host sessionmgr01 --port 27720`

c) Get the replica members by executing the following command:

Execute `rs.conf()` from any one member.

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:37720",
                        "arbiterOnly" : true
                },
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27720",
                        "priority" : 2
                },
                {
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27720",
                        "priority" : 2
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr05-site1:27720",
                        "priority" : 2
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr06-site1:27720",
                        "votes" : 0,
                        "priority" : 2
                },
                {
                        "_id" : 5,
                        "host" : "sessionmgr01-site2:27720"
                },
                {
                        "_id" : 6,
                        "host" : "sessionmgr02-site2:27720"
                },
                {
                        "_id" : 7,
                        "host" : "sessionmgr05-site2:27720"
                },
                {
                        "_id" : 8,
                        "host" : "sessionmgr06-site2:27720",
                        "votes" : 0
                }
        ],
        "settings" : {
                "heartbeatTimeoutSecs" : 1
        }
}
```

d) Now from the list, find out the members of Site1 and Site2 to be tagged (excluding the arbiter). After finding member, execute the following command from Primary member to tag members.

Note To modify priorities, you must update the `members` array in the replica configuration object. The array index begins with 0. The array index value is different than the value of the replica set member's `members[n]._id` field in the array.

```
conf = rs.conf();
conf.members[1].tags = { "sprLocalGeoSiteTag": "Site1" }
conf.members[2].tags = { "sprLocalGeoSiteTag": "Site1"}
conf.members[3].tags = { "sprLocalGeoSiteTag": "Site1"}
conf.members[4].tags = { "sprLocalGeoSiteTag": "Site1"}
conf.members[5].tags = { "sprLocalGeoSiteTag": "Site2"}
conf.members[6].tags = { "sprLocalGeoSiteTag": "Site2"}
conf.members[7].tags = { "sprLocalGeoSiteTag": "Site2"}
conf.members[8].tags = { "sprLocalGeoSiteTag": "Site2"}
rs.reconfig(conf);
```

Note This is a sample output. Configuration, members and tag can be different as per your environment.

`conf.members[1]` means member with *_id = 1* in output of `rs.conf()`.

After executing this command, primary member can be changed.

Verify tags are properly set by log in on any member and executing the following command:

```
rs.conf();
```

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:37720",
                        "arbiterOnly" : true
                },
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27720",
                        "priority" : 2,
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27720",
                        "priority" : 2,
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr05-site1:27720",
                        "priority" : 2,
                        "tags" : {
                                "sprLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr06-site1:27720",
                        "votes" : 0,
                        "priority" : 2,
                        "tags" : {
```

```
                        "sprLocalGeoSiteTag" : "Site1"
                }
        },
        {
                "_id" : 5,
                "host" : "sessionmgr01-site2:27720",
                "tags" : {
                        "sprLocalGeoSiteTag" : "Site2"
                }
        },
        {
                "_id" : 6,
                "host" : "sessionmgr02-site2:27720",
                "tags" : {
                        "sprLocalGeoSiteTag" : "Site2"
                }
        },
        {
                "_id" : 7,
                "host" : "sessionmgr05-site2:27720",
                "tags" : {
                        "sprLocalGeoSiteTag" : "Site2"
                }
        },
        {
                "_id" : 8,
                "host" : "sessionmgr06-site2:27720",
                "votes" : 0,
                "tags" : {
                        "sprLocalGeoSiteTag" : "Site2"
                }
        }
    ],
    "settings" : {
            "heartbeatTimeoutSecs" : 1
    }
}
```

Step 4 Repeat Step 3, on page 54 for all other sites. Tag names should be unique for each site.

This change overrides the read preference configured in **USuM Configuration** in Policy Builder.

Step 5 Execute `rs.reconfig()` command to make the changes persistent across replica-sets.

Balance Location Identification based on End Point/Listen Port

Prerequisites

Policy Builder configuration on both the sites should be the same.

Configuration

Consider there are two sites: Site1 (Primary) and Site2 (Secondary).

Add new entry on Site2 in `haproxy.cfg` (`/etc/haproxy/haproxy.cfg`) file listening on port 8081 (or any other free port can be used) with custom header *RemoteBalanceDbName*. Same configuration needs to be done on both load balancers.

```
listen pcrf_a_proxy lbvip01:8081
mode http
reqadd RemoteBalanceDbName:\ BAL_SITE1
balance roundrobin
option httpclose
option abortonclose
option httpchk GET /ua/soap/KeepAlive
server qns01_A qns01:8080 check inter 30s
server qns02_A qns02:8080 check inter 30s
server qns03_A qns03:8080 check inter 30s
server qns04_A qns04:8080 check inter 30s
# If there are more qns add all entries here
```

where,

RemoteBalanceDbName is the custom header.

BAL_SITE1 is the remote database name configured in Remote Balance Lookup based on IMSI/MSISDN Prefix, on page 36.

Figure 19: Balance Site

Balance Query Restricted to Local Site

The following steps need to be performed for balance query from restricted to local site only (Geo aware query) during the database failover:

Consider there are two sites: Site1 and Site2

Note If there are more than one balance databases, follow the below mentioned steps for all the databases.

The following steps are not needed to be performed for backup or hot standby databases.

Step 1 Add new entry in Site1 `qns.conf` file (`/etc/broadhop/qns.conf`) on Cluster Manager.

```
-DbalanceLocalGeoSiteTag=Site1
```

a) Run `copytoall.sh` to restart the qns processes.

Step 2 Add new entry on Site2 `qns.conf` file (`/etc/broadhop/qns.conf`) on Cluster Manager.

```
-DbalanceLocalGeoSiteTag=Site2
```

a) Run `copytoall.sh` to restart the qns processes.

Step 3 Add balance geo tag in MongoDBs.

a) Run `diagnostics.sh` command on pcrfclient01 and find balance database primary member and port number.

```
$ diagnostics.sh --get_replica_status
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

b) Login to balance database using the primary replica set hostname and port number.

For example, `$ mongo --host sessionmgr01 --port 27720`

c) Get the balance database replica members information.

Execute `rs.conf()` from any one member.

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:27718",
                        "arbiterOnly" : true
                },
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27718",
                        "priority" : 4
   },
                {
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27718",
                        "priority" : 3
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr01-site2:27718",
                        "priority" : 2
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr02-site2:27718"
                        "priority" : 1
                }
        ],
        "settings" : {
                "heartbeatTimeoutSecs" : 1
        }
}
```

d) Now from the list, find out the members of Site1 and Site2 to be tagged (excluding the arbiter).

e) After finding member, execute the following command from primary member to tag members.

```
conf = rs.conf();
conf.members[1].tags = { "balanceLocalGeoSiteTag": "Site1" }
conf.members[2].tags = { "balanceLocalGeoSiteTag": "Site1"}
conf.members[3].tags = { "balanceLocalGeoSiteTag": "Site2"}
conf.members[4].tags = { "balanceLocalGeoSiteTag": "Site2"}
rs.reconfig(conf);
```

Note This is a sample configuration. Members and tag can be different according to your deployment.

`conf.members[1]` means member with *_id = 1* in output of `rs.conf()`.

After tagging the members, primary member may get changed if all the members have same priority.

To verify that the tags are properly set, log in to any member and execute `rs.conf()` command.

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:27718",
                        "arbiterOnly" : true
                },
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27718",
                        "priority" : 4,
                        "tags" : {
                                "balanceLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27718",
                        "priority" : 3,
                        "tags" : {
                                "balanceLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr01-site2:27718",
                        "priority" : 2,
                        "tags" : {
                                "balanceLocalGeoSiteTag" : "Site2"
                        }
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr01-site2:27718",
                        "priority" : 1,
                        "tags" : {
                                "balanceLocalGeoSiteTag" : "Site2"
                        }
                }
        ],
        "settings" : {
                "heartbeatTimeoutSecs" : 1
```

```
        }
}
```

Session Query Restricted to Local Site during Failover

The following steps need to be performed for session query from restricted to local site only (Geo aware query) during the database failover:

Consider there are two sites: Site1 and Site2

Note If there are more than one session databases, follow the below mentioned steps for all the databases.

The following steps are not needed to be performed for backup or hot standby databases.

This geo tagging is applicable only during **database failover time period**. In normal case, session database query/update always happen on primary member.

Step 1 Add new entry in Site1 `qns.conf` file (`/etc/broadhop/qns.conf`) on Cluster Manager.

```
-DsessionLocalGeoSiteTag=Site1
```

a) Run `copytoall.sh` to restart the qns processes.

Step 2 Add new entry on Site2 `qns.conf` file (`/etc/broadhop/qns.conf`) on Cluster Manager.

```
-DsessionLocalGeoSiteTag=Site2
```

a) Run `copytoall.sh` to restart the qns processes.

Step 3 Add session geo tag in MongoDBs.

a) First, get the session database replica members information.

Execute `rs.conf()` from any one member.

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:27717",
                        "arbiterOnly" : true
                },
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27717",
                        "priority" : 4
   },
                {
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27717",
```

```
                        "priority" : 3
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr01-site2:27717",
                        "priority" : 2
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr02-site2:27717"
                        "priority" : 1
                }
        ],
        "settings" : {
                "heartbeatTimeoutSecs" : 1
        }
}
```

b) Now from the list, find out the members of Site1 and Site2 to be tagged (excluding the arbiter).
c) After finding member, execute the following command from Primary member to tag members.

Note To modify priorities, you must update the `members` array in the replica configuration object. The array index begins with 0. The array index value is different than the value of the replica set member's `members[n]._id` field in the array.

```
conf = rs.conf();
conf.members[1].tags = { "sessionLocalGeoSiteTag": "Site1" }
conf.members[2].tags = { "sessionLocalGeoSiteTag": "Site1"}
conf.members[3].tags = { "sessionLocalGeoSiteTag": "Site2"}
conf.members[4].tags = { "sessionLocalGeoSiteTag": "Site2"}
rs.reconfig(conf);
```

THIS IS SAMPLE CONFIG, MEMBERS and TAG can be different according to your deployment.

`conf.members[1]` means member with "_id" = "1" in output of rs.conf();

After executing this command, primary member may get changed if all members have same priority.

Verify that the tags are properly set by log in to on any member and executing `rs.conf()` command.

SAMPLE OUTPUT

```
set04:PRIMARY> rs.conf();
{
        "_id" : "set04",
        "version" : 319396,
        "members" : [
                {
                        "_id" : 0,
                        "host" : "pcrfclient-arbiter-site3:27717",
                        "arbiterOnly" : true
                },
                {
                        "_id" : 1,
                        "host" : "sessionmgr01-site1:27717",
                        "priority" : 4,
                        "tags" : {
                                "sessionLocalGeoSiteTag" : "Site1"
                        }
                },
                {
```

```
                        "_id" : 2,
                        "host" : "sessionmgr02-site1:27717",
                        "priority" : 3,
                        "tags" : {
                                "sessionLocalGeoSiteTag" : "Site1"
                        }
                },
                {
                        "_id" : 3,
                        "host" : "sessionmgr01-site2:27717",
                        "priority" : 2,
                        "tags" : {
                                "sessionLocalGeoSiteTag" : "Site2"
                        }
                },
                {
                        "_id" : 4,
                        "host" : "sessionmgr01-site2:27717",
                        "priority" : 1,
                        "tags" : {
                                "sessionLocalGeoSiteTag" : "Site2"
                        }
                }
        ],
        "settings" : {
                "heartbeatTimeoutSecs" : 1
        }
}
```

Publishing Configuration Changes When Primary Site becomes Unusable

Step 1 Configure auto SVN sync across sites on GR secondary site.

a) Configure this site's pcrfclient01 public IP in `/var/qps/config/deploy/csv/AdditionalHosts.csv` file of Cluster Manager (use same host name in 2.a, on page 64).

Example: `public-secondary-pcrfclient01,,XX.XX.XX.XX,`

For OpenStack, add the new hosts in the AdditionalHosts section. For more information, refer to */api/system/config/additional-hosts* section in *CPS Installation Guide for OpenStack*

b) Recreate SVN repository on secondary site pcrfclient01/02.

Note Take backup of SVN repository for rollback purpose.

From pcrfclient01, execute the following commands:

```
/bin/rm -fr /var/www/svn/repos
```

```
/usr/bin/svnadmin create /var/www/svn/repos
```

```
/etc/init.d/vm-init-client
```

From pcrfclient02, execute the following commands:

```
/bin/rm -fr /var/www/svn/repos

/usr/bin/svnadmin create /var/www/svn/repos

/etc/init.d/vm-init-client
```

c) Login to pcrfclient01 and recover svn using `/var/qps/bin/support/recover_svn_sync.sh` script.

d) Verify SVN status using `diagnostics.sh` script from Cluster Manager. Output should look like:

```
diagnostics.sh --svn

CPS Diagnostics HA Multi-Node Environment
---------------------------
Checking svn sync status between pcrfclient01 and pcrfclient02...[PASS]
```

Step 2 Configure auto SVN sync across site on GR primary site.

For OpenStack, add the new hosts in the AdditionalHosts section. For more information, refer to */api/system/config/additional-hosts* section in *CPS Installation Guide for OpenStack*

a) Configure remote/secondary pcrfclient01 public IP in `/var/qps/config/deploy/csv/AdditionalHosts.csv` file of Cluster Manager.

Example: `public-secondary-pcrfclient01,,XX.XX.XX.XX,`

b) Configure svn_slave_list as `svn_slave_list,pcrfclient02 public-secondary-pcrfclient01` in `/var/qps/config/deploy/csv/Configuration.csv` file of Cluster Manager (replace public-secondary-pcrfclient01 with the host name assigned in Step 2 a).

c) Configure svn recovery to be every 10 minutes that is, `auto_svn_recovery,enabled` in `/var/qps/config/deploy/csv/Configuration.csv` file of Cluster Manager.

d) Execute `/var/qps/install/current/scripts/import/import_deploy.sh` on Cluster Manager.

e) Execute the following command to validate the imported data:

```
cd /var/qps/install/current/scripts/deployer/support/

python jvalidate.py
```

Note The above script validates the parameters in the Excel/csv file against the ESX servers to make sure ESX server can support the configuration and deploy VMs.

f) Login to pcrfclient01 and re-initiate pcrfclient01 using `/etc/init.d/vm-init-client` command.

g) Verify SVN status using `diagnostics.sh` script from Cluster Manager. Output should look like (Wait for maximum 10 mins as per cron interval for PASS status):

```
diagnostics.sh --svn
CPS Diagnostics HA Multi-Node Environment
---------------------------
Checking svn sync status between pcrfclient01 & pcrfclient02...[PASS]
Checking svn sync status between pcrfclient01 & remote-pcrfclient01...[PASS]
```

Test scenario

1. Both primary and secondary sites are up:
   - Login to primary site Policy Builder and update policy.
   - Verify primary site SVNs are in sync between pcrfclient01 and 02.
   - Verify primary site and secondary site SVN are in sync (this takes approx 10 mins as per cron interval).
2. Both primary and secondary sites are up (This is not recommended).

- Login to secondary site Policy Builder and update policy.
- Verify secondary site SVNs are in sync between pcrfclient01 and 02.
- Verify primary site and secondary site SVN are in sync (this takes approx 10 mins as per cron interval).

3. Primary site up and secondary site down.
   - Login to primary site Policy Builder and update policy.
   - Verify primary site SVNs are in sync between pcrfclient01 and 02.
   - Recover secondary site. Verify primary site and secondary site SVN are in sync (this takes approx10 mins as per cron interval).
4. Secondary site up and primary site down.
   - Login to secondary site Policy Builder and update policy.
   - Verify secondary site SVNs are sync between pcrfclient01 and 02.
   - Recover primary site. Verify primary site and secondary site SVN are in sync (this takes approx 10 mins as per cron interval).

Graceful Cluster Shutdown

Utility script `/var/qps/bin/support/mongo/migrate_primary.sh` is a part of the build and will be available in newly installed or upgraded setups.

This utility simply reads the cluster configuration file and `mongoConfig.cfg` file, and migrates Mongo Primary status from one cluster to another before doing an upgrade by setting priority '1' (or given in the command). After upgrade, utility script again migrates Mongo Primary status to original cluster by restoring the original priority.

Here is a help page of `migrate_primary.sh` (`/var/qps/bin/support/mongo/migrate_primary.sh -help`) utility:

```
/var/qps/bin/support/mongo/migrate_primary.sh [--options]
[--db-options] [--hosts-all|--hosts-files <host-file-name> --hosts <host list..> ]
/var/qps/bin/support/mongo/migrate_primary.sh --restore <restore-filename>

  --hosts      CPS Host list which are upgrading (like sessionmgr01, lb01, pcrfclient01)
  --hosts-file File name which contains CPS upgrading hosts
  --hosts-all  Upgrading all hosts from this cluster
  --restore    Restore priority back

DB Options - you can provide multiples DBs
  --all         All databases in the configuration
  --spr         All SPR databases in the configuration
  --session     All Session databases in the configuration
  --balance     All Balance databases in the configuration
  --admin       All Admin databases in the configuration
  --report      All Report databases in the configuration
  --portal      All Portal databases in the configuration
  --audit       All Aduit databases in the configuration
```

```
Options:
  --setpriority <priority-num> Set specific priority (default is 1)
  --noprompt                   Do not ask verfiication & set priority, without y/n prompt
  --prompt                     Prompt before setting priority (Default)
  --nonzeroprioritychk         Validate all upgrading members have non-zero priority
  --zeroprioritychk            Validate all upgrading members have zero priority
  --debug                      For debug messages (default is non-debug)
  --force                      Set priority if replica set is not healthy or member down
case
  --h [ --help ]               Display this help and exit

Description:
Reconfiguring Mongo DB priorities while doing an upgrade of a set of session managers,
the Mongo DBs that exist on that session manager need to be moved to priority 1
(provided priority) so that they will never be elected as the primary at time of upgrade.

Examples:
  /var/qps/bin/support/mongo/migrate_primary.sh --all --hosts sessionmgr01
  /var/qps/bin/support/mongo/migrate_primary.sh --session --hosts-all
  /var/qps/bin/support/mongo/migrate_primary.sh --noprompt --spr --hosts sessionmgr01
sessionmgr02
  /var/qps/bin/support/mongo/migrate_primary.sh --setpriority 1 --all --hosts sessionmgr01
```

Note When you execute `migrate_primary.sh --restore` command, ignore the following error message if encountered:

[WARN] Failed to set priority to <priority number> to <hostname> (<set number>), due to "errmsg" : "replSetReconfig should only be run on PRIMARY, but my state is SECONDARY; use the "force" argument to override",

Active/Active Geo HA - Multi-Session Cache Port Support

CPS supports communication with multiple session cache databases and process the Gx and Rx messages in Active/Active Geo HA model.

The criteria to select which Session Cache for Gx Requests for a given session should be based on configurable criteria, for example, origin realm and/or host. Wildcards are also supported. For Rx requests, CPS uses secondaryKey lookup to load session.

When session cache database is not available, backup database is used.

By default, Geo HA feature is not installed and is not enabled. To install and enable the Geo HA, perform the following steps:

Note To configure Active/Active Geo HA using APIs, refer to *Active-Active Geo HA Support* section in *CPS Installation Guide for OpenStack*.

Install Geo HA

Step 1 Edit feature file on cluster manager: `/etc/broadhop/pcrf/features`

Step 2 Remove policy feature entry from feature file.

```
com.broadhop.policy.feature
```

Step 3 Add policy Geo HA feature entry in feature file.

```
com.broadhop.policy.geoha.feature
```

Step 4 Execute the following commands:

Note *${CPS version}* - Input the CPS version in the following commands.

```
/var/qps/install/${CPS version}/scripts/build/build_all.sh
/var/qps/install/${CPS version}/scripts/upgrade/reinit.sh
```

Example:

```
/var/qps/install/9.0.0/scripts/build/build_all.sh
/var/qps/install/9.0.0//scripts/upgrade/reinit.sh
```

Enable Geo HA

Step 1 Set GeoHA flag to `true` in `qns.conf` file to enable Geo HA feature.

```
-DisGeoHAEnabled=true
```

Step 2 Remove `-DclusterFailureDetectionMS` parameter from `/etc/broadhop/qns.conf` file.

Step 3 Verify other site lb IP addresses are present in `/etc/hosts` file.

If entries are missing, modify `AdditionalHost.csv` to add entries in `/etc/hosts` file. Remote load balancer should be accessible from load balancer by host name.

Configuration

Consider there are two sites Site1, Site2. Both are active sites and Site1 failovers to Site2.

Session database is replicated across site. Session database of the site can be selected based on realm or host or local information.

Step 1 Configure the lookup type in `qns.conf` file. Possible values can be realm/host/local.

For example, `-DgeoHASessionLookupType=realm`

Note When session lookup type is set to “local”, local session database is used for read/write session irrespective of site lookup configuration. For “local” session lookup type, site lookup configuration is not required. Even if it is configured, it is not used. However, user still needs to add site and shards.

For “local” session lookup type, add the entry for “diameter” under **Lookaside Key Prefixes** under Cluster configuration (if it is not already configured) in Policy Builder.

For local session capacity planning, refer to Local Session Affinity - Capacity Planning, on page 72.

Note For memory and performance impact when session lookup is configured as local, refer to Memory and Performance Impact, on page 71.

Note In case there is an error in configuring `geoHASessionLookupType`, CPS behaves incorrectly and drop messages. The following logs come continuously if there is an error in the configuration:

```
GeoHA is enabled, unknown lookuptype is configured: <>. Possible values are…
```

Step 2 Clean up the following data from the database if any data exists.

a) (Session database) Clean the sessions:

```
session_cache_ops.sh --remove
```

b) Run `diagnostics.sh` command on pcrfclient01 and find session database primary member and port number.

```
diagnostics.sh --get_replica_status
```

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

c) Login to session database using the primary replica set hostname and port number.

For example, `mongo --host sessionmgr01 --port 27720`

d) (Admin database) Remove shard entries from shards collection:

```
use sharding
db.shards.remove({});
db.buckets.drop(); ===> This collection is not used in GeoHA any more, so deleting.
```

e) (Admin database) Clear endpoints:

```
use queueing
db.endpoints.remove({});
use diameter
db.endpoints.remove({});
```

f) Exit the database:

```
exit
```

Step 3 Enable dictionary reload flag (Only for GR) in `/etc/broadhop/qns.conf` file.

```
-DenableReloadDictionary=true
```

Step 4 Update the following parameters in `/etc/broadhop/qns.conf` file as per site IDs.

Examples:

```
-DGeoSiteName=Site1
```

```
-DSiteId=Site1
```

```
-DRemoteSiteId=Site2
```

Step 5 Add Sites - configure/add all physical sites.

a) Login to qns OSGi console. All the following commands are to be executed from OSGi console.

```
telnet qns01 9091
```

b) Run addsite command for each primary (active) site with its secondary site ID.

```
addsite <SiteId> <SecondarySiteId>
```

where,

<*SiteId*> is the primary site ID.

<*SecondarySiteId*> is the secondary site ID.

Example: `addsite Site1 Site2`

This primary and secondary site IDs should be in sync with following entries in `/etc/broadhop/qns.conf` file.

Examples:

`-DGeoSiteName=Site1` ===> (this should be <*SiteId*> from the above `addsite` command

`-DSiteId=Site1` ===> (this should be <*SiteId*> from the above `addsite` command)

`-DRemoteSiteId=Site2` ===> (this should be <*SecondarySiteId*> from above `addsite` command)

c) Configure Site to realm or host mapping.

User needs to configure all the realms for all the interfaces (such as, Gx, Rx, and so on) here:

```
addsitelookup <SiteId> <LookupValue>
```

where,

<*SiteId*> is the primary site ID.

<*LookupValue*> is the realm/host value.

If `geoHASessionLookupType` is configured as realm in Step 1, on page 67.

Provide lookup value as realm as follows:

Example: If you have multiple Gx and Gy clients connected to CPS and the details for realms of clients are as follows:

```
Client-1: pcef-1-Gx.cisco.com
Client-2: pcef-1-Gy.cisco.com
Client-3: pcef-2-Gx.cisco.com
Client-4: pcef-2-Gy.cisco.com
```

For example, Client-1 and Client-2 are connected to Site1. Client-3 and Client-4 are connected to the Site2 then,

```
addsitelookup Site1 pcef-1-Gx.cisco.com
addsitelookup Site1 pcef-1-Gy.cisco.com
addsitelookup Site2 pcef-2-Gx.cisco.com
addsitelookup Site2 pcef-2-Gy.cisco.com
```

If `geoHASessionLookupType` is configured as host in Step 1, on page 67.

Provide lookup value as host as follows:

Example: If you have multiple Gx and Gy clients connected to CPS and the details for hostnames of clients are as follows:

```
Client-1: pcef-1-Gx
Client-2: pcef-1-Gy
Client-3: pcef-2-Gx
Client-4: pcef-2-Gy
```

For example, Client-1 and Client-2 are connected to Site1. Client-3 and Client-4 are are connected to the Site2 then,

```
addsitelookup Site1 pcef-1-Gx
addsitelookup Site1 pcef-1-Gy
addsitelookup Site2 pcef-2-Gx
addsitelookup Site2 pcef-2-Gy
```

Note The pattern matching is supported for site lookup mapping. In case the incoming host/realm does not match any of the values configured under LookupValues, request is dropped with the following exception in log:

```
GeoHASiteMappingNotFound - No realm/host to site mapping matched for:
<incoming value>
```

Note If you configure the same realm/host look up pattern for both sites, application picks only one site look up (first one in the list for same realms/hosts) to process the incoming requests. If the configured realms/host look-ups entries are same for both sites and you send calls to any site, the application will always try to use one site look up (first one in the list for same realms/hosts) to process the sessions.

So, Cisco recommends to configure identical realms/hosts names for the sites, and not configure same realms/hosts for the both sites.

Other commands:

- `listsitelookup`: To see all the configured site lookup mapping.
- `deletesitelookup <SiteId> <LookupValue>`: To delete specific site lookup mapping.

d) Add Shards: Configure/add shards for the site.

```
addshard <Seed1>[,<Seed2>] <Port> <Index> <SiteId> [<BackupDb>]
```

where,

<SiteId> is the primary site of the shard. This maps the shard with site.

[<BackupDb>] is an optional parameter.

Example: `addshard sessionmgr01,sessionmgr02 27717 1 Site1`

Note By default, there may not be any default shards added when Geo HA is enabled. So add the shards starting from index 1.

To configure hot standby feature, use `addshard` command with backup database parameter:

```
addshard <Seed1>[,<Seed2>] <Port> <Index> <SiteId> [<BackupDb>]
```

Examples:

```
addshard sessionmgr09,sessionmgr10 27717 1 Site1 backup
addshard sessionmgr09,sessionmgr10 27717 2 Site1 backup
```

e) Rebalance the shards by executing the following command:

```
rebalance <SiteId>
```

Example: `rebalance Site1`

f) Migrate the shards by executing the following command:

```
migrate <SiteId>
```

Example: `migrate Site1`

Step 6 (Optional) The following parameter should be updated in `/etc/broadhop/qns.conf` file for SP Wi-Fi based deployments:

```
-DisRadiusBasedGeoHAEnabled=true
```

Note For SP Wi-Fi based deployments, lookup value can be configured as NAS IP of ASR1K or ASR9K in Step 5, on page 69.

Note RADIUS-based policy control is no longer supported in CPS 14.0.0 and later releases as 3GPP Gx Diameter interface has become the industry-standard policy control interface.

Memory and Performance Impact

Performance Impact

In in-service migration (ISSU), if sessions are stored in remote database, there are additional queries on memcache and session databases. This leads to performance impact till all the sessions are migrated to local.

In fresh installation, there is no performance impact.

Memory Impact

As additional primary keys are stored in the cache ring, it takes additional memory for storing primary keys in memcache.

Failover Impact

In case of failover, Policy Server (QNS) queries local session database first and if sessions are not found in local, it queries memcache to find the remote shard and loads session from remote database.

As there is one additional local query, it leads to some performance impact during/after failover period till all the existing sessions are migrated. There is no impact for new sessions.

GR Configuration with Session Replication Across Sites

Login to pcrfclient01/02 to create/update rings from Policy Server (QNS) OSGi console. Assuming there are two session cache replica-sets. By default, Ring-1 Set-1 get configured automatically and remaining rings need to be configured manually.

Configure cluster-1 (Ring-1)

```
osgi> setSkRingSet 1 1 <hostname_primary_site_sessionmgr01>:11211,
<hostname_primary_site_sessionmgr02>:11211
Ring updated
osgi> setSkRingSet 1 2 <hostname_primary_site_sessionmgr03>:11211,
<hostname_primary_site_sessionmgr04>:11211
Ring updated
```

Configure cluster-2 (Ring-2)

```
telnet qns01 9091
osgi> createSkRing 2
Successfully added ring with ID: 2
osgi> setSkRingSet 2 1 <hostname_secondary_site_sessionmgr01>:11211,
<hostname_secondary_site_sessionmgr02>:11211
osgi> setSkRingSet 2 2 <hostname_secondary_site_sessionmgr03>:11211,
<hostname_secondary_site_sessionmgr04>:11211
```

An example configuration is given below:

- Configure cluster-1 (Ring-1):

  ```
  osgi> setSkRingSet 1 1 L2-CA-PRI-sessionmgr01:11211, L2-CA-PRI-sessionmgr02:11211
  Ring updated
  osgi> setSkRingSet 1 2 L2-CA-PRI-sessionmgr03:11211, L2-CA-PRI-sessionmgr04:11211
  Ring updated
  ```

- Configure cluster-2 (Ring-2):

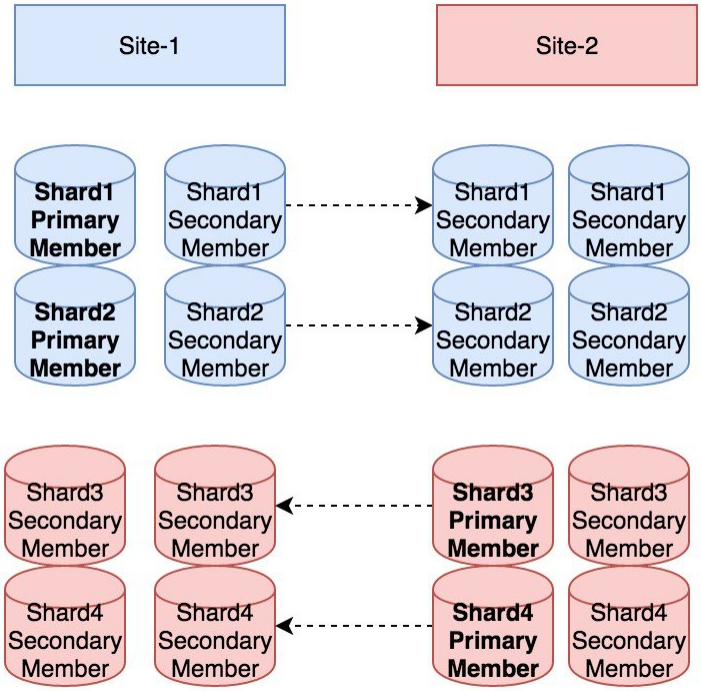
  ```
  telnet qns01 9091
  osgi> createSkRing 2
  Successfully added ring with ID: 2
  osgi> setSkRingSet 2 1 L2-CA-SEC-sessionmgr01:11211, L2-CA-SEC-sessionmgr02:11211
  osgi> setSkRingSet 2 2 L2-CA-SEC-sessionmgr03:11211, L2-CA-SEC-sessionmgr04:11211
  ```

Local Session Affinity - Capacity Planning

Consider there are two sites Site-1 and Site-2. Site-1 failovers to Site-2 and vice-versa.

With Local session affinity, both sites store new sessions to their local database in normal and failover condition. In normal conditions both sites need two session shards as shown in Figure 20: Site1 and Site2 - Normal Conditions, on page 73.

Figure 20: Site1 and Site2 - Normal Conditions

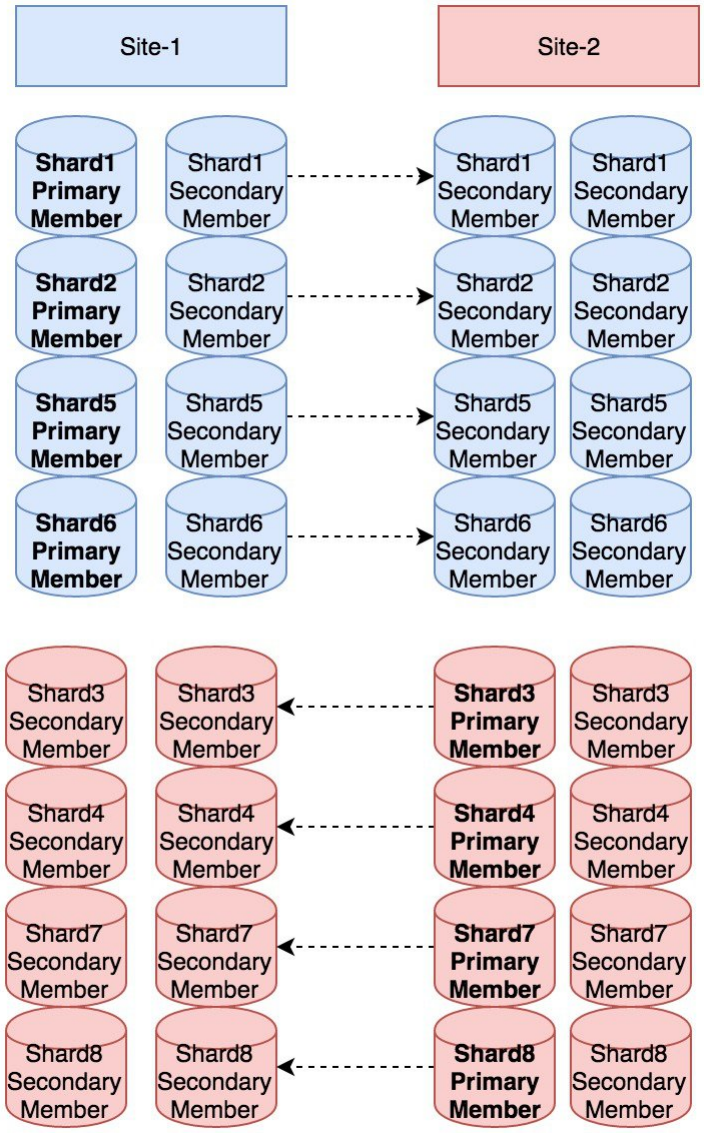


Site-1 writes sessions to Shard1 and Shard2, while Site-2 writes to Shard3 and Shard4.

Now consider, Site-2 goes down and Site-1 receives traffic for both the sites. Post failover, for new sessions, Site-1 will use Shard-1 and Shard-2 and it will not use the Shard-3 and Shard-4.

Site-1 needs to have extra session shards to handle the traffic from Site-1 and Site2 (after failover). Same for Site-2 (refer to Figure 21: Failover Scenario, on page 74 with additional shards).

Figure 21: Failover Scenario



Site-1 writes sessions to Shard1, Shard2, Shard5, Shard6 and Site-2 writes to Shard3, Shard4, Shard7, Shard8. Now when failover occurs both sites have additional local shards to accommodate sessions from another site.

Limitation

In this case, for profile refresh scenario, there is no 'smart' search (for example, IMSI-based match in given set of shards). In case a session for given profile is not found in concerned site's all session shards, search would be extended to all shards in all the sites.

For SP Wi-Fi deployments: Portal call flows are currently not supported in SP Wi-Fi Active-Active with session replication deployments. Currently, only ASR1K and ASR9K devices are supported for Active-Active deployments.

Note RADIUS-based policy control is no longer supported in CPS 14.0.0 and later releases as 3GPP Gx Diameter interface has become the industry-standard policy control interface.

Handling RAR Switching

When Gx session is created on Site2 and SPR update or Rx call comes on Site1, CPS sends Gx RAR from Site1 and in response PCEF sends RAA and next CCR request to Site1.

This leads to cross-site call switches from Site2 to Site1. If there are lot of call switches, Site1 may get overloaded.

By default, cross-site switching is enabled in CPS. To prevent cross-site switching, user needs to configure `-DRemoteGeoSiteName` parameter in `/etc/broadhop/qns.conf` file. This parameter enables cross-site communication for outbound messages like for RAR if we do not have DRA outside policy director (lb) and want to avoid RAR Switches.

Parameter Name: `-DRemoteGeoSiteName=<`*sitename*`>`

where, <*sitename*> is remote site name to be provided, and only to be added if we want to disable Gx-RAR switches from PCRF. It should match with `-DRemoteSiteId`.

Example:

```
-DRemoteSiteId=clusterA_SBY

-DRemoteGeoSiteName=clusterA_SBY
```

Prerequisite: Both remote site and local site policy server (QNS) should be able to communicate to load balancer on same interfaces. To change interface, flag `-Dcom.broadhop.q.if=<enter replication interface name>` can be used.

After configuring `-DRemoteGeoSiteName` parameter in `qns.conf` file, execute the following commands from Cluster Manager:

```
/var/qps/bin/control/copytoall.sh

/var/qps/bin/control/restartall.sh
```

If Redis IPC is used, make sure remote/peer policy director (lb) VM information is configured on local site for RAR switching to work. For more information, refer to Policy Director (lb) VM Information on Local Site.

Configure Cross-site Broadcast Messaging

The cross-site broadcast message configuration is required when there are separate sessions (no replication for sessions DB) but common subscriber profile and subscriber provisioning event needs to be done on single site. In this case, the profile updates for subscriber sessions on remote sites need to be broadcasted to respective sites so that the corresponding RARs go from the remote sites to their respective diameter peers.

Edit `/etc/broadhop/qns.conf` file and add the following line:

`-DclusterPeers=failover:(tcp://<remote-site-lb01>:<activemq port>,tcp://<remote-site-lb02>:<activemq port>)?updateURIsSupported=false!<remote-site-cluster-name>.default`

where,

- <*remote-site-lb01*> is the IP address of the remote site lb01.
- <*activemq port*> is the port on which activemq is listening. Default is 61616.
- <*remote-site-lb02*> is the IP address of the remote site lb02.
- <*remote-site-cluster-name*> is the cluster name of remote site. To get the cluster name of remote site, check the parameter value of `-Dcom.broadhop.run.clusterId` in `/etc/broadhop/qns.conf` file on remote site.

Example:

```
-DclusterPeers=failover:(tcp://107.250.248.144:61616,tcp://107.250.248.145:61616)?updateURIsSupported=false!Cluster-Site-2.default
```

Example

The following example considers three sites (SITE-A, SITE-B and SITE-C) to configure cluster broadcast messaging between them.

Note Separator between two site configurations is colon (;).

- SITE-A configuration: Edit `/etc/broadhop/qns.conf` file and add the following lines:

```
-Dcom.broadhop.run.clusterId=Cluster-Site-A
-DclusterPeers=failover:(tcp://105.250.250.150:61616,tcp://105.250.250.151:61616)?
updateURIsSupported=false!Cluster-SITE-B.default;failover:(tcp://105.250.250.160:61616,
tcp://105.250.250.161:61616)?updateURIsSupported=false!Cluster-SITE-C.default
```

- SITE-B configuration: Edit `/etc/broadhop/qns.conf` file and add the following lines:

```
-Dcom.broadhop.run.clusterId=Cluster-Site-B
-DclusterPeers=failover:(tcp://105.250.250.140:61616,tcp://105.250.250.141:61616)?
updateURIsSupported=false!Cluster-SITE-A.default;failover:(tcp://105.250.250.160:61616,
tcp://105.250.250.161:61616)?updateURIsSupported=false!Cluster-SITE-C.default
```

- SITE-C configuration: Edit `/etc/broadhop/qns.conf` file and add the following lines:

```
-Dcom.broadhop.run.clusterId=Cluster-Site-C
-DclusterPeers=failover:(tcp://105.250.250.140:61616,tcp://105.250.250.141:61616)?
updateURIsSupported=false!Cluster-SITE-A.default;failover:(tcp://105.250.250.150:61616,
tcp://105.250.250.151:61616)?updateURIsSupported=false!Cluster-SITE-B.default
```

Configure Redundant Arbiter (arbitervip) between pcrfclient01 and pcrfclient02

After the upgrade is complete, if the user wants a redundant arbiter (ArbiterVIP) between pcrfclient01 and pcrfclient02, perform the following steps:

Currently, this is only supported for HA setups.

Step 1 Update the `AdditionalHosts.csv` and `VLANs.csv` files with the redundant arbiter information:

- **Update AdditionalHosts.csv:**

  Assign one internal IP for Virtual IP (arbitervip).

  Syntax:

  <alias for Virtual IP>,<alias for Virtual IP>,<IP for Virtual IP>

  For example,

  `arbitervip,arbitervip,`*< IP for Virtual IP>*

- **Update VLANs.csv:**

  Add a new column **Pcrfclient VIP Alias** in the `VLANs.csv` file to configure the redundant arbiter name for the pcrfclient VMs:

  Figure 22: VLANs.csv

	VLAN Name	Network Target Name	Netmask	Gateway	VIP Alias	Pcrfclient VIP Alias	guestNic
2	Internal	VM Network	255.255.255.0	NA	lbvip02	arbitervip	eth0
3	Management	VLAN 94	255.255.255.0	NA	lbvip01		eth1
4	Gx	VM Network	255.255.255.0	NA	lbvip03		eth2
5							

  Execute the following command to import csv files into the Cluster Manager VM:

  `/var/qps/install/current/scripts/import/import_deploy.sh`

  This script converts the data to JSON format and outputs it to `/var/qps/config/deploy/json/`.

Step 2 SSH to the pcrfclient01 and pcrflclient02 VMs and run the following command to create arbitervip:

`/etc/init.d/vm-init-client`

Step 3 Synchronize `/etc/hosts` files across VMs by running the following command the Cluster Manager VM:

`/var/qps/bin/update/synchosts.sh`

Moving Arbiter from pcrfclient01 to Redundant Arbiter (arbitervip)

In this section we are considering the impacts to a session database replica set when the arbiter is moved from the pcrfclient01 VM to a redundant arbiter (arbitervip). The same steps need to be performed for SPR/balance/report/audit/admin databases.

Step 1 Remove pcrfclient01 from replica set (set01 is an example in this step) by executing the following command from Cluster Manager:

To find the replica set from where you want to remove pcrfclient01, refer to your `/etc/broadhop/mongoConfig.cfg` file.

```
build_set.sh --session --remove-members --setname set01
```

This command asks for member name and port number. You can find the port number from your `/etc/broadhop/mongoConfig.cfg` file.

```
Member:Port   --------> pcrfclient01:27717
pcrfclient01:27717
Do you really want to remove [yes(y)/no(n)]: y
```

Step 2 Verify whether the replica set member has been deleted by executing the following command from Cluster Manager:

```
diagnostics.sh --get_replica_status
```

```
|-------------------------------------------------------------------------------------------|
| SESSION:set01                                                                             |
|  Member-1  -  27717 : 221.168.1.5  -  PRIMARY   -  sessionmgr01 - ON-LINE  -  --------  -  1 |
|  Member-2  -  27717 : 221.168.1.6  -  SECONDARY -  sessionmgr02 - ON-LINE  -   0 sec    -  1 |
|-------------------------------------------------------------------------------------------|
```

The output of `diagnostics.sh --get_replica_status` should not display pcrfclient01 as the member of replica set (set01 in this case).

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.

Step 3 Change arbiter member from pcrfclient01 to redundant arbiter (arbitervip) in the `/etc/broadhop/mongoConfig.cfg` file by executing the following command from Cluster Manager:

```
vi /etc/broadhop/mongoConfig.cfg
[SESSION-SET1]
SETNAME=set01
OPLOG_SIZE=1024
ARBITER1=pcrfclient01:27717             <-- change pcrfclient01 to arbitervip
ARBITER_DATA_PATH=/var/data/sessions.1
MEMBER1=sessionmgr01:27717
MEMBER2=sessionmgr02:27717
DATA_PATH=/var/data/sessions.1
[SESSION-SET1-END]
```

Step 4 Edit the `/etc/broadhop/mongoConfig.cfg` file add a new replica set member. Run `build_etc.sh` to accept the updated configuration and wait for AIDO server to create the replica-set. `

To verify the replica-set has been added, run the following command:

```
build_set.sh --session
```

Step 5 Verify whether the replica set member is UP by executing the following command from Cluster Manager:

```
diagnostics.sh --get_replica_status
```

```
|-------------------------------------------------------------------------------------------|
| SESSION:set01                                                                             |
|  Member-1  -  27717 : 221.168.1.5   -  PRIMARY    -  sessionmgr01  - ON-LINE  -  --------  -  1 |
|  Member-2  -  27717 : 221.168.1.6   -  SECONDARY  -  sessionmgr02  - ON-LINE  -   0 sec    -  1 |
|  Member-3  -  27717 : 221.168.1.9   -  ARBITER    -  arbitervip    - ON-LINE  -  --------  -  1 |
|-------------------------------------------------------------------------------------------|
```

The output of `diagnostics.sh --get_replica_status` should now display arbitervip as the member of replica set (set01 in this case).

Note If a member is shown in an unknown state, it is likely that the member is not accessible from one of other members, mostly an arbiter. In that case, you must go to that member and check its connectivity with other members.

Also, you can login to mongo on that member and check its actual status.