Dual Center Operations

all

Sirenia

September 23, 2020

Contents

1 Content

This document provides an overview of the Dual Center - High Availability (HA) Operation supported by the Sirenia Context Manager and Sirenia Automation Software Suite. You should read the architecture documentation beforehand, in order to understand the components and their roles.

2 Architecture

High availability is a requirement for many deployments. This is especially important for deployments of Sirenia Automation for Desktop Automation and for Sirenia Context Management, as the end user

will be using the solution in their daily work. The supported High Availability Operation replicate data across multiple servers, and failover traffic when e.g. a primary server stops responding.

2.1 Master-Master

The HA support is built on data replication on top of PostgreSQL in a master-master architecture. The master-master setup implements a solution with several master databases (both read and write mode) and several hot application servers (both read and write mode). These servers will remain synchronized utilizing an asynchronous replication. If one server fails, the other contains all of the data of the failed server, and will quickly start serving the disconnected clients.

2.2 Bucardo

The asynchronous replication is build utilizing a Bucardo solution. Bucardo is a replication program for two or more Postgres databases. Specifically, it is an asynchronous, multi-master, table-based replication system. It is written in Perl, and uses extensive use of triggers, PL/PgSQL, and PL/PerlU.

2.3 Failover

The master-master server setup is combined with a client side failover funktionality, in order to ensure high availability. Once a Manatee client detects a failure on its current application server, it will automatically fail over to a secondary application server. The secondary application server is also a master in the complete solution, and the client will continue operation without degradation of functionality. Operations will have to take appropriate actions to restore service of the failed application server. Cuesta instances will not failover automatically, but it is possible to deploy Cuesta on both sites.

2.4 Load Balancing

In order to utilize the resources of the provisioned application servers, the Manatee clients will load balance across all available application servers running live Kwanza instances. Load balancing is implemented in client side Manatee functionality and follows a randomized round robin server selection with a first choice. The possible application servers must be configured in the Manatee configuration and follow a DNS naming scheme as `0.ha-domain`, `1.ha-domain ... n.ha-domain` until the number of possible application servers are reached.

3 HA Deployment

To perform a deployment on a dual site setup including the two example sites

- 0.ha.test
- 1.ha.test

setup DNS for each site and follow this procedure.

3.1 Prepare the Servers

Deploy a valid single-site application server deployment (including postgres, kwanza and cuesta) on **Site 0**, by following the install guides for your operation system. This could also be a single site deployment, which have been in operation for some time, which now have to be dual site HA enabled.

On **Site 1**, deploy a valid single-site application server deployment (including postgres, kwanza and cuesta), by following the install guides for your operation system. This must be a clean install, as we will be using any existing data from Site 0 as starting point for the master-master replication.

You now have two full stacks running, one on each site. Site 0 may include data. Site 1 is an empty clean install.

3.2 Setup Replication

Site 0 will be the site running the replication process. Any of the sites could have been responsible for this is a master-master setup with two sites. We chose Site 0.

Add this to the end of your `docker-compose.yaml` file on **Site 0**.

```
  postgres_replication:
    image: plgr/bucardo
    restart: always
    volumes:
      - "/usr/local/etc/sirenia/postgres/conf:/media/bucardo"
    depends_on:
      - postgres
```

Ensure that only the postgres docker is running on both sites

On **Site 0**:

```
root@0:~/deploy# docker-compose stop
Stopping deploy_cuesta_1    ... done
```

```
Stopping deploy_kwanza_1    ... done
Stopping deploy_postgres_1 ... done
root@0:~/deploy# docker-compose up -d postgres
Starting deploy_postgres_1 ... done
root@0:~/deploy# docker-compose ps
             Name                           Command               State
                            Ports
------------------------------------------------------------------------------------

deploy_cuesta_1                 /bin/sh -c /bin/sh -c "if  ...   Exit 0
deploy_kwanza_1                 kwanza serve                     Exit 2
deploy_postgres_1               docker-entrypoint.sh postgres    Up
            0.0.0.0:5444->5432/tcp
deploy_postgres_replication_1   /bin/bash -c /entrypoint.sh      Exit
    137
```

On **Site 1**

```
root@1:~/deploy# docker-compose stop
Stopping deploy_cuesta_1    ... done
Stopping deploy_kwanza_1    ... done
Stopping deploy_postgres_1 ... done
root@1:~/deploy# docker-compose up -d postgres
Starting deploy_postgres_1 ... done
root@1:~/deploy# docker-compose ps
      Name                     Command               State
          Ports
------------------------------------------------------------------------------------

deploy_cuesta_1     /bin/sh -c /bin/sh -c "if  ...   Exit 0
deploy_kwanza_1     kwanza serve                     Exit 2
deploy_postgres_1   docker-entrypoint.sh postgres    Up
    0.0.0.0:5444->5432/tcp
```

TODO: Ensure connection both ways

From **Site 0** populate Site 1 database with structrures and initial data. On **Site 0**:

```
docker exec -it deploy_postgres_1 "/bin/bash"
...
su postgres
...
echo "DROP DATABASE IF EXISTS kwanza" | psql -U postgres -h 1.ha.test -
    p 5444
```

```
...
echo "CREATE DATABASE kwanza" | psql -U postgres -h 1.ha.test -p 5444
...
pg_dump --schema-only kwanza | psql -U postgres -h 1.ha.test -p 5444 -d
    kwanza
...
exit
...
exit
```

Initial load DB on secondary (master-slave with full copy)

```
nano /usr/local/etc/sirenia/postgres/conf/bucardo.json

 "databases":[
   {
     "id": 0,
     "dbname": "kwanza",
     "host": "0.ha.test port=5444",
     "user": "postgres",
     "pass": "postgres"
   },{
     "id": 1,
     "dbname": "kwanza",
     "host": "1.ha.test port=5444",
     "user": "postgres",
     "pass": "postgres"
   }],
 "syncs" : [
   {
     "sources": [0],
     "targets": [1],
     "tables": "all",
     "onetimecopy": 1
   }
 ]
}
```

Start initial replication

```
docker-compose up postgres_replication
```

Wait for an entry with state `Good`.

```
...
postgres_replication_1  |  Name    State    Last good    Time    Last I
   /D    Last bad    Time
postgres_replication_1  |
   =======+========+============+=======+==========+==========+=======

postgres_replication_1  |  sync0 | Good    | 12:56:06    | 2s    | 0/17
         | none      |
```

Stop the replication process with `ctrl-c` and adjust configuration to a master-master setup

```
nano /usr/local/etc/sirenia/postgres/conf/bucardo.json

{
 ”databases”:[
   {
     ”id”: 0,
     ”dbname”: ”kwanza”,
     ”host”: ”0.ha.test port=5444”,
     ”user”: ”postgres”,
     ”pass”: ”postgres”
   },{
     ”id”: 1,
     ”dbname”: ”kwanza”,
     ”host”: ”1.ha.test port=5444”,
     ”user”: ”postgres”,
     ”pass”: ”postgres”
   }],
 ”syncs” : [
   {
     ”sources”: [0,1],
     ”targets”: [],
     ”tables”: ”all”,
     ”onetimecopy”: 0
   }
 ]
}
```

Start the master-master replication and ensure that it runs in state `Good`

```
root@0:~/deploy# docker-compose up -d postgres_replication
deploy_postgres_1 is up-to-date
```

```
Starting deploy_postgres_replication_1 ... done
root@0:~/deploy# docker logs -t --tail 100 -f
   deploy_postgres_replication_1
...

Name    State    Last good    Time    Last I/D    Last bad    Time
=======+========+============+=======+===========+===========+=======
sync0 | Good   | 13:19:08   | 7s    | 37/106    | none      |
```

Stop the replication log tail with `ctrl-c`

3.3 Start the full stack on both sites

On **Site 0**:

```
root@0:~/deploy# docker-compose up -d
deploy_postgres_1 is up-to-date
deploy_postgres_replication_1 is up-to-date
Starting deploy_kwanza_1 ... done
Starting deploy_cuesta_1 ... done
root@0:~/deploy# docker-compose ps
            Name                           Command               State
                                                 Ports
-------------------------------------------------------------------------------------------

deploy_cuesta_1                 /bin/sh -c /bin/sh -c "if  ...   Up
        0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp
deploy_kwanza_1                 kwanza serve                     Up
        0.0.0.0:6060->6060/tcp, 0.0.0.0:8000->8000/tcp,
   0.0.0.0:8001->8001/tcp
deploy_postgres_1               docker-entrypoint.sh postgres    Up
        0.0.0.0:5444->5432/tcp
deploy_postgres_replication_1   /bin/bash -c /entrypoint.sh      Up
```

On **Site 1**:

```
root@1:~/deploy# docker-compose up -d
deploy_postgres_1 is up-to-date
Starting deploy_kwanza_1 ... done
Starting deploy_cuesta_1 ... done
root@1:~/deploy# docker-compose ps
      Name                     Command               State
                                          Ports
```

```
7  ------------------------------------------------------------------------------

8  deploy_cuesta_1     /bin/sh -c /bin/sh -c "if  ...   Up
       0.0.0.0:443->443/tcp, 0.0.0.0:80->80/tcp
9  deploy_kwanza_1     kwanza serve                    Up
       0.0.0.0:6060->6060/tcp, 0.0.0.0:8000->8000/tcp, 0.0.0.0:8001->8001/
       tcp
10 deploy_postgres_1   docker-entrypoint.sh postgres   Up
       0.0.0.0:5444->5432/tcp
```

4 Configure Manatee

In order to have Manatee do automatically failover and load balance between sites, the configuration of Manatee must be adapted to this situation. The following settings must be set:

- URL for Kwanza: `grpc://ha.test:8001`
- Number of load-balancing Kwanza servers: `2`

For test purpose you may want to lower the reconnect delay. Don't do this in production. (In production this should be approx `180` seconds.)

- Kwanza reconnect delay: `5`

Given these settings Manatee will random round robin between the servers

- 0.ha.test - Site 0
- 1.ha.test - Site 1

on a 50/50 load approximation. If you would like to achieve a non-symmetric load setup, you could assign more DNS names to one site. This setup will eg. realize a 66/33 load on Site 0 vs. Site 1.

- 0.ha.test - Site 0
- 1.ha.test - Site 0
- 2.ha.test - Site 1

Remember the implications on load on Site 1 if Site 0 fails.

5 Test the HA setup

The new dual center master-master setup is running. You should perform the following tests:

- Will changes made in Cuesta on Site 0 show in Site 1?

- Will changes made in Cuesta on Site 1 show in Site 0?
- Will Manatee connect to Site 0 if configured directly to that?
- Will Manatee connect to Site 1 if configured directly to that?
- Will Manatee connect to Site 0 or Site 1 if configured to HA operations?
- Will Manatee fail over to alternative if configured to HA operations and current site is killed?

6 Monitor

In order to realise a reliable operation of the replication process, operations should monitor the state of the replication.

6.1 Host and Container

Oerations should monitor the Docker Host, Docker Container, and the state of the replication process inside the replication container. For Docker Host and Docker Container refer to the Operations Guide for your operating system.

6.2 Replication Process

To get the state of the replication process inside the replication container look in the logfile from the container:

```
root@0:~/deploy# docker logs -t --tail 100 -f
    deploy_postgres_replication_1
...

Name    State    Last good    Time    Last I/D    Last bad    Time
=======+========+============+=======+===========+===========+=======
sync0 | Good   | 13:19:08   | 7s    | 37/106    | none      |
```

For more details enter the container and look for the details there:

```
root@0:~/deploy# docker exec -it deploy_postgres_replication_1 "/bin/
    bash"
root@85e23b6fc26d:/# su postgres
postgres@85e23b6fc26d:/$ bucardo status
PID of Bucardo MCP: 199
 Name    State    Last good    Time     Last I/D    Last bad    Time
=======+========+============+========+===========+===========+=======
```

```
 sync0 | Good    | 15:38:58   | 4m 10s | 1/3       | none      |
```

And detailed status of the sync job

```
postgres@85e23b6fc26d:/$ bucardo status sync0
================================================================================
Last good                : Jan 16, 2020 15:38:56 (time to run: 2s)
Rows deleted/inserted    : 1 / 3
Sync name                : sync0
Current state            : Good
Source relgroup/database : sync0_25 / db1
Tables in sync           : 22
Status                   : Active
Check time               : None
Overdue time             : 00:00:00
Expired time             : 00:00:00
Stayalive/Kidsalive      : Yes / Yes
Rebuild index            : No
Autokick                 : Yes
Onetimecopy              : No
Post-copy analyze        : Yes
Last error:              :
================================================================================
```

7 Security

It's important to consider security on the HA environment before going into production. There are several security aspects such as encryption, role management and access restriction by IP address. These topics are beyond the scope of this guide and best practice for postgres administration should be followed.

7.1 Minimum Requirement

A minimum requirement is to alter the postgres user password, as going into a HA setup, requires that the postgres port is exposed to the network (this is not the case in a single server configuration)

The environment variable `POSTGRES_PASSWORD` sets the superuser password for PostgreSQL. The default superuser is defined by the `POSTGRES_USER` environment variable. The `docker-compose.yaml` files should be adjusted to use these. Refer to the docker image documentation at https://hub.docker.com/_/postgres

7.2 Further Security Measures

Further security measures can be handled through the `pg_hba.conf` file which handles the client authentication. Limit the type of connection, the source IP or network, which database, and with which users can be controlled here.

Correct user management, either using secure passwords or limiting access and privileges, is also an important piece of the security settings. It is recommended to assign the minimum amount of privileges possible to users, as well as to specify, if possible, the source of the connection.

Finally, it is recommended to keep servers up to date with the latest patches, to avoid security risks.