
Dual Center Operations Setup and Monitoring Guide

March 20, 2025



Contents

Architecture	2
Master-Master	3
MSSQL Always On	3
Postgres Bucardo	3
Failover	3
Load Balancing	4
HA Deployment	4
Prepare the Servers	5
Setup Replication (Postgres Example)	5
Start the full stack on both sites	9
Configure Manatee	10
Test the HA setup	10
Monitor	10
Host and Container	11
Replication Process	11
Security	12
Minimum Requirement	12
Further Security Measures	12

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.

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.

Master-Master

The HA support is built on data replication 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.

MSSQL Always On

One possibility is to utilize the MSSQL Always On technology. The Always On availability groups feature is a high-availability and disaster-recovery solution that provides an enterprise-level alternative to database mirroring. Introduced in SQL Server 2012 (11.x), Always On availability groups maximizes the availability of a set of user databases for an enterprise. An availability group supports a failover environment for a discrete set of user databases, known as availability databases, that fail over together. An availability group supports a set of read-write primary databases and one to eight sets of corresponding secondary databases.

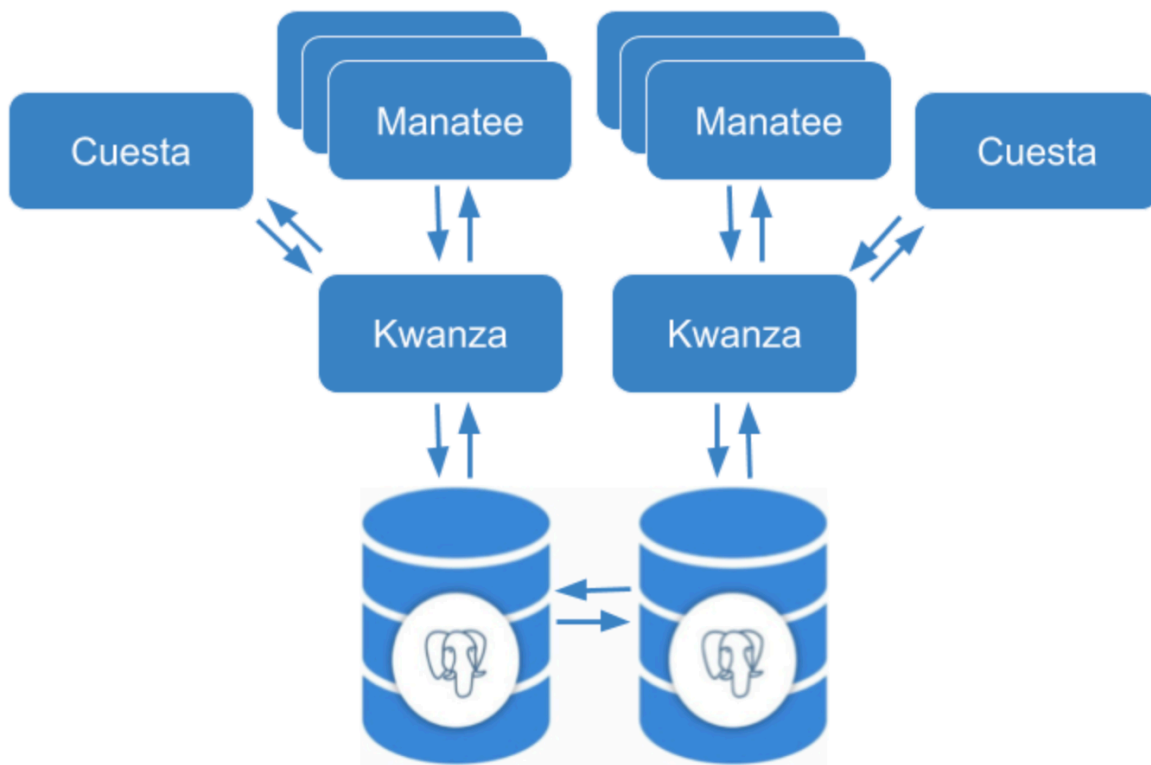
If MSSQL Always On is chosen as solution, appropriate licenses should be acquired both from Microsoft and Sirenia, and the Always On solution should be setup according to best-practice.

Postgres Bucardo

Asynchronous replication build utilizing a Bucardo solution could also be chosen as replication 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. Bucardo replication is not supported by Sirenia. It's only a suggestion. Other replication mechanisms may be utilized as well. The party operating the system should select a replication mechanism they know and have been educated in.

Failover

The master-master server setup is combined with a client side failover functionality, 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.



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. Alternatively a list of active servers can be provided which does not follow the naming scheme.

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.

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.

Setup Replication (Postgres Example)

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**.

```
1  postgres_replication:
2    image: plgr/bucardo
3    restart: always
4    volumes:
5      - "/usr/local/etc/sirenia/postgres/conf:/media/bucardo"
6    depends_on:
7      - postgres
```

Ensure that only the postgres docker is running on both sites

On **Site 0**:

```
1 root@0:~/deploy# docker-compose stop
2 Stopping deploy_cuesta_1 ... done
3 Stopping deploy_kwanza_1 ... done
4 Stopping deploy_postgres_1 ... done
5 root@0:~/deploy# docker-compose up -d postgres
6 Starting deploy_postgres_1 ... done
7 root@0:~/deploy# docker-compose ps
8
9      Name                                Command                                State
10  -----
11  deploy_cuesta_1                        /bin/sh -c /bin/sh -c "if ...      Exit 0
```

```

11 deploy_kwanza_1          kwanza serve          Exit 2
12 deploy_postgres_1       docker-entrypoint.sh postgres Up
                           0.0.0.0:5444->5432/tcp
13 deploy_postgres_replication_1 /bin/bash -c /entrypoint.sh Exit
    137

```

On Site 1

```

1 root@1:~/deploy# docker-compose stop
2 Stopping deploy_cuesta_1 ... done
3 Stopping deploy_kwanza_1 ... done
4 Stopping deploy_postgres_1 ... done
5 root@1:~/deploy# docker-compose up -d postgres
6 Starting deploy_postgres_1 ... done
7 root@1:~/deploy# docker-compose ps
8      Name                                Command                State
9      Ports
-----
10 deploy_cuesta_1      /bin/sh -c /bin/sh -c "if ... Exit 0
11 deploy_kwanza_1     kwanza serve          Exit 2
12 deploy_postgres_1   docker-entrypoint.sh postgres Up
                           0.0.0.0:5444->5432/tcp

```

Ensure connection both ways.

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

```

1 docker exec -it deploy_postgres_1 "/bin/bash"
2 ...
3 su postgres
4 ...
5 echo "DROP DATABASE IF EXISTS kwanza" | psql -U postgres -h 1.ha.test -
  p 5444
6 ...
7 echo "CREATE DATABASE kwanza" | psql -U postgres -h 1.ha.test -p 5444
8 ...
9 pg_dump --schema-only kwanza | psql -U postgres -h 1.ha.test -p 5444 -d
  kwanza
10 ...
11 exit
12 ...
13 exit

```

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

```

1 nano /usr/local/etc/sirenia/postgres/conf/bucardo.json
2
3 "databases":[
4   {
5     "id": 0,
6     "dbname": "kwanza",
7     "host": "0.ha.test port=5444",
8     "user": "postgres",
9     "pass": "postgres"
10  },{
11    "id": 1,
12    "dbname": "kwanza",
13    "host": "1.ha.test port=5444",
14    "user": "postgres",
15    "pass": "postgres"
16  }],
17 "syncs" : [
18   {
19     "sources": [0],
20     "targets": [1],
21     "tables": "all",
22     "onetimecopy": 1
23   }
24 ]
25 }

```

Start initial replication

```
1 docker-compose up postgres_replication
```

Wait for an entry with state Good.

```

1 ...
2 postgres_replication_1 | Name      State      Last good      Time      Last I
   /D      Last bad   Time
3 postgres_replication_1 |
   =====+=====+=====+=====+=====+=====+=====
4 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

```

1 nano /usr/local/etc/sirenia/postgres/conf/bucardo.json
2
3 {
4   "databases": [
5     {
6       "id": 0,
7       "dbname": "kwanza",
8       "host": "0.ha.test port=5444",
9       "user": "postgres",
10      "pass": "postgres"
11    }, {
12      "id": 1,
13      "dbname": "kwanza",
14      "host": "1.ha.test port=5444",
15      "user": "postgres",
16      "pass": "postgres"
17    } ],
18   "syncs" : [
19     {
20       "sources": [0,1],
21       "targets": [],
22       "tables": "all",
23       "onetimecopy": 0
24     }
25   ]
26 }

```

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

```

1 root@0:~/deploy# docker-compose up -d postgres_replication
2 deploy_postgres_1 is up-to-date
3 Starting deploy_postgres_replication_1 ... done
4 root@0:~/deploy# docker logs -t --tail 100 -f
   deploy_postgres_replication_1
5 ...
6
7 Name      State    Last good    Time    Last I/D    Last bad    Time
8 =====+=====+=====+=====+=====+=====+=====
9 sync0 | Good   | 13:19:08   | 7s     | 37/106    | none      |

```

Stop the replication log tail with `ctrl-c`

Start the full stack on both sites**On Site 0:**

```

1 root@0:~/deploy# docker-compose up -d
2 deploy_postgres_1 is up-to-date
3 deploy_postgres_replication_1 is up-to-date
4 Starting deploy_kwanza_1 ... done
5 Starting deploy_cuesta_1 ... done
6 root@0:~/deploy# docker-compose ps
7
8
9
10
11
12

```

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

On Site 1:

```

1 root@1:~/deploy# docker-compose up -d
2 deploy_postgres_1 is up-to-date
3 Starting deploy_kwanza_1 ... done
4 Starting deploy_cuesta_1 ... done
5 root@1:~/deploy# docker-compose ps
6
7
8
9
10

```

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

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.

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?

Monitor

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

Host and Container

Operations 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.

Replication Process

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

```
1 root@0:~/deploy# docker logs -t --tail 100 -f
  deploy_postgres_replication_1
2 ...
3
4 Name      State    Last good    Time    Last I/D    Last bad    Time
5 =====+=====+=====+=====+=====+=====+=====
6 sync0 | Good    | 13:19:08    | 7s     | 37/106     | none       |
```

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

```
1 root@0:~/deploy# docker exec -it deploy_postgres_replication_1 "/bin/
  bash"
2 root@85e23b6fc26d:/# su postgres
3 postgres@85e23b6fc26d:/$ bucardo status
4 PID of Bucardo MCP: 199
5 Name      State    Last good    Time    Last I/D    Last bad    Time
6 =====+=====+=====+=====+=====+=====+=====
7 sync0 | Good    | 15:38:58    | 4m 10s | 1/3        | none       |
```

And detailed status of the sync job

```
1 postgres@85e23b6fc26d:/$ bucardo status sync0
2 =====
3 Last good           : Jan 16, 2020 15:38:56 (time to run: 2s)
4 Rows deleted/inserted : 1 / 3
5 Sync name           : sync0
6 Current state       : Good
7 Source relgroup/database : sync0_25 / db1
8 Tables in sync      : 22
9 Status              : Active
10 Check time         : None
11 Overdue time       : 00:00:00
```

```
12 Expired time           : 00:00:00
13 Stayalive/Kidsalive   : Yes / Yes
14 Rebuild index         : No
15 Autokick              : Yes
16 Onetimecopy           : No
17 Post-copy analyze     : Yes
18 Last error:           :
19 =====
```

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.

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.yml` files should be adjusted to use these. Refer to the docker image documentation at https://hub.docker.com/_/postgres

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.