Create an Active-Active geo-replicated database

Active-Active geo-replicated databases (formerly known as CRDBs) give applications write access to replicas of the dataset in different geographical locations.

The participating Redis Enterprise Software clusters that host the instances can be in distributed geographic locations. Every instance of an Active-Active database can receive write operations, and all operations are synchronized to all of the instances without conflict.

Steps to create an Active-Active database

1. **Create a service account** - On each participating cluster, create a dedicated user account with the Admin role.
2. **Confirm connectivity** - Confirm network connectivity between the participating clusters.
3. **Create Active-Active database** - Connect to one of your clusters and create a new Active-Active database.
4. **Add participating clusters** - Add the participating clusters to the Active-Active database with the user credentials for the service account.
5. **Verify creation** - Log in to each of the participating clusters and verify your Active-Active database was created on them.
6. **Confirm Active-Active database synchronization** - Test writing to one cluster and reading from a different cluster.

Prerequisites

- Two or more machines with the same version of RS installed
- Network connectivity and cluster FQDN name resolution between all participating clusters
- Network time service listener (ntpd) configured and running on each node in all clusters

Create an Active-Active database

1. To create service accounts, on each participating cluster:
   1. In your web browser, open the admin console of the cluster that you want to connect to in order to create the Active-Active database. By default, the address is: https://<RS_address>:8443
   2. Go to settings > team and click  
   3. Enter the name, email, and password for the user, select the Admin role, and click  

2. To make sure that there is network connectivity between the participating clusters, telnet on port 9443 from each participating cluster to each of the other participating clusters.

```
telnet <target FQDN> 9443
```

3. In your web browser, open the admin console of the cluster that you want to connect to in order to create the Active-Active database. By default, the address is: `https://<RS_address>:8443`

4. In databases, click 🔄.
   
   If you do not have any databases on the node, you are prompted to create a database.

5. In the Deployment box, select Geo-Distributed and click Next to create an Active-Active database on RAM.
   
   If your cluster supports Redis on Flash, in Runs on you can select Flash so that your database uses Flash memory. We recommend that you use AOF every 1 sec for the best performance during the initial Active-Active database sync of a new replica.
6. Enter the name of the new Active-Active database and select from the options:

- **Replication** - We recommend that all Active-Active database use replication for best intercluster synchronization performance. When replication is enabled, every Active-Active database master shard is replicated to a corresponding replica shard. The replica shards are then used to synchronize data between the instances, and the master shards are dedicated to handling client requests. We also recommend that you enable replica HA to ensure that the replica shards are highly-available for this synchronization.

- **Data persistence** - To protect against loss of data stored in RAM, you can enable data persistence and select to store a copy of the data on disk with snapshots or Append Only File (AOF). AOF provides the fastest and most reliable method for instance failure recovery.

- **Default database access** - When you configure a password for your database, all connections to the database must authenticate with the `AUTH` command. If you also configure an access control list, connections can specify other users for authentication, and requests are allowed according to the Redis ACLs specified for that user.

7. Configure the advanced options that you want for the database:

- **Access Control List** - You can specify the user roles that have access to the database and the Redis ACLs that apply to those connections. You can only configure access control after the Active-Active database is created.

  To define an access control list:

  1. In the Access control list section of the database configuration, click 📌.
  2. Select the role that you want to have access to the database.
  3. Select the ACL that you want the role to have in the database.
  4. Click **Save** to save the ACL.
  5. Click **Update** to save the changes to the database.

- **Endpoint port number** (Required) - The port in the range 10000-19999 that clients must use to connect to the Active-Active database.

In the **Database clustering** option, you can either:

- Make sure the Database clustering is enabled and select the number of shards that you want to have in the database. When database clustering is enabled, databases are subject to limitations on Multi-key commands. You can increase the number of shards in the database at any time.

- Clear the **Database clustering** option to use only one shard so that you can use Multi-key commands without the limitations.

- **OSS Cluster API** -

  Redis OSS Cluster API reduces access times and latency with near-linear scalability. The Redis OSS Cluster API provides a simple mechanism for Redis clients to know the cluster topology.

  Clients must first connect to the master node to get the cluster topology, and then they connect directly to the Redis proxy on each node that hosts a master shard.

  **Note**: You must use a client that supports the OSS cluster API to connect to a database that has the OSS cluster API enabled.

- **Eviction policy** - The eviction policy for Active-Active databases is `noeviction`.

- **Participating Clusters** - You must specify the URL of the clusters that you want to host instances of an Active-Active database and the admin user account to connect to each cluster.
1. In the **Participating Clusters** list, click ![add cluster icon] to add clusters.

2. For each cluster, enter the URL for the cluster (https://<cluster_fqdn>:9443), enter the credentials (email address and password) for the service account that you created, and click ![add cluster icon].

- **Causal Consistency** - Causal Consistency in an Active-Active database guarantees that the order of operations on a specific key is maintained across all instances of an Active-Active database. To enable Causal Consistency for an existing Active-Active database, use the REST API.

- **TLS** - If you enable TLS when you create the Active-Active database, the nodes use the TLS mode **Require TLS for CRDB communication only** to require TLS authentication and encryption for communications between participating clusters. After you create the Active-Active database, you can set the TLS mode to **Require TLS for all communications** so that client communication from applications are also authenticated and encryption.

---

**Test the connection to your member Redis Active-Active databases**

With the Redis database created, you are ready to connect to your database to store data. You can use one of the following ways to test connectivity to your database:

- Connect with redis-cli, the built-in command-line tool
- Connect with a *Hello World* application written in Python

Remember we have two member Active-Active databases that are available for connections and concurrent reads and writes. The member Active-Active databases are using bi-directional replication to for the global Active-Active database.

---

**Connecting using redis-cli**

redis-cli is a simple command-line tool to interact with redis database.

1. To use redis-cli on port 12000 from the node 1 terminal, run:

   ```bash
   redis-cli -p 12000
   ```

2. Store and retrieve a key in the database to test the connection with these commands:
- set key1 123
- get key1

The output of the command looks like this:

```
127.0.0.1:12000> set key1 123
OK
127.0.0.1:12000> get key1
"123"
```

3. Enter the terminal of node 1 in cluster 2, run the redis-cli, and retrieve key1.

The output of the commands looks like this:

```
$ redis-cli -p 12000
127.0.0.1:12000> get key1
"123"
```

**Connecting using Hello World application in Python**

A simple python application running on the host machine can also connect to the database.

1. In the command-line terminal, create a new file called "redis_test.py"

   `vi redis_test.py`

2. Paste this code into the "redis_test.py" file.

   This application stores a value in key1 in cluster 1, gets that value from key1 in cluster 1, and gets the value from key1 in cluster 2.

   ```python
   import redis

   rp1 = redis.StrictRedis(host='localhost', port=12000, db=0)
   rp2 = redis.StrictRedis(host='localhost', port=12002, db=0)

   print("set key1 123 in cluster 1")
   print (rp1.set('key1', '123'))
   print("get key1 cluster 1")
   print (rp1.get('key1'))
   print("get key1 from cluster 2")
   print (rp2.get('key1'))
   ```

3. To run the "redis_test.py" application, run:

   `python redis_test.py`

   If the connection is successful, the output of the application looks like:
set key1 123 in cluster 1
True
get key1 cluster 1
"123"
get key1 from cluster 2
"123"

Updated: April 14, 2022