Active-Active Redis deployments don’t have a built-in failover or failback mechanism. An application deployed with an Active-Active database connects to a replica of the database that is geographically nearby. If that replica is not available, the application can failover to a remote replica, and failback again if necessary. In this article we explain how this process works.

Active-Active connection failover can improve data availability, but can negatively impact data consistency. Active-Active replication, like Redis replication, is asynchronous. An application that fails over to another replica can miss write operations. If the failed replica saved the write operations in persistent storage, then the write operations are processed when the failed replica recovers.

Detecting Failure

Your application can detect two types of failure:

1. **Local failures** - The local replica is down or otherwise unavailable
2. **Replication failures** - The local replica is available but fails to replicate to or from remote replicas

Local Failures

Local failure is detected when the application is unable to connect to the database endpoint for any reason. Reasons for a local failure can include: multiple node failures, configuration errors, connection refused, connection timed out, unexpected protocol level errors.

Replication Failures

Replication failures are more difficult to detect reliably without causing false positives. Replication failures can include: network split, replication configuration issues, remote replica failures.

The most reliable method for health-checking replication is by using the Redis publish/subscribe (pub/sub) mechanism.

![Note:](tip) Note: Note that this document does not suggest that Redis pub/sub is reliable in the common sense. Messages can get lost in certain conditions, but that is acceptable in this case because typically the application determines that replication is down only after not being able to deliver a number of messages over a period of time.

When you use the pub/sub data type to detect failures, the application:

1. Connects to all replicas and subscribes to a dedicated channel for each replica.
2. Connects to all replicas and periodically publishes a uniquely identifiable message.
3. Monitors received messages and ensures that it is able to receive its own messages within a predetermined window of time.

You can also use known dataset changes to monitor the reliability of the replication stream, but pub/sub is preferred method because:

1. It does not involve dataset changes.
2. It does not make any assumptions about the dataset.
3. Pub/sub messages are delivered as replicated effects and are a more reliable indicator of a live replication link. In certain cases, dataset keys may appear to be modified even if the replication link fails. This happens because keys may receive updates through full-state replication (re-sync) or through online replication of effects.

Impact of sharding on failure detection
If your sharding configuration is symmetric, make sure to use at least one key (PUB/SUB channels or real dataset key) per shard. Shards are replicated individually and are vulnerable to failure. Symmetric sharding configurations have the same number of shards and hash slots for all replicas. We do not recommend an asymmetric sharding configuration, which requires at least one key per hash slot that intersects with a pair of shards.

To make sure that there is at least one key per shard, the application should:

1. Use the Cluster API to retrieve the database sharding configuration.
2. Compute a number of key names, such that there is one key per shard.
3. Use those key names as channel names for the pub/sub mechanism.

**Failing over**

When the application needs to failover to another replica, it should simply re-establish its connections with the endpoint on the remote replica. Because Active/Active and Redis replication are asynchronous, the remote endpoint may not have all of the locally performed and acknowledged writes.

It’s best if your application doesn’t read its own recent writes. Those writes can be either:

1. Lost forever, if the local replica has an event such as a double failure or loss of persistent files.
2. Temporarily unavailable, but will be available at a later time if the local replica’s failure is temporary.

**Failback decision**

Your application can use the same checks described above to continue monitoring the state of the failed replica after failover.

To monitor the state of a replica during the failback process, you must make sure the replica is available, re-synced with the remote replicas, and not in stale mode. The PUB/SUB mechanism is an effective way to monitor this.

Dataset-based mechanisms are potentially less reliable for several reasons:

1. In order to determine that a local replica is not stale, it is not enough to simply read keys from it. You must also attempt to write to it.
2. As stated above, remote writes for some keys appear in the local replica before the replication link is back up and while the replica is still in stale mode.
3. A replica that was never written to never becomes stale, so on startup it is immediately ready but serves stale data for a longer period of time.

**Replica Configuration Changes**

All failover and failback operations should be done strictly on the application side, and should not involve changes to the Active-Active configuration. The only valid case for re-configuring the Active-Active deployment and removing a replica is when memory consumption becomes too high because garbage collection cannot be performed. Once a replica is removed, it can only be re-joined as a new replica and it loses any writes that were not converged.

Updated: September 15, 2021