Creatio Academy

Caching server

Redis Cluster

Version 7.17

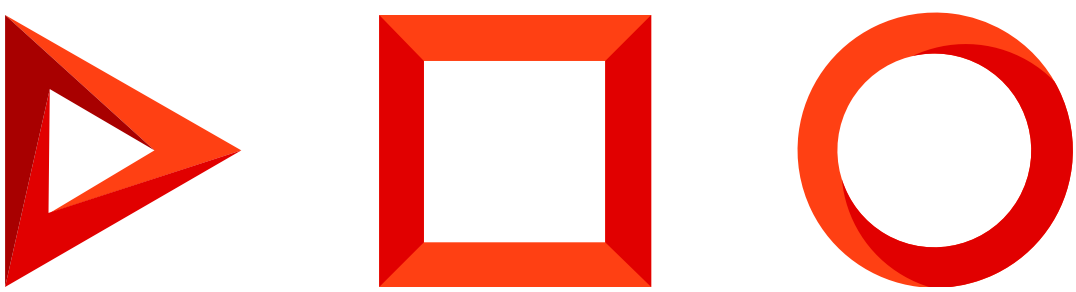


Table of Contents

Redis Cluster

PRODUCTS: ALL CREATIO PRODUCTS

The Redis Cluster mechanism provides fault tolerance for Creatio Redis repositories.

Redis Cluster's fault tolerance

Redis Cluster technology will function even if most Redis instances fail. This is achieved thanks to the following features:

- **Data replication**. Redis Cluster uses asynchronous data replication that does not merge the values. Due to asynchronous replication, the Redis Cluster system may be unable to save all data in case of failure.
- **Monitoring**. Redis Cluster uses the TCP bus to connect cluster nodes. Every node is directly connected to the other cluster nodes via the cluster bus. The nodes use a gossip protocol to disseminate the information about changes to the cluster, for example, detecting new instances, checking the connection to existing nodes, sending other cluster messages. Redis Cluster also uses the bus to send out Pub/Sub messages to the cluster and handle failures upon user request manually. Monitor the fault tolerance of the configuration regularly and use test failures to confirm the tolerance through test failures further.
- **Failover**. Redis Cluster can function when some master instances do not work as expected provided that every unavailable master instance has at least one slave instance. Thanks to replica migration, master instances that are no longer replicated by a slave instance will receive a new slave instance from a master instance serviced by several slave instances. Creatio applications that use Redis will reconfigure the connection according to the Redis Cluster status. To ensure fault tolerance, use at least six Redis instances running on separate computers or virtual machines.
- **Scalability**. Redis Cluster technology lets you add and delete nodes while the cluster is in operation. You can scale Redis Cluster up to 1000 instances thanks to automatic data sharding.

Redis Cluster system requirements

Deploy Redis Cluster on Linux with Redis Server 4.0 or later.

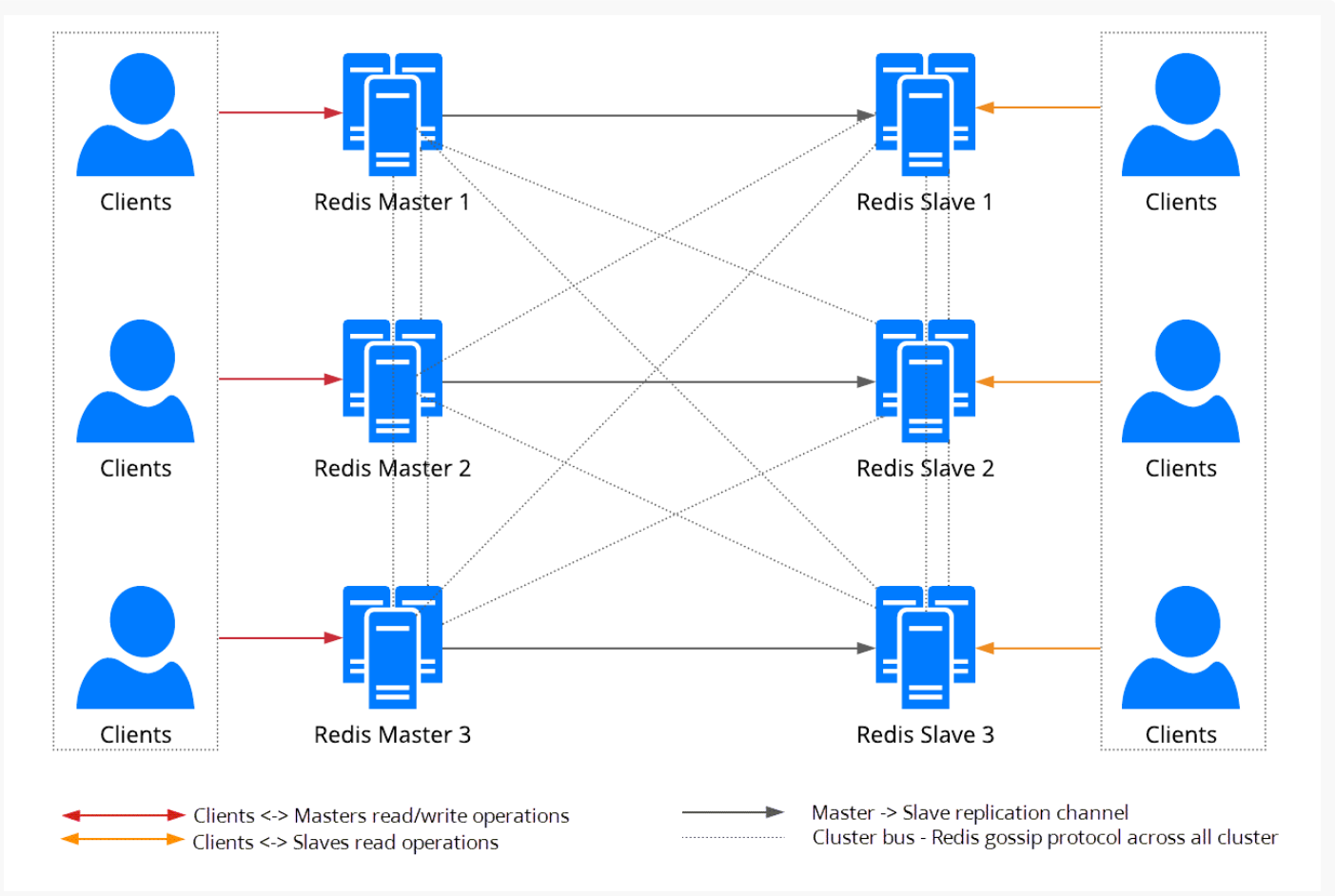
Since Redis is a single-threaded application and uses a single processor core, use a node (a computer or virtual machine) with at least a dual-core processor to deploy a **single** Redis instance. To ensure the **maximum fault tolerance**, use a physical machine for each Redis instance so that every master-slave instance is physically separated. Use the requirements calculator to check the server requirements.

Install and configure Redis Cluster

Minimal fault-tolerant Redis Cluster configuration

A configuration with at least six Redis instances is recommended. Learn more about the configuration in the “Creating and using a Redis Cluster” section of the Redis Cluster documentation. This configuration is based on six nodes (physical or virtual machines), each with a running Redis instance (Fig. 1).

Fig. 1 A six-node Redis Cluster configuration



Slave instances are read-only under normal conditions. Their data is asynchronously replicated from the master instances. Redis Cluster can both read from and write to master instances. Should a node receive a command with a key that belongs to a different node, the node that received the command will return the information about the node that has to run the operation.

Should a master Redis instance become unavailable, Redis Cluster will initiate failover. During the failover, Redis Cluster promotes one of the Redis slave instances to master. Creatio will use it instead of the previous master instance.

Attention. There is a risk of losing records with any configuration that replicates data asynchronously. This is because Redis Claster may have insufficient time to write data to a newly-promoted slave instance. The automatic reconfiguration that assigns a new master instance may take up to 15 seconds.

Install Redis Cluster

Redis Cluster is bundled with the Redis distribution. The latest Redis version is recommended.

Learn more about the installation process in the Redis documentation. There is a Redis Cluster configuration setup example in the “Creating and using a Redis Cluster” section.

To **monitor the cluster status**, run the following checks when connecting to one of the nodes:

- **View the cluster configuration** with the `cluster nodes` command.
- **Test the general cluster health** with redis-cli: `redis-cli --cluster check ClusterIP` where “ClusterIP” is the IP address of the node.

Configure Creatio to work with Redis Cluster

1. Specify the Redis Cluster node IPs in the `ConnectionStrings.config` file:

   ```
   <add name="redis" connectionString="clusterHosts=ClusterIP1:port1,ClusterIP2:port2,ClusterIP3
   ```

   where ClusterIP1:port1–ClusterIPn:portn are the IP addresses and ports of the cluster nodes.
2. Make sure that the `Feature-UseRetryRedisOperation` functionality is enabled in Creatio. This feature runs an internal Creatio mechanism that will retry any Redis operations that ended with errors. Check this:
   a. in the `web.config` file in Creatio root directory for Creatio on **Windows**
   b. in the `Terrasoft.WebHost.dll.config` file for Creatio on **Linux**

   ```
   <add key="Feature-UseRetryRedisOperation" value="true" />
   ```
3. Make sure that the **redis** section of the `web.config` (Windows) or `Terrasoft.WebHost.dll.config` (Linux) file uses the recommended parameters:

   - **enablePerformanceMonitor** – toggles the execution time monitoring of Redis operations. We recommend enabling this feature when debugging and troubleshooting. It is disabled by default as it may affect Creatio performance.
   - **executionTimeLoggingThresholdSec** – Creatio will log Redis operations that took longer than the specified time. By default, “5 seconds”.
   - **clientConnectTimeoutMs** – the time allocated for establishing a network connection to the Redis server. By default, “5000 milliseconds”.
   - **clientSyncTimeoutMs** – the time allocated for executing synchronous Redis operations. By default, “5000 milliseconds”.
   - **clientAsyncTimeoutMs** – the time allocated for executing asynchronous Redis operations. By default, “5000 milliseconds”.
   - **operationRetryIntervalMs** – if the internal retry process for failed operations is unsuccessful, Creatio will postpone the operation for the specified time period. After that, Creatio will run the operation with a new client which may have already established a connection to the new master instance. By default, “5000 milliseconds”.
   - **operationRetryCount** – the number of repeated attempts to run an operation with a new Redis client. By default, “25”.

   These settings must have the following values:

```
<redis connectionStringName="redis" enablePerformanceMonitor="false" executionTimeLoggingThre
```