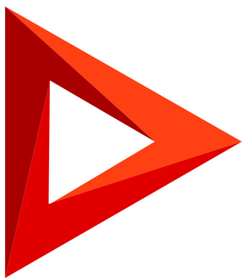
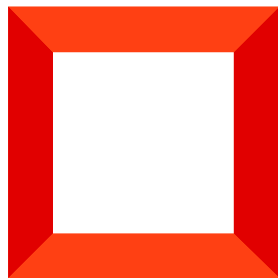
Creatio Academy

Caching server

Version 7.17

This documentation is provided under restrictions on use and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this documentation, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

© 2023 Creatio. All rights reserved.

Table of Contents

General setup procedure for data caching server (Redis) 4
Redis Sentinel 5
Notable Sentinel specifics 5
Minimal fault tolerance requirements for Redis Sentinel 5
Network separation issues 6
System requirements 7
Install and configure Sentinel 8
Configure Creatio to work with Redis Sentinel 8
Redis Cluster 9
Redis Cluster's fault tolerance 10
Redis Cluster system requirements 10
Install and configure Redis Cluster 10
Configure Creatio to work with Redis Cluster 12

© 2023 Creatio. All rights reserved.

General setup procedure for data caching server (Redis)

PRODUCTS: ALL CREATIO PRODUCTS

Use Redis caching server to optimize execution of heavy database queries. Caching improves Creatio performance and reduces the resource usage.

The Redis server package is available in the standard Debian repositories. This article covers installing Redis on Debian and Debian derivatives (such as Ubuntu and Linux Mint). To install Redis:

1. Log in as root:

   ```
   sudo su
   ```

2. Update the package lists:

   ```
   apt-get update
   ```

3. Install Redis:

   ```
   apt-get install redis-server
   ```

4. Enable Redis to run as a **systemd** service. To do this:

   a. Open **redis.conf** in a text editor as root. For example, use the Nano text editor:

      ```
      nano /etc/redis/redis.conf
      ```

   b. Locate the “**supervised no**” entry. Replace the entry with “**supervised systemd**.”

   c. Save changes and exit the editor.

   d. Restart the Redis server.

      ```
      systemctl restart redis-server
      ```

   e. Log out from your root session:

      ```
      exit
      ```

© 2023 Creatio. All rights reserved.

Redis Sentinel

PRODUCTS: ALL CREATIO PRODUCTS

Attention. The deprecated Redis Sentinel mechanism will be retired in Creatio version 7.18.3. We recommend switching to Redis Cluster after updating Creatio to version 7.18.0.

The Redis Sentinel mechanism is used to provide fault tolerance for the Redis repositories used by Creatio. It provides the following features:

- **Monitoring**. Sentinel makes sure that the Master/Slave instances work correctly in Redis.
- **Notifications**. Sentinel alerts system administrators if any instance–related errors occur in Redis.
- **Automatic failover**. If the Redis Master instance is not working correctly, Sentinel promotes one of the Slave instances to a Master, and reconfigures the rest to work with the new Master instance. Creatio is also notified about the new Redis connection address.

Attention. Creatio does not support Redis clusters in version 7.17.4 and earlier.

Redis Sentinel is a distributed system that is designed to run multiple instances that cooperate together. This approach has the following advantages:

- The fault is registered only if multiple Sentinel instances (which form a quorum) agree that the Master instance is unavailable in Redis. This is done to reduce the number of false alerts.
- The Sentinel mechanism will still be available even if multiple Sentinel instances are not responding or do not work altogether. This is done to increase fault tolerance.

Notable Sentinel specifics

- At least three Sentinel instances are required for a robust deployment. These instances should be placed into computers or virtual machines that are believed to fail in an independent way, i. e., the faults registered by these Sentinel instances should be caused by different sources. For example, the computers are located in different network zones.
- Due to asynchronous replication, the distributed system (Sentinel + Redis) does not guarantee that all data will be saved if a failure does occur.
- The fault tolerance of the configuration should be regularly monitored and further confirmed through tests that simulate failures.
- Docker port remapping creates certain issues with Sentinel processes (see the Sentinel, Docker, NAT, and possible issues block of the Sentinel documentation).

Minimal fault tolerance requirements for Redis Sentinel

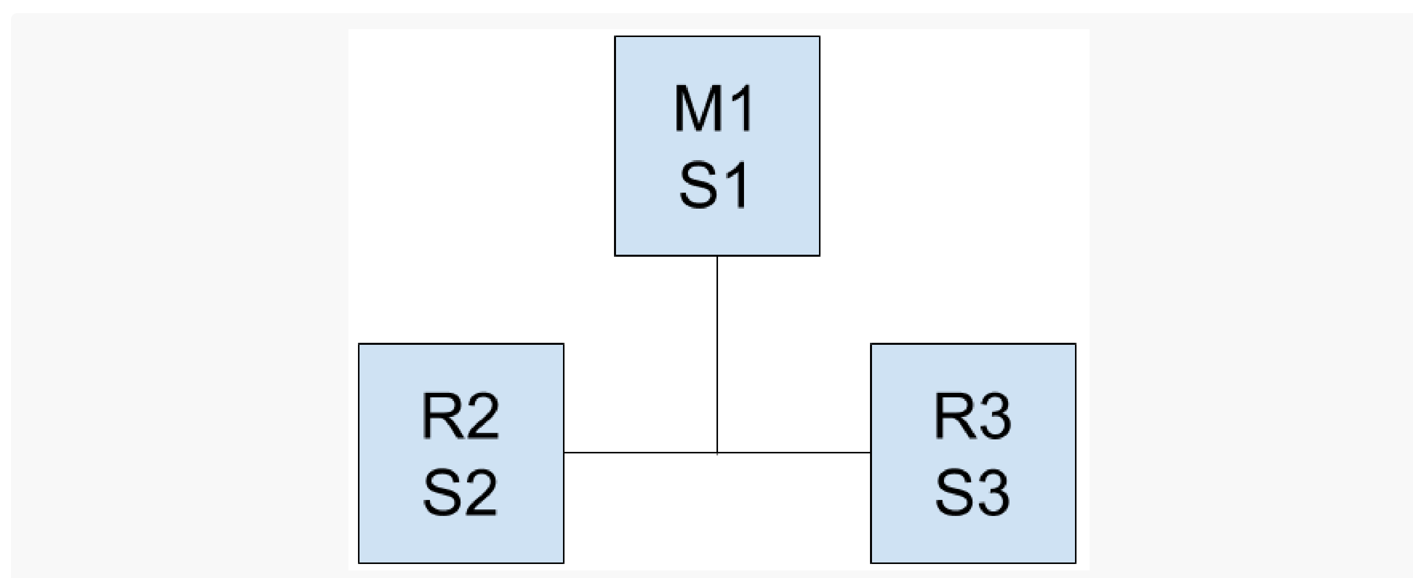
Legend:

© 2023 Creatio. All rights reserved.

- M1, M2 are Redis Master instances.
- R1,R2, R3 are Redis Slave instances.
- S1, S2, S3 are Sentinel instances.
- C1 is Creatio application.
- [M2] is promoted instance (e. g., from Slave to Master).

We recommend using a configuration with at least three Redis and Sentinel instances (see the Example 2: basic setup with three boxes block of the Sentinel documentation). This configuration is based on three nodes (computers or virtual machines), each containing running instances of both Redis and Sentinel (Fig. 1). Two Sentinel instances (S2 and S3) form a quorum (the number of instances required to ensure the fault tolerance of the current Master instance).

Fig. 1 The three nodes configuration: quorum = 2



During regular operation, the Creatio client application writes its data to a Master instance (M1). This data is then replicated asynchronously to the Slave instances (R2 and R3).

If the Redis Master instance (M1) becomes unavailable, the Sentinel instances (S1 and S2) consider this a failure and start the failover process. One of the Redis Slave instances (R2 or R3) is promoted to the Master, enabling the application to use it instead of the previous Master instance.

Attention. There is a risk of losing records in any Sentinel configuration, which uses asynchronous data replication. This occurs if the data were not written to the Slave instance, promoted to a Master.

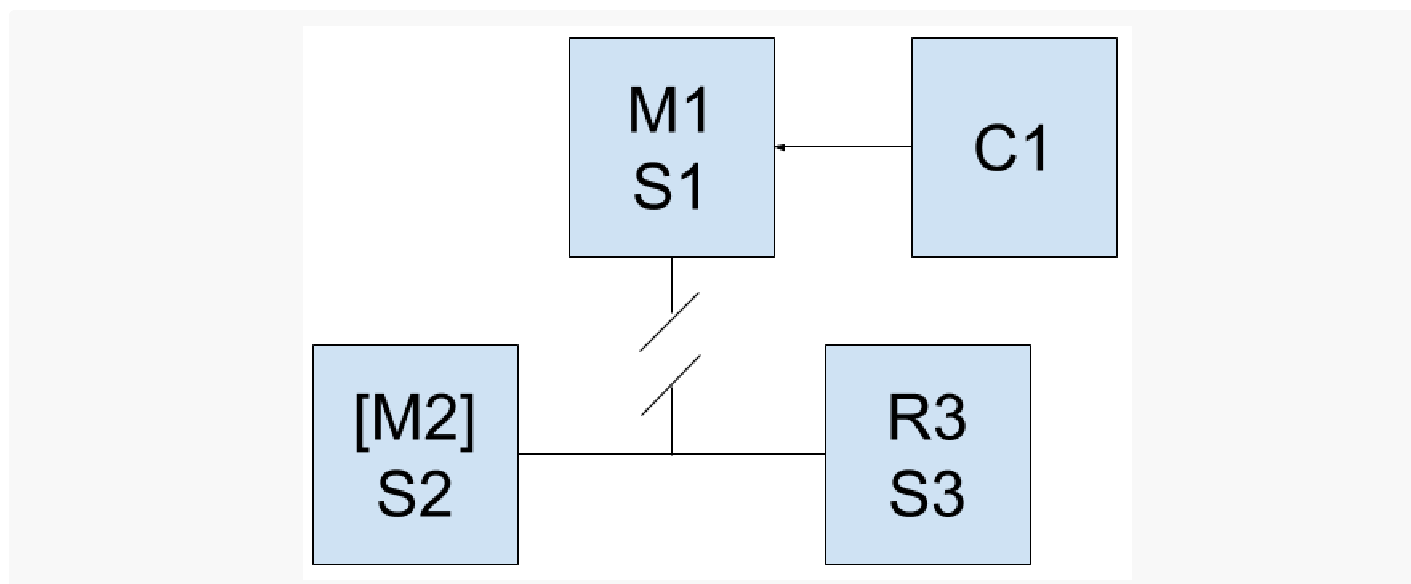
Note. Other possible fault tolerant configurations are described in the Sentinel documentation.

Network separation issues

If the network connection is lost, there is a risk that Creatio will continue to work with the old Redis Master instance (M1), while the newly promoted Master instance ([M2]) has already been assigned (Fig. 2).

© 2023 Creatio. All rights reserved.

Fig. 2 Network separation



This is easily avoided by enabling the option to stop writing data in case the Master instance detects that the number of Slave instances has decreased. To do this, set the following values in the redis.conf configuration file of the Redis Master instance:

```
min-slaves-to-write 1
min-slaves-max-lag 10
```

As a result, if the Redis Master instance (M1) will not be able to transfer data to at least one Slave instance, it will stop receiving data in 10 seconds after the first attempt. Once the system is recovered by Sentinel instances that form a quorum (S2 and S3), the Creatio application (C1) will be reconfigured to work with the new Master–instance (M2).

Attention. If the network is restored, the Master instance will not be able to continue its operation automatically after stopping. If the remaining Redis Slave instance (R3) also becomes unavailable, the system will stop working altogether.

System requirements

Redis is an in-memory database, therefore RAM capacity and performance rate are the main requirements for its correct operation. Since Redis is a single–threaded application that uses a single processor core, a single node (computer or virtual machine) with a dual-core processor is required to work with a single Redis instance. Sentinel instances require relatively few resources and can run on the same node as Redis.

It is recommended to deploy Redis and Sentinel on Linux OS.

The table below shows the recommended system requirements for a single node (computer or virtual machine), depending on the number of Creatio users.

© 2023 Creatio. All rights reserved.

Number of users	CPU	RAM
1–300	Intel Xeon E3-1225v5	2 GB
300–500		4 GB
500–1000		6 GB
1000–3000		12 GB
3000–5000		18 GB
5000–7000		26 GB
7000–10000		36 GB

Install and configure Sentinel

Redis Sentinel comes bundled up with the Redis distribution. The installation process is described in the Redis documentation. We recommend using the latest version of Redis.

Learn more about configuring Sentinel in the Quick tutorial section of the Sentinel documentation.

Configure Creatio to work with Redis Sentinel

Obtain customized libraries and set up ConnectionStrings.config

Contact Creatio support to obtain customized libraries.

Specify the following in the **"redis"** connection string:

- sentinelHosts. The unlimited number of comma-separated addresses and ports of Sentinel instances in the <address>:<port> format.
- masterName. The name of the Redis Master instance.

Connection string example:

```
<add name="redis" connectionString="sentinelHosts=localhost:26380,localhost:26381,localhost:2638
```

Web.config

Make sure the **appSettings** section includes the following parameters:

- Feature-UseRetryRedisOperation enables the internal Creatio mechanism that will retry any Redis operations that ended with errors.

© 2023 Creatio. All rights reserved.

- SessionLockTtlSec – the lifespan of the session lock key.
- SessionLockTtlProlongationIntervalSec – the period for which the lifespan of the session lock key is prolonged.

These settings must have the following values:

```
<add key="aspnet:UseLegacyRequestUrlGeneration" value="true" />
<add key="SessionLockTtlSec" value="60" />
<add key="SessionLockTtlProlongationIntervalSec" value="20" />
```

Please make sure that the **redis** section includes the following parameters:

- enablePerformanceMonitor. Enables the mechanism for monitoring the execution time of Redis operations. We recommend enabling this mechanism for debugging and troubleshooting. Default value: “Off” (enabling this mechanism might affect application performance).
- executionTimeLoggingThresholdSec. If the execution of a Redis operation exceeds this threshold, it will be recorded in the log. By default, “5 seconds.”
- featureUseCustomRedisTimeouts. Enables the use of timeouts specified in the configuration file. Default value: “Off.”
- clientRetryTimeoutMs. All Redis operations that result in errors will be retried using the same client until they reach the timeout specified in this parameter. This parameter is used to eliminate errors caused by short network interruptions. At the same time, getting a new client from the pool is not required. By default, “4000 milliseconds.”
- clientSendTimeoutMs. The time allocated for sending requests to the Redis server. By default, “3000 milliseconds.”
- clientReceiveTimeoutMs. The time allocated for receiving responses from the Redis server. By default, “3000 milliseconds.”
- clientConnectTimeoutMs. The time allocated for establishing a network connection to the Redis server. By default, “100 milliseconds.”
- deactivatedClientsExpirySec. Delay in deleting failed Redis clients after they are removed from the pool. The “0” value represents immediate removal. By default, “0.”
- operationRetryIntervalMs. If the process for retrying a failed operation does not result in successful execution, it will be postponed for the time period specified here. The operation will be performed with a new client, which may have already established a connection to the new Master instance. By default, “1000 milliseconds.”
- operationRetryCount. The number of repeated attempts to perform an operation with a new Redis client. By default, “10.”

These settings must have the following values:

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

Redis Cluster

© 2023 Creatio. All rights reserved.

PRODUCTS: ALL CREATIO PRODUCTS

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

Attention. Redis Cluster is available for Creatio version 7.18.0 and later.

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.

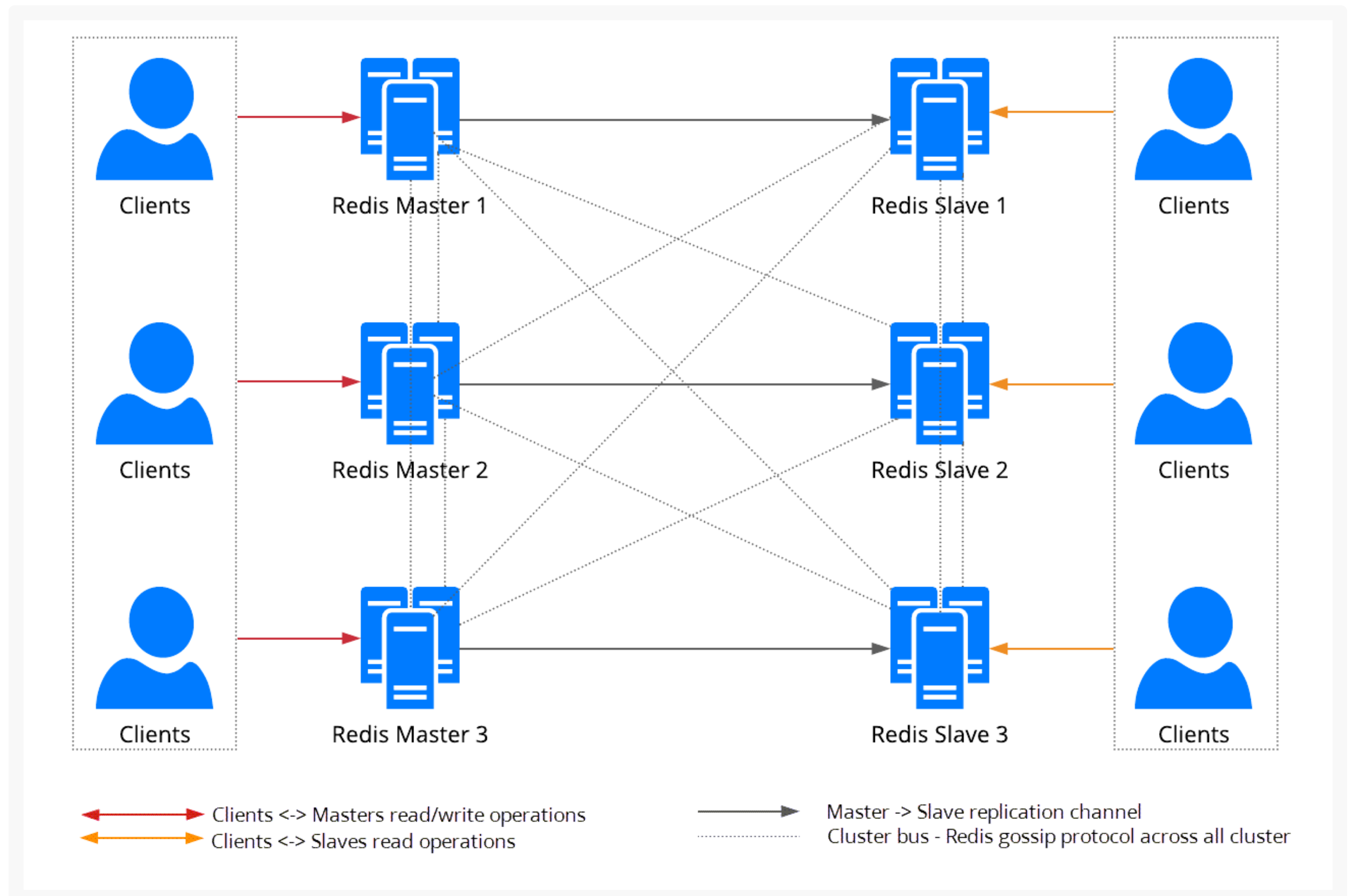
Install and configure Redis Cluster

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).

© 2023 Creatio. All rights reserved.

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:

© 2023 Creatio. All rights reserved.

- **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,ClusterIP2,ClusterIP3,ClusterIP4,
   ```

   where ClusterIP1–ClusterIPn are the IP addresses 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:

© 2023 Creatio. All rights reserved.

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

© 2023 Creatio. All rights reserved.