ALIBABA CLOUD

Alibaba Cloud

ApsaraDB for Redis
Product Introduction

Document Version: 20220712

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.What is ApsaraDB for Redis?

ApsaraDB for Redis is a database service that is compatible with the open source Redis protocol and supports a hybrid of memory and disks for storage. ApsaraDB for Redis provides the hot standby architecture and the cluster architecture, and can scale to meet requirements for high-throughput and low-latency operations.

Benefits

- ApsaraDB for Redis is a fully-managed cloud database service provided by Alibaba Cloud. The service hardware is deployed in the cloud. Alibaba Cloud provides advanced infrastructure planning, network security, and system maintenance. This allows you to focus on business innovation.
- ApsaraDB for Redis supports a variety of data types, including strings, lists, sets, sorted sets, hash tables, and streams. This service also supports advanced features, such as transactions and Pub/Sub commands.
- ApsaraDB for Redis Enhanced Edition (Tair) is an in-memory database service that is developed based on ApsaraDB for Redis Community Edition for enterprises. ApsaraDB for Redis Enhanced Edition (Tair) provides the following instance series: Performance-enhanced instances, Persistent memory-optimized instances, Storage-optimized instances, and Hybrid-storage instances (phased out).

For more information, see Comparison between ApsaraDB for Redis and self-managed Redis and Scenarios.

Purchase methods

Step 1: Create an ApsaraDB for Redis instance

Instance editions

Edition	Description
ApsaraDB for Redis Community Edition	ApsaraDB for Redis Community Edition is compatible with the high-performance in-memory database service of open source Redis and supports master-replica instances, cluster instances, and read/write splitting instances.

Edition	Description
ApsaraDB for Redis Enhanced Edition (Tair)	ApsaraDB for Redis Enhanced Edition (Tair) is developed based on ApsaraDB for Redis Community Edition. ApsaraDB for Redis Enhanced Edition (Tair) provides a variety of instance series based on storage media such as DRAM, NVM, and enhanced SSDs (ESSDs) to meet your requirements for low-latency access, persistence, and reduced overall costs. For more information about ESSDs, see ESSDs. ApsaraDB for Redis Enhanced Edition (Tair) provides higher performance, more data structures, and more flexible storage methods to meet your business requirements in different scenarios. • Performance-enhanced instances: Performance-enhanced instances adopt a multi-threading model and integrate multiple features from Tair of Alibaba Group. These instances support a variety of Tair data structures and are highly suitable for specific scenarios. • Persistent memory-optimized instances: Persistent memory-optimized instances adopt Intel Optane DC persistent memory (AEP) to provide large-capacity in-memory databases that are compatible with open source Redis. A persistent memory-optimized instance does not use disks to implement data persistence. It provides almost the same performance as an ApsaraDB for Redis Community Edition instance in terms of throughput and latency while maintaining persistence of each operation. This helps improve the reliability of business data. • Storage-optimized instances: Storage-optimized instances are developed based on ESSDs and are compatible with core data structures and interfaces of open source Redis. These instances can provide large-capacity, low-cost, and persistent database services. These instances are suitable for scenarios that store warm and cold data and that require compatibility with open source Redis, large capacity, and high access performance.

Instance architectures

ApsaraDB for Redis supports multiple deployment architectures that are suitable for different scenarios.

Architecture	Description
Standard master-replica instances	Data is synchronized from the master node to replica nodes in real time. If the master node fails, workloads are switched from the master node to a replica node. This process is automated and does not affect workloads to ensure high availability.
Cluster master-replica instances	Cluster instances use a distributed architecture. Each shard supports the master-replica architecture to provide automatic disaster recovery and failover and ensure high availability. Multiple cluster specifications are available. You can determine the specifications based on your business requirements. The cluster architecture supports the following connection modes: • Proxy mode is the default connection mode of a cluster instance. This mode supports automatic connections and reduces application development costs. For more information about the proxy mode, see Proxy mode. • Direct connection mode allows a client to bypass proxy nodes and directly access backend shards to reduce the network overhead and service response time. This mode is suitable for business scenarios that are latency-sensitive. For more information about the direct connection mode, see Direct connection mode.

Architecture	Description
Read/write splitting instances	Read/write splitting instances use the master-replica architecture to provide high availability. Read replicas are attached to the master node to facilitate data replication and implement linear scaling of read performance. Read replicas can alleviate performance issues caused by hotkeys. Read/write splitting instances are suitable for business scenarios that feature high read/write ratios. Read/write splitting instances are available in cluster and non-cluster types: • A non-cluster read/write splitting instance supports one, three, or five read replicas. • For a cluster read/write splitting instance, a read replica is attached to each shard to achieve automatic read/write splitting on individual shards. Cluster read/write splitting instances are suitable for ultra-large-scale business scenarios that feature high read/write ratios.

Instance specifications

ApsaraDB for Redis Community Edition and Enhanced Edition (Tair) provide different instance specifications for each architecture. For more information, see Overview.

2.Overview and selection of ApsaraDB for Redis

2.1. Select ApsaraDB for Redis instances

When you create an ApsaraDB for Redis instance, you must select the most cost-effective and stable instance type that can suit your performance, price, scenario, and workload needs. This topic describes the instance types, editions, architectures, and storage media of ApsaraDB for Redis to help you select instances.

Overview of ApsaraDB for Redis

ApsaraDB for Redis is a database service that is compatible with the open source Redis protocol and supports a variety of storage media. ApsaraDB for Redis provides the hot standby architecture and the cluster architecture, and can scale to meet requirements for high-throughput and low-latency operations.

Selection procedure

To create an ApsaraDB for Redis instance, you must select the instance type and specifications that can suit your performance, price, scenario, and workload needs. For example, you need to select different instance types and specifications based on whether you use the instance as a or an The following table describes the recommended procedure for selecting ApsaraDB for Redis instances.high-speed cachein-memory database

> **Note** For more information about the pricing of ApsaraDB for Redis, see Billable items and prices.

Step	Description
Select ApsaraDB for Redis Community Edition or Enhanced Edition (Tair)	ApsaraDB for Redis provides Community Edition and Enhanced Edition (Tair). ApsaraDB for Redis Enhanced Edition (Tair) is developed based on Tair to provide higher performance, more data structures, and more flexible storage methods. Tair is an internal service of Alibaba Cloud.
Select a deployment architecture	ApsaraDB for Redis provides instances that use the standard architecture, cluster architecture, or read/write splitting architecture. You can select an architecture that suits your business data volume and your requirements for read and write capabilities and business performance. For more information about the standard, cluster, and read/writing architectures, see Standard master-replica instances, Cluster master-replica instances, and Read/write splitting instances.
Select a disaster recovery solution	An ApsaraDB for Redis instance may fail due to unexpected reasons, such as a device failure or a power failure in a data center. In this case, disaster recovery can help ensure data consistency and service availability. ApsaraDB for Redis provides a variety of disaster recovery solutions to meet the requirements in different business scenarios.
Select a major version	We recommend that you use the latest major version that boasts more features.

Step	Description
Estimate the memory size	We recommend that you estimate the size of memory to be consumed. This reduces costs and prevents your business from being affected by frequent specification changes.
Create an ApsaraDB for Redis instance	After you determine the instance type and specifications, create an ApsaraDB for Redis instance by using the ApsaraDB for Redis console or an API operation of ApsaraDB for Redis.
Verify and adjust the selected instance	After you create and start to use an ApsaraDB for Redis instance, we recommend that you monitor the performance of the instance when your business is running as expected. This allows you to check whether the type and specifications of the instance are suitable.

Select ApsaraDB for Redis cloud disk-based instances or local disk-based instances

ApsaraDB for Redis local disk-based instances provide a complete set of features. ApsaraDB for Redis cloud disk-based instances use a cloud native architecture and support imperceptible scaling. This allows you to specify a custom number of shards to scale a cluster instance without affecting your business. In the future, more and more ApsaraDB for Redis instances will use cloud disks. The following table compares these two types of ApsaraDB for Redis instance.

Item	Instance that uses local disks	Instance that uses cloud disks
Architecture	The instances that use local disks are developed based on the traditional management architecture of ApsaraDB for Redis.	The instances that use cloud disks are developed based on the new-generation management architecture of ApsaraDB for Redis. **Note** New types of instances will evolve based on this architecture. ApsaraDB for Redis allows you to migrate data from an instance that uses local disks to an instance that uses cloud disks.
Scale-out	• A scale-out consumes more time. • Transient connections occur when you scale out a cluster instance. • In a scale-out, you can only double the number of shards in a cluster instance. For example, if the original cluster instance has two shards, you can scale the instance only to four shards.	• A scale-out consumes less time. • Transient connections do not occur when you scale out a cluster instance. • You can scale shards in a cluster instance to a number that is no less than one and scale up or down a single shard. This allows the instance to better cope with read/write hotspots and skewed requests.

Select ApsaraDB for Redis Community Edition or Enhanced Edition (Tair)

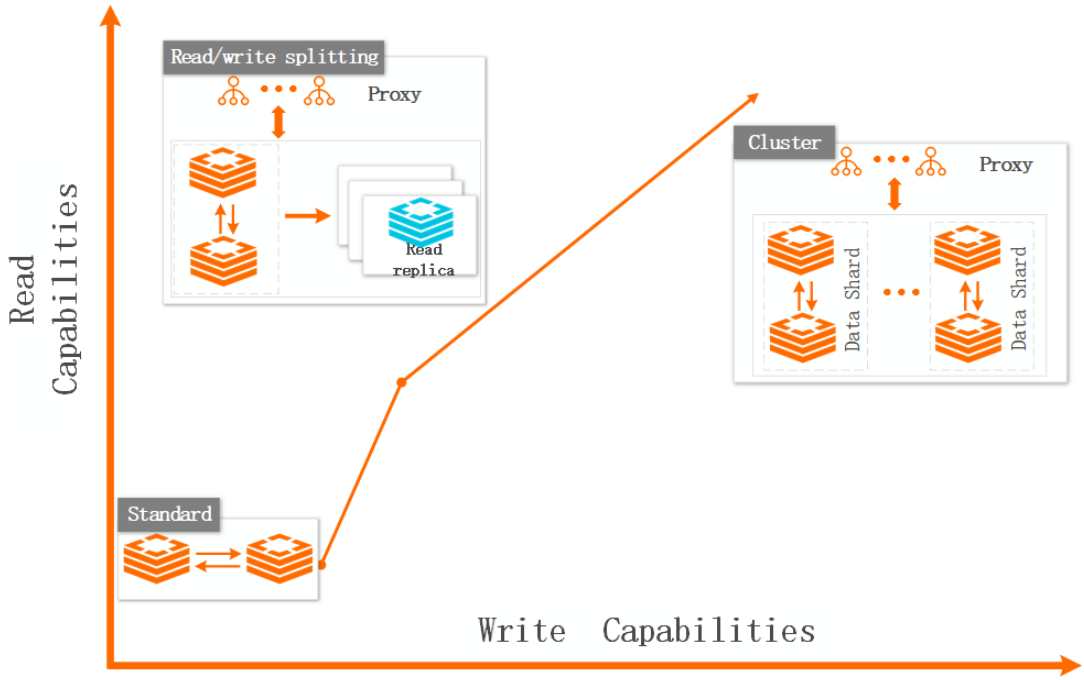
ApsaraDB for Redis provides Community Edition and Enhanced Edition (Tair). ApsaraDB for Redis Enhanced Edition (Tair) is an in-memory database service that is developed based on Tair. Tair is an internal service of Alibaba Cloud. ApsaraDB for Redis Enhanced Edition (Tair) provides a variety of series of instances based on storage media such as DRAM, NVM, and enhanced SSDs (ESSDs) to meet your requirements for low-latency access, persistence, and reduced overall costs. For more information about ESSDs, see ESSDs. ApsaraDB for Redis Enhanced Edition (Tair) provides you with higher performance, more data structures, and more flexible storage methods. This helps you meet business requirements in different scenarios.

> **Notice**
>
> - ApsaraDB for Redis Enhanced Edition (Tair) supports all features of Community Edition. In addition, ApsaraDB for Redis Enhanced Edition (Tair) provides advanced features such as data flashback, proxy query cache, and global distributed cache. For more information about the comparison between Community Edition and Enhanced Edition (Tair), see Feature comparison. For more information about data flashback, proxy query cache, and global distributed cache, see Use data flashback to restore data by point in time, Use proxy query cache to address issues caused by hotkeys, and Overview.
> - For more information about the commands and parameters that are supported by Community Edition and Enhanced Edition (Tair), see Overview and Modify parameters of an instance.

Select a deployment architecture

ApsaraDB for Redis instances support three deployment architectures. You can select an architecture that suits your business data volume and your requirements for read and write capabilities and business performance.

Comparison among deployment architectures



Note

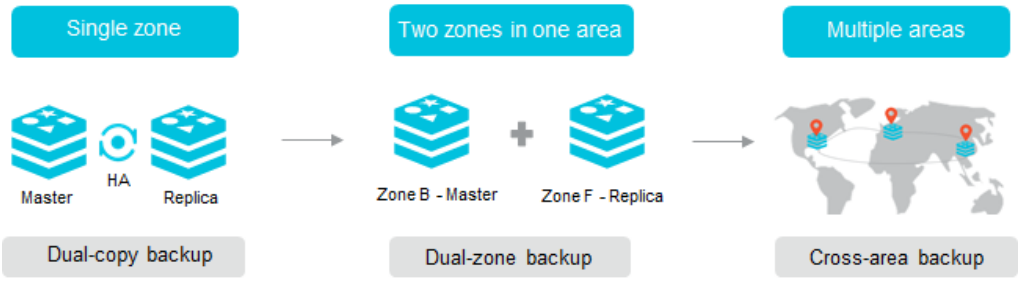
- By default, cluster instances and read/write splitting instances provide proxy endpoints. Requests of clients are routed to data shards by proxy nodes. Proxy nodes help implement features such as load balancing, read/write splitting, failover, proxy query cache, and persistent connection. Only performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) support proxy query cache. For more information about proxy query cache, see Use proxy query cache to address issues caused by hotkeys. For more information about performance-enhanced instances, see Performance-enhanced instances. For more information, see Features of proxy nodes.
- If you use a cluster instance, you can use a private endpoint to bypass proxy nodes. This way, you can directly access backend data shards. This is similar to the connection to a native Redis cluster. For more information about private endpoints, see Enable the direct connection mode. Compared with the proxy mode, the direct connection mode reduces the routing time and accelerates the response of ApsaraDB for Redis.

Architecture	Description	Scenario
Standard	Standard instances adopt a master-replica architecture. The master node serves your workloads, whereas the replica node stays in hot standby mode to ensure high availability. If the master node fails, the system switches the workloads to the replica node within 30 seconds after the failure occurs. This ensures the high availability of your instance.	• Support for more native Redis features. • Persistent storage in ApsaraDB for Redis instances. • Stable query rate on a single node of ApsaraDB for Redis. • Use of simple Redis commands, when only a few sorting and computing commands are required.
Cluster	• A cluster instance contains proxy nodes, data shards, and config servers. You can scale out a cluster instance by increasing the number of data shards. • A cluster master-replica instance contains multiple data shards. Each data shard works in a high-availability architecture in which a master node and a replica node are deployed on different devices. If the master node is faulty, the master-replica cluster instance fails over to the replica node to ensure high service availability.	• Large data volume. • High queries per second (QPS). • High throughput and high performance.

Architecture	Description	Scenario
Read/write splitting	• A read/write splitting instance contains proxy nodes, master and replica nodes, and read replicas. • Read replicas support chained replication. This allows you to scale out read replicas to increase the read capacity.	• High QPS scenarios caused by a large number of read requests, such as data hotspots. • Support for more native Redis features. You can use read/write splitting instances to overcome the limits of cluster instances. For more information, see Limits on commands supported by cluster instances. **Note** Data synchronization to read replicas has a latency. Therefore, in scenarios that require high data consistency, we recommend that you use cluster instances instead of read/write splitting instances.

Select a disaster recovery solution

Roadmap of disaster recovery for ApsaraDB for Redis



Disaster recovery solution	Protection level	Description
Single-zone HA solution	★★★☆☆	The master node and replica node are deployed on different devices in the same zone. If the master node fails, the high-availability system performs a failover to prevent service interruption caused by a single point of failure (SPOF).
Zone-disaster recovery solution	★★★★☆	The master node and replica node are deployed in two different zones in the same region. If the zone in which the master node resides is disconnected due to force majeure factors such as a power or network failure, the high-availability system performs a failover to ensure continuous availability of the entire instance.

Disaster recovery solution	Protection level	Description
Cross-region disaster recovery solution	★★★★★	In the architecture of Global Distributed Cache for Redis, a distributed instance consists of multiple child instances that synchronize data among each other in real time by using synchronization channels. The channel manager monitors the health status of child instances and handles exceptions that occur on child instances, such as a switchover between the primary and secondary databases. Global Distributed Cache for Redis is suitable for scenarios such as geo-disaster recovery, active geo-redundancy, nearby application access, and load balancing. For more information, see Overview.

Select a major version

Select a major version of ApsaraDB for Redis based on your business requirements. All major versions are regularly maintained. You can use the latest major version that boasts more features. For more information, see New features of ApsaraDB for Redis 6.0, New features of ApsaraDB for Redis 5.0, and Features of ApsaraDB for Redis 4.0.

You must consider the following limits when you select a major version.

Instance disk type and creation method	Supported instance edition	Supported engine version	Supported architecture
Local disk-based instance Create Redis本地盘实例 an ApsaraDB for Redis Community Edition instance or a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair)	ApsaraDB for Redis Community Edition	4.0 5.0	Cluster architecture Standard architecture Read/write splitting architecture
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	5.0	Cluster architecture Standard architecture Read/write splitting architecture
Cloud disk-based instance Create an ApsaraDB for Redis instance	ApsaraDB for Redis Community Edition	7.0 6.0 5.0	Standard architecture Cluster architecture
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	5.0	Standard architecture Cluster architecture
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	6.0	Standard architecture Cluster architecture

Instance disk type and creation method	Supported instance edition	Supported engine version	Supported architecture
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	4.0	Standard architecture

Estimate the memory size

To create an ApsaraDB for Redis instance, you must estimate the size of memory to be consumed based on the following factors and select the most suitable memory specifications. This reduces costs, prevents your business from being affected by frequent specification changes, and allows you to migrate your business to the cloud at a faster pace.

> **Notice** When you determine the memory size of an ApsaraDB for Redis instance, you must first consider the volume of the business data to be stored. In addition, you must take into account the memory required for the operation of the instance itself, such as the memory occupied by the process metadata, replication buffer, and fragments.
>
> For self-managed Redis databases, you must consider the memory overhead caused by the write-time replication of forks and advanced features such as whitelist configuration, audit logging, large key management, and hotkey management. If you use ApsaraDB for Redis instances, the preceding memory overhead is borne by Alibaba Cloud and is not included in the memory size of the purchased instance.

- The data types, lengths, and quantity of keys.

  > **Note** If a key, such as a key of the hash type, contains elements, you must consider the quantity and lengths of these elements.

- The lengths of values.
- The expiration time and eviction policy of each key. For more information about eviction policies, see How does ApsaraDB for Redis evict data by default?
- The access models. For example, if the access model involves a large number of client connections, Lua scripts, and transactions, you must reserve memory for them.
- Medium- and long-term business growth.

Create an ApsaraDB for Redis instance

After you determine the instance type and specifications, you can create an ApsaraDB for Redis instance by using the ApsaraDB for Redis console or an API operation of ApsaraDB for Redis. For more information, see the following topics:

- ApsaraDB for Redis console: Create an ApsaraDB for Redis instance
- API operation: CreateInstance

Verify and adjust the selected instance

ApsaraDB for Redis supports a variety of metrics. After you create and start to use an ApsaraDB for Redis instance, we recommend that you monitor the performance of the instance when your business is running as expected. This allows you to check whether the instance type and specifications of the instance are suitable. For more information, see View monitoring data.

> **Note** You can also use the Redis-benchmark tool to perform a performance stress test. For more information, see Description of redis-benchmark.

For example, if the performance monitoring data of an ApsaraDB for Redis instance indicates high memory usage, you must check whether an exception exists. If no exception exists, upgrade the instance specifications. For more information, see Change the configurations of an instance. For more information about how to troubleshoot performance issues for ApsaraDB for Redis instances, see the following topics:

- Troubleshoot high CPU utilization on an ApsaraDB for Redis instance
- Troubleshoot the high memory usage of an ApsaraDB for Redis instance
- Troubleshoot high traffic usage on an ApsaraDB for Redis instance

Related information

- Development and O&M standards for ApsaraDB for Redis

2.2. Comparison between ApsaraDB for Redis and self-managed Redis

Compared with self-managed Redis databases deployed on your servers, ApsaraDB for Redis instances have multiple benefits, such as high data security, easy O&M, and optimized kernel.

Item	ApsaraDB for Redis	Self-managed Redis
Security protection	Beforehand prevention: • Uses virtual private clouds (VPCs) to build isolated network environments. • Uses whitelists for access control. For more information, see Step 2: Configure whitelists. • Supports custom accounts and permissions. For more information, see Create and manage database accounts.	Beforehand prevention: • Requires a self-managed network security system. It is expensive and difficult to build such a system. • Comes with risks of data leaks due to security vulnerabilities in the default access configuration of open source Redis. • Does not have an account authentication system.
	In-process protection: implements SSL encryption. For more information, see Configure SSL encryption for an ApsaraDB for Redis instance.	In-process protection: requires a third-party tool to implement SSL encryption.
	Post audit: supports audit logs. For more information, see Query audit logs.	Post audit: does not provide the auditing feature.

Item	ApsaraDB for Redis	Self-managed Redis
Backup and restoration	Performance-enhanced instances of ApsaraDB for Redis Enhanced Edition (Tair) support the data flashback feature that allows you to restore data to a specific point in time. For more information about performance-enhanced instances, see Performance-enhanced instances. For more information about the data flashback feature, see Use data flashback to restore data by point in time.	Only full data restoration is supported.
O&M	• Supports more than ten groups of metrics and a minimum monitoring frequency of 5 seconds. For more information, see Metrics. • Supports alert settings. For more information, see Alert settings. • Allows you to create instances of different architectures based on your business requirements and change specifications. • Supports big key analytics based on snapshots to ensure high accuracy and prevent performance loss. For more information, see Offline key analysis.	• Requires a more sophisticated third-party monitoring tool to monitor services. • Stops services when you change the specifications or architecture. The operation to change the specifications or architecture is complex. • Supports big key analytics based on sampling, which provides limited precision.
Deployment and scaling	Supports instant activation and elastic scaling.	Requires a long period of time to complete hardware procurement, hosting of data centers, and machine deployment. You must manage node relationships when you add nodes.
High availability (HA)	• Single-zone HA solution. • Zone-disaster recovery solution. • Uses an independent and central module to ensure high availability. This module ensures stable and efficient decision-making and helps prevent split-brain issues.	• Allows you to deploy a high-availability architecture in Sentinel mode in a data center. • Allows you to deploy a zone-disaster recovery architecture in Sentinel mode. • Uses the Sentinel mode to ensure high availability. The decision-making efficiency is low during peak hours and the cost is high. Split-brain issues may occur and affect your services.

Item	ApsaraDB for Redis	Self-managed Redis
Kernel optimization	• Provides performance-enhanced instances based on the multi-threading model. For more information, see Performance-enhanced instances. The performance of a performance-enhanced instance is three times that of a standard instance with the same specifications. • Provides storage-optimized and persistent memory-optimized instances to support large-capacity storage and command-level persistence. For more information about storage-optimized and persistent memory-optimized instances, see Storage-optimized instances and Persistent memory-optimized instances.	• Clusters of Redis 6.0 or later support multiple I/O threads to enhance performance. The performance of a Redis cluster can be increased by up to twice the original performance. However, the CPU utilization is high. • Uses a storage service such as SSDB or Pika as the persistent storage. However, these storage services are not fully compatible with the Redis protocol, and they can manage hot and cold data only at the key level. It is expensive to transfer big keys between the memory and disks in these storage services, and these services are also difficult to manage.

Note ApsaraDB for Redis is fully compatible with native Redis. For more information, see Which version of Redis is ApsaraDB for Redis compatible with? The way to connect to a database is basically the same for ApsaraDB for Redis and native Redis. You can choose a client application that is compatible with the Redis protocol to connect to the database. For more information, see Use a client to connect to an ApsaraDB for Redis instance.

2.3. Comparison between ApsaraDB for Redis Enhanced Edition (Tair) and ApsaraDB for Redis Community Edition

This topic compares the features of ApsaraDB for Redis Enhanced Edition (Tair) and Community Edition to help you select an instance type that meets your business requirements.

Key features and supported scenarios

Edition	Series	Feature	Scenario

Edition	Series	Feature	Scenario
	Performance-enhanced instances	• Performance-enhanced instances use the multi-threading model and provide read and write performance approximately three times that of ApsaraDB for Redis Community Edition instances with the same specifications. • Performance-enhanced instances provide multiple enhanced data structure modules such as TairString (including CAS and CAD commands), TairHash, TairGIS, TairBloom, TairDoc, TairTS, TairCpc, TairZset, TairRoaring, and TairSearch. These instances eliminate concerns about storage structure and timeliness and allow you to focus on application development. • Performance-enhanced instances provide high compatibility. Performance-enhanced instances are fully compatible with open source Redis. You can switch from open source Redis to ApsaraDB for Redis without the need to modify application code. • Performance-enhanced instances provide a variety of enterprise-grade features such as data flashback, proxy query cache, and Global Distributed Cache for Redis. For more information, see Use data flashback to restore data by point in time, Use proxy query cache to address issues caused by hotkeys, and Overview.	Performance-centric business scenarios.

Edition	Series	Feature	Scenario
ApsaraDB for Redis Enhanced Edition (Tair)	Persistent memory-optimized instances	• Persistent memory-optimized instances provide ultra-high cost-effectiveness. The price of persistent memory-optimized instances is 30% lower than that of ApsaraDB for Redis Community Edition instances with the same capacity. The performance of persistent memory-optimized instances reaches 90% of that of native Redis databases. • Persistent memory-optimized instances support the TairString (including CAS and CAD commands) and TairCpc enhanced data structure modules. • Persistent memory-optimized instances prevent data loss when power failures occur. These instances implement persistence for each command. The system will return a success response for each write operation only after the data is persistently stored. You can use persistent memory-optimized instances as in-memory databases instead of caches. • Persistent memory-optimized instances optimize the append-only file (AOF) rewriting performance for large-sized Redis databases. These instances reduce the latency and jitters that are caused when Redis calls forks to rewrite the AOF. • Persistent memory-optimized instances deliver high compatibility. These instances are compatible with most data structures and commands of native Redis databases.	Data cache and storage scenarios that require high performance and high data persistence, and can bear high costs.

Edition	Series	Feature	Scenario
	Storage-optimized instances	• Storage-optimized instances reduce costs. These instances reduce up to 85% of costs compared with ApsaraDB for Redis Community Edition instances. • Storage-optimized instances store data in disks. These instances store data in enhanced SSDs (ESSDs) with high data reliability. The capacity of a storage-optimized instance reaches hundreds of terabytes. For more information about ESSDs, see ESSDs. • Storage-optimized instances optimize memory usage for large-sized Redis databases. These instances reduce the amount of reserved memory that is required for the forks of native Redis databases. • Storage-optimized instances deliver high compatibility. Storage-optimized instances are compatible with most data structures and commands of native Redis databases.	Data storage scenarios that require a large capacity and low costs, involve only infrequent data access, and can bear high access latency.
ApsaraDB for Redis Community Edition	None	ApsaraDB for Redis Community Edition instances are compatible with open source Redis databases and provides high performance.	Small-sized applications, medium-sized applications, applications for verification, and standard Redis usage and data migration scenarios.

Note For more information about instance types, see Select ApsaraDB for Redis instances.

Feature comparison

In the following table, ☺✔☺ indicates that this feature is supported, and □ indicates that this feature is not supported.

Note Hybrid-storage instances are discontinued. For more information, see Sales of ApsaraDB for Redis hybrid-storage instances are discontinued. We recommend that you use performance-enhanced instances that provide higher performance and more features.

If you have purchased s hybrid-storage instance, you can submit a ticket to migrate the data of the instance.

Category	Item	ApsaraDB for Redis Enhanced Edition (Tair)		
		Performance-enhanced instances	Hybrid-storage instances (phased out)	Redis 2.8, Redis 4.0, and Redis 5.0
Basic performance	Performance benchmark (based on Community Edition)	300%	90% to 40% ②	Same
	Maximum number of connections to each data node	30,000	10,000	10,000
		450,000	120,000 to 60,000 ②	140,000
		Local disks		
		ESSDs (PL1)		

Category Specifications	Item	ApsaraDB for Redis Enhanced Edition (Tair)		
	Disk type		Local disks	Local disks
		Performance-enhanced instances	Hybrid-storage instances (phased out)	Redis 2.8, Redis 4.0, and Redis 5.0
	Threading model	Multiple I/O threads+ single worker thread (the Real Multi-I/O model) ③	Single I/O thread + multiple worker threads	Single I/O thread + single worker thread
	Cost per unit (based on Community Edition)	117%	30%	Same

Category	Item	ApsaraDB for Redis Enhanced Edition (Tair)		
		Performance-enhanced instances	Hybrid-storage instances (phased out)	Redis 2.8, Redis 4.0, and Redis 5.0
Data structure	Basic data structures and supported commands	Different instances support different commands. For more information, see Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair).		

Category	Item	ApsaraDB for Redis Enhanced Edition (Tair)		
		Performance-enhanced instances	Hybrid-storage instances (phased out)	Redis 2.8, Redis 4.0, and Redis 5.0
	Integration with multiple Redis modules	✔☺	□	□
Data persistence	Master-replica replication consistency	Eventual consistency	Eventual consistency	Eventual consistency
	Persistent data consistency ④	Write Back	Write Back	Write Back
	Persistence level	Within seconds	Within seconds	Within seconds
	Enable the new audit log feature	✔☺	□	✔☺

Category	Item	ApsaraDB for Redis Enhanced Edition (Tair)		
Security		Performance-enhanced instances	Hybrid-storage instances (phased out)	Redis 2.8, Redis 4.0, and Redis 5.0
	Configure SSL encryption	✔	✔	✔
	Step 2: Configure whitelists	✔	✔	✔
	Use the real-time key statistics feature	✔	□	✔
	Query historical hotkeys	✔	□	✔
	Enable key analysis	✔	□	✔ (not supported in Redis 2.8)
	Offline key analysis	✔	□	✔

Performance analysis		ApsaraDB for Redis Enhanced Edition (Tair)		
Category	Item	Performance-enhanced instances	Hybrid-storage instances (phased out)	Redis 2.8, Redis 4.0, and Redis 5.0
	Use data flashback to restore data by point in time	✔	□	□
	Use proxy query cache to address issues caused by hotkeys	✔	□	□
	Overview	✔	□	□
	Configure one-way data synchronization between ApsaraDB for Redis instances	✔	✔	✔

Advanced features		ApsaraDB for Redis Enhanced Edition (Tair)		
Category	Item	Performance-enhanced instances	Hybrid-storage instances (phased out)	Redis 2.8, Redis 4.0, and Redis 5.0
	Configure two-way data synchronization between ApsaraDB for Redis Enhanced Edition (Tair) instances	✔	□	□

The following sections show the description of each numeric label:

- ①: The QPS reference value is measured by a command with a time complexity of O(1). A higher time complexity indicates a lower QPS reference value.
- ②: This metric is related to the distribution of cold and hot data that is accessed. A higher hit rate on memory indicates that hybrid-storage instances provide higher performance that is closer to that of Community Edition.
- ③: Different from the multi-threading model of Community Edition 6.0, the Real Multi-I/O model of performance-enhanced instances provides fully accelerated I/O threads to support multiple connections and linearly increases throughput.

- ④: ApsaraDB for Redis uses the following methods to store data:
  - Write Through: writes data directly to disks and returns a success response.
  - Write Back: writes data to the cache and returns a success response, and then writes the data to disks.

Related information

- Overview
- Overview
- Overview

2.4. Comparison between ApsaraDB for Redis instances that use local disks and those that use cloud disks

You can purchase an ApsraDB for Redis instance that uses local disks or cloud disks based on your needs. This topic compares ApsraDB for Redis instances that use local disks and those that use cloud disks.

Overview

Item	Instance that uses local disks	Instance that uses cloud disks
Architecture	The instances that use local disks are developed based on the traditional management architecture of ApsaraDB for Redis.	The instances that use cloud disks are developed based on the new-generation management architecture of ApsaraDB for Redis. **Note** New types of instances will evolve based on this architecture. ApsaraDB for Redis allows you to migrate data from an instance that uses local disks to an instance that uses cloud disks.
Scale-out	• A scale-out consumes more time. • Transient connections occur when you scale out a cluster instance. • In a scale-out, you can only double the number of shards in a cluster instance. For example, if the original cluster instance has two shards, you can scale the instance only to four shards.	• A scale-out consumes less time. • Transient connections do not occur when you scale out a cluster instance. • You can scale shards in a cluster instance to a number that is no less than one and scale up or down a single shard. This allows the instance to better cope with read/write hotspots and skewed requests.

Feature comparison

An ApsaraDB for Redis instance that uses local disks provides a complete suite of features. An ApsaraDB for Redis instance that uses cloud disks adopts the cloud native architecture. An ApsaraDB for Redis cluster instance supports the In the future, more and more ApsaraDB for Redis instances will use cloud disks. In the following table, ✔☺ indicates that this feature is supported, □ indicates that this feature is not supported, and □ indicates that this feature is not applicable. imperceptible scaling

Item	Feature	Local disk			Cloud disk	
		Standard architecture	Cluster architecture	Read/write splitting architecture	Standard architecture	Cluster architecture
Lifecycle management	Change the configurations of an instance	✔☺	✔☺	✔☺	✔☺ Imperceptible scaling	✔☺ Imperceptible scaling Support a custom number of shards. For more information, see Adjust the number of shards for an ApsaraDB for Redis instance.
	Restart one or more ApsaraDB for Redis instances	✔☺	✔☺	✔☺	✔☺	✔☺
	Change the billing method to subscription	✔☺	✔☺	✔☺	□	□
	Renew an instance	✔☺	✔☺	✔☺	✔☺	✔☺
	Specify a custom number of shards	□	□	□	□	✔☺
	Adjust the number of shards for an ApsaraDB for Redis instance	□	□	□	□	✔☺
	Upgrade the major version	✔☺	✔☺	✔☺	□	□
	Update the minor version	✔☺	✔☺	✔☺	✔☺	✔☺

Item	Feature	Local disk			Cloud disk	
		Standard architecture	Cluster architecture	Read/write splitting architecture	Standard architecture	Cluster architecture
	Release 或退订 instances	✔	✔	✔	✔	✔
	Manage instances in the recycle bin	✔	✔	✔	✔	✔
Network connection management	Change the VPC or vSwitch of an ApsaraDB for Redis instance	✔	✔	✔	□	□
	Apply for a public endpoint for an ApsaraDB for Redis instance	✔	✔	✔	✔	✔
	Enable the direct connection mode	□	✔	□	□	✔
	Change the endpoint or port number of an ApsaraDB for Redis instance	✔	✔	✔	✔	✔
Bandwidth management	开启带宽弹性伸缩	✔	✔	✔	□	□
	Adjust the bandwidth of an ApsaraDB for Redis instance	✔	✔	✔	□	□
High availability (HA) management	Manually switch workloads from a master node to a replica node	✔	✔	✔	□	□
	Restart or rebuild a proxy node	□	✔	✔	□	□
	Upgrade proxy nodes	□	✔	✔	□	□
Parameter management	Modify parameters of an instance	✔	✔	✔	✔	✔

Item	Feature	Local disk			Cloud disk	
		Standard architecture	Cluster architecture	Read/write splitting architecture	Standard architecture	Cluster architecture
Tag management	Create tags	✔	✔	✔	✔	✔
	Filter ApsaraDB for Redis instances by tag	✔	✔	✔	✔	✔
	Remove or delete tags	✔	✔	✔	✔	✔
Other management	Set a maintenance window	✔	✔	✔	✔	✔
	Migrate an instance across zones	✔	✔	✔	✔	□
	Export the instance list	✔	✔	✔	✔	✔
Security management	Create and manage database accounts	✔	✔	✔	✔	✔
	Change or reset the password	✔	✔	✔	✔	✔
	Configure whitelists	✔	✔	✔	✔	✔
	Add security groups	✔	✔	✔	✔	□
	Configure SSL encryption	□	✔	□	□	□
	Enable password-free access	✔	✔	✔	✔	✔
	Enable the release protection feature	✔	✔	✔	□	□
	View monitoring data	✔	✔	✔	✔	✔
	Customize metrics (previous version)	✔	✔	✔	✔	✔

Item	Feature	Local disk			Cloud disk	
		Standard architecture	Cluster architecture	Read/write splitting architecture	Standard architecture	Cluster architecture
Performance monitoring	Modify the data collection interval (previous version)	✔	✔	✔	✔	✔
	Alert settings	✔	✔	✔	✔	✔
	Performance trends	✔	✔	✔	✔	✔
	View performance metrics in real time	✔	✔	✔	✔	✔
	Instance sessions	✔	✔	✔	✔	✔
	Slow queries	✔	✔	✔	✔	✔
	Latency insight	✔	✔	✔	✔	✔
	Offline key analysis	✔	✔	✔	✔	✔
	Use the real-time key statistics feature	✔	✔	✔	✔	✔
	Create a diagnostic report	✔	✔	✔	✔	✔
Log management	Enable the new audit log feature	✔	✔	✔	□	□
	View slow logs	✔	✔	✔	✔	✔
	View active logs	✔	✔	✔	✔	✔
Backup and recovery	Automatic or manual backup	✔	✔	✔	✔	✔ (Only automatic backups are supported.)
	Download a backup file	✔	✔	✔	✔	✔
	Use data flashback to restore data by point in time	✔	✔	□	□	□

Item	Feature	Local disk			Cloud disk	
		Standard architecture	Cluster architecture	Read/write splitting architecture	Standard architecture	Cluster architecture
	Restore data from a backup set to a new instance	✔	✔	✔	✔	✔
Extended feature	Global Distributed Cache for Redis	✔	✔	✔	□	□
	Use proxy query cache to address issues caused by hotkeys	□	✔	□	□	□
	Extended data structures of ApsaraDB for Redis Enhanced Edition (Tair)	✔	✔	✔	✔	✔
	Multi-threading model	✔	✔	✔	✔	✔

Supported instance types, engine versions, and architectures

Instance disk type and creation method	Supported instance edition	Supported engine version	Supported architecture
Local disk-based instance Create Redis本地盘实例 an ApsaraDB for Redis Community Edition instance or a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair)	ApsaraDB for Redis Community Edition	4.0 5.0	Cluster architecture Standard architecture Read/write splitting architecture
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	5.0	Cluster architecture Standard architecture Read/write splitting architecture
	ApsaraDB for Redis Community Edition	7.0 6.0 5.0	Standard architecture Cluster architecture

Instance disk type and creation method	Supported instance edition	Supported engine version	Supported architecture
Cloud disk-based instance Create an ApsaraDB for Redis instance	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	5.0	Standard architecture Cluster architecture
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	6.0	Standard architecture Cluster architecture
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	4.0	Standard architecture

2.5. Features

ApsaraDB for Redis supports multiple architectures, persistent data storage, high availability, auto scaling, and intelligent O&M.

Flexible architectures

- Dual-node hot standby architecture

  The system synchronizes data between the master node and replica node in real time. If the master node fails, the system fails over to the replica node and restores services within seconds. Transient connections that last for seconds occur during the failover. This architecture ensures high availability of system services. For more information, see Standard master-replica instances.

- Cluster architecture

  Cluster instances run in a distributed architecture. Each instance node uses a high-availability master-replica architecture to ensure automatic failover and disaster recovery. A variety of cluster specifications are provided for different workloads. You can scale up a cluster instance as your needs evolve. For more information, see Cluster master-replica instances.

- Read/write splitting architecture

  Read/write splitting instances consist of proxy nodes, master-replica nodes, and read replicas. This architecture allows a large number of concurrent requests to read hot data from read replicas and provides high availability, high performance, and multiple specifications. This reduces the loads on the master node and minimizes Q&M costs. For more information, see Read/write splitting instances.

Data security

- Persistent data storage

  ApsaraDB for Redis provides high-speed data read/write capabilities and supports data persistence by using the hybrid storage of memory and disks.

- Backup and restoration

  ApsaraDB for Redis creates data backups for an instance to ensure strong disaster recovery capacities. You can use the backups to restore the instance for free with a few clicks. This mechanism reduces data loss caused by accidental operations.

- Multi-layer network security protection
  - Virtual private clouds (VPCs) are used to isolate network transmission at the transport layer.
  - Anti-DDoS monitors and protects against DDoS attacks.
  - Up to 1,000 IP addresses can be configured in a single instance whitelist to mitigate access risks.
  - Password authentication ensures secure and reliable access.
- In-depth kernel optimization

  Alibaba Cloud performed an in-depth engine optimization on the Redis source code to prevent out of memory (OOM) errors, fix security vulnerabilities, and protect your business.

High availability

- Master-replica architecture

  Standard and cluster instances support the master-replica architecture to prevent service interruptions caused by single points of failure (SPOFs).
- Automatic failure detection and restoration

  ApsaraDB for Redis detects hardware failures. In the case of a failure, the system performs a failover and restores services within seconds.
- Resource isolation

  In ApsaraDB for Redis, resources are isolated within instances. A failure on an instance does not affect other instances. This is a better solution to ensure stability for individual services.

Elastic scalability

- Data capacity scaling

  ApsaraDB for Redis provides multiple memory specifications. You can upgrade your memory specifications based on your business requirements.
- Performance scaling

  The cluster architecture supports auto scaling for storage space and throughput. This eliminates performance bottlenecks caused by the storage of large data volumes and the handling of high queries per second (QPS). With auto scaling, you can easily handle millions of read and write requests per second.

For more information, see Change the configurations of an instance.

Intelligent O&M

- Monitoring platform

  ApsaraDB for Redis monitors instance statistics such as CPU utilization, the number of connections, and disk storage usage and sends alerts in real time. This allows you to keep track of instance status. For more information, see Observability.
- Visualized management

  ApsaraDB for Redis allows you to perform frequent and high-risk operations with a few clicks, such as instance cloning, instance backup, and data restoration.
- Visualized DMS platform

  ApsaraDB for Redis supports the specialized Data Management (DMS) platform. This platform provides visualized data management to improve the efficiency of research and development (R&D) and O&M.

- Database kernel version management

  ApsaraDB for Redis supports automatic upgrades to quickly fix vulnerabilities. This way, you do not need to manage each kernel version. Parameter settings are optimized to maximize the utilization of system resources.

2.6. Scenarios

This topic describes how to use ApsaraDB for Redis for data storage in different scenarios, especially in high concurrency scenarios.

Gaming industry applications

ApsaraDB for Redis can serve as an important architecture component in the gaming industry.

- Scenario 1: use ApsaraDB for Redis as a storage service

  Gaming applications can be deployed in a simple architecture, in which the main program runs on an Elastic Compute Service (ECS) instance and the business data is stored in an ApsaraDB for Redis instance. ApsaraDB for Redis can be used for persistent storage. It uses a master-replica architecture to implement redundancy.

- Scenario 2: use ApsaraDB for Redis as a caching service to accelerate connections to applications

  ApsaraDB for Redis can serve as a caching service to accelerate connections to applications. Data is stored only on backend databases (ApsaraDB RDS instances).

The high availability of ApsaraDB for Redis is essential to your business. If your ApsaraDB for Redis service becomes unusable, the RDS instances may be overwhelmed by the requests that are sent from your applications. ApsaraDB for Redis adopts the master-replica architecture to ensure high availability. In this architecture, the master node provides services for your business. If this node fails, the system automatically switches workloads to the replica node. The complete failover process is transparent.

E-commerce industry applications

ApsaraDB for Redis is widely used in the e-commerce industry for business such as commodity presentation and recommendation.

- Scenario 1: online shopping systems

  An online shopping system is likely to be overwhelmed by user traffic during large promotional activities such as flash sales. Most databases cannot handle the heavy load. To resolve this issue, you can use ApsaraDB for Redis for persistent storage.

- Scenario 2: inventory management systems that support stocktaking

  You can store commodity data in RDS and inventory data in ApsaraDB for Redis. ApsaraDB for Redis supports high concurrency and high traffic scenarios with its high performance. It can handle each inventory query with ease. This storage method ensures that not all queries are forwarded to RDS to improve response speed and user experience.

Live streaming applications

Live streaming is strongly reliant on ApsaraDB for Redis to store user data and chat records.

- High availability

  ApsaraDB for Redis can be deployed in a master-replica architecture to significantly improve service availability.

- High performance

ApsaraDB for Redis provides cluster instances to eliminate the performance bottleneck that is caused by the single-thread mechanism of open source Redis. Cluster instances can handle traffic spikes during live streaming and meet high-performance requirements.

- High scalability

  ApsaraDB for Redis allows you to deal with traffic spikes during peak hours by scaling out an instance with a few clicks. The upgrade is completely transparent to users.

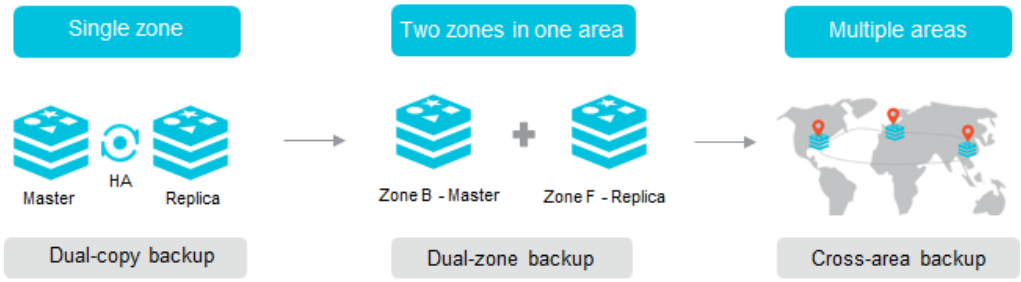
2.7. Disaster recovery

ApsaraDB for Redis is a high-performance key-value database service. This service helps you store large amounts of important data in various scenarios. This topic describes the disaster recovery solutions provided by ApsaraDB for Redis.

Roadmap of disaster recovery for ApsaraDB for Redis

An ApsaraDB for Redis instance may fail due to unexpected reasons, such as a device or power failure in the data center. In this case, disaster recovery can help ensure data consistency and service availability. ApsaraDB for Redis provides a variety of disaster recovery solutions to meet the requirements in different business scenarios.

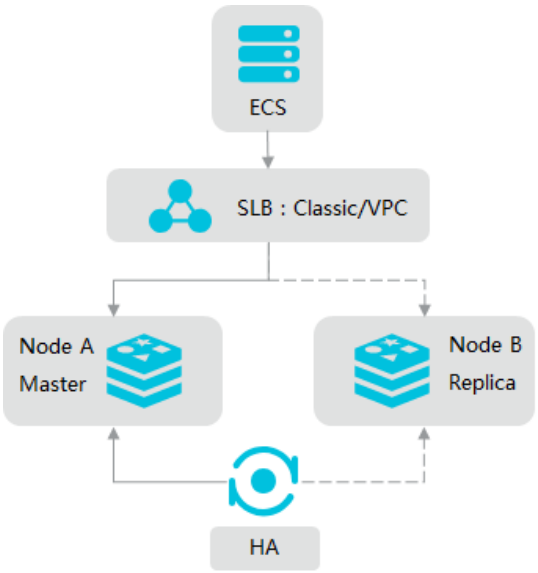
Roadmap of disaster recovery for ApsaraDB for Redis

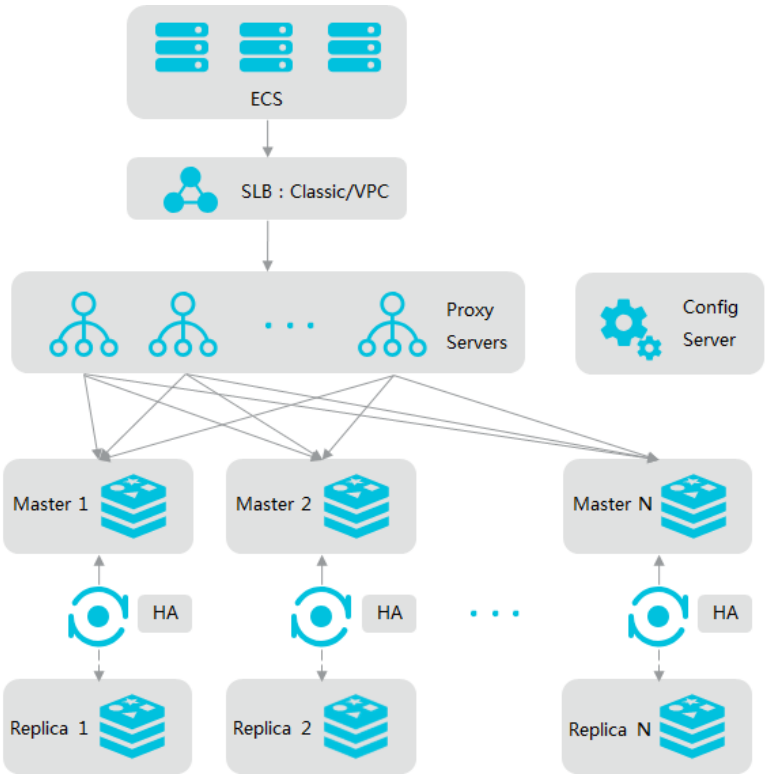


Disaster recovery solution	Protection level	Description
Single-zone HA solution	★★★☆☆	The master node and replica node are deployed on different machines in the same zone. If the master node fails, the high availability (HA) system performs a failover to prevent service interruption caused by a single point of failure (SPOF).
Zone-disaster recovery solution	★★★★☆	The master node and replica node are deployed in two different zones in the same region. If the zone in which the master node resides is disconnected due to force majeure factors such as a power or network failure, the HA system performs a failover to ensure continuous availability of the entire instance.
Cross-region disaster recovery solution	★★★★★	In the architecture of Global Distributed Cache for Redis, a distributed instance consists of multiple child instances that synchronize data among each other in real time by using synchronization channels. The channel manager monitors the health status of child instances and handles exceptions that occur on child instances, such as a switchover between the primary and secondary databases. Global Distributed Cache for Redis is applicable in scenarios such as geo-disaster recovery, active geo-redundancy, nearby application access, and load sharing.

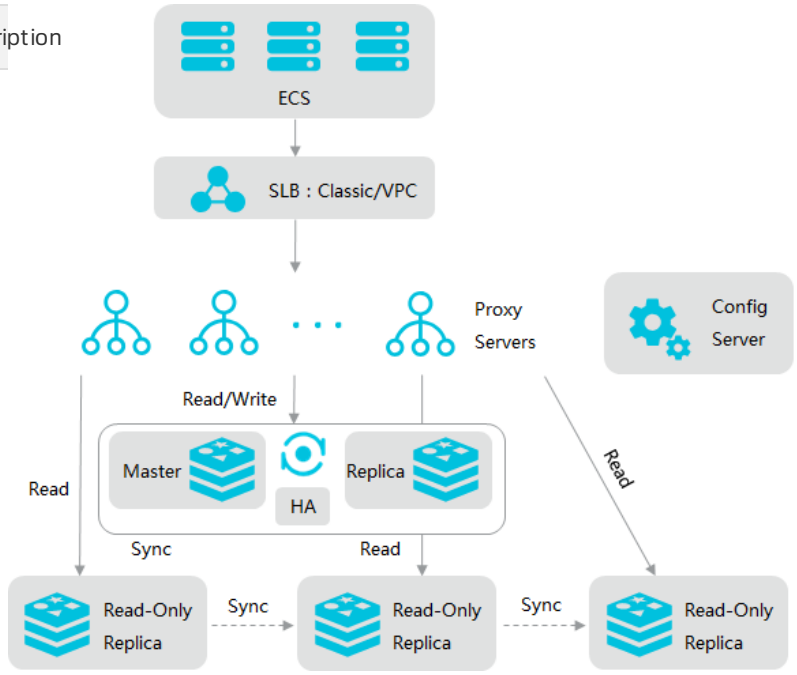
Single-zone HA solution

All ApsaraDB for Redis instances support a single-zone HA architecture. The HA system monitors the health status of the master node and replica node and performs failovers to prevent service interruption caused by SPOFs.

Instance type	Description
Standard master-replica instance	HA architecture for a standard master-replica instance  A standard master-replica instance runs in a master-replica architecture. If the HA system detects a failure on the master node, the system switches the workloads from the master node to the replica node and the replica node takes over the role of the master node. The original master node works as the replica node after recovery.

Instance type	Description
Master-replica cluster instance	HA architecture for a master-replica cluster instance  In a master-replica cluster instance, data is stored on data shards. Each data shard consists of a master node and a replica node. The master node and replica node are deployed on different machines in an HA architecture. If the master node fails, the HA system performs a failover to ensure high service availability. For more information about the components of a cluster instance, see Cluster master-replica instances.
	HA architecture for a read/write splitting instance

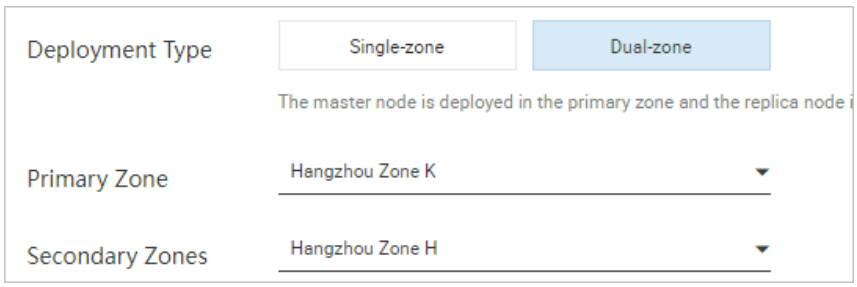
Instance type	Description

Read/write splitting instance	
	• The HA system monitors the health status of each node. If the master node fails, the HA system performs a failover between the master node and the replica node. If a read replica fails, the HA system creates another read replica to process read requests. During this process, the HA system updates the routing and weight information.
	• Proxy servers monitor the status of each read replica in real time. If a read replica is in one of the following states, proxy servers stop routing traffic to the node:
	  ◦ Abnormal: If a read replica is abnormal, proxy servers reduce the weight of traffic to the read replica. If the read replica fails to be connected for the specified number of times, the proxy servers stop routing traffic to the read replica until it is recovered.
	  ◦ Full data synchronization: If proxy servers detect that full data is being synchronized on a read replica, the proxy servers stop routing traffic to the read replica until the synchronization is complete.
	For more information about the components of a read/write splitting instance, see Read/write splitting instances.

Zone-disaster recovery solution

ApsaraDB for Redis standard instances and cluster instances support zone-disaster recovery across two data centers. If your workloads are deployed in a single region and require disaster recovery, you can select the zones that support zone-disaster recovery when you create an ApsaraDB for Redis instance. For more information, see Step 1: Create an ApsaraDB for Redis instance.

Create a zone-disaster recovery instance



When you create a multi-zone instance, the master node and replica node with the same specifications are deployed in different zones. The master node synchronizes data to the replica node through a dedicated channel.

If a power failure or a network error occurs on the master node, the replica node takes over the role of the master node. The system calls an API operation on the configuration server to update routing information for proxy servers. In addition, ApsaraDB for Redis provides an optimized Redis synchronization mechanism. Similar to global transaction identifiers (GTIDs) of MySQL, ApsaraDB for Redis uses global operation identifiers (OpIDs) to indicate synchronization offsets and runs lock-free threads in the background to search for OpIDs. The system asynchronously synchronizes append-only file (AOF) binary logs (binlogs) from the master node to the replica node. You can throttle synchronization to ensure service performance.

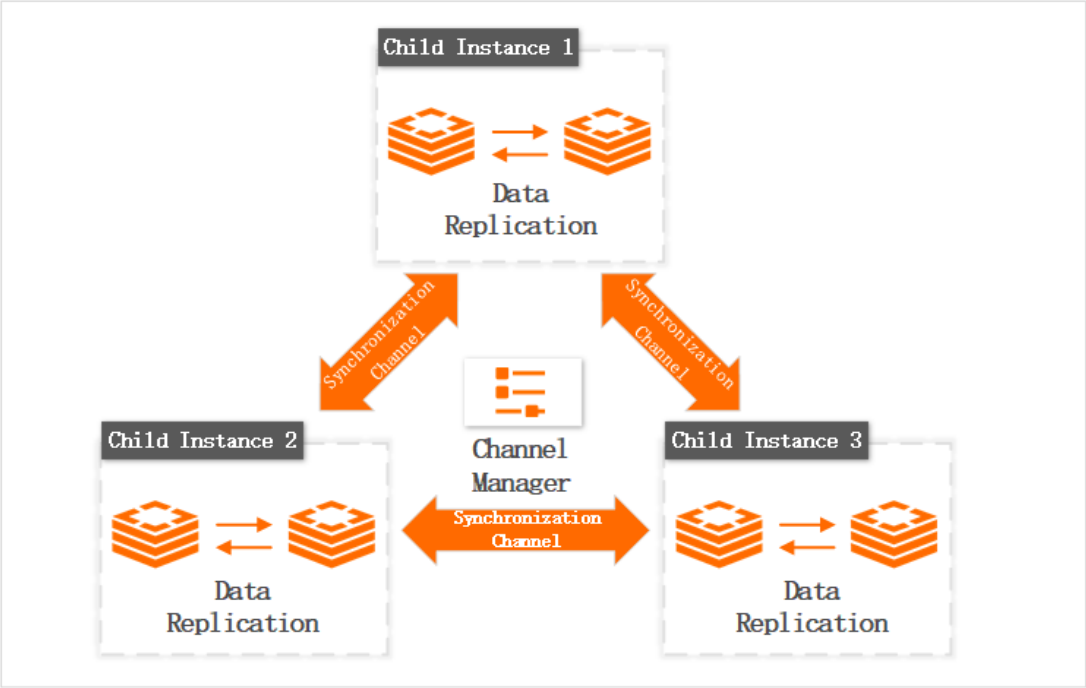
Cross-region disaster recovery solution

As your business rapidly grows to cover a wide range of areas, the architecture of cross-region long-distance access causes a high latency that affects user experience. The Global Distributed Cache for Redis feature of Alibaba Cloud can help you reduce the high latency caused by cross-region access. Global Distributed Cache for Redis has the following benefits:

- You can directly create child instances or specify the child instances that need to be synchronized without the need to implement redundancy in your business logic. This greatly reduces the complexity of business design and allows you to focus on the development of upper-layer business.
- The geo-replication capability is provided for you to implement geo-disaster recovery or active geo-redundancy.

This feature applies to cross-region data synchronization scenarios and global business deployment in industries such as multimedia, gaming, and e-commerce. For more information, see Overview.

Architecture of Global Distributed Cache for Redis



2.8. Observability

provides the observability that contains more dimensions, categories, and advanced features than open source Redis.

Context

Observability is the ability to access monitoring data, analyze issues, and perform systematic diagnostics based on three pillars of data: metrics, traces, and logs.

- Metrics: A metric is a numeric value of a dimension that is measured over a period of time to display specific states and trends of a system.
- Logs: A log is a record of discrete events that happened during the runtime of an application.
- Traces: A trace records the end-to-end lifecycle of a request.

integrates metrics, traces and logs to provide data analytics. The following table compares the observability of ApsaraDB for Redis and open source Redis. The following list describes the symbols that are used in the table.

- The ✔☺ symbol indicates that the feature is supported.
- The □ symbol indicates that the feature is not supported.
- The □ symbol indicates that no features are involved.

Observability		Open source Redis	ApsaraDB for Redis	ApsaraDB for Redis Enhanced Edition (Tair)
Metric	View monitoring data	✔☺	✔☺☺(fine-grained)	☺✔☺(fine-grained)
	Query active logs of an instance	✔☺	✔☺	✔☺

Observability Log		Open source Redis	ApsaraDB for Redis	ApsaraDB for Redis Enhanced Edition (Tair)
	View slow logs	✔	✔	✔
	View audit logs	□	✔	✔
	Latency insight	□	✔	✔
Trace	□	□	□	□
Analytics	Use the real-time key statistics feature	□	✔	✔
	Use the real-time key statistics feature	□	✔	✔
	Offline key analysis	□	✔	✔
	Create a diagnostic report	□	✔	✔

Metrics

Open source Redis provides a variety of metrics, including memory-related metrics (such as memory distribution, memory usage, and memory fragmentation ratio), statistics-related metrics (such as the number of connections and commands, network traffic, and synchronization status), CPU utilization, and keyspace information. provides more fine-grained metrics in addition to the metrics supported by open source Redis based on user experience. The fine-grained metrics include read queries per second (QPS) and write QPS. For more information about these metrics, see View monitoring data.

The fine-grained metrics provided by also have the following benefits in implementing observability:

- View performance metrics in real time: displays metrics in real time.
- Instance sessions: displays sessions between an ApsaraDB for Redis instance and clients in real time.
- Performance trends: displays performance trends over a specific period of time.

Logs

allows you to view active logs, slow logs, audit logs, and latency insight of an ApsaraDB for Redis instance.

- Active logs

  Active logs of an instance record in rows the persistence, synchronous replication, and debugging operations that take place and error messages that appear when the instance is running.

  You can go to the details page of an instance in the ApsaraDB for Redis console and choose **Logs > Active Logs** in the left-side navigation pane to view the active logs of the instance. For more information, see View active logs.

- Slow logs

Slow logs record requests that take longer than specific thresholds to execute. The execution time of a request does not include the amount of time that the request spends in queue or in transmission. Slow log statistics include execution timestamps, execution durations, command parameters, and clients. You can view slow logs of an instance, identify commands in the instance that take longer than required to run, and optimize these commands to prevent ApsaraDB for Redis congestion.

You can go to the details page of an instance in the ApsaraDB for Redis console and choose **Logs > Slow Logs** in the left-side navigation pane to view the slow logs of the instance. For more information, see View slow logs.

- Audit logs

  provides audit logs based on Log Service. For more information about Log Service, see What is Log Service? Audit logs include statistics such as log types, execution durations, database numbers, client IP addresses, account names, command details, and extension information. Audit logs allow you to search and analyze online operation logs (including logs about sensitive operations related to the `FLUSHALL`, `FLUSHDB`, and `DEL` commands), slow logs, and O&M logs, and export these logs.

  You can go to the details page of an instance in the ApsaraDB for Redis console and choose **Logs > Audit Log** in the left-side navigation pane to view the audit logs of the instance. For more information, see Enable the new audit log feature.

- Latency insight

  The latency insight feature is an advanced feature provided by ApsaraDB for Redis that allows you to collect latency statistics. With latency insight, up to 27 events and execution durations of all commands can be recorded, and all latency statistics within the last three days can be saved.

  You can go to the details page of an instance in the ApsaraDB for Redis console and choose **CloudDBA > Latency Insight** in the left-side navigation pane to view the latency insight information of the instance. For more information, see Latency insight.

Analytics

integrates metrics, traces, and logs to provide data analytics, which is a critical feature of ApsaraDB for Redis.

- Hotkey and large key analysis

  If a key receives significantly more requests than other keys, the key is considered a hotkey. If a hotkey is not timely handled, skewed requests or even cache breakdowns may occur. If a key contains a large number of members or occupies a large amount of memory, the key is considered a large key. If a large key is not timely handled, commands that involve the key take longer to run and an out-of-memory (OOM) error may occur for the key.

  You can use the **Real-time Key Statistics** feature of to identify hotkeys and large keys. The **Real-time Key Statistics** feature displays hotkeys and large keys in real time and allows you to view the hotkeys and large keys that were generated within the last four days. The **Real-time Key Statistics** feature is high in precision and has minimal impact on performance. This feature allows you to view the amount of memory occupied by a key and the frequency at which a key is requested and troubleshoot hotkeys and large keys to optimize instances.

  You can go to the details page of an instance in the ApsaraDB for Redis console and choose **CloudDBA > Real-time Key Statistics** in the left-side navigation pane to view statistics about hotkeys and large keys of the instance. For more information, see Use the real-time key statistics feature.

- Offline key analysis

The **Offline Key Analysis** feature supports the processing of offline Redis Database (RDB) files of all data structures and from all instance architectures and Redis versions supported by ApsaraDB for Redis and does not affect online services provided by ApsaraDB for Redis. The **Offline Key Analysis** feature can process a combination of 10% large keys and 90% small keys four times faster than redis-rdb-tools, and a combination of medium keys and large keys 20 times faster than redis-rdb-tools. During the process, memory usage is kept within 1 GB to prevent OOM errors that may occur due to large key processing. The **Offline Key Analysis** feature also allows you to search for the longest subelement to troubleshoot issues.

You can go to the details page of an instance in the ApsaraDB for Redis console and choose **CloudDBA > Offline Key Analysis** in the left-side navigation pane to view the offline key analysis of the instance. For more information, see Offline key analysis.

- Instance diagnostics

  integrates statistics such as performance metrics, slow logs, and key analysis to provide the diagnostic reports feature. This feature performs one-stop diagnostics to evaluate the health of instances based on multiple metrics (such as performance metrics, skewed request statistics, and slow logs) and puts forward suggestions. This feature improves the automatic Q&M capabilities of instances and reduces instance usage costs.

  You can go to the details page of an instance in the ApsaraDB for Redis console and choose **CloudDBA > Diagnostic reports** in the left-side navigation pane to perform diagnostics on the instance. For more information, see Create a diagnostic report.

3.ApsaraDB for Redis Enhanced Edition (Tair)

3.1. Overview

ApsaraDB for Redis Enhanced Edition (Tair) is a cloud-based in-memory database service for enterprises. It is developed based on the Tair service used by Alibaba Group. ApsaraDB for Redis Enhanced Edition (Tair) has officially been in use within Alibaba Group since 2009 and has proven its outstanding performance in a variety of scenarios, including Double 11, Youku Spring Festival Gala, Cainiao, and AMAP.

Emergence of Tair

In 2004, Taobao started using caching technologies to support its business operations. The first caching technology that was put into use is frontend page caching. This technology uses Edge Side Includes (ESI) to identify web content segments that can or cannot be accelerated. Frontend page caching reduces the number of times that entire pages are fetched from the server.

The rapid growth of customer traffic on Taobao exerted more pressure on databases. Backend caching emerged in response to this situation. Backend caching has evolved over many iterations from TBStore to Taobao Database Manager (TDBM). TBStore provides services such as persisting Taobao details and verification codes. TDBM was initially used in Taobao User Center. In 2009, the large-scale and high-performance in-memory database service Tair was released based on the success of previous systems and technical experience.

Tair is one of the most popular systems of Alibaba Group. Tair can handle hundreds of millions of calls per second and has provided core online access acceleration during Double 11 events for many years.

Milestones of Tair

Time	Event
July 2021	The following new series were released: • Persistent memory-optimized instances: Persistent memory-optimized instances adopt Intel Optane DC persistent memory (AEP) to provide you with in-memory databases that have a large capacity and are compatible with open source Redis. A persistent memory-optimized instance does not use disks to implement data persistence. Compared with an ApsaraDB for Redis Community Edition instance, it reduces costs by up to 30% and delivers almost the same throughput and latency while maintaining persistence of each operation. This helps improve the reliability of business data. • Storage-optimized instances: Storage-optimized instances were developed based on enhanced SSDs (ESSDs) and are compatible with core data structures and APIs of open source Redis. These instances can provide large-capacity, low-cost, and persistent database services. In the future, ApsaraDB for Redis Enhanced Edition (Tair) will focus on core cloud-native capabilities, such as hardware-software integration, intelligent data distribution, and integration of data storage and processing.

Time	Event
November 2019	Tair 3.0, also called ApsaraDB for Redis Enhanced Edition (Tair), was released. • Performance-enhanced instances: Performance-enhanced instances use the multi-threading model and are fully compatible with open source Redis. These instances integrate multiple Tair data structures to provide high-performance, high-compatibility database services that come with enterprise-grade features.
April 2019	The ApsaraDB for Redis team was recognized as one of the top three contributors in the Redis open source community and delivered a public speech at RedisConf 2019.
August 2018	ApsaraDB for Redis started to offer hybrid-storage instances in China to separate cold data from hot data for the first time and reduce costs for main users.
November 2017	Tair dynamic hashing provided support for the 2017 Double 11 event and was able to resolve several critical cache issues within the industry.
April 2017	Tair 2.0 was released and started to provide services to the AMAP and Youku new business units. ApsaraDB for Memcache (OCS) was upgraded to ApsaraDB for Redis.
August 2016	Tair Smart O&M Platform was released. The release of this platform contributed to the breakthrough of hundreds of billions of sales during the 2016 Double 11 event.
March 2015	The Tair team released ApsaraDB for Redis. This release helped Tair enter the cloud era.
May 2014	The Tair team released OCS, a cloud caching service. After OCS was released, OCS became a basic service of Alibaba Cloud and started to provide services for OCS users.
April 2013	The Fastdump service was released. It can significantly reduce import time and access latency. Tair was implemented on a large scale in Alimama.
October 2012	The Redis Database (RDB) cache engine was released with APIs that are similar to those used in open source Redis. This engine supports more flexible and complex data structures.
June 2011	The Local Database (LDB) persistence engine was released to meet key-value storage requirements on the Internet.
November 2009	Tair was used to support high traffic during the Double 11 event for the first time.
April 2009	Tair 1.0 was released and used in business segments such as the Taobao core system, Memory-Mapped Database (MDB) cache, and User Center.

Series and features of Tair

The rapid development of the Internet makes business scenarios more diverse and complicated. ApsaraDB for Redis Enhanced Edition (Tair) is a high-availability and high-performance NoSQL database service. It provides several series of instances based on storage media such as dynamic random-access memory (DRAM), non-volatile memory (NVM), and ESSDs to meet your requirements for low-latency access, persistence, and reduced overall costs. Tair provides higher performance, more data structures, and more flexible storage methods to meet your requirements in a variety of scenarios. For more information about ESSDs, see ESSDs.

Series type	Description
Performance-enhanced instances	• Performance-enhanced instances use the multi-threading model and provide read and write performance approximately three times that of ApsaraDB for Redis Community Edition instances with the same specifications. • Performance-enhanced instances provide multiple enhanced data structure modules such as TairString (including CAS and CAD commands), TairHash, TairGIS, TairBloom, TairDoc, TairTS, TairCpc, TairZset, TairRoaring, and TairSearch. These instances eliminate concerns about storage structure and timeliness and allow you to focus on application development. • Performance-enhanced instances provide high compatibility. Performance-enhanced instances are fully compatible with open source Redis. You can switch from open source Redis to ApsaraDB for Redis without the need to modify application code. • Performance-enhanced instances provide a variety of enterprise-grade features such as data flashback, proxy query cache, and Global Distributed Cache for Redis. For more information, see Use data flashback to restore data by point in time, Use proxy query cache to address issues caused by hotkeys, and Overview.
Persistent memory-optimized instances	• Persistent memory-optimized instances provide ultra-high cost-effectiveness. The price of persistent memory-optimized instances is 30% lower than that of ApsaraDB for Redis Community Edition instances with the same capacity. The performance of persistent memory-optimized instances reaches 90% of that of native Redis databases. • Persistent memory-optimized instances support the TairString (including CAS and CAD commands) and TairCpc enhanced data structure modules. • Persistent memory-optimized instances prevent data loss when power failures occur. These instances implement persistence for each command. The system will return a success response for each write operation only after the data is persistently stored. You can use persistent memory-optimized instances as in-memory databases instead of caches. • Persistent memory-optimized instances optimize the append-only file (AOF) rewriting performance for large-sized Redis databases. These instances reduce the latency and jitters that are caused when Redis calls forks to rewrite the AOF. • Persistent memory-optimized instances deliver high compatibility. These instances are compatible with most data structures and commands of native Redis databases.
Storage-optimized instances	Storage-optimized instances are developed based on ESSDs and are compatible with core data structures and APIs of open source Redis. These instances can provide large-capacity, low-cost, and persistent database services. Storage-optimized instances reduce costs and improve data reliability. In addition, storage-optimized instances reduce the amount of reserved memory that is required for the forks of open source Redis. This series type is suitable for scenarios that store warm and cold data, and require compatibility with open source Redis, large capacity, and high access performance.

Series type	Description
Hybrid-storage instances (phased out)	Hybrid-storage instances store data in both memory and disks. During off-peak hours, hybrid-storage instances can separate hot data from cold data to ensure a high memory access speed and provide a larger storage than ApsaraDB for Redis Community Edition instances. This allows hybrid-storage instances to strike a balance between performance and costs. **Note** Hybrid-storage instances are discontinued. For more information, see Sales of ApsaraDB for Redis hybrid-storage instances are discontinued. We recommend that you use performance-enhanced instances that provide higher performance and more features. If you have purchased s hybrid-storage instance, you can submit a ticket to migrate the data of the instance.

Purchase guide

Note For more information about the differences between local disks and cloud disks listed in the following table, see Comparison between ApsaraDB for Redis instances that use local disks and those that use cloud disks.

Series type	URL for purchase	Procedure
Performance-enhanced instances	Buy now	Create Redis本地盘实例an ApsaraDB for Redis Community Edition instance or a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair)
	Buy now	Create Redis企业版云盘实例a persistent memory-optimized or storage-optimized instance of ApsaraDB for Redis Enhanced Edition (Tair)
Persistent memory-optimized instances	Buy now	Create Redis企业版云盘实例a persistent memory-optimized or storage-optimized instance of ApsaraDB for Redis Enhanced Edition (Tair)
Storage-optimized instances		

Related information

- Performance test for performance-enhanced instances
- Performance test for persistent memory-optimized instances
- Performance test for storage-optimized instances

3.2. Performance-enhanced instances

Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) are suitable for scenarios that require high concurrency, a large number of read and write operations on hot data, and higher performance than ApsaraDB for Redis Community Edition instances. Compared with ApsaraDB for Redis Community Edition instances, performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) provide more benefits, including enhanced multi-threading performance and integration with multiple Redis modules.

Benefits

Item	Description
Compatibility	• Performance-enhanced instances are fully compatible with native Redis databases. To switch from native Redis databases to ApsaraDB for Redis instances, you do not need to modify the application code.
Performance	• Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) use the multi-threading model and provide three times the performance of ApsaraDB for Redis Community Edition instances with the same specifications. This eliminates the performance limits on high-frequency read and write requests for hot data. • Compared with native Redis databases, performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) can process a large number of queries per second (QPS) with higher performance at higher speeds. • Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) ensure stable performance in high-concurrency scenarios and alleviate connection issues that are caused by an unexpected rise in the number of requests during peak hours. • Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) run full data synchronization tasks and incremental data synchronization tasks in I/O threads to accelerate synchronizations.
Architecture	• Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) support the standard, cluster, and read/write splitting architecture.
Integration with multiple Redis modules	• Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) are integrated with multiple self-developed Redis modules to expand the applicable scope of ApsaraDB for Redis. These modules include TairString (including CAS and CAD commands), TairHash, TairGIS, TairBloom, TairDoc, TairTS, TairCpc, TairZset, TairRoaring, and TairSearch. These modules remove your worries about storage structures and timeliness and allow you to focus on your business development.
Enterprise-grade feature	• Performance-enhanced instances provide a variety of enterprise-grade features such as data flashback, proxy query cache, and Global Distributed Cache for Redis. For more information, see Use data flashback to restore data by point in time, Use proxy query cache to address issues caused by hotkeys, and Overview.

Item	Description
Data security	• Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) support SSL encryption for enhanced data security. For more information about SSL encryption, see Configure SSL encryption. • Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) support Transparent Data Encryption (TDE). TDE can be used to encrypt and decrypt Redis Database (RDB) files to ensure data security. For more information about TDE, see Enable TDE.

Scenarios

Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) can be used in scenarios such as live video streaming, flash sales, and online education. The following section describes typical scenarios:

- Scenario 1: During flash sales, the number of QPS on some cached hotkeys may exceed 200,000. Standard master-replica instances of the ApsaraDB for Redis Community Edition cannot meet this requirement.

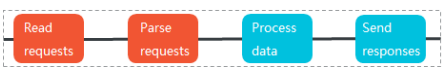
  Performance-enhanced standard instances of the ApsaraDB for Redis Enhanced Edition (Tair) can efficiently process requests during these flash sales without performance issues.
- Scenario 2: Cluster instances of the ApsaraDB for Redis Community Edition have limits on database transactions and Lua scripts.

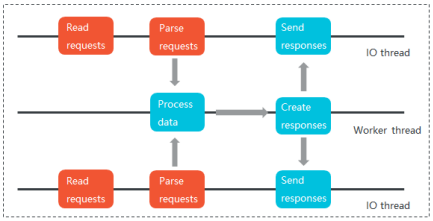
  Performance-enhanced cluster instances of the ApsaraDB for Redis Enhanced Edition (Tair) provide high performance and eliminate the limits on the usage of commands in cluster instances of the ApsaraDB for Redis Community Edition.
- Scenario 3: You have created a self-managed Redis instance that consists of one master node and multiple replica nodes. The number of replica nodes and O&M costs increase as your workloads increase.

  Performance-enhanced read/write splitting instances of the ApsaraDB for Redis Enhanced Edition (Tair) can provide one data node and up to five read replicas to help you handle millions of QPS.
- Scenario 4: You have created a self-managed Redis cluster to handle tens of millions of QPS. The number of data shards and O&M costs increase as your workloads increase.

  Performance-enhanced cluster instances of the ApsaraDB for Redis Enhanced Edition (Tair) can downsize clusters by two thirds and significantly reduce O&M costs.

Comparison between threading models

Threading model	Description
Single-threading model 	ApsaraDB for Redis Community Edition instances and native Redis databases adopt the single-threading model. During request handling, native Redis databases and ApsaraDB for Redis Community Edition instances must undergo the following steps: read requests, parse requests, process data, and then send responses. In this situation, network I/O operations and request parsing consume most of available resources.

Threading model	Description
Multi-threading model 	To improve performance, each performance-enhanced instance runs on multiple threads to process the tasks in these steps in parallel. • I/O threads are used to read requests, send responses, and parse commands. • Worker threads are used to process commands and timer events. • Auxiliary threads are used to monitor the heartbeat and status of nodes to ensure high availability. Each performance-enhanced instance reads and parses requests in I/O threads, places the parsed requests as commands in a queue, and then sends these commands to worker threads. Then, the worker threads run the commands to process the requests and send the responses to I/O threads by using a different queue. Each performance-enhanced instance supports a maximum of four parallel I/O threads. Unlocked queues and pipelines are used to transmit data between I/O threads and worker threads to improve multi-threading performance. **Note** • The processing speeds of threads are accelerated for common data structures, such as string, list, set, hash, ZSET, HyperLogLog, Geo, and extension structures. • API operations such as pub, sub, and blocking are replicated in worker threads. Therefore, these API operations can be accelerated to increase throughput and to increase performance by about 50%. • Transactions and Lua scripts must be executed in sequence. No acceleration can be achieved.

Note The multi-threading model of Redis 6.0 consumes large amounts of CPU resources to deliver performance that is up to two times higher than that delivered by the single-threading model of a major version earlier than Redis 6.0. The Real Multi-I/O model of performance-enhanced instances provides fully accelerated I/O threads to sustain a large number of concurrent connections and offer a linear increase in throughput.

Performance comparison

ApsaraDB for Redis Community Edition instances and native Redis databases adopt the single-threading model. In the single-threading model, each data node supports 80,000 to 100,000 QPS. Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) adopt the multi-threading model. The multi-threading model allows the I/O, worker, and auxiliary threads to process requests in parallel. Each data node of a performance-enhanced instance delivers performance that is about three times that delivered by each data node of an ApsaraDB for Redis Community Edition instance. The following table compares the use scenarios of ApsaraDB for Redis Community Edition instances and performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) in different architectures.

Architecture	Instance	Description
Standard	ApsaraDB for Redis Community Edition instances	These instances are not suitable if the number of QPS that is required on a single node exceeds 100,000.
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	These instances are suitable if the number of QPS that is required on a single node exceeds 100,000.
Cluster	ApsaraDB for Redis Community Edition instances	A cluster instance consists of multiple data nodes. Each data node provides performance that is similar to that of a standard instance. If a data node stores hot data and receives a large number of concurrent requests for hot data, the read and write operations on the other data that is stored on the data node may be affected. As a result, the performance of the data node deteriorates.
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	These instances provide high performance to read and write hot data at reduced maintenance costs.
Read /write splitting	ApsaraDB for Redis Community Edition instances	These instances provide high read performance and are suitable for scenarios in which the number of read operations is larger than the number of write operations. However, these instances cannot support a large number of concurrent write operations.
	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)	These instances provide high read performance and can support a large number of concurrent write operations. These instances are suitable for scenarios in which a large number of write operations need to be processed but the number of read operations is larger than the number of write operations.

Integration with multiple Redis modules

Similar to open source Redis, ApsaraDB for Redis Community Edition supports a variety of data structures such as strings, lists, hashes, sets, sorted sets, and streams. These data structures support common development workloads but not sophisticated workloads. To manage sophisticated workloads, you must modify your application data or run Lua scripts.

Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) are integrated with multiple Redis modules to expand the applicable scope of ApsaraDB for Redis. These modules include TairString (including CAS and CAD commands), TairHash, TairGIS, TairBloom, TairDoc, TairTS, TairCpc, TairZset, TairRoaring, and TairSearch. These modules simplify business development in complex scenarios and allow you to focus on your business innovation.

Data structure	Description
CAS and CAD commands	These commands are developed to enhance the functionality of Redis strings. You can use the commands to implement simple and efficient distributed locks based on Redis. For more information, see Implement high-performance distributed locks by using TairString.
TairString	A TairString is a string that consists of a key, a value, and a version number. Moreover, TairStings can be used to limit the range of outputs returned by the INCRBY and INCRBYFLOAT commands. These commands are used to increase or decrease the values of Redis strings. If outputs are out of range, error messages are returned by these commands.
TairHash	Similar to a native Redis hash, a TairHash is a hash that supports a variety of data structures and provides high processing performance. To simplify the development process, TairHashes also allow you to specify the expiration time and version number for a field. TairHashes use the efficient active expiration algorithm to check the expiration time of fields and delete expired fields. This process does not increase the database response time.
TairGIS	TairGIS is a data structure that uses R-tree indexes and supports APIs related to a geographic information system (GIS). Native Redis GEO commands allow you to use one-dimensional indexes to query points. TairGIS commands allow you to use two-dimensional indexes to query points, linestrings, and polygons. You can also use TairGIS commands to check the relationships between different elements, such as whether A contains B or A intersects with B.
TairBloom	A TairBloom is a Bloom filter that supports dynamic scaling and is fully compatible with RedisBloom commands. Compared with traditional methods that achieve a similar feature, TairBlooms consume less memory and maintain a stable false positive rate during scaling. You can use TairBlooms to check whether a large amount of data exists. In this case, a specific false positive rate is allowed.
TairDoc	TairDoc is a document data structure. It supports JSON standards and is fully compatible with RedisJSON commands. TairDoc data is stored in binary trees and allows fast access to child elements.
TairTS	TairTS is a time series data structure that is developed on top of Redis modules. This data structure provides low-latency and high-concurrency in-memory read and write access, supports fast filtering and aggregate queries, and has both storage and computing power. TairTS simplifies the processing of time series data and significantly improves performance.
TairCpc	TairCpc is a data structure developed based on the compressed probability counting (CPC) sketch. It allows you to perform high-performance computing on sampled data while using only a small amount of memory.

Data structure	Description
TairZset	Native Redis Sorted Sets (ZSETs) allow you to sort elements based on score data of the DOUBLE type only in one dimension. To exceed the limit, Alibaba Cloud has developed the TairZset data structure that allows you to sort elements based on score data of the DOUBLE type with respect to different dimensions. This data structure improves the efficiency of data processing and is also easy to use on the client side without the need to encode, decode, or encapsulate the data.
TairRoaring	The TairRoaring data structure is developed on top of Roaring bitmaps of Tair. TairRoaring uses two-level indexes and introduces multiple dynamic containers. TairRoaring also adopts optimization methods such as single instruction, multiple data (SIMD), vectorization, and popcount to provide less memory consumption and deliver higher computing efficiency for collections.
TairSearch	TairSearch is a full-text search module developed in-house based on Redis modules instead of open source search engine software libraries such as Lucene. TairSearch uses query syntax that is similar to that of Elasticsearch.

Enterprise-grade feature

Enterprise-grade feature	Description
Use data flashback to restore data by point in time	After you enable the data flashback feature, ApsaraDB for Redis retains append-only file (AOF) backup data for up to seven days. During the retention period, you can specify a point in time that is accurate to the second to create an instance and restore the backup data at the specified point in time to the new instance.
Proxy query cache	After you enable the proxy query cache feature, the configured proxy nodes cache requests and responses for hotkeys. If the same requests are received from a client within the specified validity period, ApsaraDB for Redis retrieves the responses to the requests from the cache and returns the responses to the client. During this process, ApsaraDB for Redis does not need to interact with backend data shards. For more information, see Use proxy query cache to address issues caused by hotkeys.
Global Distributed Cache for Redis	Global Distributed Cache for Redis is an active geo-redundancy database system that is developed based on ApsaraDB for Redis. Global Distributed Cache for Redis supports business scenarios in which multiple sites in different regions provide services at the same time. It helps enterprises replicate the active geo-redundancy architecture of Alibaba.
Two-way data synchronization by using DTS	Data Transmission Service (DTS) supports two-way data synchronization between instances of the ApsaraDB for Redis Enhanced Edition (Tair). For more information about DTS, see What is DTS? This synchronization solution is suitable for scenarios such as active geo-redundancy and geo-disaster recovery. For more information, see Configure two-way data synchronization between ApsaraDB for Redis Enhanced Edition (Tair) instances.

FAQ

Q: What do I do if a client does not support the commands that are provided by new modules?

A: Two methods are available for you. The first method is to define the commands that are provided by new modules in your application code before you use these commands in your client. The second method is to use the TairJedis client that is developed by Alibaba Cloud based on Jedis.

Related information

- Performance test for performance-enhanced instances
- Step 1: Create an ApsaraDB for Redis instance
- Overview

3.3. Persistent memory-optimized instances

Persistent memory-optimized instances of the ApsaraDB for Redis Enhanced Edition (Tair) adopt Intel Optane DC persistent memory (AEP) to provide large-capacity in-memory databases that are compatible with open source Redis. A persistent memory-optimized instance persists each operation and does not use disks to implement data persistence. Compared with an ApsaraDB for Redis Community Edition instance, a persistent memory-optimized instance reduces costs by up to 30% and delivers almost the same throughput and latency. This improves the reliability of business data.

Instance purchase

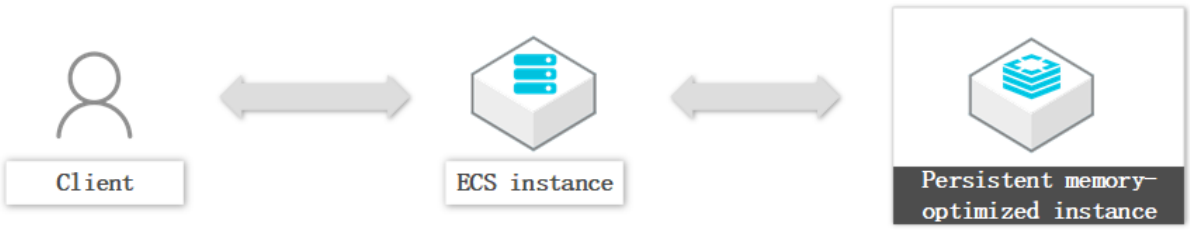
Create Redis企业版云盘实例a persistent memory-optimized or storage-optimized instance of ApsaraDB for Redis Enhanced Edition (Tair)

Background information

The high price and low capacity of memory limit the large-scale use of memory in specific scenarios. Alibaba Cloud began to invest in the research and implementation of Intel AEP in 2018. Intel AEP was applied to the core cluster of e-commerce products during Double 11 that year and significantly reduced costs. The cluster became a product that officially deployed Intel AEP in a production environment.

Maturer cloud environments and improved AEP-related technologies help Alibaba Cloud develop a new engine for AEP-based data persistence implementation. Alibaba Cloud integrates the new engine with Elastic Compute Service (ECS) bare metal instances to introduce persistent memory-optimized instances of the ApsaraDB for Redis Enhanced Edition (Tair). These instances replace the traditional volatile memory of Redis with persistent memory to significantly reduce the risk of data loss. For more information about ECS bare metal instances, see Overview.

Persistent memory-optimized instances provide not only memory-level access latency and throughput but also data persistence. In addition to reducing costs, persistent memory-optimized instances can simplify the application architecture. The popular architecture that consists of applications, cache, and persistent storage can be simplified to an architecture that consists of applications and persistent memory-optimized instances, as shown in the following figure.



Benefits

Persistent memory-optimized instances of the ApsaraDB for Redis Enhanced Edition (Tair) adopt Intel AEP to provide large-capacity in-memory databases that are compatible with open source Redis. A persistent memory-optimized instance persists each operation and does not rely on disks for data persistence. This helps improve the reliability of business data. Compared with an ApsaraDB for Redis Community Edition instance, a persistent memory-optimized instance reduces costs by up to 30% and delivers almost the same throughput and latency. Persistent memory-optimized instances are suitable for scenarios that store a large amount of hot and warm data, have high requirements for data persistence and service stability, and require compatibility with Redis.

Item	Description
Cost-effectiveness	• A persistent memory-optimized instance costs about 30% less than an ApsaraDB for Redis Community Edition instance. • Persistent memory-optimized instances can provide 90% of the performance of open source Redis.
Integration with multiple Redis modules	• TairString (including CAS and CAD commands) and TairCpc are supported.
Optimization for high specifications	• Persistent memory-optimized instances resolve issues such as high latency, high network jitter, and slow service data loading during fork operations triggered by append-only file (AOF) rewrites in scenarios where advanced memory specifications are used. You do not need to choose between performance and persistence. • The memory size of master-replica instances ranges from 128 GB to 512 GB. The memory size of cluster instances can reach up to 32 TB.
Data loss prevention during power outages	• Persistent memory-optimized instances support command-level persistence. A response is returned for each persisted write operation. • If you use open source Redis, you must take the risk of data loss calculated at seconds. In comparison, persistent memory-optimized instances are more reliable in high queries per second (QPS) scenarios.
High compatibility	• Persistent memory-optimized instances are compatible with the ApsaraDB for Redis database system to support high availability, auto scaling, logging, intelligent diagnostics, and flexible backup and restoration. • Persistent memory-optimized instances are compatible with most of the data structures and interfaces of open source Redis.

Scenarios

- Scenarios that require high performance and reduced costs for processing a large amount of data

  Intermediate data computing requires high performance. If you use ApsaraDB for Redis Community Edition for intermediate data computing, the costs are high. Other database types such as HBase cannot meet the performance requirements. Persistent memory-optimized instances not only ensure data persistence but also provide almost the same performance as ApsaraDB for Redis Community Edition instances in terms of throughput and latency. This helps you strike a balance between performance and costs.

- Scenarios that have high requirements for data persistence

For gaming services, persistent memory-optimized instances are used for data storage. Compared with a Redis and MySQL-based architecture, persistent memory-optimized instances provide a simpler architecture, higher performance, higher cost-effectiveness, and higher data reliability.

Instance types

Persistent memory-optimized instances

Related information

- Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair)
- Performance test for persistent memory-optimized instances

3.4. Storage-optimized instances

Storage-optimized instances of ApsaraDB for Redis Enhanced Edition (Tair) are developed based on enhanced solid-state disks (ESSDs) and compatible with core data structures and interfaces of open source Redis. These instances can provide large-capacity, low-cost, and persistent database services. Storage-optimized instances reduce costs and improve data reliability. In addition, storage-optimized instances reduce the amount of reserved memory that is required for the forks of native Redis databases. This instance type applies to scenarios that store warm and cold data, and require compatibility with Redis, large capacity, and high access performance.

Instance purchase

Create Redis企业版云盘实例a persistent memory-optimized or storage-optimized instance of ApsaraDB for Redis Enhanced Edition (Tair)

Background information

Native Redis uses volatile memory as storage media. This causes the following issues when business continuously grows and data volume rapidly increases:

- Latency and jitter occur when Redis processes a large amount of data.
- When data accumulates along with the development of business and the access volume decreases, the actual performance is more excessive than the requirements.

To resolve the preceding issues, Alibaba Cloud released storage-optimized instances that use ESSDs as the storage media. Storage-optimized instances are cost-effective and reduce up to 85% costs compared with ApsaraDB for Redis Community Edition instances. A storage-optimized instance can provide a storage capacity that reaches hundreds of TBs and higher data reliability. Storage-optimized instances reduce costs and improve data reliability. In addition, storage-optimized instances reduce the amount of reserved memory that is required for the forks of native Redis databases.

Comparison

Item	Storage-optimized instance	Hybrid-storage instances (phased out) (phased out)	Open source Pika
Compatibility	Delivers compatibility with most commands of native Redis databases.	Delivers compatibility with all commands of native Redis databases.	Uses quite different commands from native Redis databases.
Data persistence	Half synchronization or asynchronous synchronization.	Asynchronous synchronization.	Asynchronous synchronization.

Item	Storage-optimized instance	Hybrid-storage instances (phased out) (phased out)	Open source Pika
Data distribution	Stores data only in disks by using the Alibaba Cloud TairDB storage engine.	Stores the keys and values of hot data in the memory and all keys and values in disks.	Stores data only in disks by using the RocksDB storage engine.
Performance	Delivers a performance that is 60% of the performance of native Redis databases. The performance of storage-optimized instances for complicated data structures, such as LIST, HASH, SET, and ZSET, is 10% higher than the performance of open source Pika.	Delivers the same performance as native Redis databases for hot data in the memory. The performance for cold data is not guaranteed. The overall performance of a hybrid-storage instance depends on the distribution of cold and hot data.	Delivers a performance that is 60% of the performance of native Redis databases.
Scenarios	Warm data.	Data that is randomly accessed.	Warm data.
Costs	Saves up to 85% of costs compared with ApsaraDB for Redis Community Edition instances.	Requires 20% higher costs on average than storage-optimized instances.	Requires lower costs for on-premises physical devices but higher O&M costs.

Instance type

Storage-optimized instances

FAQ

- Q: What is the version of the engine that is used for storage-optimized instances?

  A: Storage-optimized instances use an engine that is developed by Alibaba Cloud. This engine is compatible with Redis 4.0. For more information about the support for commands, see Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair).

  > **Note** To support specific components or clients that need to detect the engine version, such as Redis-shake, storage-optimized instances retain the redis_version parameter in the response of the **INFO** command. The value of the redis_version parameter is 4.9.9.

Related information

- Performance test for storage-optimized instances

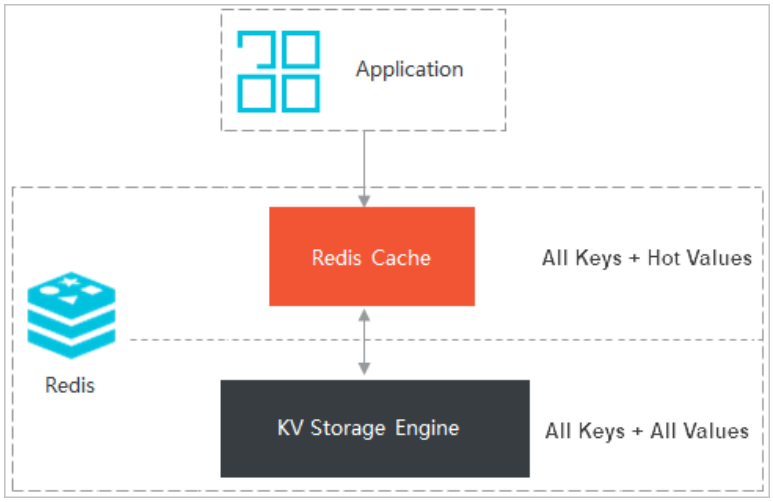
3.5. Hybrid-storage instances (phased out)

ApsaraDB for Redis Enhanced Edition (Tair) supports performance-enhanced instances and hybrid-storage instances. Different from instances of ApsaraDB for Redis Community Edition, hybrid-storage instances store data in both memory and disks. Hybrid-storage instances offer the benefits of both memory and disks to support data persistence and high read and write performance at the same time.

> **Note** Hybrid-storage instances are discontinued. For more information, see Sales of ApsaraDB for Redis hybrid-storage instances are discontinued. We recommend that you use performance-enhanced instances that provide higher performance and more features.
>
> If you have purchased s hybrid-storage instance, you can submit a ticket to migrate the data of the instance.

Overview

Architecture



Hybrid-storage instances of ApsaraDB for Redis Enhanced Edition (Tair) are developed by Alibaba Cloud and are compatible with the Redis protocol. These instances store all data in disks and hot data in memory to ensure high read and write performance. Hybrid-storage instances deliver a cost-effective solution that supports high-performance queries for frequently accessed data without the limit of memory capacity.

- In-memory storage: Keys and values of hot data are stored in memory. The information about all keys is also cached in memory. This way, you can check whether a specified key exists in a fast and convenient manner.
- Disk storage: All keys and values are stored in disks. Redis data structures such as hashes are also stored in disks by using a specific format.

Scenarios

Scenario	Description
Live streaming	Live streams generate a large amount of hot data. Most of the data comes from popular live channels. You can use hybrid-storage instances to store data from popular live channels in memory and data from less popular live channels in disks. This allows you to make full use of the limited memory capacity.
E-commerce	E-commerce applications need to access a large amount of commodity data. You can use hybrid-storage instances to store data and optimize memory usage. The data of popular items is stored in memory and the data of less popular items is stored in disks.

Scenario	Description
Online education	Online education applications need to access a large amount of data. Such data includes online courses, question libraries, and messages between teachers and students. Only popular courses and the latest question libraries are frequently accessed. You can use hybrid-storage instances to store the data of online courses in disks, and store the data of popular courses and question libraries in memory. ApsaraDB for Redis provides a cost-effective solution to ensure high read and write performance for frequently accessed data.

The following examples show the benefits of hybrid-storage instances:

- Example 1: A native Redis cluster is used to store 100 GB of data. The queries per second (QPS) at peak hours is less than 20,000 and 80% of the data is not frequently accessed.

  In this case, you can use a hybrid-storage instance of ApsaraDB for Redis Enhanced Edition (Tair) that has 32 GB memory and 128 GB disk capacity. This saves about 70 GB of memory resources and reduces storage costs by more than 50%.

- Example 2: An on-premises Pika instance is used to reduce the storage costs of a native Redis deployment. The total size of the data is about 400 GB and only about 10% of the data is frequently accessed. High costs are incurred for cluster O&M.

  In this case, you can use a hybrid-storage instance of ApsaraDB for Redis Enhanced Edition (Tair) that has 64 GB memory and 512 GB disk capacity. This reduces your O&M costs and ensures high availability.

Select instances based on scenarios

Hybrid-storage instances of ApsaraDB for Redis Enhanced Edition (Tair) and memory-only instances of ApsaraDB for Redis Community Edition are applicable to different scenarios. The following table describes the scenarios to which the two types of instances are applicable.

Scenario category	Hybrid-storage instance	Community Edition instance
Data size and budget	• You have a large amount of data to store and want to reduce the storage costs. • You are using the Pika, SSDB, or Antibiotic Resistance Genes Database (ARDB) services. • You are using master-replica or cluster instances of ApsaraDB for Redis to store a large amount of data.	You have a small amount of data, or your budget is sufficient for storage costs.
Separation of hot and cold data	Your business data can be divided into hot data and cold data. **Note** Cold data is the data that is not frequently accessed.	Your business data is accessed at random.

Scenario category	Hybrid-storage instance	Community Edition instance
QPS performance and latency requirements for hot and cold data	• High QPS performance is required for hot data. • The access to cold data is not sensitive to latency.	• High QPS performance is required for all data. • The access to all data is sensitive to latency.
Access frequency of big keys	• No big key exists in your business data. • Big keys are frequently accessed and need to be stored in memory. • Big keys are not frequently accessed and are not sensitive to latency.	Big keys are accessed at random and are sensitive to latency.

Select instances based on specifications

When you create a hybrid-storage instance of ApsaraDB for Redis Enhanced Edition (Tair), you must specify the memory capacity and the disk capacity. The memory capacity determines the maximum amount of hot data that can be stored and the disk capacity determines the total amount of data that can be stored. ApsaraDB for Redis allocates CPU resources to the instance based on the specified memory capacity and disk capacity. For example, if you select a **master-replica instance that has 64 GB of memory and 256 GB of disk capacity**, the instance can store up to 256 GB of data and can cache 64 GB of data in memory.

For more information, see Overview.

> **Note**
> - You cannot customize the number of CPU cores.
> - Hybrid-storage instances store the metadata of keys in memory and disks. The metadata includes the time-to-live (TTL) value, the least recently used (LRU) clock value, and the type of each key. When you specify the memory capacity and the disk capacity, you must reserve storage capacity for metadata. For more information, see the following description in this topic.

- Memory capacity

  To support more native Redis features, hybrid-storage instances store all keys in memory. These instances store the values of frequently accessed keys in memory and the values of all keys in disks. Therefore, make sure that the specified memory capacity is sufficient to store all keys and related metadata. The following table lists the recommended memory capacity based on the total number of keys.

Total number of keys	Recommended memory capacity
Less than 20 million	64 GB, 32 GB, and 16 GB
20 million to 50 million	64 GB and 32 GB
50 million to 100 million	128 GB, 64 GB, and 32 GB

Total number of keys	Recommended memory capacity
More than 100 million	128 GB and 64 GB

> **Note** The memory capacity that you specify when you create an ApsaraDB for Redis instance determines the CPU resources that are allocated to the instance. Instances that have higher memory capacities provide higher performance.

- Disk capacity

  Hybrid-storage instances store all data in disks, including the metadata generated for each key. The metadata occupies extra storage. Therefore, we recommend that you select a disk capacity that is 20% to 50% more than the required storage.

Purchase an ApsaraDB for Redis instance

For more information, see Step 1: Create an ApsaraDB for Redis instance.

Instance performance

The performance of a hybrid-storage instance depends on the instance type and memory hit ratio. The hit ratio indicates the probability that the requested data is found in memory. Higher specifications and higher hit ratios indicate higher performance. If all requested data is found in the memory of the hybrid-storage instance, the performance is the same as that of a Community Edition instance. A lower memory hit ratio indicates the lower performance of the hybrid-storage instance. If all requested data is found in disks, the hybrid-storage instance delivers the lowest performance.

In the following performance tests, each ApsaraDB for Redis instance stores 20 million keys and the size of each value is 1 KB. The **GET** command is used to read values. Different types of keys are accessed in the following scenarios.

Test results

Testing scenario	QPS of a Community Edition instance	QPS of a hybrid-storage instance
Keys are accessed at random.	123,000	15,000
Keys are accessed based on the Gaussian distribution, and 20% of the keys are accessed at a probability of 80%.	120,000	54,000
Keys are accessed based on the Gaussian distribution, and 1% of the keys are accessed at a probability of 99%.	135,000	114,000

FAQ about features

- Q: Do hybrid-storage instances of ApsaraDB for Redis Enhanced Edition (Tair) have limits on commands?

A: Hybrid-storage instances of ApsaraDB for Redis Enhanced Edition (Tair) support most native Redis commands. To ensure high performance, hybrid-storage instances impose limits on the use of specific commands. For more information, see Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair).

- Q: What is the threshold of used memory? What happens if the threshold is exceeded?

  A: The threshold is 90% of the memory capacity of the instance. For example, if the amount of used memory exceeds 14.4 GB for a hybrid-storage instance with a memory capacity of 16 GB, ApsaraDB for Redis evicts values of less frequently accessed keys from memory. The values of such keys are still stored in disks.

- Q: What is the granularity of data eviction if the amount of used memory exceeds the threshold?

  A: Keys are the smallest unit for eviction. For example, if the memory usage of a hash exceeds the threshold, the entire hash table is evicted.

- Q: How is a key-value pair stored in memory?

  A: Compared with instances of Community Edition, hybrid-storage instances store additional metadata. The metadata specifies whether the values of keys are cached in memory. The metadata can also be used to check whether data is hot or cold when the data is evicted.

- Q: How is data written to disks?

  A: To prevent direct access to disks, ApsaraDB for Redis modifies data in memory and then uses background threads to continuously synchronize the modifications to disks. This is similar to the process for a page cache.

  > **Note** Hot data is directly modified in memory. Cold data must be loaded to memory before the code data can be modified.

- Q: How does a hybrid-storage instance work if the requested data is not cached in memory?

  A: ApsaraDB for Redis checks whether the requested data is cached in memory before it runs a command. If the data is not cached in memory, ApsaraDB for Redis loads the data from disks to memory before it runs the command.

  > **Note** It takes longer to load data with more complex data structures from disks. We recommend that you evaluate access models and data models to prevent frequent requests to cold data.

- Q: Why does a write timeout occur?

  A: A write timeout may occur due to the following issues:

  - When the cold data is requested, it takes an extended period of time to load data from disks to the memory.
  - A large number of concurrent write requests are processed. After the data is modified in memory, background threads synchronize the modifications to disks. However, the write speed of disks is lower than that of the memory. When the data is written to memory at a much faster rate than to disks, a write speed limit must be set on the memory to prevent write timeouts.

Other FAQ

- Q: An `out of memory (OOM) error` occurred when I wrote data to a hybrid-storage instance. All memory is consumed, but the disks still have free space. How can I fix this issue?

A: The memory capacity is insufficient to store all keys and related metadata. Therefore, the memory capacity must be increased. For more information, see Change the configurations of an instance.

- Q: Can I change instance configurations to convert a hybrid-storage instance to a Community Edition instance or a performance-enhanced instance?

  A: No, you cannot change instance configurations to convert a hybrid-storage instance to a Community Edition instance or a performance-enhanced instance.

- Q: How do I migrate data from hybrid-storage instances to Community Edition instances or performance-enhanced instances?

  A: You can use the redis-shake tool to migrate full data. For more information, see Configure unidirectional data migration between ApsaraDB for Redis instances.

  > Note
  > - Hybrid storage instances use different encoding formats and replication protocols from Community Edition instances. You can use only the rump or sync mode to migrate full data. The migration of incremental data is not supported.
  > - You cannot use Data Transmission Service (DTS) to migrate data.

- Q: How do I export data from a hybrid storage instance to local storage?

  A: You can use the redis-shake tool to export RDB files in dump mode. If you use the Community Edition replication protocol to connect to a hybrid-storage instance, the hybrid-storage instance converts data to the RDB format and sends the data to the redis-shake tool. For more information, see Use the redis-shake tool to back up data.

- Q: Can I attach a Community Edition instance or a performance-enhanced instance of ApsaraDB for Redis Enhanced Edition (Tair) to a hybrid-storage instance as a secondary instance?

  A: Yes, you can perform this operation in specific scenarios. Hybrid-storage instances support only full data synchronization.

3.6. Extended data structures of ApsaraDB for Redis Enhanced Edition (Tair)

3.6.1. Extended data structures of ApsaraDB for Redis Enhanced Edition (Tair)

This topic describes data structures integrated into ApsaraDB for Redis Enhanced Edition (Tair) and compares the features of Tair and Redis Stack.

Similar to open source Redis, ApsaraDB for Redis Community Edition supports a variety of data structures such as strings, lists, hashes, sets, sorted sets, and streams. These data structures are sufficient for most development workloads but not complex workloads. To manage complex workloads, you must write large amounts of code or run Lua scripts.

ApsaraDB for Redis Enhanced Edition (Tair) instances integrate multiple self-developed data structures, including TairString (including CAS and CAD commands), TairHash, TairGIS, TairBloom, TairDoc, TairTS, TairCpc, TairZset, TairRoaring, and TairSearch. These data structures allow ApsaraDB for Redis to support more scenarios, simplify business development in complex scenarios and large amounts of code for better performance, and allow you to focus on business innovation.

> **Note**
> - Performance-enhanced instances support all data structures listed in the preceding table.
> - Persistent memory-optimized instances support TairString (including CAS and CAD commands) and TairCpc.

Extended data structures of Tair and modules of Redis Stack

Item	Extended data structure of Tair	Redis Stack Server	Description
String	• TairString • CAS and CAD commands	None	• A TairString is a string that contains a version number. Moreover, TairStings can be used to limit the range of results returned by the INCRBY and INCRBYFLOAT commands. These commands are used to increase or decrease the values of Redis strings. If a result is out of range, error messages are returned by these commands. This data structure is open-sourced. For more information, see TairString. • TairString commands include CAS and CAD commands. CAS and CAD commands can be used to implement simple and efficient Redis distributed locking. For more information about Redis distributed locking, see Implement high-performance distributed locks by using TairString. For information about the best practices, see Implement high-performance optimistic locking by using TairString and Implement bounded counters by using TairString.
Hash	TairHash	None	A TairHash is a hash that allows you to specify the expiration time and version number for a field. TairHash is more flexible in use and simplifies business development in most scenarios. This data structure is open-sourced. For more information, see TairHash.

Item	Extended data structure of Tair	Redis Stack Server	Description
ZSET	TairZset	None	TairZset allows DOUBLE-typed scores to be sorted with respect to 256 dimensions. You can use TairZset to implement and The module is open-sourced. For more information, see TairZset. regular leaderboardsdistributed leaderboards For information about the best practices, see Implement multidimensional leaderboards by using TairZset and Implement distributed leaderboards by using TairZset.
Doc (JSON)	TairDoc	RedisJSON	TairDoc is compatible with RedisJSON, supports JSONPath and JSON Pointer syntax, and allows conversion from the JSON format to the XML or YAML format.
Search	TairSearch	RedisSearch	TairSearch uses syntax similar to that of Elasticsearch but provides more and better tokenizers to improve query performance.
Geospatial	TairGIS	None	A TairGIS is a data structure that uses R-tree indexes and supports APIs related to a geographic information system (GIS). TairGIS allows points, linestrings, and polygons to be queried. You can use TairGIS to check whether A contains B, whether A is contained by B, or whether A intersects with B. For information about the best practices, see Monitor user trajectories by using TairGIS.
Time series	TairTS	RedisTimeSeries	Compared with RedisTimeSeries, TairTS extends the capability of tags. In the TairTS data structure, an extra hash layer is added to support your aggregate queries on timelines. You can also use TairTS to update or add data to historical time series data. For information about the best practices, see Implement fine-grained monitoring by using TairTS.

Item	Extended data structure of Tair	Redis Stack Server	Description
Sketch	TairBloom	RedisBloom	TairBloom is compatible with RedisBloom, supports dynamic scaling, and provides 64-bit hash algorithms to significantly reduce the probability of collision for large amounts of data. The best practices of TairBloom include the implementation of recommendation and crawler systems. For more information about the best practices, see TairBloom.
	TairCpc	None	TairCpc is a data structure developed based on the compressed probability counting (CPC) sketch. It allows you to perform high-performance computing on sampled data with a small amount of memory. TairCpc supports rolling and sliding windows to implement data streaming and common aggregation operators in big data analytics such as `DISTINCT`, `COUNT`, `MAX`, `MIN`, `FIRST`, `LAST`, and `SQUARED`.
Bitmap	TairRoaring	None	TairRoaring is an efficient computing module and provides high stability to support multi-bit computing power and improve performance and space complexity. For information about the best practices, see Select users by using TairRoaring.

FAQ

- Q: What do I do if a client does not support the commands that are provided by new modules?

  A: You can define the commands that are provided by new modules in your application code before you use these commands in your client. You can also use the TairJedis client that is developed by Alibaba Cloud based on Jedis. The TairJedis client allows you to directly use new modules. We recommend that you use TairJedis.

3.6.2. TairString

This topic describes TairString, a self-developed data structure that comes with ApsaraDB for Redis Enhanced Edition (Tair). A TairString is a string that contains a version number.

Overview

Native Redis strings use a key-value pair structure. TairStrings contain keys, values, as well as version numbers to support scenarios including the implementation of optimistic locking. In native Redis strings, the **INCRBY** and **INCRBYFLOAT** commands are used to increase or decrease string values. In TairStrings, you can limit the range of the outputs returned by these commands. If an output falls outside of the specified range, an error message is returned.

Main features

- A TairString contains a version number.
- TairStrings allow you to limit the range of the **INCRBY** and **INCRBYFLOAT** command outputs when you run these commands to increase values.

This module is open-sourced. For more information, see TairString.

Best practices

- Implement high-performance distributed locks by using TairString
- Implement high-performance optimistic locking by using TairString
- Implement bounded counters by using TairString

Prerequisites

The instance is a performance-enhanced or persistent memory-optimized instance of the ApsaraDB for Redis Enhanced Edition (Tair) whose minor version is 1.2.3 or later. For more information about performance-enhanced instances and persistent memory-optimized instances, see Performance-enhanced instances and Persistent memory-optimized instances. For more information about how to update the minor version, see Update the minor version.

Precautions

- The TairStrings that you want to manage are stored in the instance.

  > **Note** You can manage native Redis strings and TairStrings on the instance at the same time. However, native Redis strings do not support the commands described in this topic.

- Latest minor version provides more features and higher stability. We recommend that you update the instance to the latest minor version.

  > **Note** If your instance is a cluster instance or read/write splitting instance, we recommend that you update the proxy nodes in the instance to the latest minor version to ensure that all commands can be run as expected. For more information about cluster instances and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Supported commands

TairString commands

Command	Syntax	Description
EXSET	`EXSET key value [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version]`	Creates a TairString key if the key does not exist and writes a value to the key. If the key already exists, the command overwrites the value of the key.

Command	Syntax	Description
EXGET	`EXGET key`	Retrieves the value and version number of a TairString key.
EXSETVER	`EXSETVER key version`	Specifies the version number of a TairString key.
EXINCRBY	`EXINCRBY key num [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version] [MIN minval] [MAX maxval]`	Increases or decreases the value of a TairString key. The value of the num parameter must be of the long type.
EXINCRBYFLOAT	`EXINCRBYFLOAT key num [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version] [MIN minval] [MAX maxval]`	Increases or decreases the value of a TairString key. The value of the num parameter must be of the double type.
EXCAS	`EXCAS key newvalue version`	Updates the value of the specified TairString key if the current version number of the key matches the specified one. If the numbers do not match, the command returns the original value and version number of the key.
EXCAD	`EXCAD key version`	Deletes a key when the current version number of the key matches the specified one.
DEL	`DEL key [key ...]`	Deletes one or more TairString keys.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `A|B` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

EXSET

Item	Description
Syntax	`EXSET key value [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version]`

Item	Description
Time complexity	O(1)
Command description	Creates a TairString key if the key does not exist and writes a value to the key. If the key already exists, the command overwrites the value of the key.

Parameter	• Key: the key that you want to manage by running this command.
	• value: the value that you want to write to the key.
	• EX: the relative expiration time of the key. Unit: seconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• EXAT: the absolute expiration time of the key. Unit: seconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• PX: the relative expiration time of the key. Unit: milliseconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• PXAT: the absolute expiration time of the key. Unit: milliseconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• NX: specifies that the value is written to the key only if the key does not exist.
	• XX: specifies that the value is written to the key only if the key exists.
	• VER: the version number of the key.
	  ◦ If the key exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the value is written to the key and the version number is increased by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the key does not exist or if the current version number of the key is 0, the specified version number does not take effect. In this case, the value is written to the key and the version number is set to 1.
	• ABS: the absolute version number of the key. After this parameter is specified, the value is written to the key regardless of the current version number of the key. Then, the version number is overwritten with the ABS value.

Output	• If the operation is successful, OK is returned. • If the XX parameter is specified and the key does not exist, nil is returned. • If the NX parameter is specified and the key already exists, nil is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXSET foo bar NX ABS 100`
	Sample output:
	`OK`

EXGET

Item	Description
Syntax	`EXGET key`
Time complexity	O(1)
Command description	Retrieves the value and version number of a TairString key.
Parameter	• Key: the key that you want to manage by running this command.
Output	• If the operation is successful, the value and version number of the key are returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXGET foo`
	Sample output:
	`1) "bar"`
	`2) (integer) 1`

EXSETVER

Item	Description
Syntax	`EXSETVER key version`
Time complexity	O(1)

Item	Description
Command description	Specifies the version number of a TairString key.
Parameter	• Key: the key that you want to manage by running this command. • version: the version number that you want to specify.
Output	• If the operation is successful, a value of 1 is returned. • If the key does not exist, a value of 0 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXSETVER foo 2`
	Sample output:
	`(integer) 1`

EXINCRBY

Item	Description
Syntax	`EXINCRBY key num [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version] [MIN minval] [MAX maxval]`
Time complexity	O(1)
Command description	Increases or decreases the value of a TairString key. The value of the num parameter must be of the long type.

Item	Description

Parameter	• Key: the key that you want to manage by running this command.
	• num: the value by which the key is increased or decreased. This value must be an integer.
	• EX: the relative expiration time of the key. Unit: seconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• EXAT: the absolute expiration time of the key. Unit: seconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• PX: the relative expiration time of the key. Unit: milliseconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• PXAT: the absolute expiration time of the key. Unit: milliseconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• NX: specifies that the value is written to the key only if the key does not exist.
	• XX: specifies that the value is written to the key only if the key exists.
	• VER: the version number of the key.
	  ◦ If the key exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the value of the key is increased by the num value and the version number is increased by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the key does not exist or if the current version number of the key is 0, the specified version number does not take effect. In this case, the value is increased by the num value and the version number is set to 1.
	• ABS: the absolute version number of the key. After this parameter is specified, the value is written to the key regardless of the current version number of the key. Then, the version number is overwritten with the ABS value.
	• MIN: the minimum value of the key.
	• MAX: the maximum value of the key.

Output	• If the operation is successful, the updated value of the key is returned. • If the MAX or MIN parameter is specified and the updated value of the key is out of the value range, the "(error) ERR increment or decrement would overflow" error message is returned. • Otherwise, an error message is returned.

Example	The `EXSET foo 1` command is run in advance.
	Sample command:
	`EXINCRBY foo 100 MAX 300`
	Sample output:
	`(integer) 101`

EXINCRBYFLOAT

Item	Description
Syntax	`EXINCRBYFLOAT key num [EX time] [PX time] [EXAT time] [PXAT time] [NX | XX] [VER | ABS version] [MIN minval] [MAX maxval]`
Time complexity	O(1)
Command description	Increases or decreases the value of a TairString key. The value of the num parameter must be of the double type.

Parameter	• Key: the key that you want to manage by running this command.
	• num: the value by which the key is increased or decreased. The value must be a floating-point number.
	• EX: the relative expiration time of the key. Unit: seconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• EXAT: the absolute expiration time of the key. Unit: seconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• PX: the relative expiration time of the key. Unit: milliseconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• PXAT: the absolute expiration time of the key. Unit: milliseconds. A value of 0 indicates that the key immediately expires. If this parameter is not specified, the key does not expire.
	• NX: specifies that the value is written to the key only if the key does not exist.
	• XX: specifies that the value is written to the key only if the key exists.
	• VER: the version number of the key.
	  ◦ If the key exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the value of the key is increased by the num value and the version number is increased by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the key does not exist or if the current version number of the key is 0, the specified version number does not take effect. In this case, the value is increased by the num value and the version number is set to 1.
	• ABS: the absolute version number of the key. After this parameter is specified, the value is written to the key regardless of the current version number of the key. Then, the version number is overwritten with the ABS value.
	• MIN: the minimum value of the key.
	• MAX: the maximum value of the key.

Output	• If the operation is successful, the updated value of the key is returned. • If the MAX or MIN parameter is specified and the updated value of the key is out of the value range, the "(error) ERR increment or decrement would overflow" error message is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `EXSET foo 1` command is run in advance.
	Sample command:
	`EXINCRBYFLOAT foo 10.123`
	Sample output:
	`(integer) 11.123`

EXCAS

Item	Description
Syntax	`EXCAS` *key newvalue version*
Time complexity	O(1)
Command description	Updates the value of the specified TairString key if the current version number of the key matches the specified one. If the numbers do not match, the command returns the original value and version number of the key.
Parameter	• Key: the key that you want to manage by running this command. • newvalue: the new value that you want to use to overwrite the current value of the key if the current version number of the key matches the specified one. • version: the version number that you want to compare with the current version number of the key.
Output	• If the operation is successful, the "["OK", "",latest version]" message is returned. The "" characters in the middle of the message indicate empty strings that do not carry meanings. • If the operation fails, the following error message is returned: ["ERR update version is stale", value, version]. value indicates the current value of the key. version indicates the current version number of the key. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `EXSET foo bar` command is run in advance.
	Sample command:
	`EXCAS foo bzz 1`
	Sample output:
	`1) OK`
	`2)`
	`3) (integer) 2`

EXCAD

Item	Description
Syntax	`EXCAD key version`
Time complexity	O(1)
Command description	Deletes a key when the current version number of the key matches the specified one.
Parameter	• Key: the key that you want to manage by running this command. • version: the version number that you want to compare with the current version number of the key.
Output	• If the operation is successful, a value of 1 is returned. • If the operation fails, a value of 0 is returned. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Example	The `EXSET foo bar` command is run in advance.
	Sample command:
	`EXCAD foo 1`
	Sample output:
	`(integer) 1`

3.6.3. CAS and CAD commands

This topic describes the commands that you can run on ApsaraDB for Redis Enhanced Edition (Tair) instances to enhance the string function. These commands include Compare And Set (CAS) and Compare And Delete (CAD).

Prerequisites

The instance is a performance-enhanced or persistent memory-optimized instance of the ApsaraDB for Redis Enhanced Edition (Tair) whose minor version is 1.2.3 or later. For more information about performance-enhanced instances and persistent memory-optimized instances, see Performance-enhanced instances and Persistent memory-optimized instances. For more information about how to update the minor version, see Update the minor version.

Precautions

- In this topic, the strings that you want to manage are native Redis strings.

  > **Note** You can manage Redis strings and TairStrings on an ApsaraDB for Redis Enhanced Edition (Tair) instance. However, CAS and CAD commands are applicable only to Redis strings.

- Latest minor version provides more features and higher stability. We recommend that you update the instance to the latest minor version.

  > **Note** If your instance is a cluster instance or read/write splitting instance, we recommend that you update the proxy nodes in the instance to the latest minor version to ensure that all commands can be run as expected. For more information about cluster instances and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Supported commands

Commands that enhance the string function

Command	Syntax	Description
CAS	`CAS key oldvalue newvalue`	Changes the existing value of a specified key if the existing value matches a specified one. Otherwise, the existing value remains unchanged. **Note** This command applies only to Redis strings. To change TairString values, run the EXCAS command.

Command	Syntax	Description
CAD	`CAD key value`	Deletes a specified key if the existing value of the key matches a specified one. Otherwise, the key is not deleted. **Note** This command applies only to Redis strings. To delete TairString keys, run the EXCAD command.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

CAS

Item	Description
Syntax	`CAS key oldvalue newvalue`
Time complexity	O(1)
Command description	Changes the existing value of a specified key if the existing value matches a specified one. Otherwise, the existing value remains unchanged. **Note** This command applies only to Redis strings. To change TairString values, run the EXCAS command.
Parameter	• Key: the key of the Redis string that you want to manage by running the command. • oldvalue: the specified value that you want to compare with the existing value of the key. • newvalue: the new value to which you want to change the existing value of the key if the existing value matches the specified one.
Output	• If the operation is successful, a value of 1 is returned. • If the operation fails, a value of 0 is returned. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `SET foo bar` command is run in advance.
	Sample command:
	`CAS foo bar bzz`
	Sample output:
	`(integer) 1`
	If you run the `GET foo` command after this command is run, `"bzz"` is returned.

CAD

Item	Description
Syntax	`CAD key value`
Time complexity	O(1)
Command description	Deletes a specified key if the existing value of the key matches a specified one. Otherwise, the key is not deleted. **Note** This command applies only to Redis strings. To delete TairString keys, run the EXCAD command.
Parameter	• Key: the key of the Redis string that you want to manage by running the command. • value: the specified value that you want to compare with the existing value of the key.
Output	• If the operation is successful, a value of 1 is returned. • If the operation fails, a value of 0 is returned. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `SET foo bar` command is run in advance.
	Sample command:
	`CAD foo bar`
	Sample output:
	`(integer) 1`
	After this command is run, the foo key is deleted. Then, if you run the `GET foo` command, `(nil)` is returned.

3.6.4. TairHash

TairHash is a hash data type that allows expiration times and version numbers to be specified for fields. This makes TairHash more flexible and simplifies business development in most scenarios.

Overview

TairHashes and native Redis hashes support a variety of data interfaces and provide high performance in data processing. However, native Redis hashes allow you to specify expiration times only for keys. TairHashes allow you to specify expiration times for both keys and fields. You can also use TairHashes to specify the version numbers of fields. These improved features of TairHashes add flexibility to TairHash and simplify business development in most scenarios. TairHashes use the efficient active expiration algorithm to check the expiration times of fields and delete expired fields. This process does not increase the database response time.

Main features

- The expiration time and version number for each field can be specified.
- The efficient and flexible active and passive expiration strategies for fields are supported.
- TairHashes and native Redis hashes use similar syntax.

This module is open-sourced. For more information, see TairHash.

Prerequisites

The instance is a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair). For more information about performance-enhanced instances, see Performance-enhanced instances.

> **Note** The latest minor version provides more features and higher stability. We recommend that you update your instance to the latest minor version. For more information, see Update the minor version. If your instance is a cluster or read/write splitting instance, we recommend that you update the proxy nodes in the instance to the latest minor version. This ensures that all commands can be run as expected. For more information about cluster and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Precautions

The TairHash data that you want to manage is stored on the performance-enhanced instance.

Supported commands

TairHash commands

Command	Syntax	Description
EXHSET	`EXHSET key field value [EX time] [EXAT time] [PX time] [PXAT time] [NX | XX] [VER | ABS version] [KEEPTTL]`	Adds a field to a TairHash key. If the key does not exist, the key is created. If the field has an existing value, this command overwrites the existing value. **Note** If you do not set an expiration time for the field that has an expiration time when you run this command, the field does not expire.
EXHGET	`EXHGET key field`	Retrieves the value of a field from a TairHash key. If the key or field does not exist, nil is returned.
EXHMSET	`EXHMSET key field value [field value ...]`	Adds multiple fields to a TairHash key. If the key does not exist, the key is created. If these fields already exist in the key, this command overwrites the values of these fields.
EXHPEXPIREAT	`EXHPEXPIREAT key field milliseconds-timestamp [VER | ABS version]`	Specifies the absolute expiration time of a field in a TairHash key. Unit: milliseconds.
EXHPEXPIRE	`EXHPEXPIRE key field milliseconds [VER | ABS version]`	Specifies the relative expiration time of a field in a TairHash key. Unit: milliseconds.
EXHEXPIREAT	`EXHEXPIREAT key field timestamp [VER | ABS version]`	Specifies the absolute expiration time of a field in a TairHash key. Unit: seconds.
EXHEXPIRE	`EXHEXPIRE key field seconds [VER | ABS version]`	Specifies the relative expiration time of a field in a TairHash key. Unit: seconds.
EXHPTTL	`EXHPTTL key field`	Queries the remaining expiration time of a field in a TairHash key. Unit: milliseconds.
EXHTTL	`EXHTTL key field`	Queries the expiration time of a field in a TairHash key. Unit: seconds.

Command	Syntax	Description
EXHVER	`EXHVER key field`	Queries the current version number of a field in a TairHash key.
EXHSETVER	`EXHSETVER key field version`	Specifies the version number of a field in a TairHash key.
EXHINCRBY	`EXHINCRBY key field num [EX time] [EXAT time] [PX time] [PXAT time] [VER | ABS version] [MIN minval] [MAX maxval] [KEEPTTL]`	Increases the value of a field by the num value in a TairHash key. The num value must be an integer. If the key does not exist, the key is created. If the field does not exist, this command adds the field and sets the value of the field to 0 before increasing the value of the field. **Note** If you do not set an expiration time for the field that has an expiration time when you run this command, the field does not expire.
EXHINCRBYFLOAT	`EXHINCRBYFLOAT key field num [EX time] [EXAT time] [PX time] [PXAT time] [VER | ABS version] [MIN minval] [MAX maxval] [KEEPTTL]`	Increases the value of a field by the num value in a TairHash key. The num value must be a floating-point number. If the key does not exist, the key is created. If the field does not exist, this command adds the field and sets the value of the field to 0 before increasing the value of the field. **Note** If you do not set an expiration time for the field that has an expiration time when you run this command, the field does not expire.
EXHGETWITHVER	`EXHGETWITHVER key field`	Retrieves the value and version number of a field from a TairHash key. If the key or field does not exist, nil is returned.
EXHMGET	`EXHMGET key field [field ...]`	Retrieves multiple field values from a TairHash key in each query. If the key does not exist or if the fields do not exist, nil is returned.
EXHMGETWITHVER	`EXHMGETWITHVER key field [field ...]`	Retrieves the values and version numbers of multiple fields from a TairHash key in each query.

Command	Syntax	Description
EXHLEN	`EXHLEN key [NOEXP]`	Retrieves the number of fields in a TairHash key. The output may include the number of expired fields that are not deleted because this command does not trigger a passive eviction of or filter out expired fields. If you want to retrieve only the number of fields that have not expired, you can set the NOEXP parameter in your command.
EXHEXISTS	`EXHEXISTS key field`	Checks whether a field exists in a TairHash key.
EXHSTRLEN	`EXHSTRLEN key field`	Retrieves the length of a field value from a TairHash key.
EXHKEYS	`EXHKEYS key`	Retrieves all fields from a TairHash key. This command filters out expired fields but does not delete them so that the response time of this command is not increased.
EXHVALS	`EXHVALS key`	Retrieves the values of all fields from a TairHash key. This command filters out expired fields but does not delete them so that the response time of this command is not increased.
EXHGETALL	`EXHGETALL key`	Retrieves all fields from a TairHash key and their values. This command filters out expired fields but does not delete these fields so that the response time of this command is not increased.
EXHSCAN	`EXHSCAN key op subkey [MATCH pattern] [COUNT count]`	Scans a TairHash key. You can set the op parameter to values such as >, >=, <, <=, ==, ^, and $. This parameter specifies a scan method. When you run this command, you can also set the MATCH parameter to specify a regular expression to filter subkeys and set the COUNT parameter to limit the number of return values. If you do not set the COUNT parameter, the default value 10 is used. This command filters out expired fields but does not delete them so that the response time of this command is not increased.

Command	Syntax	Description
EXHDEL	`EXHDEL key field [field ...]`	Deletes a field from a TairHash key. If the key or field does not exist, a value of 0 is returned. If the field is deleted, a value of 1 is returned.
DEL	`DEL <key> [key ...]`	Deletes one or more TairHash keys.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

EXHSET

Item	Description
Syntax	`EXHSET key field value [EX time] [EXAT time] [PX time] [PXAT time] [NX | XX] [VER | ABS version] [KEEPTTL]`
Time complexity	O(1)
Command description	Adds a field to a TairHash key. If the key does not exist, the key is created. If the field has an existing value, this command overwrites the existing value. **Note** If you do not set an expiration time for the field that has an expiration time when you run this command, the field does not expire.

Item	Description

Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
	• field: an element of the key. A key can contain multiple fields.
	• value: the value of the field. A field can have only a single value.
	• EX: the relative expiration time of the field. Unit: seconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• EXAT: the absolute expiration time of the field. Unit: seconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• PX: the relative expiration time of the field. Unit: milliseconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• PXAT: the absolute expiration time of the field. Unit: milliseconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• NX: specifies that the field is added only if the field does not exist.
	• XX: specifies that the field is added only if the field exists.
	• VER: the version number of the field.
	  ◦ If the field exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the system continues to run the command and increases the version number by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the field does not exist or the current version number of the field is 0, this parameter is ignored and this command continues to run. After the operation succeeds, the version number is set to 1.
	• ABS: the absolute version number of the field. If you set this parameter, the system forcefully adds the field to the key regardless of whether the field already exists. When the field is added, the version number is set to the specified ABS value.
	• KEEPTTL: retains the current time-to-live (TTL) of the field if none of the *EX, EXAT, PX, and PXAT* parameters are specified.
	Note By default, if the KEEPTTL parameter is not specified, the EXHSET command deletes the current TTL of the field.

Output	• If a field is created and a value is set for the field, a value of 1 is returned. • If the field already exists and the specified value overwrites the current value of the field, a value of 0 is returned. • If the XX parameter is specified and the field does not exist, a value of -1 is returned. • If the NX parameter is specified and the field exists, a value of -1 is returned. • If the VER value does not match the current version number, the "ERR update version is stale" error message is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXHSET myhash field1 val EX 10`
	Sample output:
	`(integer) 1`

EXHGET

Item	Description
Syntax	`EXHGET key field`
Time complexity	O(1)
Command description	Retrieves the value of a field from a TairHash key. If the key or field does not exist, nil is returned.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.
Output	• If the field exists and the operation is successful, the value of the field is returned. • If the key or field does not exist, nil is returned. • Otherwise, an error message is returned.

Example	The `EXHSET myhash field1 val` command is run in advance.
	Sample command:
	`EXHGET myhash field1`
	Sample output:
	`"val"`

EXHMSET

Item	Description
Syntax	`EXHMSET key field value [field value ...]`

Item	Description
Time complexity	O(N)
Command description	Adds multiple fields to a TairHash key. If the key does not exist, the key is created. If these fields already exist in the key, this command overwrites the values of these fields.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields. • value: the value of the field. A field can have only a single value.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHMSET myhash field1 val1 field2 val2`
	Sample output:
	`OK`

EXHPEXPIREAT

Item	Description
Syntax	`EXHPEXPIREAT` *key field milliseconds-timestamp* `[VER | ABS` *version*`]`
Time complexity	O(1)
Command description	Specifies the absolute expiration time of a field in a TairHash key. Unit: milliseconds.

Item	Description

Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
	• field: an element of the key. A key can contain multiple fields.
	• milliseconds-timestamp: a timestamp, which must be accurate to the millisecond.
	• VER: the version number of the field.
	  ◦ If the field exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the system continues to run the command and increases the version number by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the field does not exist or the current version number of the field is 0, this parameter is ignored and this command continues to run. After the operation succeeds, the version number is set to 1.
	• ABS: the absolute version number of the field. If you set this parameter, the system forcefully adds the field to the key regardless of whether the field already exists. When the field is added, the version number is set to the specified ABS value.

Output	• If the field exists and a value is set for the field, a value of 1 is returned. • If the field does not exist, a value of 0 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHPEXPIREAT myhash field1 1293840000`
	Sample output:
	`(integer) 1`

EXHPEXPIRE

Item	Description
Syntax	`EXHPEXPIRE` *key field milliseconds* `[VER | ABS` *version*`]`
Time complexity	O(1)
Command description	Specifies the relative expiration time of a field in a TairHash key. Unit: milliseconds.

Item	Description

Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
	• field: an element of the key. A key can contain multiple fields.
	• milliseconds: the relative expiration time. Unit: milliseconds.
	• VER: the version number of the field.
	  ◦ If the field exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the system continues to run the command and increases the version number by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the field does not exist or the current version number of the field is 0, this parameter is ignored and this command continues to run. After the operation succeeds, the version number is set to 1.
	• ABS: the absolute version number of the field. If you set this parameter, the system forcefully adds the field to the key regardless of whether the field already exists. When the field is added, the version number is set to the specified ABS value.

Output	• If the field exists and a value is set for the field, a value of 1 is returned. • If the field does not exist, a value of 0 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHPEXPIRE myhash field1 1000`
	Sample output:
	`(integer) 1`

EXHEXPIREAT

Item	Description
Syntax	`EXHEXPIREAT` *key field timestamp* `[VER | ABS` *version*`]`
Time complexity	O(1)
Command description	Specifies the absolute expiration time of a field in a TairHash key. Unit: seconds.

Item	Description

Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
	• field: an element of the key. A key can contain multiple fields.
	• timestamp: a timestamp, which must be accurate to the millisecond.
	• VER: the version number of the field.
	  ◦ If the field exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the system continues to run the command and increases the version number by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the field does not exist or the current version number of the field is 0, this parameter is ignored and this command continues to run. After the operation succeeds, the version number is set to 1.
	• ABS: the absolute version number of the field. If you set this parameter, the system forcefully adds the field to the key regardless of whether the field already exists. When the field is added, the version number is set to the specified ABS value.

Output	• If the field exists and a value is set for the field, a value of 1 is returned. • If the field does not exist, a value of 0 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHEXPIREAT myhash field1 1293840000`
	Sample output:
	`(integer) 1`

EXHEXPIRE

Item	Description
Syntax	`EXHEXPIRE` *key field seconds* `[VER | ABS` *version*`]`
Time complexity	O(1)
Command description	Specifies the relative expiration time of a field in a TairHash key. Unit: seconds.

Item	Description

Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
	• field: an element of the key. A key can contain multiple fields.
	• seconds: the relative expiration time. Unit: seconds.
	• VER: the version number of the field.
	  ◦ If the field exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the system continues to run the command and increases the version number by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the field does not exist or the current version number of the field is 0, this parameter is ignored and this command continues to run. After the operation succeeds, the version number is set to 1.
	• ABS: the absolute version number of the field. If you set this parameter, the system forcefully adds the field to the key regardless of whether the field already exists. When the field is added, the version number is set to the specified ABS value.

Output	• If the field exists and a value is set for the field, a value of 1 is returned. • If the field does not exist, a value of 0 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHEXPIRE myhash field1 100`
	Sample output:
	`(integer) 1`

EXHPTTL

Item	Description
Syntax	`EXHPTTL` *key field*
Time complexity	O(1)
Command description	Queries the remaining expiration time of a field in a TairHash key. Unit: milliseconds.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.

Item	Description
Output	• If the key or field does not exist, a value of -2 is returned. • If the field exists and the field does not have an expiration time, a value of -1 is returned. • If the field exists and the field has an expiration time, the expiration time is returned. Unit: milliseconds. • Otherwise, an error message is returned.

Example	The `EXHSET myhash field1 val1 EX 100` command is run in advance.
	Sample command:
	`EXHPTTL myhash field1`
	Sample output:
	`(integer) 97213`

EXHTTL

Item	Description
Syntax	`EXHTTL key field`
Time complexity	O(1)
Command description	Queries the expiration time of a field in a TairHash key. Unit: seconds.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.
Output	• If the key or field does not exist, a value of -2 is returned. • If the field exists and the field does not have an expiration time, a value of -1 is returned. • If the field exists and the field has an expiration time, the expiration time is returned. Unit: seconds. • Otherwise, an error message is returned.

Item	Description

Example	The `EXHSET myhash field1 val1 EX 100` command is run in advance.
	Sample command:
	`EXHTTL myhash field1`
	Sample output:
	`(integer) 85`

EXHVER

Item	Description
Syntax	`EXHVER key field`
Time complexity	O(1)
Command description	Queries the current version number of a field in a TairHash key.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.
Output	• If the key does not exist, a value of -1 is returned. • If the field does not exist, a value of -2 is returned. • If the operation is successful, the version number of the field is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHVER myhash field1`
	Sample output:
	`(integer) 1`

EXHSETVER

Item	Description
Syntax	`EXHSETVER key field version`
Time complexity	O(1)
Command description	Specifies the version number of a field in a TairHash key.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.
Output	• If the key or field does not exist, a value of 0 is returned. • If the version number is specified, a value of 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHSETVER myhash field1 3`
	Sample output:
	`(integer) 1`

EXHINCRBY

Item	Description
Syntax	`EXHINCRBY key field num [EX time] [EXAT time] [PX time] [PXAT time] [VER | ABS version] [MIN minval] [MAX maxval] [KEEPTTL]`
Time complexity	O(1)
Command description	Increases the value of a field by the num value in a TairHash key. The num value must be an integer. If the key does not exist, the key is created. If the field does not exist, this command adds the field and sets the value of the field to 0 before increasing the value of the field. **Note** If you do not set an expiration time for the field that has an expiration time when you run this command, the field does not expire.

Item	Description

Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
	• field: an element of the key. A key can contain multiple fields.
	• num: the integer by which you want to increase the value of the field.
	• EX: the relative expiration time of the field. Unit: seconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• EXAT: the absolute expiration time of the field. Unit: seconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• PX: the relative expiration time of the field. Unit: milliseconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• PXAT: the absolute expiration time of the field. Unit: milliseconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• VER: the version number of the field.
	  ◦ If the field exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the system continues to run the command and increases the version number by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the field does not exist or the current version number of the field is 0, this parameter is ignored and this command continues to run. After the operation succeeds, the version number is set to 1.
	• ABS: the absolute version number of the field. If you set this parameter, the system forcefully adds the field to the key regardless of whether the field already exists. When the field is added, the version number is set to the specified ABS value.
	• MIN: the minimum value of the field. If the field value is less than this lower limit, an error message is returned.
	• MAX: the maximum value of the field. If the field value is greater than this upper limit, an error message is returned.
	• KEEPTTL: retains the current TTL of the field if none of the *EX, EXAT, PX, and PXAT* parameters are specified.

Output	• If the operation is successful, the value increased by the num value is returned. • Otherwise, an error message is returned.

Example	The `EXHMSET myhash field1 10` command is run in advance.
	Sample command:
	`EXHINCRBY myhash field1 100`
	Sample output:
	`(integer) 110`

EXHINCRBYFLOAT

Item	Description
Syntax	`EXHINCRBYFLOAT key field num [EX time] [EXAT time] [PX time] [PXAT time] [VER | ABS version] [MIN minval] [MAX maxval] [KEEPTTL]`
Time complexity	O(1)
Command description	Increases the value of a field by the num value in a TairHash key. The num value must be a floating-point number. If the key does not exist, the key is created. If the field does not exist, this command adds the field and sets the value of the field to 0 before increasing the value of the field. **Note** If you do not set an expiration time for the field that has an expiration time when you run this command, the field does not expire.

Item	Description

Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
	• field: an element of the key. A key can contain multiple fields.
	• num: the floating-point number by which you want to increase the value for the field.
	• EX: the relative expiration time of the field. Unit: seconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• EXAT: the absolute expiration time of the field. Unit: seconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• PX: the relative expiration time of the field. Unit: milliseconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• PXAT: the absolute expiration time of the field. Unit: milliseconds. A value of 0 indicates that the field immediately expires. If this parameter is not specified, the field does not expire.
	• VER: the version number of the field.
	  ◦ If the field exists, the version number specified by this parameter is matched against the current version number.
	  ▪ If the version numbers match, the system continues to run the command and increases the version number by 1.
	  ▪ If the version numbers do not match, an error message is returned.
	  ◦ If the field does not exist or the current version number of the field is 0, this parameter is ignored and this command continues to run. After the operation succeeds, the version number is set to 1.
	• ABS: the absolute version number of the field. If you set this parameter, the system forcefully adds the field to the key regardless of whether the field already exists. When the field is added, the version number is set to the specified ABS value.
	• MIN: the minimum value of the field. If the field value is less than this lower limit, an error message is returned.
	• MAX: the maximum value of the field. If the field value is greater than this upper limit, an error message is returned.
	• KEEPTTL: retains the current TTL of the field if none of the *EX, EXAT, PX, and PXAT* parameters are specified.

Output	• If the operation is successful, the value increased by the num value is returned. • Otherwise, an error message is returned.

Example	The `EXHMSET myhash field1 10` command is run in advance.
	Sample command:
	`EXHINCRBYFLOAT myhash field1 9.235`
	Sample output:
	`"19.235"`

EXHGETWITHVER

Item	Description
Syntax	`EXHGETWITHVER key field`
Time complexity	O(1)
Command description	Retrieves the value and version number of a field from a TairHash key. If the key or field does not exist, nil is returned.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.
Output	• If the field exists and the operation is successful, the value and version number of the field are returned. • If the key or field does not exist, nil is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHGETWITHVER myhash field1`
	Sample output:
	`1) "19.235"`
	`2) (integer) 5`

EXHMGET

Item	Description
Syntax	`EXHMGET key field [field ...]`
Time complexity	O(1)
Command description	Retrieves multiple field values from a TairHash key in each query. If the key does not exist or if the fields do not exist, nil is returned.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.

Item	Description
Output	• If the key does not exist, nil is returned. • If the key and fields exist, an array is returned. Each element in the array corresponds to a field value. • If the key exists and specific fields do not exist, an array is returned. Each element in the array corresponds to a field value. The elements that correspond to the fields that do not exist are displayed as nil. • Otherwise, an error message is returned.

Example	The `EXHMSET myhash field1 10 field2 var1` command is run in advance.
	Sample command:
	`EXHMGET myhash field1 field2`
	Sample output:
	`1) "10"`
	`2) "var1"`

EXHMGETWITHVER

Item	Description
Syntax	`EXHMGETWITHVER key field [field ...]`
Time complexity	O(1)
Command description	Retrieves the values and version numbers of multiple fields from a TairHash key in each query.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.
Output	• If the key does not exist, nil is returned. • If the key and fields exist, an array is returned. Each element in the array corresponds to a field value and a version number. • If the key exists and specific fields do not exist, an array is returned. Each element in the array corresponds to a field value and a field version number. The elements that correspond to the fields that do not exist are displayed as nil. • Otherwise, an error message is returned.

Item	Description

Example	The `EXHMSET myhash field1 10 field2 var1` command is run in advance.
	Sample command:
	`EXHMGETWITHVER myhash field1 field2`
	Sample output:
	`1) 1) "10"`
	`2) (integer) 1`
	`2) 1) "var1"`
	`2) (integer) 1`

EXHLEN

Item	Description
Syntax	`EXHLEN key [NOEXP]`
Time complexity	The time complexity is O(1) if the *NOEXP* parameter is not specified and O(N) if the *NOEXP* parameter is specified.
Command description	Retrieves the number of fields in a TairHash key. The output may include the number of expired fields that are not deleted because this command does not trigger a passive eviction of or filter out expired fields. If you want to retrieve only the number of fields that have not expired, you can set the NOEXP parameter in your command.

Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
	• NOEXP: specifies to return the number of fields that have not expired. By default, this command does not filter out or passively evict expired fields. For this reason, the output of this command may include the number of expired fields that are not deleted. If you want to retrieve only the number of fields that have not expired, you can set the *NOEXP* parameter. When you set the *NOEXP* parameter, take note of the following items:
	  ◦ The response time of the **EXHLEN** command is determined by the size of the key because the system scans all data in the key.
	  ◦ The output of the **EXHLEN** command does not include the number of expired fields because this command filters out expired fields but does not evict them.

Output	• If the key or field does not exist, a value of 0 is returned. • If the operation is successful, the number of fields in the key is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXHLEN myhash`
	Sample output:
	`(integer) 2`

EXHEXISTS

Item	Description
Syntax	`EXHEXISTS key field`
Time complexity	O(1)
Command description	Checks whether a field exists in a TairHash key.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.
Output	• If the key or field does not exist, a value of 0 is returned. • If the field exists, a value of 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHEXISTS myhash field1`
	Sample output:
	`(integer) 1`

EXHSTRLEN

Item	Description
Syntax	`EXHSTRLEN key field`
Time complexity	O(1)

Item	Description
Command description	Retrieves the length of a field value from a TairHash key.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.
Output	• If the key or field does not exist, a value of 0 is returned. • If the operation is successful, the length of the field value is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXHSTRLEN myhash field1`
	Sample output:
	`(integer) 2`

EXHKEYS

Item	Description
Syntax	`EXHKEYS key`
Time complexity	O(N)
Command description	Retrieves all fields from a TairHash key. This command filters out expired fields but does not delete them so that the response time of this command is not increased.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
Output	• If the key does not exist, an empty array is returned. • If the key exists, an array is returned. Each element in the array corresponds to a field in the key. • Otherwise, an error message is returned.

Item	Description

Example	The `EXHMSET myhash field1 10 field2 var1` command is run in advance.
	Sample command:
	`EXHKEYS myhash`
	Sample output:
	`1) "field1"`
	`2) "field2"`

EXHVALS

Item	Description
Syntax	`EXHVALS key`
Time complexity	O(N)
Command description	Retrieves the values of all fields from a TairHash key. This command filters out expired fields but does not delete them so that the response time of this command is not increased.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
Output	• If the key does not exist, an empty array is returned. • If the key exists, an array is returned. Each element in the array corresponds to a field value in the key. • Otherwise, an error message is returned.

Example	The `EXHMSET myhash field1 10 field2 var1` command is run in advance.
	Sample command:
	`EXHVALS myhash`
	Sample output:
	`1) "10"`
	`2) "var1"`

EXHGETALL

Item	Description
Syntax	`EXHGETALL key`
Time complexity	O(N)
Command description	Retrieves all fields from a TairHash key and their values. This command filters out expired fields but does not delete these fields so that the response time of this command is not increased.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
Output	• If the key does not exist, an empty array is returned. • If the key exists, an array is returned. Each element in the array corresponds to a field-value pair in the key. • Otherwise, an error message is returned.

Example	The `EXHMSET myhash field1 10 field2 var1` command is run in advance.
	Sample command:
	`EXHGETALL myhash`
	Sample output:
	`1) "field1"`
	`2) "10"`
	`3) "field2"`
	`4) "var1"`

EXHSCAN

Item	Description
Syntax	`EXHSCAN key op subkey [MATCH pattern] [COUNT count]`
Time complexity	The time complexity is O(1) for every call and O(N) for a complete iteration.
Command description	Scans a TairHash key. You can set the op parameter to values such as >, >=, <, <=, ==, ^, and $. This parameter specifies a scan method. When you run this command, you can also set the MATCH parameter to specify a regular expression to filter subkeys and set the COUNT parameter to limit the number of return values. If you do not set the COUNT parameter, the default value 10 is used. This command filters out expired fields but does not delete them so that the response time of this command is not increased.

Item	Description

Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command.
	• op: the position from which a scan starts. Valid values:
	  ○ >: The scan starts from the first field that has a value greater than the subkey value.
	  ○ >=: The scan starts from the first field that has a value greater than or equal to the subkey value.
	  ○ <: The scan starts from the first field that has a value less than the subkey value.
	  ○ <=: The scan starts from the first field that has a value less than or equal to the subkey value.
	  ○ ==: The scan starts from the first field that has a value equal to the subkey value.
	  ○ ^: The scan starts from the first field.
	  ○ $: The scan starts from the last field.
	• subkey: a parameter that is used together with the op parameter to determine the position from which a scan starts. If the op parameter is set to ^ or $, this parameter is ignored.
	• MATCH: a pattern that is used to filter scan results.
	• COUNT: specifies the number of fields that can be scanned in each query. The default value is 10.
	Note The COUNT parameter specifies the number of fields scanned in each query and does not ensure that the number of fields returned is the number of fields scanned. The number of fields returned is determined by the number of fields that exist in the key and whether the MATCH parameter is specified.

Output	• If the key does not exist, an empty array is returned.
	• If the key exists, an array that contains two elements is returned.
	  ○ The first element is the first field from which the next scan starts. If the key scan has been complete, this element is left empty.
	  ○ The second element is the scan result which includes fields and values.
	• Otherwise, an error message is returned.

Item	Description

Example	The `EXHMSET myhashkey field1 val1 field2 val2 field3 val3 field4 val4 field5 val5` command is run in advance.
	Sample command:
	`EXHSCAN myhashkey ^ xx COUNT 3`
	Sample output:
	`1) "field4"`
	`2) 1) "field1"`
	`2) "val1"`
	`3) "field2"`
	`4) "val2"`
	`5) "field3"`
	`6) "val3"`

EXHDEL

Item	Description
Syntax	`EXHDEL key field [field ...]`
Time complexity	O(1)
Command description	Deletes a field from a TairHash key. If the key or field does not exist, a value of 0 is returned. If the field is deleted, a value of 1 is returned.
Parameter	• Key: the key that specifies the piece of TairHash data that you want to manage by running this command. • field: an element of the key. A key can contain multiple fields.
Output	• If the key or field does not exist, a value of 0 is returned. • If the operation is successful, a value of 1 is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXHDEL myhash field1`
	Sample output:
	`(integer) 1`

3.6.5. TairGIS

TairGIS is a data structure that uses R-tree indexes and supports APIs related to a geographic information system (GIS). Compared with native Redis GEO commands that allow you to use Geohash and Redis Sorted Set to query points, TairGIS provides more features by allowing you to query points, linestrings, and polygons.

Main features

- R-tree indexes for query and storage.
- Line and polygon queries, including queries for the intersection of sets.
- GIS.SEARCH command, which functions like native Redis `GEORADIUS` command.

Best practices

Monitor user trajectories by using TairGIS

Prerequisites

The instance is a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair). For more information about performance-enhanced instances, see Performance-enhanced instances.

> **Note** The latest minor version provides more features and higher stability. We recommend that you update your instance to the latest minor version. For more information, see Update the minor version. If your instance is a cluster or read/write splitting instance, we recommend that you update the proxy nodes in the instance to the latest minor version. This ensures that all commands can be run as expected. For more information about cluster and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Precautions

The TairGIS data that you want to manage is stored on the performance-enhanced instance.

Supported commands

TairGIS commands

Command	Syntax	Description
GIS.ADD	`GIS.ADD area polygonName polygonWkt [polygonName polygonWkt ...]`	Adds one or more specific polygons to a specific area. The polygons are described in Well Known Text (WKT). **Note** WKT is a text markup language for representing vector geometry objects on a map, spatial reference systems of spatial objects, and transformations between spatial reference systems.
GIS.GET	`GIS.GET area polygonName`	Retrieves the WTK of a specific polygon within a specific area.
GIS.GETALL	`GIS.GETALL area [WITHOUTWKT]`	Retrieves the names and WKTs of all polygons within a specific area. If you specify the WITHOUTWKT parameter, only the names of the polygons are returned.
GIS.CONTAINS	`GIS.CONTAINS area polygonWkt [WITHOUTWKT]`	Checks whether a specified point, linestring, or polygon is located in polygons within a specific area. If yes, this command returns the number and WKTs of polygons that contain the point, linestring, or polygon in the area.
GIS.WITHIN	`GIS.WITHIN area polygonWkt [WITHOUTWKT]`	Checks whether a specific area is located within a specified point, linestring, or polygon. If yes, this command returns the number and WKTs of polygons that are located within the point, linestring, or polygon.
GIS.INTERSECTS	`GIS.INTERSECTS area polygonWkt`	Checks whether a specific point, linestring, or polygon intersects with polygons within a specific area. If yes, this command returns the number and WKTs of polygons within the area that intersect with the point, linestring, or polygon.
GIS.SEARCH	`GIS.SEARCH area [RADIUS longitude latitude distance MKMFTMI] [MEMBER field distance MKMFTMI] [GEOM geom] [COUNT count] [ASC|DESC] [WITHDIST] [WITHOUTWKT]`	Queries the points within a specific area that are located within a specific radius of a specific longitude and latitude position.
GIS.DEL	`GIS.DEL area polygonName`	Deletes a specific polygon from a specific area.

Command	Syntax	Description
DEL	`DEL key [key ...]`	Deletes one or more TairGIS keys. This is a native Redis command.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

GIS.ADD

Item	Description
Syntax	`GIS.ADD area polygonName polygonWkt [polygonName polygonWkt ...]`
Time complexity	O(log n)
Command description	Adds one or more specific polygons to a specific area. The polygons are described in Well Known Text (WKT). **Note** WKT is a text markup language for representing vector geometry objects on a map, spatial reference systems of spatial objects, and transformations between spatial reference systems.

Item	Description

Parameter	• area: a geometric concept.
	• PolygonName: the name of the polygon that you want to manage.
	• polygonWkt: the description of the polygon that is written in WKT. The description includes longitudes and latitudes. The following polygon types can be described in WKT:
	  ◦ POINT: the WKT that describes a point. Example: `'POINT (120.086631 30.138141)'`, which indicates that the point is located at longitude 120.086631 and latitude 30.138141.
	  ◦ LINESTRING: the WKT that describes a linestring, which consists of two points. Example: `'LINESTRING (30 10, 40 40)'`.
	  ◦ POLYGON: the WKT that describes a polygon, which consists of multiple points. Example: `'POLYGON ((31 20, 29 20, 29 21, 31 31))'`.
	Note
	◦ Valid values for a longitude are -180 to 180 and valid values for a latitude are -90 to 90.
	◦ MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRY, and COLLECTION collection types are not supported.

Output	• If the operation is successful, the number of polygons that were added and updated is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'`
	Sample output:
	`(integer) 1`

GIS.GET

Item	Description
Syntax	`GIS.GET area polygonName`
Time complexity	O(1)
Command description	Retrieves the WTK of a specific polygon within a specific area.
Parameter	• area: a geometric concept. • PolygonName: the name of the polygon that you want to manage.

Item	Description
Output	• If the operation is successful, the WKT of the polygon is returned. • If the area or polygon does not exist, nil is returned. • Otherwise, an error message is returned.

Example	The `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` command is run in advance.
	Sample command:
	`GIS.GET hangzhou campus`
	Sample output:
	`'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'`

GIS.GETALL

Item	Description
Syntax	`GIS.GETALL area [WITHOUTWKT]`
Time complexity	O(n)
Command description	Retrieves the names and WKTs of all polygons within a specific area. If you specify the WITHOUTWKT parameter, only the names of the polygons are returned.
Parameter	• area: a geometric concept. • WITHOUTWKT: specifies whether to return the WKTs of polygons. If this parameter is specified, the WKTs of the polygons are not returned.
Output	• If the operation is successful, the names and WKTs of the polygons are returned. If the WITHOUTWKT parameter is specified, only the names of the polygons are returned. • If the area does not exist, nil is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` command is run in advance.
	Sample command:
	`GIS.GETALL hangzhou`
	Sample output:
	`1) "campus"`
	`2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"`

GIS.CONTAINS

Item	Description
Syntax	`GIS.CONTAINS area polygonWkt [WITHOUTWKT]`
Time complexity	• Optimal time complexity: $O(\log_M n)$ • Least desirable time complexity: log(n)
Command description	Checks whether a specified point, linestring, or polygon is located in polygons within a specific area. If yes, this command returns the number and WKTs of polygons that contain the point, linestring, or polygon in the area.

Parameter	• area: a geometric concept.
	• polygonWkt: the description of the polygon that is written in WKT. The description includes longitudes and latitudes. The following polygon types can be described in WKT:
	  ◦ POINT: the WKT that describes a point.
	  ◦ LINESTRING: the WKT that describes a linestring.
	  ◦ POLYGON: the WKT that describes a polygon.
	• WITHOUTWKT: specifies whether to return the WKTs of polygons. If this parameter is specified, the WKTs of the polygons are not returned.

Output	• If the operation is successful, the number and WKTs of the polygons are returned. • If the area does not exist, the "empty list or set" message is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` command is run in advance.
	Sample command:
	`GIS.CONTAINS hangzhou 'POINT (30 11)'`
	Sample output:
	`1) "1"`
	`2) 1) "campus"`
	`2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"`

GIS.WITHIN

Item	Description
Syntax	`GIS.WITHIN area polygonWkt [WITHOUTWKT]`
Time complexity	• Optimal time complexity: $O(\log_M n)$ • Least desirable time complexity: log(n)
Command description	Checks whether a specific area is located within a specified point, linestring, or polygon. If yes, this command returns the number and WKTs of polygons that are located within the point, linestring, or polygon.

Parameter	• area: a geometric concept.
	• polygonWkt: the description of the polygon that is written in WKT. The description includes longitudes and latitudes. The following polygon types can be described in WKT:
	  ○ POINT: the WKT that describes a point.
	  ○ LINESTRING: the WKT that describes a linestring.
	  ○ POLYGON: the WKT that describes a polygon.
	Note MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, GEOMETRY, and COLLECTION are not supported.
	• WITHOUTWKT: specifies whether to return the WKTs of polygons. If this parameter is specified, the WKTs of the polygons are not returned.

Output	• If the operation is successful, the number and WKTs of the polygons are returned. • If the area does not exist, the "empty list or set" message is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` command is run in advance.
	Sample command:
	`GIS.WITHIN hangzhou 'POLYGON ((30 5, 50 50, 20 50, 5 20, 30 5))'`
	Sample output:
	`1) "1"`
	`2) 1) "campus"`
	`2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"`

GIS.INTERSECTS

Item	Description
Syntax	`GIS.INTERSECTS area polygonWkt`
Time complexity	• Optimal time complexity: $O(\log_M n)$ • Least desirable time complexity: log(n)
Command description	Checks whether a specific point, linestring, or polygon intersects with polygons within a specific area. If yes, this command returns the number and WKTs of polygons within the area that intersect with the point, linestring, or polygon.

Parameter	• area: a geometric concept.
	• polygonWkt: the description of the polygon that is written in WKT. The description includes longitudes and latitudes. The following polygon types can be described in WKT:
	  ◦ POINT: the WKT that describes a point.
	  ◦ LINESTRING: the WKT that describes a linestring.
	  ◦ POLYGON: the WKT that describes a polygon.
	• WITHOUTWKT: specifies whether to return the WKTs of polygons. If this parameter is specified, the WKTs of the polygons are not returned.

Output	• If the operation is successful, the number and WKTs of the polygons are returned. • If the area does not exist, the "empty list or set" message is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` command is run in advance.
	Sample command:
	`GIS.INTERSECTS hangzhou 'LINESTRING (30 10, 40 40)'`
	Sample output:
	`1) "1"`
	`2) 1) "campus"`
	`2) "POLYGON((30 10,40 40,20 40,10 20,30 10))"`

GIS.SEARCH

Item	Description

Syntax	`GIS.SEARCH area [RADIUS longitude latitude distance MKMFTMI]`
	`[MEMBER field distance MKMFTMI]`
	`[GEOM geom]`
	`[COUNT count]`
	`[ASC|DESC]`
	`[WITHDIST]`
	`[WITHOUTWKT]`

Time complexity	• Optimal time complexity: $O(\log_M n)$ • Least desirable time complexity: log(n)
Command description	Queries the points within a specific area that are located within a specific radius of a specific longitude and latitude position.

Item	Description
Parameter	• area: a geometric concept. • RADIUS: specifies the longitude, latitude, radius, and radius unit. Valid values for the unit: meters (M), kilometers (KM), feet (FT), and miles (MI). Example: `RADIUS 15 37 200 KM`. • MEMBER: specifies the point in the area that is used as the base point and the radius for the point. Valid values for the radius unit: meters (M), kilometers (KM), feet (FT), and miles (MI). Example: `MEMBER Agrigento 100 KM`. In this example, Agrigento indicates the name of the polygon, 100 indicates the radius, and KM indicates the radius unit. This order must be used for the parameter syntax. • GEOM: the polygon written in WKT that indicates the search range. Example: `GEOM 'POLYGON((10 30,20 30,20 40,10 40))'`. • COUNT: the maximum number of entries returned. Example: `COUNT 3`. • ASC|DESC: specifies the order in which the returned entries are ranked based on distance. ASC indicates that the returned entries are ranked from low to high based on their distance to the center. DESC: indicates that the returned entries are ranked from high to low based on their distance to the center. • WITHDIST: specifies whether to return the distance between a specific point and the point specified by the MEMBER parameter. • WITHOUTWKT: specifies whether to return the WKTs of points. If this parameter is specified, the WKTs of the points are not returned. **Note** You can specify only one of the RADIUS, MEMBER, and GEOM parameters.
Output	• If the operation is successful, the number and WKTs of the points are returned. • If the area does not exist, the "empty list or set" message is returned. • Otherwise, an error message is returned.
Example	The `GIS.ADD Sicily "Palermo" "POINT (13.361389 38.115556)" "Catania" "POINT(15.087269 37.502669)"` command is run in advance. Sample command: `GIS.SEARCH Sicily RADIUS 15 37 200 km WITHDIST ASC` Sample output: 1) (integer) 2 2) 1) "Catania" 2) "POINT(15.087269 37.502669)" 3) "56.4413" 4) "Palermo" 5) "POINT(13.361389 38.115556)" 6) "190.4424"

GIS.DEL

Item	Description
Syntax	`GIS.DEL area polygonName`
Time complexity	O(log n)
Command description	Deletes a specific polygon from a specific area.
Parameter	• area: a geometric concept. • PolygonName: the name of the polygon that you want to manage.
Output	• If the operation is successful, OK is returned. • If the area or polygon does not exist, nil is returned. • Otherwise, an error message is returned.

Example	The `GIS.ADD hangzhou campus 'POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))'` command is run in advance.
	Sample command:
	`GIS.DEL hangzhou campus`
	Sample output:
	`OK`

3.6.6. TairBloom

TairBloom is a space-efficient probabilistic data structure that consumes minimal memory to check whether an element exists. It features dynamic scalability while maintaining a stable false positive rate during scaling.

Introduction to TairBloom

TairBloom is a space-efficient probabilistic data structure that consumes minimal memory to check whether an element exists. It features dynamic scalability while maintaining a stable false positive rate during scaling.

You can use bitmaps on Redis data structures, such as hashes, sets, and strings, to implement similar features of TairBloom. However, these data structures may consume a large amount of memory or fail to maintain a stable false positive rate during dynamic scaling. For this reason, TairBloom is ideal for checking for large amounts of data when a specific false positive rate is allowed. You can use the built-in Bloom filters of TairBloom without further encapsulation or the need to create extra Bloom filters on your on-premises devices.

Main features

- Minimum consumption of memory.
- Dynamic scaling.
- Stable custom false positive rate during scaling.

Typical scenarios

TairBloom can be used for recommendation and crawler systems in industries such as live-streaming, music, and e-commerce.

- Recommendation system: TairBloom records the IDs of articles that may have been recommended, queries these articles, determines duplicate articles, and then recommends new articles in which users might be interested.
- Crawler system: TairBloom filters out the URLs that have been crawled to improve productivity.

Best practices

☐ Build recommendation systems based on TairBloom

TairBloom records the IDs of articles that may have been recommended, queries these articles, determines duplicate articles, and then recommends new articles in which users might be interested. Pseudocode:

```
void recommendedSystem(userid) {
    while (true) {
        // Obtain a random article ID or a desired article ID.
        docid = getDocByRandom()
        if (bf.exists(userid, docid)) {
            // If the article may have been recommended to a user, the next article is
recommended.
            continue;
        } else {
            // If the article has not been recommended to the user, the article is
recommended.
            sendRecommendMsg(docid);
            // If the article may have been recommended, the article ID is recorded in a
Bloom filter.
            bf.add(userid, docid);
            break;
        }
    }
}
```

☐ Optimize crawler systems based on TairBloom

TairBloom filters out the URLs that may have been crawled to improve productivity. Pseudocode:

```
bool crawlerSystem( ) {
    while (true) {
        // Obtain a URL that you want to crawl.
        url = getURLFromQueue()
        if (bf.exists(url_bloom, url)) {
            // If the URL may have been crawled, the URL is skipped.
            continue;
        } else {
            // Crawl this URL.
            doDownload(url)
            // Add this URL to a Bloom filter.
            bf.add(url_bloom, url);
        }
    }
}
```

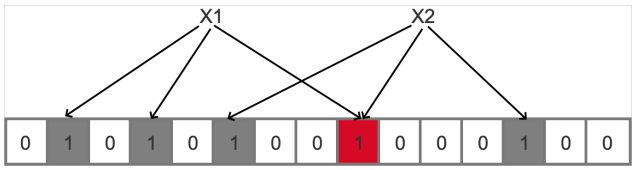
How it works

TairBloom is an implementation of scalable Bloom filters (SBFs). It features dynamic scalability while maintaining a stable false positive rate. SBFs are optimized Bloom filters. The following section describes the basic principles of Bloom filters and SBFs.

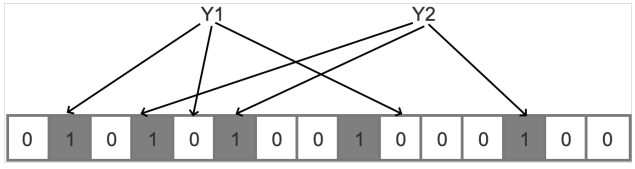
- Bloom Filter

  A Bloom filter is a space-efficient probabilistic data structure conceived by Burton Howard Bloom in 1970. It is used to test whether an element is a member of a set.

  A new Bloom filter is a bit array of m bits, all of which are set to 0. The Bloom filter also includes a set of k different hash functions to generate a uniform random distribution. k is a constant that is less than m. When you add elements to the Bloom filter, the k hash functions map these elements to k bits in the bit array and set the k bits to a value of 1. In this case, a bit can be shared by multiple pieces of data. The following figure shows how to insert X1 and X2 into a Bloom filter that includes 3 hash functions.

  When you query an element in a Bloom filter, you can use k hash functions to obtain k bits. If all of k bits have a value of 1, the element exists in the Bloom filter. Otherwise, the element does not exist. The following figure shows how to query Y1 and Y2 in the Bloom filter.

  The preceding figure demonstrates that although Y2 was never inserted into the Bloom filter, Y2 is determined to be existent in the Bloom filter. This scenario shows that Bloom filters have a false positive rate. The preceding analysis shows that Bloom filters have the following features:

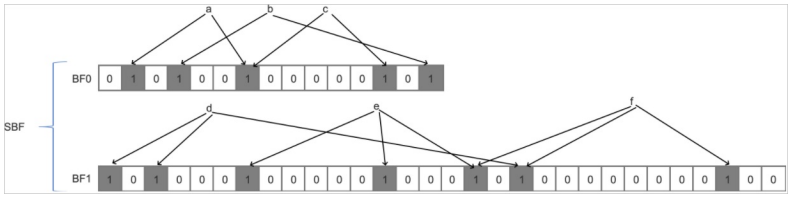
  - A bit can be shared by multiple pieces of data.
  - Bloom filters have a false positive rate and the rate gets higher as the number of elements in a Bloom filter increases. However, Bloom filters do not have a false negative rate. If an element exists in a Bloom filter, the element is always determined to be existent.

- An element can be added to a Bloom filter but cannot be removed from the Bloom filter. The reason is that bits can be shared. If you remove an element from the Bloom filter, other elements in the Bloom filter are affected.

- Scalable Bloom Filter

  When an increasing number of elements are added to a Bloom filter, the false positive rate becomes higher. If you want to maintain a stable false positive rate, you must accordingly increase the size of the Bloom filter. However, the size cannot be increased due to structural limits. SBFs emerged in response to this issue. SBFs are a new type of Bloom filter that combines multiple Bloom filters into one.

  The following figure shows the basic model of an SBF. This SBF has two layers: BF0 and BF1. At first, the SBF contains only the BF0 layer. If you insert elements a, b, and c into this SBF and the BF0 layer is not large enough to maintain a specified false positive rate, the BF1 layer is created to increase the SBF size. Then, elements d, e, and f are inserted into the BF1 layer. A new BF2 layer is created if the BF1 layer also cannot maintain the specified positive rate.

  SBFs insert data only into the last layer and query data from the last layer to the BF0 layer. For more information, see Scalable Bloom Filters.

Prerequisites

The instance is a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair). For more information about performance-enhanced instances, see Performance-enhanced instances.

> **Note** The latest minor version provides more features and higher stability. We recommend that you update your instance to the latest minor version. For more information, see Update the minor version. If your instance is a cluster or read/write splitting instance, we recommend that you update the proxy nodes in the instance to the latest minor version. This ensures that all commands can be run as expected. For more information about cluster and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Precautions

- The TairBloom data that you want to manage is stored on a performance-enhanced instance.
- The initial capacity and false positive rate that meet your requirements must be calculated in advance. To create a TairBloom key that has a capacity far more than 100 elements, run the `BF.RESERVE` command instead of the `BF.ADD` command.

  The following section demonstrates the difference between the `BF.ADD` command and the `BF.RESERVE` command:

  - `BF.ADD` (also known as `BF.MADD`): If a TairBloom key does not exist when this command is run, the key is automatically created. The default capacity of the key is 100, and the default false positive rate of the key is 0.01. If you require a key capacity that is far more than 100, you can only scale up the key to support more elements.

    When a key is scaled up, more layers of Bloom filters are added to it. Every time a Bloom filter layer is added, the key capacity increases. In this case, if you perform a query on the key, multiple layers of Bloom filters need to be traversed and the performance of the key significantly deteriorates.

- `BF.RESERVE` (also known as `BF.INSERT`): An initial capacity is specified when this command is run. This command specifies an initial capacity at the first layer of a key. If a key contains a smaller number of Bloom filter layers, the query speed of the key is faster.

  > **Note** For example, assume that you want to insert 10,000,000 elements into a key and allows an false positive rate of 0.01. The created key is 176 MB in size if you use the `BF.ADD` command, or 16 MB in size if you use the `BF.RESERVE` command.

  Although a key can be scaled up, we recommend that you consider key scale-ups only as a safety measure and do not frequently use them. If the actual capacity exceeds the provisioned capacity of a key, you can scale up the key to make sure that write operations can be performed on the key and prevent live accidents.

  The following table lists a variety of key sizes supported by the `BF.RESERVE` command and their matched initial capacities and false positive rates.

Capacity (number of elements)	false positive:0.01	false positive:0.001	false positive:0.0001
100,000	0.12 MB	0.25 MB	0.25 MB
1,000,000	2 MB	2 MB	4 MB
10,000,000	16 MB	32 MB	32 MB
100,000,000	128 MB	256 MB	256 MB
1,000,000,000	2 GB	2 GB	4 GB

- The size of a key cannot be decreased because elements can only be added to a key, not removed. To prevent capacity issues such as an out of memory (OOM) error, we recommend that you implement the following solutions:
  - Split business data. You can split business data to prevent large keys that affect query performance. If large keys exist, most requests are made to the instance that contains these keys. This may cause hotkeys or even skewed requests.

    You can distribute the split data to multiple keys. If your business data is stored in an ApsaraDB for Redis cluster instance, you can distribute the split data to multiple nodes in the instance to ensure that large keys and hotkeys do not occur.
  - Regularly rebuild keys. If possible, you can run the `DEL` command to delete data from a key, and then insert data from backend databases into the key to manage the key size.

    You can also create multiple keys to interchangeably use them. This way, the size of a single key is kept appropriate. The benefit of this solution is that you need to create keys only once. However, multiple keys must be created and some memory may be wasted.

Supported commands

TairBloom commands

Command	Syntax	Description

Command	Syntax	Description
BF.RESERVE	`BF.RESERVE key error_rate capacity`	Creates an empty TairBloom key that has a specified capacity and false positive rate. The capacity parameter specifies the capacity of the key and the error_rate parameter specifies the false positive rate of the key.
BF.ADD	`BF.ADD key item`	Adds an element to a TairBloom key.
BF.MADD	`BF.MADD key item [item ...]`	Adds multiple elements to a TairBloom key.
BF.EXISTS	`BF.EXISTS key item`	Checks whether an element exists in a TairBloom key.
BF.MEXISTS	`BF.MEXISTS key item [item ...]`	Checks whether multiple elements exist in a TairBloom key.
BF.INSERT	`BF.INSERT key [CAPACITY cap] [ERROR error] [NOCREATE] ITEMS item [item ...]`	Adds multiple elements to a TairBloom key. If the key does not exist, you can specify whether to create the key. You can also specify the capacity and false positive rate of the key.
DEL	`DEL key [key ...]`	Deletes one or more TairBloom keys. **Note** The elements that are already added to a key cannot be deleted. You can run the DEL command to delete all data from a key.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

BF.RESERVE

Item	Description
Syntax	`BF.RESERVE key error_rate capacity`

Item	Description
Time complexity	O(1)
Command description	Creates an empty TairBloom key that has a specified capacity and false positive rate. The capacity parameter specifies the capacity of the key and the error_rate parameter specifies the false positive rate of the key.
Parameter	• Key: the name of the key that you want to manage by running this command. • error_rate: the expected false positive rate. The value of this parameter must be between 0 and 1. A lower value indicates higher accuracy, memory usage, and CPU utilization of the key. • capacity: the initial capacity of the key. This parameter specifies the maximum number of elements that can be added to the key. When the number of elements added to the key exceeds the capacity value, a Bloom filter layer is added for the key. This process may deteriorate the query performance of the key. Every time a Bloom filter layer is added, the key capacity increases. In this case, if you perform a query on the key, multiple layers of Bloom filters may need to be traversed. If your workloads require high performance, we recommend that you add elements to a key based on your business requirements to prevent automatic scaling.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`BF.RESERVE BFKEY 0.01 100`
	Sample output:
	`OK`

BF.ADD

Item	Description
Syntax	`BF.ADD key item`
Time complexity	O(log N), where N specifies the number of Bloom filter layers.
Command description	Adds an element to a TairBloom key. **Note** If the key does not exist, the key is automatically created. The default capacity of the key is 100, and the default false positive rate of the key is 0.01.

Item	Description
Parameter	• Key: the name of the key that you want to manage by running this command. • item: the element that you want to add to the key.
Output	• If the element does not exist, the element is added and a value of 1 is returned. • If the element may already exist, the element is not added or updated and a value of 0 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`BF.ADD BFKEY item1`
	Sample output:
	`(integer) 1`

BF.MADD

Item	Description
Syntax	`BF.MADD key item [item ...]`
Time complexity	O(log N), where N specifies the number of Bloom filter layers.
Command description	Adds multiple elements to a TairBloom key. **Note** If the key does not exist, the key is automatically created. The default capacity of the key is 100, and the default false positive rate of the key is 0.01.
Parameter	• Key: the name of the key that you want to manage by running this command. • item: the element that you want to add to the key. Multiple elements can be specified.
Output	• If the element does not exist, the element is added and a value of 1 is returned. • If the element may already exist, the element is not added or updated and a value of 0 is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`BF.MADD BFKEY item1 item2 item3`
	Sample output:
	`(integer) 1`
	`(integer) 1`
	`(integer) 1`

BF.EXISTS

Item	Description
Syntax	`BF.EXISTS key item`
Time complexity	O(log N), where N specifies the number of Bloom filter layers.
Command description	Checks whether an element exists in a TairBloom key.
Parameter	• Key: the name of the key that you want to manage by running this command. • item: the element that you want to query.
Output	• If the element does not exist, a value of 0 is returned. • If the element may exist, a value of 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`BF.EXISTS BFKEY item1`
	Sample output:
	`(integer) 1`

BF.MEXISTS

Item	Description
Syntax	`BF.MEXISTS key item [item ...]`
Time complexity	O(log N), where N specifies the number of Bloom filter layers.

Item	Description
Command description	Checks whether multiple elements exist in a TairBloom key.
Parameter	• Key: the name of the key that you want to manage by running this command. • item: the element that you want to query. Multiple elements can be specified.
Output	• If the element does not exist, a value of 0 is returned. • If the element may exist, a value of 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`BF.MEXISTS BFKEY item1 item5`
	Sample output:
	`(integer) 1`
	`(integer) 0`

BF.INSERT

Item	Description
Syntax	`BF.INSERT key [CAPACITY cap] [ERROR error] [NOCREATE] ITEMS item [item ...]`
Time complexity	O(log N), where N specifies the number of Bloom filter layers.
Command description	Adds multiple elements to a TairBloom key. If the key does not exist, you can specify whether to create the key. You can also specify the capacity and false positive rate of the key.

Item	Description
Parameter	• Key: the name of the key that you want to manage by running this command. • capacity: the initial capacity of the key. This parameter specifies the maximum number of elements that can be added to the key. If the key exists, this parameter is ignored. If the number of elements that are added to a key exceeds the capacity value, another Bloom filter layer is automatically added for the key. During the scaling, the number of elements in the key increases and the performance decreases. To query a specific element after a layer is added to the key, multiple layers may be traversed. The capacity of each new layer is twice that of the previous layer. If your workloads require high performance, we recommend that you add elements to a key based on your business requirements to prevent automatic scaling. • error_rate: the expected false positive rate. The value of this parameter must be between 0 and 1. A lower value indicates higher accuracy, memory usage, and CPU utilization of the key. • NOCREATE: specifies that the key is not automatically created if the key does not exist. This parameter cannot be specified together with the capacity or error_rate parameter. • item: the element that you want to add. Multiple elements can be specified.
Output	• If the element does not exist, the element is added and a value of 1 is returned. • If the element may already exist, the element is not added or updated and a value of 0 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`BF.INSERT bfkey1 CAPACITY 10000 ERROR 0.001 ITEMS item1 item2 item3`
	Sample output:
	`(integer) 1`
	`(integer) 1`
	`(integer) 1`

3.6.7. TairDoc

TairDoc is a document data structure that is fully compatible with RedisJSON. You can use TairDoc to perform create, read, update, and delete (CRUD) operations.

Main features

- All JSON standards supported.
- Full compatibility with RedisJSON.
- JSONPath and JSON Pointer syntax supported.
- Binary tree data storage that simplifies the retrieval of child elements.
- Conversion from the JSON format to the XML or YAML format.

Prerequisites

The instance is a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair). For more information about performance-enhanced instances, see Performance-enhanced instances.

> **Note** The latest minor version provides more features and higher stability. We recommend that you update your instance to the latest minor version. For more information, see Update the minor version. If your instance is a cluster or read/write splitting instance, we recommend that you update the proxy nodes in the instance to the latest minor version. This ensures that all commands can be run as expected. For more information about cluster and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Precautions

The TairDoc data to be managed is stored on the performance-enhanced instance.

Supported commands

TairDoc commands

Command	Syntax	Description
JSON.SET	`JSON.SET key path json [NX | XX]`	Creates a TairDoc key and stores a JSON element in a path of the key. If the key and path already exist, this command updates the element in the path of the key.
JSON.GET	`JSON.GET key path [FORMAT XML | YAML] [ROOTNAME root] [ARRNAME arr]`	Retrieves the JSON element from the path of the TairDoc key.
JSON.DEL	`JSON.DEL key path`	Deletes the JSON element from the path in the TairDoc key. If the path is not specified, the key is deleted. If the key or path does not exist, this command is ignored.
JSON.TYPE	`JSON.TYPE key path`	Retrieves the type of the JSON element from the path in the TairDoc key. Element types include `boolean`, `string`, `number`, `array`, `object`, `raw`, `reference`, `const`, and `null`.
JSON.NUMINCRBY	`JSON.NUMINCRBY key path value`	Increases the JSON element value in the path of the TairDoc key. The element and the value that you want to add to the element both must be of the same data type of integer or double.
JSON.STRAPPEND	`JSON.STRAPPEND key path json-string`	Adds the json-string value to the JSON element in the path of the TairDoc key. The json-string value and JSON element both must be of the string type.

Command	Syntax	Description
JSON.STRLEN	`JSON.STRLEN key path`	Retrieves the string length of the JSON element from the path of the TairDoc key. The JSON element must be of the string type.
JSON.ARRAPPEND	`JSON.ARRAPPEND key path json [json ...]`	Adds one or more JSON elements to the end of an array in the path of the TairDoc key.
JSON.ARRPOP	`JSON.ARRPOP key path [index]`	Removes and returns the element that matches the index in an array of the TairDoc key path.
JSON.ARRINSERT	`JSON.ARRINSERT key path [index] json [json ...]`	Adds one or more JSON elements to an array in the path of the TairDoc key.
JSON.ARRLEN	`JSON.ARRLEN key path`	Retrieves the length of an array in the path of the TairDoc key.
JSON.ARRTRIM	`JSON.ARRTRIM key path start stop`	Trims the array in the path of the TairDoc key. This command retains the elements in the array that are within the start value and the stop value.
DEL	`DEL <key> [key ...]`	Deletes one or more TairDoc keys.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

JSON.SET

Item	Description
Syntax	`JSON.SET key path json [NX | XX]`
Time complexity	O(N)

Item	Description
Command description	Creates a TairDoc key and stores a JSON element in a path of the key. If the key and path already exist, this command updates the element in the path of the key. **Note** If the key does not exist, the path is specified as root (or `.`).
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key. • json: the element that you want to add or update. • NX: specifies that the element is written only if the path does not exist. • XX: specifies that the element is written only if the path exists.
Output	• If the operation is successful, OK is returned. • If the XX parameter is specified but the path does not exist, nil is returned. • If the NX parameter is specified and the path already exists, nil is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`JSON.SET doc . '{"foo": "bar", "baz" : 42}'`
	Sample output:
	`OK`

JSON.GET

Item	Description
Syntax	`JSON.GET key path [FORMAT XML | YAML] [ROOTNAME root] [ARRNAME arr]`
Time complexity	O(N)
Command description	Retrieves the JSON element from the path of the TairDoc key.

Item	Description
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key. • FORMAT: the JSON format of the element. Valid values: XML and YAML. • ROOTNAME : the tag of the root element in XML syntax. • ARRNAME: the tag of the array element in XML syntax. **Note** The ROOTNAME and ARRNAME parameters are valid only if the FORMAT parameter is set to XML.
Output	• If the operation is successful, the JSON element is returned. • Otherwise, an error message is returned.

Example	The `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` command is run in advance.
	Sample command:
	`JSON.GET doc . FORMAT XML ROOTNAME ROOT ARRNAME ARR`
	Sample output:
	`"<?xml version=\"1.0\" encoding=\"UTF-8\"?><ROOT><foo>bar</foo><baz>42</baz></ROOT>"`

JSON.DEL

Item	Description
Syntax	`JSON.DEL` *key path*
Time complexity	O(N)
Command description	Deletes the JSON element from the path in the TairDoc key. If the path is not specified, the key is deleted. If the key or path does not exist, this command is ignored.
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key.
Output	• If the operation is successful, a value of 1 is returned. • If the operation fails, a value of 0 is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` command is run in advance.
	Sample command:
	`JSON.DEL doc .foo`
	Sample output:
	`(integer) 1`

JSON.TYPE

Item	Description
Syntax	`JSON.TYPE key path`
Time complexity	O(N)
Command description	Retrieves the type of the JSON element from the path in the TairDoc key. Element types include `boolean`, `string`, `number`, `array`, `object`, `raw`, `reference`, `const`, and `null`.
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key.
Output	• If the operation is successful, the element type is returned. • If the operation fails, a value of 0 is returned. • If the key or path does not exist, nil is returned. • Otherwise, an error message is returned.

Example	The `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` command is run in advance.
	Sample command:
	`JSON.TYPE doc .foo`
	Sample output:
	`string`

JSON.NUMINCRBY

Item	Description
Syntax	`JSON.NUMINCRBY key path value`
Time complexity	O(N)
Command description	Increases the JSON element value in the path of the TairDoc key. The element and the value that you want to add to the element both must be of the same data type of integer or double.
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key. • value: the value that you want to add to the element.
Output	• If the operation is successful, the increased element in the path is returned. • If the key or path does not exist, the "error" message is returned. • Otherwise, an error message is returned.

Example	The `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` command is run in advance.
	Sample command:
	`JSON.NUMINCRBY doc .baz 10`
	Sample output:
	`"52"`

JSON.STRAPPEND

Item	Description
Syntax	`JSON.STRAPPEND key path json-string`
Time complexity	O(N)
Command description	Adds the json-string value to the JSON element in the path of the TairDoc key. The json-string value and JSON element both must be of the string type.
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key. • json-string: the sting that you want to add to the JSON element in the path.

Item	Description
Output	• If the operation is successful, the increased string length of the JSON element in the path is returned. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Example	The `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` command is run in advance.
	Sample command:
	`JSON.STRAPPEND doc .foo rrrrr`
	Sample output:
	`(integer) 8`

JSON.STRLEN

Item	Description
Syntax	`JSON.STRLEN` *key path*
Time complexity	O(N)
Command description	Retrieves the string length of the JSON element from the path of the TairDoc key. The JSON element must be of the string type.
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key.
Output	• If the operation is successful, the string length of the JSON element in the path is returned. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Example	The `JSON.SET doc . '{"foo": "bar", "baz" : 42}'` command is run in advance.
	Sample command:
	`JSON.STRLEN doc .foo`
	Sample output:
	`(integer) 3`

JSON.ARRAPPEND

Item	Description
Syntax	`JSON.ARRAPPEND` *key path json* [*json* ...]
Time complexity	O(M×N), where M specifies the number of JSON elements that you want to add and N specifies the number of elements in the array.
Command description	Adds one or more JSON elements to the end of an array in the path of the TairDoc key.
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key. • json: the element that you want to add to the array.
Output	• If the operation is successful, the number of elements in the array is returned. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Example	The `JSON.SET doc . '{"id": [1,2,3]}'` command is run in advance.
	Sample command:
	`JSON.ARRAPPEND doc .id null false true`
	Sample output:
	`(integer) 6`

JSON.ARRPOP

Item	Description
Syntax	`JSON.ARRPOP` *key path* [*index*]
Time complexity	O(M×N), where M specifies the number of child elements in the key and N specifies the number of elements in the array.
Command description	Removes and returns the element that matches the index in an array of the TairDoc key path.

Item	Description
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key. • index: the index in the array. The beginning index is 0. Negative values are used to designate elements starting at the end of the array. If this parameter is not specified, the index that matches the last element in the array is used.
Output	• If the operation is successful, the element is removed and returned. • If the array is empty, the "ERR array index outflow" message is returned. • Otherwise, an error message is returned.

Example	The `JSON.SET doc . '{"id": [1,2,3]}'` command is run in advance.
	Sample command:
	`JSON.ARRPOP doc .id 0`
	Sample output:
	`"1"`

JSON.ARRINSERT

Item	Description
Syntax	`JSON.ARRINSERT` *key path* [*index*] *json* [*json* ...]
Time complexity	O(M×N), where M specifies the number of JSON elements that you want to add and N specifies the number of elements in the array.
Command description	Adds one or more JSON elements to an array in the path of the TairDoc key.
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key. • index: the index in the array. The beginning index is 0. Negative values are used to designate elements starting from the end of the array. If this parameter is not specified, the index that matches the last element in the array is used. • json: the element that you want to add to the array.
Output	• If the operation is successful, the number of elements in the array is returned. • If the array is empty, the "ERR array index outflow" message is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `JSON.SET doc . '{"id": [1,2,3]}'` command is run in advance.
	Sample command:
	`JSON.ARRINSERT doc .id 0 10 15`
	Sample output:
	`(integer) 5`

JSON.ARRLEN

Item	Description
Syntax	`JSON.ARRLEN key path`
Time complexity	O(N)
Command description	Retrieves the length of an array in the path of the TairDoc key.
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key.
Output	• If the operation is successful, the length of the array is returned. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Example	The `JSON.SET doc . '{"id": [1,2,3]}'` command is run in advance.
	Sample command:
	`JSON.ARRLEN doc .id`
	Sample output:
	`(integer) 3`

JSON.ARRTRIM

Item	Description
Syntax	`JSON.ARRTRIM key path start stop`

Item	Description
Time complexity	O(N)
Command description	Trims the array in the path of the TairDoc key. This command retains the elements in the array that are within the start value and the stop value.
Parameter	• Key: the key that you want to manage by running this command. • path: the path in the key. • start: the start of the range in which elements are retained after a trim. The parameter value is an index value that is equal to or greater than 0. The element at the start position is retained. • stop: the end of the range in which elements are retained after a trim. The parameter value is an index value that is equal to or greater than 0. The element at the end position is retained.
Output	• If the operation is successful, the length of the trimmed array is returned. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Example	The `JSON.SET doc . '{"id": [1,2,3,4,5,6]}'` command is run in advance.
	Sample command:
	`JSON.ARRTRIM doc .id 3 4`
	Sample output:
	`(integer) 2`

JSON Pointer and JSONPath

TairDoc supports the JSON Pointer syntax and also supports specific JSONPath syntax. The following table compares the examples of the two syntax types.

The `JSON.SET doc . '{"foo": "bar", "baz" : [1,2,3]}'` command is run in advance.

JSONPointer	JSONPath

Sample command:
`JSON.GET doc .baz[0]`
Sample output:
`"1"`
	Sample command:
	`JSON.GET doc /baz/0`
	Sample output:
	`"1"`

The following table shows how TairDoc supports the JSONPath and JSON Pointer syntax.

Item	JSONPath	JSONPointer
A root element	.	""
An individual element	.a.b.c	/a/b/c
An array	.a[2]	/a/2
A combination of multiple elements	.a["b.c"]	/a/b.c
	.a['b.c']	/a/b.c

3.6.8. TairTS

TairTS is a time series data structure that is developed on top of Redis modules. This data structure provides low-latency and high-concurrency in-memory read and write access, supports fast filtering and aggregate queries, and has both storage and computing power. TairTS simplifies the processing of time series data and significantly improves performance.

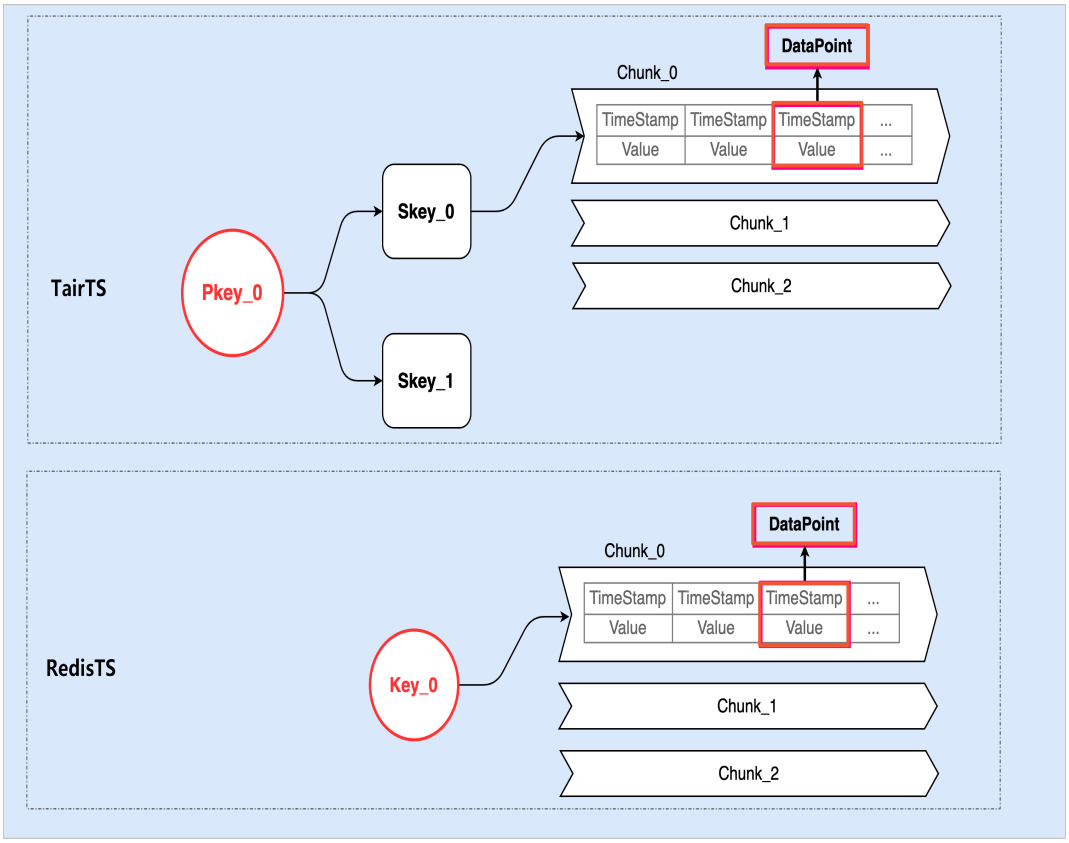
Overview

TairTS provides more features than RedisTimeSeries. For more information about RedisTimeSeries, visit RedisTimeSeries. The following features are supported by TairTS:

- Multi-timeline aggregate queries by using pkeys developed by Alibaba Cloud. Pkeys have an extra hash layer.

  For example, you can create multiple skeys that are named after metric names and device IDs in the foo pkey. Examples: temperature:1, pressure:1, and distance:1. Then, you can run the EXTS.S.MRANGE command provided by TairTS to retrieve custom monitoring data such as skeys that have a device ID of 1. If you want to implement a similar feature by using RedisTimeSeries, you must insert a large number of aggregate queries into business logic code.

  Comparison between TairTS and RedisTimeSeries



The following section describes the structure of TairTS data:

- Pkey: A pkey is a piece of TairTS data that indicates a set of timelines. A pkey consists of multiple skeys.
- Skey: a timeline. An skey consists of multiple chunks that have a fixed capacity. You can attach one or more different labels to each skey. Then, skeys can be filtered by label.
- Chunk: a data chunk. A chunk can store multiple data points.
  - A chunk can have a custom capacity of up to 256 data points.
  - Chunks are the smallest expiration units. A chunk is deleted after all data points in the chunk are expired.
- DataPoint: A data point is a piece of time series data that includes a timestamp and a DOUBLE-typed value.

- Aggregate queries in scenarios such as downsampling, attribute filtering, batch query, and the use of multiple numerical functions. This feature integrates batch query and aggregation in a single command to reduce network interaction and provide responses within milliseconds.
- Update and accumulation of historical time series data.
- Configuration of time to live (TTL) for skeys. Each skey can be specified with a TTL and can automatically roll based on time windows.
- Efficient Gorilla compression algorithm and specific storage to drastically reduce costs.

Typical scenarios

- Storage and computing of monitoring data
- Per-second monitoring for application performance management (APM)
- Data analysis and processing for IoT

- Risk control in throttling scenarios
- Cache of hot news
- Use of time window functions

Best practices

Implement fine-grained monitoring by using TairTS

Prerequisites

The instance is a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair) that runs the minor version of 1.7.20 or later. For more information about performance-enhanced instances, see Performance-enhanced instances.

> **Note** The latest minor version provides more features and higher stability. We recommend that you update your instance to the latest minor version. For more information, see Update the minor version. If your instance is a cluster or read/write splitting instance, we recommend that you update the proxy nodes in your instance to the latest minor version. This ensures that all commands can be run as expected. For more information about cluster and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Precautions

- The TairTS data that you want to manage is stored on the performance-enhanced instance.
- TairTS supports real-time, high-concurrency writes and queries but provides limited storage capacity. As such, we recommend that you specify TTL for TairTS data to ensure that expired data is deleted in a timely manner.
- To reduce storage costs, we recommend that you specify the `CHUNK_SIZE` parameter in the following ways:
  - If an skey contains more than 5,000 data points on average, set `CHUNK_SIZE` to 256. This is the default value.
  - If an skey contains less than 5,000 data points on average, set `CHUNK_SIZE` to a value that is calculated by using the following formula: Chunk size = Average number of data points of an skey/20. If an skey contains 1,000 data points on average, set `CHUNK_SIZE` to 50.

Supported commands

TairTS commands

Type	Command	Syntax	Description
	EXTS.P.CREATE	`EXTS.P.CREATE` *Pkey*	Creates a TairTS pkey. If a pkey with the same name already exists, the pkey cannot be created.

Type	Command	Syntax	Description
Basic write operation	EXTS.S.CREATE	`EXTS.S.CREATE Pkey Skey [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 label2 val2 ...]`	Creates an skey in a specified pkey. If the pkey does not exist, it is automatically created. If an skey with the same name already exists, the skey cannot be created. **Note** You can specify parameters for the skey, such as the expiration time and whether to enable compression.
	EXTS.S.ALTER	`EXTS.S.ALTER Pkey Skey [DATA_ET time]`	Modifies the metadata of a specified skey. Currently, only the DATA_ET time value can be modified.
	EXTS.S.ADD	`EXTS.S.ADD Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	Inserts a data point into an skey. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
	EXTS.S.MADD	`EXTS.S.MADD Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	Inserts a data point into multiple skeys of a pkey. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
	EXTS.S.INCRBY	`EXTS.S.INCRBY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	Inserts a data point into an skey. If the data point that you want to insert has a positive value, the inserted value is added to the value of the last data point that exists in the skey. If the data point has a negative value, the inserted value is subtracted from the value of the last data point that exists in the skey. If the pkey or skey does not exist, the pkey or skey is automatically created. The default initial value is 0. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.

Type	Command	Syntax	Description
	EXTS.S.MINCRBY	`EXTS.S.MINCRBY Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	Inserts a data point into multiple skeys of a pkey. If the data point that you want to insert has a positive value, the inserted value is added to the value of the last data point that exists in the skey. If the data point has a negative value, the inserted value is subtracted from the value of the last data point that exists in the skey. If the pkey or skey does not exist, the pkey or skey is automatically created. The default initial value is 0. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
	EXTS.S.DEL	`EXTS.S.DEL Pkey Skey`	Deletes an skey from a pkey and all data points in the skey.
Basic read operation	EXTS.S.GET	`EXTS.S.GET Pkey Skey`	Queries the value of the last data point in a specified skey.
	EXTS.S.INFO	`EXTS.S.INFO Pkey Skey`	Queries the metadata of a specified skey. The metadata includes the number of data points, the timestamp and value of the last data point, and the label information of the skey.
	EXTS.S.QUERYINDEX	`EXTS.S.QUERYINDEX Pkey filter1 [filter2 ...]`	Queries the skeys that meet specified filter conditions.
Aggregate operation	EXTS.S.RANGE	`EXTS.S.RANGE Pkey Skey fromTs toTs [MAXCOUNT count] [AGGREGATION aggregationType timeBucket]`	Queries the number of data points that exist in an skey during a specified time range. The time range is a closed interval.
	EXTS.S.MRANGE	`EXTS.S.MRANGE Pkey fromTs toTs [MAXCOUNT count] [AGGREGATION aggregationType timeBucket] [WITHLABELS] FILTER filter1 [filter2 ...]`	Queries the number of data points that meet specified filter conditions in multiple skeys during a specified time range. The time range is a closed interval.

Type	Command	Syntax	Description
	EXTS.P.RANGE	`EXTS.P.RANGE Pkey fromTs toTs pkeyAggregationType pkeyTimeBucket [MAXCOUNT count] [AGGREGATION aggregationType timeBucket] [WITHLABELS] FILTER filter1 [filter2 ...]`	Aggregates data points in a pkey that meet specified filter conditions. If you specify one or more skeys for aggregation, the skeys are first aggregated in the same manner as when the EXTS.S.MRANGE command is used. Then, pkeys are aggregated based on the skey aggregation results.
Concurrent write operation	EXTS.S.RAW_MODIFY	`EXTS.S.RAW_MODIFY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	Modifies the value of a data point in a specified skey. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
	EXTS.S.RAW_MMODIFY	`EXTS.S.RAW_MMODIFY Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	Modifies the values of multiple data points in a specified skey at a time. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
	EXTS.S.RAW_INCRBY	`EXTS.S.RAW_INCRBY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	Modifies the value of a data point in a specified skey. You can specify an increment or a decrement by which to modify the value. If the pkey or skey does not exist, the pkey or skey is automatically created. The default initial value is 0. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
	EXTS.S.RAW_MINCRBY	`EXTS.S.RAW_MINCRBY Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`	Modifies the values of multiple data points in a specified skey at a time. You can specify an increment or a decrement by which to modify the values. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.

Type	Command	Syntax	Description
General-purpose operation	DEL	`DEL key [key ...]`	Deletes one or more TairTS keys.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

EXTS.P.CREATE

Item	Description
Syntax	`EXTS.P.CREATE Pkey`
Time complexity	O(1)
Command description	Creates a TairTS pkey. If a pkey with the same name already exists, the pkey cannot be created.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.P.CREATE foo`
	Sample output:
	`OK`

EXTS.S.CREATE

Item	Description

Item	Description
Syntax	`EXTS.S.CREATE Pkey Skey [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 label2 val2 ...]`
Time complexity	O(1)
Command description	Creates an skey in a specified pkey. If the pkey does not exist, it is automatically created. If an skey with the same name already exists, the skey cannot be created. **Note** You can specify parameters for the skey, such as the expiration time and whether to enable compression.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • Skey: the name of the skey. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire. • CHUNK_SIZE: the number of data points that can be stored in a chunk. Default value: 256. Valid values: 1 to 256. • UNCOMPRESSED: specifies that compression is disabled for the skey. By default, this parameter is left empty, which indicates that compression is enabled for the skey. • LABELS: the label of the skey. One or more labels and their values can be specified. Example: `LABELS sensor_id 1`.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.S.CREATE foo temperature DATA_ET 10000000 LABELS sensor_id 1`
	Sample output:
	`OK`

EXTS.S.ALTER

Item	Description
Syntax	`EXTS.S.ALTER Pkey Skey [DATA_ET time]`
Time complexity	O(1)

Item	Description
Command description	Modifies the metadata of a specified skey. Currently, only the DATA_ET time value can be modified.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • Skey: the name of the skey. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.S.ALTER foo temperature DATA_ET 100000`
	Sample output:
	`OK`

EXTS.S.ADD

Item	Description
Syntax	`EXTS.S.ADD Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
Time complexity	O(1)
Command description	Inserts a data point into an skey. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.

Item	Description
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • Skey: the name of the skey. • ts: the UNIX timestamp of the data point. Unit: milliseconds. A value of `*` can be used to specify the timestamp of the current time. • value: the value of the data point. The value must be a double-precision floating-point number. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire. • CHUNK_SIZE: the number of data points that can be stored in a chunk. Default value: 256. Valid values: 1 to 256. • UNCOMPRESSED: specifies that compression is disabled for the skey. By default, this parameter is left empty, which indicates that compression is enabled for the skey. • LABELS: the label of the skey. One or more labels and their values can be specified. Example: `LABELS sensor_id 1`.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.S.ADD foo temperature 1644310456023 30.5 DATA_ET 1000000 LABELS sensor_id 1`
	Sample output:
	`OK`

EXTS.S.MADD

Item	Description
Syntax	`EXTS.S.MADD` *`Pkey keynumber Skey ts value`* `[`*`Skey ts value ...`*`] [DATA_ET` *`time`*`] [CHUNK_SIZE` *`size`*`] [UNCOMPRESSED] [LABELS` *`label1 val1 ...`*`]`
Time complexity	O(n), where n indicates the number of data points.
Command description	Inserts a data point into multiple skeys of a pkey. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.

Item	Description
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • keynumber: the number of data points. • Skey: the name of the skey. • ts: the UNIX timestamp of the data point. Unit: milliseconds. A value of `*` can be used to specify the timestamp of the current time. • value: the value of the data point. The value must be a double-precision floating-point number. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire. • CHUNK_SIZE: the number of data points that can be stored in a chunk. Default value: 256. Valid values: 1 to 256. • UNCOMPRESSED: specifies that compression is disabled for the skey. By default, this parameter is left empty, which indicates that compression is enabled for the skey. • LABELS: the label of the skey. One or more labels and their values can be specified. Example: `LABELS sensor_id 1`.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.S.MADD foo 3 temperature * 30.2 pressure * 2.05 distance * 0.5`
	Sample output:
	`1) OK`
	`2) OK`
	`3) OK`

EXTS.S.INCRBY

Item	Description
Syntax	`EXTS.S.INCRBY` *`Pkey Skey ts value`* `[DATA_ET` *`time`*`] [CHUNK_SIZE` *`size`*`] [UNCOMPRESSED] [LABELS` *`label1 val1 ...`*`]`
Time complexity	O(1)

Item	Description
Command description	Inserts a data point into an skey. If the data point that you want to insert has a positive value, the inserted value is added to the value of the last data point that exists in the skey. If the data point has a negative value, the inserted value is subtracted from the value of the last data point that exists in the skey. If the pkey or skey does not exist, the pkey or skey is automatically created. The default initial value is 0. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • Skey: the name of the skey. • ts: the UNIX timestamp of the data point. Unit: milliseconds. A value of `*` can be used to specify the timestamp of the current time. • value: the value of the data point that you want to insert into the skey. The value can be a negative one. In this case, the inserted value is subtracted from the value of the last data point that exists in the skey. The value must be a double-precision floating-point number. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire. • CHUNK_SIZE: the number of data points that can be stored in a chunk. Default value: 256. Valid values: 1 to 256. • UNCOMPRESSED: specifies that compression is disabled for the skey. By default, this parameter is left empty, which indicates that compression is enabled for the skey. • LABELS: the label of the skey. One or more labels and their values can be specified. Example: `LABELS sensor_id 1`.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	The `EXTS.S.ADD foo temperature 1644310456023 30.0` command is run in advance.
	Sample command:
	`EXTS.S.INCRBY foo temperature 1644372093031 2`
	Sample output:
	`OK`
	If the `EXTS.S.GET foo temperature` command is run at this time, the following output is returned:
	`1) (integer) 1644372093031`
	`2) "32"`

EXTS.S.MINCRBY

Item	Description
Syntax	`EXTS.S.MINCRBY Pkey keynumber Skey ts value [Skey ts value ...] [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
Time complexity	O(n), where n indicates the number of data points.
Command description	Inserts a data point into multiple skeys of a pkey. If the data point that you want to insert has a positive value, the inserted value is added to the value of the last data point that exists in the skey. If the data point has a negative value, the inserted value is subtracted from the value of the last data point that exists in the skey. If the pkey or skey does not exist, the pkey or skey is automatically created. The default initial value is 0. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • keynumber: the number of data points. • Skey: the name of the skey. • ts: the UNIX timestamp of the data point. Unit: milliseconds. A value of `*` can be used to specify the timestamp of the current time. • value: the value of the data point that you want to insert into the skey. The value can be a negative one. In this case, the inserted value is subtracted from the value of the last data point that exists in the skey. The value must be a double-precision floating-point number. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire. • CHUNK_SIZE: the number of data points that can be stored in a chunk. Default value: 256. Valid values: 1 to 256. • UNCOMPRESSED: specifies that compression is disabled for the skey. By default, this parameter is left empty, which indicates that compression is enabled for the skey. • LABELS: the label of the skey. One or more labels and their values can be specified. Example: `LABELS sensor_id 1`.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.S.MINCRBY foo 3 temperature * 0.2 pressure * -0.1 distance * 0.0`
	Sample output:
	`1) OK`
	`2) OK`
	`3) OK`

EXTS.S.DEL

Item	Description
Syntax	`EXTS.S.DEL Pkey Skey`
Time complexity	O(1)
Command description	Deletes an skey from a pkey and all data points in the skey.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • Skey: the name of the skey.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.S.DEL foo temperature`
	Sample output:
	`OK`

EXTS.S.GET

Item	Description
Syntax	`EXTS.S.GET Pkey Skey`
Time complexity	O(1)
Command description	Queries the value of the last data point in a specified skey.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • Skey: the name of the skey.
Output	• If the operation is successful, the data point value is returned. • If the pkey or skey does not exist, nil is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXTS.S.GET foo temperature`
	Sample output:
	`1) (integer) 1644372730150`
	`2) "32.2"`

EXTS.S.INFO

Item	Description
Syntax	`EXTS.S.INFO Pkey Skey`
Time complexity	O(1)
Command description	Queries the metadata of a specified skey. The metadata includes the number of data points, the timestamp and value of the last data point, and the label information of the skey.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • Skey: the name of the skey.
Output	• If the operation is successful, the metadata of the skey is returned. • If the pkey or skey does not exist, nil is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXTS.S.INFO foo temperature`
	Sample output:
	`1) totalDataPoints // Number of data points.`
	`2) (integer) 1`
	`3) maxDataPoints // Maximum number of data points that can be stored in the skey. The default value is 0, which indicates no limit.`
	`4) (integer) 0`
	`5) maxDataPointsPerChunk // Number of data points that can be stored in a chunk.`
	`6) (integer) 32`
	`7) dataPointsExpireTime // Relative expiration time of the skey. Unit: milliseconds. A value of 0 indicates that the skey does not expire.`
	`8) (integer) 0`
	`9) lastTimestamp // Timestamp of the last data point.`
	`10) (integer) 1644389400996`
	`11) chunkCount // Number of chunks in the skey.`
	`12) (integer) 1`
	`13) lastValue // Value of the last data point.`
	`14) (integer) 28`
	`15) labels // Label information of the skey.`
	`16) 1) 1) "sensor_id"`
	`2) "1"`

EXTS.S.QUERYINDEX

Item	Description
Syntax	`EXTS.S.QUERYINDEX` *Pkey filter1* `[`*filter2* `...]`
Time complexity	O(n), where n indicates the maximum number of sets involved in filter conditions.
Command description	Queries the skeys that meet specified filter conditions.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • filter: the filter condition. You can filter skeys by label. For more information, see the "Index filtering syntax" section of this topic. **Note** When you specify a filter condition, you must use one of the EQ, CONTAINS, and LIST_MATCH logics. Otherwise, filter queries cannot be performed.

Item	Description
Output	• If the operation is successful, the skeys that meet specified filter conditions are returned. • If the pkey or skey does not exist, nil is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.S.QUERYINDEX foo sensor_id=1`
	Sample output:
	`1) "temperature"`

EXTS.S.RANGE

Item	Description
Syntax	`EXTS.S.RANGE Pkey Skey fromTs toTs [MAXCOUNT count] [AGGREGATION aggregationType timeBucket]`
Time complexity	O(n), where n indicates the number of chunks to which the data points belong.
Command description	Queries the number of data points that exist in an skey during a specified time range. The time range is a closed interval.

Item	Description

Parameter	• Pkey: the name of the pkey that you want to manage by running this command.
	• Skey: the name of the skey.
	• fromTs: the beginning of the time range to query. This value must be a UNIX timestamp. Unit: milliseconds.
	• toTs: the end of the time range to query. This value must be a UNIX timestamp. Unit: milliseconds. A value of `*` can be used to specify the timestamp of the current time. If this value is equal to the fromTs value, a point in time is used for the query.
	• MAXCOUNT: the number of data points to return. This parameter is left empty by default. The upper limit is 1,000,000 for ApsaraDB for Redis Enhanced Edition (Tair) instances.
	• AGGREGATION:
	  ◦ aggregationType: the aggregation type, such as `MAX`, `AVG`, and `SUM`. For more information, see the "Aggregation syntax" section of this topic.
	  ◦ timeBucket: the time bucket used to aggregate data. Unit: milliseconds. The lower limit is 1,000.
	  In ApsaraDB for Redis Enhanced Edition (Tair), data that exists in multiple time buckets is aggregated together and returned in a single result. The points in time returned are the beginning of each time bucket.
	For example, `AGGREGATION AVG 5000` returns the average number of data points that exist in each time bucket that lasts for 5,000 milliseconds within the specified time range.

Output	• If the operation is successful, the number of data points that exist in an skey during a specified time range is returned. If the AGGREGATION parameter is specified in the command, the aggregation result is returned. **Note** An additional token value is returned. A value of 0 indicates that all data points that meet specified filter conditions are displayed. A value of 1 indicates that specific data points that meet specified filtered conditions are not displayed. You can retrieve the last data point from the result based on the token value. To implement batch aggregation, you can use the timestamp of the last data point as the beginning of the next time range to query. • If the pkey or skey does not exist, nil is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXTS.S.RANGE foo test 1644459031662 * AGGREGATION AVG 10000 MAXCOUNT 2`
	`// Specify to return the average number of data points that exist in each time bucket that lasts for 10,000 milliseconds within the specified time range and to return two data entries.`
	Sample output:
	`1) 1) 1) (integer) 1644459730000`
	`2) "20.6"`
	`2) 1) (integer) 1644459790000`
	`2) "21.2"`
	`2) (integer) 1 // A value of 0 indicates that all data points that meet specified filter conditions are displayed. A value of 1 indicates that specific data points that meet specified filtered conditions are not displayed.`

EXTS.S.MRANGE

Item	Description
Syntax	`EXTS.S.MRANGE` *`Pkey fromTs toTs`* `[MAXCOUNT` *`count`*`] [AGGREGATION` *`aggregationType timeBucket`*`] [WITHLABELS] FILTER` *`filter1`* `[`*`filter2 ...`*`]`
Time complexity	O(n), where n indicates the number of chunks to which the data points belong.
Command description	Queries the number of data points that meet specified filter conditions in multiple skeys during a specified time range. The time range is a closed interval.

Item	Description

Parameter	• Pkey: the name of the pkey that you want to manage by running this command.
	• fromTs: the beginning of the time range to query. This value must be a UNIX timestamp. Unit: milliseconds.
	• toTs: the end of the time range to query. This value must be a UNIX timestamp. Unit: milliseconds. A value of `*` can be used to specify the timestamp of the current time. If this value is equal to the fromTs value, a point in time is used for the query.
	• MAXCOUNT: the number of data points to return in an skey. This parameter is left empty by default. The upper limit is 1,000,000 for ApsaraDB for Redis Enhanced Edition (Tair) instances.
	• AGGREGATION:
	  ◦ aggregationType: the aggregation type, such as `MAX`, `AVG`, and `SUM`. For more information, see the "Aggregation syntax" section of this topic.
	  ◦ timeBucket: the time bucket used to aggregate data. Unit: milliseconds. The lower limit is 1,000.
	  In ApsaraDB for Redis Enhanced Edition (Tair), data that exists in multiple time buckets is aggregated together and returned in a single result. The points in time returned are the beginning of each time bucket.
	• WITHLABELS: specifies whether to include label information in returned results. This parameter is left empty by default, which indicates that label information is not displayed.
	• filter: the filter condition. You can filter skeys by label. For more information, see the "Index filtering syntax" section of this topic.
	Note When you specify a filter condition, you must use one of the EQ, CONTAINS, and LIST_MATCH logics. Otherwise, filter queries cannot be performed.

Output	• If the operation is successful, the skeys that meet specified filter conditions are returned. • If the pkey or skey does not exist, nil is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXTS.S.MRANGE foo 1644451031662 * AGGREGATION MAX 10000 WITHLABELS FILTER sensor_id=1`
	Sample output:
	`1) 1) "temperature"`
	`2) 1) 1) "sensor_id"`
	`2) "1"`
	`3) 1) 1) (integer) 1644481000000`
	`2) "30"`
	`4) (integer) 0`
	`2) 1) "test"`
	`2) 1) 1) "sensor_id"`
	`2) "1"`
	`3) 1) 1) (integer) 1644459730000`
	`2) "20"`
	`2) 1) (integer) 1644459790000`
	`2) "20"`
	`3) 1) (integer) 1644460620000`
	`2) "29"`
	`4) (integer) 0`

EXTS.P.RANGE

Item	Description
Syntax	`EXTS.P.RANGE Pkey fromTs toTs pkeyAggregationType pkeyTimeBucket [MAXCOUNT count] [AGGREGATION aggregationType timeBucket] [WITHLABELS] FILTER filter1 [filter2 ...]`
Time complexity	O(n), where n indicates the number of chunks to which the data points belong.
Command description	Aggregates data points in a pkey that meet specified filter conditions. If you specify one or more skeys for aggregation, the skeys are first aggregated in the same manner as when the EXTS.S.MRANGE command is used. Then, pkeys are aggregated based on the skey aggregation results.

Item	Description
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • fromTs: the beginning of the time range to query. This value must be a UNIX timestamp. Unit: milliseconds. • toTs: the end of the time range to query. This value must be a UNIX timestamp. Unit: milliseconds. A value of `*` can be used to specify the timestamp of the current time. If this value is equal to the fromTs value, a point in time is used for the query. • pkeyAggregationType: the aggregation type of the pkey. For more information, see the "Aggregation syntax" section of this topic. • pkeyTimeBucket: the time bucket used to aggregate data in the pkey. Unit: milliseconds. The lower limit is 1,000. In ApsaraDB for Redis Enhanced Edition (Tair), data that exists in multiple time buckets is aggregated together and returned in a single result. The points in time returned are the beginning of each time bucket. • MAXCOUNT: the number of data points to return in an skey. This parameter is left empty by default. The upper limit is 1,000,000 for ApsaraDB for Redis Enhanced Edition (Tair) instances. • AGGREGATION: ◦ aggregationType: the aggregation type of the skey. For more information, see the "Aggregation syntax" section of this topic. ◦ timeBucket: the time bucket used to aggregate data in the skey. Unit: milliseconds. The lower limit is 1,000. In ApsaraDB for Redis Enhanced Edition (Tair), data that exists in multiple time buckets is aggregated together and returned in a single result. The points in time returned are the beginning of each time bucket. • WITHLABELS: specifies whether to include label information in returned results. This parameter is left empty by default, which indicates that label information is not displayed. • filter: the filter condition. You can filter skeys by label. For more information, see the "Index filtering syntax" section of this topic. **Note** When you specify a filter condition, you must use one of the EQ, CONTAINS, and LIST_MATCH logics. Otherwise, filter queries cannot be performed.
Output	• If the operation is successful, the aggregation results are returned. • If the pkey or skey does not exist, nil is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXTS.P.RANGE foo 1644451031662 * SUM 500000 AGGREGATION SUM 10000 FILTER sensor_id=1`
	Sample output:
	`1) 1) 1) (integer) 1644459500000`
	`2) "40"`
	`2) 1) (integer) 1644460500000`
	`2) "29"`
	`3) 1) (integer) 1644481000000`
	`2) "30"`
	`2) (integer) 0`

EXTS.S.RAW_MODIFY

Item	Description
Syntax	`EXTS.S.RAW_MODIFY` *`Pkey Skey ts value`* `[DATA_ET` *`time`*`] [CHUNK_SIZE` *`size`*`] [UNCOMPRESSED] [LABELS` *`label1 val1 ...`*`]`
Time complexity	O(1)
Command description	Modifies the value of a data point in a specified skey. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • Skey: the name of the skey. • ts: the UNIX timestamp of the data point whose value you want to update. Unit: milliseconds. • value: the data point value that you want to update. The value must be a double-precision floating point number. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire. • CHUNK_SIZE: the number of data points that can be stored in a chunk. Default value: 256. Valid values: 1 to 256. • UNCOMPRESSED: specifies that compression is disabled for the skey. By default, this parameter is left empty, which indicates that compression is enabled for the skey. • LABELS: the label of the skey. One or more labels and their values can be specified. Example: `LABELS sensor_id 1`.

Item	Description
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.S.RAW_MODIFY foo temperature 1644310456023 31.5`
	Sample output:
	`OK`

EXTS.S.RAW_MMODIFY

Item	Description
Syntax	`EXTS.S.RAW_MMODIFY` *`Pkey keynumber Skey ts value`* `[`*`Skey ts value ...`*`] [DATA_ET` *`time`*`] [CHUNK_SIZE` *`size`*`] [UNCOMPRESSED] [LABELS` *`label1 val1 ...`*`]`
Time complexity	O(n), where n indicates the number of data points.
Command description	Modifies the values of multiple data points in a specified skey at a time. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • keynumber: the number of data points. • Skey: the name of the skey. • ts: the UNIX timestamp of the data point whose value you want to update. Unit: milliseconds. • value: the data point value that you want to update. The value must be a double-precision floating point number. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire. • CHUNK_SIZE: the number of data points that can be stored in a chunk. Default value: 256. Valid values: 1 to 256. • UNCOMPRESSED: specifies that compression is disabled for the skey. By default, this parameter is left empty, which indicates that compression is enabled for the skey. • LABELS: the label of the skey. One or more labels and their values can be specified. Example: `LABELS sensor_id 1`.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`EXTS.S.RAW_MMODIFY foo 3 temperature 1644565954814 30.2 pressure 1644565954814 2.05 distance 1644565954814 0.5`
	Sample output:
	`1) OK`
	`2) OK`
	`3) OK`

EXTS.S.RAW_INCRBY

Item	Description
Syntax	`EXTS.S.RAW_INCRBY Pkey Skey ts value [DATA_ET time] [CHUNK_SIZE size] [UNCOMPRESSED] [LABELS label1 val1 ...]`
Time complexity	O(1)
Command description	Modifies the value of a data point in a specified skey. You can specify an increment or a decrement by which to modify the value. If the pkey or skey does not exist, the pkey or skey is automatically created. The default initial value is 0. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • Skey: the name of the skey. • ts: the UNIX timestamp of the data point whose value you want to update. Unit: milliseconds. • value: the value that you want to add to the data point value. The value can be a negative one. In this case, the value is subtracted from the data point value. The value must be a double-precision floating-point number. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire. • CHUNK_SIZE: the number of data points that can be stored in a chunk. Default value: 256. Valid values: 1 to 256. • UNCOMPRESSED: specifies that compression is disabled for the skey. By default, this parameter is left empty, which indicates that compression is enabled for the skey. • LABELS: the label of the skey. One or more labels and their values can be specified. Example: `LABELS sensor_id 1`.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `EXTS.S.ADD foo temperature 1644310456 30.0` command is run in advance.
	Sample command:
	`EXTS.S.RAW_INCRBY foo temperature 1644310456 3.3`
	Sample output:
	`OK`
	If the `EXTS.S.GET foo temperature` command is run at this time, the following output is returned:
	`1) (integer) 1644310456`
	`2) "33.3"`

EXTS.S.RAW_MINCRBY

Item	Description
Syntax	`EXTS.S.RAW_MINCRBY` *Pkey keynumber Skey ts value* `[`*Skey ts value ...*`] [DATA_ET` *time*`] [CHUNK_SIZE` *size*`] [UNCOMPRESSED] [LABELS` *label1 val1 ...*`]`
Time complexity	O(n), where n indicates the number of data points.
Command description	Modifies the values of multiple data points in a specified skey at a time. You can specify an increment or a decrement by which to modify the values. If the pkey or skey does not exist, the pkey or skey is automatically created. The parameters for the skey, such as the expiration time and whether to enable compression, take effect only if the skey is automatically created.

Item	Description
Parameter	• Pkey: the name of the pkey that you want to manage by running this command. • keynumber: the number of data points. • Skey: the name of the skey. • ts: the UNIX timestamp of the data point whose value you want to update. Unit: milliseconds. • value: the value that you want to add to the data point value. The value can be a negative one. In this case, the value is subtracted from the data point value. The value must be a double-precision floating-point number. • DATA_ET time: the relative expiration time of the data point. Unit: milliseconds. By default, this parameter is left empty, which indicates that the data point does not expire. • CHUNK_SIZE: the number of data points that can be stored in a chunk. Default value: 256. Valid values: 1 to 256. • UNCOMPRESSED: specifies that compression is disabled for the skey. By default, this parameter is left empty, which indicates that compression is enabled for the skey. • LABELS: the label of the skey. One or more labels and their values can be specified. Example: `LABELS sensor_id 1` .
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`EXTS.S.RAW_MMODIFY foo 3 temperature 1644565954814 30.2 pressure 1644565954814 2.05 distance 1644565954814 0.5`
	Sample output:
	`1) OK`
	`2) OK`
	`3) OK`

Index filtering syntax

Skeys can be filtered by label. The following syntax is used for filter conditions.

Note When you specify a filter condition, you can use one or more of the following commands and you must use one of the EQ, CONTAINS, and LIST_MATCH logics.

Filter command	Description	Logic
`L = V`	The label L equals V.	EQ (equals)
`L !=`	The label L is not NULL, which indicates that the skey includes the label L.	CONTAINS

Filter command	Description	Logic
`L = (v1,v2,...)`	The label L is v1, v2, or another value.	LIST_TMATCH
`L != V`	The label L does not equal V.	NOEQ (equals)
`L =`	The label L is NULL, which indicates that the skey does not include the label L.	NOCONTAINS
`L != (v1,v2,...)`	The label L is not v1, v2, or another value.	LIST_NOTMATCH

Aggregation syntax

In aggregation operations, data that exists in multiple time buckets is aggregated. The following aggregation types are supported:

- `MAX`: returns the maximum value.
- `MIN`: returns the minimum value.
- `AVG`: returns the average value.
- `SUM`: returns the sum of all values.
- `FIRST`: returns the first value.
- `LAST`: returns the last value.
- `RANGE`: returns the range from the minimum value to the maximum value.
- `COUNT`: returns the number of values.
- `STD.P`: returns the population variance.
- `STD.S`: returns the sample variance.
- `VAR.P`: returns the population standard deviation.
- `VAR.S`: returns the sample standard deviation.

3.6.9. TairCpc

TairCpc is a data structure developed based on the compressed probability counting (CPC) sketch. It allows you to perform high-performance computing on sampled data while using a small amount of memory.

Context

In real-time decision-making scenarios that involve big data, the real-time computing system processes incoming business logs, the online storage system stores the processing results, and then the real-time rule-based or decision-making system makes decisions. Sample scenarios:

- Prevention and control of credit card fraud: In this scenario, your systems must determine whether a credit card is used in a safe environment and stop suspicious transactions at the earliest opportunity.
- Prevention and control of ticket scalping: In this scenario, your systems must identify and stop activities in real time that use virtual devices and fake IP addresses to undermine platform interests.

In this case, you can use TairCpc to deduplicate real-time data by dimension and structurally store the data in Tair databases. These operations allow fast access to data and the integration of storage and computing. TairCpc also supports multiple aggregation operations to allow data to be aggregated within nanoseconds and provide real-time risk control.

Overview

CPC is a high-performance data deduplication algorithm that counts different values as data streams. It allows you to combine data blocks and deduplicate the blocks to obtain a total number. For more information about CPC, see Back to the Future: an Even More Nearly Optimal Cardinality Estimation Algorithm. CPC achieves the same accuracy as HyperLogLog (HLL) with about 40% less memory.

Developed based on open source CPC, TairCpc reduces the error rate to 0.008%, as opposed to 0.67% of open source CPC and 1.95% of HLL.

Main features

- Low memory usage, incremental reads and writes, and minimal I/O
- High-performance and ultra-high-accuracy deduplication
- Reduced stable error rate

Typical scenarios

- Security systems for banks
- Flash sales
- Prevention and control of ticket scalping

Prerequisites

- The instance is a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair) that runs the minor version of 1.7.20 or later. For more information about performance-enhanced instances, see Performance-enhanced instances.
- The instance is a persistent memory-optimized instance of the ApsaraDB for Redis Enhanced Edition (Tair) that runs the minor version of 1.2.3.3 or later. For more information about persistent memory-optimized instances, see Persistent memory-optimized instances.

> **Note** The latest minor version provides more features and higher stability. We recommend that you update your instance to the latest minor version. For more information, see Update the minor version. If your instance is a cluster instance or read/write splitting instance, we recommend that you update the proxy nodes in the instance to the latest minor version. This ensures that all commands can be run as expected. For more information about cluster instances and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Precautions

The TairCpc data to be managed is stored on the instance.

Supported commands

TairCpc commands

Command	Syntax	Description
CPC.UPDATE	`CPC.UPDATE key item [EX|EXAT|PX|PXAT time]`	Adds an item to the specified TairCpc key. If the key does not exist, the key is created. If the item already exists in the key, the item is not added.

Command	Syntax	Description
CPC.ESTIMATE	`CPC.ESTIMATE key`	Retrieves the cardinality estimate of the specified TairCpc key after deduplication. The return value is of the DOUBLE type, but you can ignore the decimals and round it to the nearest integer.
CPC.UPDATE2EST	`CPC.UPDATE2EST key item [EX|EXAT|PX|PXAT time]`	Adds an item to the specified TairCpc key and returns the updated cardinality estimate of the key. If the key does not exist, the key is created.
CPC.UPDATE2JUD	`CPC.UPDATE2JUD key item [EX|EXAT|PX|PXAT time]`	Adds an item to the specified TairCpc key and returns the updated cardinality estimate of the key and the difference between the original and updated estimates. If the item is added and no duplication exists, a difference of 1 is returned. If the item already exists, a difference of 0 is returned. If the key does not exist, the key is created.

Command	Syntax	Description
CPC.ARRAY.UPDATE	`CPC.ARRAY.UPDATE key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`	Adds an item to the time window to which the specified timestamp belongs in the specified TairCpc key. If the key does not exist, the key is created. `SIZE` indicates the number of time windows, and `WIN` indicates the length of each time window. The length is measured in milliseconds. The key is updated as data streams are added to the key. During this process, data that is generated during a time-window range is saved. The time-window range is calculated by using the following formula: Time-window range = `SIZE` × `WIN`. Data that is generated outside of this time-window range is overwritten and deleted. `SIZE` and `WIN` are valid only at the point in time when the key is created. **Note** For example, if you want to calculate the amount of data in the key that was generated per minute during the last 10 minutes, you can set `SIZE` to 10 (10 time windows) and `WIN` to 60000 (1 minute for each time window). In this case, if you write the data that was generated during the 11th minute to the key, the data that was generated during the first minute is overwritten and deleted.
CPC.ARRAY.ESTIMATE	`CPC.ARRAY.ESTIMATE key timestamp`	Retrieves the cardinality estimate of a specified TairCpc key within the time window to which the specified timestamp belongs.
CPC.ARRAY.ESTIMATE.RANGE	`CPC.ARRAY.ESTIMATE.RANGE key start_time end_time`	Retrieves the cardinality estimates of the time windows that reside in the specified time range in the specified TairCpc key. The time range is a closed interval.
CPC.ARRAY.ESTIMATE.RANGE.MERGE	`CPC.ARRAY.ESTIMATE.RANGE.MERGE key timestamp range`	Retrieves the cardinality estimate of the specified TairCpc key after merging and deduplication at a specific point in time and during a subsequent time range.

Command	Syntax	Description
CPC.ARRAY.UPDATE2EST	`CPC.ARRAY.UPDATE2EST key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`	Adds an item to the time window to which the specified timestamp belongs in the specified TairCpc key and returns the updated cardinality estimate of the key within the time window. If the key does not exist, the key is created. This command acts like CPC.ARRAY.UPDATE.
CPC.ARRAY.UPDATE2JUD	`CPC.ARRAY.UPDATE2JUD key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`	Adds an item to the time window to which the specified timestamp belongs in the specified TairCpc key and returns the updated cardinality estimate of the key within the time window and the difference between the original and updated estimates. If the item is added and no duplication exists, a difference of 1 is returned. If the item already exists, a difference of 0 is returned. If the key does not exist, the key is created. This command acts like CPC.ARRAY.UPDATE.
DEL	`DEL key [key ...]`	Deletes one or more TairCpc keys.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

CPC.UPDATE

Item	Description
Syntax	`CPC.UPDATE key item [EX|EXAT|PX|PXAT time]`
Time complexity	O(1)
Command description	Adds an item to the specified TairCpc key. If the key does not exist, the key is created. If the item already exists in the key, the item is not added.

Item	Description
Parameter	• key: the name of the TairCpc key that you want to manage by running this command. • item: the item that you want to add. • EX: the relative expiration time of the key. Unit: seconds. If this parameter is not specified, the key does not expire. • EXAT: the absolute expiration time of the key. Unit: seconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire. • PX: the relative expiration time of the key. Unit: milliseconds. If this parameter is not specified, the key does not expire. • PXAT: the absolute expiration time of the key. Unit: milliseconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`CPC.UPDATE foo f1 EX 3600`
	Sample output:
	`OK`

CPC.ESTIMATE

Item	Description
Syntax	`CPC.ESTIMATE key`
Time complexity	O(1)
Command description	Retrieves the cardinality estimate of the specified TairCpc key after deduplication. The return value is of the DOUBLE type, but you can ignore the decimals and round it to the nearest integer.
Parameter	• key: the name of the TairCpc key.
Output	• If the operation is successful, the DOUBLE-typed estimate is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`CPC.ESTIMATE foo`
	Sample output:
	`"19.000027716212127"`

CPC.UPDATE2EST

Item	Description
Syntax	`CPC.UPDATE2EST key item [EX|EXAT|PX|PXAT time]`
Time complexity	O(1)
Command description	Adds an item to the specified TairCpc key and returns the updated cardinality estimate of the key. If the key does not exist, the key is created.
Parameter	• key: the name of the TairCpc key that you want to manage by running this command. • item: the item that you want to add. • EX: the relative expiration time of the key. Unit: seconds. If this parameter is not specified, the key does not expire. • EXAT: the absolute expiration time of the key. Unit: seconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire. • PX: the relative expiration time of the key. Unit: milliseconds. If this parameter is not specified, the key does not expire. • PXAT: the absolute expiration time of the key. Unit: milliseconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire.
Output	• If the operation is successful, the DOUBLE-typed updated estimate is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`CPC.UPDATE2EST foo f3`
	Sample output:
	`"3.0000004768373003"`

CPC.UPDATE2JUD

Item	Description
Syntax	`CPC.UPDATE2JUD key item [EX|EXAT|PX|PXAT time]`
Time complexity	O(1)
Command description	Adds an item to the specified TairCpc key and returns the updated cardinality estimate of the key and the difference between the original and updated estimates. If the item is added and no duplication exists, a difference of 1 is returned. If the item already exists, a difference of 0 is returned. If the key does not exist, the key is created.
Parameter	• key: the name of the TairCpc key that you want to manage by running this command. • item: the item that you want to add. • EX: the relative expiration time of the key. Unit: seconds. If this parameter is not specified, the key does not expire. • EXAT: the absolute expiration time of the key. Unit: seconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire. • PX: the relative expiration time of the key. Unit: milliseconds. If this parameter is not specified, the key does not expire. • PXAT: the absolute expiration time of the key. Unit: milliseconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire.
Output	• If the operation is successful, the DOUBLE-typed updated estimate and difference are returned. • Otherwise, an error message is returned.

Example	Sample command:
	`CPC.UPDATE2JUD foo f20`
	Sample output:
	`1) "20.000027716212127"  // The updated cardinality estimate of the key is 20.`
	`2) "1.0000014901183398"  // 20 - 19 = 1`

CPC.ARRAY.UPDATE

Item	Description
Syntax	`CPC.ARRAY.UPDATE key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`
Time complexity	O(1)

Item	Description
Command description	Adds an item to the time window to which the specified timestamp belongs in the specified TairCpc key. If the key does not exist, the key is created. `SIZE` indicates the number of time windows, and `WIN` indicates the length of each time window. The length is measured in milliseconds. The key is updated as data streams are added to the key. During this process, data that is generated during a time-window range is saved. The time-window range is calculated by using the following formula: Time-window range = `SIZE` × `WIN`. Data that is generated outside of this time-window range is overwritten and deleted. `SIZE` and `WIN` are valid only at the point in time when the key is created. **Note** For example, if you want to calculate the amount of data in the key that was generated per minute during the last 10 minutes, you can set `SIZE` to 10 (10 time windows) and `WIN` to 60000 (1 minute for each time window). In this case, if you write the data that was generated during the 11th minute to the key, the data that was generated during the first minute is overwritten and deleted.
Parameter	• key: the name of the TairCpc key that you want to manage by running this command. • timestamp: the UNIX timestamp. Unit: milliseconds. • item: the item that you want to add. • EX: the relative expiration time of the key. Unit: seconds. If this parameter is not specified, the key does not expire. • EXAT: the absolute expiration time of the key. Unit: seconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire. • PX: the relative expiration time of the key. Unit: milliseconds. If this parameter is not specified, the key does not expire. • PXAT: the absolute expiration time of the key. Unit: milliseconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire. • SIZE: the number of time windows. Default value: 10. Valid values: 1 to 1000. We recommend that you set this parameter to a value that is less than 120. • WIN: the length of each time window. Unit: milliseconds. Default value: 60000. 60000 milliseconds are equal to 1 minute.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`CPC.ARRAY.UPDATE foo 1645584510000 f1 SIZE 120 WIN 10000`
	Sample output:
	`OK`

CPC.ARRAY.ESTIMATE

Item	Description
Syntax	`CPC.ARRAY.ESTIMATE key timestamp`
Time complexity	O(1)
Command description	Retrieves the cardinality estimate of a specified TairCpc key within the time window to which the specified timestamp belongs.
Parameter	• key: the name of the TairCpc key that you want to manage by running this command. • timestamp: the UNIX timestamp. Unit: milliseconds.
Output	• If the operation is successful, the cardinality estimate of the key within the time window is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`CPC.ARRAY.ESTIMATE foo 1645584532000`
	Sample output:
	`"2"`

CPC.ARRAY.ESTIMATE.RANGE

Item	Description
Syntax	`CPC.ARRAY.ESTIMATE.RANGE key start_time end_time`
Time complexity	O(1)
Command description	Retrieves the cardinality estimates of the time windows that reside in the specified time range in the specified TairCpc key. The time range is a closed interval.
Parameter	• key: the name of the TairCpc key that you want to manage by running this command. • start_time: the beginning of the time range to query. Unit: milliseconds. The value must be a UNIX timestamp. • end_time: the end of the time range to query. Unit: milliseconds. The value must be a UNIX timestamp.
Output	• If the operation is successful, the cardinality estimates of the key within the time windows are returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`CPC.ARRAY.ESTIMATE.RANGE foo 1645584510000 1645584550000`
	Sample output:
	`1) "2"`
	`2) "0"`
	`3) "1"`
	`4) "0"`
	`5) "0"`

CPC.ARRAY.ESTIMATE.RANGE.MERGE

Item	Description
Syntax	`CPC.ARRAY.ESTIMATE.RANGE.MERGE` *key timestamp range*
Time complexity	O(1)
Command description	Retrieves the cardinality estimate of the specified TairCpc key after merging and deduplication at a specific point in time and during a subsequent time range.
Parameter	• key: the name of the TairCpc key that you want to manage by running this command. • timestamp: the beginning of the time range to query. Unit: milliseconds. The value must be a UNIX timestamp. • range: the time range to query. Unit: milliseconds. The value must be a UNIX timestamp.
Output	• If the operation is successful, the cardinality estimate of the key after deduplication in the specified time range is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`CPC.ARRAY.ESTIMATE.RANGE.MERGE foo 1645584510000 100000`
	Sample output:
	`"6"`

CPC.ARRAY.UPDATE2EST

Item	Description
Syntax	`CPC.ARRAY.UPDATE2EST key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`
Time complexity	O(1)
Command description	Adds an item to the time window to which the specified timestamp belongs in the specified TairCpc key and returns the updated cardinality estimate of the key within the time window. If the key does not exist, the key is created. This command acts like CPC.ARRAY.UPDATE.
Parameter	• key: the name of the TairCpc key that you want to manage by running this command. • timestamp: the UNIX timestamp. Unit: milliseconds. • item: the item that you want to add. • EX: the relative expiration time of the key. Unit: seconds. If this parameter is not specified, the key does not expire. • EXAT: the absolute expiration time of the key. Unit: seconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire. • PX: the relative expiration time of the key. Unit: milliseconds. If this parameter is not specified, the key does not expire. • PXAT: the absolute expiration time of the key. Unit: milliseconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire. • SIZE: the number of time windows. Default value: 10. Valid values: 1 to 1000. We recommend that you set this parameter to a value that is less than 120. • WIN: the length of each time window. Unit: milliseconds. Default value: 60000. 60000 milliseconds are equal to 1 minute.
Output	• If the operation is successful, the updated cardinality estimate is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`CPC.ARRAY.UPDATE2EST foo 1645584530000 f3`
	Sample output:
	`"3"`

CPC.ARRAY.UPDATE2JUD

Item	Description
Syntax	`CPC.ARRAY.UPDATE2JUD key timestamp item [EX|EXAT|PX|PXAT time] [SIZE size] [WIN window_length]`

Item	Description
Time complexity	O(1)
Command description	Adds an item to the time window to which the specified timestamp belongs in the specified TairCpc key and returns the updated cardinality estimate of the key within the time window and the difference between the original and updated estimates. If the item is added and no duplication exists, a difference of 1 is returned. If the item already exists, a difference of 0 is returned. If the key does not exist, the key is created. This command acts like CPC.ARRAY.UPDATE.
Parameter	• key: the name of the TairCpc key that you want to manage by running this command. • item: the item that you want to add. • EX: the relative expiration time of the key. Unit: seconds. If this parameter is not specified, the key does not expire. • EXAT: the absolute expiration time of the key. Unit: seconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire. • PX: the relative expiration time of the key. Unit: milliseconds. If this parameter is not specified, the key does not expire. • PXAT: the absolute expiration time of the key. Unit: milliseconds. The value must be a UNIX timestamp. If this parameter is not specified, the key does not expire. • SIZE: the number of time windows. Default value: 10. Valid values: 1 to 1000. We recommend that you set this parameter to a value that is less than 120. • WIN: the length of each time window. Unit: milliseconds. Default value: 60000. 60000 milliseconds are equal to 1 minute.
Output	• If the operation is successful, the updated estimate and the difference within the time window are returned. • Otherwise, an error message is returned.

Example	Sample command:
	`CPC.ARRAY.UPDATE2JUD foo 1645584530000 f7`
	Sample output:
	`1) "8"  // The updated cardinality estimate of the key is 8.`
	`2) "1"  // 8 - 7 = 1`

3.6.10. TairZset

TairZset is a data structure that allows you to sort data of the DOUBLE type with respect to 256 dimensions. It is ideal for sorting and ranking data for use in industries such as gaming, live streaming, music, and e-commerce. This data structure improves data processing efficiency and is also easy to implement on the client side without the need to encode, decode, or encapsulate the data.

Overview

The Sorted Set (or ZSET) data structure of open source Redis allows you to sort DOUBLE-typed scores only by one dimension instead of multiple dimensions. For example, you can use the IEEE Standard for Floating-Point Arithmetic (IEEE 754) standard to implement multidimensional sorting by concatenating score data. However, this method has drawbacks, such as complex logic, reduced precision, and unavailability of the **EXZINCRBY** command.

To help you implement multidimensional sorting, Alibaba Cloud developed the TairZset data structure. Compared with the preceding method, TairZset has the following advantages:

- Allows DOUBLE-typed scores to be sorted based on a maximum of 256 dimensions. The scores are displayed from left to right based on their priorities.

  In a multidimensional sorting, a left score has higher priority than a right score. Take the comparison of three-dimensional scores in the score1#score2#score3 format as an example. TairZset compares the score1s of multiple three-dimensional scores and moves on to score2s only when score1s are equal. If score1s are not equal, the ranking of score1s represents the ranking of the three-dimensional scores involved. By the same logic, score3s are compared only if score2s are equal. If all score1s are equal and the same holds true for score2s and score3s, the involved multidimensional scores are ranked in ASCII sort order.

  For easier understanding, you can imagine number signs (#) as decimal points (.). This way, 0#99 < 99#90 < 99#99 can be seen as 0.99 < 99.90 < 99.99.
- Supports the **EXZINCRBY** command. You no longer need to perform the following operations: retrieve current data, apply the increments to the data, and then write the data back to Redis databases.
- Supports APIs similar to those available for native Redis ZSET.
- Allows you to implement and regular leaderboardsdistributed leaderboards
- Supports the open source TairJedis client. For more information about the TairJedis client, visit alibabacloud-tairjedis-sdk. You can use the TairJedis client without the need to encode, decode, or encapsulate data. You can also encapsulate clients for other programming languages by referring to the open source code.

Typical scenarios

TairZset is ideal for implementing leaderboards for use in industries such as gaming, live streaming, music, and e-commerce. The following section lists some examples:

- Leaderboards for the live streaming industry: In live matches, commentators are sorted by their current popularity value. If the popularity value is the same, they are sorted by the number of their likes. If the number of likes is also the same, they are sorted by the value of their gifts.
- Leaderboards for medals: Participants are sorted by the number of their gold, silver, and bronze medals. If the number of gold medals is the same, they are sorted by the number of silver medals. If the number of silver medals is also the same, they are sorted by the number of bronze medals.
- Leaderboards for the gaming industry: Players are sorted based on multiple factors, including their scores, task completion speeds, and achievement levels.

The module is open-sourced. For more information, visit TairZset.

Best practices

- Implement multidimensional leaderboards by using TairZset
- Implement distributed leaderboards by using TairZset

Prerequisites

The instance is a performance-enhanced instance of the AsaraDB for Redis Enhanced Edition (Tair) that runs the minor version of 1.7.1 or later. For more information about performance-enhanced instances, see Performance-enhanced instances.

> **Note** We recommend that you update your instance to the latest minor version for more features and higher stability. For more information, see Update the minor version. If your instance is a cluster or read/write splitting instance, we recommend that you update the proxy nodes in the instance to the latest minor version. This ensures that all commands can be run as expected. For more information about cluster and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Precautions

The TairZset data that you want to manage is stored on the performance-enhanced instance.

TairZset commands

Command	Syntax	Description
EXZADD	`EXZADD key [NX|XX] [CH] [INCR] score member [score member ...]`	Stores the specified element and its score in a TairZset key. You can specify multiple scores and elements. **Note** To implement multidimensional sorting, you can separate the scores of each dimension with number signs (#), such as `111#222#121`. All elements of a key must have the same score format.
EXZINCRBY	`EXZINCRBY key increment member`	Increases the score of an element in a TairZset key. increment indicates the value that you want to add to the score.
EXZSCORE	`EXZSCORE key member`	Returns the score of an element in a TairZset key. If the key or the element does not exist, a value of nil is returned.
EXZRANGE	`EXZRANGE key min max [WITHSCORES]`	Returns the elements of a TairZset key within the specified score range.
EXZREVRANGE	`EXZREVRANGE key min max [WITHSCORES]`	Returns the elements of a TairZset key within the specified score range. Elements are stored by score in descending order, and elements with the same score are sorted in reverse lexicographical order. **Note** This command is similar to EXZRANGE except that this command sorts the results in reverse.

Command	Syntax	Description
EXZRANGEBYSCORE	`EXZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]`	Returns the elements of a TairZset key whose scores are greater than or equal to the min value and are less than or equal to the max value. The returned elements are sorted by score in ascending order, and elements with the same score are returned in lexicographical order.
EXZREVRANGEBYSCORE	`EXZREVRANGEBYSCORE key max min [WITHSCORES] [LIMIT offset count]`	Returns the elements of a TairZset key whose scores are greater than or equal to the min value and are less than or equal to the max value. Contrary to the default sorting of TairZset elements, the elements returned by this command are sorted by score in descending order, and elements with the same score are sorted in reverse lexicographical order. **Note** This command is similar to EXZRANGEBYSCORE except that this command sorts the results in reverse and places max in front of min.
EXZRANGEBYLEX	`EXZREMRANGEBYLEX key min max`	Returns the elements whose scores are between the min value and the max value when all elements of a key have the same score to ensure that elements are sorted in lexicographical order.
EXZREVRANGEBYLEX	`EXZREVRANGEBYLEX key max min [LIMIT offset count]`	Returns the elements whose scores are between the min value and the max value when all elements of a key have the same score to ensure that elements are sorted in lexicographical order. **Note** This command is similar to EXZRANGEBYLEX except that this command sorts the results in reverse and places max in front of min.
EXZREM	`EXZREM key member [member ...]`	Removes specified elements from a TairZset key. If the specified element does not exist, the element is ignored.
EXZREMRANGEBYSCORE	`EXZREMRANGEBYSCORE key min max`	Removes the elements from a TairZset key whose scores are greater than or equal to the min value and are less than or equal to the max value.
EXZREMRANGEBYRANK	`EXZREMRANGEBYRANK key start stop`	Removes the elements from a TairZset key whose ranks are within the range of the start value and the stop value.

Command	Syntax	Description
EXZREMRANGEBYLEX	`EXZREMRANGEBYLEX key min max`	Removes the elements whose scores are between the min value and the max value when all elements of a key have the same score to ensure that elements are sorted in lexicographical order. **Note** If you use the min and max parameters to specify the same range for the EXZREMRANGEBYLEX and EXZRANGEBYLEX commands, the elements removed by the EXZREMRANGEBYLEX command are the same as those returned by the EXZRANGEBYLEX command.
EXZCARD	`EXZCARD key`	Returns the cardinality of a TairZset key. Cardinality indicates the number of elements in a key.
EXZRANK	`EXZRANK key member`	Returns the rank of an element in a TairZset key. Ranks are sorted by score in ascending order. Ranks use a zero-based numbering scheme, and the rank of the element that has the lowest score is 0. Rank is also known as index.
EXZREVRANK	`EXZREVRANK key member`	Returns the rank of an element in a TairZset key. Ranks are sorted by score in descending order. Ranks use a zero-based numbering scheme, and the rank of the element that has the highest score is 0. Rank is also known as index. **Note** This command is similar to EXZRANK except that this command sorts the results in reverse.
EXZCOUNT	`EXZCOUNT key min max`	Returns the number of elements in a TairZset key whose scores are between the min value and the max value.
EXZLEXCOUNT	`EXZLEXCOUNT key min max`	Returns the number of elements whose scores are between the min value and the max value when all elements of a key have the same score to ensure that elements are sorted in lexicographical order.
EXZRANKBYSCORE	`EXZRANKBYSCORE key score`	Calculates the rank of the specified element score in a TairZset key. Ranks are sorted by score in ascending order. Ranks use a zero-based numbering scheme, and the rank of the element that has the lowest score is 0. Rank is also known as index. **Note** If the score does not exist, the estimated rank of the score in the key is returned. By default, if the score already exists, Tair ranks the score in front of the existing score in the key.

Command	Syntax	Description
EXZREVRANKBYSCORE	`EXZREVRANKBYSCORE key score`	Calculates the rank of the specified element score in a TairZset key. Ranks are sorted by score in descending order. Ranks use a zero-based numbering scheme, and the rank of the element that has the highest score is 0. Rank is also known as index. **Note** If the score does not exist, the estimated rank of the score in the key is returned. By default, if the score already exists, Tair ranks the score behind the existing score in the key.
DEL	`DEL key [key ...]`	Deletes one or more TairZset keys.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

EXZADD

Item	Description
Syntax	`EXZADD key [NX|XX] [CH] [INCR] score member [score member ...]`
Time complexity	O(N)
Command description	Adds the specified element with the specified score to a TairZset key. You can specify multiple scores and elements. The system uses different strategies based on whether the key and element exist: **Note** To implement multidimensional sorting, you can separate the scores of each dimension with number signs (#), such as `111#222#121`. All elements of a key must have the same score format. • If the key exists but its data structure is not TairZset, an error is returned. • If the key does not exist, the system creates a TairZset key and adds the element to the key. • If the element is already an element of the key, the score of the element is updated and the element is reinserted into the right position to avoid disruptions to sorting. • Each score is represented by a string of double-precision floating-point numbers. The +inf and -inf values are valid values.

Item	Description
Parameter	• NX: adds new elements and does not update existing elements. • XX: updates existing elements and does not add new elements. • CH: changes the command output from the number of new elements added to the total number of elements that have changed. **Note** Changed elements include new elements and existing elements with updated scores. If the score of an existing element in the command line does not change, the element is not counted as a changed element. • INCR: When this parameter is specified, EXZADD acts similarly to EXZINCRBY, which indicates that only one pair of score and element can be specified in this mode.
Output	The output is an integer. The following rules describe the output: • If no parameters are specified, the output is the number of elements added to a TairZset key. • If the CH parameter is specified, the output is the number of elements that have changed (elements that have been added or updated). • If the INCR parameter is specified, the command returns the new score of the element as a string. If multidimensional scores are used, the score is returned in the `"score1# score2# score3#..."` format, such as `2#0#6`. **Note** If this command is stopped because the XX or NX parameter is included in this command, a value of nil is returned.

Example	Sample command:
	`EXZADD testkey NX 1#0#3 a 1#0#2 b`
	Sample output:
	`(integer) 2`

EXZINCRBY

Item	Description
Syntax	`EXZINCRBY` *key increment member*
Time complexity	O(log(N))

Item	Description
Command description	Increases the score of an element in a TairZset key. increment indicates the value that you want to add to the score. The system uses different strategies based on whether the key and element exist: • If the key exists but its data structure is not TairZset, an error is returned. • If the key does not exist, the system creates a TairZset key and uses the element as the only element of the key. • If the key has no element, the system adds an element whose score is the increment value to the key, which indicates that the original score of the element is assumed to be 0.0. **Note** • To implement multidimensional sorting, you can separate the scores of each dimension with number signs (#), such as `111#222#121`. All elements of a key must have the same score format. • The score value is a string of numeric values and can be a string of double-precision floating-point numbers. If you want to lower the score of an element, specify a negative number.
Parameter	None
Output	The new score of an element as a string. If multidimensional scores are used, the scores are returned in the `"score1# score2# score3#..."` format, such as `2#0#6`.

Example	Sample command:
	`EXZINCRBY testkey 2#2#1 a`
	Sample output:
	`"3#2#4"`

EXZSCORE

Item	Description
Syntax	`EXZSCORE key member`
Time complexity	O(1)
Command description	Returns the score of an element in a TairZset key. If the key or the element does not exist, a value of nil is returned.
Parameter	None
Output	The score of an element as a string. If multidimensional scores are used, the scores are returned in the `"score1# score2# score3#..."` format, such as `2#0#6`.

Item	Description

Example	Sample command:
	`EXZSCORE testkey a`
	Sample output:
	`"3#2#4"`

EXZRANGE

Item	Description
Syntax	`EXZRANGE key min max [WITHSCORES]`
Time complexity	O(log(N)+M), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be returned.
Command description	Returns the elements of a TairZset key within the specified score range.

Parameter	• min and max: indicate zero-based indexes, where 0 is the first element and 1 is the next element. Other elements follow the same rule. You can use the two parameters to specify a closed interval.
	Note
	◦ The indexes can also be negative numbers, which indicate offsets from the end of TairZset elements returned. For example, -1 indicates the last element of a key and -2 indicates the second to last element of a key. Other elements follow the same rule.
	◦ To query all elements, you can set min to 0 and max to -1.
	◦ If the min value is greater than the index of the last element of a key or the max value, an empty list is returned.
	• WITHSCORES: returns the scores of elements. The scores are returned in the `value1, score1, ..., valueN, scoreN` format. Example:
	`1) "b"`
	`2) "1#0#2"`
	`3) "a"`
	`4) "3#2#4"`

Output	A list of elements within the specified score range. If the WITHSCORES parameter is specified, the scores of elements are returned.

Item	Description

Example	Sample command:
	`EXZRANGE testkey 0 -1 WITHSCORES`
	Sample output:
	`1) "b"`
	`2) "1#0#2"`
	`3) "a"`
	`4) "3#2#4"`

EXZREVRANGE

Item	Description
Syntax	`EXZREVRANGE` *key min max* `[WITHSCORES]`
Time complexity	O(log(N)+M), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be returned.
Command description	Returns the elements of a TairZset key within the specified score range. Elements are stored by score in descending order, and elements with the same score are sorted in reverse lexicographical order. **Note** This command is similar to EXZRANGE except that this command sorts the results in reverse.

Item	Description

Parameter	• min and max: indicate zero-based indexes, where 0 is the first element and 1 is the next element. Other elements follow the same rule. You can use the two parameters to specify a closed interval.
	Note
	◦ The indexes can also be negative numbers, which indicate offsets from the end of TairZset elements returned. For example, -1 indicates the last element of a key and -2 indicates the second to last element of a key. Other elements follow the same rule.
	◦ To query all elements, you can set min to 0 and max to -1.
	◦ If the min value is greater than the index of the last element of a key or the max value, an empty list is returned.
	• WITHSCORES: returns the scores of elements. The scores are returned in the `value1, score1, ..., valueN, scoreN` format. Example:
	`1) "b"`
	`2) "1#0#2"`
	`3) "a"`
	`4) "3#2#4"`

Output	A list of elements within the specified score range. If the WITHSCORES parameter is specified, the scores of elements are returned.

Example	Sample command:
	`EXZREVRANGE testkey 0 -1 WITHSCORES`
	Sample output:
	`1) "a"`
	`2) "3#2#4"`
	`3) "b"`
	`4) "1#0#2"`

EXZRANGEBYSCORE

Item	Description
Syntax	`EXZRANGEBYSCORE` *key min max* `[WITHSCORES] [LIMIT` *offset count*`]`

Item	Description
Time complexity	O(log(N)+M), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be returned. **Note** When M is set to a constant, this expression can be regarded as O(log(N)). For example, you can specify that the first 10 elements are always returned by using the LIMIT clause.
Command description	Returns the elements of a TairZset key whose scores are greater than or equal to the min value and are less than or equal to the max value. The returned elements are sorted by score in ascending order, and elements with the same score are returned in lexicographical order.
Parameter	• min and max: indicate the lowest score and the highest score. If multidimensional scores are used, scores of each dimension are separated by a number sign (#). **Note** ◦ If you are not sure about the highest and lowest element scores of a key and you want to query elements whose scores are greater than or equal to or are less than or equal to the specified value, set min to negative infinity (`-inf`) and max to positive infinity (`+inf`). ◦ The default score range is a closed interval. To specify an open interval, add a parenthesis. For example, `(1 5` indicates that elements whose scores are greater than 1 and are less than or equal to 5 are returned. • WITHSCORES: returns the scores of elements. • LIMIT offset count: specifies the number and interval of returned elements. If you set count to a negative integer, all elements starting from the specified offset are returned. **Note** If the offset value is large, the key needs to be traversed to identify offset elements before elements are returned. This increases time complexity.
Output	A list of elements within the specified score range. If the WITHSCORES parameter is specified, the scores of elements are returned.

Item	Description

Example	Sample command:
	`EXZRANGEBYSCORE testkey 0#0#0 6#6#6 WITHSCORES`
	Sample output:
	`1) "b"`
	`2) "1#0#2"`
	`3) "a"`
	`4) "3#2#4"`

EXZREVRANGEBYSCORE

Item	Description
Syntax	`EXZREVRANGEBYSCORE` *key max min* `[WITHSCORES] [LIMIT` *offset count*`]`
Time complexity	O(log(N)+M), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be returned. **Note** When M is set to a constant, this expression can be regarded as O(log(N)). For example, you can specify that the first 10 elements are always returned by using the LIMIT clause.
Command description	Returns the elements of a TairZset key whose scores are greater than or equal to the min value and are less than or equal to the max value. Contrary to the default sorting of TairZset elements, the elements returned by this command are sorted by score in descending order, and elements with the same score are sorted in reverse lexicographical order. **Note** This command is similar to EXZRANGEBYSCORE except that this command sorts the results in reverse and places max in front of min.

Item	Description
Parameter	• min and max: indicate the lowest score and the highest score. If multidimensional scores are used, scores of each dimension are separated by a number sign (#). **Note** ◦ If you are not sure about the highest and lowest element scores of a key and you want to query elements whose scores are greater than or equal to or are less than or equal to the specified value, set min to negative infinity (`-inf`) and max to positive infinity (`+inf`). ◦ The default score range is a closed interval. To specify an open interval, add a parenthesis. For example, `(1 5` indicates that elements whose scores are greater than 1 and are less than or equal to 5 are returned. • WITHSCORES: returns the scores of elements. • LIMIT offset count: specifies the number and interval of returned elements. If you set count to a negative integer, all elements starting from the specified offset are returned. **Note** If the offset value is large, the key needs to be traversed to identify offset elements before elements are returned. This increases time complexity.
Output	A list of elements within the specified score range. If the WITHSCORES parameter is specified, the scores of elements are returned.
Example	Sample command: `EXZREVRANGEBYSCORE testkey 6#6#6 0#0#0` Sample output: 1) "a" 2) "3#2#4" 3) "b" 4) "1#0#2"

EXZRANGEBYLEX

Item	Description
Syntax	`EXZRANGEBYLEX` *key min max* `[LIMIT` *offset count*`]`

Item	Description
Time complexity	O(log(N)+M), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be returned. **Note** When M is set to a constant, this algorithm can be regarded as O(log(N)). For example, you can specify that the first 10 elements are always returned by using the LIMIT offset count parameter.
Command description	Returns the elements whose scores are between the min value and the max value when all elements of a key have the same score to ensure that elements are sorted in lexicographical order. **Note** • If the elements of a key have different scores, the returned elements are unknown. • You can use the `memcmp() C` function to compare two string elements byte-by-byte. Elements are sorted in ascending order based on the preceding comparison. If two strings contain the same substring, the longer string is assigned a greater value than the shorter string.
Parameter	• min and max: indicate the minimum and maximum string representation values of element names. You must specify the intervals of characters. The following section shows how to specify intervals: ◦ To specify an open interval, add a parenthesis. Example: `(a`. ◦ To specify a closed interval, add a bracket. Example: `[a`. **Note** `+` indicates positive infinity, and `-` indicates negative infinity. • LIMIT offset count: specifies the number and interval of returned elements. If you set count to a negative integer, all elements starting from the specified offset are returned. **Note** If the offset value is large, the key needs to be traversed to identify offset elements before elements are returned. This increases time complexity.
Output	A list of elements whose values are within the specified range.

Example	Sample command:
	`EXZRANGEBYLEX zzz [a [b`
	Sample output:
	`1) "aba"`
	`2) "abc"`

EXZREVRANGEBYLEX

Item	Description
Syntax	`EXZREVRANGEBYLEX key max min [LIMIT offset count]`
Time complexity	O(log(N)+M), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be returned. **Note** When M is set to a constant, this expression can be regarded as O(log(N)). For example, you can specify that the first 10 elements are always returned by using the LIMIT clause.
Command description	Returns the elements whose scores are between the min value and the max value when all elements of a key have the same score to ensure that elements are sorted in lexicographical order. **Note** This command is similar to EXZRANGEBYLEX except that this command sorts the results in reverse and places max in front of min.
Parameter	• min and max: indicate the minimum and maximum string representation values of element names. You must specify the intervals of characters. The following section shows how to specify intervals: ◦ To specify an open interval, add a parenthesis. Example: `(a`. ◦ To specify a closed interval, add a bracket. Example: `[a`. **Note** `+` indicates positive infinity, and `-` indicates negative infinity. • LIMIT offset count: specifies the number and interval of returned elements. If you set count to a negative integer, all elements starting from the specified offset are returned. **Note** If the offset value is large, the key needs to be traversed to identify offset elements before elements are returned. This increases time complexity.
Output	A list of elements whose values are within the specified range.

Example	Sample command:
	`EXZREVRANGEBYLEX zzz [b [a`
	Sample output:
	`1) "abc"`
	`2) "aba"`

EXZREM

Item	Description
Syntax	`EXZREM key member [member ...]`
Time complexity	O(M*log(N)), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be removed.
Command description	Removes specified elements from a TairZset key. If the specified element does not exist, the element is ignored. **Note** If the key exists but its data structure is not TairZset, an error is returned.
Parameter	None
Output	The number of elements removed from the key. The elements that do not exist are not included.

Example	Sample command:
	`EXZREM testkey a`
	Sample output:
	`(integer) 1`

EXZREMRANGEBYSCORE

Item	Description
Syntax	`EXZREMRANGEBYSCORE key min max`
Time complexity	O(log(N)+M), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be removed.
Command description	Removes the elements from a TairZset key whose scores are greater than or equal to the min value and are less than or equal to the max value.

Item	Description
Parameter	min and max: indicate the lowest score and the highest score. If multidimensional scores are used, scores of each dimension are separated by a number sign (#). **Note** • If you are not sure about the highest and lowest element scores of a key and you want to remove elements whose scores are greater than or equal to or are less than or equal to the specified value, set min to negative infinity (`-inf`) and max to positive infinity (`+inf`). • The default score range is a closed interval. To specify an open interval, add a parenthesis. For example, `EXZREMRANGEBYSCORE (1 5` indicates that elements whose scores are greater than 1 and are less than or equal to 5 are deleted.
Output	The number of elements removed.

Example	Sample command:
	`EXZREMRANGEBYSCORE testkey 3#2#4 6#6#6`
	Sample output:
	`(integer) 1`

EXZREMRANGEBYRANK

Item	Description
Syntax	`EXZREMRANGEBYRANK` *key start stop*
Time complexity	O(log(N)+M), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be removed.
Command description	Removes the elements from a TairZset key whose ranks are within the range of the start value and the stop value.

Item	Description
Parameter	Both start and stop indicate zero-based indexes. 0 indicates the element that has the lowest score. These indexes can be negative numbers. These numbers indicate offsets that start at the element that has the highest score. For example, -1 indicates the element that has the highest score, and -2 indicates element that has the second highest score. Other elements follow the same rule.
Output	The number of elements removed.

Example	Sample command:
	`EXZREMRANGEBYRANK testkey 0 1`
	Sample output:
	`(integer) 1`

EXZREMRANGEBYLEX

Item	Description
Syntax	`EXZREMRANGEBYLEX` *key min max*
Time complexity	O(log(N)+M), where N indicates the number of elements in the TairZset key and M indicates the number of elements to be removed.
Command description	Removes the elements whose scores are between the min value and the max value when all elements of a key have the same score to ensure that elements are sorted in lexicographical order. **Note** If you use the min and max parameters to specify the same range for the EXZREMRANGEBYLEX and EXZRANGEBYLEX commands, the elements removed by the EXZREMRANGEBYLEX command are the same as those returned by the EXZRANGEBYLEX command.
Parameter	min and max: indicate the minimum and maximum string representation values of element names. You must specify the intervals of characters. The following section shows how to specify intervals: • To specify an open interval, add a parenthesis. Example: `(a`. • To specify a closed interval, add a bracket. Example: `[a`.
Output	The number of elements removed.

Item	Description

Example	Sample command:
	`EXZREMRANGEBYLEX [a [b`
	Sample output:
	`(integer) 2`

EXZCARD

Item	Description
Syntax	`EXZCARD key`
Time complexity	O(1)
Command description	Returns the cardinality of a TairZset key. Cardinality indicates the number of elements in a key.
Parameter	None
Output	The number of elements in a TairZset key. If the key does not exist, a value of 0 is returned.

Example	Sample command:
	`EXZCARD testkey`
	Sample output:
	`(integer) 2`

EXZRANK

Item	Description
Syntax	`EXZRANK key member`
Time complexity	O(log(N))
Command description	Returns the rank of an element in a TairZset key. Ranks are sorted by score in ascending order. Ranks use a zero-based numbering scheme, and the rank of the element that has the lowest score is 0. Rank is also known as index.
Parameter	None

Item	Description
Output	• If the element exists, the integer rank of the element is returned. • If the key or the element in the key does not exist, a value of nil is returned.

Example	Sample command:
	`EXZRANK testkey b`
	Sample output:
	`(integer) 0`

EXZREVRANK

Item	Description
Syntax	`EXZREVRANK key member`
Time complexity	O(log(N))
Command description	Returns the rank of an element in a TairZset key. Ranks are sorted by score in descending order. Ranks use a zero-based numbering scheme, and the rank of the element that has the highest score is 0. Rank is also known as index. **Note** This command is similar to EXZRANK except that this command sorts the results in reverse.
Parameter	None
Output	• If the element exists, the integer rank of the element is returned. • If the key or the element in the key does not exist, a value of nil is returned.

Example	Sample command:
	`EXZREVRANK testkey b`
	Sample output:
	`(integer) 1`

EXZCOUNT

Item	Description
Syntax	`EXZCOUNT key min max`
Time complexity	O(log(N)), where N indicates the number of elements in the TairZset key. **Note** This command uses element ranks to obtain the query range. Therefore, the workloads associated with this command are not proportional to the size of the range.
Command description	Returns the number of elements in a TairZset key whose scores are between the min value and the max value.
Parameter	min and max: indicate the lowest score and the highest score. If multidimensional scores are used, scores of each dimension are separated by a number sign (#). **Note** • If you are not sure about the highest and lowest element scores of a key and you want to query elements whose scores are greater than or equal to or are less than or equal to the specified value, set min to negative infinity and max to positive infinity. • The default score range is a closed interval. To specify an open interval, add a parenthesis. For example, `(1 5` indicates that elements whose scores are greater than 1 and are less than or equal to 5 are returned.
Output	The integer number of elements within the specified score range.

Example	Sample command:
	`EXZCOUNT testkey (1#0#2 6#6#6`
	Sample output:
	`(integer) 1`

EXZLEXCOUNT

Item	Description
Syntax	`EXZLEXCOUNT key min max`
Time complexity	O(log(N)), where N indicates the number of elements in the TairZset key. **Note** This command uses element ranks to obtain the query range. Therefore, the workloads associated with this command are not proportional to the size of the range.

Item	Description
Command description	Returns the number of elements whose scores are between the min value and the max value when all elements of a key have the same score to ensure that elements are sorted in lexicographical order. **Note** • If the elements of a key have different scores, the returned elements are unknown. • You can use the `memcmp() C` function to compare two string elements byte-by-byte. Elements are sorted in ascending order based on the preceding comparison. If two strings contain the same substring, the longer string is assigned a greater value than the shorter string.
Parameter	min and max: indicate the minimum and maximum string representation values of element names. You must specify the intervals of characters. The following section shows how to specify intervals: • To specify an open interval, add a parenthesis. Example: `(a`. • To specify a closed interval, add a bracket. Example: `[a`.
Output	The integer number of elements within the specified score range.

Example	Sample command:
	`EXZLEXCOUNT zzz [a [b`
	Sample output:
	`(integer) 2`

EXZRANKBYSCORE

Item	Description
Syntax	`EXZRANKBYSCORE` *key score*
Time complexity	O(log(N))

Item	Description
Command description	Calculates the rank of the specified element score in a TairZset key. Ranks are sorted by score in ascending order. Ranks use a zero-based numbering scheme, and the rank of the element that has the lowest score is 0. Rank is also known as index. **Note** If the score does not exist, the estimated rank of the score in the key is returned. By default, if the score already exists, Tair ranks the score in front of the existing score in the key.
Parameter	None
Output	The rank of the score in the key.

Example	Sample command:
	`EXZRANKBYSCORE testkey 2#0#2`
	Sample output:
	`(integer) 1`

EXZREVRANKBYSCORE

Item	Description
Syntax	`EXZREVRANKBYSCORE` *key score*
Time complexity	O(log(N))
Command description	Calculates the rank of the specified element score in a TairZset key. Ranks are sorted by score in descending order. Ranks use a zero-based numbering scheme, and the rank of the element that has the highest score is 0. Rank is also known as index. **Note** If the score does not exist, the estimated rank of the score in the key is returned. By default, if the score already exists, Tair ranks the score behind the existing score in the key.
Parameter	None
Output	The rank of the score in the key.

Item	Description

Example	Sample command:
	`EXZREVRANKBYSCORE testkey 2#0#2`
	Sample output:
	`(integer) 1`

3.6.11. TairRoaring

The TairRoaring data structure is a Tair-based Roaring bitmap implementation. This topic introduces TairRoaring and its supported commands.

Overview

Bitmap (also known as bitset) is a common data structure that uses small amounts of storage to optimize queries of large amounts of data. Bitmaps save more space than hash-based implementations but are unsuitable for storing sparse data. Compressed bitmaps were developed in response to this issue. Roaring bitmaps are an industry-recognized bitmap type that is more efficient and balanced than other compressed bitmaps.

Roaring bitmaps are optimized in TairRoarings in the following ways:

- TairRoarings can strike a balance between performance and space complexity in a large number of scenarios by means of two-level indexes and dynamic containers.
- TairRoarings use optimization techniques such as single instruction, multiple data (SIMD), vectorization, and popcount algorithms to improve computing efficiency and deliver efficient time and space complexity.
- TairRoarings provide powerful computing performance and high stability for a variety of business scenarios based on ApsaraDB for Redis Enhanced Edition (Tair).

Typical scenarios

TairRoarings are suitable for use in industries such as live streaming, music, and e-commerce. You can use TairRoarings to add multidimensional tags to users for scenarios such as personalized recommendation and precision marketing.

Release notes

> **Notice** Changes in TairRoaring V2:
>
> - TR.RANGEINTARRAY: TR.RANGEINTARRAY in TairRoaring V1 is changed to TR.RANGE in TairRoaring V2.
> - TR.SETRANGE: The output of TR.SETRANGE is changed from `OK` in TairRoaring V1 to the number of bits that have been set to 1 in TairRoaring V2.

1. On Sep 13, 2021, TairRoaring was released. V1
2. On March 11, 2022, TairRoaring We recommend that you update the minor version of your instance to 1.7.27 or later. For more information, see Update the minor version.

TairRoaring V2 optimizes some command implementations and improves command performance. TairRoaring V2 comes with nine new commands including TR.SETBITS and TR.CLEARBITS, updates three commands with two of them still compatible with TairRoaring V1, and changes the name of one command.

V2

3. On April 20, 2022, TairRoaring We recommend that you update the minor version of your instance to 1.8.1 or later. For more information, see Update the minor version.

   TairRoaring V2.2 comes with three new commands, which are TR.JACCARD, TR.CONTAINS, and TR.RANK, and updates specific command outputs. For example, the `ERR key not found` error message is removed when the specified key does not exist. For more information, see Supported commands.

   V2.2

Best practices

Select users by using TairRoaring

Prerequisites

The instance is a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair) that runs the minor version of 1.7.7 or later. For more information about performance-enhanced instances, see Performance-enhanced instances.

> **Note** We recommend that you update your instance to the latest minor version for more features and higher stability. For more information, see Update the minor version. Specific new commands and parameters are not currently supported by cluster instances, but developers will add support over time. If you want to use the new commands and parameters on your cluster instances, submit a ticket. For more information about cluster instances, see Cluster master-replica instances.

Precautions

The TairRoarings that you want to manage are stored in the performance-enhanced instance.

Supported commands

Type	Command	Syntax	Description	Version change
	TR.SETBIT	`TR.SETBIT key offset value`	Sets the specified bit in a TairRoaring key to a value of 1 or 0 and returns the original bit value. The offset starts from 0.	- (N/A)
	TR.SETBITS	`TR.SETBITS key offset [offset1 offset2 ... offsetN]`	Sets the value of the specified bit in a TairRoaring key to 1. You can set multiple bits to a value of 1.	Added in V2.

Type	Command	Syntax	Description	Version change
Write operation	TR.CLEARBITS	`TR.CLEARBITS key offset [offset1 offset2 ... offsetN]`	Sets the value of the specified bit in a TairRoaring key to 0. If the specified bit already has a value of 0, the operation is not performed. You can set multiple bits to a value of 0.	Added in V2.
	TR.SETRANGE	`TR.SETRANGE key start end`	Sets the bits within the specified range in a TairRoaring key to a value of 1. The range is a closed interval.	Updated in V2. After the command update in V2, the command output is changed to the number of bits that have been set to 1.
	TR.APPENDBITARRAY	`TR.APPENDBITARRAY key offset bitarray`	Inserts a bit array into a position right behind the specified bit in a TairRoaring key and overwrites the original data. The bit array consists of 0 and 1.	Added in V2.
	TR.FLIPRANGE	`TR.FLIPRANGE key start end`	Changes the values for bits within the specified range in a TairRoaring key from 0 to 1 or from 1 to 0. The range is a closed interval. If the key does not exist, the key is created as an empty data set and the operation is performed on the key.	Added in V2.
	TR.APPENDINTARRAY	`TR.APPENDINTARRAY key value [value1 value2 ... valueN]`	Sets the value of the specified bit in a TairRoaring key to 1. You can set multiple bits to a value of 1. **Note** We recommend that you use TR.SETBITS in TairRoaring V2 instead of this command.	-

Type	Command	Syntax	Description	Version change
	TR.SETINTARRAY	`TR.SETINTARRAY key value [value1 value2 ... valueN]`	Creates a TairRoaring key from an integer array. If the key already exists, this command overwrites the data in the key. **Note** We recommend that you use TR.SETBITS in TairRoaring V2 instead of this command.	-
	TR.SETBITARRAY	`TR.SETBITARRAY key value`	Creates a TairRoaring key based on the specified bit array string. The bit array string consists of 0 and 1. If the key already exists, this command overwrites the data in the key. **Note** We recommend that you use TR.APPENDBITARRAY in TairRoaring V2 instead of this command.	-
	TR.BITOP	`TR.BITOP destkey operation key [key1 key2 ... keyN]`	Performs a bitwise operation on multiple TairRoaring keys and stores the result in the destination key. The *AND*, *OR*, *XOR*, *NOT*, and *DIFF* bitwise operations are supported. **Note** This command is unavailable for cluster instances.	-
	TR.BITOPCARD	`TR.BITOPCARD operation key [key1 key2 ... keyN]`	Performs a bitwise operation on multiple TairRoaring keys. The *AND*, *OR*, *XOR*, *NOT*, and *DIFF* bitwise operations are supported. **Note** This command is unavailable for cluster instances.	Added in V2.
	TR.OPTIMIZE	`TR.OPTIMIZE key`	Optimizes the storage of a TairRoaring key. If the key is relatively large and is used mainly for read operations after the key is created, you can run this command.	-

Type	Command	Syntax	Description	Version change
Read operation	TR.GETBIT	`TR.GETBIT key offset`	Retrieves the value of the specified bit from a TairRoaring key.	-
	TR.GETBITS	`TR.GETBITS key offset [offset1 offset2 ... offsetN]`	Retrieves the value of the specified bit from a TairRoaring key. You can specify multiple bits for value retrieval.	Added in V2.
	TR.BITCOUNT	`TR.BITCOUNT key [start end]`	Counts the number of bits that have a value of 1 within the specified range in a TairRoaring key. The range is a closed interval.	Updated in V2 and compatible with V1.
	TR.BITPOS	`TR.BITPOS <key> <value> [count]`	Retrieves the offset of the bit that has an ordinal number of count. A bit can have a value of 1 or 0. The count parameter is optional and has a default value of 1. A value of 1 indicates the first bit retrieved by using the left-right counting approach.	Updated in V2 and compatible with V1.
	TR.SCAN	`TR.SCAN key start_offset [COUNT count]`	Scans all bits located after a specified bit in a TairRoaring key and returns the offsets corresponding a count of the scanned bits that have a value of 1. The returned cursor is the offset corresponding to the key. **Note** This command may or may not scan and return added or deleted elements.	Added in V2.
	TR.RANGE	`TR.RANGE key start end`	Returns the offsets of the bits that have a value of 1 within the specified range in a TairRoaring key. The range is a closed interval.	Renamed as TR.RANGE in V2 from TR.RANGEINTARRAY in V1.
	TR.RANGEBITARRAY	`TR.RANGEBITARRAY key start end`	Retrieves a string that consists of bit values of 0 and 1 within the specified range in a TairRoaring key. The range is a closed interval.	Added in V2.

Type	Command	Syntax	Description	Version change
	TR.MIN	`TR.MIN key`	Retrieves the offset of the first bit that has a value of 1 in a TairRoaring key. If no bits have a value of 1, a value of -1 is returned.	-
	TR.MAX	`TR.MAX key`	Returns the offset of the last bit that has a value of 1 in a TairRoaring key. If no bits have a value of 1, -1 is returned.	-
	TR.STAT	`TR.STAT key [JSON]`	Returns the statistical information of the specified TairRoaring key. The information includes the number of containers and the memory usage.	Added in V2.
	TR.JACCARD	`TR.JACCARD key1 key2`	Retrieves the Jaccard similarity coefficient of two TairRoaring keys. The higher the coefficient, the higher the similarity. **Note** This command is unavailable for cluster instances.	Added in V2.2.
	TR.CONTAINS	`TR.CONTAINS key1 key2`	Checks whether key2 contains key1. If yes, key1 is a subset of key2 and a value of 1 is returned. Otherwise, key1 is not a subset of key2 and a value of 0 is returned. **Note** This command is unavailable for cluster instances.	Added in V2.2.
	TR.RANK	`TR.RANK key offset`	Retrieves the number of bits that have a value of 1 within the range from the first bit and the specified bit. The range is a closed interval.	Added in V2.2.
General-purpose operation	DEL	`DEL key [key ...]`	Deletes one or more TairRoaring keys.	-

Note

- The following section describes the command syntax:
  - `Uppercase keyword` : the command keyword.
  - *Italic* : Words in italic indicate variable information that you supply.
  - `[options]` : optional parameters. Parameters that are not included in brackets are required.
  - `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
  - `...` : specifies to repeat the preceding content.
- In this topic, the letters used in expressions indicate topic-specific meanings:
  - C indicates the argc or range of parameters.
  - M indicates the number of bits set to 1 in a data structure, such as the number of nodes in a list or the number of fields in a hash.

time complexity

TR.SETBIT

Item	Description
Syntax	`TR.SETBIT` *key offset value*
Time complexity	O(1)
Command description	Sets the specified bit in a TairRoaring key to a value of 1 or 0 and returns the original bit value. The offset starts from 0.
Parameter	• Key: the key that you want to manage by running this command. • offset: the integer offset of the bit that you want to process. Valid values: 0 to 2^32. • value: the value that you want to set for the bit. The value can be set to 1 or 0.
Output	• If the operation is successful, a bit value of 0 or 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.SETBIT foo 0 1`
	Sample output:
	`(integer) 0`

TR.SETBITS

Item	Description
Syntax	`TR.SETBITS key offset [offset1 offset2 ... offsetN]`
Time complexity	O(C)
Command description	Sets the value of the specified bit in a TairRoaring key to 1. You can set multiple bits to a value of 1.
Parameter	• Key: the key that you want to manage by running this command. • offset: the integer offset of the bit that you want to process. Valid values: 0 to 2^32.
Output	• If the operation is successful, the number of bits in the key that have been set to 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.SETBITS foo 9 10`
	Sample output:
	`(integer) 5`

TR.CLEARBITS

Item	Description
Syntax	`TR.CLEARBITS key offset [offset1 offset2 ... offsetN]`
Time complexity	O(C)
Command description	Sets the value of the specified bit in a TairRoaring key to 0. If the specified bit already has a value of 0, the operation is not performed. You can set multiple bits to a value of 0.
Parameter	• Key: the key that you want to manage by running this command. • offset: the integer offset of the bit that you want to process. Valid values: 0 to 2^32.
Output	• If the operation is successful, the number of bits in the key that have been set to 0 is returned. • If the key does not exist, a value of 0 is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`TR.CLEARBITS foo 9 10`
	Sample output:
	`(integer) 2`

TR.SETRANGE

Item	Description
Syntax	`TR.SETRANGE key start end`
Time complexity	O(C)
Command description	Sets the bits within the specified range in a TairRoaring key to a value of 1. The range is a closed interval. For example, if you run the `TR.SETRANGE foo 1 3` command, the system creates the "0111" foo key.
Parameter	• Key: the key that you want to manage by running this command. • start: the start value of the interval. Valid values: 0 to 2 ^ 32. • end: the end value of the interval. Valid values: 0 to 2 ^ 32.
Output	• If the operation is successful, the number of bits in the key that have been set to 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.SETRANGE foo 1 3`
	Sample output:
	`(integer) 3`

TR.APPENDBITARRAY

Item	Description
Syntax	`TR.APPENDBITARRAY key offset bitarray`

Item	Description
Time complexity	O(C)
Command description	Inserts a bit array into a position right behind the specified bit in a TairRoaring key and overwrites the original data. The bit array consists of 0 and 1.
Parameter	• Key: the key that you want to manage by running this command. • offset: the offset of the specified bit. Valid values: -1 to 2 ^ 32. • bitarray: the bit array that you want to insert. The bit array consists of 0 and 1 and overwrites the original data. Valid values: 0 to 2 ^ 32. **Note** The specified bit and bit array that you specify cannot have a combined length longer than 2 ^ 32. Otherwise, the operation cannot be performed.
Output	• If the operation is successful, the number of bits in the key that have been set to 1 is returned. • Otherwise, an error message is returned.

Example	The `TR.SETBITS foo 0` command is run in advance.
	Sample command:
	`TR.APPENDBITARRAY foo 1 1101`
	Sample output:
	`(integer) 3`
	In this case, the TairRoaring foo key is " 101101 ".

TR.FLIPRANGE

Item	Description
Syntax	`TR.FLIPRANGE key start end`
Time complexity	O(C)
Command description	Changes the values for bits within the specified range in a TairRoaring key from 0 to 1 or from 1 to 0. The range is a closed interval. If the key does not exist, the key is created as an empty data set and the operation is performed on the key.

Item	Description
Parameter	• Key: the key that you want to manage by running this command. • start: the start value of the interval. Valid values: 0 to 2 ^ 32. • end: the end value of the interval. Valid values: 0 to 2 ^ 32.
Output	• If the operation is successful, the number of bits in the key that have been set to 1 is returned. • If the key does not exist, the key is created as an empty data set and the operation is performed on the key. If the operation is successful, the number of bits in the key that have been set to 1 is returned. • Otherwise, an error message is returned.

Example	The `TR.SETBITS foo 0 2 3 5` command is run in advance.
	Sample command:
	`TR.FLIPRANGE foo 0 5`
	Sample output:
	`(integer) 2`
	In this case, the TairRoaring foo key is "01001".

TR.APPENDINTARRAY

Item	Description
Syntax	`TR.APPENDINTARRAY key value [value1 value2 ... valueN]`
Time complexity	O(C)
Command description	Sets the value of the specified bit in a TairRoaring key to 1. You can set multiple bits to a value of 1. **Note** We recommend that you use TR.SETBITS in TairRoaring V2 instead of this command.
Parameter	• Key: the key that you want to manage by running this command. • value: the integer offset of the bit that you want to process. Valid values: 0 to 4294967296.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`TR.APPENDINTARRAY foo 9 10`
	Sample output:
	`OK`

TR.SETINTARRAY

Item	Description
Syntax	`TR.SETINTARRAY key value [value1 value2 ... valueN]`
Time complexity	O(C)
Command description	Creates a TairRoaring key from an integer array. If the key already exists, this command overwrites the data in the key. **Note** We recommend that you use TR.SETBITS in TairRoaring V2 instead of this command.
Parameter	• Key: the key that you want to manage by running this command. • value: the integer offset of the bit that you want to process.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.SETINTARRAY foo 2 4 5 6`
	Sample output:
	`OK`

TR.SETBITARRAY

Item	Description
Syntax	`TR.SETBITARRAY key value`

Item	Description
Time complexity	O(C)
Command description	Creates a TairRoaring key based on the specified bit array string. The bit array string consists of 0 and 1. If the key already exists, this command overwrites the data in the key. **Note** We recommend that you use TR.APPENDBITARRAY in TairRoaring V2 instead of this command.
Parameter	• key: the key that you want to manage by running this command. • value: the bit array string that consists of 0 and 1.
Output	• If the operation is successful, `OK` is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`tr.setbitarray foo 10101001`
	Sample output:
	`OK`

TR.BITOP

Item	Description
Syntax	`TR.BITOP` *destkey operation key* [*key1 key2 ... keyN*]
Time complexity	O(C * M)
Command description	Performs a bitwise operation on multiple TairRoaring keys and stores the result in the destination key. The *AND*, *OR*, *XOR*, *NOT*, and *DIFF* bitwise operations are supported. **Note** This command is unavailable for cluster instances.

Item	Description
Parameter	• key: the key that you want to manage by running this command. Multiple keys can be specified. • operation: the type of the bitwise operation. Valid values: *AND*, *OR*, *XOR*, *NOT*, and *DIFF*. **Note** ◦ *NOT* operations can be performed only on one key. ◦ *DIFF* operations calculate the difference between two keys only. Take note of the order of operations for calculating the difference. For example, the `TR.BITOP result DIFF key1 key2` command subtracts the key2 value from the key1 value. • destkey: the destination key that stores the result.
Output	• If the operation is successful, the integer number of bits in the result that have a value of 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.BITOP result OR foo bar`
	Sample output:
	`(integer) 6`

TR.BITOPCARD

Item	Description
Syntax	`TR.BITOPCARD operation key [key1 key2 ... keyN]`
Time complexity	O(C * M)
Command description	Performs a bitwise operation on multiple TairRoaring keys. The *AND*, *OR*, *XOR*, *NOT*, and *DIFF* bitwise operations are supported. **Note** This command is unavailable for cluster instances.

Item	Description
Parameter	• key: the key that you want to manage by running this command. Multiple keys can be specified. • operation: the type of the bitwise operation. Valid values: *AND*, *OR*, *XOR*, *NOT*, and *DIFF*. **Note** ◦ *NOT* operations can be performed only on one key. ◦ *DIFF* operations calculate the difference between two keys only. Take note of the order of operations for calculating the difference. For example, the `TR.BITOP result DIFF key1 key2` command subtracts the key2 value from the key1 value.
Output	• If the operation is successful, the integer number of bits in the result that have a value of 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.BITOPCARD NOT foo`
	Sample output:
	`(integer) 2`

TR.OPTIMIZE

Item	Description
Syntax	`TR.OPTIMIZE key`
Time complexity	O(M)
Command description	Optimizes the storage of a TairRoaring key. If the key is relatively large and is used mainly for read operations after the key is created, you can run this command.
Parameter	• Key: the key that you want to manage by running this command.
Output	• If the operation is successful, the number of bits in the key that have been set to 1 is returned. • If the key does not exist, `nil` is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`TR.OPTIMIZE foo`
	Sample output:
	`OK`

TR.GETBIT

Item	Description
Syntax	`TR.GETBIT key offset`
Time complexity	O(1)
Command description	Retrieves the value of the specified bit from a TairRoaring key.
Parameter	• Key: the key that you want to manage by running this command. • offset: the offset of the bit whose value you want to retrieve.
Output	• If the operation is successful, a value of 0 or 1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.GETBIT foo 0`
	Sample output:
	`(integer) 1`

TR.GETBITS

Item	Description
Syntax	`TR.GETBITS key offset [offset1 offset2 ... offsetN]`
Time complexity	O(C)

Item	Description
Command description	Retrieves the value of the specified bit from a TairRoaring key. You can specify multiple bits for value retrieval.
Parameter	• Key: the key that you want to manage by running this command. • offset: the offset of the bit whose value you want to retrieve.
Output	• If the operation is successful, the bit value is returned. • If the key does not exist, an empty array is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.GETBITS foo 3 4 6 8`
	Sample output:
	`1) (integer) 1`
	`2) (integer) 1`
	`3) (integer) 1`
	`4) (integer) 0`

TR.BITCOUNT

Item	Description
Syntax	`TR.BITCOUNT key [start end]`
Time complexity	O(M)
Command description	Counts the number of bits that have a value of 1 within the specified range in a TairRoaring key. The range is a closed interval.
Parameter	• Key: the key that you want to manage by running this command. • start: the start value of the interval. Valid values: 0 to 2 ^ 32. • end: the end value of the interval. Valid values: 0 to 2 ^ 32.
Output	• If the operation is successful, the integer number of bits that have a value of 1 within the specified range in the key is returned. • If the key does not exist, a value of 0 is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`TR.BITCOUNT foo 4 9`
	Sample output:
	`(integer) 3`

TR.BITPOS

Item	Description
Syntax	`TR.BITPOS <key> <value> [count]`
Time complexity	O(C)
Command description	Retrieves the offset of the bit that has an ordinal number of count. A bit can have a value of 1 or 0. The count parameter is optional and has a default value of 1. A value of 1 indicates the first bit retrieved by using the left-right counting approach.
Parameter	• Key: the key that you want to manage by running this command. • value: the value of the bit that you want to retrieve. The bit value can be 0 or 1. • count: the ordinal number of the bit. A negative number indicates that the right-left counting approach is used.
Output	• If the operation is successful, the offset of the bit that has a value of 1 or 0 is returned. • If the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.BITPOS foo 1 -1`
	Sample output:
	`(integer) 6`

TR.SCAN

Item	Description
Syntax	`TR.SCAN key start_offset [COUNT count]`

Item	Description
Time complexity	O(C)
Command description	Scans all bits located after a specified bit in a TairRoaring key and returns the offsets corresponding a count of the scanned bits that have a value of 1. The returned cursor is the offset corresponding to the key. **Note** This command may or may not scan and return added or deleted elements.
Parameter	• Key: the key that you want to manage by running this command. • start_offset: the offset of the bit. The bit that matches this offset is scanned. • COUNT: the number of bits that you want to query. Default value: 10.

Output	• If the operation is successful, an array that contains two elements is returned:
	  ◦ The first element is the next start_offset value. If the key is scanned, a value of 0 is returned.
	  ◦ The second element is the next offset that you specify.
	Note If the key does not exist, an array that consists of 0 and empty elements is returned.
	• Otherwise, an error message is returned.

Example	Sample command:
	`TR.SCAN foo 0 COUNT 2`
	Sample output:
	`1) (integer) 3`
	`2) 1) (integer) 0`
	`2) (integer) 2`

TR.RANGE

Item	Description
Syntax	`TR.RANGE key start end`
Time complexity	O(C)

Item	Description
Command description	Returns the offsets of the bits that have a value of 1 within the specified range in a TairRoaring key. The range is a closed interval.
Parameter	• Key: the key that you want to manage by running this command. • start: the start value of the interval. • end: the end value of the interval.
Output	• If the operation is successful, the offsets of the bits that have a value of 1 are returned. • If the key does not exist, an empty array is returned. • Otherwise, an error message is returned.

Example	The `TR.SETBITS foo 0 2 3 5` command is run in advance.
	Sample command:
	`TR.RANGE foo 0 5`
	Sample output:
	`1) (integer) 0`
	`2) (integer) 2`
	`3) (integer) 3`
	`4) (integer) 5`

TR.RANGEBITARRAY

Item	Description
Syntax	`TR.RANGEBITARRAY key start end`
Time complexity	O(C)
Command description	Retrieves a string that consists of bit values of 0 and 1 within the specified range in a TairRoaring key. The range is a closed interval.
Parameter	• Key: the key that you want to manage by running this command. • start: the start value of the interval. • end: the end value of the interval.
Output	• If the operation is successful, the offsets of the bits that have a value of 1 are returned. • If the key does not exist, `nil` is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `TR.SETBITS foo 0 2 3 5` command is run in advance.
	Sample command:
	`TR.RANGEBITARRAY foo 0 5`
	Sample output:
	`"101101"`

TR.MIN

Item	Description
Syntax	`TR.MIN key`
Time complexity	O(1)
Command description	Retrieves the offset of the first bit that has a value of 1 in a TairRoaring key. If no bits have a value of 1, a value of -1 is returned.
Parameter	• Key: the key that you want to manage by running this command.
Output	• ◦ If the operation is successful, the integer offset of the first bit that has a value of 1 is returned. ◦ If the operation is successful and no bits have a value of 1 or if the operation is successful and the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.MIN foo`
	Sample output:
	`4`

TR.MAX

Item	Description
Syntax	`TR.MAX key`

Item	Description
Time complexity	O(1)
Command description	Returns the offset of the last bit that has a value of 1 in a TairRoaring key. If no bits have a value of 1, -1 is returned.
Parameter	• Key: the key that you want to manage by running this command.
Output	• ◦ If the operation is successful, the integer offset of the last bit that has a value of 1 is returned. ◦ If the operation is successful and no bits have a value of 1 or if the operation is successful and the key does not exist, a value of -1 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.MAX foo`
	Sample output:
	`6`

TR.STAT

Item	Description
Syntax	`TR.STAT key [JSON]`
Time complexity	O(M)
Command description	Returns the statistical information of the specified TairRoaring key. The information includes the number of containers and the memory usage.
Parameter	• Key: the key that you want to manage by running this command. • JSON: If this parameter is specified, the statistical information is returned in the JSON format.

Item	Description

Output	• If the operation is successful, the following output is returned:
	`"{\"cardinality\":3, # Total number of elements in the key`
	`"number_of_containers":1, # Total number of containers in the key`
	`"max_value":6, # Maximum element value in the key`
	`\"min_value\":3, # Minimum element value in the key`
	`\"sum_value\":13,`
	`\"array_container\" :{# Number of array containers in the key`
	`\"number_of_containers\":1,`
	`\"container_cardinality\":3,`
	`\"container_allocated_bytes\":6},`
	`\"bitset_container\" :{# Number of bitset containers in the key`
	`\"number_of_containers\":0,`
	`\"container_cardinality\":0,`
	`\"container_allocated_bytes\":0},`
	`\"run_container\" :{# Number of run containers in the key`
	`\"number_of_containers\":0,`
	`\"container_cardinality\":0,`
	`\"container_allocated_bytes\":0}}"`
	• If the key does not exist, `nil` is returned.
	• Otherwise, an error message is returned.

Example	Sample command:
	`TR.STAT foo JSON`
	Sample output:
	`"{\"cardinality\":4,\"number_of_containers\":1,\"max_value\":5,\"min_value\":0,\"sum_value\":10,\"array_container\":{\"number_of_containers\":1,\"container_cardinality\":4,\"container_allocated_bytes\":8},\"bitset_container\":{\"number_of_containers\":0,\"container_cardinality\":0,\"container_allocated_bytes\":0},\"run_container\":{\"number_of_containers\":0,\"container_cardinality\":0,\"container_allocated_bytes\":0}}"`

TR.JACCARD

Item	Description
Syntax	`TR.JACCARD` *key1 key2*

Item	Description
Time complexity	O(M)
Command description	Retrieves the Jaccard similarity coefficient of two TairRoaring keys. The higher the coefficient, the higher the similarity. **Note** This command is unavailable for cluster instances.
Parameter	• Key: the key that you want to manage by running this command.
Output	• If the operation is successful, a double-typed Jaccard similarity coefficient is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TR.JACCARD foo1 foo2`
	Sample output:
	`"0.20000000000000001"`

TR.CONTAINS

Item	Description
Syntax	`TR.CONTAINS key1 key2`
Time complexity	O(M)
Command description	Checks whether key2 contains key1. If yes, key1 is a subset of key2 and a value of 1 is returned. Otherwise, key1 is not a subset of key2 and a value of 0 is returned. **Note** This command is unavailable for cluster instances.
Parameter	• Key: the key that you want to manage by running this command.
Output	• ◦ If key2 contains key1, a value of 1 is returned. ◦ If key2 does not contain key1, a value of 0 is returned. • Otherwise, an error message is returned.

Item	Description

Example	The `TR.SETBITS fooM 1 2 3 10` and `TR.SETBITS foom 1 2` commands are run in advance.
	Sample command:
	`TR.CONTAINS foom fooM`
	Sample output:
	`(integer) 1`

TR.RANK

Item	Description
Syntax	`TR.RANK key offset`
Time complexity	O(M)
Command description	Retrieves the number of bits that have a value of 1 within the range from the first bit and the specified bit. The range is a closed interval.
Parameter	• Key: the key that you want to manage by running this command. • offset: the offset of the specified bit. The value must be an integer.
Output	• If the operation is successful, the number of bits that have a value of 1 is returned. • Otherwise, an error message is returned.

Example	The `TR.SETBITS fooM 1 2 3 10` command is run in advance.
	Sample command:
	`TR.RANK fooM 10`
	Sample output:
	`(integer) 4`

Error messages

Error message	Description
`WRONGTYPE Operation against a key holding the wrong kind of value`	The key type is incorrect: The key is not a TairRoaring key.
`ERR bad arguments, must be unsigned 32-bit integer`	The parameter type is incorrect: The parameter values cannot be converted into 32-bit integers.
`ERR invalid arguments, maybe out of range or illegal`	The parameters are invalid: • The offset is not a 32-bit integer. • The start and end offsets are invalid. • The parameter value exceeds the maximum number of elements allowed for a TairRoaring key.
`ERR key already exist`	The TairRoaring key already exists, and its data cannot be overwritten. **Note** This error is fixed in TairRoaring V2.2.
`ERR key not found`	The TairRoaring key does not exist. **Note** This error is fixed in TairRoaring V2.2.

3.6.12. TairSearch

TairSearch is a full-text search data structure developed in-house based on Redis modules instead of Lucene or other open source search engine software libraries. TairSearch uses query syntax that is similar to that of Elasticsearch.

Overview

Compared with Elasticsearch, TairSearch has the following advantages:

- All data and indexes are stored in memory to ensure higher throughput and lower read/write latency.
- Update of local indexes for documents is supported. Update operations include adding a field, updating a field, deleting a field, and auto-incrementing a field.
- Real-time indexes are supported. Data is immediately available after it is written.

Prerequisites

The instance is a performance-enhanced instance of the ApsaraDB for Redis Enhanced Edition (Tair) that runs the minor version of 1.7.27 or later. For more information about performance-enhanced instances, see Performance-enhanced instances.

> **Note** We recommend that you update your instance to the latest minor version for more features and higher stability. For more information, see Update the minor version. If your instance is a cluster or read/write splitting instance, we recommend that you update the proxy nodes in the instance to the latest minor version. This ensures that all commands can be run as expected. For more information about cluster and read/write splitting instances, see Cluster master-replica instances and Read/write splitting instances.

Precautions

- The TairSearch data that you want to manage is stored on the performance-enhanced instance.
- To reduce memory usage, we recommend that you perform the following operations:
  - When you create indexes, set index to true for document fields to specify these fields as indexed fields. For other document fields, set index to false.
  - Specify the includes and excludes properties of the _source field to filter out document fields that you do not need and save fields that you need.
- Do not add a large number of documents to a single index. Add the documents to multiple indexes. We recommend that you keep the number of documents per index within 500 million to prevent data skew in cluster instances, balance read and write traffic, and reduce the number of large keys and hotkeys.

Supported commands

Full-text search commands

Command	Syntax	Description
TFT.CREATEINDEX	`TFT.CREATEINDEX index mappings`	Creates an index and a mapping for the index. The syntax used to create a mapping is similar to that used to create an explicit mapping in Elasticsearch. For more information, see Explicit mapping. You must create an index before you can add documents to the index.
TFT.UPDATEINDEX	`TFT.UPDATEINDEX index mappings`	Adds a properties field to the specified index. The field that you want to add cannot have the same name as an existing one. Otherwise, the field cannot be added.
TFT.GETINDEX	`TFT.GETINDEX index`	Obtains the mapping content of an index.
TFT.ADDDOC	`TFT.ADDDOC index document [WITH_ID doc_id]`	Adds a document to an index. You can specify a unique ID for the document in the index by using WITH_ID doc_id. If the document ID already exists, the existing ID is overwritten. If you do not specify WITH_ID doc_id, a document ID is automatically generated. By default, WITH_ID doc_id is not specified.
TFT.MADDDOC	`TFT.MADDDOC index document doc_id [document1 doc_id1] ...`	Adds multiple documents to an index. Each document must have a document ID that is specified by doc_id.

Command	Syntax	Description
TFT.UPDATEDOCFIELD	`TFT.UPDATEDOCFIELD index doc_id document`	Updates the document specified by doc_id in an index. If the fields that you want to add to the document are mapped and indexed fields, the fields must be of the same data type as those used by the mapping. If the fields that you want to add are not indexed fields, the fields can be of a random data type. **Note** If the fields that you want to add to the document already exist, the document is updated. If the fields that you want to add do not exist, the fields are added. If the document does not exist, the document is automatically created. In this case, the command is equivalent to TFT.ADDDOC.
TFT.DELDOCFIELD	`TFT.DELDOCFIELD index doc_id field [field1 field2 ...]`	Deletes the specified fields from the document specified by doc_id in an index. If the fields are indexed fields, the information in the fields is also deleted from the index. **Note** If the fields do not exist, the operation cannot be performed. For example, the fields may not exist because they are filtered out by using _source.
TFT.INCRLONGDOCFIELD	`TFT.INCRLONGDOCFIELD index doc_id field increment`	Adds an increment to the specified field in the document specified by doc_id in an index. The increment can be a positive or negative integer. The data type of the field can only be long or int. **Note** If the document does not exist, the document is automatically created. In this case, the existing value of the field is 0, and the updated field value is obtained by adding the increment value to the existing field value. If the field does not exist, the operation cannot be performed. For example, the field may not exist because it is filtered out by using _source.

Command	Syntax	Description
TFT.INCRFLOATDOCFIELD	`TFT.INCRFLOATDOCFIELD index doc_id field increment`	Adds an increment to the specified field in the document specified by doc_id in an index. The increment can be a positive or negative floating-point number. The data type of the field can be int, long, or double. **Note** If the document does not exist, the document is automatically created. In this case, the existing value of the field is 0, and the updated field value is obtained by adding the increment value to the existing field value. If the field does not exist, the operation cannot be performed. For example, the field may not exist because it is filtered out by using _source.
TFT.GETDOC	`TFT.GETDOC index doc_id`	Obtains the content of the document specified by doc_id in an index.
TFT.EXISTS	`TFT.EXISTS index doc_id`	Checks whether the document specified by doc_id exists in an index.
TFT.DOCNUM	`TFT.DOCNUM index`	Obtains the number of documents in an index.
TFT.SCANDOCID	`TFT.SCANDOCID index cursor [MATCH *value*] [COUNT count]`	Obtains the IDs of all documents in an index.
TFT.DELDOC	`TFT.DELDOC index doc_id [doc_id] ...`	Deletes the document specified by doc_id from an index. Multiple document IDs can be specified.
TFT.DELALL	`TFT.DELALL index`	Deletes all documents from an index but retains the index.
TFT.SEARCH	`TFT.SEARCH index query`	Queries the documents in an index. The query syntax is similar to that of the Query Domain Specific Language (DSL) of Elasticsearch. For more information, see Query DSL.
DEL	`DEL key [key ...]`	Deletes one or more TairSearch keys.

Auto-complete commands

Command	Syntax	Description

Command	Syntax	Description
TFT.ADDSUG	`TFT.ADDSUG index text weight [text weight] ...`	Adds one or more auto-complete texts and their weights to the specified index.
TFT.DELSUG	`TFT.DELSUG index text [text] ...`	Deletes one or more auto-complete texts from the specified index.
TFT.SUGNUM	`TFT.SUGNUM index`	Obtains the number of auto-complete texts in the specified index.
TFT.GETSUG	`TFT.GETSUG index prefix [MAX_COUNT count] [FUZZY]`	Obtains the auto-complete texts that can be matched based on the specified prefix. Texts are returned in descending order of weights.
TFT.GETALLSUGS	`TFT.GETALLSUGS index`	Obtains the full auto-complete texts in the specified index.

Note The following section describes command syntax used in this topic:

- `Uppercase keyword` : the command keyword.
- *Italic* : Words in italic indicate variable information that you supply.
- `[options]` : optional parameters. Parameters that are not included in brackets are required.
- `AB` : specifies that these parameters are mutually exclusive. Select one of two or more parameters.
- `...` : specifies to repeat the preceding content.

TFT.CREATEINDEX

Item	Description
Syntax	`TFT.CREATEINDEX index mappings`
Command description	Creates an index and a mapping for the index. The syntax used to create a mapping is similar to that used to create an explicit mapping in Elasticsearch. For more information, see Explicit mapping. You must create an index before you can add documents to the index.

	• index: the name of the index that you want to manage.
	• mappings: the mapping content. The following syntax is supported:
	  ◦ dynamic: the mapping mode. Valid values:
	  ▪ strict: the strict mapping. Only fields that have been specified in the properties field can be mapped. Only mapped fields in a document can be written. If you do not specify the mapping mode, non-strict mapping is used.

Item	Description
Parameter	○ enabled: specifies to forcefully check whether the field types in a document to be written are the same as those specified in the properties field. If they are not the same, the document cannot be written. Valid values: true and false. The default value is true, which indicates that a forceful check is performed. This parameter takes effect only for non-indexed fields (index set to true). For indexed fields, a forceful check is performed. ○ _source: the source of the original document. This parameter does not affect the creation of field indexes. Valid values: ■ enabled: specifies whether to store the original document. Valid values: ■ true: stores the original document. By default, enabled is set to true and all fields in the document are stored. You can set excludes to specify the fields that you do not want to store in the original document and includes to specify fields that you want to store. Wildcards are accepted. If a field is specified by both excludes and includes, excludes takes precedence over includes and the field is not stored in the original document. Examples: Example 1: `"_source":{"enabled":true}` specifies to store all fields and works like `"_source":{"enabled":true,"excludes":[],"includes":[]}`. Example 2: `"_source":{"enabled":true,"excludes":[],"includes":["id","name"]}` specifies to store only id and name fields. ■ false: does not store the original document. ○ properties: a collection of mapped fields. You can set the following properties for each field: ■ index: specifies whether to add the field to the index. Valid values: true and false. Default value: true. ■ boost: the weight of the field. Default value: 1. The parameter value must be a positive floating-point number. Tair automatically calculates the estimated relative score of each field. ■ type: the data type of the field. Valid values: ■ keyword: a string that cannot be split. The following parameter can be specified for the string: ■ ignore_above: the maximum number of characters that can be contained in the keyword string. Default value: 128. Example: `"ignore_above":128`. ■ text: a string that can be parsed by a tokenizer and then stored in an index. The following parameters can be specified for the string: ■ analyzer: the tokenizer that parses the text value and then stores the value in the index. Valid values: standard, chinese, arabic, cjk, czech, brazilian, german, greek, persian, french, dutch, russian, and jieba. The default value is standard, which indicates an English tokenizer. If you want a Chinese tokenizer, we recommend that you use jieba because it performs better than chinese. For example, `"analyzer":"jieba"` specifies to use the jieba tokenizer. ■ search_analyzer: the tokenizer that is used in the search. The supported tokenizer types are the same as those supported by the analyzer parameter. The default value of this parameter is the same as that of the analyzer parameter. ■ long: the long data type. You can convert a point in time into a UNIX timestamp and store the timestamp as a piece of long-typed data. ■ integer: the int data type. ■ double: the double data type.

Item	Description
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.CREATEINDEX idx:product '{"mappings":{"_source":{"enabled":true},"properties":{"product_id":{"type":"keyword","ignore_above":128},"product_name":{"type":"text"},"product_title":{"type":"text","analyzer":"jieba"},"price":{"type":"double"}}}}'`
	`# Create a product ID (product_id) of the keyword type and set the length limit of the ID to 128 characters.`
	`# Create a product name (product_name) of the text type.`
	`# Create a product title (product_title) of the text type and set analyzer to jieba.`
	`# Create a price (price) of the double type.`
	Sample output:
	`OK`

TFT.UPDATEINDEX

Item	Description
Syntax	`TFT.UPDATEINDEX index mappings`
Command description	Adds a properties field to the specified index. The field that you want to add cannot have the same name as an existing one. Otherwise, the field cannot be added.
Parameter	• index: the name of the index that you want to manage. • mappings: the mapping content. You do not need to specify parameters such as dynamic and _source. You can specify only the properties field that you want to add. For more information about the syntax, see TFT.CREATEINDEX.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`TFT.UPDATEINDEX idx:product '{"mappings":{"properties":{"product_group":{"type":"text","analyzer":"chinese"}}}}'`
	Sample output:
	`OK`

TFT.GETINDEX

Item	Description
Syntax	`TFT.GETINDEX index`
Command description	Obtains the mapping content of an index.
Parameter	• index: the name of the index that you want to manage.
Output	• If the operation is successful, the JSON-typed mapping content of the index is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.GETINDEX idx:product`
	Sample output:
	`{"idx:product0310":{"mappings":{"_source":{"enabled":true,"excludes":[],"includes":["product_id"]},"dynamic":"false","properties":{"price":{"boost":1.0,"enabled":true,"ignore_above":-1,"index":true,"similarity":"classic","type":"double"},"product_id":{"boost":1.0,"enabled":true,"ignore_above":128,"index":true,"similarity":"classic","type":"keyword"},"product_name":{"boost":1.0,"enabled":true,"ignore_above":-1,"index":true,"similarity":"classic","type":"text"},"product_title":{"analyzer":"chinese","boost":1.0,"enabled":true,"ignore_above":-1,"index":true,"similarity":"classic","type":"text"}}}}}`

TFT.ADDDOC

Item	Description
Syntax	`TFT.ADDDOC index document [WITH_ID doc_id]`
Command description	Adds a document to an index. You can specify a unique ID for the document in the index by using WITH_ID doc_id. If the document ID already exists, the existing ID is overwritten. If you do not specify WITH_ID doc_id, a document ID is automatically generated. By default, WITH_ID doc_id is not specified.
Parameter	• index: the name of the index that you want to manage. • document: the JSON-typed document that you want to add to the index. **Note** If you use _source and includes to specify to retain only some fields, only the fields specified by includes are retained when you add a document to the index or update a document added to the index. • WITH_ID doc_id: specifies whether to configure the ID of the document. If yes, you must also enter a string as the doc_id value.
Output	• If the operation is successful, the JSON-typed document ID is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.ADDDOC idx:product '{"product_id":"product test"}' WITH_ID 00001`
	Sample output:
	`"{\"_id\":\"00001\"}"`

TFT.MADDDOC

Item	Description
Syntax	`TFT.MADDDOC index document doc_id [document1 doc_id1] ...`
Command description	Adds multiple documents to an index. Each document must have a document ID that is specified by doc_id.

Item	Description
Parameter	• index: the name of the index that you want to manage. • document: the JSON-typed document that you want to add to the index. **Note** If you use _source and includes to specify to retain only some fields, only the fields specified by includes are retained when you add a document to the index or update a document added to the index. • doc_id: the ID of the document. The value is a string.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.MADDDOC idx:product '{"product_id":"test1"}' 00011 '{"product_id":"test2"}' 00012`
	Sample output:
	`OK`

TFT.UPDATEDOCFIELD

Item	Description
Syntax	`TFT.UPDATEDOCFIELD index doc_id document`
Command description	Updates the document specified by doc_id in an index. If the fields that you want to add to the document are mapped and indexed fields, the fields must be of the same data type as those used by the mapping. If the fields that you want to add are not indexed fields, the fields can be of a random data type. **Note** If the fields that you want to add to the document already exist, the document is updated. If the fields that you want to add do not exist, the fields are added. If the document does not exist, the document is automatically created. In this case, the command is equivalent to TFT.ADDDOC.

Item	Description
Parameter	• index: the name of the index that you want to manage. • doc_id: the ID of the document. • document: the JSON-typed content that is used to update the document. **Note** If you use _source and includes to specify to retain only some fields, only the fields specified by includes are retained when you add a document to the index or update a document added to the index.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.UPDATEDOCFIELD idx:product 00011 '{"product_id":"test8","product_group":"BOOK"}'`
	Sample output:
	`OK`

TFT.DELDOCFIELD

Item	Description
Syntax	`TFT.DELDOCFIELD index doc_id field [field1 field2 ...]`
Command description	Deletes the specified fields from the document specified by doc_id in an index. If the fields are indexed fields, the information in the fields is also deleted from the index. **Note** If the fields do not exist, the operation cannot be performed. For example, the fields may not exist because they are filtered out by using _source.
Parameter	• index: the name of the index that you want to manage. • doc_id: the ID of the document. • field: the field that you want to delete.
Output	• If the operation is successful, the number of deleted fields is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`TFT.DELDOCFIELD idx:product 00011 product_group`
	Sample output:
	`(integer) 1`

TFT.INCRLONGDOCFIELD

Item	Description
Syntax	`TFT.INCRLONGDOCFIELD` *index doc_id field increment*
Command description	Adds an increment to the specified field in the document specified by doc_id in an index. The increment can be a positive or negative integer. The data type of the field can only be long or int. **Note** If the document does not exist, the document is automatically created. In this case, the existing value of the field is 0, and the updated field value is obtained by adding the increment value to the existing field value. If the field does not exist, the operation cannot be performed. For example, the field may not exist because it is filtered out by using _source.
Parameter	• index: the name of the index that you want to manage. • doc_id: the ID of the document. • field: the field to which you want to add an increment. The data type of the field can only be long or int. • increment: the increment value that you want to add to the field. The value must be an integer. You can specify a negative integer. In this case, the updated field value is obtained by subtracting the increment value from the existing field value.
Output	• If the operation is successful, the updated field value is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.INCRLONGDOCFIELD idx:product 00011 stock 100`
	Sample output:
	`(integer)100`

TFT.INCRFLOATDOCFIELD

Item	Description
Syntax	`TFT.INCRFLOATDOCFIELD index doc_id field increment`
Command description	Adds an increment to the specified field in the document specified by doc_id in an index. The increment can be a positive or negative floating-point number. The data type of the field can be int, long, or double. **Note** If the document does not exist, the document is automatically created. In this case, the existing value of the field is 0, and the updated field value is obtained by adding the increment value to the existing field value. If the field does not exist, the operation cannot be performed. For example, the field may not exist because it is filtered out by using _source.
Parameter	• index: the name of the index that you want to manage. • doc_id: the ID of the document. • field: the field to which you want to add an increment. The data type of the field can be long, int, or double. • increment: the increment value that you want to add to the field. The value can be a positive or negative floating-point number. If the value is a negative one, the updated field value is obtained by subtracting the increment value from the existing field value.
Output	• If the operation is successful, the updated field value is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.INCRFLOATDOCFIELD idx:product 00011 stock 299.6`
	Sample output:
	`"299.6"`

TFT.GETDOC

Item	Description
Syntax	`TFT.GETDOC index doc_id`
Command description	Obtains the content of the document specified by doc_id in an index.
Parameter	• index: the name of the index that you want to manage. • doc_id: the ID of the document.

Item	Description
Output	• If the operation is successful, the ID and content of the document are returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.GETDOC idx:product 00011`
	Sample output:
	`"{\"_id\":\"00011\",\"_source\":{\"product_id\":\"test8\"}}"`

TFT.EXISTS

Item	Description
Syntax	`TFT.EXISTS index doc_id`
Command description	Checks whether the document specified by doc_id exists in an index.
Parameter	• index: the name of the index that you want to manage. • doc_id: the ID of the document.
Output	• ◦ If the operation is successful and the document exists, a value of 1 is returned. ◦ If the operation is successful and the field does not exist or if the operation is successful and the document does not exist, a value of 0 is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.EXISTS idx:product 00011`
	Sample output:
	`(integer) 1`

TFT.DOCNUM

Item	Description
Syntax	`TFT.DOCNUM index`

Item	Description
Command description	Obtains the number of documents in an index.
Parameter	• index: the name of the index that you want to manage.
Output	• If the operation is successful, the number of documents in the index is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.DOCNUM idx:product`
	Sample output:
	`(integer) 3`

TFT.SCANDOCID

Item	Description
Syntax	`TFT.SCANDOCID index cursor [MATCH *value*] [COUNT count]`
Command description	Obtains the IDs of all documents in an index.
Parameter	• index: the name of the index that you want to manage. • cursor: the cursor used in this scan. • MATCH *value*: the matching pattern. value specifies the value that you use for matching. Example: `MATCH *redis*` . • COUNT count: the maximum number of items that can be scanned at a time. Default value: 100.

Output	• If the operation is successful, an array is returned.
	  ◦ The first element of the array is the cursor used for the next scan. If the index is scanned, a value of 0 is returned.
	  ◦ The second element of the array is the IDs of the documents that are obtained in this scan.
	• Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`TFT.SCANDOCID idx:product 0 COUNT 3`
	Sample output:
	`1) "0"`
	`2) 1) "00001"`
	`2) "00011"`
	`3) "00012"`

TFT.DELDOC

Item	Description
Syntax	`TFT.DELDOC index doc_id [doc_id] ...`
Command description	Deletes the document specified by doc_id from an index. Multiple document IDs can be specified.
Parameter	• index: the name of the index that you want to manage. • doc_id: the ID of the document.
Output	• If the operation is successful, the number of deleted documents is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.DELDOC idx:product 00011 00014`
	Sample output:
	`"1"  # In this example, the document with an ID of 00014 does not exist. In this case, the 00011 document is deleted only and a value of 1 is returned.`

TFT.DELALL

Item	Description
Syntax	`TFT.DELALL index`
Command description	Deletes all documents from an index but retains the index.

Item	Description
Parameter	• index: the name of the index that you want to manage. • doc_id: the ID of the document.
Output	• If the operation is successful, OK is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.DELALL idx:product`
	Sample output:
	`OK`

TFT.SEARCH

Item	Description
Syntax	`TFT.SEARCH index query`
Command description	Queries the documents in an index. The query syntax is similar to that of the Query Domain Specific Language (DSL) of Elasticsearch. For more information, see Query DSL.
	• index: the name of the index that you want to manage. • query: the query statement. The following fields are supported: ◦ query: the query clause. The following syntax is supported:

Item	Description

- match: the basic match query. The following parameters are supported:

  In scenarios that do not require document characters to be broken into individual tokens or do not require fuzzy match, we recommend that you use commands such as term and prefix to improve query efficiency.

  - query: the documents in the index obtained by using the query.

    > **Note**
    > - If fuzzy match is disabled by using fuzziness, documents obtained by the query are broken into individual tokens by a tokenizer specified by analyzer. You can also specify the relationship between tokens by using operator and the minimum number of tokens that need to be matched by using minimum_should_match.
    > - If fuzzy match is enabled by using fuzziness, the preceding parameters are invalid. In this case, you can specify prefix_length.

  - analyzer: the tokenizer used by the match statement. Valid values: standard, chinese, arabic, cjk, czech, brazilian, german, greek, persian, french, dutch, russian, and jieba. The default value is standard, which indicates an English tokenizer. To parse Chinese characters, you can use jieba.
  - minimum_should_match: the minimum number of tokens that need to be matched. The tokenizer breaks the query statement into tokens. This parameter takes effect when fuzzy match is disabled and operator is set to OR.
  - operator: the relationship between tokens. Valid values: AND and OR. Default value: OR. For example, `'{"query":{"match":{"f0":{"query":"redis cluster","operator":"OR"}}}}'` specifies that the search condition is met when "redis" or "cluster" is contained in the documents for which you want to perform a query.
  - fuzziness: specifies whether to enable fuzzy match for the query. By default, fuzzy match is disabled. `"fuzziness":1` specifies that fuzzy match is enabled. Example: `'{"query":{"match":{"f0":{"query":"excelentl","fuzziness":1}}}}'`.
  - prefix_length: the number of start characters that must be retained for a fuzzy match. Default value: 0. This parameter is valid only when fuzzy match is enabled.

- term: the token query. If this parameter is specified, the query statement is not split. This query type is more efficient than the `match` query. The token query supports all field types. Examples: `'{"query":{"term":{"f0":{"value":"redis","boost": 2.0}}}}'` and `'{"query":{"term":{"f0":"redis"}}}'`.
- terms: the multi-token query. Example: `'{"query":{"terms":{"f0":["redis","excellent"]}}}'`.
- range: the range query. This parameter supports all field types. Valid values: `lt` (less than), `gt` (greater than), `lte` (less than or equal to), and `gte` (greater than or equal to). Example: `'{"query":{"range":{"f0":{"lt":"abcd","gt":"ab"}}}}'`.
- wildcard: the wildcard query. Format: `"wildcard":{"field to query":{"value":"query content"}}`. Supported wildcards: `*` and `?`. Example: `'{"query":{"wildcard":{"f0":{"value":"*b?Se"}}}}'`.
- prefix: the prefix query. For example, `'{"query":{"prefix":{"f0":"abcd"}}'` specifies to query documents whose f0 fields start with `abcd`.

Item	Description
Parameter	■ bool: the compound query. Valid values: ■ must: the must list. If bool is set to must, the query results must match all query clauses in this list. ■ must_not: the must_not list. If bool is set to must_not, the query results do not need to match all query clauses in this list. ■ should: the should list. If bool is set to should, the query results must match one query clause in this list. You can use minimum_should_match to specify the minimum number of query clauses that must be matched. For example, `'{"query":{"bool":{"must":[{"match":{"DB":"redis"}},{"match":{"Architectures":"cluster"}}]}}}'` specifies that the type of the database that you want to query is Redis and the architecture of the instance that contains the database is cluster. ■ dis_max: the compound query for best match. By default, this parameter returns documents that match one or more query clauses. If a returned document matches multiple query clauses, the dis_max query assigns the document the highest score from the matching clauses and a tie-breaking increment to the document score if tie_breaker is specified. Valid values of tie_breaker: 0 to 1. Default value: 0. After tie_breaker is specified, the document score can be calculated based on the following formula: Score from a matching clause that has the highest score + (Scores from other matching clauses × tie_breaker value). Assume that doc1 matches three query clauses whose document scores are 5, 1, and 3 and a tie_break value of 0.5 is specified. In this case, the score of doc1 can be calculated by using this formula: Score of doc1 = 5 + [(1 + 3) × 0.5]. Example: `'{"query":{"dis_max":{"queries":[{"term":{"f0":"redis"}},{"term":{"f1":"database"}},{"match":{"title":"redis memory database"}}],"tie_breaker": 0.5}}}'` . ■ constant_score: the compound query that returns a constant score. This query wraps another query and returns a constant score equal to the query boost for every document in the filter. The boost value is of the double type, and the default boost value is 1.0. Example: `'{"query":{"constant_score":{"filter":{"term":{"f0":"abc"}},"boost":1.2}}}'` . ■ match_all: the match all query. This query matches all documents and gives them all a boost value. The boost value is of the double type, and the default boost value is 1.0. Example: `'{"query":{"match_all":{"boost":2.3}}}'` . **Note** If you perform a compound query by using bool, dis_max, and constant_score, you can set boost values for term, terms, range, and wildcard. The default boost value is 1. You must specify the parameter to which the boost value is applied in the query syntax. Example: `'{"query":{"bool":{"must":[{"match":{"f0":"v0"}},"term":{"f0":{"value":"redis","boost":2}}]}}}'` .

Item	Description
	○ sort: specifies to sort query results by a factor. Default value: _score. Valid values: ■ _score: sorts query results by weight in descending order. Example: `"sort":"_score"`. ■ _doc: sorts query results by document ID in ascending order. Example: `"sort":"_doc"`. ■ Specified field: sorts query results by specified field in ascending order. The field must be an indexed field (index set to true) or a field that is forcefully checked (enabled set to true). The field must be of the keyword, long, integer, or double type. For example, `"sort":"price"` specifies to sort query results by price in ascending order. You can also use an array to sort query results. In this case, query results are sorted by array order. You can use order to modify the default array order. Valid values of order: desc and acs. For example, `'{"sort":[{"price":{"order":"desc"}},{"_doc":{"order":"desc"}}]}'` specifies to sort query results by price or by document ID if the prices are the same. ○ _source: the fields displayed in query results. You can use includes to specify fields that you want to retain and use excludes to specify fields that you do not want to retain. Wildcards are accepted. For example, `"_source":{"includes":["f0"]` specifies to return only the f0 fields in query results. ○ track_total_hits: specifies whether to query all documents when you use the term, terms, or match query. In this case, compound query is supported. If you use the match query, fuzzy match is disabled. Default value: false. Valid values: ■ false: Only the top 100 documents are queried. Documents are sorted by score. ■ true: All documents are queried. **Warning** If track_total_hits is set to true and the number of documents to query is large, slow queries can be observed. Proceed with caution. ○ from: the start document to return. The default value is 0, which specifies to return the first document and all subsequent documents that are queried. ○ size: the number of documents that can be returned. Default value: 10. Valid values: 0 to 10000. **Note** A combination of from and size works like paging. However, query performance degrades as the number of discrete pages increases.

Item	Description
Output	• If the operation is successful, the documents in the index are returned. **Note** In the following sample output, relation indicates the number of returned documents. Valid values of relation: eq (equal to the value specified by value), gte (greater than or equal to the value specified by value). `{ "hits": { "hits": [], "max_score": null, "total": { "relation": "gte", "value": 100 } } }` • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.SEARCH idx:product '{"sort":[{"price":{"order":"desc"}}]}'`
	Sample output:
	`'{"hits":{"hits": [{"_id":"fruits_3","_index":"idx:product","_score":1.0,"_source": {"product_id":"fruits_3","product_name":"orange","price":30.2,"stock":3000 }},{"_id":"fruits_2","_index":"idx:product","_score":1.0,"_source": {"product_id":"fruits_2","product_name":"banana","price":24.0,"stock":2000 }},{"_id":"fruits_1","_index":"idx:product","_score":1.0,"_source": {"product_id":"fruits_1","product_name":"apple","price":19.5,"stock":1000} }],"max_score":1.0,"total":{"relation":"eq","value":3}}}'`

TFT.ADDSUG

Item	Description
Syntax	`TFT.ADDSUG index text weight [text weight] ...`
Command description	Adds one or more auto-complete texts and their weights to the specified index.

Item	Description
Parameter	• index: the name of the index that you want to manage. • text: the auto-complete text that you want to add to the index. • weight: the weight of the text. The weight must be a positive integer.
Output	• If the operation is successful, the number of added texts is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.ADDSUG idx:redis 'redis is a memory database' 3 'redis cluster' 10`
	Sample output:
	`(integer) 2`

TFT.DELSUG

Item	Description
Syntax	`TFT.DELSUG index text [text] ...`
Command description	Deletes one or more auto-complete texts from the specified index.
Parameter	• index: the name of the index that you want to manage. • text: the text that you want to delete from the index. The text value must be complete and accurate.
Output	• If the operation is successful, the number of deleted texts is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.DELSUG idx:redis 'redis is a memory database' 'redis cluster'`
	Sample output:
	`(integer) 2`

TFT.SUGNUM

Item	Description
Syntax	`TFT.SUGNUM index`
Command description	Obtains the number of auto-complete texts in the specified index.
Parameter	• index: the name of the index that you want to manage.
Output	• If the operation is successful, the number of texts in the index is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.SUGNUM idx:redis`
	Sample output:
	`(integer) 3`

TFT.GETSUG

Item	Description
Syntax	`TFT.GETSUG index prefix [MAX_COUNT count] [FUZZY]`
Command description	Obtains the auto-complete texts that can be matched based on the specified prefix. Texts are returned in descending order of weights.
Parameter	• index: the name of the index that you want to manage. • prefix: the prefix that you specify. • MAX_COUNT count: the maximum number of texts that can be returned. Valid values: 0 to 255. • FUZZY: specifies whether to enable fuzzy match.
Output	• If the operation is successful, the number of texts in the index is returned. • Otherwise, an error message is returned.

Item	Description

Example	Sample command:
	`TFT.GETSUG idx:redis res MAX_COUNT 2 FUZZY`
	Sample output:
	`1) "redis cluster"`
	`2) "redis lock"`

TFT.GETALLSUGS

Item	Description
Syntax	`TFT.GETALLSUGS` *index*
Command description	Obtains the full auto-complete texts in the specified index.
Parameter	• index: the name of the index that you want to manage by running this command.
Output	• If the operation is successful, a list of auto-complete texts is returned. • Otherwise, an error message is returned.

Example	Sample command:
	`TFT.GETALLSUGS idx:redis`
	Sample output:
	`1) "redis cluster"`
	`2) "redis lock"`
	`3) "redis is a memory database"`

3.7. Imperceptible scaling

This topic lists the drawbacks of the scaling solutions available for open source Redis clusters and ApsaraDB for Redis cluster instances and describes the imperceptible scaling solution available for ApsaraDB for Redis Enhanced Edition (Tair) cluster instances.

Requirements for elastic scaling of data nodes and migration of data between shards are often involved with open source Redis clusters. However, these common scaling solutions have issues such as slow key migration, unavailability of commands that involve multiple keys, inability to migrate Lua scripts, freezing or even high availability switchover triggered by large key migration, and complex rollback process.

In response to these issues, ApsaraDB for Redis Enhanced Edition (Tair) developed a new architecture based on slot replication to ensure imperceptible data migration. This architecture optimizes the thread-scheduling algorithm within instances to allow instances to be managed in an efficient and accurate manner. This is the principle of imperceptible scaling. This topic lists the drawbacks of the scaling solutions available for open source Redis clusters and ApsaraDB for Redis cluster instances and describes the imperceptible scaling solution available for ApsaraDB for Redis Enhanced Edition (Tair) cluster instances.

Common scaling solutions for open source Redis clusters and their drawbacks

- ### Elastic scaling of open source Redis clusters

  Open source Redis clusters use gossip protocols to transfer data. During data transfer, a slot is migrated as the smallest dataset by traversing and migrating keys in the slot. This scaling solution has the following issues:

  - Low stability
    - Commands that involve multiple keys in the same slot may fail to run because data is being migrated by key.
    - Lua scripts cannot be replicated at the same time when data is migrated. As such, Lua scripts may be lost after the data migration is complete.
    - When data is replicated, migration of large keys may cause freezing or even errors that may trigger high availability switchover.
  - High O&M difficulty
    - If an error is returned during data migration, you must manually restore the database data. This process is difficult, time-consuming, and error-prone.
    - It takes a long time to scale clusters because migrating data by key is time-consuming. As such, scaling is often performed during off-peak hours, and business may be affected.

- ### Scaling based on data synchronization and migration components

  This solution involves middleware components rather than open source Redis clusters to migrate data. For example, to perform scaling operations, you can create a cluster, use a middleware component to migrate data to the cluster, and then use a load balancer component to switch access paths. This scaling solution has the following issues:

  - It takes a long time to synchronize full data.
  - The costs are high because you must create two sets of resources to perform scaling operations.
  - Clients are disconnected when a load balancer component performs a switch. It takes a long time for the switch to take effect, and the service may be unavailable for up to 10 seconds.

Imperceptible scaling solution for ApsaraDB for Redis cluster instances and its drawbacks

The imperceptible scaling solution available for ApsaraDB for Redis cluster instances can address the preceding issues of open source Redis clusters. However, the solution also has the following issues:

> **Note** This solution supports ApsaraDB for Redis cluster instances that use cloud disks.

- Data migration performed during scaling operations affects instances, and instances may become read-only for a few seconds.

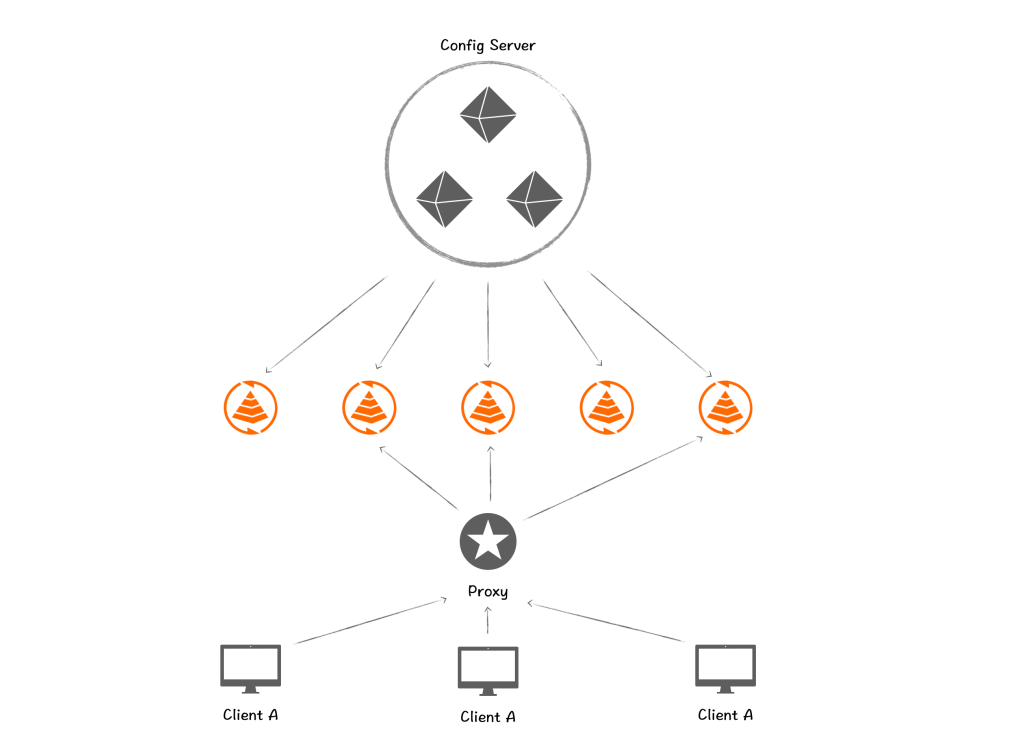
> **Note** Data migration needs may arise when you perform scaling operations. During data migration, when the proportion of synchronized incremental data to the total amount of incremental data to be synchronized is less than a threshold, the instance becomes read-only. After the remaining incremental data is synchronized, clients can re-connect to the instance by using a new endpoint. If an update request is sent to the instance while the instance is in the read-only state, the request is rejected and a read-only error is returned. Read-only error responses cannot be smoothly handled, which may affect your business.

- Data migration performed during scaling operations competes for resources with regular operations. As such, scaling operations are often performed during off-peak hours and become less flexible.

Imperceptible scaling solution for ApsaraDB for Redis Enhanced Edition (Tair)

ApsaraDB for Redis Enhanced Edition (Tair) cluster architecture is developed based on the new management architecture of ApsaraDB for Redis. The cluster architecture uses centralized components to manage instances in an efficient and accurate manner. This allows the architecture to implement imperceptible scaling.

> **Note** This solution supports performance-enhanced instances that use cloud disks and persistent memory-optimized instances that use cloud disks of the ApsaraDB for Redis Enhanced Edition (Tair). For more information, see Performance-enhanced instances and Persistent memory-optimized instances.



This solution has the following benefits:

- Imperceptible scaling

When an ApsaraDB for Redis Enhanced Edition (Tair) cluster instance is being scaled, you clients are not affected, your business is not interrupted, and the instance does not stay in the read-only state. You can scale an ApsaraDB for Redis Enhanced Edition (Tair) cluster instance at any time.

The key to imperceptible scaling is reducing the read-only time period on instances during data migration. ApsaraDB for Redis Enhanced Edition (Tair) cluster instances dynamically estimate the amount of time required to migrate remaining incremental data and keep the read-only time period within milliseconds. This theoretically prevents instances from entering the read-only state because this read-only time period is far less than the TCP retransmission time that is measured in hundreds of milliseconds. When an instance stays in the read-only state, the write requests made for the keys to be migrated are not written but cached to the instance. After the data migration is complete, clients receive redirection messages. At the same time, the management system and the database engine work together to update instance information as soon as the data migration is complete. This process ensures that scaling operations are imperceptible to clients.

- Smooth scaling

  ApsaraDB for Redis Enhanced Edition (Tair) optimizes the thread-scheduling algorithm within cluster instances to implement fine-grained management of data migration tasks. This improves thread execution efficiency from 10% to a maximum of 80%. You can specify a custom efficiency value within this range. This way, the data migration speed is maximized without impacting your business. ApsaraDB for Redis Enhanced Edition (Tair) cluster instances also support fine-grained scaling without increasing the reaction time (RT) to prevent high availability switchover caused by network jitter. This ensures high data reliability.

- Efficient and easy O&M

  ApsaraDB for Redis Enhanced Edition (Tair) cluster instances can address scaling issues of open source Redis clusters by using the following methods:

  - Pre-backup in the background: Pre-backup in the background can be implemented for instances. This method does not affect online services, and the full data of your instance can be replicated in advance. This prevents freezing caused by large key migration.
  - Rollback with a few clicks: You can roll back instances with a few clicks if exceptions occur during scaling.
  - Data migration by slot: Data can be migrated by slot. This ensures that commands that involve multiple keys in the same slot can run as expected.
  - Lua script replication: During data migration, Lua scripts can be replicated to prevent Lua script loss.
  - Horizontal scaling: Up to 256 shards can be added to or deleted from a single instance.

- Low costs

  Compared with solutions that require a middleware component, this solution reduces costs because you do no need to create two sets of resources.

4.Service architecture

4.1. Overview

ApsaraDB for Redis supports the standard, cluster, and read/write splitting architectures. You can select the instances suitable for your business requirements.

Architectures

The following table lists the supported architectures. You can click the architecture name to view more details.

Warning

Architecture	Edition	Description	Scenario
Standard master-replica instances	• Enhanced Edition • Community Edition	Master-replica architecture to ensure high availability	• Support for more native Redis features. • Persistent storage on ApsaraDB for Redis instances. • Stable query rate on a single node of ApsaraDB for Redis. • Use of simple Redis commands, when only a few sorting and computing commands are used.
Cluster master-replica instances	• Enhanced Edition • Community Edition	Uses a cluster architecture with multiple data shards. Each data shard runs in the master-replica mode and supports connections through proxy servers and internal endpoints.	• Large data volume. • High queries per second (QPS). • Throughput-intensive workloads.
Read/write splitting instances	• Enhanced Edition • Community Edition	Consists of multiple proxy servers, one master node, one replica node, and one or more read replicas.	• High QPS. • Support for more native Redis features. • Persistent storage on ApsaraDB for Redis instances.

Documentation applicability

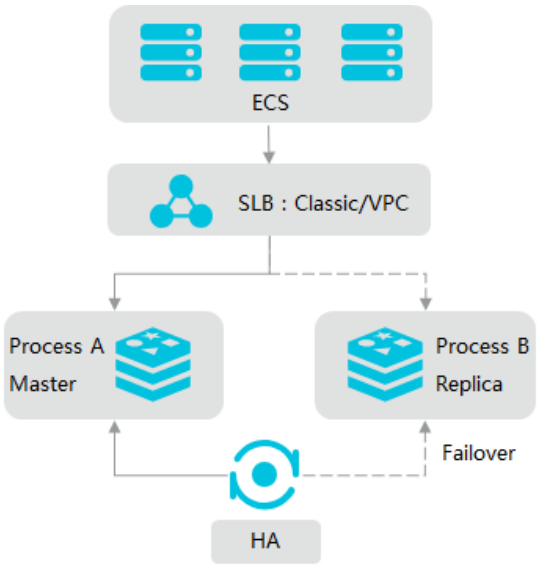
You must understand the following concepts in ApsaraDB for Redis: architectures, editions such as Community Edition and Enhanced Edition, series types such as performance-enhanced instances, and engine versions such as Redis 4.0 or 5.0. The descriptions and topics listed in the preceding table are applicable to related editions, series types, and engine versions. For standard instances, you can view architecture information in Standard master-replica instances. This rule also applies to cluster instances and read/write splitting instances.

4.2. Standard master-replica instances

Standard master-replica instances of ApsaraDB for Redis provide high-performance caching services and support high data reliability.

Overview

Standard master-replica instances of ApsaraDB for Redis run in a master-replica architecture. The master node serves your workloads and the replica node stays in hot standby mode to ensure high availability. If the master node fails, the system switches the workloads to the replica node within 30 seconds after the failure occurs. This mechanism guarantees the high availability for your workloads.



Key features

- Reliability
  - Service reliability

    Standard master-replica instances adopt a master-replica architecture with master and replica nodes deployed on different physical machines. The master node serves your workloads. You can use the Redis command-line interface (CLI) and common clients to add, delete, modify, and query data on the master node. Alibaba Cloud has developed a high-availability (HA) system for standard master-replica instances. If the master node fails, the HA system performs a failover to guarantee the high availability for your workloads.
  - Data reliability

    By default, data persistence is enabled for standard master-replica instances. Instances support data backup. You can clone or roll back an instance based on a specified backup set to restore data after misoperations. Instances created in zones that provide disaster recovery, such as Hangzhou Zone H and Zone I, also support zone disaster recovery.
- Compatibility

  Standard instances are compatible with all Redis protocols. You can migrate your workloads on an on-premises Redis database to a standard master-replica instance of ApsaraDB for Redis without service disruption. Alibaba Cloud also provides Data Transmission Service (DTS) to support incremental migration to ApsaraDB for Redis. This ensures a stable transition for your business.
- Proprietary systems developed by Alibaba Cloud

- HA system

  ApsaraDB for Redis uses the HA system to detect failures on the master node, such as disk input and output (I/O) failures and CPU failures, and performs failovers to ensure high availability.

- Master-replica replication mechanism

  Alibaba Cloud has customized the master-replica replication mechanism of ApsaraDB for Redis. You can replicate data in the format of incremental logs between the master node and the replica node. If the replication is interrupted, system performance and stability is unchanged. This resolves issues caused by the master-replica replication mechanism of native Redis databases.

  Some issues caused by the master-replica replication mechanism of native Redis databases are described in the following section:

  - If the replication is interrupted, the replica node runs the Partial Resynchronization (PSYNC) command to resynchronize partial data. During this process, if the resynchronization fails, the master node synchronizes all RDB files to the replica node.
  - To synchronize all RDB files, the master node must perform a full replication as a response to the single-threading mode. As a result, the master node has a latency of several milliseconds or seconds.
  - Child processes are created to perform copy-on-write (COW) tasks. The child processes consume memory on the master node. The master node may run out of memory and cause the application to exit abnormally.
  - The replica files that the master node generates consume disk I/O and CPU resources.
  - The replication of GB-level files may lead to outbound traffic bursts on the server and increase the sequential I/O throughput of disks. This delays responses and causes more issues.

Scenarios

- Support for more native Redis features

  Standard instances are compatible with all Redis protocols. You can migrate your workloads to instances without service disruption.

- Persistent storage on an ApsaraDB for Redis instance

  Standard instances support data persistence, backup, and recovery features to ensure data reliability.

- Stable query rate on a single node of ApsaraDB for Redis.

  The single-threading mode of native Redis databases is applied. Therefore, if your workloads support a query rate of lower than 100,000 QPS, we recommend that you use a standard instance. To achieve higher performance, select a cluster instance.

- Use of simple Redis commands, where only a few sorting and computing commands are used

  CPU performance is the main bottleneck due to the single-threading mode of native Redis databases. We recommend that you use a cluster instance of ApsaraDB for Redis to process a large number of sorting and computing workloads.

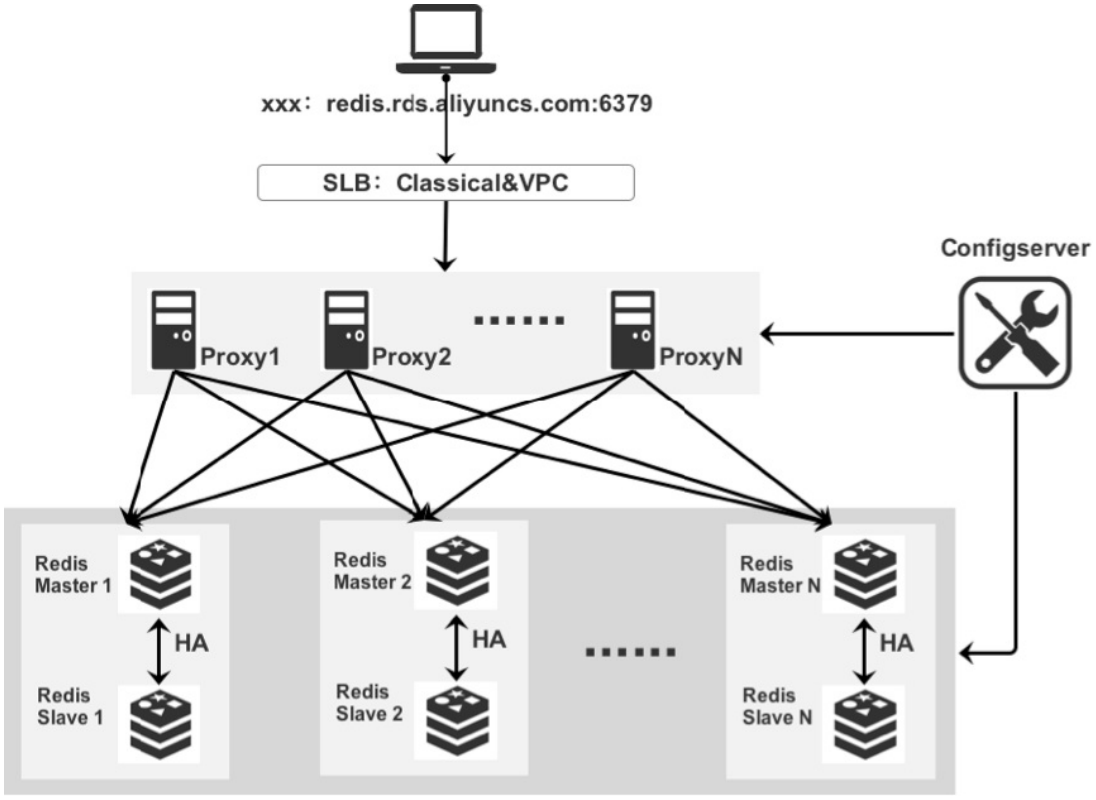
4.3. Cluster master-replica instances

This topic describes ApsaraDB for Redis cluster master-replica instances. They can address the bottlenecks of the single-threading model of open source Redis and provide large capacity and high performance. Cluster master-replica instances support two connection modes: proxy mode and direct connection mode. You can select a connection mode based on your business requirements.

Proxy mode

By default, cluster master-replica instances run in proxy mode. If a cluster master-replica instance runs in proxy mode, you can connect to the instance by using a unified endpoint (or domain name). Requests from clients are sent to the configured proxy nodes. Then, the proxy nodes forward the requests to the data shards of the instance. The proxy nodes, data shards, and config server of the instance do not provide separate endpoints. This simplifies application development and coding. For more information about the architecture and components of a cluster master-replica instance in proxy mode, see the following figure and table.

Architecture of a cluster master-replica instance in proxy mode



Components of a cluster master-replica instance in proxy mode

Component	Description
Proxy node	A cluster master-replica instance contains multiple proxy nodes. Each proxy node works in a single-node architecture. The cluster master-replica instance automatically balances loads and can fail over among the configured proxy nodes. **Note** For more information about the features of proxy nodes, see Features of proxy nodes.
Data shard	A cluster master-replica instance contains multiple data shards. Each data shard works in a high availability architecture in which a master node and a replica node are deployed on different hosts. If the master node is faulty, the cluster master-replica instance fails over to the replica node to ensure high availability.

Component	Description
Config server	The config server of a cluster master-replica instance works in a high availability architecture in which a master node and a replica node are deployed. The config server stores the configuration data and partitioning policies of the cluster master-replica instance.

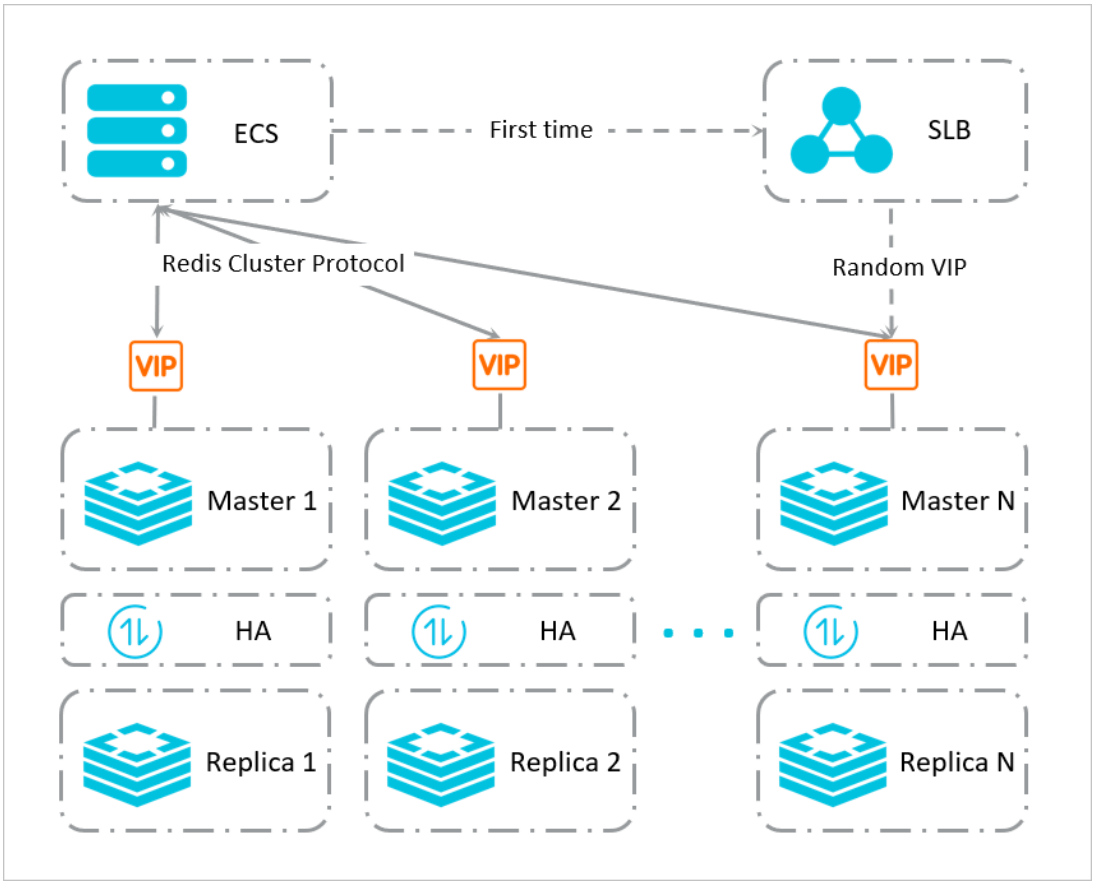
The component quantities and configurations in a cluster master-replica instance vary with the specifications of the instance. You cannot change the quantities or configurations. However, you can change the specifications and architecture of the instance. For more information about instance specifications, specification change, and architecture change, see Overview, Change the configurations of an instance, and Overview.

Direct connection mode

The proxy mode simplifies application development but decreases response speed because all requests must be forwarded by proxy nodes. If you require quick response, we recommend that you enable the direct connection mode. In this mode, you can bypass proxy nodes and directly connect to data shards. This reduces network overheads and increases response speed. For more information about the architecture of a cluster master-replica instance in direct connection mode, see the following figure and description.

> **Note** Compared with the proxy mode, the direct connection mode does not provide features such as load balancing or hotkey cache. For more information, see Features of proxy nodes.

Architecture of a cluster master-replica instance in direct connection mode



To use the direct connection mode on a cluster instance, you must enable the direct connection mode, obtain the private endpoint, and then connect to the instance in the same method as you connect to a native Redis cluster. For more information about how to enable the direction connection mode, see Enable the direct connection mode. The first time a client connects to the instance, the Domain Name System (DNS) resolves the private endpoint of the instance into a random virtual IP address (VIP). Then, the client can connect to the data shards of the instance by using the Redis cluster protocol. Direct connection mode and proxy mode are quite different from each other. For more information about the precautions and connection examples, see Use a private endpoint to connect to an ApsaraDB for Redis instance.

Scenarios

- Large amounts of data

  Compared with a standard master-replica instance, a cluster master-replica instance supports a storage capacity of up to 4,096 GB.

- High QPS

  A standard master-replica instance cannot support high queries per second (QPS) scenarios. A cluster master-replica instance allows you to deploy multiple data shards. These data shards can work together to eliminate the performance bottleneck of the single-threading model used by open source Redis. For more information, see Master-replica cluster instances.

- Throughput-intensive applications

  Compared with a standard master-replica instance, a cluster master-replica instance provides higher throughput over an internal network. You can efficiently read hot data and manage high-throughput workloads.

- Applications that do not require high compatibility with Redis protocols

  A cluster master-replica instance contains multiple components. As such, cluster master-replica instances are not as compatible with Redis protocols as standard master-replica instances. For information, see Limits on commands supported by cluster instances.

References

- For more information about how to troubleshoot abnormal memory usage of data shards in a cluster master-replica instance, see How do I search for large keys?
- For more information about how to analyze the distribution of data in memory, see Offline key analysis.

FAQ

Q: Can I enable both the direct connection mode and the proxy mode?

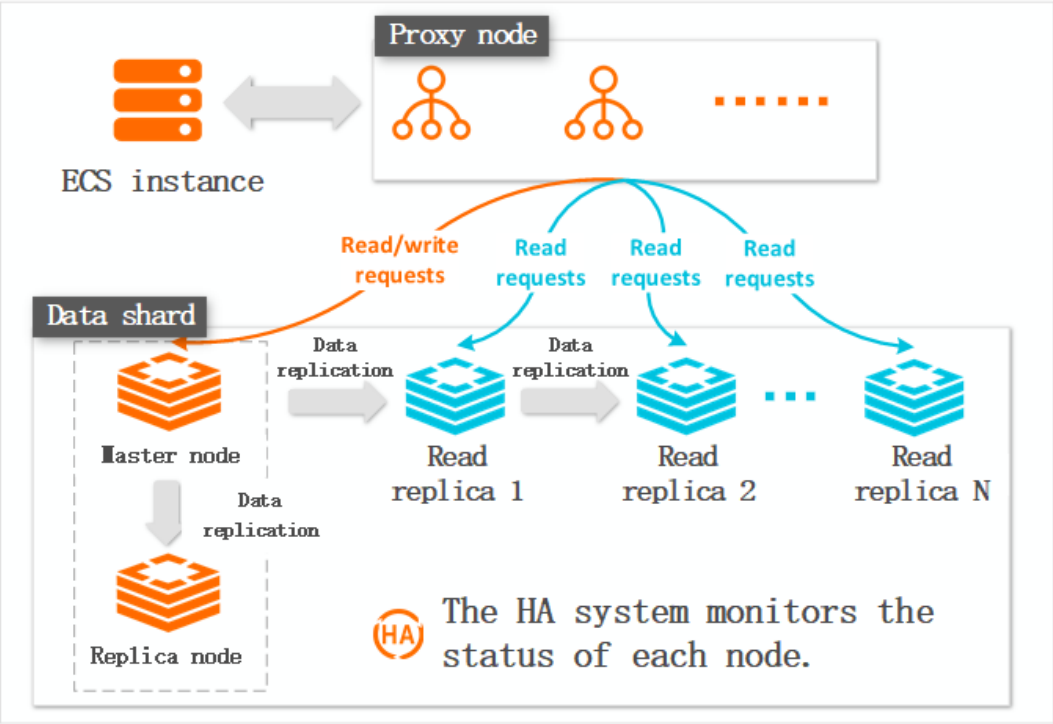
A: Yes, you can enable both the direct connection mode and the proxy mode. If you use a local disk-based cluster instance, you can enable the two modes at the same time. However, if you use a cloud disk-based one, you can only enable a single mode.

4.4. Read/write splitting instances

Read/write splitting instances of ApsaraDB for Redis are suitable for read-heavy workloads. These instances ensure high availability and high performance, and support multiple specifications. The read/write splitting architecture allows a large number of clients to concurrently read hot data from read replicas. This helps minimize O&M costs.

Overview

A read/write splitting instance of ApsaraDB for Redis consists of a master node, a replica node, read replicas, proxy servers, and a high-availability (HA) system.



Component	Description
Master node	The master node processes all write requests. It also processes specific read requests together with read replicas.
Replica node	As a hot standby node, the replica node does not provide services.
Read replicas	Read replicas process only read requests. The read/write splitting architecture supports chained replication. This allows you to scale out read replicas to increase the read capacity. Optimized binlog files are used to replicate data. This way, full synchronization can be avoided.
Proxy server	When clients send requests to a proxy server, the proxy server automatically identifies the type of requests and forwards the requests to different nodes based on the weights of the nodes. You cannot customize the weights of the nodes. For example, the proxy server forwards write requests to the master node and forwards read requests to the master node or read replicas. **Note** • Clients can connect only to proxy servers. Clients cannot directly connect to the nodes. • The system evenly distributes read requests among the master node and read replicas. You cannot customize the weights of these nodes. For example, if you purchase an instance with three read replicas, the weights of the master node and three read replicas are all 25%.

Component	Description
HA system	The HA system monitors the status of each node. If the master node fails, the HA system performs a failover between the master node and the replica node. If a read replica fails, the HA system creates another read replica to process read requests. During this process, the HA system updates the routing and weight information.

Benefits

- High availability
  - Alibaba Cloud has developed an HA system for read/write splitting instances. The HA system monitors the status of all nodes on an instance to ensure high availability. If the master node fails, the HA system switches the workloads from the master node to the replica node and updates the instance topology. If a read replica fails, the HA system creates another read replica. The HA system synchronizes data and forwards read requests to the new read replica, and suspends the failed read replica.
  - A proxy server monitors the service status of each read replica in real time. If a read replica is unavailable due to an exception, the proxy server reduces the weight of this read replica. If a read replica fails to be connected for a specified number of times, the system suspends the read replica and forwards read requests to available read replicas. The proxy server continuously monitors the status of the unavailable read replica. After the read replica recovers, the proxy server adds it to serviceable replicas and forwards requests to it.
- High performance

  The read/write splitting architecture supports chained replication. This allows you to scale out read replicas to increase the read capacity. The replication process is optimized based on the Redis source code to maximize workloads stability during replication and improve the usage of physical resources for each read replica.

Scenarios

- High QPS

  Standard instances of ApsaraDB for Redis do not support high queries per second (QPS). If your application is read-heavy, you can deploy multiple read replicas to relieve the stress on the master node. This allows you to resolve the performance bottleneck issue caused by the single-threading architecture of Redis. You can configure one, three, or five read replicas for a cluster instance of ApsaraDB for Redis. The QPS of a cluster instance can be about five times higher than that of a standard instance.

  > **Note** The read/write splitting architecture adopts chained replication and a latency exists in data synchronization to all read replicas. The longer the chain, the higher the latency at the end of the chain. Therefore, you can choose the read/write splitting architecture only in scenarios that allow dirty data. In scenarios that require high data consistency, we recommend that you choose the cluster architecture.
- Support for more open source Redis features

  Read/write splitting instances of ApsaraDB for Redis are compatible with all open source Redis commands. You can smoothly migrate data from a self-managed Redis database to a read/write splitting instance. You can also upgrade a master-replica standard instance to a read/write splitting instance.

> **Note** Read/write splitting instances have limits on specific commands. For more information, see Limits on the commands supported by read/write splitting instances.

Usage notes

- If a read replica fails, requests are forwarded to other available read replicas. If all read replicas are unavailable, requests are forwarded to the master node. Read replica failures may result in an increased number of workloads on the master node and an increased response time. To process a large number of read requests, we recommend that you use multiple read replicas.
- If an error occurs on a read replica, the HA system suspends the read replica and creates another read replica. This failover process involves resource allocation, data synchronization, and service loading. The amount of time that is required by a failover is based on the system workloads and data volume. ApsaraDB for Redis does not provide a guarantee on the time period within which a failed read replica can recover.
- Full synchronization between read replicas is triggered in specific scenarios. For example, it can be triggered when a failover occurs on the master node. During full synchronization, read replicas are unavailable. If your requests are forwarded to the read replicas, the following error message is returned: `-LOADING Redis is loading the dataset in memory\r\n` .
- The master node conforms to the Service Level Agreement of ApsaraDB for Redis.

Related information

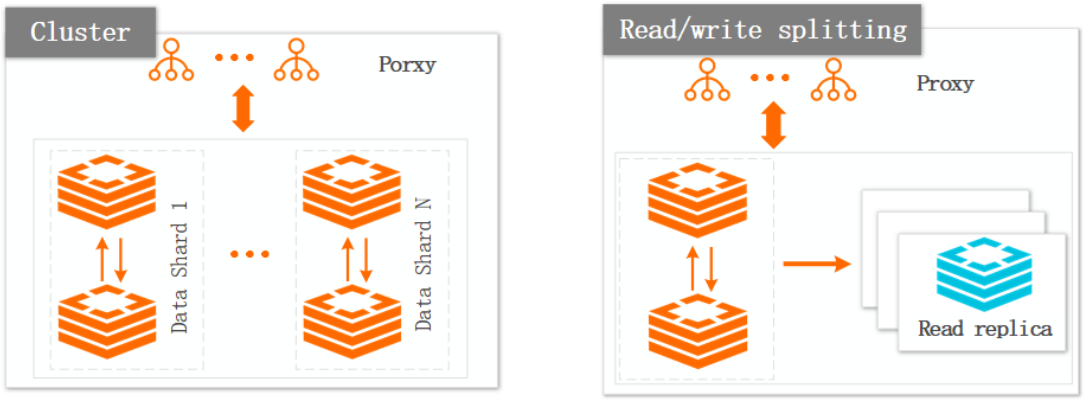
- Read/write splitting in Redis
- Limits on the commands supported by read/write splitting instances

4.5. Features of proxy nodes

ApsaraDB for Redis cluster instances and read/write splitting instances use proxy nodes to route commands, balance loads, and perform failovers. If you know how proxy nodes route commands and handle specific commands, you can understand how to design more effective business systems.

Overview of proxy nodes

Architectures of cluster instances and read/write splitting instances



A proxy node is a component that runs in the standalone architecture in an ApsaraDB for Redis instance. Proxy nodes do not occupy resources of data shards. An ApsaraDB for Redis instance uses multiple proxy nodes to balance loads and perform failovers.

Capability	Description
Enable architecture changes	Proxy nodes allow you to use a cluster instance the same manner as you would use a standard instance. If your business requirements grow beyond the capabilities that can be provided by a standard instance, you can migrate the data of the standard instance to a cluster instance that has proxy nodes without having to modify code to reduce costs.
Balance loads and route commands	Proxy nodes establish persistent connections with backend data shards to balance loads and route commands. For more information, see Routing methods of proxy nodes.
Manage the traffic to read replicas	Proxy nodes monitor the state of each read replica in real time. If a read replica is in one of the following states, proxy nodes stop routing traffic to the read replica: • Abnormal: If a read replica is abnormal, proxy nodes reduce the weight of traffic to the read replica. If the read replica fails to connect for the specified number of times consecutively, the proxy nodes stop routing traffic to the read replica until the read replica is normal again. • Full data synchronization: If proxy nodes detect that full data is being synchronized on a read replica, the proxy nodes stop routing traffic to the read replica until the synchronization is complete.
Cache hotkey data	After you enable the proxy query cache feature, proxy nodes cache the request and response data of hotkeys. If a proxy node receives a duplicate request during the validity period, the proxy node directly returns a response to the client without interacting with backend data shards. This prevents skewed requests caused by hotkeys that send a large number of read requests. For more information, see Use proxy query cache to address issues caused by hotkeys. **Note** This feature is supported only by performance-enhanced cluster instances of the ApsaraDB for Redis Enhanced Edition (Tair). For more information about performance-enhanced instances, see Performance-enhanced instances.
Support multiple databases	In cluster mode, multiple databases are not supported by open source Redis and Redis cluster clients. In this case, only the default `0` database can be used and the `SELECT` command is not supported. You can use proxy nodes to access ApsaraDB for Redis cluster instances. In this case, multiple databases can be used and the `SELECT` command is supported. By default, 256 databases can be specified for a single cluster instance.

Note As proxy nodes develop, the capabilities of the proxy nodes in clusters are no longer based only on the number of proxy nodes. Alibaba Cloud ensures that the number of proxy nodes meets the requirements described in the cluster specifications.

Routing methods of proxy nodes

Note For information about commands, see Overview.

Architecture	Routing method	Description
Cluster	Basic routing method	• When proxy nodes receive a command that manages an individual key, the proxy nodes identify the hash slot to which the key belongs and route the command to the data shard where the hash slot resides. • When proxy nodes receive a command that manages two or more keys stored in different shards, the proxy nodes split the command into multiple commands and route the commands to the corresponding shards.
	Routing method for specific commands	• Pub/Sub commands When proxy nodes receive Pub/Sub commands such as **PUBLISH** and **SUBSCRIBE**, the proxy nodes hash the channel names and route the commands to the corresponding data shards. **Note** To view the Pub/Sub monitoring data in a data shard, you can select **Data Node** on the Performance Monitor page in the ApsaraDB for Redis console and set Customize Metrics to **Pub/Sub Monitoring Group**. By default, the Pub/Sub data of the first data shard is displayed. For more information, see Customize metrics (previous version). • Redis commands developed by Alibaba Cloud When proxy nodes receive commands that are developed by Alibaba Cloud, such as **IINFO** and **ISCAN**, and the data shard IDs are specified in the idx parameter, the proxy nodes route the commands to the specified data shards. For more information, see Redis commands developed by Alibaba Cloud.
Read/write splitting	Basic routing method	• Proxy nodes route write commands to the master node. • The system evenly distributes read requests among the master node and read replicas. You cannot change the weights of these nodes. For example, if you purchase an instance that has three read replicas, the weights of the master node and three read replicas are all 25%. **Note** **SLOWLOG** and **DBSIZE** are read commands.

Architecture	Routing method	Description
	Routing method for specific commands	• SCAN commands Proxy nodes route the **HSCAN**, **SSCAN**, and **ZSCAN** commands to the If exceptions occur on all read replicas, these commands are routed to the master node. first read replica • Redis commands developed by Alibaba Cloud When proxy nodes receive commands that are developed by Alibaba Cloud, such as **RIINFO** and **RIMONITOR**, the proxy nodes route the commands to the read replicas specified by the ro_slave_idx parameter and the data shards specified by the idx parameter. For more information, see Redis commands developed by Alibaba Cloud. • Other commands Proxy nodes route transaction commands (such as **MULTI** and **EXEC**), Lua script commands (such as **EVAL** and **EVALSHA**), **SCAN** commands, **INFO** commands, and Pub/Sub commands (such as **PUBLISH** and **SUBSCRIBE**), to the master node.

Number of connections

In most cases, proxy nodes establish persistent connections with backend data shards to process requests from users. If the requests contain the following special commands, the proxy nodes establish additional connections with the data shards to process subsequent requests. Run the following commands with caution and make sure that the number of connections to each data shard does not exceed the upper limit.

> **Note** In proxy mode, Community Edition instances allow up to 10,000 connections to each data shard, whereas Enhanced Edition (Tair) instances allow up to 30,000 connections to each data shard.

- Blocking commands: **BRPOP**, **BRPOPLPUSH**, **BLPOP**, **BZPOPMAX**, and **BZPOPMIN**.
- Transaction commands: **MULTI**, **EXEC**, and **WATCH**.
- Monitoring commands: **MONITOR**, **IMONITOR**, and **RIMONITOR**.
- Sub commands: **SUBSCRIBE**, **UNSUBSCRIBE**, **PSUBSCRIBE**, and **PUNSUBSCRIBE**.
- **SCAN** command: scans multiple databases.

FAQ

- Q: What cross-slot commands that manage multiple keys can I run in proxy mode?

  A: You can run the following cross-slot commands that manage multiple keys in proxy mode: **DEL**, **EXISTS**, **MGET**, **MSET**, **SDIFF**, **SDIFFSTORE**, **SINTER**, **SINTERSTORE**, **SUNION**, **SUNIONSTORE**, and **UNLINK**.

- Q: Can proxy nodes route a Lua script command that only reads data to a read replica?

  A: Yes, proxy nodes can route a Lua script command that only reads data to a read replica. For this purpose, you must set the readonly_lua_route_ronode_enable parameter to *1* for your ApsaraDB for Redis instance. A value of 1 indicates that Lua script commands that only read data are routed to read replicas. For more information, see Modify parameters of an instance.

- Q: What is the difference between the proxy mode and the direct connection mode? Which mode is

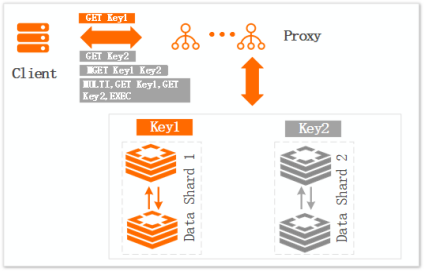
recommended?

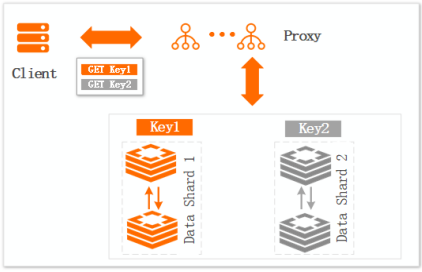
A: We recommend that you use the proxy mode. The following items describe the difference between the proxy mode and the direct connection mode:

- By default, cluster instances and read/write splitting instances provide proxy endpoints. Client requests are routed to data shards by proxy nodes. Proxy nodes provide rich features such as load balancing, read/write splitting, failover, proxy query cache, and persistent connection. For more information about proxy query cache, see Use proxy query cache to address issues caused by hotkeys.
- If you use a cluster instance, you can use a private endpoint to bypass proxy nodes. This way, you can directly access backend data shards. This is similar to a connection to a native Redis cluster. For more information about private endpoints, see Enable the direct connection mode. Compared with the proxy mode, the direct connection mode reduces the routing time and accelerates the response of ApsaraDB for Redis.

- Q: How does an abnormal data shard affect data reads and writes?

  A: Each data shard runs in a high-availability master-replica architecture. If the master node fails, the system switches the workloads to the replica node to ensure high availability. The following table describes the impacts of abnormal data shards on data reads and writes in specific scenarios and provides optimization methods for each scenario.

Scenario	Impact and optimization
Commands that manage multiple keys 	○ Impact: The client sends four requests over four connections. If Data Shard 2 is abnormal, timeout errors are returned for the requests that are routed to Data Shard 2. The queried data is returned only for Request 1 **GET Key1**. ○ Optimization methods: ■ Reduce the frequency of the commands that manage multiple keys such as **MGET** or reduce the number of keys that are managed by a request. This ensures that not all requests fail just because a single data shard is abnormal. ■ Reduce the frequency of transaction commands or reduce the transaction size. This way, when a subtransaction fails, it does not cause the entire transaction to fail.

Scenario	Impact and optimization
Single connection 	○ Impact: The client sends two requests over the same connection. If Data Shard 2 is abnormal, timeout errors are returned for Request 1 **GET Key1** and Request 2 **GET Key2**. In this example, Request 1 fails because it uses the same connection as Request 2. ○ Optimization methods: ■ Use **pipelines** as little as possible. ■ Do not use clients that support only a single connection such as Lettuce. We recommend that you use clients that support connection pools such as Jedis. If you use Jedis, you must configure a reasonable timeout period and connection pool size. For more information about Jedis, see Jedis client.

5.Instance specifications

5.1. Overview

ApsaraDB for Redis provides multiple editions, series types, and architectures. This topic helps you find the references about the specifications of different instances.

Instance types that support local disks

This topic lists the references that describe the specifications of ApsaraDB for Redis instances based on editions, series types, and architectures. You can find the detailed specifications of an instance type by clicking a link in the Reference column in the following table.

Edition	Series type	Reference	Overview
Community Edition	None	Master-replica standard instances	A master-replica instance of ApsaraDB for Redis. The maximum memory is 64 GB, and the maximum queries per second (QPS) is about 80,000. This maximum QPS value is for reference only.
		Master-replica cluster instances	A cluster instance of ApsaraDB for Redis. Each shard is deployed in a master-replica architecture. The instance with the highest specifications contains 32 shards and supports 512 GB of memory. The maximum QPS is about 2,560,000. This maximum QPS value is for reference only.
		Read/write splitting instances	A read/write splitting instance of ApsaraDB for Redis that contains a master node and one or more read replicas. The instance with the highest specifications contains five read replicas and supports 64 GB of memory.
		Read/write splitting cluster instances (retired)	A read/write splitting instance of ApsaraDB for Redis that contains a cluster and one or more read replicas. A read replica is attached to each shard in the cluster. The instance with the highest specifications contains 32 shards and 32 read replicas, and supports 512 GB of memory.
		Standard instances	A standard master-replica instance that uses the multi-threading model and provides a performance about three times that of a Community Edition instance of the same specifications. The maximum memory is 64 GB, and the maximum QPS is about 240,000. This maximum QPS value is for reference only.

Edition	Series type	Reference	Overview
Overview	Performance-enhanced instances	Cluster instances	A cluster instance of ApsaraDB for Redis that uses the multi-threading model and provides a performance about three times that of a Community Edition instance of the same specifications. Each shard uses a master-replica architecture. The maximum memory is 4,096 GB, and the maximum QPS is about 61,440,000. This maximum QPS value is for reference only.
		Read/write splitting instances	A read/write splitting instance of ApsaraDB for Redis that uses the multi-threading model, runs in a master-replica architecture, and contains a master node and one or more read replicas. A read/writing splitting instance provides a performance three times that of a Community Edition instance of the same specifications. The instance with the highest specifications contains five read replicas and supports 64 GB of memory.
	Persistent memory-optimized instances	Standard instances and cluster instances	An instance that uses the standard architecture or the cluster architecture. The instance does not use disks to implement data persistence. The instance provides almost the same performance as a Community Edition instance in terms of throughput and latency while persisting each operation. A standard persistent memory-optimized instance is only about 70% the price of a Community Edition instance.
	Storage-optimized instances	Standard instances	A storage-optimized instance that runs in a master-replica architecture. The instance reduces up to 85% of costs compared with a Community Edition instance. The instance is suitable for scenarios that store warm and cold data, and require compatibility with Redis, large capacity, and high access performance.
Retired instance types	N/A	N/A	ApsaraDB for Redis instances that are phased out. If you have purchased one or more of these instances, you can continue to use them. For more information about the specifications of these instances, such as the maximum number of connections, maximum bandwidth value, and maximum QPS value, see Retired instance types.

Instance types that support cloud disks

Edition	Series type	Reference	Overview

Edition	Series type	Reference	Overview
Community Edition	None	Community Edition with cloud disks	A Community Edition instance that uses the standard architecture or the cluster architecture. • If the instance uses the standard architecture, the maximum memory is 64 GB and the maximum QPS is 100,000. • If the instance uses the cluster architecture, the maximum memory is 2,048 GB and the instance performance is the product of shard quantity and performance per shard specified by shard specifications.
Enhanced Edition (Tair)	Performance-enhanced instances	Performance-enhanced instances	A performance-enhanced instance that uses the standard architecture or the cluster architecture. The instance uses the multi-threading model, provides up to 64 GB of memory, and supports about 240,000 QPS. It provides a performance about three times that of a Community Edition instance.
	Persistent memory-optimized instances	Persistent memory-optimized instances	A performance-enhanced instance that uses the standard architecture or the cluster architecture. The instance does not use disks to implement data persistence. The instance provides almost the same performance as a Community Edition instance in terms of throughput and latency while persisting each operation. A standard persistent memory-optimized instance is only about 70% the price of a Community Edition instance.
	Storage-optimized instances	Standard instances	A storage-optimized instance that runs in a master-replica architecture. The instance reduces up to 85% of costs compared with a Community Edition instance. The instance is suitable for scenarios that store warm and cold data, and require compatibility with Redis, large capacity, and high access performance.

FAQ

- Q: Do I need to reserve memory for snapshots when I select specifications?

  A: No, you do not need to reserve memory for snapshots when you select specifications. ApsaraDB for Redis resources are sold as instances of the Community Edition and Enhanced Edition (Tair). You do not need to reserve memory for snapshots. The memory capacity of each specification type is the maximum memory that is available to you. The memory capacity includes the memory occupied by user data, the static memory consumed by your instance, and the memory occupied by network connections.

- Q: Each specification type has a maximum QPS value. What happens if the QPS of an instance exceeds the maximum value?

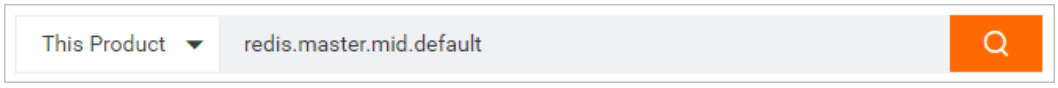
  A: This may cause an accumulation of requests. If the QPS reference value is exceeded for an extended period of time, we recommend that you choose higher specifications. For more information, see Change the configurations of an instance.

- Q: Why is a specific specification type unavailable?

  A: The specification type may be phased out. For more information, see Retired instance types.

- Q: How can I check the specifications of an ApsaraDB for Redis instance by using InstanceClass?

A: You can enter the value of InstanceClass in the search box at the top of an Alibaba Cloud document.



- Q: How do I test the performance of ApsaraDB for Redis instances?

  A: You can test the performance of ApsaraDB for Redis instances by using the methods that are described in Performance White Paper. For more information, see Test environment

Related information

- Comparison between ApsaraDB for Redis Enhanced Edition (Tair) and ApsaraDB for Redis Community Edition
- Comparison between ApsaraDB for Redis instances that use local disks and those that use cloud disks

5.2. Community Edition

5.2.1. Master-replica standard instances

This topic describes the specifications of master-replica standard instances of ApsaraDB for Redis Community Edition. These specifications include the memory capacity, maximum number of connections to each instance, maximum bandwidth, and queries per second (QPS) reference value.

Instance types

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth value (MB/s)	QPS reference value
256 MB master-replica instance	redis.master.micro.default	2	10,000	10,000	10	80,000
1 GB master-replica instance	redis.master.small.default	2	10,000	10,000	10	80,000
2 GB master-replica instance	redis.master.mid.default	2	10,000	10,000	16	80,000
4 GB master-replica instance	redis.master.stand.default	2	10,000	10,000	24	80,000
8 GB master-replica instance	redis.master.large.default	2	10,000	10,000	24	80,000

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth value (MB/s)	QPS reference value
16 GB master-replica instance	redis.master.2xlarge.default	2	10,000	10,000	32	80,000
32 GB master-replica instance	redis.master.4xlarge.default	2	10,000	10,000	32	80,000
64 GB master-replica instance	redis.master.8xlarge.default	2	10,000	10,000	48	80,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth values

- The bandwidth value in the preceding table is the bandwidth of the instance. The bandwidth value is the sum of the bandwidth of all shards or nodes in the instance.
- The upper limit of the total bandwidth for a read/write splitting instance is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards or nodes to the read/write splitting instance.
- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidths of the instance are both 10 MB/s.

  > **Note** If your instance has unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance as needed. For more information, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

Calculation rules for connections

The following example shows how the maximum number of new connections per second affects the maximum number of connections at a specific second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections of the instance is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the maximum number of connections is 22,000, which is calculated by using the following formula: 12,000 + 10,000.

FAQ

- Q: Why is the actual bandwidth of my instance different from that described in this topic?

  A: The bandwidth of specific instance types may be adjusted after the service is upgraded. If the bandwidth of your instance is different from that described in this topic, you can change your instance specifications by selecting the same instance type to update the bandwidth. The price for your instance remains unchanged after the specifications change. For more information, see Change the configurations of an instance.

- Q: How do I create an instance with 256 MB of memory?

  A: On the Subscription tab of the buy page, you can select an instance type with 256 MB of memory.

5.2.2. Master-replica cluster instances

This topic describes the specifications of master-replica cluster instances of the ApsaraDB for Redis Community Edition. These specifications include the memory capacity, maximum number of connections to each instance, maximum bandwidth, and queries per second (QPS) reference value.

Instance types

Instance types: two shards, four shards, eight shards, 16 shards, 32 shards, 64 shards, 128 shards, and 256 shards.

- 2 shards

  For these instance types, the is 2, the number of CPU cores is 4, and the maximum number of new connections per second is 20,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
2 GB cluster instance (2 shards)	redis.logic.sharding.1g.2db.0rodb.4proxy.default	1	480,000	20,000	96	48	200,000
4 GB cluster instance (2 shards)	redis.logic.sharding.2g.2db.0rodb.4proxy.default	2	480,000	20,000	192	96	200,000

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
8 GB cluster instance (2 shards)	redis.logic.sharding.4g.2db.0rodb.4proxy.default	4	480,000	20,000	192	96	200,000
16 GB cluster instance (2 shards)	redis.logic.sharding.8g.2db.0rodb.4proxy.default	8	480,000	20,000	192	96	200,000
32 GB cluster instance (2 shards)	redis.logic.sharding.16g.2db.0rodb.4proxy.default	16	480,000	20,000	192	96	200,000

- 4 shards

For these instance types, the is 4, the number of CPU cores is 8, and the maximum number of new connections per second is 40,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
4 GB cluster instance (4 shards)	redis.logic.sharding.1g.4db.0rodb.4proxy.default	1	480,000	40,000	192	48	400,000
8 GB cluster instance (4 shards)	redis.logic.sharding.2g.4db.0rodb.4proxy.default	2	480,000	40,000	384	96	400,000
16 GB cluster instance (4 shards)	redis.logic.sharding.4g.4db.0rodb.4proxy.default	4	480,000	40,000	384	96	400,000

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
24 GB cluster instance (4 shards)	redis.logic.sharding.6g.4db.0rodb.4proxy.default	6	480,000	40,000	384	96	400,000
32 GB cluster instance (4 shards)	redis.logic.sharding.8g.4db.0rodb.4proxy.default	8	480,000	40,000	384	96	400,000
64 GB cluster instance (4 shards)	redis.logic.sharding.16g.4db.0rodb.4proxy.default	16	480,000	40,000	384	96	400,000
128 GB cluster instance (4 shards)	redis.logic.sharding.32g.4db.0rodb.8proxy.default	32	480,000	40,000	384	96	400,000

- 8 shards

  For these instance types, the is 8, the number of CPU cores is 16, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
8 GB cluster instance (8 shards)	redis.logic.sharding.1g.8db.0rodb.8proxy.default	1	500,000	80,000	384	48	800,000
16 GB cluster instance (8 shards)	redis.logic.sharding.2g.8db.0rodb.8proxy.default	2	500,000	80,000	768	96	800,000

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
32 GB cluster instance (8 shards)	redis.logic.sharding.4g.8db.0rodb.8proxy.default	4	500,000	80,000	768	96	800,000
48 GB cluster instance (8 shards)	redis.logic.sharding.6g.8db.0rodb.8proxy.default	6	500,000	80,000	768	96	800,000
64 GB cluster instance (8 shards)	redis.logic.sharding.8g.8db.0rodb.8proxy.default	8	500,000	80,000	768	96	800,000
128 GB cluster instance (8 shards)	redis.logic.sharding.16g.8db.0rodb.8proxy.default	16	500,000	80,000	768	96	800,000

- 16 shards

  For these instance types, the is 16, the number of CPU cores is 32, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
16 GB cluster instance (16 shards)	redis.logic.sharding.1g.16db.0rodb.16proxy.default	1	500,000	160,000	768	48	1,600,000
32 GB cluster instance (16 shards)	redis.logic.sharding.2g.16db.0rodb.16proxy.default	2	500,000	160,000	1,536	96	1,600,000

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
64 GB cluster instance (16 shards)	redis.logic.sharding.4g.16db.0rodb.16proxy.default	4	500,000	160,000	1,536	96	1,600,000
96 GB cluster instance (16 shards)	redis.logic.sharding.6g.16db.0rodb.16proxy.default	6	500,000	160,000	1,536	96	1,600,000
128 GB cluster instance (16 shards)	redis.logic.sharding.8g.16db.0rodb.16proxy.default	8	500,000	160,000	1,536	96	1,600,000
256 GB cluster instance (16 shards)	redis.logic.sharding.16g.16db.0rodb.16proxy.default	16	500,000	160,000	1,536	96	1,600,000

- 32 shards

  For these instance types, the is 32, the number of CPU cores is 64, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
64 GB cluster instance (32 shards)	redis.logic.sharding.2g.32db.0rodb.32proxy.default	2	500,000	320,000	2,048	96	3,200,000

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
128 GB cluster instance (32 shards)	redis.logic.sharding.4g.32db.0rodb.32proxy.default	4	500,000	320,000	2,048	96	3,200,000
192 GB cluster instance (32 shards)	redis.logic.sharding.6g.32db.0rodb.32proxy.default	6	500,000	320,000	2,048	96	3,200,000
256 GB cluster instance (32 shards)	redis.logic.sharding.8g.32db.0rodb.32proxy.default	8	500,000	320,000	2,048	96	3,200,000
512 GB cluster instance (32 shards)	redis.logic.sharding.16g.32db.0rodb.32proxy.default	16	500,000	320,000	2,048	96	3,200,000

- 64 shards

For these instance types, the is 64, the number of CPU cores is 128, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
128 GB cluster instance (64 shards)	redis.logic.sharding.2g.64db.0rodb.64proxy.default	2	500,000	640,000	2,048	96	6,400,000

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
256 GB cluster instance (64 shards)	redis.logic.sharding.4g.64db.0rodb.64proxy.default	4	500,000	640,000	2,048	96	6,400,000
512 GB cluster instance (64 shards)	redis.logic.sharding.8g.64db.0rodb.64proxy.default	8	500,000	640,000	2,048	96	6,400,000

- 128 shards

  For these instance types, the is 128, the number of CPU cores is 256, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
512 GB cluster instance (128 shards)	redis.logic.sharding.4g.128db.0rodb.128proxy.default	4	500,000	1,280,000	2,048	96	12,800,000

- 256 shards

  For these instance types, the is 256, the number of CPU cores is 512, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total maximum bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
1024 GB cluster instance (256 shards)	redis.logic.sharding.4g.256db.0rodb.256proxy.default	4	500,000	2,560,000	2,048	96	25,600,000
2048 GB cluster instance (256 shards)	redis.logic.sharding.8g.256db.0rodb.256proxy.default	8	500,000	2,560,000	2,048	96	25,600,000
4096 GB cluster instance (256 shards)	redis.logic.sharding.16g.256db.0rodb.256proxy.default	16	500,000	2,560,000	2,048	96	25,600,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth values

- The bandwidth value in the preceding tables is the bandwidth of the instance. The bandwidth value is the sum of the bandwidths of all shards or nodes in the instance. Each shard has the following bandwidth limits:
  - Community Edition: If the memory of each shard is less than or equal to 1 GB, the maximum bandwidth of each shard is 48 Mbit/s. If the memory of each shard is greater than 1 GB, the maximum bandwidth of each shard is 96 Mbit/s.
  - Enhanced Edition (Tair): The maximum bandwidth of each shard is 96 Mbit/s.
- If the default proxy endpoint is used by a cluster instance, the upper limit of bandwidth for the cluster instance is 2,048 Mbit/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards to the cluster instance.

  To handle network traffic peaks, you can enable the direct connection mode. For more information, see Enable the direct connection mode. The direct connection mode is applicable only to cluster instances. After you enable the direct connection mode, take note of the following items:
  - The upper limit of connections is equal to the maximum number of connections for a single shard multiplied by the number of shards. The maximum number of connections for a single shard in Community Edition is 10,000. The maximum number of connections for a single shard in Enhanced Edition (Tair) is 30,000.

- The upper limit of the bandwidth is equal to the maximum bandwidth of a single shard multiplied by the number of shards. For example, if a cluster instance contains 128 shards and each shard is allocated with a memory of more than 1 GB and a maximum bandwidth of 96 Mbit/s, the bandwidth limit of the instance is 12,288 Mbit/s after you enable the direct connection mode.

- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 Mbit/s, the upstream and downstream bandwidths of the instance are both 10 Mbit/s.

  > **Note** If your instance may encounter unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance. For more information about, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding tables is the internal bandwidth of ApsaraDB for Redis instances. The public bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

Calculation rules for connections

Item	Description
Maximum number of concurrent connections	• : The maximum number of connections to an instance is 500,000. If the upper limit of 500,000 connections is reached, subsequent connections cannot be established even if you add more shards or nodes. Proxy mode • : The maximum number of connections for a single shard is 10,000. The maximum number of connections for an instance can be calculated by using the following formular: Number of shards × 10,000. Direct connection mode
Number of new connections per second	The upper limit of the number of new connections per second refers to the number of connections that are allowed to be established per second. For example, assume that the maximum number of new connections to an instance per second is 10,000, the maximum number of connections to the instance is 50,000, and the actual number of connections at the Nth second after the instance starts to run is 12,000. In this case, at the (N+1)th second, the maximum number of connections is 22,000. This value is calculated by using the following formula: 12,000 + 10,000. **Note** If the instance is connected in direct connection mode, connection performance degrades because proxy nodes are bypassed. For more information about the direct connection mode, see Enable the direct connection mode. The maximum number of new connections per second for a single data shard is 2,000. For example, if an instance has four data shards, the maximum number of new connections per second is 8,000.

FAQ

- Q: Why is the actual bandwidth of my instance different from that described in this topic?

  A: The bandwidths of specific instance types may be adjusted after the service is upgraded. If the actual bandwidth of your instance is different from that described in this topic, you can change the configurations of your instance by selecting the same instance type to update the bandwidth. The price for your instance remains unchanged after the configuration change. For more information, see Change the configurations of an instance.

- Q: Why am I unable to write data to a cluster instance when instance memory is not exhausted?

  A: This is because the instance contains exhausted data shards. ApsaraDB for Redis uses the hash algorithm to evenly write data to different data shards. If the instance contains large keys, resources are unevenly distributed in the instance and the data shards that house these large keys may even be exhausted. In this case, part of instance write operations may fail.

  You can use the **Performance Monitor** feature to check and optimize the performance of each data shard. For more information, see How do I view the memory of child instances of an ApsaraDB for Redis cluster instance?

5.2.3. Read/write splitting instances

This topic describes the specifications of read/write splitting instances of ApsaraDB for Redis Community Edition. The specifications include the memory capacity, maximum number of concurrent connections to each instance, maximum bandwidth, and queries per second (QPS) value.

Instance types

Note The following table describes the instance types that contain only one shard. For example, **1 GB read/write splitting instance (1 shard, 3 read replicas)** indicates that the instance has one shard that contains three read replicas.

Instance type	InstanceClass that is used in API operations	CPU cores	Read replicas	Maximum bandwidth (MB/s)	Maximum new connections per second	Maximum concurrent connections	QPS
1 GB read/write splitting instance (1 shard, 1 read replica)	redis.logic.splitrw.small.1db.1rodb.4proxy.default	4	1	96	20,000	20,000	200,000
1 GB read/write splitting instance (1 shard, 3 read replicas)	redis.logic.splitrw.small.1db.3rodb.4proxy.default	8	3	192	40,000	40,000	400,000
1 GB read/write splitting instance (1 shard, 5 read replicas)	redis.logic.splitrw.small.1db.5rodb.6proxy.default	12	5	288	50,000	60,000	600,000
2 GB read/write splitting instance (1 shard, 1 read replica)	redis.logic.splitrw.mid.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
2 GB read/write splitting instance (1 shard, 3 read replicas)	redis.logic.splitrw.mid.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000

Instance type	InstanceClass that is used in API operations	CPU cores	Read replicas	Maximum bandwidth (MB/s)	Maximum new connections per second	Maximum concurrent connections	QPS
2 GB read/write splitting instance (1 shard, 5 read replicas)	redis.logic.splitrw.mid.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000
4 GB read/write splitting instance (1 shard, 1 read replica)	redis.logic.splitrw.stand.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
4 GB read/write splitting instance (1 shard, 3 read replicas)	redis.logic.splitrw.stand.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
4 GB read/write splitting instance (1 shard, 5 read replicas)	redis.logic.splitrw.stand.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000
8 GB read/write splitting instance (1 shard, 1 read replica)	redis.logic.splitrw.large.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
8 GB read/write splitting instance (1 shard, 3 read replicas)	redis.logic.splitrw.large.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
8 GB read/write splitting instance (1 shard, 5 read replicas)	redis.logic.splitrw.large.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000
16 GB read/write splitting instance (1 shard, 1 read replica)	redis.logic.splitrw.2xlarge.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
16 GB read/write splitting instance (1 shard, 3 read replicas)	redis.logic.splitrw.2xlarge.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000

Instance type	InstanceClass that is used in API operations	CPU cores	Read replicas	Maximum bandwidth (MB/s)	Maximum new connections per second	Maximum concurrent connections	QPS
16 GB read/write splitting instance (1 shard, 5 read replicas)	redis.logic.splitrw.2xlarge.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000
32 GB read/write splitting instance (1 shard, 1 read replica)	redis.logic.splitrw.4xlarge.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
32 GB read/write splitting instance (1 shard, 3 read replicas)	redis.logic.splitrw.4xlarge.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
32 GB read/write splitting instance (1 shard, 5 read replicas)	redis.logic.splitrw.4xlarge.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000
64 GB read/write splitting instance (1 shard, 1 read replica)	redis.logic.splitrw.8xlarge.1db.1rodb.4proxy.default	4	1	192	20,000	20,000	200,000
64 GB read/write splitting instance (1 shard, 3 read replicas)	redis.logic.splitrw.8xlarge.1db.3rodb.4proxy.default	8	3	384	40,000	40,000	400,000
64 GB read/write splitting instance (1 shard, 5 read replicas)	redis.logic.splitrw.8xlarge.1db.5rodb.6proxy.default	12	5	576	50,000	60,000	600,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth values

- The bandwidth value in the preceding table is the bandwidth of the instance. The bandwidth value is the sum of the bandwidth of all shards or nodes in the instance.
- The upper limit of the total bandwidth for a read/write splitting instance is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards or nodes to the read/write splitting instance.

- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidths of the instance are both 10 MB/s.

  > **Note** If your instance has unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance as needed. For more information, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

Calculation rules for connections

The following example shows how the maximum number of new connections per second affects the maximum number of connections at a specific second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections of the instance is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the maximum number of connections is 22,000, which is calculated by using the following formula: 12,000 + 10,000.

5.3. Enhanced Edition

5.3.1. Performance-enhanced standard instances

This topic describes the specifications of performance-enhanced standard instances of ApsaraDB for Redis Enhanced Edition (Tair). These specifications include the memory capacity, maximum number of connections to each instance, maximum bandwidth, and queries per second (QPS) reference value.

Instance types

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Number of I/O threads	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth value (MB/s)	QPS reference value
1 GB master-replica performance-enhanced instance	redis.amber.master.small.multithread	6	4	10,000	30,000	96	240,000
2 GB master-replica performance-enhanced instance	redis.amber.master.mid.multithread	6	4	10,000	30,000	96	240,000

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Number of I/O threads	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth value (MB/s)	QPS reference value
4 GB master-replica performance-enhanced instance	redis.amber.master.stand.multithread	6	4	10,000	30,000	96	240,000
8 GB master-replica performance-enhanced instance	redis.amber.master.large.multithread	6	4	10,000	30,000	96	240,000
16 GB master-replica performance-enhanced instance	redis.amber.master.2xlarge.multithread	6	4	10,000	30,000	96	240,000
32 GB master-replica performance-enhanced instance	redis.amber.master.4xlarge.multithread	6	4	10,000	30,000	96	240,000
64 GB master-replica performance-enhanced instance	redis.amber.master.8xlarge.multithread	6	4	10,000	30,000	96	240,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth values

- The bandwidth value in the preceding table is the bandwidth of the instance. The bandwidth value is the sum of the bandwidth of all shards or nodes in the instance.
- The upper limit of the total bandwidth for a read/write splitting instance is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards or nodes to the read/write splitting instance.
- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidths of the instance are both 10 MB/s.

> **Note** If your instance has unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance as needed. For more information, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

Calculation rules for connections

The following example shows how the maximum number of new connections per second affects the maximum number of connections at a specific second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections of the instance is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the maximum number of connections is 22,000, which is calculated by using the following formula: 12,000 + 10,000.

FAQ

Q: Why is the maximum number of connections of my instance different from that described in this topic?

A: The maximum number of connections of specific instance types may be adjusted after the service is upgraded. If the maximum number of connections of your instance is different from that described in this topic, change your instance specifications to update the bandwidth. Make sure that you select the same instance type when you change the specifications. For more information, see Change the configurations of an instance.

5.3.2. Performance-enhanced cluster instances

This topic describes the specifications of performance-enhanced cluster instances of the ApsaraDB for Redis Enhanced Edition (Tair). The specifications include memory capacity, maximum number of connections, maximum bandwidth, and queries per second (QPS) reference value.

Instance types

Instance types: 2 shards, 4 shards, 8 shards, 16 shards, 32 shards , 64 shards, 128 shards, and 256 shards.

- 2 shards

  For these instance types, the is 2, the number of CPU cores is 12, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
2 GB performance-enhanced cluster instance (2 shards)	redis.amber.logic.sharding.1g.2db.0rodb.6proxy.multithread	1	480,000	60,000	192	96	480,000
4 GB performance-enhanced cluster instance (2 shards)	redis.amber.logic.sharding.2g.2db.0rodb.6proxy.multithread	2	480,000	60,000	192	96	480,000

- 4 shards

  For these instance types, the is 4, the number of CPU cores is 24, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
8 GB performance-enhanced cluster instance (4 shards)	redis.amber.logic.sharding.2g.4db.0rodb.12proxy.multithread	2	500,000	120,000	384	96	960,000

- 8 shards

  For these instance types, the is 8, the number of CPU cores is 48, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
16 GB performance-enhanced cluster instance (8 shards)	redis.amber.logic.sharding.2g.8db.0rodb.24proxy.multithread	2	500,000	240,000	768	96	1,920,000
32 GB performance-enhanced cluster instance (8 shards)	redis.amber.logic.sharding.4g.8db.0rodb.24proxy.multithread	4	500,000	240,000	768	96	1,920,000
64 GB performance-enhanced cluster instance (8 shards)	redis.amber.logic.sharding.8g.8db.0rodb.24proxy.multithread	8	500,000	240,000	768	96	1,920,000

- 16 shards

  For these instance types, the is 16, the number of CPU cores is 96, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
128 GB performance-enhanced cluster instance (16 shards)	redis.amber.logic.sharding.8g.16db.0rodb.48proxy.multithread	8	500,000	480,000	1,536	96	3,840,000
256 GB performance-enhanced cluster instance (16 shards)	redis.amber.logic.sharding.16g.16db.0rodb.48proxy.multithread	16	500,000	480,000	1,536	96	3,840,000

- 32 shards

  For these instance types, the is 32, the number of CPU cores is 192, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
512 GB performance-enhanced cluster instance (32 shards)	redis.amber.logic.sharding.16g.32db.0rodb.96proxy.multithread	16	500,000	960,000	2,048	96	7,680,000

- 64 shards

  For these instance types, the is 64, the number of CPU cores is 384, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
1024 GB performance-enhanced cluster instance (64 shards)	redis.amber.logic.sharding.16g.64db.0rodb.192proxy.multithread	16	500,000	1,920,000	2,048	96	15,360,000

- 128 shards

  For these instance types, the is 128, the number of CPU cores is 768, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
2048 GB performance-enhanced cluster instance (128 shards)	redis.amber.logic.sharding.16g.128db.0rodb.384proxy.multithread	16	500,000	3,840,000	2,048	96	30,720,000

- 256 shards

  For these instance types, the is 256, the number of CPU cores is 1,536, and the maximum number of new connections per second is 50,000. number of shards

Instance type	InstanceClass value (used in API operations)	Memory per shard (GB)	Maximum number of concurrent connectionsProxy mode	Maximum number of concurrent connectionsDirect connection mode	Total bandwidth (Mbit/s)	Bandwidth per shard (Mbit/s)	QPS reference value
4096 GB performance-enhanced cluster instance (256 shards)	redis.amber.logic.sharding.16g.256db.0rodb.768proxy.multithread	16	500,000	7,680,000	2,048	96	61,440,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth values

- The bandwidth value in the preceding tables is the bandwidth of the instance. The bandwidth value is the sum of the bandwidths of all shards or nodes in the instance. Each shard has the following bandwidth limits:
  - Community Edition: If the memory of each shard is less than or equal to 1 GB, the maximum bandwidth of each shard is 48 Mbit/s. If the memory of each shard is greater than 1 GB, the maximum bandwidth of each shard is 96 Mbit/s.
  - Enhanced Edition (Tair): The maximum bandwidth of each shard is 96 Mbit/s.
- If the default proxy endpoint is used by a cluster instance, the upper limit of bandwidth for the cluster instance is 2,048 Mbit/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards to the cluster instance.

  To handle network traffic peaks, you can enable the direct connection mode. For more information, see Enable the direct connection mode. The direct connection mode is applicable only to cluster instances. After you enable the direct connection mode, take note of the following items:
  - The upper limit of connections is equal to the maximum number of connections for a single shard multiplied by the number of shards. The maximum number of connections for a single shard in Community Edition is 10,000. The maximum number of connections for a single shard in Enhanced Edition (Tair) is 30,000.
  - The upper limit of the bandwidth is equal to the maximum bandwidth of a single shard multiplied by the number of shards. For example, if a cluster instance contains 128 shards and each shard is allocated with a memory of more than 1 GB and a maximum bandwidth of 96 Mbit/s, the bandwidth limit of the instance is 12,288 Mbit/s after you enable the direct connection mode.
- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 Mbit/s, the upstream and downstream bandwidths of the instance are

both 10 Mbit/s.

> **Note** If your instance may encounter unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance. For more information about, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding tables is the internal bandwidth of ApsaraDB for Redis instances. The public bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

Calculation rules for connections

Item	Description
Maximum number of concurrent connections	• : The maximum number of connections to an instance is 500,000. If the upper limit of 500,000 connections is reached, subsequent connections cannot be established even if you add more shards or nodes. Proxy mode • : The maximum number of connections for a single shard is 30,000. The maximum number of connections for an instance = Number of shards × 30,000. Direct connection mode
Number of new connections per second	The upper limit of the number of new connections per second refers to the number of connections that are allowed to be established per second. For example, assume that the maximum number of new connections to an instance per second is 10,000, the maximum number of connections to the instance is 50,000, and the actual number of connections at the Nth second after the instance starts to run is 12,000. In this case, at the (N+1)th second, the maximum number of connections is 22,000. This value is calculated by using the following formula: 12,000 + 10,000. **Note** If the instance is connected in direct connection mode, connection performance degrades because proxy nodes are bypassed. For more information about the direct connection mode, see Enable the direct connection mode. The maximum number of new connections per second for a single data shard is 2,000. For example, if an instance has four data shards, the maximum number of new connections per second is 8,000.

FAQ

- Q: Why is the actual bandwidth of my instance different from that described in this topic?

  A: The bandwidths of specific instance types may be adjusted after the service is upgraded. If the actual bandwidth of your instance is different from that described in this topic, you can change the configurations of your instance by selecting the same instance type to update the bandwidth. The price for your instance remains unchanged after the configuration change. For more information, see Change the configurations of an instance.

- Q: Why am I unable to write data to a cluster instance when instance memory is not exhausted?

  A: This is because the instance contains exhausted data shards. ApsaraDB for Redis uses the hash algorithm to evenly write data to different data shards. If the instance contains large keys, resources are unevenly distributed in the instance and the data shards that house these large keys may even be exhausted. In this case, part of instance write operations may fail.

You can use the **Performance Monitor** feature to check and optimize the performance of each data shard. For more information, see How do I view the memory of child instances of an ApsaraDB for Redis cluster instance?

5.3.3. Performance-enhanced read/write splitting instances

This topic describes the specifications of read/write splitting instances of ApsaraDB for Redis Enterprise Edition (Tair). These specifications include the number of read replicas, memory capacity, maximum number of connections to each instance, maximum bandwidth, and queries per second (QPS) reference value.

Instance types

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Number of I/O threads	Number of read replicas	Maximum bandwidth value (MB/s)	Maximum number of new connections per second	Maximum number of connections	QPS reference value
1 GB performance-enhanced read/write splitting instance (1 shard, 1 read replica)	redis.amber.logic.splitrw.small.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	480,000	480,000
2 GB performance-enhanced read/write splitting instance (1 shard, 1 read replica)	redis.amber.logic.splitrw.mid.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	480,000	480,000
4 GB performance-enhanced read/write splitting instance (1 shard, 1 read replica)	redis.amber.logic.splitrw.stand.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	480,000	480,000

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Number of I/O threads	Number of read replicas	Maximum bandwidth value (MB/s)	Maximum number of new connections per second	Maximum number of connections	QPS reference value
8 GB performance-enhanced read/write splitting instance (1 shard, 1 read replica)	redis.amber.logic.splitrw.large.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	480,000	480,000
16 GB performance-enhanced read/write splitting instance (1 shard, 1 read replica)	redis.amber.logic.splitrw.2xlarge.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	480,000	480,000
32 GB performance-enhanced read/write splitting instance (1 shard, 1 read replica)	redis.amber.logic.splitrw.4xlarge.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	480,000	480,000
64 GB performance-enhanced read/write splitting instance (1 shard, 1 read replica)	redis.amber.logic.splitrw.8xlarge.1db.1rodb.6proxy.multithread	12	4	1	192	20,000	480,000	480,000
1 GB performance-enhanced read/write splitting instance (1 shard, 3 read replicas)	redis.amber.logic.splitrw.small.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	480,000	960,000

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Number of I/O threads	Number of read replicas	Maximum bandwidth value (MB/s)	Maximum number of new connections per second	Maximum number of connections	QPS reference value
2 GB performance-enhanced read/write splitting instance (1 shard, 3 read replicas)	redis.amber.logic.splitrw.mid.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	480,000	960,000
4 GB performance-enhanced read/write splitting instance (1 shard, 3 read replicas)	redis.amber.logic.splitrw.stand.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	480,000	960,000
8 GB performance-enhanced read/write splitting instance (1 shard, 3 read replicas)	redis.amber.logic.splitrw.large.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	480,000	960,000
16 GB performance-enhanced read/write splitting instance (1 shard, 3 read replicas)	redis.amber.logic.splitrw.2xlarge.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	480,000	960,000
32 GB performance-enhanced read/write splitting instance (1 shard, 3 read replicas)	redis.amber.logic.splitrw.4xlarge.1db.3rodb.12proxy.multithread	24	4	3	384	40,000	480,000	960,000

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Number of I/O threads	Number of read replicas	Maximum bandwidth value (MB/s)	Maximum number of new connections per second	Maximum number of connections	QPS reference value
64 GB performance-enhanced read/write splitting instance (1 shard, 3 read replicas)	redis.amber.logic.splitrw.8xlarge.1db.3rodb.12proxy.multithread	24	4	3	384	40000	480,000	960,000
1 GB performance-enhanced read/write splitting instance (1 shard, 5 read replicas)	redis.amber.logic.splitrw.small.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
2 GB performance-enhanced read/write splitting instance (1 shard, 5 read replicas)	redis.amber.logic.splitrw.mid.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
4 GB performance-enhanced read/write splitting instance (1 shard, 5 read replicas)	redis.amber.logic.splitrw.stand.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
8 GB performance-enhanced read/write splitting instance (1 shard, 5 read replicas)	redis.amber.logic.splitrw.large.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Number of I/O threads	Number of read replicas	Maximum bandwidth value (MB/s)	Maximum number of new connections per second	Maximum number of connections	QPS reference value
16 GB performance-enhanced read/write splitting instance (1 shard, 5 read replicas)	redis.amber.logic.splitrw.2xlarge.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
32 GB performance-enhanced read/write splitting instance (1 shard, 5 read replicas)	redis.amber.logic.splitrw.4xlarge.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000
64 GB performance-enhanced read/write splitting instance (1 shard, 5 read replicas)	redis.amber.logic.splitrw.8xlarge.1db.5rodb.18proxy.multithread	36	4	5	576	50,000	480,000	1,440,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth values

- The bandwidth value in the preceding table is the bandwidth of the instance. The bandwidth value is the sum of the bandwidth of all shards or nodes in the instance.
- The upper limit of the total bandwidth for a read/write splitting instance is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards or nodes to the read/write splitting instance.
- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidths of the instance are both 10 MB/s.

> **Note** If your instance has unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance as needed. For more information, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

Calculation rules for connections

The following example shows how the maximum number of new connections per second affects the maximum number of connections at a specific second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections of the instance is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the maximum number of connections is 22,000, which is calculated by using the following formula: 12,000 + 10,000.

5.4. Community Edition with cloud disks

This topic describes the specifications of standard instances of ApsaraDB for Redis Community Edition that use cloud disks. These specifications include the memory capacity, maximum number of connections to each instance, maximum bandwidth, and queries per second (QPS) reference value.

Performance

The specifications in this topic apply to instances in the standard architecture or cluster architecture that use cloud disks.

Architecture	Performance
Standard instances	The overall performance of standard instances is as described in the instance specifications table.
Cluster instances	The overall performance equals the number of shards multiplied by the performance of each shard. For example, a cluster instance has four shards of the redis.shard.small.ce type, which is the first instance type in Instance types. The following items show the performance of each shard: • CPU cores: 2 cores • Bandwidth: 24 MB/s • Maximum number of connections: 10,000 • QPS reference value: 100,000. The following items show the overall performance of the cluster instance: • CPU cores: 8 cores • Bandwidth: 96 MB/s • Maximum number of connections: 40,000 • QPS reference value: 400,000.

Instance types

> **Note**
>
> - Enhanced solid-state disks (ESSDs) are used only to store system operating data, such as logs and backup data. ESSDs are not used as the media to read or write data.
> - You can adjust the number of shards in a cluster instance that uses cloud disks. Such a cluster instance can contain 3 to 32 shards with the same specifications. For more information, see Adjust the number of shards for an 云盘ApsaraDB for Redis instance.

Instance types

Instance type	InstanceClass that is used in API operations	Number of CPU cores	Memory (GB)	ESSDs (GB)	Maximum bandwidth value (MB/s)	Maximum number of connections	QPS reference value
1 GB instance that uses shared vCPUs	redis.shard.small.ce	2	1	5	24	10,000	100,000
2 GB instance that uses shared vCPUs	redis.shard.mid.ce	2	2	10	24	10,000	100,000
4 GB instance that uses shared vCPUs	redis.shard.large.ce	2	4	20	32	10,000	100,000
8 GB instance that uses shared vCPUs	redis.shard.xlarge.ce	2	8	40	40	10,000	100,000
16 GB instance that uses shared vCPUs	redis.shard.2xlarge.ce	2	16	80	80	10,000	100,000
24 GB instance that uses shared vCPUs	redis.shard.3xlarge.ce	2	24	120	96	10,000	100,000
32 GB instance that uses shared vCPUs	redis.shard.4xlarge.ce	2	32	160	96	10,000	100,000
64 GB instance that uses shared vCPUs	redis.shard.8xlarge.ce	2	64	320	96	10,000	100,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth

- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 Mbit/s, the upstream and downstream bandwidths of the instance are both 10 Mbit/s.

- The bandwidth value in the preceding tables is the internal bandwidth of ApsaraDB for Redis instances. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between an ApsaraDB for Redis instance and a client. To ensure higher security and lower network latency, we recommend that you connect to an ApsaraDB for Redis instance over a virtual private cloud (VPC). For more information about VPCs, see What is a VPC?

Calculation rules for connections

The following example shows how the maximum number of new connections per second affects the maximum number of connections at a specific second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections of the instance is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the maximum number of connections is 22,000, which is calculated by using the following formula: 12,000 + 10,000.

FAQ

Q: Why is the actual bandwidth of my instance different from that described in this topic?

A: The bandwidth of specific instance types may be adjusted after the service is upgraded. If the maximum number of connections of your instance is different from that described in this topic, change your instance specifications to update the bandwidth. Make sure that you select the same instance type when you change the specifications. For more information, see Change the configurations of an instance.

5.5. Cloud Disk Enhanced Edition

5.5.1. Performance-enhanced instances

This topic describes the specifications of performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) that use cloud disks. These specifications include the memory capacity, maximum number of connections to each instance, maximum bandwidth, and queries per second (QPS) reference value.

> **Note** For more information about instances that use local disks and those that use cloud disks, see Comparison between ApsaraDB for Redis instances that use local disks and those that use cloud disks.

Performance

The specifications in this topic apply to performance-enhanced instances in the standard architecture or cluster architecture.

Architecture	Performance
Standard	The overall performance of standard instances is as described in the instance types table.

Architecture	Performance
Cluster	The overall performance of cluster instances equals the number of shards multiplied by the performance of each shard specified by shard specifications. **Note** When a cluster instance that runs in proxy mode is accessed, the maximum bandwidth is 2,048 Mbit/s, the maximum total QPS is 10,000,000, and the maximum number of connections is 500,000. To handle ultra-high network traffic for your business, you can enable the direct connection mode. For more information, see Enable the direct connection mode. For example, a cluster instance has four shards of the tair.rdb.256m type. This type is the first instance type in the Instance types table. The following items show the performance of each shard: • CPU cores: 6 cores • Bandwidth: 96 Mbit/s • Maximum number of connections: 30,000 The following items show the overall performance of the cluster instance: • CPU cores: 24 cores • Bandwidth: 384 Mbit/s • Maximum number of connections: 120,000

Instance types

Note Enhanced SSDs (ESSDs) are used only to store system operating data, such as logs and backup data. ESSDs are not used as the media to read or write data. For more information about ESSDs, see ESSDs.

Instance types

Instance type	InstanceClass value (used in API operations)	Number of cores	Memory (GB)	ESSDs (GB)	Number of I/O threads	Maximum bandwidth (Mbit/s)	Number of new connections per second	Maximum number of connections	QPS reference value
256 256 MB performance-enhanced instance	tair.rdb.256m	6	0.25	1.25	4	96	30,000	30,000	300,000

Instance type	InstanceClass value (used in API operations)	Number of cores	Memory (GB)	ESSDs (GB)	Number of I/O threads	Maximum bandwidth (Mbit/s)	Number of new connections per second	Maximum number of connections	QPS reference value
1 GB performance-enhanced instance	tair.rdb.1g	6	1	5	4	96	30,000	30,000	300,000
2 GB performance-enhanced instance	tair.rdb.2g	6	2	10	4	96	30,000	30,000	300,000
4 GB performance-enhanced instance	tair.rdb.4g	6	4	20	4	96	30,000	30,000	300,000
8 GB performance-enhanced instance	tair.rdb.8g	6	8	40	4	96	30,000	30,000	300,000
16 GB performance-enhanced instance	tair.rdb.16g	6	16	80	4	96	30,000	30,000	300,000
24 GB performance-enhanced instance	tair.rdb.24g	6	24	120	4	96	30,000	30,000	300,000
32 GB performance-enhanced instance	tair.rdb.32g	6	32	160	4	96	30,000	30,000	300,000
64 GB performance-enhanced instance	tair.rdb.64g	6	64	320	4	96	30,000	30,000	300,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth

- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 Mbit/s, the upstream and downstream bandwidths of the instance are both 10 Mbit/s.
- The bandwidth value in the preceding tables is the internal bandwidth of ApsaraDB for Redis instances. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between an ApsaraDB for Redis instance and a client. To ensure higher security and lower network latency, we recommend that you connect to an ApsaraDB for Redis instance over a virtual private cloud (VPC). For more information about VPCs, see What is a VPC?

Notes on connections

The maximum number of new connections per second affects the maximum number of connections at a specific second. For example, assume that the maximum number of new connections per second of an instance is 10,000 and the maximum number of connections of the instance is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. In this case, at the (N+1)th second, the maximum number of connections is 22,000. This value is calculated by using the following formula: 12,000 + 10,000.

Related information

- Performance-enhanced instances
- Standard master-replica instances

5.5.2. Persistent memory-optimized instances

This topic describes the specifications of persistent memory-optimized instances of the ApsaraDB for Redis Enhanced Edition (Tair), including the instances in the standard or cluster architecture. These specifications include the memory capacity, maximum number of connections, and maximum bandwidth.

Instance purchase

Create Redis企业版云盘实例a persistent memory-optimized or storage-optimized instance of ApsaraDB for Redis Enhanced Edition (Tair)

Terms

persistent memory-optimized instance

Persistent memory-optimized instances adopt Intel Optane DC persistent memory (AEP) to support large-capacity in-memory databases that are compatible with open source Redis. A persistent memory-optimized instance persists each operation and does not use disks to implement data persistence. Compared with an ApsaraDB for Redis Community Edition instance, it reduces costs by up to 30% and delivers almost the same throughput and latency. This improves the reliability of business data. For more information, see Persistent memory-optimized instances.

ESSD

Enhanced SSDs (ESSDs) are disks provided by Alibaba Cloud. For more information about the terms and performance levels of ESSDs, see ESSDs.

> **Note** ESSDs are used only to store system operating data, such as logs, and backup data and are not used to read or write data.

Instance types

The following table lists the specifications of a single shard. The performance of a cluster instance can be calculated by using the following formula: Performance of a cluster instance = Number of shards in the instance × Performance of each shard specified by instance specifications.

Instance type	InstanceClass value (used in API operations)	Specification		Maximum number of connections	Maximum bandwidth value (Mbit/s)	QPS reference value
		Number of CPU cores	Persistent memory (GB)			
Master-replica 4 GB persistent memory	tair.scm.standard.1m.4d	3	4	10,000	96	100,000
Master-replica 8 GB persistent memory	tair.scm.standard.2m.8d	3	8	10,000	96	100,000
Master-replica 16 GB persistent memory	tair.scm.standard.4m.16d	3	16	10,000	96	100,000
Master-replica 32 GB persistent memory	tair.scm.standard.8m.32d	3	32	10,000	96	100,000
Master-replica 64 GB persistent memory	tair.scm.standard.16m.64d	3	64	10,000	96	100,000

Performance

The specifications described in this topic apply to persistent memory-optimized instances in the standard or cluster architecture. For more information about the standard architecture and the cluster architecture, see Standard master-replica instances and Cluster master-replica instances.

Architecture	Performance
Standard	The performance of standard instances is as described in the instance specifications table.

Architecture	Performance
Cluster	The performance of cluster instances is calculated by using the following formula: Performance of a cluster instance = Number of shards in the instance × Performance of each shard specified by instance specifications. **Note** When a cluster instance that runs in proxy mode is accessed, the maximum bandwidth is 2,048 Mbit/s, the maximum total QPS is 10,000,000, and the maximum number of connections is 500,000. After one of the upper limits is reached, the instance performance cannot be improved even if you add more shards to the cluster instance. To handle ultra-high network traffic for your business, you can enable the direct connection mode. For more information, see Enable the direct connection mode. For example, a cluster instance has four shards of the tair.scm.standard.2m.8d type. This type is the second instance type in Instance types. The following items show the performance of each shard: • CPU cores: 3 cores • Bandwidth: 96 Mbit/s • Maximum number of connections: 30,000 The following items show the performance of the cluster instance: • CPU cores: 12 cores • Bandwidth: 384 Mbit/s • Maximum number of connections: 120,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Related information

- Performance test for persistent memory-optimized instances
- Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair)

5.5.3. Storage-optimized instances

This topic describes the specifications of standard storage-optimized instances of ApsaraDB for Redis Enhanced Edition (Tair), such as the memory capacity, the number of connections, and the bandwidth.

Instance purchase

Create Redis企业版云盘实例a persistent memory-optimized or storage-optimized instance of ApsaraDB for Redis Enhanced Edition (Tair)

Terms

storage-optimized instance

Storage-optimized instances of ApsaraDB for Redis Enhanced Edition (Tair) are developed based on enhanced SSDs (ESSDs) and compatible with core data structures and interfaces of open source Redis. These instances can provide large-capacity, low-cost, and persistent database services.

Storage-optimized instances reduce costs and improve data reliability. In addition, storage-optimized instances reduce the amount of reserved memory that is required for the forks of native Redis databases. These instances apply to scenarios that store warm and cold data, and that require compatibility with open source Redis, large capacity, and high access performance. For more information, see Storage-optimized instances.

ESSD

Alibaba Cloud ESSDs are ultra-high performance disks based on the next-generation distributed block storage architecture. For more information about the related terms and performance levels of ESSDs, see ESSDs.

ECS

Elastic Compute Service (ECS) is a high-performance, stable, reliable, and scalable IaaS-level cloud computing service provided by Alibaba Cloud. For more information, see What is ECS? Storage-optimized instances of ApsaraDB for Redis use g6e, a general-purpose ECS instance family with enhanced performance. This instance family offloads a large number of virtualization features to dedicated hardware with the use of the third-generation X-Dragon architecture to provide predictable and consistent ultra-high performance and reduce virtualization overheads. This instance family utilizes the fast path acceleration feature of chips to improve storage performance, network performance, and computing stability by an order of magnitude.

For more information about the instance families of ECS, see Instance families.

Instance types

InstanceClass value (used in API operations)	Specifications			System disk		Corresponding ECS instance type	Maximum connections	Maximum bandwidth (Gbit/s)
	CPU cores	Memory (GB)	Range of storage capacity (GB)	ESSD capacity (GB)	ESSD performance level			
tair.essd.standard.xlarge	4	16	60~130	40	PL1	ecs.g6e.xlarge	10,000	Burstable up to 0.625
tair.essd.standard.2xlarge	8	32	60~260	40	PL1	ecs.g6e.2xlarge	10,000	Burstable up to 0.625
tair.essd.standard.4xlarge	16	64	60~530	40	PL1	ecs.g6e.4xlarge	30,000	Burstable up to 0.625
tair.essd.standard.8xlarge	32	128	60~1030	40	PL1	ecs.g6e.8xlarge	30,000	Burstable up to 0.625
tair.essd.standard.13xlarge	52	192	60~1540	40	PL1	ecs.g6e.13xlarge	30,000	Burstable up to 0.625

> **Note**
>
> - Only standard master-replica instances are supported. For more information, see Standard master-replica instances.
> - The underlying operating system and management service of a standard storage-optimized instance of ApsaraDB for Redis Enhanced Edition (Tair) take up memory space. For this reason, the actual available memory of the instance is 75% to 90% of the memory size specified by the instance specifications.

Related information

- Performance test for storage-optimized instances
- Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair)

5.6. Phased-out specifications

5.6.1. Read/write splitting cluster instances (retired)

This topic describes the specifications of read/write splitting cluster instances of ApsaraDB for Redis Community Edition (Tair). The specifications include the memory capacity, number of connections, maximum bandwidth, and queries per second (QPS) reference value.

Instance types

> **Note** In the following table, two shards indicate that the instance types contain two data shards. For example, **4 GB read/write splitting instance (2 shards, 1 read replica)** indicates that the instance has two data shards. Each data shard contains one read replica.

Instance type	InstanceClass value (used in API operations)	Number of CPU cores	Number of shards	Number of read replicas per shard	Maximum bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of connections	QPS reference value
4 GB read/write splitting instance (2 shards, 1 read replica)	redis.logic.splitrw.sharding2g.2db.1rodb.4proxy.default	8	2	1	384	40,000	40,000	400,000

Instance type	InstanceClass value (used in API operations)	Number of CPU cores	Number of shards	Number of read replicas per shard	Maximum bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of connections	QPS reference value
8 GB read/write splitting instance (2 shards, 1 read replica)	redis.logic.splitrw.sharding4g.2db.1rodb.4proxy.default	8	2	1	384	40,000	40,000	400,000
16 GB read/write splitting instance (2 shards, 1 read replica)	redis.logic.splitrw.sharding8g.2db.1rodb.8proxy.default	8	2	1	384	40,000	40,000	400,000
32 GB read/write splitting instance (8 shards, 1 read replica)	redis.logic.splitrw.sharding4g.8db.1rodb.16proxy.default	32	8	1	1,536	50,000	160,000	1,600,000
64 GB read/write splitting instance (16 shards, 1 read replica)	redis.logic.splitrw.sharding4g.16db.1rodb.32proxy.default	64	16	1	2,048	50,000	320,000	3,200,000
128 GB read/write splitting instance (16 shards, 1 read replica)	redis.logic.splitrw.sharding8g.16db.1rodb.32proxy.default	64	16	1	2,048	50,000	320,000	3,200,000
256 GB read/write splitting instance (32 shards, 1 read replica)	redis.logic.splitrw.sharding8g.32db.1rodb.64proxy.default	128	32	1	2,048	50,000	500,000	6,400,000

Instance type	InstanceClass value (used in API operations)	Number of CPU cores	Number of shards	Number of read replicas per shard	Maximum bandwidth (MB/s)	Maximum number of new connections per second	Maximum number of connections	QPS reference value
512 GB read/write splitting instance (32 shards, 1 read replica)	redis.logic.splitrw.sharding16g.32db.1rodb.64proxy.default	128	32	1	2,048	50,000	500,000	6,400,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth values

- The bandwidth value in the preceding table is the bandwidth of the instance. The bandwidth value is the sum of the bandwidth of all shards or nodes in the instance.
- The upper limit of the total bandwidth for a read/write splitting instance is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards or nodes to the read/write splitting instance.
- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidths of the instance are both 10 MB/s.

  > **Note** If your instance has unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance as needed. For more information, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

Calculation rules for connections

The following example shows how the maximum number of new connections per second affects the maximum number of connections at a specific second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections of the instance is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the maximum number of connections is 22,000, which is calculated by using the following formula: 12,000 + 10,000.

5.6.2. Hybrid-storage standard instances (retired)

This topic describes the specifications of hybrid-storage standard instances of ApsaraDB for Redis Enhanced Edition (Tair). These specifications include the memory capacity, number of connections, maximum bandwidth, and queries per second (QPS) reference value.

> **Note** Hybrid-storage instances are discontinued. For more information, see Sales of ApsaraDB for Redis hybrid-storage instances are discontinued. We recommend that you use performance-enhanced instances that provide higher performance and more features.
>
> If you have purchased s hybrid-storage instance, you can submit a ticket to migrate the data of the instance.

Instance types

Instance type	InstanceClass value (used in API operations)	Number of CPU cores	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth (MB/s)	QPS reference value
Master-replica instance with 16 GB memory and 32 GB disk storage	redis.amber.master.16g.2x.ext4.default	6	10,000	50,000	48	40,000
Master-replica instance with 16 GB memory and 64 GB disk storage	redis.amber.master.16g.4x.ext4.default	6	10,000	50,000	48	40,000
Master-replica instance with 16 GB memory and 128 GB disk storage	redis.amber.master.16g.8x.ext4.default	6	10,000	50,000	48	40,000
Master-replica instance with 32 GB memory and 64 GB disk storage	redis.amber.master.32g.2x.ext4.default	6	10,000	50,000	48	40,000
Master-replica instance with 32 GB memory and 128 GB disk storage	redis.amber.master.32g.4x.ext4.default	6	10,000	50,000	48	40,000
Master-replica instance with 32 GB memory and 256 GB disk storage	redis.amber.master.32g.8x.ext4.default	6	10,000	50,000	48	40,000

Instance type	InstanceClass value (used in API operations)	Number of CPU cores	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth (MB/s)	QPS reference value
Master-replica instance with 64 GB memory and 128 GB disk storage	redis.amber.master.64g.2x.ext4.default	6	10,000	50,000	48	40,000
Master-replica instance with 64 GB memory and 256 GB disk storage	redis.amber.master.64g.4x.ext4.default	6	10,000	50,000	48	40,000
Master-replica instance with 64 GB memory and 512 GB disk storage	redis.amber.master.64g.8x.ext4.default	6	10,000	50,000	48	40,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth values

- The bandwidth value in the preceding table is the bandwidth of the instance. The bandwidth value is the sum of the bandwidth of all shards or nodes in the instance.
- The upper limit of the total bandwidth for a read/write splitting instance is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards or nodes to the read/write splitting instance.
- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidths of the instance are both 10 MB/s.

  > **Note** If your instance has unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance as needed. For more information, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

Calculation rules for connections

The following example shows how the maximum number of new connections per second affects the maximum number of connections at a specific second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections of the instance is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the maximum number of connections is 22,000, which is calculated by using the following formula: 12,000 + 10,000.

5.6.3. Hybrid-storage cluster instances (retired)

This topic describes the specifications of hybrid-storage cluster instances of ApsaraDB for Redis Enhanced Edition (Tair). These specifications include the memory and disk capacities, number of connections, maximum bandwidth, and queries per second (QPS) reference value.

> **Note** Hybrid-storage instances are discontinued. For more information, see Sales of ApsaraDB for Redis hybrid-storage instances are discontinued. We recommend that you use performance-enhanced instances that provide higher performance and more features.
>
> If you have purchased s hybrid-storage instance, you can submit a ticket to migrate the data of the instance.

Instance types

Instance type	InstanceClass value (used in API operations)	Number of CPU cores	Number of shards	Maximum number of new connections per second	Maximum number of connections ( )direct connection mode	Maximum number of connections ( )proxy mode	Maximum bandwidth (MB/s)	QPS reference value
64 GB memory and 256 GB disk storage (4 shards)	redis.amber.sharding.16g.4db.0rodb.12proxy.4x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
64 GB memory and 512 GB disk storage (4 shards)	redis.amber.sharding.16g.4db.0rodb.12proxy.8x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
128 GB memory and 512 GB disk storage (4 shards)	redis.amber.sharding.32g.4db.0rodb.12proxy.4x.ext4.default	24	4	40,000	120,000	500,000	384	120,000

Instance type	InstanceClass value (used in API operations)	Number of CPU cores	Number of shards	Maximum number of new connections per second	Maximum number of connections ( )direct connection mode	Maximum number of connections ( )proxy mode	Maximum bandwidth (MB/s)	QPS reference value
128 GB memory and 1,024 GB disk storage (4 shards)	redis.amber.sharding.32g.4db.0rodb.12proxy.8x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
256 GB memory and 1,024 GB disk storage (4 shards)	redis.amber.sharding.64g.4db.0rodb.12proxy.4x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
256 GB memory and 2,048 GB disk storage (4 shards)	redis.amber.sharding.64g.4db.0rodb.12proxy.8x.ext4.default	24	4	40,000	120,000	500,000	384	120,000
128 GB memory and 512 GB disk storage (8 shards)	redis.amber.sharding.16g.8db.0rodb.24proxy.4x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
128 GB memory and 1,024 GB disk storage (8 shards)	redis.amber.sharding.16g.8db.0rodb.24proxy.8x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
256 GB memory and 1,024 GB disk storage (8 shards)	redis.amber.sharding.32g.8db.0rodb.24proxy.4x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
256 GB memory and 2,048 GB disk storage (8 shards)	redis.amber.sharding.32g.8db.0rodb.24proxy.8x.ext4.default	48	8	50,000	240,000	500,000	768	240,000

Instance type	InstanceClass value (used in API operations)	Number of CPU cores	Number of shards	Maximum number of new connections per second	Maximum number of connections ( )direct connection mode	Maximum number of connections ( )proxy mode	Maximum bandwidth (MB/s)	QPS reference value
512 GB memory and 2,048 GB disk storage (8 shards)	redis.amber.sharding.64g.8db.0rodb.24proxy.4x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
512 GB memory and 4,096 GB disk storage (8 shards)	redis.amber.sharding.64g.8db.0rodb.24proxy.8x.ext4.default	48	8	50,000	240,000	500,000	768	240,000
256 GB memory and 1,024 GB disk storage (16 shards)	redis.amber.sharding.16g.16db.0rodb.48proxy.4x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000
256 GB memory and 2,048 GB disk storage (16 shards)	redis.amber.sharding.16g.16db.0rodb.48proxy.8x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000
512 GB memory and 2,048 GB disk storage (16 shards)	redis.amber.sharding.32g.16db.0rodb.48proxy.4x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000
512 GB memory and 4,096 GB disk storage (16 shards)	redis.amber.sharding.32g.16db.0rodb.48proxy.8x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000
1,024 GB memory and 4,096 GB disk storage (16 shards)	redis.amber.sharding.64g.16db.0rodb.48proxy.4x.ext4.default	96	16	50,000	480,000	500,000	1,536	480,000

Instance type	InstanceClass value (used in API operations)	Number of CPU cores	Number of shards	Maximum number of new connections per second	Maximum number of connections ( )direct connection mode	Maximum number of connections ( )proxy mode	Maximum bandwidth (MB/s)	QPS reference value
1,024 GB memory and 8,192 GB disk storage (16 shards)	redis.amber.sharding.64g.16db.0rodb.48proxy.8x.ext4.default	96	16	50,000	480,000	500,000	1536	480,000

Notes about CPU cores

To ensure service stability, the system reserves a CPU core to process . In a cluster instance or a read/write splitting instance, the system reserves a CPU core for each data shard or each read replica to process background tasks. background tasks

Calculation rules for bandwidth values

- The bandwidth value in the preceding tables is the bandwidth of the instance. The bandwidth value is the sum of the bandwidths of all shards or nodes in the instance. Each shard has the following bandwidth limits:
  - Community Edition: If the memory of each shard is less than or equal to 1 GB, the maximum bandwidth of each shard is 48 Mbit/s. If the memory of each shard is greater than 1 GB, the maximum bandwidth of each shard is 96 Mbit/s.
  - Enhanced Edition (Tair): The maximum bandwidth of each shard is 96 Mbit/s.
- If the default proxy endpoint is used by a cluster instance, the upper limit of bandwidth for the cluster instance is 2,048 Mbit/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards to the cluster instance.

  To handle network traffic peaks, you can enable the direct connection mode. For more information, see Enable the direct connection mode. The direct connection mode is applicable only to cluster instances. After you enable the direct connection mode, take note of the following items:
  - The upper limit of connections is equal to the maximum number of connections for a single shard multiplied by the number of shards. The maximum number of connections for a single shard in Community Edition is 10,000. The maximum number of connections for a single shard in Enhanced Edition (Tair) is 30,000.
  - The upper limit of the bandwidth is equal to the maximum bandwidth of a single shard multiplied by the number of shards. For example, if a cluster instance contains 128 shards and each shard is allocated with a memory of more than 1 GB and a maximum bandwidth of 96 Mbit/s, the bandwidth limit of the instance is 12,288 Mbit/s after you enable the direct connection mode.
- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 Mbit/s, the upstream and downstream bandwidths of the instance are both 10 Mbit/s.

> **Note** If your instance may encounter unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance. For more information about, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding tables is the internal bandwidth of ApsaraDB for Redis instances. The public bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

Calculation rules for connections

Item	Description
Maximum number of concurrent connections	• : The maximum number of connections to an instance is 500,000. If the upper limit of 500,000 connections is reached, subsequent connections cannot be established even if you add more shards or nodes. Proxy mode • : The maximum number of connections for a single shard is 10,000. The maximum number of connections for an instance can be calculated by using the following formular: Number of shards × 10,000. Direct connection mode
Number of new connections per second	The upper limit of the number of new connections per second refers to the number of connections that are allowed to be established per second. For example, assume that the maximum number of new connections to an instance per second is 10,000, the maximum number of connections to the instance is 50,000, and the actual number of connections at the Nth second after the instance starts to run is 12,000. In this case, at the (N+1)th second, the maximum number of connections is 22,000. This value is calculated by using the following formula: 12,000 + 10,000. **Note** If the instance is connected in direct connection mode, connection performance degrades because proxy nodes are bypassed. For more information about the direct connection mode, see Enable the direct connection mode. The maximum number of new connections per second for a single data shard is 2,000. For example, if an instance has four data shards, the maximum number of new connections per second is 8,000.

5.6.4. Retired instance types

This topic describes the ApsaraDB for Redis instances that are phased out. If you have purchased these instances, you can continue to use them. The following table lists the maximum number of connections, maximum bandwidth, and queries per second (QPS) reference value for these instances.

Standard zone-disaster recovery instances of Community Edition

Instance type	InstanceClass value (used in API operations)	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth (MB/s)	QPS reference value
1 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb1g.1db.0rodb.4proxy.default	10000	10000	10	80000
2 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb2g.1db.0rodb.4proxy.default	10000	10000	16	80000
4 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb4g.1db.0rodb.4proxy.default	10000	10000	24	80000
8 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb8g.1db.0rodb.4proxy.default	10000	10000	24	80000
16 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb16g.1db.0rodb.4proxy.default	10000	10000	32	80000
32 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb32g.1db.0rodb.4proxy.default	10000	10000	32	80000
64 GB zone-disaster recovery instance	redis.logic.sharding.drredissdb64g.1db.0rodb.4proxy.default	10000	10000	48	80000

Master-replica cluster instances of Community Edition

Instance type	InstanceClass value (used in API operations)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth (MB/s)	QPS reference value
1 GB cluster instance (2 shards)	redis.logic.sharding.512m.2db.0rodb.4proxy.default	2	20000	20000	48	200000

Instance type	InstanceClass value (used in API operations)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth (MB/s)	QPS reference value
1 GB cluster instance (4 shards)	redis.logic.sharding.256m.4db.0rodb.4proxy.default	4	40000	40000	96	400000
2 GB cluster instance (4 shards)	redis.logic.sharding.512m.4db.0rodb.4proxy.default	4	40000	40000	96	400000
2 GB cluster instance (8 shards)	redis.logic.sharding.256m.8db.0rodb.8proxy.default	8	50000	80000	192	800000
4 GB cluster instance (8 shards)	redis.logic.sharding.512m.8db.0rodb.8proxy.default	8	50000	80000	192	800000
4 GB cluster instance (16 shards)	redis.logic.sharding.256m.16db.0rodb.16proxy.default	16	50000	160000	384	1600000
8 GB cluster instance (16 shards)	redis.logic.sharding.512m.16db.0rodb.16proxy.default	16	50000	160000	384	1600000
8 GB cluster instance (32 shards)	redis.logic.sharding.256m.32db.0rodb.32proxy.default	32	50000	320000	768	3200000
16 GB cluster instance (32 shards)	redis.logic.sharding.512m.32db.0rodb.32proxy.default	32	50000	320000	768	3200000
16 GB cluster instance	redis.sharding.small.default	8	50000	80000	768	640000
32 GB cluster instance	redis.sharding.mid.default	8	50000	80000	768	640000
64 GB cluster instance	redis.sharding.large.default	8	50000	80000	768	640000

Instance type	InstanceClass value (used in API operations)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth (MB/s)	QPS reference value
128 GB cluster instance	redis.sharding.2xlarge.default	16	50000	160000	1536	1280000
256 GB cluster instance	redis.sharding.4xlarge.default	16	50000	160000	1536	1280000
512 GB cluster instance	redis.sharding.8xlarge.default	32	50000	320000	2048	2560000

Zone-disaster recovery cluster instances of Community Edition

Instance type	InstanceClass value (used in API operations)	Number of shards	Maximum number of new connections per second	Maximum number of connections	Maximum bandwidth (MB/s)	QPS reference value
16 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb16g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
32 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb32g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
64 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb64g.8db.0rodb.8proxy.default	8	50000	80000	768	640000
128 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb128g.16db.0rodb.16proxy.default	16	50000	160000	1536	1280000
256 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb256g.16db.0rodb.16proxy.default	16	50000	160000	1536	1280000
512 GB zone-disaster recovery cluster instance	redis.logic.sharding.drredismdb512g.32db.0rodb.32proxy.default	32	50000	320000	2048	2560000

Calculation rules for bandwidth values

- The bandwidth value in the preceding table is the bandwidth of the instance. The bandwidth value is the sum of the bandwidth of all shards or nodes in the instance.
- The upper limit of the total bandwidth for a read/write splitting instance is 2,048 MB/s. After the upper limit is reached, the bandwidth cannot be increased even if you add more shards or nodes to the read/write splitting instance.
- The bandwidth value applies to the upstream and downstream bandwidths. For example, if the bandwidth of an instance is 10 MB/s, the upstream and downstream bandwidths of the instance are both 10 MB/s.

  > **Note** If your instance has unexpected or scheduled traffic peaks, you can adjust the bandwidth of the instance as needed. For more information, see Adjust the bandwidth of an ApsaraDB for Redis instance.

- The bandwidth value in the preceding table is the internal bandwidth of the ApsaraDB for Redis instance. The Internet bandwidth is determined by the internal bandwidth and is limited by the bandwidth of the connection between the ApsaraDB for Redis instance and the client. We recommend that you connect to the instance over an internal network to maximize performance.

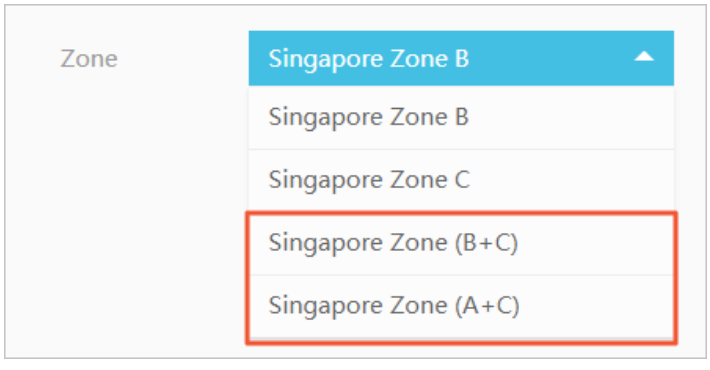
Calculation rules for connections

The following example shows how the maximum number of new connections per second affects the maximum number of connections at a specific second: The maximum number of new connections per second of an instance is 10,000. The maximum number of connections of the instance is 50,000. The actual number of connections at the Nth second after the instance starts to run is 12,000. Then, at the (N+1)th second, the maximum number of connections is 22,000, which is calculated by using the following formula: 12,000 + 10,000.

FAQ

Q: Zone-disaster recovery instances of ApsaraDB for Redis are phased out. How can I create a zone-disaster recovery instance when I use ApsaraDB for Redis?

A: Zone-disaster recovery instances of ApsaraDB for Redis are upgraded. To use the zone-disaster recovery feature, you can select a zone that supports zone-disaster recovery when you create an ApsaraDB for Redis instance.



6.Commands

6.1. Overview

ApsaraDB for Redis provides instances of multiple editions, series, and architectures. The commands supported by AparaDB for Redis instances vary based on their instance types. This topic describes the commands supported by ApsaraDB for Redis instances and the limits on the commands. You can refer to the topics in the following tables to view details of commands.

Supported commands

Topic	Description
Commands supported by ApsaraDB for Redis Community Edition	ApsaraDB for Redis Community Edition provides instances of multiple editions and architectures. The commands supported by the instances vary based on their instance types. This topic describes the native Redis commands supported by ApsaraDB for Redis Community Edition and provides the limits on these commands.
Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair)	ApsaraDB for Redis Enhanced Edition (Tair) instances support most commands of open source Redis. To ensure high performance, these instances have limits on the use of specific native Redis commands.
Commands supported by extended data structures of ApsaraDB for Redis Enhanced Edition (Tair)	ApsaraDB for Redis Enhanced Edition (Tair) instances support the commands that are supported by the Community Edition and the following new commands: • CAS and CAD commands • TairString • TairHash • TairGIS • TairBloom • TairDoc • TairTS • TairCpc • TairZset • TairRoaring • TairSearch Note • Performance-enhanced instances support all data structures listed in the preceding table. • Persistent memory-optimized instances support TairString (including CAS and CAD commands) and TairCpc.
Redis commands developed by Alibaba Cloud	ApsaraDB for Redis ensures compatibility with some native Redis commands and also supports specific Redis commands developed by Alibaba Cloud. You can use these commands to manage ApsaraDB for Redis cluster or read/write splitting instances.

Limits on the commands supported by different instance architectures

Topic	Description
Limits on commands supported by cluster instances	ApsaraDB for Redis cluster instances and standard instances are deployed in different architectures. The commands supported by the instances vary based on their instance types.
Limits on the commands supported by read/write splitting instances	ApsaraDB for Redis read/write splitting instances are classified into non-cluster and cluster read/write splitting instances. The commands supported by the instances vary based on their instance types.

6.2. Commands supported by ApsaraDB for Redis Community Edition

ApsaraDB for Redis Community Edition provides instances of multiple editions and architectures. The supported native Redis commands vary based on different instance types. This topic describes the native Redis commands supported by ApsaraDB for Redis Community Edition and provides the limits of these commands.

Precautions

- Read/write splitting instances do not support the **CLIENT ID** command and have limits on some commands. For more information, see Limits on the commands supported by read/write splitting instances.
- Cluster instances do not support the **SWAPDB**, **CLIENT ID**, or **SORT** (with the BY and GET options) command, and have limits on some commands. For more information, see Limits on commands supported by cluster instances.

To simplify management and O&M, cluster and read/write splitting instances support multiple commands that are developed by Alibaba Cloud. For more information, see Redis commands developed by Alibaba Cloud.

Symbols in the tables

This section describes the symbols that are used in the tables of this topic.

- ✔ indicates that this command is supported.
- ✘ indicates that this command is not supported.
- ✘ indicates that the command is not supported in open source Redis. For example, the **TOUCH** command is supported only by Redis 3.2.1 and later. This command is marked as ✘ in the Redis 2.8 column of the table.
- Footnote ①: If you want to run the command on a cluster instance, you must enable the direct connection mode to use a private endpoint to connect to the instance. For more information, see Use a private endpoint to connect to an ApsaraDB for Redis instance. You can also run the command if you use the endpoint of a proxy node to connect to an instance.
- Footnote ②: The **CONFIG SET** command returns only `OK`. The command does not modify the parameters. This ensures that the instance remains compatible with some client frameworks.

Note By default, commands in all command groups are supported by standard instances, cluster instances, and read/write splitting instances. For more information about instance architectures, see Overview.

Supported command groups

- Cluster command group
- Connection command group
- Geo command group
- Hashes command group
- HyperLogLog command group
- Keys command group
- Lists command group
- Pub and Sub command group
- Scripting command group
- Sentinel command group
- Server command group
- Sets command group
- Sorted sets command group
- Streams command group
- Strings command group
- Transaction command group

Cluster command group

Note

- The commands in the cluster command group are not supported by standard instances.
- However, if you use the endpoint of a proxy node to connect to an instance, some commands in the cluster command group are supported. These commands include **CLUSTER INFO**, **CLUSTER KEYSLOT**, **CLUSTER NODES**, **CLUSTER SLAVES**, and **CLUSTER SLOTS**.
- Since ApsaraDB for Redis 0.1.14, the READONLY and READWRITE commands are supported in ApsaraDB for Redis V5.0.

Command	Redis 2.8	Redis 4.0	Redis 5.0
CLUSTER ADDSLOTS	□	□	□
CLUSTER BUMPEPOCH	□	□	□
CLUSTER COUNT-FAILURE-REPORTS	□	□	□
CLUSTER COUNTKEYSINSLOT ①	□	□	□
CLUSTER DELSLOTS	□	□	□
CLUSTER FAILOVER	□	□	□

Command	Redis 2.8	Redis 4.0	Redis 5.0
CLUSTER FLUSHSLOTS	□	□	□
CLUSTER FORGET	□	□	□
CLUSTER GETKEYSINSLOT	□☺	□☺	□☺
CLUSTER INFO ①	✔☺	✔☺	✔☺
CLUSTER KEYSLOT ①	✔☺	✔☺	✔☺
CLUSTER MEET	□	□	□
CLUSTER MYID	□	□	□
CLUSTER NODES ①	✔☺	✔☺	✔☺
CLUSTER REPLICAS	□	□	□
CLUSTER REPLICATE	□	□	□
CLUSTER RESET	□	□	□
CLUSTER SAVECONFIG	□	□	□
CLUSTER SET-CONFIG-EPOCH	□	□	□
CLUSTER SETSLOT	□	□	□
CLUSTER SLAVES	□	□	□
CLUSTER SLOTS	✔☺	✔☺	✔☺
READONLY	□	□	✔☺
READWRITE	□	□	✔☺

Connection command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
AUTH	✔☺	✔☺	✔☺
CLIENT CACHING	□	□	□
CLIENT GETNAME	✔☺	✔☺	✔☺
CLIENT GETREDIR	□	□	□
CLIENT ID	□	□	✔☺
CLIENT KILL	✔☺	✔☺	✔☺

Command	Redis 2.8	Redis 4.0	Redis 5.0
CLIENT LIST	✔	✔	✔
CLIENT PAUSE	□	□	□
CLIENT REPLY	□	□	□
CLIENT SETNAME	✔	✔	✔
CLIENT TRACKING	□	□	□
CLIENT UNBLOCK	□	□	✔
ECHO	✔	✔	✔
HELLO	□	□	□
PING	✔	✔	✔
QUIT	✔	✔	✔
SELECT	✔	✔	✔

Geo command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
GEOADD	✔	✔	✔
GEODIST	✔	✔	✔
GEOHASH	✔	✔	✔
GEOPOS	✔	✔	✔
GEORADIUS	✔	✔	✔
GEORADIUSBYMEMBER	✔	✔	✔

Hashes command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
HDEL	✔	✔	✔
HEXISTS	✔	✔	✔
HGET	✔	✔	✔
HGETALL	✔	✔	✔
HINCRBY	✔	✔	✔

Command	Redis 2.8	Redis 4.0	Redis 5.0
HINCRBYFLOAT	✔	✔	✔
HKEYS	✔	✔	✔
HLEN	✔	✔	✔
HMGET	✔	✔	✔
HMSET	✔	✔	✔
HSCAN	✔	✔	✔
HSET	✔	✔	✔
HSETNX	✔	✔	✔
HSTRLEN	✔	✔	✔
HVALS	✔	✔	✔

HyperLogLog command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
PFADD	✔	✔	✔
PFCOUNT	✔	✔	✔
PFMERGE	✔	✔	✔

Keys command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
DEL	✔	✔	✔
DUMP	✔	✔	✔
EXISTS	✔	✔	✔
EXPIRE	✔	✔	✔
EXPIREAT	✔	✔	✔
KEYS	✔	✔	✔
MIGRATE	□	□	□
MOVE	✔	✔	✔
OBJECT	✔	✔	✔

Command	Redis 2.8	Redis 4.0	Redis 5.0
PERSIST	✔	✔	✔
PEXPIRE	✔	✔	✔
PEXPIREAT	✔	✔	✔
PTTL	✔	✔	✔
RANDOMKEY	✔	✔	✔
RENAME	✔	✔	✔
RENAMENX	✔	✔	✔
RESTORE	✔	✔	✔
SCAN	✔	✔	✔
SORT	✔	✔	✔
TOUCH	□	✔	✔
TTL	✔	✔	✔
TYPE	✔	✔	✔
UNLINK	□	✔	✔
WAIT	□	✔	✔

Note The WAIT command is not supported by proxy nodes of cluster instances. You can run the WAIT command by using the private endpoint of a cluster instance to connect to the instance.

Lists command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
BLPOP	✔	✔	✔
BRPOP	✔	✔	✔
BRPOPLPUSH	✔	✔	✔
LINDEX	✔	✔	✔
LINSERT	✔	✔	✔
LLEN	✔	✔	✔
LPOP	✔	✔	✔

Command	Redis 2.8	Redis 4.0	Redis 5.0
LPUSH	✔	✔	✔
LPUSHX	✔	✔	✔
LRANGE	✔	✔	✔
LREM	✔	✔	✔
LSET	✔	✔	✔
LTRIM	✔	✔	✔
RPOP	✔	✔	✔
RPOPLPUSH	✔	✔	✔
RPUSH	✔	✔	✔
RPUSHX	✔	✔	✔

Pub and Sub command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
PSUBSCRIBE	✔	✔	✔
PUBLISH	✔	✔	✔
PUBSUB	✔	✔	✔
PUNSUBSCRIBE	✔	✔	✔
SUBSCRIBE	✔	✔	✔
UNSUBSCRIBE	✔	✔	✔

Scripting command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
EVAL	✔	✔	✔
EVALSHA	✔	✔	✔
SCRIPT DEBUG	□	□	□
SCRIPT EXISTS	✔	✔	✔
SCRIPT FLUSH	✔	✔	✔
SCRIPT KILL	✔	✔	✔

Command	Redis 2.8	Redis 4.0	Redis 5.0
SCRIPT LOAD	✔	✔	✔

Sentinel command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
SENTINEL sentinels		✔	✔
SENTINEL get-master-addr-by-name		✔	✔

Server command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
ACL CAT			
ACL DELUSER			
ACL GENPASS			
ACL GETUSER			
ACL HELP			
ACL LIST			
ACL LOAD			
ACL LOG			
ACL SAVE			
ACL SETUSER			
ACL USERS			
ACL WHOAMI			
BGREWRITEAOF			
BGSAVE			
COMMAND	✔	✔	✔
COMMAND COUNT	✔	✔	✔
COMMAND GETKEYS	✔	✔	✔
COMMAND INFO	✔	✔	✔
CONFIG GET	✔	✔	✔

Command	Redis 2.8	Redis 4.0	Redis 5.0
CONFIG RESETSTAT	✔	✔	✔
CONFIG REWRITE	✘	✘	✘
CONFIG SET ②	✔	✔	✔
DBSIZE	✔	✔	✔
DEBUG OBJECT	✘	✘	✘
DEBUG SEGFAULT	✘	✘	✘
FLUSHALL	✔	✔	✔
FLUSHDB	✔	✔	✔
INFO	✔	✔	✔
LASTSAVE	✘	✘	✘
LATENCY DOCTOR	✔	✔	✔
LATENCY GRAPH	✔	✔	✔
LATENCY HELP	✘	✘	✔
LATENCY HISTORY	✔	✔	✔
LATENCY LATEST	✔	✔	✔
LATENCY RESET	✔	✔	✔
LOLWUT	✘	✘	✔
MEMORY DOCTOR	✘	✔	✔
MEMORY HELP	✘	✔	✔
MEMORY MALLOC-STATS	✘	✔	✔
MEMORY PURGE	✘	✔	✔
MEMORY STATS	✘	✔	✔
MEMORY USAGE	✘	✔	✔
MODULE LIST	✘	✘	✘
MODULE LOAD	✘	✘	✘
MODULE UNLOAD	✘	✘	✘
MONITOR	✔	✔	✔

Command	Redis 2.8	Redis 4.0	Redis 5.0
PSYNC	□	□	□
REPLICAOF	□	□	□
ROLE	□	✔☺	✔☺
SAVE	□	□	□
SHUTDOWN	□	□	□
SLAVEOF	□	□	□
SLOWLOG	✔☺	✔☺	✔☺
SWAPDB	□	✔☺	✔☺
SYNC	□	□	□
TIME	✔☺	✔☺	✔☺

Sets command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
SADD	✔☺	✔☺	✔☺
SCARD	✔☺	✔☺	✔☺
SDIFF	✔☺	✔☺	✔☺
SDIFFSTORE	✔☺	✔☺	✔☺
SINTER	✔☺	✔☺	✔☺
SINTERSTORE	✔☺	✔☺	✔☺
SISMEMBER	✔☺	✔☺	✔☺
SMEMBERS	✔☺	✔☺	✔☺
SMISMEMBER	□	□	□
SMOVE	✔☺	✔☺	✔☺
SPOP	✔☺	✔☺	✔☺
SRANDMEMBER	✔☺	✔☺	✔☺
SREM	✔☺	✔☺	✔☺
SSCAN	✔☺	✔☺	✔☺

Command	Redis 2.8	Redis 4.0	Redis 5.0
SUNION	✔	✔	✔
SUNIONSTORE	✔	✔	✔

Sorted sets command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
BZPOPMAX	□	□	✔
BZPOPMIN	□	□	✔
ZADD	✔	✔	✔
ZCARD	✔	✔	✔
ZCOUNT	✔	✔	✔
ZINCRBY	✔	✔	✔
ZINTERSTORE	✔	✔	✔
ZLEXCOUNT	✔	✔	✔
ZPOPMAX	□	□	✔
ZPOPMIN	□	□	✔
ZRANGE	✔	✔	✔
ZRANGEBYLEX	✔	✔	✔
ZRANGEBYSCORE	✔	✔	✔
ZRANK	✔	✔	✔
ZREM	✔	✔	✔
ZREMRANGEBYLEX	✔	✔	✔
ZREMRANGEBYRANK	✔	✔	✔
ZREMRANGEBYSCORE	✔	✔	✔
ZREVRANGE	✔	✔	✔
ZREVRANGEBYLEX	✔	✔	✔
ZREVRANGEBYSCORE	✔	✔	✔
ZREVRANK	✔	✔	✔

Command	Redis 2.8	Redis 4.0	Redis 5.0
ZSCAN	✔	✔	✔
ZSCORE	✔	✔	✔
ZUNIONSTORE	✔	✔	✔

Streams command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
XACK	□	□	✔
XADD	□	□	✔
XCLAIM	□	□	✔
XDEL	□	□	✔
XGROUP	□	□	✔
XINFO	□	□	✔
XLEN	□	□	✔
XPENDING	□	□	✔
XRANGE	□	□	✔
XREAD	□	□	✔
XREADGROUP	□	□	✔
XREVRANGE	□	□	✔
XTRIM	□	□	✔

Strings command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
APPEND	✔	✔	✔
BITCOUNT	✔	✔	✔
BITFIELD	✔	✔	✔
BITOP	✔	✔	✔
BITPOS	✔	✔	✔
DECR	✔	✔	✔

Command	Redis 2.8	Redis 4.0	Redis 5.0
DECRBY	✔	✔	✔
GET	✔	✔	✔
GETBIT	✔	✔	✔
GETRANGE	✔	✔	✔
GETSET	✔	✔	✔
INCR	✔	✔	✔
INCRBY	✔	✔	✔
INCRBYFLOAT	✔	✔	✔
MGET	✔	✔	✔
MSET	✔	✔	✔
MSETNX	✔	✔	✔
PSETEX	✔	✔	✔
SET	✔	✔	✔
SETBIT	✔	✔	✔
SETEX	✔	✔	✔
SETNX	✔	✔	✔
SETRANGE	✔	✔	✔
STRALGO	□	□	□
STRLEN	✔	✔	✔

Transaction command group

Command	Redis 2.8	Redis 4.0	Redis 5.0
DISCARD	✔	✔	✔
EXEC	✔	✔	✔
MULTI	✔	✔	✔
UNWATCH	✔	✔	✔
WATCH	✔	✔	✔

6.3. Commands supported by extended data structures of ApsaraDB for Redis Enhanced Edition (Tair)

To help you simplify the development process and improve data processing efficiency, ApsaraDB for Redis Enhanced Edition (Tair) integrates a variety of extended data structures and supports the commands provided by ApsaraDB for Redis Community Edition.

Extended data structures

Data structure	Description
CAS and CAD commands	These commands are developed to enhance the functionality of Redis strings. You can use the commands to implement simple and efficient distributed locks based on Redis. For more information, see Implement high-performance distributed locks by using TairString.
TairString	A TairString is a string that consists of a key, a value, and a version number. Moreover, TairStings can be used to limit the range of outputs returned by the INCRBY and INCRBYFLOAT commands. These commands are used to increase or decrease the values of Redis strings. If outputs are out of range, error messages are returned by these commands.
TairHash	Similar to a native Redis hash, a TairHash is a hash that supports a variety of data structures and provides high processing performance. To simplify the development process, TairHashes also allow you to specify the expiration time and version number for a field. TairHashes use the efficient active expiration algorithm to check the expiration time of fields and delete expired fields. This process does not increase the database response time.
TairGIS	TairGIS is a data structure that uses R-tree indexes and supports APIs related to a geographic information system (GIS). Native Redis GEO commands allow you to use one-dimensional indexes to query points. TairGIS commands allow you to use two-dimensional indexes to query points, linestrings, and polygons. You can also use TairGIS commands to check the relationships between different elements, such as whether A contains B or A intersects with B.
TairBloom	A TairBloom is a Bloom filter that supports dynamic scaling and is fully compatible with RedisBloom commands. Compared with traditional methods that achieve a similar feature, TairBlooms consume less memory and maintain a stable false positive rate during scaling. You can use TairBlooms to check whether a large amount of data exists. In this case, a specific false positive rate is allowed.
TairDoc	TairDoc is a document data structure. It supports JSON standards and is fully compatible with RedisJSON commands. TairDoc data is stored in binary trees and allows fast access to child elements.

Data structure	Description
TairTS	TairTS is a time series data structure that is developed on top of Redis modules. This data structure provides low-latency and high-concurrency in-memory read and write access, supports fast filtering and aggregate queries, and has both storage and computing power. TairTS simplifies the processing of time series data and significantly improves performance.
TairCpc	TairCpc is a data structure developed based on the compressed probability counting (CPC) sketch. It allows you to perform high-performance computing on sampled data while using only a small amount of memory.
TairZset	Native Redis Sorted Sets (ZSETs) allow you to sort elements based on score data of the DOUBLE type only in one dimension. To exceed the limit, Alibaba Cloud has developed the TairZset data structure that allows you to sort elements based on score data of the DOUBLE type with respect to different dimensions. This data structure improves the efficiency of data processing and is also easy to use on the client side without the need to encode, decode, or encapsulate the data.
TairRoaring	The TairRoaring data structure is developed on top of Roaring bitmaps of Tair. TairRoaring uses two-level indexes and introduces multiple dynamic containers. TairRoaring also adopts optimization methods such as single instruction, multiple data (SIMD), vectorization, and popcount to provide less memory consumption and deliver higher computing efficiency for collections.
TairSearch	TairSearch is a full-text search module developed in-house based on Redis modules instead of open source search engine software libraries such as Lucene. TairSearch uses query syntax that is similar to that of Elasticsearch.

Note

- Performance-enhanced instances support all data structures listed in the preceding table.
- Persistent memory-optimized instances support TairString (including CAS and CAD commands) and TairCpc.

Other commands

In addition to the commands of the preceding data structures, Enhanced Edition (Tair) instances support all commands provided by Community Edition instances. For more information, see Commands supported by ApsaraDB for Redis Community Edition.

6.4. Redis commands developed by Alibaba Cloud

ApsaraDB for Redis supports native Redis commands and certain Redis commands developed by Alibaba Cloud. You can use these commands to manage cluster instances or read/write splitting instances of ApsaraDB for Redis.

Description

- INFO KEY: You can run this command to query slots and databases (DBs) to which keys belong. The native Redis command INFO can contain only one optional section by following this syntax: `info [section]`. When you run certain commands for cluster instances of ApsaraDB for Redis, all keys must be in the same slot. The `INFO KEY` command allows you to check whether keys are in the same slot or node. Follow this syntax to run the command:

  ```
  127.0.0.1:6379> info key test_key
  slot:15118 node_index:0
  ```

  Notice

  - In earlier versions, the `INFO KEY` command may return a `node index` that is different from the `node index` in the topology of an instance. This issue is fixed in the latest version. If you are using an instance of an earlier version, upgrade the minor version. For more information, see Update the minor version.
  - The `INFO KEY` command returns the node indexes of shard servers on cluster instances. These shard servers are different from databases that are used in the SELECT command.

- IINFO: You can run this command to specify the node of ApsaraDB for Redis to run the INFO command. This command is similar to the INFO command. Follow this syntax to run the command:

  ```
  iinfo db_idx [section]
  ```

  In this command, db_idx supports the range of [0, nodecount]. You can obtain the nodecount value by running the INFO command, and specify the section option in the same way as you specify this option for a native Redis database. To view a node of ApsaraDB for Redis, you can run the IINFO command or check the instance topology in the console.

- RIINFO: You can run this command in a similar way as you run the IINFO command, but only in read/write splitting mode. This command specifies the idx value as the identifier of the read replica where you want to run the INFO command. If you run this command on instances that are not in read/write splitting mode, the system returns an error. Follow this syntax to run the command:

  ```
  riinfo db_idx ro_slave_idx [section]
  ```

- ISCAN: You can run this command to specify the node of a cluster where you want to run the SCAN command. This command provides the db_idx parameter based on SCAN. The db_idx parameter supports the range of [0, nodecount]. You can obtain the nodecount value by running the INFO command or by checking the instance topology in the console. Follow this syntax to run the command:

  ```
  iscan db_idx cursor [MATCH pattern] [COUNT count]
  ```

- IMONITOR: Similar to IINFO and ISCAN, this parameter provides the db_idx parameter based on the MONITOR command. The db_idx parameter specifies the node where you want to run MONITOR. The db_idx parameter supports the range of [0, nodecount). You can obtain the nodecount value by running the INFO command or by checking the instance topology in the console. Follow this syntax to run the command:

  ```
  imonitor db_idx
  ```

- RIMONITOR: Similar to RIINFO, you can run this command to specify the read replica in a specified shard where you want to run the MONITOR command. This command is used in read/write splitting mode. Follow this syntax to run the command:

```
rimonitor db_idx ro_slave_idx
```

Note Before you run the IMONITOR or RIMONITOR command, use telnet to make sure that your application is connected to the target ApsaraDB for Redis instance. To terminate these commands, run the **QUIT** command.

6.5. Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair)

ApsaraDB for Redis Enhanced Edition (Tair) instances support most commands provided by open source Redis. However, to ensure service performance, limits are imposed on the usage of specific commands.

Introduction of Tair

The rapid development of the Internet makes business scenarios more diverse and complicated. ApsaraDB for Redis Enhanced Edition (Tair) is a high-availability and high-performance NoSQL database service. It provides several series of instances based on storage media such as dynamic random-access memory (DRAM), non-volatile memory (NVM), and ESSDs to meet your requirements for low-latency access, persistence, and reduced overall costs. Tair provides higher performance, more data structures, and more flexible storage methods to meet your requirements in a variety of scenarios. For more information about ESSDs, see ESSDs.

For information about series types of Tair instances, see the following topics:

- Performance-enhanced instances
- Persistent memory-optimized instances
- Storage-optimized instances
- Hybrid-storage instances (phased out)

Performance-enhanced instances

For performance-enhanced instances, no limits are imposed on the use of these commands. Performance-enhanced instances also support some new commands. For more information, see Commands supported by extended data structures of ApsaraDB for Redis Enhanced Edition (Tair).

Persistent memory-optimized instances

Command group	Unsupported command
Keys	RENAME, RENAMENX, and MOVE
Server	SWAPDB

Command group	Unsupported command
Streams	XACK, XADD, XCLAIM, XDEL, XGROUP, XINFO, XLEN, XPENDING, XRANGE, XREAD, XREADGROUP, XREVRANGE, and XTRIM **Note** Since 1.2.3, persistent memory-optimized instances support the Streams command group.

If you use persistent memory-optimized instances to run specific new commands, the performance may degrade as the key capacity increases. If a key has a capacity that is in megabytes, operations on the key encounter significant latency increase. These commands include SETRANGE, SETBIT, APPEND, and BITFIELD, Geo, and HyperLogLog-related commands. To run the preceding commands, we recommend that you use performance-enhanced instances.

Note To use large keys as bitmaps, we recommend that you choose TairRoaring of performance-enhanced instances because TairRoaring provides high computing performance and saves storage capacity. For more information, see TairRoaring.

Storage-optimized instances

Command group	Unsupported command
Geo	GEOADD, GEODIST, GEOHASH, GEOPOS, GEORADIUS, GEORADIUS_RO, GEORADIUSBYMEMBER, and GEORADIUSBYMEMBER_RO
HyperLogLog	PFADD, PFDEBUG, PFCOUNT, PFMERGE, and PFSELFTEST
Keys	RENAME, RENAMENX, MOVE, OBJECT, SORT, and TOUCH
Lists	BRPOP, BLPOP, and BRPOPLPUSH
Scripting	EVAL, EVALSHA, SCRIPT DEBUG, SCRIPT EXISTS, SCRIPT FLUSH, SCRIPT KILL, and SCRIPT LOAD
Strings	BITCOUNT, BITFIELD, BITOP, BITPOS, GETBIT, and SETBIT
Server	MEMORY DOCTOR, MEMORY HELP, MEMORY MALLOC-STATS, MEMORY PURGE, MEMORY STATS, MEMORY USAGE, and SWAPDB
Transactions	DISCARD, EXEC, MULTI, UNWATCH, and WATCH

Hybrid-storage instances

Command group	Unsupported command
Keys	RENAME, RENAMENX, MOVE, and SORT (STORE option)
Lists	LINSERT and LREM
Server	SWAPDB

Command group	Unsupported command
Scripting	SCRIPT DEBUG and SCRIPT LOAD

6.6. Limits on commands supported by cluster instances

This topic describes the limits on commands supported by cluster instances. Cluster instances and standard instances use different architectures and follow different rules to run Redis commands.

Supported commands

- Cluster instances of ApsaraDB for Redis Community Edition: For more information, see Commands supported by ApsaraDB for Redis Community Edition.
- Cluster instances of ApsaraDB for Redis Enhanced Edition (Tair): ApsaraDB for Redis Enhanced Edition (Tair) provides various instance series for various scenarios. Different series of instances follow different rules to run Redis commands. For more information, see Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair).

> **Note** ApsaraDB for Redis cluster instances support the direct connection mode and the proxy mode. For more information, see Enable the direct connection mode and Features of proxy servers. If you want to run the MULTI or EXEC command by using the JedisCluster client, you must use the proxy mode.

Unsupported commands

- SWAPDB
- CLIENT ID
- SORT, which is used together with the BY and GET options

Limited commands

> **Note** To run the following commands on cluster instances, use hash tags to ensure that all keys involved in the commands are distributed in one hash slot. For more information about hash tags, see Redis Cluster Specification.

Command group	Command
HyperLogLog	PFMERGE and PFCOUNT
Keys	RENAME, RENAMENX, and SORT
Lists	RPOPLPUSH, BRPOP, BLPOP, and BRPOPLPUSH
Scripting	EVAL, EVALSHA, SCRIPT EXISTS, SCRIPT FLUSH, SCRIPT KILL, and SCRIPT LOAD
Strings	MSETNX
Transaction	DISCARD, EXEC, MULTI, UNWATCH, and WATCH

Limits on the SELECT command

Connection mode	Description
Proxy mode	The proxy mode supports the SELECT command. Note that some Redis clients such as stackExchange.redis cannot distinguish the SELECT command. To resolve this issue, you can set the cluster_compat_enable parameter to *0* for these clients to disable their compatibility with the Redis Cluster syntax. Then, you can restart these clients and run the SELECT command again. For more information, see Modify parameters of an instance. You can also use other clients that support this command, such as the Jedis client. For more information about the Jedis client, see Jedis client.
Direct connection mode	The direct connection mode does not support the SELECT command due to some limits of common clients such as the Jedis client. For more information about the Jedis client, see Jedis client.

Limits imposed on Lua scripts by the cluster architecture

Warning The cluster architecture imposes limits on Lua scripts. When you change the architecture of an instance to the cluster architecture by performing a configuration change, the Lua scripts may be lost because the script content does not meet the requirements. You must back up the Lua scripts in advance. For more information about configuration changes, see Change the configurations of an instance.

Redis clusters impose limits on the usage of Lua scripts. The following additional limits exist for ApsaraDB for Redis cluster instances:

Note If an error message indicating that the EVAL command fails to run is returned, such as `ERR command eval not support for normal user`, update the minor version of the ApsaraDB for Redis instance to the latest version. For more information, see Update the minor version.

- All keys that a script uses must be allocated to the same hash slot. Otherwise, the following error message is returned:

  ```
  -ERR eval/evalsha command keys must be in same slot\r\n
  ```

  Note You can run the CLUSTER KEYSLOT command to obtain the hash slot of a key.

- A Lua script may not be stored in other nodes when you run the SCRIPT LOAD command on one node.
- The following Pub/Sub commands are not supported: **PSUBSCRIBE**, **PUBSUB**, **PUBLISH**, **PUNSUBSCRIBE**, **SUBSCRIBE**, and **UNSUBSCRIBE**.
- The UNPACK function is not supported.

If all the operations can be performed in the same hash slot and you want to break through the limits that the cluster architecture imposes on your Lua script, you can set the script_check_enable parameter to *0* in the ApsaraDB for Redis console. This way, the system does not check your Lua script at the backend. In this case, you still need to specify at least one key in the KEYS array so that proxy nodes can route commands in the Lua script. If you cannot make sure that all the operations are performed in the same hash clot, an error is returned. For more information, see Modify parameters of an instance.

Additional limits on the proxy mode

- Lua scripts use the **redis.call or redis.pcall** function to run Redis commands. For Redis commands, all

keys must be specified by using the KEYS array, which cannot be replaced by Lua variables. If you do not use the KEYS array to specify the keys, the following error message is returned:

```
-ERR bad lua script for redis cluster, all the keys that the script uses should be passed u
sing the KEYS array\r\n
```

Examples of valid and invalid usage:

```
# The following two commands must be run in advance.
SET foo foo_value
SET {foo}bar bar_value
# Example of valid usage
EVAL "return redis.call('mget', KEYS[1], KEYS[2])" 2 foo {foo}bar
# Examples of invalid usage
EVAL "return redis.call('mget', KEYS[1], '{foo}bar')" 1 foo
EVAL "return redis.call('mget', KEYS[1], ARGV[1])" 1 foo {foo}bar
```

- Keys must be included in all the commands that you want to run. Otherwise, the following error message is returned:

```
-ERR for redis cluster, eval/evalsha number of keys can't be negative or zero\r\n
```

Examples of valid and invalid usage:

```
# Example of valid usage
EVAL "return redis.call('get', KEYS[1])" 1 foo
# Example of invalid usage
EVAL "return redis.call('get', 'foo')" 0
```

- You cannot run the EVAL, EVALSHA, or SCRIPT command in the MULTI or EXEC transactions.

> **Note** If you want to use the features that are unavailable for the proxy mode, you can enable the direction connection mode for an ApsaraDB for Redis cluster instance. However, migrations or configuration changes fail for cluster instances when Lua scripts that do not conform to the requirements of the proxy mode are executed in direct connection mode. This is because cluster instances rely on proxy nodes to migrate data during migrations and configuration changes.
>
> To prevent subsequent migrations and configuration changes based on Lua scripts from failing, we recommend that you conform to the usage limits of Lua scripts in proxy mode when you use Lua scripts in direct connection mode.

Other limits

- You can run the CLIENT LIST command to retrieve information about the connections to the specified proxy node. The following list describes the fields in the command output:
  - The following fields have the same meaning as the fields in open source Redis: `id`, `age`, `idle`, `addr`, `fd`, `name`, `db`, `multi`, `omem`, and `cmd`.
  - The values of the `sub` and `psub` fields are the same. The values are 1 or 0.
  - The `qbuf`, `qbuf-free`, `obl`, and `oll` fields are reserved. You can ignore these fields.
- You can run the CLIENT KILL command in the `client kill ip:port` or `client kill addr ip:port` format.
- If you run commands on a cluster instance in proxy mode, a transaction can be split into multiple sub-

transactions to ensure compatibility with the master-replica architecture. In this case, the atomicity of the transaction cannot be ensured. If you run commands on a cluster instance in direct connection mode, the keys involved in a transaction must be in the same hash slot. This requirement is the same as that of open source Redis Cluster.

> **Notice** In proxy mode, if you run the WATCH command or a transaction involves commands that process multiple keys, such as the MSET command, the transaction is not split. If all keys involved in the transaction are in the same hash slot, the atomicity of the transaction can be ensured.

- The commands that process multiple keys include DEL, SORT, MGET, MSET, BITOP, EXISTS, MSETNX, RENAME, RENAMENX, BLPOP, BRPOP, RPOPLPUSH, BRPOPLPUSH, SMOVE, SUNION, SINTER, SDIFF, SUNIONSTORE, SINTERSTORE, SDIFFSTORE, ZUNIONSTORE, ZINTERSTORE, PFMERGE, and PFCOUNT.
- The commands that are not supported in transactions include WATCH, UNWATCH, RANDOMKEY, KEYS, SUBSCRIBE, UNSUBSCRIBE, PSUBSCRIBE, PUNSUBSCRIBE, PUBLISH, PUBSUB, SCRIPT, EVAL, EVALSHA, SCAN, ISCAN, DBSIZE, ADMINAUTH, AUTH, PING, ECHO, FLUSHDB, FLUSHALL, MONITOR, IMONITOR, RIMONITOR, INFO, IINFO, RIINFO, CONFIG, SLOWLOG, TIME, and CLIENT.

6.7. Limits on the commands supported by read/write splitting instances

Read/write splitting instances of ApsaraDB for Redis are classified into non-cluster read/write splitting instances and read/write splitting cluster instances. The limits on these supported commands vary based on different instance types.

Architecture	Description	Limits
Non-cluster read/write splitting instance	A non-cluster read/write splitting instance is a single-shard read/write splitting instance. The architecture consists of one read/write shard and one, three, or five read replicas. The read/write shard runs in a master-replica structure.	The instance supports the commands that are supported by standard instances of ApsaraDB for Redis. For more information, see Commands supported by ApsaraDB for Redis Community Edition.
Read/write splitting cluster instance	A read/write splitting cluster instance consists of multiple read/write shards. Each shard runs in a master-replica structure and has one read replica attached.	The instance supports the commands that are supported by cluster instances of ApsaraDB for Redis. For more information, see Limits on commands supported by cluster instances.

To simplify operations and maintenance, cluster and read/write splitting instances support multiple commands that are developed by Alibaba Cloud. For more information, see Redis commands developed by Alibaba Cloud.

7.Version description

7.1. Features of ApsaraDB for Redis major versions

7.1.1. New features of ApsaraDB for Redis 7.0

ApsaraDB for Redis 7.0 comes with multiple new features and improvements.

Feature description

- Custom function libraries are added. These function libraries can be persisted and replicated.
- Outputs of commands can be persisted and replicated. However, code of Lua scripts can no longer be persisted or replicated.
- Access control list (ACL) V2 is available, and access control at the level of keys is supported.
- Access control for Pub/Sub channels is supported.
- A multi part append-only file (AOF) mechanism is supported.
- The client eviction mechanism is supported.
- Sharded Pub/Sub is supported.
- Command execution duration can be presented in a histogram.
- Performance statistics on subcommands can be collected.
- The ziplist encoding is replaced with the listpack encoding.
- Global replication buffers are supported.

7.1.2. New features of ApsaraDB for Redis 6.0

This topic describes the new features and key improvements of ApsaraDB for Redis 6.0.

Description

- Supports more API operations in Redis modules.
- Supports RESP3. RESP3 is a new version of the Redis protocol.
- Adds the feature of server-assisted client-side caching. This feature allows you to implement caching in multiple modes.
- Supports multi-thread I/O.
- Supports diskless replication in replicas.
- Adds the Redis cluster mode to the redis-benchmark tool.
- Supports systemd rewriting.
- Supports the Disque module.

7.1.3. New features of ApsaraDB for Redis 5.0

ApsaraDB for Redis 5.0 significantly optimizes kernel performance and system stability. It supports new features, such as the Redis Streams data type, account management, and audit logging, to meet diverse needs.

New features

- Redis Streams data type added. For more information, see Introduction to Redis Streams.
- Account management feature supported. For more information, see Create and manage database accounts.
- Log management feature supported. You can manage audit logs, active logs, and slow logs. For more information about these logs, see Enable the new audit log feature, Query active logs of an instance, and Query slow logs. You can query the records of commands used to manage databases, read and write operations, and sensitive operations such as KEYS and FLUSHALL operations. You can also query slow logs.
- Snapshot-based offline key statistics feature supported. For more information, see Offline key analysis.
- Timer, cluster, and dictionary APIs added.
- Least frequently used (LFU) and least recently used (LRU) caching strategies added to Redis Database Backup (RDB) files.
- Cluster management by using C language in redis-cli instead of Ruby language in redis-trib.rb.
- ZPOPMIN, ZPOPMAX, BZPOPMIN, and BZPOPMAX commands for sorted sets added.
- Active Defragmentation v2 supported.
- Enhanced performance of HyperLogLog supported.
- Statistical memory reports optimized.
- HELP subcommand added for various commands that can include subcommands.
- Enhanced performance stability when connections between databases and clients are frequently closed and established.
- Jemalloc 5.1.0 supported.
- CLIENT ID and CLIENT UNBLOCK commands added.
- LOLWUT command added, which is used to produce interesting outputs.
- Term "slave" discarded in all scenarios unless you need to ensure backward compatibility of APIs.
- Network layer optimized.
- Lua script-related configurations improved.
- Dynamic-hz parameter added to optimize CPU usage and response performance.
- Redis core code reconstruction and improvement.

7.1.4. Features of ApsaraDB for Redis 4.0

Alibaba Cloud has developed ApsaraDB for Redis 4.0 based on Redis 4.0. ApsaraDB for Redis 4.0 supports a variety of new features and fixes several issues. Redis 4.0 provides all the benefits of the Redis 2.8 engine and provides the following features.

Lazyfree

Redis 4.0 supports the Lazyfree feature. This feature can avoid congestion on the Redis-server caused by the `DEL`, `FLUSHDB`, `FLUSHALL` and `RENAME` commands and ensure service stability. The following sections describes this feature.

unlink

For versions that are earlier than Redis 4.0, `OK` is returned only after all memory resources of a key are released when the `DEL` command is run. If the key contains a large amount of data, such as 10 million entries in a hash table, other connections may remain pending for a long time period. To be compatible with the existing `DEL` syntax, Redis 4.0 uses the `UNLINK` command. The UNLINK command can be used in the same way that you use the `DEL` command. However, the background thread releases the memory when Redis 4.0 runs the UNLINK command.

```
UNLINK key [key ...]
```

flushdb/flushall

The `FLUSHDB and FLUSHALL` commands in Redis 4.0 allow you to specify whether to use the Lazyfree feature to release the memory.

```
FLUSHALL [ASYNC]
FLUSHDB [ASYNC]
```

rename

When you run the `RENAME OLDKEY NEWKEY` command, if the specified new key exists, Redis first deletes the existing new key. If the key contains a large amount of data, other connections remain pending for a long time period. To use the Lazyfree feature to delete the key, set the following configuration in the console:

```
lazyfree-lazy-server-del yes/no
```

> **Note**
>
> This parameter is not available in the console.

Expire or evict data

You can specify data expiration time and allow Redis to delete expired data. However, CPU jitter may occur when a large expired key is deleted. Redis 4.0 allows you to specify whether to use the Lazyfree feature to expire or evict data.

```
lazyfree-lazy-eviction yes/no
lazyfree-lazy-expire yes/no
```

New commands

swapdb

The `SWAPDB` command is used to swap two Redis databases. After you run the `SWAPDB` command, you can query the new data from the other database without executing the `SELECT` statement.

```
127.0.0.1:6379> select 0
OK
127.0.0.1:6379> set key value0
OK
127.0.0.1:6379> select 1
OK
127.0.0.1:6379[1]> set key value1
OK
127.0.0.1:6379[1]> swapdb 0 1
OK
127.0.0.1:6379[1]> get key
"value0"
127.0.0.1:6379[1]> select 0
OK
127.0.0.1:6379> get key
"value1"
```

zlexcount

The `ZLEXCOUNT` command is used to return the number of elements in a sorted set. The `ZLEXCOUNT` command is similar to the `ZRANGEBYLEX` command. However, the `ZRANGEBYLEX` command returns all the elements in the sorted set.

memory

Versions that are earlier than Redis 4.0 use the `INFO MEMORY` command to provide the memory information. Redis 4.0 allows you to run the `MEMORY` command to obtain further information about the memory status.

```
127.0.0.1:6379> memory help
1) "MEMORY DOCTOR                        - Outputs memory problems report"
2) "MEMORY USAGE <key> [SAMPLES <count>] - Estimate memory usage of key"
3) "MEMORY STATS                         - Show memory usage details"
4) "MEMORY PURGE                         - Ask the allocator to release memory"
5) "MEMORY MALLOC-STATS                  - Show allocator internal stats"
```

- `memory usage`

  The `USAGE` child command is used to view the memory usage of a specified key in Redis.

  > **Notice**
  > - Key-value pairs in Redis consume memory. Redis also consumes memory to manage data.
  > - The memory usage of keys such as hash tables, lists, sets, and sorted sets is calculated based on sampling. The `SAMPLES` command is used to specify the number of samples.

- `memory stats`

```
27.0.0.1:6379> memory stats
       1) "peak.allocated"    // The maximum memory that Redis used after startup.
       2) (integer) 423995952
       3) "total.allocated"    //The current memory usage.
       4) (integer) 11130320
       5) "startup.allocated"    //The memory that Redis uses after startup and initialization.
       6) (integer) 9942928
       7) "replication.backlog"    //The amount of memory of the backlog used in resuming an interrupted master-replica replication. Default value: 10 MB.
       8) (integer) 1048576
       9) "clients.slaves"    // The amount of memory used in a master-replica replication.
      10) (integer) 16858
      11) "clients.normal"    //The amount of memory used by read and write buffers for common clients.
      12) (integer) 49630
      13) "aof.buffer"    //The sum of the cache used for append-only file (AOF) persistence and the cache generated during the AOF rewrite operations.
      14) (integer) 3253
      15) "db.0"    //The memory used by metadata in each database.
      16) 1) "overhead.hashtable.main"
          2) (integer) 5808
          3) "overhead.hashtable.expires" //The memory used to manage the data that has TTL configured.
          4) (integer) 104
      17) "overhead.total"    //The total used memory for the preceding items.
      18) (integer) 11063904
      19) "keys.count"    //The total number of keys in the current storage.
      20) (integer) 94
      21) "keys.bytes-per-key"    //The average size of each key in the current memory.
      22) (integer) 12631
      23) "dataset.bytes"        //The memory used by user data (= Total memory - Memory used by Redis metadata).
      24) (integer) 66416
      25) "dataset.percentage"    //100 * dataset.bytes / (total.allocated - startup.allocated)
      26) "5.5934348106384277"
      27) "peak.percentage"    // 100 * total.allocated / peak_allocated
      28) "2.6251003742218018"
      29) "fragmentation"    //The memory fragmentation ratio.
      30) "1.1039986610412598"
```

- `memory doctor`

  This command is used to provide diagnostics and identify potential issues.

```
Peak memory: peak.allocated/total.allocated > 1.5. This indicates that the memory fragmentation ratio may be high.
  High fragmentation: fragmentation > 1.4. This indicates a high memory fragmentation ratio.
  Big slave buffers: the average memory for each replica buffer is more than 10 MB. This may be caused by high traffic of write operations on the master node.
  Big client buffers: the average memory for a common client buffer is more than 200 KB. This may be caused by the improper use of a pipeline or caused by Pub/Sub clients that delay processing messages.
```

- `malloc stats` & `malloc purge`

  Both commands take effect only when you use jemalloc.

Least Frequently Used (LFU) mechanism and hotkeys

Redis 4.0 supports allkey-lfu and volatile-lfu data eviction policies, and allows you to run the `OBJECT` command to obtain the frequency of use for a specific key.

```
object freq user_key
```

Based on the LFU mechanism, you can run the `SCAN and OBJECT FREQ` commands to find hotkeys. You can also use the Redis command-line interface (redis-cli) as shown in the following example.

```
$./redis-cli --hotkeys
# Scanning the entire keyspace to find hot keys as well as
# average sizes per key type.  You can use -i 0.1 to sleep 0.1 sec
# per 100 SCAN commands (not usually needed).
[00.00%] Hot key 'counter:000000000002' found so far with counter 87
[00.00%] Hot key 'key:000000000001' found so far with counter 254
[00.00%] Hot key 'mylist' found so far with counter 107
[00.00%] Hot key 'key:000000000000' found so far with counter 254
[45.45%] Hot key 'counter:000000000001' found so far with counter 87
[45.45%] Hot key 'key:000000000002' found so far with counter 254
[45.45%] Hot key 'myset' found so far with counter 64
[45.45%] Hot key 'counter:000000000000' found so far with counter 93
-------- summary -------
Sampled 22 keys in the keyspace!
hot key found with counter: 254 keyname: key:000000000001
hot key found with counter: 254 keyname: key:000000000000
hot key found with counter: 254 keyname: key:000000000002
hot key found with counter: 107 keyname: mylist
hot key found with counter: 93  keyname: counter:000000000000
hot key found with counter: 87  keyname: counter:000000000002
hot key found with counter: 87  keyname: counter:000000000001
hot key found with counter: 64  keyname: myset
```

7.2. Release notes of minor versions

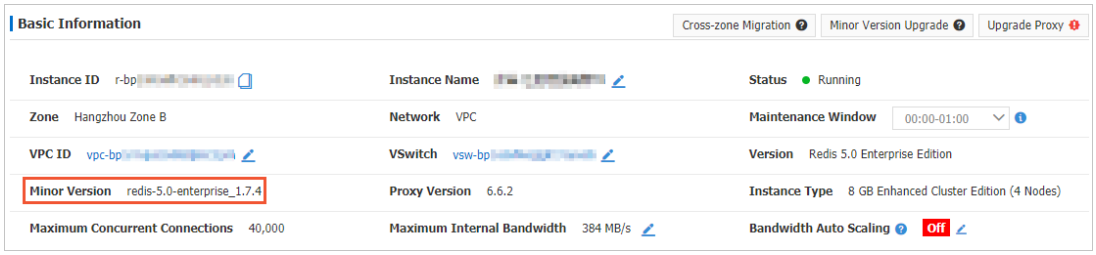
7.2.1. ApsaraDB for Redis Enhanced Edition (Tair)

Alibaba Cloud releases minor versions of ApsaraDB for Redis from time to time to implement new features, fix known issues, and improve user experience. This topic describes the release notes for minor versions of ApsaraDB for Redis Enhanced Edition (Tair). We recommend that you update minor versions of instances during off-peak hours.

View or update the minor version of an instance

You can view the current minor version of an instance in the ApsaraDB for Redis console. For more information about update operations and usage notes, see Update the minor version.

View the minor version



Notice

- The system automatically detects the minor version of an instance. If the instance is of the latest minor version, the **Minor Version Upgrade** button is not displayed or is dimmed.
- Minor version updates may differ from region to region. The minor version of an instance displayed in the ApsaraDB for Redis console prevails.

Introduction to ApsaraDB for Redis Enhanced Edition (Tair)

The rapid development of the Internet makes business scenarios more diverse and complicated. ApsaraDB for Redis Enhanced Edition (Tair) is a high-availability and high-performance NoSQL database service. It provides several series of instances based on storage media such as dynamic random-access memory (DRAM), non-volatile memory (NVM), and ESSDs to meet your requirements for low-latency access, persistence, and reduced overall costs. Tair provides higher performance, more data structures, and more flexible storage methods to meet your requirements in a variety of scenarios. For more information about ESSDs, see ESSDs.

Series type	Description
Performance-enhanced instances	• Performance-enhanced instances use the multi-threading model and provide read and write performance approximately three times that of ApsaraDB for Redis Community Edition instances with the same specifications. • Performance-enhanced instances provide multiple enhanced data structure modules such as TairString (including CAS and CAD commands), TairHash, TairGIS, TairBloom, TairDoc, TairTS, TairCpc, TairZset, TairRoaring, and TairSearch. These instances eliminate concerns about storage structure and timeliness and allow you to focus on application development. • Performance-enhanced instances provide high compatibility. Performance-enhanced instances are fully compatible with open source Redis. You can switch from open source Redis to ApsaraDB for Redis without the need to modify application code. • Performance-enhanced instances provide a variety of enterprise-grade features such as data flashback, proxy query cache, and Global Distributed Cache for Redis. For more information, see Use data flashback to restore data by point in time, Use proxy query cache to address issues caused by hotkeys, and Overview.

Series type	Description
Persistent memory-optimized instances	• Persistent memory-optimized instances provide ultra-high cost-effectiveness. The price of persistent memory-optimized instances is 30% lower than that of ApsaraDB for Redis Community Edition instances with the same capacity. The performance of persistent memory-optimized instances reaches 90% of that of native Redis databases. • Persistent memory-optimized instances support the TairString (including CAS and CAD commands) and TairCpc enhanced data structure modules. • Persistent memory-optimized instances prevent data loss when power failures occur. These instances implement persistence for each command. The system will return a success response for each write operation only after the data is persistently stored. You can use persistent memory-optimized instances as in-memory databases instead of caches. • Persistent memory-optimized instances optimize the append-only file (AOF) rewriting performance for large-sized Redis databases. These instances reduce the latency and jitters that are caused when Redis calls forks to rewrite the AOF. • Persistent memory-optimized instances deliver high compatibility. These instances are compatible with most data structures and commands of native Redis databases.
Storage-optimized instances	Storage-optimized instances are developed based on ESSDs and are compatible with core data structures and APIs of open source Redis. These instances can provide large-capacity, low-cost, and persistent database services. Storage-optimized instances reduce costs and improve data reliability. In addition, storage-optimized instances reduce the amount of reserved memory that is required for the forks of open source Redis. This series type is suitable for scenarios that store warm and cold data, and require compatibility with open source Redis, large capacity, and high access performance.
Hybrid-storage instances (phased out)	Hybrid-storage instances store data in both memory and disks. During off-peak hours, hybrid-storage instances can separate hot data from cold data to ensure a high memory access speed and provide a larger storage than ApsaraDB for Redis Community Edition instances. This allows hybrid-storage instances to strike a balance between performance and costs. **Note** Hybrid-storage instances are discontinued. For more information, see Sales of ApsaraDB for Redis hybrid-storage instances are discontinued. We recommend that you use performance-enhanced instances that provide higher performance and more features. If you have purchased s hybrid-storage instance, you can submit a ticket to migrate the data of the instance.

Update levels

- LOW: regular updates. LOW-level updates include routine feature updates, such as adding a feature.
- MEDIUM: recommended updates. MEDIUM-level updates include optimization of features and modules.

LOW-level updates are also included in MEDIUM-level updates.

- HIGH: major updates. HIGH-level updates include major updates that ensure stability or security, such as fixing a vulnerability or defect. LOW-level and MEDIUM-level updates are also included in HIGH-level updates.

Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair)

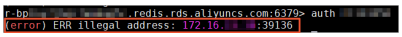
Minor version	Update level	Release date	Type	Description
1.8.9	LOW	2022-06-07	Feature optimization	• The service stability is improved for performance-enhanced instances that have the direct connection mode enabled.
1.8.8	LOW	2022-06-01	Feature optimization	• TairSearch is updated to improve the efficiency of JOIN queries.
1.8.5	HIGH	2022-05-24	Security hardening	• The configuration change stability is improved for cluster performance-enhanced instances that have the direct connection mode enabled. • TairSearch aggregation is supported to improve document write efficiency.
1.8.4	LOW	2022-05-17	Feature optimization	• The stability and execution efficiency of TairSearch are improved.
1.8.3	LOW	2022-04-28	Feature optimization	• The stability of TairTS is enhanced.
1.8.2	HIGH	2022-04-22	Security hardening	• The following issue is fixed: The migration of large keys may fail during configuration changes of cluster instances that have the direct connection mode enabled.
			Feature optimization	• The query efficiency of TairSearch is optimized.
1.8.1	LOW	2022-04-20	Feature optimization	• TairRoaring V2.2 is available. For more information, see TairRoaring.
1.7.28	LOW	2022-03-24	Feature optimization	• Statistics of Tair module commands can be collected and presented in latency histograms. For more information, see Latency insight.

Minor version	Update level	Release date	Type	Description
1.7.27	LOW	2022-03-11	Feature optimization	• TairRoaring V2 is available. For more information, see TairRoaring. • TairSearch is available. For more information, see TairSearch.
1.7.20	MEDIUM	2022-01-04	Feature optimization	• The performance of TairRoaring is optimized.
1.7.17	LOW	2021-11-29	Feature optimization	• The following issue is fixed: Cluster instances may fail to be restarted in specific scenarios. Note Only performance-enhanced instances that use cloud disks are available.
1.7.16	LOW	2021-11-24	Feature optimization	• The following issue is fixed: Statistics of SPOP commands are not collected when the real-time key analysis feature is used.
1.7.12	MEDIUM	2021-10-26	Feature optimization	• The following issue is fixed to enhance stability: An excessive number of slow logs are recorded during configuration changes of cluster instances.
1.7.11	MEDIUM	2021-10-15	Feature optimization	• Transparent Data Encryption (TDE) information can be included in the output of the INFO command. • The stability is enhanced.
1.7.9	LOW	2021-10-13	Feature optimization	• The autonomous capabilities to implement imperceptible slot migration during configuration changes of cluster instances are enhanced.
1.7.8	MEDIUM	2021-09-26	Feature optimization	• The ray casting algorithm is optimized for the TairGIS data structure to yield more precise search results when you run the GIS.CONTAINS command.
1.7.7	MEDIUM	2021-09-13	Feature optimization	• TairRoaring is available. For more information, see TairRoaring. • The stability is enhanced.

Minor version	Update level	Release date	Type	Description
1.7.6	MEDIUM	2021-08-30	Feature optimization	• Memory leaks that may occur during the persistence of append-only file (AOF) rewrites are fixed. • Statistics of queries per second (QPS) are classified. Three types of QPS can be calculated: read QPS, write QPS, and other QPS.
1.7.5	MEDIUM	2021-08-16	Feature optimization	• The reliability of imperceptible slot migration is enhanced to strengthen stability.
1.7.4	HIGH	2021-08-11	Fixed issue	• The following issue is fixed: Configuration changes may fail when the direct connection mode is enabled for cluster instances.
1.7.2	MEDIUM	2021-07-27	Feature optimization	• The stability is enhanced.
1.7.1	MEDIUM	2021-07-20	New feature	• TDE can be used to perform real-time I/O encryption and decryption on Redis Database (RDB) files.
			Feature optimization	• The memory usage of TairHash is optimized.
1.6.15	MEDIUM	2021-07-12	Feature optimization	• The stability is enhanced.
1.6.9	LOW	2021-06-22	New feature	• Latency histograms are supported.
1.6.6	MEDIUM	2021-06-08	Feature optimization	• The stability is enhanced.

1.6.3	LOW	2021-05-17	New feature	• Specific functions of keys can be retained based on keys or key patterns when data is cleared. This feature can be used in the following scenarios:
				  ◦ Restore specific keys or key patterns when you use the data flashback feature to restore data to a point in time. For more information, see Use data flashback to restore data by point in time.
				  ◦ Delete or retain the keyspace content based on keys or key patterns.

Minor version	Update level	Release date	Type	Description
1.6.2	MEDIUM	2021-04-25	Feature optimization	• The performance and migration speed during imperceptible scaling operations are optimized. • Virtual IP addresses (VIPs) can be obtained over the Internet. This provides better support for clients in direct connection mode. For more information, see Enable the direct connection mode. • The large key format is optimized.
1.6.1	MEDIUM	2021-04-08	New feature	• By default, the statistics feature is enabled for large keys. • TairString supports the GT parameter. For more information, see TairString.
			Feature optimization	• The capabilities to migrate slots and implement imperceptible scaling are improved.
1.5.1	HIGH	2021-03-15	Fixed issue	• The issue that the real-time statistics of large keys become inaccurate when keys with the same name are written is fixed.
1.5.0	MEDIUM	2021-02-22	New feature	• Statistics of large keys can be collected in real time.
			Feature optimization	• The failure detection capability of high-availability (HA) systems is improved when you call fork() in memory-intensive scenarios. This prevents long pauses that may occur.
1.4.16	HIGH	2021-01-11	New feature	• Metadata is cleared after an active geo-redundancy link is available. The link is built based on Global Distributed Cache for Redis or Data Transmission Service (DTS). This accelerates subsequent synchronization operations. For more information, see Overview and Configure two-way data synchronization between ApsaraDB for Redis Enhanced Edition (Tair) instances.
			Fixed issue	• Memory leaks that may occur when the **FLUSHALL** command is frequently run in TairHash scenarios are fixed. For more information, see TairHash.

Minor version	Update level	Release date	Type	Description
1.4.13	LOW	2020-11-27	New feature	• When the `illegal address` error message is returned, the IP address of the client can be included in the error message. An IP address whitelist can be configured for your instance based on the IP address prompt. IP address prompt 
1.4.12	MEDIUM	2020-11-26	New feature	• Flags such as FLAGS are used in the syntax of TairString. This syntax is compatible with Memcached semantics.
			Feature optimization	• The failure detection capability of HA systems is improved. **Note** We recommend that you update the minor version to 1.5.0 or later to obtain the latest optimization for this feature.
1.4.9	HIGH	2020-10-22	Fixed issue	• Correct binary logs are generated when TairString expires. This prevents data inconsistency between the master and replica nodes. • The abnormal switchover of HA systems is fixed. This issue occurs when TairHash still uses the active expire algorithm in read-only scenarios. • The following issue is fixed: The system may not respond when an instance that was forcefully stopped is restarted. • The following issue is fixed: Expired keys are deleted when RDB files are loaded to ApsaraDB for Redis instances that have the data flashback feature enabled. For more information about the data flashback feature, see Use data flashback to restore data by point in time.
1.4.8	HIGH	2020-10-14	Fixed issue	• Memory leaks that may occur when specific modules are loading are fixed.
1.4.7	MEDIUM	2020-10-12	Feature optimization	• The output of the **CLUSTER NODES** command can be stored in the cache. This accelerates the command execution.
1.4.6	MEDIUM	2020-09-28	Feature optimization	• The processing capabilities of specific modules are improved in special scenarios.

Minor version	Update level	Release date	Type	Description
1.4.1	MEDIUM	2020-09-08	New feature	• Proxy nodes can pass through the IP addresses of clients to operational logs and audit logs. This helps you interpret logs and identify clients that have issues.
			Feature optimization	• The data collection capability is improved. This minimizes the impact on data shards when a large number of connections are queued up and I/O threads are busy.
1.3.17	MEDIUM	2020-08-04	Feature optimization	• The communication latency is reduced when DTS is used to implement two-way data synchronization. For more information, see Configure two-way data synchronization between ApsaraDB for Redis Enhanced Edition (Tair) instances.
1.3.16	HIGH	2020-07-19	New feature	• The security group feature provided by Elastic Compute Service (ECS) is supported to simplify O&M. To allow ECS instances to access an ApsaraDB for Redis instance, you can add the security groups to which the ECS instances belong to the ApsaraDB for Redis instance. You do not need to manually add the IP addresses of the ECS instances to the whitelists of the ApsaraDB for Redis instance. For more information, see Step 2: Configure whitelists. • The TairString module is updated to support more API operations (flags) that are compatible with Memcached semantics.
			Fixed issue	• The following issue is fixed: The **BGREWRITEAOF** command is interrupted when you use the data flashback feature to restore data to a point in time. For more information about the data flashback feature, see Use data flashback to restore data by point in time. • Latency flag bits in audit logs are modified to help you identify these bits in master and replica audit logs.

Minor version	Update level	Release date	Type	Description
1.3.9	MEDIUM	2020-06-19	Feature optimization	• Saved point metadata is automatically cleared when data is cleared. This way, replicators are quickly restored in scenarios where multi-way data synchronization is implemented based on Global Distributed Cache for Redis or DTS. For more information, see Overview and Configure two-way data synchronization between ApsaraDB for Redis Enhanced Edition (Tair) instances.
1.3.7	LOW	2020-05-19	New feature	• The Replication part in the output of the **INFO** command shows role information, such as `role:master`. This allows Redisson clients to call the role information in specific scenarios.
1.3.6	MEDIUM	2020-05-19	New feature	• Statistics of hotkeys can be regularly recorded in logs. • The data statistics feature is available for performance metrics. This feature allows the system to differentiate the QPS that are generated in read, write, and read/write synchronization operations. This improves the accuracy of the statistics.
			Feature optimization	• The kernel capabilities for restoring data to a point in time are improved to simplify the data restoration process. For more information, see Use data flashback to restore data by point in time. • Commands such as **AUTH**, **ADMINAUTH**, and **CONFIG** do not record sensitive information when these commands are used. This improves data security.
1.3.5	HIGH	2020-04-22	Fixed issue	• The issue that deadlocks may occur when multi-threaded engines asynchronously close client connections is fixed. • The issue that file descriptors in engines cannot linearly expand is fixed.
			New feature	• The 64-bit hash algorithm is applied to TairBloom. • The final memory usage of TairBloom can be estimated to allow the system to record more accurate memory statistics. • The exhgetAll2 interface is used in TairHash to revise command output formats.

Minor version	Update level	Release date	Type	Description
1.3.3	HIGH	2020-04-22	Fixed issue	• The error message that is returned by ApsaraDB for Redis when a whitelist is improperly configured is changed from `(error) ERR invalid password` to `(error) ERR illegal address`. • Memory leaks that may occur when you use TairGIS to manage multiple polygons are fixed. • The issue that the default path for TairDoc is incorrect is fixed. • The issue that Pub and Sub commands may compete for resources on multi-threaded engines is fixed.

1.3.1	HIGH	2020-04-03	New feature	• The data flashback feature is supported. This feature allows you to restore instance data to a point in time within the last seven days. This helps prevent data loss caused by accidental operations, simplify O&M, and protect databases in real time. For more information, see Use data flashback to restore data by point in time.
				• TairGIS is compatible with Redis GEO commands.
				• TairBloom can be used in capacity security validation for the BFRESERVE interface.
				• TairHash supports the following new features:
				  ◦ The NOACTIVE option is added to multiple commands including **EXHSET**, **EXHEXPIRE**, **EXHINCRBY**, and **EXHINCRBYFLOAT**. This option can reduce memory overheads in specific scenarios.
				  ◦ The MAX and MIN options are added to the **EXHINCRBY** command to define the upper and lower boundaries of the value range.
				  ◦ The noexp option is added to the **EXHLEN** command to return the true length of the EXHLEN command.
				• The **HINCRBY** and **HINCRBYFLOAT** commands in the hash structure are supported. The transaction processing capability of these commands can be used to increase and decrease different fields that constitute a key at the same time.

			Feature optimization	• Data structure modules are improved. For more information, see Integration with multiple Redis modules. • The JedisCluster client can run the **MGET** and **MSET** commands on cluster instances with much higher performance.

Minor version	Update level	Release date	Type	Description
			Fixed issue	• The following issue is fixed: Binary logs take up more space than can be provided by an instance. • The issue that the system may not respond when hotkeys are evicted is fixed. • The issue that the system may not respond due to double deallocation in TairHash commands is fixed. • The issue that the system may not respond due to use-after-free (UAF) is fixed. The issue occurs when the audit log feature is disabled.
1.0.10	LOW	2020-02-19	New feature	• The **BITFIELD_RO** command is added. This command significantly optimizes the performance of the BITFIELD command in read/write splitting scenarios. **Note** If **BITFIELD** commands contain only the get option, proxy nodes convert the commands into **BITFIELD_RO** commands and route these new commands to multiple backend data shards.
1.0.9	HIGH	2020-02-19	Fixed issue	• The following issue is fixed: The replication process stops when specific complex commands in Lua scripts are run.
1.0.8	HIGH	2020-02-10	Feature optimization	• The algorithm and performance related to traffic throttling are improved.
			Fixed issue	• The issue that the service stops because the congestion of client output buffers triggers server overload protection is fixed.
			New feature	• Global Distributed Cache for Redis is supported. Global Distributed Cache for Redis is an active geo-redundancy database system that is developed based on ApsaraDB for Redis. Global Distributed Cache for Redis supports business scenarios in which multiple sites in different regions provide services at the same time. It helps enterprises replicate the active geo-redundancy architecture of Alibaba. For more information, see Overview. • Binary logs and their protocols are available to support capabilities such as active geo-redundancy.

Minor version	Update level	Release date	Type	Description
1.0.5	HIGH	2020-02-01	Fixed issue	• The issue that the output of the INFO command can contain the `cluster_enabled` information when the direct connection mode is used is fixed. This allows specific SDKs to automatically negotiate to enter the cluster mode. For more information, see Enable the direct connection mode. • The issue that the number of controlled clients is inaccurately calculated is fixed. • The issue that the system may not respond when a client is released is fixed. • The issue that the system may not respond when a **pipeline** contains complex commands is fixed.
0.2.9	HIGH	2020-01-06	Feature optimization	• The memory usage of TairHash is optimized.
			Fixed issue	• The issue that the system may not respond when traffic throttling is executed is fixed.
0.2.7	HIGH	2019-12-23	New feature	• The direct connection mode is supported. Clients can bypass proxy nodes to connect to ApsaraDB for Redis instances by using private endpoints. This is similar to the connection to open source Redis clusters. The direct connection mode can reduce communication overheads and further improve the response speed of ApsaraDB for Redis. For more information, see Enable the direct connection mode. • The identification logic for hotkeys is supported. This allows hotkeys of engines to be accurately found. The memory usage of hotkeys is also optimized. • **EXCAS** commands are supported in optimistic locking scenarios. For more information about how to use EXCAS commands, see EXCAS and Reduce resource consumption for optimistic locking.
			Fixed issue	• The core dump issue that may occur when pipelines are used is fixed.

Minor version	Update level	Release date	Type	Description
0.2.3	LOW	2019-12-03	First release	Performance-enhanced instances of the ApsaraDB for Redis Enhanced Edition (Tair) are suitable for business scenarios that require high concurrency, high performance, and a large number of read and write operations on hot data. Compared with ApsaraDB for Redis Community Edition instances, performance-enhanced instances have the following benefits: • Performance-enhanced instances use the multi-threading model and provide read and write performance approximately three times those of ApsaraDB for Redis Community Edition instances with the same specifications. • Performance-enhanced instances provide multiple enhanced data structure modules such as TairString (including CAS and CAD commands), TairHash, TairGIS, TairBloom, and TairDoc. These modules do not have strict requirements for the structure or validity period of storage and facilitate application development.

Persistent memory-optimized instances of the ApsaraDB for Redis Enhanced Edition (Tair)

Minor version	Update level	Release date	Type	Description
1.2.3.3	LOW	2022-05-12	New feature	• TairCpc is supported. For more information, see TairCpc.
1.2.3.2	HIGH	2022-04-24	Fixed issue	• The following issue is fixed: The key eviction order is abnormal when the MAXMEMORY_VOLATILE_TTL policy is specified for key eviction. • The process of adding or deleting nodes for a cluster instance is optimized. • The stability of semi-synchronous data replication is enhanced.
1.2.3.1	LOW	2022-03-31	Feature optimization	• The following issue is fixed: CAS commands cannot be persisted. • The following issue is fixed: Client IP addresses cannot be displayed in slow and audit logs of cluster instances. • The stability is enhanced.

Minor version	Update level	Release date	Type	Description
1.2.3	MEDIUM	2022-03-03	New feature	• TairString is supported. For more information, see TairString. • Streams are supported. • The intset encoding is supported for SETs to reduce memory overheads. • Data eviction policies are supported. • The performance is optimized, and the stability is enhanced.
			Fixed issue	• The following issue is fixed: Statistics of SPOP commands are not collected when the real-time key analysis feature is used.
1.2.2.4	LOW	2022-01-21	Feature optimization	• Storage space occupied by large values is optimized to reduce used persistent storage.
1.2.2.3	HIGH	2021-12-30	Feature optimization	• Tail latency that takes place during data writes is optimized. • The stability is enhanced.
1.2.2.2	HIGH	2021-12-14	Fixed issue	• The following issue is fixed: In specific scenarios, the usage of lists and hashes cannot be correctly calculated.
1.2.2.1	LOW	2021-10-21	Feature optimization	• The stability is enhanced.
1.2.2	HIGH	2021-10-20	Feature update	• The speed at which cluster instances are scaled can be adaptively controlled. **Note** During scaling operations, the speed at which old data is cleared is well adapted to instance loads. For example, more CPU cores are used for clearance in low-load scenarios to increase the clearance speed. This feature keeps the CPU utilization at a high level during scaling operations but does not affect service availability. • The performance of commands such as CLUSTER NODES is optimized for large cluster instances.

Minor version	Update level	Release date	Type	Description
			Fixed issue	• The issue that abnormal slow logs are generated during scaling of cluster instances is fixed. • The stability is enhanced.
1.2.0	LOW	2021-09-21	Feature update	• The storage of the LIST, HASH, SET, and ZSET data structures is optimized to reduce their usage of memory and persistent memory.
1.1.8	LOW	2021-08-17	Feature update	• When memory is exhausted in specific scenarios, persistent memory can be fully utilized. • The stability is enhanced.
1.1.7	LOW	2021-08-02	New feature	• Semi-synchronous data replication is supported. By default, this feature is disabled. If you want to enable this feature, submit a ticket. **Note** If this feature is enabled, logs are transmitted from a master node to a replica node after the data update that the client initiates is complete on the master node. After the replica node receives all logs, the master node returns the log transmission information to the client. If a replica node is unavailable or the communication between a master node and a replica node is abnormal, semi-synchronous replication degrades to asynchronous replication. • In the output of the **INFO** command, the return value of the redis_version parameter is changed to 4.9.9 and the pena_version parameter is added to indicate the minor version.
1.1.6.1	MEDIUM	2021-06-10	Feature update	• The stability is enhanced.
			New feature	• Imperceptible scaling operations are supported. This allows slots to be migrated without perceptible impacts on your business.

Minor version	Update level	Release date	Type	Description
1.1.6	MEDIUM	2021-05-08	Feature optimization	• The hash and ZSET data structures support ziplist encoding to reduce memory overheads. • The failure detection capability of HA systems is improved. • Data migration of large cluster instances is optimized. Large cluster instances refer to instances that have a large number of data shards. • The scaling stability of the cluster architecture is improved.
1.1.5	MEDIUM	2021-01-15	New feature	• Cluster instances are supported. This eliminates performance bottlenecks caused by the single-threading model of open source Redis. You can use high-performance cluster instances to process large-capacity workloads. • The minor version of an instance is used as the value of the redis_version parameter in the output of the **INFO** command.
			Feature optimization	• The stability is enhanced.
1.1.4	MEDIUM	2020-10-28	New feature	• When you run the **INFO** command by using a standard account, the usage of persistent memory is included in the command output.
			Feature optimization	• The space occupied by embstr encoding is decreased to reduce memory overheads. • The stability is enhanced.

Minor version	Update level	Release date	Type	Description
1.1.3	LOW	2020-09-16	First release	Persistent memory-optimized instances of the ApsaraDB for Redis Enhanced Edition (Tair) are equipped with the Intel Optane DC Persistent Memory Module (AEP) to provide in-memory databases that have a large capacity and are compatible with open source Redis. For more information about persistent memory-optimized instances, see Persistent memory-optimized instances. Persistent memory-optimized instances have the following benefits: • A persistent memory-optimized instance costs about 30% less than an ApsaraDB for Redis Community Edition instance. • In scenarios where advanced memory specifications are used, issues such as high latency, high network jitter, and slow service data loading during fork operations triggered by AOF rewrites are resolved. You are not confronted with the trade-off between performance and persistence. • Persistent memory-optimized instances support command-level persistence. A response is returned after data persistence is complete for each write operation. • Persistent memory-optimized instances are compatible with most of the data structures and interfaces of open source Redis. The persistence of data structures except for Redis Streams is supported. Note For more information about command limits, see Limits on commands supported by ApsaraDB for Redis Enhanced Edition (Tair).

Storage-optimized instances of the ApsaraDB for Redis Enhanced Edition (Tair)

Minor version	Update level	Release date	Type	Description
2.2.15	HIGH	2022-06-06	Fixed issue	• The stability is enhanced.
2.1.13	LOW	2022-05-05	Feature optimization	• The stability of data synchronization between master and replica nodes is optimized.

Minor version	Update level	Release date	Type	Description
2.1.12	LOW	2022-04-26	Feature optimization	• The stability of data synchronization between master and replica nodes is optimized.
2.1.10	LOW	2022-04-14	Feature optimization	• The efficiency of synchronization between master and replica nodes is optimized to provide higher throughput with lower latency. • The resource usage of compact operations that are generated when a large number of large keys concurrently expire is optimized.
2.1.7	LOW	2021-08-12	Feature optimization	• The speed of master-replica switchover is optimized to improve stability. Master-replica switchover is also known as proactive HA.
2.1.5	HIGH	2021-07-13	Fixed issue	• The issue that binary logs occupy a large amount of space is fixed.
2.1.4	MEDIUM	2021-07-07	Feature optimization	• The storage parameters of storage-optimized instances with terabytes of capacity are optimized to improve performance. • The stability is enhanced.
2.1.0	LOW	2021-05-26	New feature	• The **MULTI** and **EXEC** transaction commands are supported. • The **EVAL**, **EVALSHA**, and **SCRIPT** Lua script commands are supported.
2.0.13	LOW	2021-04-24	New feature	• Notifications about keyspace events can be sent.
1.2.17	HIGH	2021-02-04	Fixed issue	• The following issue is fixed: In specific cases, the **FLUSHALL** command may cause data inconsistency between master and replica nodes.
1.2.16	HIGH	2021-01-29	Fixed issue	• The following issue is fixed to ensure data consistency between master and replica nodes: In specific cases, a master node and a replica node cannot be automatically reconnected after a disconnection.
1.2.15	MEDIUM	2021-01-20	New feature	• The cmd_slowlog_count metric is added to collect historical statistics of slow logs.
			Feature optimization	• Protection capabilities are improved in scenarios where disk space is fully occupied.

Minor version	Update level	Release date	Type	Description
1.2.14	LOW	2020-12-31	New feature	• Data verification for master and replica nodes is added.
1.2.13	HIGH	2020-12-31	Fixed issue	• The defects of the **RESTORE** command that emerge in specific scenarios are fixed.
1.2.12	MEDIUM	2020-12-23	Feature optimization	• The **FLUSHALL** and **FLUSHDB** commands cannot be run in serial.
1.2.11	MEDIUM	2020-12-10	Feature optimization	• The memory management is optimized.
1.2.10	LOW	2020-12-04	New feature	• You can access ApsaraDB for Redis instances that are deployed in virtual private clouds (VPCs) without using passwords. For more information, see Enable password-free access. • To prevent out of memory (OOM) errors caused by large transactions, the size of a single transaction is limited by using the **max-write-batch-size** option.
1.2.9	MEDIUM	2020-11-27	Feature optimization	• To make the Redis-shake tool available for data migration scenarios, the value of the redis_version parameter is added to the output of the **INFO** command.
1.2.8	HIGH	2020-11-20	Fixed issue	• The issue that the number of connections is incorrectly calculated is fixed. • The issue that the number of keys is incorrectly calculated when the **RESTORE** command is run is fixed.
1.2.7	HIGH	2020-10-28	Fixed issue	• The issue that unexpected quits occur when the **SETRANGE** command is run is fixed. • The issue that exceptions occur when keys are empty strings is fixed.
1.2.6	HIGH	2020-09-28	Fixed issue	• The issue of a sharp increase in connection management logs is fixed.
1.2.5	HIGH	2020-09-27	Fixed issue	• The issue of a sharp increase in operational logs is fixed.

Minor version	Update level	Release date	Type	Description
1.2.4	HIGH	2020-09-27	Feature optimization	• The stability of data synchronization between master and replica nodes is improved. • The scanning performance for members that have complex data structures is optimized.
			Fixed issue	• The following issue is fixed to prevent semantic errors that may occur when multiple duplicate members are managed: The **ZADD** and **ZINCR** commands are inconsistent with those of open source Redis. • The following issue is fixed: `nil` is returned for empty strings when the **MGET** command is run.
1.2.3	LOW	2020-09-27	First release	Storage-optimized instances of the ApsaraDB for Redis Enhanced Edition (Tair) are independently developed based on the TairDB key-value storage engine and use enhanced SSDs (ESSDs) to store data. These instances provide high-capacity, low-cost, and persistent database services. For more information about storage-optimized instances, see Storage-optimized instances. Storage-optimized instances have the following benefits: • Storage-optimized instances store data in cloud disks to implement data persistence at a cost that can be as low as 15% that of ApsaraDB for Redis Community Edition instances. • Storage-optimized instances use multiple I/O threads and worker nodes to improve throughput on each server and support replication of binary logs between master and replica nodes. • Storage-optimized instances are compatible with most of the open source Redis commands. • Storage-optimized instances reduce the amount of reserved memory that is required for the forks of open source Redis.

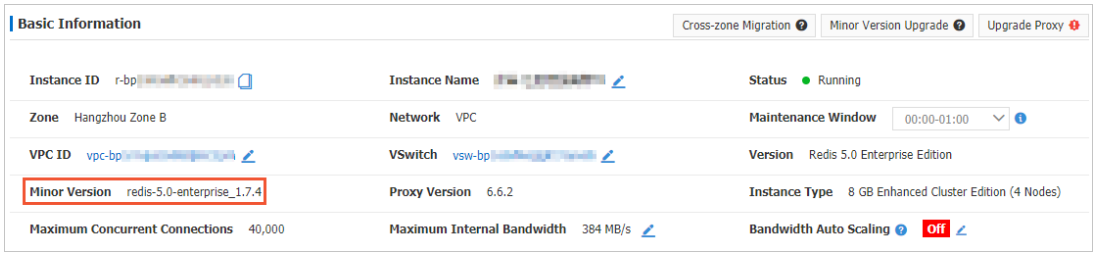
7.2.2. ApsaraDB for Redis Community Edition

Alibaba Cloud releases new minor versions of ApsaraDB for Redis from time to time to provide more features, fix known issues, and improve user experience. This topic describes the release notes for minor versions of ApsaraDB for Redis Community Edition. We recommend that you update minor versions of instances during off-peak hours.

View or update the minor version of an instance

You can view the current minor version of an instance in the ApsaraDB for Redis console. For more information about update operations and usage notes, see Update the minor version.

View the minor version



Notice

- The system automatically detects the minor version of an instance. If the instance is of the latest minor version, the **Minor Version Upgrade** button is not displayed or is dimmed.
- Minor version updates may differ from region to region. The minor version of an instance displayed in the ApsaraDB for Redis console prevails.

Update levels

- LOW: regular updates. LOW-level updates include routine feature updates, such as adding a feature.
- MEDIUM: recommended updates. MEDIUM-level updates include optimization of features and modules. LOW-level updates are also included in MEDIUM-level updates.
- HIGH: major updates. HIGH-level updates include major updates that ensure stability or security, such as fixing a vulnerability or defect. LOW-level and MEDIUM-level updates are also included in HIGH-level updates.

V7.0

Minor version	Update level	Release date	Type	Description
7.0.0.4	LOW	2022-06-20	Feature optimization	• New features and enhancements of open source Redis 7.0.2 are supported.
7.0.0.3	LOW	2022-05-27	First release	• 7.0 instances that use cloud disks are available. For more information, see Redis 7.0 release notes.

V6.0

Minor version	Update level	Release date	Type	Description
0.1.19	LOW	2022-06-22	Feature optimization	• Latency histograms are optimized. For more information, see Latency insight.
			Security hardening	• The process of adding and deleting nodes for a cluster instance is optimized. For more information, see Adjust the number of shards for an 云盘ApsaraDB for Redis instance.

Minor version	Update level	Release date	Type	Description
0.1.18	LOW	2022-05-17	Feature optimization	• The Errorstats - Selected field returned by the INFO command can be deleted.
0.1.17	LOW	2022-05-10	Feature optimization	• The MOVE command is supported by cluster instances.
0.1.16	MEDIUM	2022-04-25	Security hardening	• The process of adding and deleting nodes for a cluster instance is optimized to enhance stability.
0.1.15	LOW	2022-03-24	New feature	• Latency histograms are supported. For more information, see Latency insight. • Error statistics can be returned by the INFO command.
0.1.14	LOW	2022-02-21	Feature optimization	• The READONLY and READWRITE commands are supported. For more information, see Commands supported by ApsaraDB for Redis Community Edition.
0.1.13	LOW	2022-01-14	Feature optimization	• A metric is added to monitor the amount of memory occupied by database metadata. • The real-time key statistics feature is optimized.
0.1.12	HIGH	2021-10-26	Security hardening	• The following issue is fixed to enhance stability: An excessive number of slow logs are recorded during configuration changes of cluster instances.
0.1.11	HIGH	2021-10-13	Feature optimization	• The autonomous capabilities to implement imperceptible slot migration during configuration changes of cluster instances are enhanced. • The performance of commands such as CLUSTER NODES is optimized. • The whitelist feature is optimized.
0.1.10	HIGH	2021-09-06	Feature optimization	• Stability is enhanced.
0.1.9	MEDIUM	2021-08-16	Feature optimization	• The reliability of imperceptible slot migration is enhanced to strengthen stability.

Minor version	Update level	Release date	Type	Description
0.1.8	MEDIUM	2021-08-06	Feature optimization	• Stability is enhanced.
0.1.7	MEDIUM	2021-08-06	Feature optimization	• Stability is enhanced.
0.1.6	MEDIUM	2021-07-19	New feature	• New features and enhancements of open source Redis 6.0.14 are supported. For more information, see Redis 6.0 release notes.
			Feature optimization	• The process of deleting slots from the source instance after data migration is complete is optimized. This enhances data reliability. • The process of migrating incremental data by using slots is simplified.
0.1.5	MEDIUM	2021-06-04	Feature optimization	• Stability is enhanced.
0.1.4	MEDIUM	2021-05-27	Feature optimization	• Stability is enhanced.
0.1.3	LOW	2021-05-18	New feature	• Statistics of large keys can be collected in real time.
0.1.2	MEDIUM	2021-05-07	Feature optimization	• Stability is enhanced.
0.1.1	MEDIUM	2020-11-28	New feature	• New features and enhancements of open source Redis 6.0.9 are supported. For more information, see Redis 6.0 release notes. • Slots can be migrated without impacting the instance. • Virtual IP addresses (VIPs) can be obtained over the Internet. This provides better support for clients that use direct connection mode. For more information, see Enable the direct connection mode.
			Feature optimization	• Health checks for instances are improved to accelerate the switchover between the master and replica nodes when disk jitters occur.

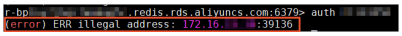
Minor version	Update level	Release date	Type	Description
0.0.5	HIGH	2020-08-21	Fixed issue	• The issue that statistics about hotkeys are inaccurate is fixed.
0.0.4	HIGH	2020-07-20	Fixed issue	• The issue that the settings of specific parameters become invalid after the instance is restarted is fixed. • The issue that replica nodes are incorrectly flagged in slow logs during data synchronization is fixed.
0.0.3	LOW	2020-06-11	New feature	• New features and enhancements of open source Redis 6.0.5 are supported. For more information, see Redis 6.0 release notes. • The role information can be contained in the Replication part of the **INFO** command output, such as `role:master`. This allows Redisson clients to call the role information in specific scenarios. • Statistics on queries per second (QPS) of read and write commands can be collected. For more information, see View monitoring data.
0.0.2	LOW	2020-06-02	New feature	• New features and enhancements of open source Redis 6.0.4 are supported. For more information, see Redis 6.0 release notes.
0.0.1	LOW	2020-05-06	First release	• The first minor version that is available. This minor version is developed based on open source Redis 6.0.1. For more information, see Redis 6.0 release notes.

V5.0

Minor version	Update level	Release date	Type	Description
5.1.9	LOW	2022-06-22	Feature optimization	• Latency histograms are optimized. For more information, see Latency insight.
			Security hardening	• The process of adding and deleting nodes for a cluster instance is optimized. For more information, see Adjust the number of shards for an 云盘ApsaraDB for Redis instance.

Minor version	Update level	Release date	Type	Description
5.1.8	LOW	2022-05-17	Feature optimization	• The Errorstats - Selected field returned by the INFO command can be deleted.
5.1.7	LOW	2022-05-06	Feature optimization	• The MOVE command is supported by cluster instances.
5.1.6	MEDIUM	2022-04-25	Feature optimization	• The process of adding and deleting nodes for a cluster instance is optimized to enhance stability.
5.1.5	LOW	2022-04-13	Feature optimization	• Stability is enhanced.
5.1.4	LOW	2022-03-24	New feature	• Latency histograms are supported. For more information, see Latency insight. • Error statistics can be returned by the INFO command.
5.1.3	LOW	2022-02-21	Feature optimization	• The READONLY and READWRITE commands are supported. For more information, see Commands supported by ApsaraDB for Redis Community Edition.
5.1.1	LOW	2022-01-04	Feature optimization	• A metric is added to monitor the amount of memory occupied by database metadata.
5.0.9	LOW	2021-12-22	Feature optimization	• The real-time key statistics feature is optimized.
5.0.8	LOW	2021-11-15	Feature optimization	• A cluster instance can be scaled without impacting the instance.
0.5.17	HIGH	2022-06-07	Security hardening	• Stability is enhanced.
0.5.16	HIGH	2022-05-23	Security hardening	• The configuration change stability is improved for cluster instances that have the direct connection mode enabled.
0.5.15	HIGH	2022-04-25	Security hardening	• The following issue is fixed: The migration of large keys may fail during configuration changes of cluster instances that have the direct connection mode enabled.

Minor version	Update level	Release date	Type	Description
0.5.14	LOW	2022-01-04	Feature optimization	• A metric is added to monitor the amount of memory occupied by database metadata.
0.5.12	LOW	2021-11-29	Feature optimization	• The following issue is fixed: Cluster instances may fail to be restarted in specific scenarios.
0.5.11	LOW	2021-11-24	Feature optimization	• The following issue is fixed: Statistics of SPOP commands are not collected when the real-time key analysis feature is used.
0.5.10	HIGH	2021-10-26	Security hardening	• The following issue is fixed to enhance stability: An excessive number of slow logs are recorded during configuration changes of cluster instances.
0.5.9	HIGH	2021-10-15	Security hardening	• Stability is enhanced.
0.5.8	MEDIUM	2021-10-13	Feature update	• The autonomous capabilities to implement imperceptible slot migration during configuration changes of cluster instances are enhanced.
0.5.7	LOW	2021-08-26	New feature	• Statistical data of QPS is classified. Three types of QPS can be calculated: read QPS, write QPS, and other QPS.
0.5.6	HIGH	2021-08-16	Fixed issue	• The reliability of imperceptible slot migration is enhanced to strengthen stability.
0.5.5	HIGH	2021-08-05	Fixed issue	• The following issue is fixed: Configuration changes may fail when the direct connection mode is enabled for cluster instances.
0.5.4	MEDIUM	2021-07-27	Feature update	• Stability is enhanced.
0.5.3	MEDIUM	2021-07-21	Feature update	• The process of deleting source data after data migration is complete is optimized to enhance data reliability. • Incremental data synchronization is simplified during data migration.
			Security hardening	• The security vulnerabilities of the open source LuaJIT compiler are fixed.

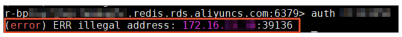
Minor version	Update level	Release date	Type	Description
0.5.2	HIGH	2021-04-26	New feature	• The migration mechanism for slots is optimized to allow instances that use cloud disks to be scaled without data loss. • Statistics of large keys can be collected in real time. • VIPs can be obtained over the Internet. This provides better support for private endpoint users.
0.5.0	MEDIUM	2021-03-25	New feature	• Slots can be migrated without impacting the instance.
			Feature optimization	• The stability is enhanced when a large number of asynchronous client requests are processed.
0.4.0	MEDIUM	2021-03-09	New feature	• Statistics of large keys can be collected in real time. • The **CONFIG RESETSTAT** command is supported. • When the `illegal address` error message is returned, the IP address of the client can be included in the error message. An IP address whitelist can be configured for your instance based on the IP address prompt. IP address prompt 
			Feature optimization	• Health checks for instances are improved to accelerate the switchover between the master and replica nodes when disk jitters occur.
0.3.10	HIGH	2020-09-25	Fixed issue	• The issue that the output of the **CLUSTER NODES** command is inconsistent with that of open source Redis is fixed. Multiple slots are separated by spaces to prevent errors in client parsing.

Minor version	Update level	Release date	Type	Description
0.3.9	LOW	2020-07-20	New feature	• The security group feature provided by Elastic Compute Service (ECS) is supported to simplify O&M. To allow ECS instances to access an ApsaraDB for Redis instance, you can add the security groups to which the ECS instances belong to the ApsaraDB for Redis instance. You do not need to manually add the IP addresses of the ECS instances to the whitelists of the ApsaraDB for Redis instance. For more information, see Step 2: Configure whitelists.
0.3.8	HIGH	2020-07-14	Feature optimization	• Subcommands of **CLIENT UNBLOCK** are available.
			Fixed issue	• The issue of expiration times being parsed incorrectly when slots are migrated is fixed. • Latency flag bits in audit logs are modified to help you identify these bits in master and replica audit logs.
0.3.7	HIGH	2020-06-17	Fixed issue	• The issue that the returned IP address cannot be accessed in direct connection mode is fixed.
0.3.6	LOW	2020-06-09	New feature	• The role information can be contained in the Replication part of the **INFO** command output, such as `role:master`. This allows Redisson clients to call the role information in specific scenarios.
0.3.5	LOW	2020-06-05	New feature	• Statistics on QPS of read and write commands can be collected. For more information, see View monitoring data.
0.3.4	HIGH	2020-04-08	Fixed issue	• The issue of the system not responding when hotkeys are evicted is fixed. • The following issue is fixed: The system does not respond due to the use-after-free (UAF) vulnerability when the audit log feature is disabled.

Minor version	Update level	Release date	Type	Description
0.3.1	HIGH	2020-02-20	New feature	• The audit log feature is supported. This feature allows you to query, analyze online, and export logs. For more information, see Enable the new audit log feature. • The direct connection mode is supported. Clients can bypass proxy nodes and use private endpoints to connect to ApsaraDB for Redis instances. This is similar to the connection to a native Redis cluster. The direct connection mode can reduce communication overheads and the response time of ApsaraDB for Redis. For more information, see Enable the direct connection mode. • Both password-free access and Internet access are supported for an ApsaraDB for Redis instance deployed in a virtual private cloud (VPC). To access the instance from the Internet, apply for a public endpoint. For more information, see Enable password-free access. • The `oom_err_count` information can be contained in the **INFO** command output when the actual memory size is greater than the value specified for the maxmemory parameter.
			Fixed issue	• The following issue is fixed: The system does not respond because the expiration mechanism is triggered when the **RPOPLPUSH** command is run with the same source and destination key. • The following issue is fixed: Authentication failures occur when instances that are deployed in VPCs are accessed without using passwords.
0.2.0	LOW	2020-01-17	New feature	• Statistics of hotkeys can be collected in real time to help you identify hotkeys in instances. For more information, see Use the real-time key statistics feature.
0.1.2	LOW	2019-11-26	New feature	• Read-only Lua scripts can be executed on read replicas of read/write splitting instances. For more information, see Read/write splitting instances.
Earlier than 0.1.2	N/A	N/A	N/A	• These earlier minor versions belong to V5.0. We recommend that you update your instances to the latest minor version.

V4.0

Minor version	Update level	Release date	Type	Description
1.9.12	HIGH	2022-06-07	Security hardening	• Stability is enhanced.
1.9.10	HIGH	2022-05-23	Security hardening	• The configuration change stability is improved for cluster instances that have the direct connection mode enabled.
1.9.9	HIGH	2022-04-25	Security hardening	• Stability is enhanced.
1.9.8	HIGH	2022-04-25	Security hardening	• The following issue is fixed: The migration of large keys may fail during configuration changes of cluster instances that have the direct connection mode enabled.
1.9.6	HIGH	2021-10-15	Security hardening	• Stability is enhanced.
1.9.5	LOW	2021-09-13	New feature	• Statistical data of QPS is classified. Three types of QPS can be calculated: read QPS, write QPS, and other QPS.
1.9.4	HIGH	2021-08-05	Fixed issue	• The following issue is fixed: Configuration changes may fail when the direct connection mode is enabled for cluster instances.
1.9.3	MEDIUM	2021-07-20	Feature update	• Stability is enhanced.
1.9.2	HIGH	2021-04-19	Security hardening	• The security vulnerabilities of the open source LuaJIT compiler are fixed.
			New feature	• VIPs can be obtained over the Internet. This provides better support for private endpoint users.
1.9.1	MEDIUM	2021-03-08	Feature optimization	• Health checks for instances are improved to accelerate the switchover between the master and replica nodes when disk jitters occur. • The capability of memory-intensive instances to run the **BGSAVE** and **REWRITE** commands by calling fork() is improved. This prevents long pauses.

Minor version	Update level	Release date	Type	Description
1.9.0	LOW	2021-02-22	New feature	• When the `illegal address` error message is returned, the IP address of the client can be included in the error message. An IP address whitelist can be configured for your instance based on the IP address prompt. IP address prompt 
1.8.8	HIGH	2020-09-25	Fixed issue	• The issue that the output of the **CLUSTER NODES** command is inconsistent with that of open source Redis is fixed. Multiple slots are separated by spaces to prevent errors in client parsing.
1.8.7	LOW	2020-07-20	New feature	• The security group feature provided by ECS is supported to simplify O&M. To allow ECS instances to access an ApsaraDB for Redis instance, you can add the security groups to which the ECS instances belong to the ApsaraDB for Redis instance. You do not need to manually add the IP addresses of the ECS instances to the whitelists of the ApsaraDB for Redis instance. For more information, see Step 2: Configure whitelists.
1.8.6	HIGH	2020-07-14	Fixed issue	• Latency flag bits in audit logs are modified to help you identify these bits in master and replica audit logs.
1.8.5	LOW	2020-06-09	New feature	• The role information can be contained in the Replication part of the **INFO** command output, such as `role:master`. This allows Redisson clients to call the role information in specific scenarios.
1.8.4	LOW	2020-06-05	New feature	• Statistics on QPS of read and write commands can be collected. For more information, see View monitoring data.
1.8.3	HIGH	2020-04-08	Fixed issue	• The issue of the system not responding when hotkeys are evicted is fixed. • The issue of the system not responding due to the UAF vulnerability is fixed. This issue occurs when the audit log feature is disabled.

Minor version	Update level	Release date	Type	Description
1.8.1	LOW	2020-02-20	New feature	• Both password-free access and Internet access are supported for an ApsaraDB for Redis instance deployed in a VPC. To access the instance from the Internet, apply for a public endpoint. For more information, see Enable password-free access.
1.8.0	HIGH	2020-01-16	New feature	• Statistics of hotkeys can be collected in real time to help you identify hotkeys in instances. For more information, see Use the real-time key statistics feature.
			Fixed issue	• In direct connection mode, the output of the **INFO** command can contain the cluster_enabled information. This cluster_enabled information allows some SDKs to automatically negotiate to enter the cluster mode. For more information, see Enable the direct connection mode.
1.7.1	MEDIUM	2019-11-20	New feature	• Read-only Lua scripts can be executed on read replicas of read/write splitting instances. • The direct connection mode is supported. Clients can bypass proxy nodes and use private endpoints to connect to ApsaraDB for Redis instances. This is similar to the connection to a native Redis cluster. The direct connection mode can reduce communication overheads and the response time of ApsaraDB for Redis. For more information, see Enable the direct connection mode. • The memory statistics of Lua scripts can be contained in the Memory part of the **INFO** command output.
			Feature optimization	• Audit logs consume less memory.
1.5.8	HIGH	2019-09-23	Fixed issue	• The following issue is fixed: The atomicity of the **SETEX** command is destroyed during two-way synchronization in Global Distributed Cache for Redis links.
			New feature	• Audit logs can record latency events.

Minor version	Update level	Release date	Type	Description
			Fixed issue	• The following issue is fixed: Switchover may occur between the master and replica nodes due to slow requests when the client issues commands such as **KEYS**, **FLUSHALL**, and **FLUSHDB**.
1.5.4	LOW	2019-07-08	New feature	• The audit log feature is supported. This feature allows you to query, analyze online, and export logs. For more information, see Enable the new audit log feature. • Latency statistics during the entire event lifecycle are recorded to help you understand the state of the engine.
1.5.2	HIGH	2019-07-04	Fixed issue	• The following issue is fixed: The system does not respond because the expiration mechanism is triggered when the **RPOPLPUSH** command is run with the same source and destination key.
1.4.0	HIGH	2019-05-15	Fixed issue	• The following issue is fixed: Master/replica switchover is triggered when Redis database (RDB) files or append-only files (AOFs) are loaded onto an instance after the instance is restarted.
Earlier than 1.4.0	N/A	N/A	N/A	• These earlier minor versions belong to V4.0. We recommend that you update your instances to the latest minor version.

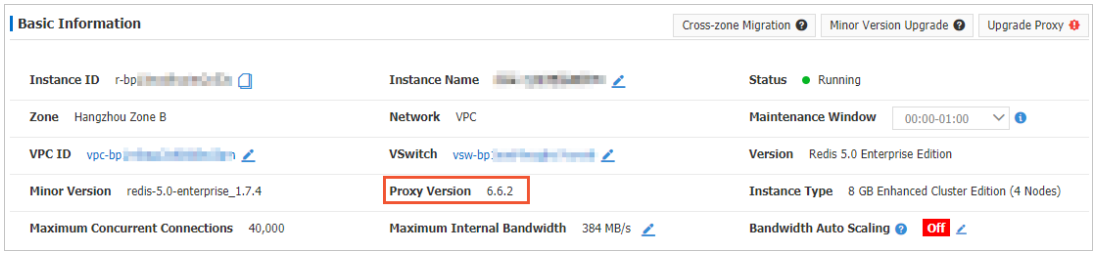
7.2.3. ApsaraDB for Redis proxy nodes

Alibaba Cloud releases minor versions for ApsaraDB for Redis proxy nodes from time to time to provide more features, fix known issues, and improve user experience. This topic describes the release notes for minor versions of proxy nodes. We recommend that you update the minor version of proxy nodes during off-peak hours.

View or update the minor version of proxy nodes

You can view the current minor version of proxy nodes that belong to an instance in the ApsaraDB for Redis console. For more information about update operations and usage notes, see Update the minor version.

View the minor version of proxy nodes



> **Note**
>
> - The system checks the minor version of proxy nodes that belong to your instance. If these proxy nodes are of the latest minor version, the **Upgrade Proxy** button is not displayed or is dimmed.
> - Minor version updates may differ from region to region. The minor version of an instance displayed in the ApsaraDB for Redis console prevails.

Overview of proxy nodes

Cluster instances and read/write splitting instances of ApsaraDB for Redis use proxy nodes to route commands, balance loads, and perform failovers. For more information, see Cluster master-replica instances and Read/write splitting instances. If you understand how proxy nodes route commands and handle specific commands, you can understand how to design more effective business systems. For more information about proxy nodes, see Features of proxy nodes.

Update levels

- LOW: regular updates. LOW-level updates include routine feature updates, such as adding a feature.
- MEDIUM: recommended updates. MEDIUM-level updates include optimization of features and modules. LOW-level updates are also included in MEDIUM-level updates.
- HIGH: major updates. HIGH-level updates include major updates that ensure stability or security, such as fixing a vulnerability or defect. LOW-level and MEDIUM-level updates are also included in HIGH-level updates.

6.8.x

Minor version	Update level	Release date	Type	Description
6.8.2	MEDIUM	2022-06-14	Feature optimization	• The stability is improved, and specific crashes are fixed.
6.8.1	LOW	2022-04-19	New feature	• Specific TairSearch commands are supported. For more information, see TairSearch. • New commands of TairRoaring V2.2 are supported. For more information, see TairRoaring.

Minor version	Update level	Release date	Type	Description
6.8.0	MEDIUM	2022-04-01	New feature	• Specific TairZset commands are supported. For more information, see TairZset. • Specific TairRoaring commands are supported. For more information, see TairRoaring. • The RC4 encryption algorithm is disabled for SSL certificates.
			Fixed issue	• The following issue is fixed: After the ptod_enabled parameter is enabled, an exception occurs when the SDIFFSTORE, SINTERSTORE, SUNIONSTORE, ZINTERSTORE, or ZUNIONSTORE command is run. For more information about the parameter, see Supported parameters. • The following issue is fixed: The CROSSSLOT error occurs when the SMOVE command is run.

6.7.x

Minor version	Update level	Release date	Type	Description
6.7.9	MEDIUM	2022-03-05	Fixed issue	• The following issue is fixed: `\n` in the command output is truncated when the DBSIZE or KEYS command is run on instances that have abnormal nodes.
6.7.8	MEDIUM	2022-03-03	Fixed issue	• The following issue is fixed: The SCRIPT DEBUG command cannot be used. • The following issue is fixed: The output score of the ZINTERSTORE or ZUNIONSTORE command has only six decimal places. • The following issue is fixed: An error message is returned when the SDIFF or SDIFFSTORE command is run on instances that use specific minor versions under Redis 5.0.
6.7.7	LOW	2022-01-30	Feature optimization	• The stability is enhanced.
6.7.6	LOW	2022-01-20	Feature optimization	• The stability is enhanced.
			Feature optimization	• The `RANDOMKEY` command is optimized to retrieve a different random node each time. This way, a node is not repeatedly retrieved when the `RANDOMKEY` command is run for several times.

Minor version	Update level	Release date	Type	Description
6.7.5	MEDIUM	2022-01-10	Fixed issue	• The following issue is fixed: The aggregations that derive from info commandstats on performance-enhanced instances encounter errors.
6.7.4	MEDIUM	2021-12-20	Feature optimization	• The stability is enhanced.
6.7.3	MEDIUM	2021-12-15	Fixed issue	• The following issue is fixed: After an SSL connection is established for an instance, the first sent request may not receive a response.
6.7.2	LOW	2021-11-30	Feature optimization	• The stability is enhanced.
6.7.1	MEDIUM	2021-11-23	Feature optimization	• The stability is enhanced.

6.6.x

Minor version	Update level	Release date	Type	Description
6.6.14	MEDIUM	2021-11-01	Feature optimization	• The following issue is fixed: In the Elastic Compute Service (ECS) architecture, the ZINTERSTORE and ZUNIONSTORE commands may fail to return responses when the split_multi_key_cmd_as_slot parameter is enabled.
6.6.13	MEDIUM	2021-10-22	Feature optimization	• The following issue is fixed: After the proxy query cache feature is enabled, hot upgrades may fail.
6.6.12	MEDIUM	2021-10-12	Feature optimization	• The stability is enhanced.
6.6.11	MEDIUM	2021-10-11	Feature optimization	• The stability is enhanced.
6.6.10	MEDIUM	2021-09-27	Fixed issue	• The following issue is fixed: An error message is returned when an ApsaraDB for Memcache instance processes only read or write requests.

Minor version	Update level	Release date	Type	Description
6.6.9	MEDIUM	2021-09-06	Fixed issue	• The CVE-2021-3711 and CVE-2021-3712 vulnerabilities are fixed.
6.6.8	MEDIUM	2021-08-30	Feature optimization	• The stability is enhanced.
6.6.7	MEDIUM	2021-08-27	Feature optimization	• Memory leaks that occur when the statistics feature is enabled are fixed.
6.6.6	LOW	2021-08-13	Feature optimization	• The stability is enhanced.
6.6.5	LOW	2021-08-03	New feature	• The Memcached gateway mode is supported. This allows you to use the Memcached protocol to forward requests.
6.6.4	HIGH	2021-07-08	New feature	• The **CLIENT LIST** or **CLIENT KILL** command can be used to query or manage client connections by process.
			Fixed issue	• The issue that TairZset commands do not support uppercase letters is fixed. For more information, see TairZset.
6.6.3	MEDIUM	2021-06-18	Feature optimization	• Internal management in cross-zone disaster recovery scenarios is optimized.
6.6.2	LOW	2021-06-08	New feature	• More internal commands of ApsaraDB for Redis are available.
6.6.1	LOW	2021-05-26	New feature	• The TairZset data structure is added. It allows you to sort score data of the DOUBLE type with respect to different dimensions. This data structure improves the data processing efficiency and is easy to use on the client side because you do not need to encode, decode, or encapsulate data. For more information, see TairZset.

Minor version	Update level	Release date	Type	Description
6.6.0	LOW	2021-04-28	New feature	• The proxy query cache feature is added. After you enable this feature, proxy nodes cache the request and response data of hotkeys. If a proxy node receives a duplicate request within the validity period of cached data, the proxy node directly returns a response to the client without the need to interact with backend data shards. This feature reduces access skew that occurs when a large number of read requests for hotkeys are sent. For more information, see Use proxy query cache to address issues caused by hotkeys.

6.5.x

Minor version	Update level	Release date	Type	Description
6.5.9	HIGH	2021-04-21	Fixed issue	• The following issue is fixed: Infinite loops occur when the commands used to manage multiple keys are run in special scenarios.
6.5.8	HIGH	2021-04-16	Fixed issue	• The following issue that occurs in 6.5.5 is fixed: Requests are out of order when multiple databases are requested. This minor version is a special version released based on 6.5.5.
6.5.7	HIGH	2021-04-16	Fixed issue	• The following issue is fixed: Requests are out of order when multiple databases are requested.
6.5.6	MEDIUM	2021-04-09	New feature	• The maximum number of data shards supported by the **SCAN** command increases from 256 to 1,024. • If the slots of the channels to which clients subscribe are migrated, proxy nodes close the client connections and reconnect to the clients. This ensures data consistency.
			Feature optimization	Proxy command processing is optimized. The following section describes the optimization: • Requests are resent to the address specified by the MOVED command when this command is run. • Proxy nodes do not route the commands without specified keys to the data shards that have empty slots.

Minor version	Update level	Release date	Type	Description
6.5.5	HIGH	2021-03-05	Fixed issue	• The following issue is fixed: Memory leaks may occur when distributed hash table (DHT) updates are triggered by master/replica switchovers or configuration changes. For more information, see Manually switch workloads from a master node to a replica node and Change the configurations of an instance.
6.5.4	HIGH	2021-02-07	Fixed issue	• The following issue is fixed: Memory leaks may occur when clients require a long time to receive responses.
6.5.3	HIGH	2021-01-21	New feature	• Variables can be used in the indexes of the KEYS array in Lua scripts.
			Fixed issue	• The following issue is fixed: Memory leaks occur when the MULTI or BLOCK command is run on a cluster instance that has more than 32 data shards. For more information, see Cluster master-replica instances.
6.5.2	HIGH	2021-01-19	Fixed issue	• The following issue is fixed: Virtual IP addresses (VIPs) cannot be obtained by using sockets in alb enat mode.
6.5.1	LOW	2021-01-14	New feature	• The IP address of the data shard that sent the most recent response can be recorded when slow logs record the commands used to manage multiple keys.
6.5.0	HIGH	2020-12-24	Fixed issue	• The following issue is fixed: The system may not respond when the **GIS.GETALL** command is run. For more information about this command, see TairGIS.

6.4.x

Minor version	Update level	Release date	Type	Description
6.4.10	MEDIUM	2020-12-01	Feature optimization	• The error message returned due to an invalid password is optimized to facilitate your understanding.

Minor version	Update level	Release date	Type	Description
6.4.9	HIGH	2020-11-06	Fixed issue	• The following issue is fixed: The system does not respond when SSL encryption is enabled in multi-threading mode. For more information, see Configure SSL encryption. • The following issue is fixed: The response protocol is incorrect due to channels that contain 0 when the **UNSUBSCRIBE** command is run.
6.4.8	HIGH	2020-10-21	Feature optimization	• Operational logs encode and then record binary requests for large packets, acknowledgment packets, and moved packets to prevent garbled characters for logs.
			Fixed issue	• The issue that the max_session_processing parameter cannot be dynamically configured is fixed. This parameter specifies the maximum number of pending requests that are allowed per connection. For more information about the parameter, see Supported parameters.
6.4.7	MEDIUM	2020-10-09	Feature optimization	• The internal monitoring of proxy nodes is optimized.
6.4.6	HIGH	2020-09-30	Fixed issue	• The following issue is fixed: The **SLOWLOG** command that is run on a standard or cluster instance times out because the node roles are not initialized. • The following issue is fixed: Connections to ApsaraDB for Memcache instances of specified specifications cannot be established by using Data Management (DMS). For more information, see Overview and What is ApsaraDB for Memcache? • The following issue is fixed: The system does not respond because no keys are specified when clients subscribe to `__keyspace@0__`.
6.4.5	LOW	2020-09-27	New feature	• More internal commands of ApsaraDB for Redis are available.
6.4.3	HIGH	2020-09-25	Feature optimization	• The special implementation of pipelines on the Jedis client is adapted to optimize the process to disable connections that are beyond the configured connection limit. For more information about Jedis connections, see Jedis client.

Minor version	Update level	Release date	Type	Description
			Fixed issue	• The following issue is fixed: The system incorrectly records slow logs by using the **BZPOPMIN** and **XREAD** commands. For more information, see View slow logs.
6.4.2	HIGH	2020-09-09	Fixed issue	• The issue that idle connections are automatically closed after 1 minute is fixed.
6.4.1	MEDIUM	2020-08-25	New feature	• Idle connections are automatically closed after a timeout configuration is added. • Statistics of slow logs on read replicas can be collected. This indicates that the **SLOWLOG** command is sent to the master node and all read replicas. For more information, see View slow logs.
			Feature optimization	• The memory usage of connections supported by the PubSub and MONITOR commands is optimized to prevent a rapid increase in memory usage due to memory fragments. • The capabilities of proxy nodes to handle new connections are improved.
6.4.0	HIGH	2020-08-18	Fixed issue	• The following issue is fixed: The system does not respond when you call the stat() function before configurations are complete by using a Configserver node.

6.3.x

Minor version	Update level	Release date	Type	Description
6.3.9	MEDIUM	2020-08-14	New feature	• The IP addresses of clients are recorded in slow logs to facilitate troubleshooting. For more information, see View slow logs.
			Feature optimization	• The capabilities of proxy nodes to handle short-lived connections are improved.
6.3.8	HIGH	2020-07-24	Fixed issue	• The following issue is fixed: Memory usage increases because the vector::clear() function does not take effect.

Minor version	Update level	Release date	Type	Description
6.3.7	HIGH	2020-07-13	Fixed issue	• The following issue is fixed: The system may not respond when you establish a connection after SSL encryption is enabled.
6.3.5	HIGH	2020-07-10	New feature	• Binary data in audit logs is encoded to improve log readability. • The **no_loose_statistics-ip-enable**, **no_loose_statistics-keys**, and no_loose_statistics-cmds parameters are added. You can set the parameters to collect statistics of specific IP addresses, keys, and commands. For more information, see Supported parameters.
			Fixed issue	• The following issue is fixed: After a connection is closed, the system may not respond when the `CheckExceedLimitAndClose` command is run. • The issue that the SSL encryption feature cannot be enabled is fixed.
6.3.4	HIGH	2020-05-21	Fixed issue	• The following issue is fixed: The system may not respond to subsequent requests due to empty packets such as `\r\n` .

8.Terms

This topic describes the terms of ApsaraDB for Redis.

Term	Description
ApsaraDB for Redis	A high-performance, key-value storage database service that supports caching and storage. ApsaraDB for Redis is developed based on BSD open source protocol.
instance ID	The ID of an instance. Each instance corresponds to a user space and serves as the basic unit of the ApsaraDB for Redis service. Different ApsaraDB for Redis instance specifications have different limits on instance capacities, such as the number of connections, bandwidth, and CPU capability. You can view the list of your instance IDs in the ApsaraDB for Redis console.
standard master-replica instance	An ApsaraDB for Redis instance that is built on top of a master-replica architecture. Master-replica instances provide limited capacity and performance. However, you can change the architecture of a master-replica instance to an instance of another architecture, such as a cluster instance or a read/write splitting instance.
cluster instance	An ApsaraDB for Redis instance that runs in a cluster architecture with scalability. Cluster instances provide higher scalability and performance. However, the instances provide limited features.
endpoint	The host address that is used to connect to ApsaraDB for Redis. The endpoint is displayed as a domain name. To obtain the endpoint, log on to the ApsaraDB for Redis console and choose **Instance Information > Connection Information**.
connection password	The password used to connect to an ApsaraDB for Redis instance. The password is in the `Instance ID:Custom password` format. For example, if you set the password to `1234` when you purchase an instance and the instance ID is `xxxx`, the connection password is `xxxx:1234`.
eviction policy	The eviction policy that is the same as that of the open source Redis. For more information, see Using Redis as an LRU cache.
database	The databases that can be created in an ApsaraDB for Redis instance. Each ApsaraDB for Redis instance can contain up to 256 databases: DB 0 to DB 255. By default, data is written into DB 0.