ALIBABA CLOUD

Alibaba Cloud

ApsaraDB for MongoDB

Product Introduction

Document Version: 20220207

Alibaba Cloud

Legal disclaimer

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 MongoDB?

ApsaraDB for MongoDB is a MongoDB-compatible database service that is developed based on the distributed system and high-reliability storage engine of Apsara. ApsaraDB for MongoDB uses a multi-node architecture to ensure high availability, and supports elastic scaling, disaster recovery, backup and restoration, and performance optimization.

Data structure of MongoDB

MongoDB is a document-oriented NoSQL database. MongoDB stores data in JSON-like documents that consist of field-value pairs. Example:

```
{
    name:"John",
    sex:"male",
    age:30
}
```

}

Storage structure of MongoDB

The storage structure of MongoDB is different from that of conventional relational databases. Data in MongoDB is organized at the following levels:

- Document: Documents are the basic unit of data in MongoDB. A document consists of BSON key-value pairs and is equivalent to a row in a relational database.
- Collection: A collection contains multiple documents. It is equivalent to a table in a relational database.
- Database: A database contains multiple collections. It is equivalent to a relational database. You can create multiple databases in an ApsaraDB for MongoDB instance.

Why ApsaraDB for MongoDB?

For more information about the benefits of ApsaraDB for MongoDB, see Comparison between ApsaraDB for MongoDB and self-managed databases and Scenarios.

Pricing

For more information, see Billing items and pricing.

Dedicated clusters

You can purchase ApsaraDB for MongoDB instances in ApsaraDB for MyBase dedicated clusters. For more information, see What is ApsaraDB for MyBase?

Deployment suggestions

Consider the following aspects when you create and use an ApsaraDB for MongoDB instance:

- Regions and zones

A region is an Alibaba Cloud data center. A zone is a physical area within a region that has its own independent power grid and network. The region and zone determine the physical location of an ApsaraDB for MongoDB instance. You cannot change the region of an ApsaraDB for MongoDB instance after the instance is created. For more information, see Regions and zones.

You can select a region and zone based on your geographical location, availability of Alibaba Cloud services, application availability requirements, and whether internal network communication is required. For example, if your application is deployed on an Elastic Compute Service (ECS) instance and requires an ApsaraDB for MongoDB instance to serve as its database, you must select the same region and zone as the ECS instance when you create your ApsaraDB for MongoDB instance.

> **Note**
>
> An ECS instance and an ApsaraDB for MongoDB instance in the same zone can be connected by using an internal network with minimal network latency.

- Network planning

  We recommend that you use Virtual Private Cloud (VPC) to connect to ApsaraDB for MongoDB instances. You can plan your own Classless Inter-Domain Routing (CIDR) blocks for VPCs. A VPC is an isolated network with higher security and performance than the classic network. You can use the default VPC or create your own VPC before you use ApsaraDB for MongoDB. For more information, see Configure a VPC for a new instance.

- Security solutions

  ApsaraDB for MongoDB eliminates your data security concerns by providing comprehensive security protection. You can ensure database security by means of zone-disaster recovery, Resource Access Management (RAM) authorization, audit logs, network isolation, whitelists, password authentication, and Transparent Data Encryption (TDE). For more information, see Best practices for data security of ApsaraDB for MongoDB.

Methods to manage ApsaraDB for MongoDB instances

You can use the following methods to manage ApsaraDB for MongoDB instances, such as creating instances, databases, and accounts, and specifying network-related settings:

- Console: The ApsaraDB for MongoDB console provides a graphical and easy-to-use web interface.
- API: All operations that are available in the console can be performed.

After you create an ApsaraDB for MongoDB instance, you can connect to the instance by using one of the following methods:

- Mongo shell: You can log on to the instance by using the official MongoDB CLI tool to manage databases. For more information, see Connect to a replica set instance by using the mongo shell.
- Client: ApsaraDB for MongoDB is compatible with the MongoDB protocol. You can use common database client tools such as Robo 3T and Studio 3T to connect to ApsaraDB for MongoDB instances.

Related services

- ECS: The best performance is achieved when you connect to ApsaraDB for MongoDB instances from ECS instances in the same region over an internal network. ECS and ApsaraDB for MongoDB instances compose a typical business architecture.

- Data Transmission Service (DTS): You can use DTS to migrate data from an on-premises MongoDB database to the cloud.
- Object Storage Service (OSS): OSS is a secure, cost-effective, and reliable cloud storage service that is provided by Alibaba Cloud. OSS allows you to store large volumes of data in the cloud.
- Data Lake Analytics (DLA): DLA is a new-generation big data solution that adopts an architecture of separated computing and storage. DLA provides serverless SQL and serverless Spark engines to meet the requirements of online interactive search, stream processing, batch processing, and machine learning. References:
    - Quick start for serverless SQL access to ApsaraDB for MongoDB
    - Quick start for serverless Spark access to ApsaraDB for MongoDB

2.Features

ApsaraDB for MongoDB is a MongoDB-compatible database service that is developed based on the Apsara distributed operating system and a high-reliability storage engine. ApsaraDB for MongoDB provides multi-node architectures to achieve high availability and supports various features, such as elastic scaling, disaster recovery, backup and recovery, and performance optimization. This topic describes the features of ApsaraDB for MongoDB.

Flexible architectures

ApsaraDB for MongoDB provides three different system architectures to meet your business requirements in various scenarios: standalone architecture, replica set architecture, and sharded cluster architecture. For more information, see Standalone instances, Architecture of replica set instances, and Architecture of sharded cluster instances.

Elastic scaling

You can change the specifications of an ApsaraDB for MongoDB instance based on your business requirements. The specifications include the instance type, storage capacity, and number of nodes. You can also specify the time at which you want to apply a specification change. We recommend that you apply a specification change during off-peak hours to prevent interruptions to your business. For more information, see Overview.

High data security

Security technology	Description
Anti-DDoS	ApsaraDB for MongoDB monitors inbound traffic in real time, filters source IP addresses to scrub large amounts of malicious traffic, and triggers blackhole filtering if traffic scrubbing becomes ineffective.
IP address whitelists	ApsaraDB for MongoDB filters traffic from IP addresses to achieve high-level security protection. You can configure up to 1,000 IP addresses and CIDR blocks in each IP address whitelist. For more information, see Configure a whitelist or an ECS security group for an ApsaraDB for MongoDB instance.
VPC	A virtual private cloud (VPC) is an isolated virtual network that provides higher security and higher performance than the classic network. Before you deploy services in VPCs, you must create VPCs. For more information, see Default VPC and default vSwitch.

Security technology	Description
Disaster recovery	ApsaraDB for MongoDB provides a zone-disaster recovery solution to achieve high reliability and high data security. When you create an ApsaraDB for MongoDB instance, you can select multiple zones. For more information, see Create a multi-zone replica set instance or Create a multi-zone sharded cluster instance. You can also migrate a replica set instance from a single zone to multiple zones. For more information, see Migrate an ApsaraDB for MongoDB instance to different zones in the same region. **Note** You can migrate an ApsaraDB for MongoDB instance to different zones only when the instance is a replica set instance that runs MongoDB 4.2 or earlier and transparent data encryption (TDE) is not enabled for the instance.
SSL encryption	ApsaraDB for MongoDB encrypts network connections at the transport layer in compliance with SSL to improve data security and ensure data integrity. For more information, see Configure SSL encryption for an ApsaraDB for MongoDB instance.
TDE	ApsaraDB for MongoDB performs real-time I/O encryption and decryption on data files. Data is encrypted before it is written into a disk. Data is also decrypted when it is read from a disk and written into the memory. TDE does not increase the size of data files. You can use TDE without the need to modify the configuration data of your application. For more information, see Configure TDE for an ApsaraDB for MongoDB instance.
Automatic backups	You can configure an ApsaraDB for MongoDB instance to create automatic backups during off-peak hours. For more information, see Configure automatic backup for an ApsaraDB for MongoDB instance.
Temporary backups	You can manually create physical backups and logical backups for an ApsaraDB for MongoDB instance. For more information, see Manually back up an ApsaraDB for MongoDB instance.
Data recovery	You can create an ApsaraDB for MongoDB instance by using the data from a backup file or the data at a specific point in time. You can also restore the data of an ApsaraDB for MongoDB instance to that instance. For more information, see Restore backup data to a new ApsaraDB for MongoDB instance by backup point, Restore backup data to a new ApsaraDB for MongoDB instance by point in time, and Restore backup data to the current instance.
Backup file download	ApsaraDB for MongoDB retains your backup files free of charge for up to seven days. During this retention period, you can log on to the ApsaraDB for MongoDB console and download the backup files to your computer.

Comprehensive monitoring

ApsaraDB for MongoDB monitors up to 20 metrics, such as the disk space usage, input/output operations per second (IOPS), number of connections, CPU utilization, network traffic, transactions per second (TPS), queries per second (QPS), and cache hit ratio. For more information, see View monitoring data.

Professional tools

Data Management (DMS) allows you to manage relational databases such as MySQL databases, SQL Server databases, and PostgreSQL databases. DMS also allows you to manage NoSQL databases such as MongoDB databases and Redis databases. DMS supports Linux servers. For more information, see Overview. DMS is a comprehensive data management service that provides various features, such as data management, schema management, server management, access control, business intelligence (BI) charts, trend analysis, data tracking, and performance monitoring and optimization.

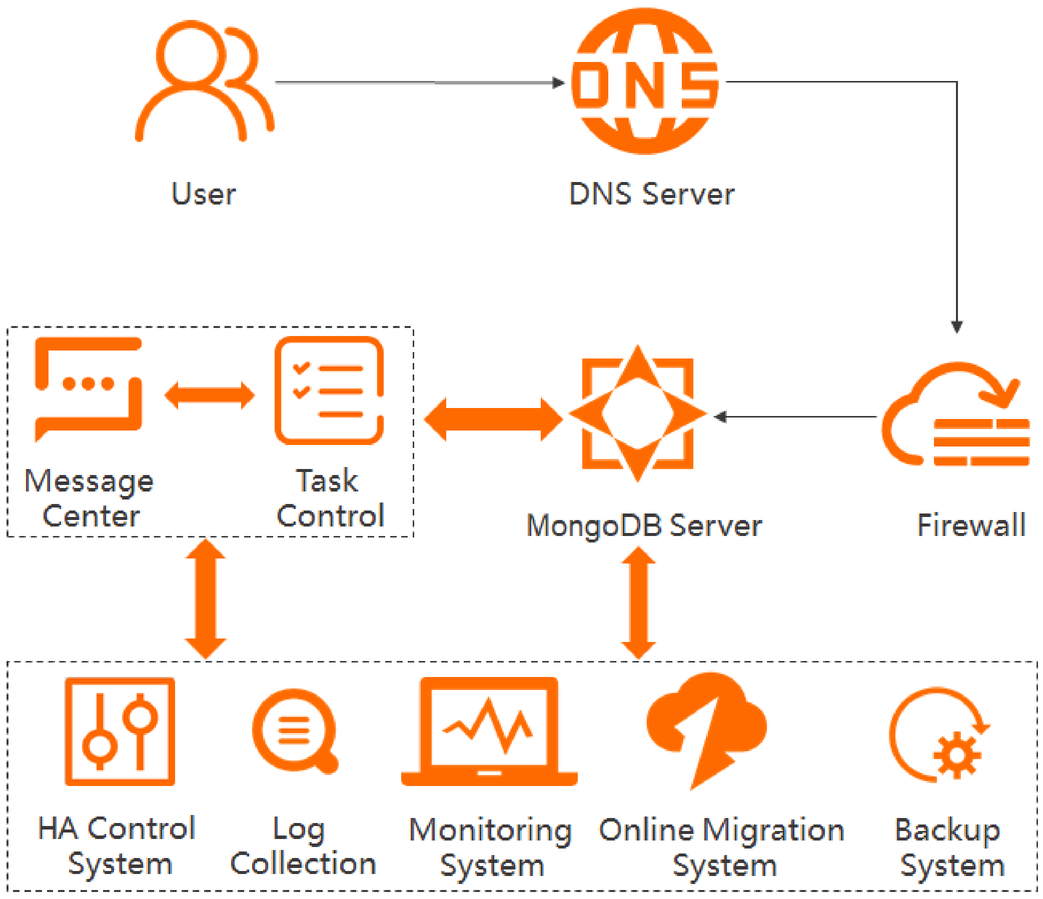
Data Transmission Service (DTS) is a data service that is provided by Alibaba Cloud to support data exchanges between data sources such as relational databases, NoSQL databases, and online analytical processing (OLAP) systems. For more information, see What is DTS? DTS provides data transmission capabilities, such as data migration, real-time data subscription, and real-time data synchronization. DTS is suitable for scenarios such as data migration without downtime, geo-disaster recovery, cross-border data synchronization, and cache refreshing. DTS helps you build a secure, scalable, and highly available data architecture.

3.System architecture

3.1. Architecture of ApsaraDB for MongoDB

This topic describes the architecture and components of ApsaraDB for MongoDB.

Architecture



Components

- Task control system

  ApsaraDB for MongoDB instances support various tasks, such as instance creation, configuration change, and instance backup. You can use the system to control tasks, track tasks, and manage errors.

- HA control system

  The system acts as a high-availability detection module to detect the running status of ApsaraDB for MongoDB instances. If the system determines that the primary node of an ApsaraDB for MongoDB instance is unavailable, the system fails over to a secondary node to maintain the availability of the instance. You can also manually switch over between the primary and secondary nodes. For more information, see Switch node roles.

- Log collection system

The system collects the running logs of ApsaraDB for MongoDB, such as slow query and audit logs. For more information about the benefits of ApsaraDB for MongoDB, see Overview and Enable the new audit log feature for an ApsaraDB for MongoDB instance.

- Monitoring system

  The system monitors the performance of ApsaraDB for MongoDB instances and collects information such as their basic metrics, disk capacities, access requests, and input/output operations per second (IOPS). For more information, see View monitoring data.

- Backup system

  The system backs up ApsaraDB for MongoDB instances and stores the generated backup files in Object Storage Service (OSS). It allows you to customize the backup policy to manually or automatically back up the instances. It retains the backup files for up to seven days. For more information about the benefits of ApsaraDB for MongoDB, see Configure automatic backup for an ApsaraDB for MongoDB instance and Manually back up an ApsaraDB for MongoDB instance.

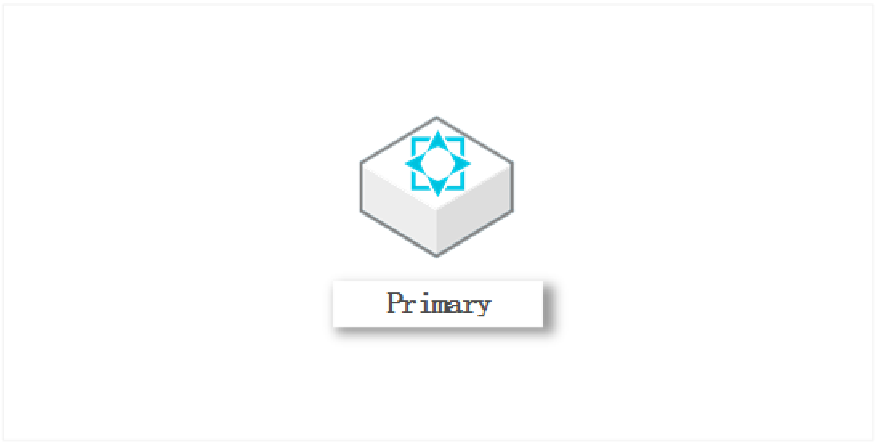
- Online migration system

  If the physical server where the instance resides fails, the system creates a new instance from the backup files in the backup system to prevent impacts on your business. For more information, see Overview.

3.2. Standalone instances

ApsaraDB for MongoDB standalone instances provide a high level of data fault tolerance and are suitable for database systems that do not store crucial data. For example, you can use standalone instances in scenarios such as development, testing, learning, and training.

Architecture



- Standalone instances provide only one primary node for reading and writing data.
- Standalone instances allow you to enjoy the O&M support and kernel optimization of ApsaraDB MongoDB at lower prices. You can select appropriate instance specifications for different business scenarios to reduce costs. For more information about the specifications of standalone instances, see Standalone instance types.

Supported MongoDB versions

Only MongoDB 3.4 and 4.0 are supported. You can create standalone instances of MongoDB versions based on your business requirements. For more information about how to create standalone instances, see Create a standalone instance.

For more information about MongoDB versions, see MongoDB 5.0.

FAQ

- Do standalone instances provide high availability?

  No. Standalone instances have only one node and cannot provide high availability. A failure may cause service interruption for more than 30 minutes in extreme cases. We recommend that you use the replica set architecture in production environments.

- Do standalone instances support incremental data migration and synchronization and point-in-time data restoration to a new instance?

  No. Standalone instances do not support oplogs. Therefore, standalone instances do not support incremental data migration and synchronization or point-in-time data restoration to a new instance.

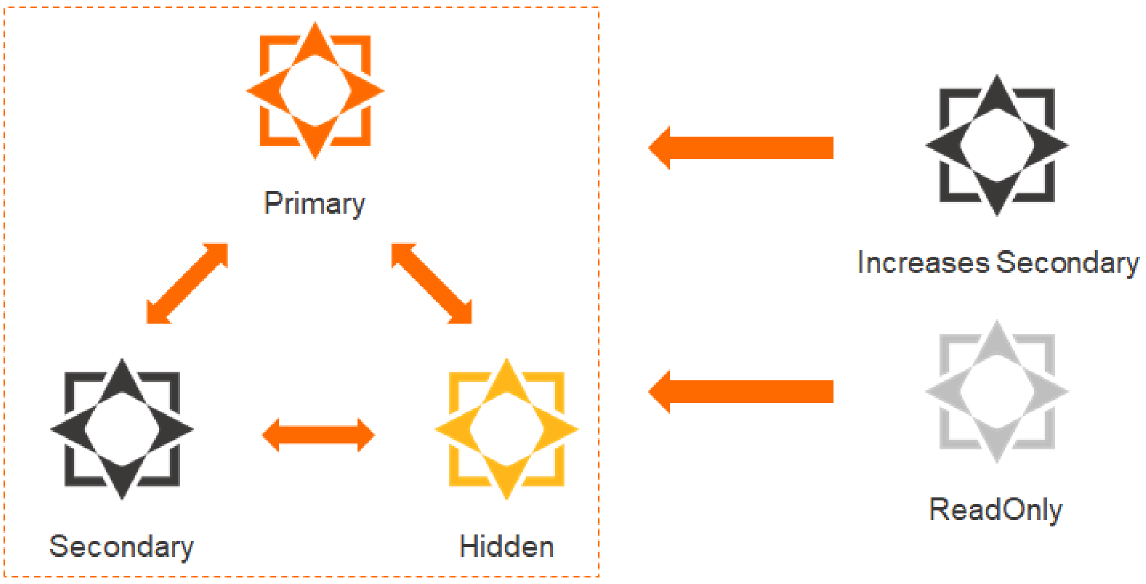
Related information

- Architecture of replica set instances
- Architecture of sharded cluster instances
-

3.3. Architecture of replica set instances

ApsaraDB for MongoDB automatically configures replica set instances. You can manage the primary node and secondary nodes of a replica set instance. Replica set instances provide advanced features such as disaster recovery and failover. When you use replica set instances, these advanced features are enabled by default.

Architecture



ApsaraDB for MongoDB uses a multi-node architecture to ensure high availability. A replica set instance consists of a primary node, one or more secondary nodes, a hidden node, and one or more optional read-only nodes. The following section describes the node types in a replica set instance:

- Primary node: processes all read and write operations. Each replica set instance contains only one primary node.
- Secondary node: synchronizes data from the primary node by using operation logs. If the primary node fails, a secondary node can be elected as the new primary node to ensure high availability.

  Note If a replica set instance is connected by using the connection string of a secondary node, you can only read data from the instance. You cannot write data to the instance.

- Hidden node: synchronizes data from the primary node by using operation logs. If a secondary node fails, the hidden node can be elected as a new secondary node to ensure high availability.

  Note The hidden node is used only to ensure high availability. It is invisible to users.

- Read-only node: In business scenarios where a large number of read requests exist, you can use read-only nodes to relieve pressure on the primary and secondary nodes. For more information, see ApsaraDB for MongoDB read-only nodes.

Scale out nodes in a replica set instance

ApsaraDB for MongoDB allows you to scale out the number of nodes in an instance. You can increase the number of secondary nodes or read-only nodes based on your business needs. For more information, see Change the configurations of a replica set instance.

Note Each replica set instance contains only one hidden node. More secondary and read-only nodes can be added to a replica set instance, whereas the number of hidden nodes cannot be increased.

For example, you run websites that provide online reading services or run systems that provide order queries. These websites and systems process a large number of read operations and a small number of write operations. In addition, the number of operations on these websites or systems may surge due to impromptu events. In these scenarios, you can add or remove secondary nodes or read-only nodes to adjust the read capability of your replica set instance.

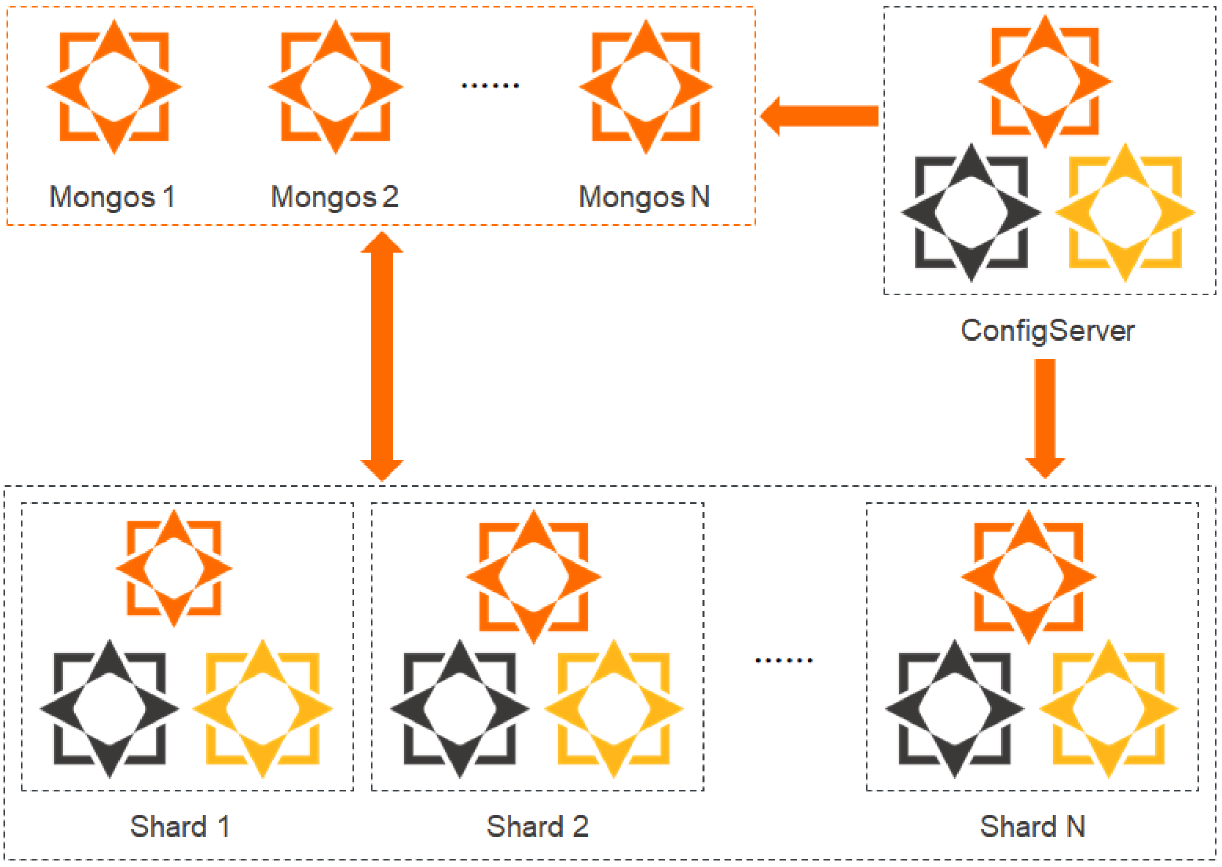
Related information

- Standalone instances
- Architecture of sharded cluster instances

3.4. Architecture of sharded cluster instances

A sharded cluster instance consists of three components: mongos, shard, and Configserver. You can choose the configuration and number of mongos and shards to create ApsaraDB for MongoDB sharded cluster instances that have different performance.

Architecture



Components

Component	Architecture	Description
Mongos	Standalone architecture	Routes queries and writes to the corresponding shards. You can purchase multiple mongos in the console to implement load balancing and failover. A single sharded cluster instance supports 2 to 32 mongos.
Shard	Replica set architecture (three nodes)	Stores database data. You can purchase multiple shards in the console to scale out the capacity of data storage and concurrent read/write operations. A single sharded cluster instance supports 2 to 32 shards.

Component	Architecture	Description
Configserver	Replica set architecture (three nodes)	Stores the metadata for clusters and shards. The metadata contains the data distribution information about each shard in a cluster. **Note** You cannot change the specifications of the Configserver (one core, 2 GB memory, and 20 GB disk storage capacity).

Related information

- Standalone instances
- Architecture of replica set instances

4.Benefits

ApsaraDB for MongoDB is a MongoDB-compatible database service that is developed based on the Apsara system and a high-reliability storage engine. ApsaraDB for MongoDB uses a multi-node architecture to ensure high availability, and supports elastic scaling, disaster recovery, backup and restoration, and performance optimization.

Flexible deployment architectures

ApsaraDB for MongoDB supports multiple deployment architectures to meet the requirements of different business scenarios.

Deployment architectures of ApsaraDB for MongoDB:

- Architecture of standalone instances

  Standalone instances are suitable for development, testing, education, and scenarios where non-core enterprise data is stored. You can select the instance specifications that are most suitable for your business scenarios to minimize costs. For more information, see Standalone instances.

- Architecture of replica set instances

  Replica set instances are suitable for burst traffic scenarios that require significantly more reads than writes or temporary activities. A replica set instance consists of a primary node that supports read and write operations, one, three, or five high-availability secondary nodes, a hidden node, and up to five optional read-only nodes. You can add or remove secondary nodes and read-only nodes based on your business needs. For more information, see Architecture of replica set instances.

- Architecture of sharded cluster instances

  Sharded cluster instances are suitable for scenarios that require highly concurrent read and write operations. A sharded cluster instance is based on multiple three-node replica set instances. Each replica set instance contains three nodes in primary/secondary mode and up to five optional read-only nodes. A sharded cluster instance consists of three types of components: mongos, shard, and Configserver nodes. You can specify the number and specifications of mongos and shard nodes to create sharded cluster instances that have different service capabilities. For more information, see Architecture of sharded cluster instances.

Elastic scaling

ApsaraDB for MongoDB allows you to change instance configurations to adapt to shifting business needs. You can change the configurations of an instance (instance specifications, storage capacity, and number of nodes) based on your business needs. You can also specify that the new configurations take effects during off-peak hours to avoid possible impacts on your business. For more information, see Overview.

Support for Alibaba Cloud developed tools

ApsaraDB for MongoDB allows you to migrate and synchronize data by using the ApsaraDB for MongoDB console or native MongoDB tools. In addition, you can use the following Alibaba Cloud developed tools.

Tool	Description

Tool	Description
NimoShake	A data synchronization tool. You can use this tool to migrate Amazon DynamoDB databases to Alibaba Cloud. For more information, see Migrate an Amazon DynamoDB database to ApsaraDB for MongoDB by using NimoShake.
MongoShake	A general tool developed in the Go language to synchronize data. You can use this tool to synchronize data among MongoDB databases. For more information, see Use MongoShake to implement one-way synchronization between ApsaraDB for MongoDB replica set instances.
NimoFullCheck	A tool used to verify the data consistency between source DynamoDB and destination MongoDB databases. You can use this tool to check data consistency when you migrate data in a DynamoDB database to an ApsaraDB for MongoDB instance. For more information, see Use NimoFullCheck to check data consistency after migration.

Others

ApsaraDB for MongoDB also excels in service availability, data reliability, security, and O&M costs. For more information, see Comparison between ApsaraDB for MongoDB and self-managed databases.

5.Scenarios

ApsaraDB for MongoDB supports standalone, replica set, and sharded cluster deployment architectures and provides enterprise-class capabilities such as security audit and point-in-time backup. These features have been widely used in the Internet, IoT, gaming, and finance industries.

Read/write splitting

ApsaraDB for MongoDB uses the architecture of three-node replica sets to guarantee high availability. Three data nodes are located on different physical servers and automatically synchronize data. The primary and secondary nodes are configured with different endpoints. MongoDB drivers allocate read/write requests to them. For more information about the architecture, see Architecture of ApsaraDB for MongoDB.

Flexible business scenarios

ApsaraDB for MongoDB has no schema and is suitable for startup business needs. You do not need to worry about changing schemas. You can store structured data in ApsaraDB RDS, flexible business data in ApsaraDB for MongoDB, and hot data in ApsaraDB for Redis or ApsaraDB for Memcache. This helps you write and read business data with high efficiency and reduce the cost of data storage.

Apps

ApsaraDB for MongoDB supports two-dimensional spatial indexes. Therefore, ApsaraDB for MongoDB can provide support for location-based apps. Its dynamic storage mode is also suitable for storing heterogeneous data from multiple systems. This satisfies the requirements of apps.

IoT scenarios

ApsaraDB for MongoDB has features such as high performance and asynchronous data writing. It can achieve the processing capability of an in-memory database in specific scenarios. In a sharded cluster instance of ApsaraDB for MongoDB, you can adjust the configuration and quantity of mongos and shards to improve performance and expand storage space without limits. ApsaraDB for MongoDB is suitable for IoT scenarios with highly concurrent write operations. For more information, see Overview.

ApsaraDB for MongoDB provides a secondary index feature for dynamic queries. It can use the MapReduce aggregation framework of MongoDB to conduct multidimensional data analysis.

Applications in various fields

- Game applications: use ApsaraDB for MongoDB as a database for game servers to store user information. Gaming equipment and credits of users are directly stored in embedded documents to facilitate queries and updates.
- Logistics applications: use ApsaraDB for MongoDB to store order information. The order status is constantly updated during the shipping process and is stored in the form of an embedded array in ApsaraDB for MongoDB. You can read all the changes in an order by using a single query, which is convenient, quick, and clear.
- Social networking applications: use ApsaraDB for MongoDB to store user information and the information of the WeChat moments published by users. They can use geographical location indexes to search nearby people and places. Additionally, ApsaraDB for MongoDB is suitable for storing chat history because ApsaraDB for MongoDB provides rich queries and is fast in both writing and reading.
- Live video streaming: uses ApsaraDB for MongoDB to store user information and gift information.
- Big data applications: uses ApsaraDB for MongoDB as the cloud storage system for big data.

ApsaraDB for MongoDB can extract and analyze data at any time to master industry trends.

6.Terms

This topic describes the terms that are used in ApsaraDB for MongoDB.

Term	Description
region	• The geographical location of the server for the ApsaraDB for MongoDB instance that you create. You must specify a region when you create an ApsaraDB for MongoDB instance. The region cannot be changed after the instance is created. • When you create an ApsaraDB for MongoDB instance, you must use an Elastic Compute Service (ECS) instance. ApsaraDB for MongoDB supports access over the internal network. The specified region must be the same as that of the ECS instance. For more information, see Connect to an ApsaraDB for MongoDB instance over the internal network across zones.
zone	• A physical area with an independent power supply and network in a region. • Resources in zones that are in the same region can communicate with each other over the internal network. The network latency within a zone is lower that across zones. Faults are isolated between zones. • In the single-zone deployment mode, the three nodes of an ApsaraDB for MongoDB replica set instance are deployed in the same zone. If an ECS instance and an ApsaraDB for MongoDB instance are both deployed in the same zone, network latency is reduced.
instance	• An ApsaraDB for MongoDB instance, which is the basic unit of ApsaraDB for MongoDB. • An instance is the runtime environment for ApsaraDB for MongoDB and exists as a separate process on a host. • You can create, modify, and delete an instance by using the ApsaraDB for MongoDB console. Instances are independent and their resources are isolated. They do not compete for resources such as CPU, memory, and I/O. • Each instance has its own unique features, such as database engine and version. ApsaraDB for MongoDB controls instance behavior by using the parameters that correspond to the features.
memory	The maximum memory that an instance can use.
disk capacity	• The disk size that you select when you create an ApsaraDB for MongoDB instance. • The disk capacity occupied by the instance includes datasets and the space required for normal instance operations, such as the system database, database rollback log, redo log, and indexes. • Make sure that an ApsaraDB for MongoDB instance has sufficient disk capacity to store data. Otherwise, the instance may be locked. If the instance is locked due to insufficient disk capacity, you can unlock the instance by expanding the disk capacity.
IOPS	The maximum number of read/write operations performed per second on block devices at a granularity of 4 KB.

Term	Description
CPU core	The maximum computing power of an ApsaraDB for MongoDB instance. A single Intel Xeon series CPU core has at least 2.3 GHz of computing power with hyper-threading capabilities.
connections	The number of Transmission Control Protocol (TCP) connections between a client and an ApsaraDB for MongoDB instance. If the client uses a connection pool, the connections between the client and the instance are persistent connections. Otherwise, they are short-lived connections.
sharded cluster instance	An ApsaraDB for MongoDB sharded cluster instance. You can purchase multiple mongos nodes, multiple shard nodes, and a Configserver node to create an ApsaraDB for MongoDB sharded cluster instance, which serves as a MongoDB distributed database system.
mongos	• The routing service that processes requests. All requests must be coordinated by using mongos that serves as a request distribution center and forwards data requests to the corresponding shard server. • You can use multiple mongos nodes to process requests. If one mongos node fails, other mongos nodes can continue to process the requests.
shard	• An ApsaraDB for MongoDB instance that holds a subset of the sharded data. • Each shard can be deployed as a three-node replica set to increase availability. You can create multiple shards to improve read and write performance and expand storage capacity. This way, you can implement a distributed database system based on your application performance and storage requirements.
Configserver	• A configuration server that stores all database metadata for mongos nodes and shards in an ApsaraDB for MongoDB sharded cluster instance. Mongos nodes cache shard data and data routing information in their memory, whereas Configservers store such data. • When mongos nodes in a sharded cluster instance are started for the first time or shut down and then restarted, they load configuration information from the Configserver node. If the information of the Configserver node changes, all mongos nodes are notified to update their status. This ensures that mongos nodes can always obtain the correct routing information. • Configserver nodes store metadata of shards and routers and have high requirements for service availability and data reliability. ApsaraDB for MongoDB uses three-node replica sets to ensure the reliability of the Configserver nodes.

7.Release notes

7.1. Features of MongoDB 5.0

MongoDB 5.0 marks a new release cycle to deliver new features to users faster than in the past. The combination of Versioned API and live resharding eliminates issues during database upgrade and business changes. The native time series platform enables MongoDB to support a wider range of workloads and business scenarios. The new Mongo Shell improves user experience. This topic describes the key new features of MongoDB 5.0.

Native time series platform

MongoDB 5.0 natively supports the entire lifecycle of time series data, from ingestion, storage, query, real-time analysis, and visualization to online archival or automatic expiration as data ages. This streamlines the building and running of time series applications and lowers costs. In version 5.0, MongoDB has expanded the universal application data platform to make it easier for developers to process time series data. This further extends the application scenarios of MongoDB to areas such as IoT, financial analysis, and logistics.

MongoDB time series collections automatically store time series data in a highly optimized and compressed format, thus reducing storage size and I/O to deliver better performance on a larger scale. Time series collections also shorten the development cycle so that you can quickly build models optimized for the performance and analysis requirements of time series applications.

You can run the following command to create a time series data collection:

```
db.createCollection("collection_name",{ timeseries: { timeField: "timestamp" } } )
```

MongoDB seamlessly adjust the acquisition frequency and automatically process out-of-order measurement values based on dynamically generated time partitions. Newly released MongoDB Connector for Apache Kafka implements local support for time series. You can directly create a time series collection from Kafka topic messages. This enables you to process and aggregate data during data collection, and then write them to a MongoDB time series collection.

The time series collection automatically creates a clustered index of data sorted by time to reduce data query latency. The MongoDB Query API also extends the window function so that you can run analytical queries (such as moving averages and cumulative sums). In relational database systems, analytical queries are usually referred to as SQL analysis functions and support windows defined in units of rows such as three-line moving averages. MongoDB adds powerful time series functions such as exponential moving average (EMA), derivative, and integral. This allows you to define a window in units of time such as a 15-minute moving average. Window functions can be used to query MongoDB time series and regular collections, which provides new analysis methods for multiple application types. Moreover, MongoDB 5.0 provides new time operators, including `$dateAdd` , `$dateSubstract` , `$dateDiff` , and `$dateTrunc` . This allows you to summarize and query data on a custom time window.

You can manage both MongoDB time series data and other data of your enterprise. Time series collections can be put together with regular MongoDB collections in the same database. You do not need to select a dedicated time series database that is unable to provide services for other types of applications, nor use complex integration to mix time series data and other data. MongoDB provides a unified platform that allows you to build high-performance and efficient time series applications, and also supports other use cases or workloads. This eliminates the cost and complexity of integrating and running multiple different databases.

Live data resharding

Configure the database version	Description	Implementation method
Earlier versions of MongoDB	The resharding process is complex and requires manual operations.	• Method 1: First dump the entire collection, and then reload the database into a new collection by using the new shard key. This process requires offline processing. Your application will remain in suspension for a long period of time while the reload is being completed. For example, the dumping and reloading of a collection that is larger than 10 TB on a three-shard cluster may take several days to complete. • Method 2: Create a new sharded cluster, configure shard keys for collections, and then perform custom migration to write the collection that you want to reshard from the existing sharded cluster to the new sharded cluster based on the configured shard keys. ◦ During this process, you must handle the query routing and migration logic and constantly check the migration progress to ensure that all data is migrated. ◦ Custom migration is a highly complex, labor-intensive, and time-consuming task that may incur risks. For example, one MongoDB user spent three months migrating 10 billion documents.

Configure the database version	Description	Implementation method
MongoDB 5.0	• You can run the **reshardCollection** command to start resharding. • The resharding process is efficient. Instead of simply rebalancing data, the resharding process copies and rewrites all of the current collection data to the new collection in the background. During this process, new writes of the application are also synchronized. • The resharding process is fully automated. The period of time of resharding is shortened from weeks or months to minutes or hours, and lengthy and complex manual data migration is avoided. • By means of online resharding, you can easily evaluate the effect of different shard keys in a development or test environment. You can also modify the shard keys.	You can change the shard key for your collection on demand as your workload grows or evolves. No database downtime or complex migration within the dataset is required in this process. You can run the **reshardCollection** command in the MongoDB Shell to select the database and collection that you want to reshard and specify the new shard key. `reshardCollection: "<database>.<collection>", key: <shardkey>` Note • <database>: the name of the database that you want to reshard. • <collection>: the name of the collection that you want to reshard. • <shardkey>: the name of the shard key. • When you run the **reshardCollection** command on MongoDB, it clones an existing collection and then applies all operations logs in the existing collection to a new one. After applying all operations logs, MongoDB switches to the new collection and deletes the existing collection.

Versioned API

- Improves compatibility with applications

  The Versioned API feature of MongoDB 5.0 defines a set of commands and parameters that are most commonly used in applications. These commands remain unchanged for all database releases, including annual major releases and quarterly rapid releases. As a result, the application lifecycle is decoupled from the database lifecycle, which allows you to pin the driver to a specific version of the MongoDB API. This way, even after your database is upgraded, your application can continue to run for several years without the need to modify any code.

- Allows you to flexibly add new features and improvements

The Versioned API allows you to add new features to the database of each version with full backward compatibility. When you change an API, you can run a new version of the API on the same server at the same time as the existing version of the API. As new MongoDB versions are released at a faster pace, the Versioned API feature provides easier access to the features of the latest versions.

Default majority write concern.

Starting from MongoDB 5.0, the default ranking of Write Concern is majority. A write operation is committed, and write success is passed back to the application only when the write operation is applied to the primary node and persisted to the logs of a majority of secondary nodes. This ensures that MongoDB 5.0 provides stronger data durability guarantees out of the box.

Note The Write Concern is fully tunable. You can set the Write Concern to balance database performance and data durability.

Optimization of connection management

By default, a client connection corresponds to a thread on the backend MongoDB server. In other words, net.serviceExecutor is set to `synchronous`. Large amounts of system resources are required for creating, switching, or destroying threads. When a large number of client connections exist, threads consume large amounts of resources

Situations where the number of connections is large or the creation of connections is out of control is referred to as a "connection storm". A connection storm may occur due to a variety of reasons. It often occurs when the services already slow down.

In response, MongoDB 5.0 takes following measures:

- Limit the number of connections that the driver attempts to create, protecting the database server from being overloaded.
- Reduce the the frequency at which the driver checks connection pools, allowing unresponsive or overloaded server nodes to buffer and recover.
- Allow the driver to switch to a faster server that has the healthiest connection pool rather than selecting an available server at random.

The preceding measures and the improvements made at the mongos query routing layer in previous versions further enhance the capabilities of MongoDB to handle high concurrency.

Long-running snapshot queries.

Long-running snapshot queries improve the versatility and flexibility of applications. By default, snapshot queries executed by this feature have a duration of 5 minutes. The execution duration is customizable. In addition, this feature maintains strong consistency with snapshot isolation guarantees without affecting the performance of your live and transactional workloads, and allows you to execute snapshot queries on secondary nodes. This allows you to run different workloads in a single cluster and scale your workloads to different shards.

MongoDB implements long-running snapshot queries by means of a project called Durable history in the underlying storage engine. The project has been available since MongoDB 4.4. Durable history stores a snapshot of all field values that have changed since the outset of a query. Queries can use durable history to maintain snapshot isolation. If data is changed, durable history also helps alleviate the caching pressure of the storage engine and enables higher query throughput in high write load scenarios.

New MongoDB Shell

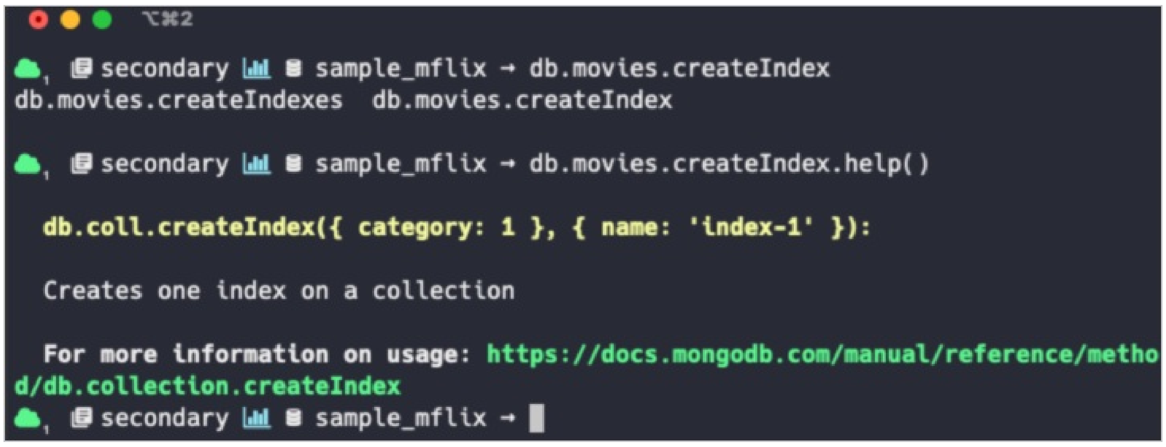
For better user experience, the MongoDB Shell has been redesigned from the ground up to provide a modern command-line experience, enhanced usability features, and a powerful scripting environment. The new MongoDB Shell has become the default shell for MongoDB. The new MongoDB Shell introduces syntax highlighting, intelligent auto-complete, contextual help, and useful error messages to create a visualized and interactive experience.

- Enhanced user experience
  - Easier implementation of queries and aggregations, and improved readability

    The new MongoDB Shell supports syntax highlighting, which makes it easy for you to distinguish fields, values, and data types to avoid syntax errors. If errors persist, the new MongoDB Shell can pinpoint the issue and provide solutions.
  - Faster query and command typing

    The new MongoDB Shell provides the intelligent auto-complete feature. The new MongoDB Shell can provide auto-complete prompts for methods, commands, and MQL expressions based on your MongoDB version.

    If you forget the syntax of a command, you can quickly search it in the MongoDB Shell. Sample syntax:



```
⌥⌘2
secondary  sample_mflix → db.movies.createIndex
db.movies.createIndexes  db.movies.createIndex

secondary  sample_mflix → db.movies.createIndex.help()

  db.coll.createIndex({ category: 1 }, { name: 'index-1' }):

  Creates one index on a collection

  For more information on usage: https://docs.mongodb.com/manual/reference/metho
d/db.collection.createIndex
secondary  sample_mflix → █
```

- Advanced scripting environment

  The scripting environment of the new MongoDB Shell is built on top of the Node.js REPL interactive interpreter. You can use all Node.js APIs and npm modules in your scripts. You can also load and run scripts from file systems. In the new MongoDB Shell, you can continue to use the load() method and eval() function to execute scripts in the same manner as you would in previous versions of MongoDB Shell.
- Expandability and plug-ins

  The new MongoDB Shell can be easily expanded. This allows you to use all the features of MongoDB to boost productivity.

  The Snippets plug-in can be installed In the new MongoDB Shell. Snippets can be automatically loaded to the MongoDB Shell, and Snippets can use all Node.js APIs and npm packages. MongoDB maintains a Snippets repository that provides features such as analyzing plug-ins for specified collection patterns. You can also configure the MongoDB Shell to use specified plug-ins.

  Note The plug-in is currently an experimental feature of the MongoDB Shell.

PyMongoArrow and data science

The new PyMongoArrow API allows you to implement complex analysis and machine learning tasks on MongoDB by using Python. PyMongoArrow can quickly convert simple MongoDB query results to prevalent data formats such as Pandas DataFrames and NumPy arrays to simplify your data science workflows.

Schema validation improvements

Schema validation is a method used by MongoDB for data application management. In MongoDB 5.0, schema validation has become more simple and user-friendly. When an operation validation fails, a descriptive error message is generated to highlight documents that do not conform to the collection validation rules and for what reason. This way, you can quickly identify and correct the error code that affects the validation rules.

Resumable index creation tasks

If an ongoing index creation task encounters a node restart, MongoDB 5.0 allows the task to automatically resume from where it left off. This reduces the impact of planned maintenance operations on business. For example, when database nodes are restarted or upgraded, you do not need to worry that the ongoing index creation tasks for large collections may fail.

Version release adjustment

MongoDB supports various versions and platforms, and each version needs to be verified on more than 20 MongoDB-supported platforms. The heavy verification workload makes it slower to deliver new MongoDB features. To increase the delivery speed, starting with the 5.0 release, MongoDB is released as two different series: Major Releases and Rapid Releases. Rapid Releases are available for evaluation and development purposes. We recommend that you do not use Rapid Releases in a production environment.

More features

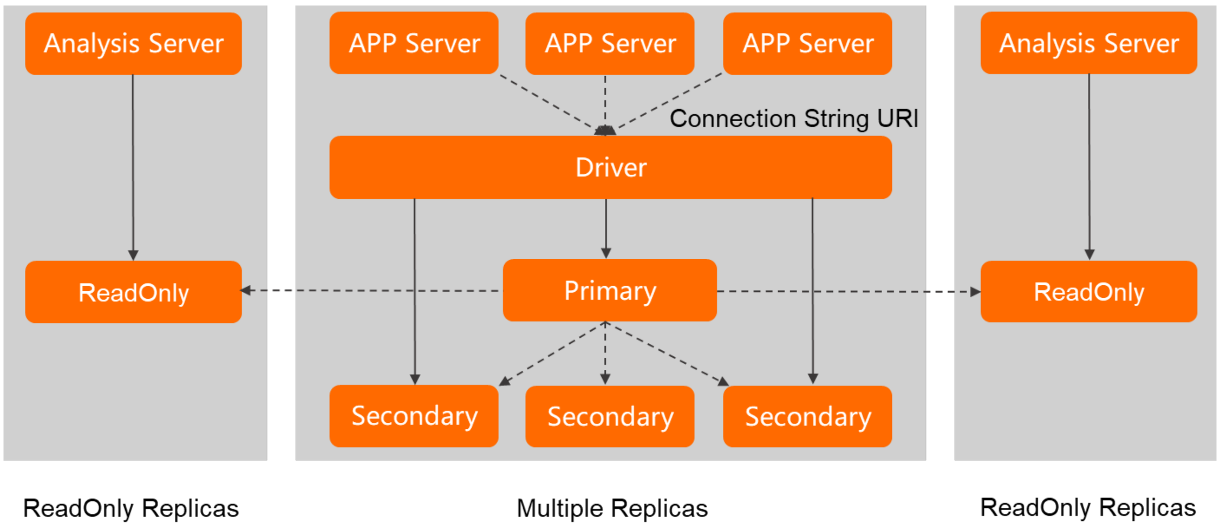
For information about other features, see Release Notes for MongoDB 5.0.

8.ApsaraDB for MongoDB read-only nodes

ApsaraDB for MongoDB provides read-only nodes that have an independent connection string URI for an instance to offload read workloads from the primary node. Read-only nodes are applicable to direct connection scenarios where primary nodes are challenged by large amounts of read requests from applications.

In scenarios where large amounts of read requests are received and no write requests are available, primary and secondary nodes may not be sufficient to handle the read pressure and your business may be affected. In this case, you can create one or more read-only nodes based on your needs to cope with massive database read requests and increase application throughput.

Structure of read-only nodes



Difference between read-only nodes and secondary nodes

Node	Description	Scenario
Secondary node	Secondary nodes can ensure high availability. When a secondary node of an instance fails, the system switches traffic to the hidden node of the instance to ensure that services are not interrupted. When the primary node of an instance fails, every secondary node of the instance has the chance to be elected as the new primary node to process data write requests.	Primary and secondary nodes can be connected by using connection string URLs. This enables read/write splitting for instances and is applicable to scenarios that require significantly more reads than writes. This way, performance is improved while the impacts of node failures on business are mitigated.

Node	Description	Scenario
Read-only node	Read-only nodes synchronize data only from primary nodes. Read-only nodes cannot ensure high availability or be elected as primary nodes. Read-only nodes in an instance have an independent connection string URL and can be connected independently of the primary and secondary nodes. Read-only nodes are suitable for direct connection from applications.	Read-only nodes are applicable to scenarios where large amounts of data need to be retrieved from existing instances. Similar scenarios include analytics platforms such as business intelligence (BI) platforms.

Limits

- Read-only nodes are supported only for subscription replica set instances, pay-as-you-go replica set instances, and subscription sharded cluster instances.
- Read-only nodes process only read requests and cannot be elected as primary or secondary nodes.
- A maximum of five read-only nodes can be added to each replica set instance.
- A maximum of five read-only nodes can be added to each shard node of a sharded cluster instance.
- Data is asynchronously replicated with millisecond-level latency between a primary node and a read-only node.
- Only ApsaraDB for MongoDB instances that run MongoDB 3.4, 4.0, or 4.2 support read-only nodes.

Note ApsaraDB for MongoDB instances must be updated to the latest minor version. For more information about how to update the minor version of an instance, see Upgrade the minor version of an ApsaraDB for MongoDB instance.

Benefits

- The number of read-only nodes can be changed as your business needs change to reduce costs.
- Read-only nodes in an instance have an independent connection string URL and can be connected independently of the primary and secondary nodes. Read-only nodes are suitable for direct connection from applications.
- Read-only nodes have the same specifications as primary and secondary nodes, and data in primary nodes can be automatically synchronized to read-only nodes without the need for maintenance.
- Read-only nodes are independent nodes that provide only read services. These nodes do not compete for resources with primary nodes, and services provided by primary nodes are not affected by the addition or removal of read-only nodes.
- The same read-only connection string URL is provided for all read-only nodes in an instance. This allows you to increase the processing power of databases by adding read-only nodes without the need to change applications.

Billing

The price of a read-only node is equivalent to that of a node in a replica set instance or that of a shard node in a sharded cluster instance.

For example, under the same specifications, the price of a three-node replica set instance is USD 3,000 and the price of a read-only node is USD 1,000.

9.Comparison between ApsaraDB for MongoDB and self-managed databases

Compared with self-managed databases, ApsaraDB for MongoDB has outstanding advantages in terms of service availability, data reliability, security, and O&M cost. ApsaraDB for MongoDB helps you quickly start your business and reduce O&M costs.

Item	ApsaraDB for MongoDB	Self-managed database
Service availability	High availability.	You must make sure security and build primary/secondary replication and RAID.
	High availability disaster recovery in one or three zones in the same city.	Zone-disaster recovery requires self-deployment and maintenance. The dual-zone conditions are difficult to implement, and database-level availability is not guaranteed.
	Allows you to create geo-disaster recovery instances.	You must use a third-party tool to create disaster recovery instances.
Data durability	• High reliability. • The recovery point objective (RPO) of single- and three-zone replica set instances is 0.	You must make sure security and build primary/secondary replication and RAID.
System security	Pre-protection: DDoS attack protection, automatic repair of various database security vulnerabilities, whitelist-based access control, and VPC isolation.	Pre-protection: requires additional security hardware or software to fix security vulnerabilities at high costs.
	In-process protection: SSL encryption and Transparent data encryption.	In-process protection: You must build SSL encryption and TDE systems.
	Event auditing: Database log auditing.	Event auditing: You must purchase an additional auditing system.
Backup and restoration	• Complete kernel enables both physical backup and logical backup to be performed in manual backup and increases backup efficiency three times. • Single-database recovery.	• The open-source version supports only logical backup and is inefficient. • Single-database recovery is unavailable and recovery efficiency is low. • You must ensure the accuracy of data recovery in the distributed architecture.
System hosting	No hosting fee.	You must pay additional server hosting fees. The more complex the architecture, the more servers you need to host and the higher the hosting fees.

Item	ApsaraDB for MongoDB	Self-managed database
O&M cost	No additional O&M costs.	Requires dedicated maintenance personnel and high O&M costs.
	CloudDBA.	Conducts performance monitoring, which is complex to troubleshoot slow queries.
Deployment and scaling	Supports real-time activation and elastic scaling.	Requires hardware procurement, hosting of data centers, and machine deployment, which takes rather a long period. You must maintain the node relationship when you add new nodes.
Kernel optimization	• Optimizes oplog synchronization performance and short-lived connection performance for general scenarios. • Ensures data consistency with technologies such as logical snapshots. This reduces costs by 50% in long-time data import scenarios.	The open source version does not provide optimization and cannot be used in some scenarios.

10.MongoDB versions and storage engines

This topic describes the MongoDB versions and storage engines that are supported by ApsaraDB for MongoDB and the relationship between MongoDB versions and storage engines.

Supported MongoDB versions

ApsaraDB for MongoDB supports the following MongoDB versions:

Note You can manually upgrade the MongoDB version while an instance is running. However, you cannot downgrade the MongoDB version. For more information, see Upgrade MongoDB versions.

- MongoDB 5.0
- MongoDB 4.4
- MongoDB 4.2
- MongoDB 4.0
- MongoDB 3.4
- MongoDB 3.2

  This version is phrased out. For more information, see Notice: ApsaraDB for MongoDB has phased MongoDB 3.2 out and released MongoDB 4.2 since February 4.

Storage engines

Storage engine	Use scenario	Description
WiredTiger	It is the default storage engine and is applicable to most business scenarios.	Data is displayed in the B-tree structure. Compared with the earlier MMAPv1 storage engine, WiredTiger significantly improves performance and reduces storage costs by compressing data.

Relationship between MongoDB versions and storage engines

Storage engine	MongoDB 5.0	MongoDB 4.4	MongoDB 4.2	MongoDB 4.0	MongoDB 3.4
WiredTiger	Replica set instances	Replica set instances	• Replica set instances • Sharded cluster instances	• Standalone instances • Replica set instances • Sharded cluster instances	• Standalone instances • Replica set instances • Sharded cluster instances

MongoDB 5.0

MongoDB 5.0 marks a new release cycle to deliver new features to users faster than in the past.

- Native time series platform

  MongoDB 5.0 natively supports the entire lifecycle of time series data, from ingestion, storage, query, real-time analysis, and visualization through to online archival or automatic expiration as data ages. This makes it faster to build and run time series applications at a lower cost. With the release of MongoDB 5.0, MongoDB has expanded the universal application data platform, which makes it easier for developers to process time series data. This further expands the use scenarios of MongoDB in areas such as IoT, financial analysis, and logistics.

- Live resharding

  Live resharding allows you to change the shard key for your collection on demand as your workload grows and evolves. No database downtime or complex migration within the dataset is required in this process. You can run the reshardCollection command in the Mongo Shell to select the database and collection that you want to reshard and specify the new shard key.

  ```
  reshardCollection: "<database>.<collection>", key: <shardkey>
  ```

  Note

  - <database>: the name of the database that you want to reshard.
  - <collection>: the name of the collection that you want to reshard.
  - <shardkey>: the name of the shard key.
  - When you run the **reshardCollection** command, MongoDB clones an existing collection and then applies all operation logs in the existing collection to a new collection. When all operation logs are applied, MongoDB switches business to the new collection and deletes the existing collection.

- Versioned API

  The Versioned API allows new features and improvements to be flexibly added to the database of each version with full backward compatibility. When you want to change an API, you can add a new version of the API and run it on the same server at the same time as the existing version of the API. With the accelerated release of MongoDB versions, the Versioned API enables quicker and easier access to the features of the latest MongoDB version.

  The Versioned API defines a set of commands and parameters that are most commonly used in applications. These commands do not change regardless of whether a database release is an annual major release or a quarterly rapid release. You can pin the driver to a specific version of the MongoDB API by decoupling the application lifecycle from the database lifecycle. This way, even if the database is upgraded and improved, your application can continue to run for several years without modifications to code.

- Default majority write concern

  Starting with MongoDB 5.0, the default write concern is majority. A write operation is committed and write success is passed back to the application only when the write operation is applied to the primary node and persisted to the logs of a majority of secondary nodes. This ensures that MongoDB 5.0 can provide stronger data durability guarantees out of the box.

- Long-running snapshot queries

Long-running snapshot queries improve the versatility and flexibility of applications. By default, snapshot queries executed by this feature have an execution duration of 5 minutes. You can also customize the duration of a snapshot query. In addition, this feature maintains strong consistency with snapshot isolation guarantees without impacting the performance of your live and transactional workloads, and enables snapshot queries to be executed on secondary nodes. This allows you to run different workloads in a single cluster and scale your workloads to different shards.

- New MongoDB Shell

  For enhanced user experience, the new MongoDB Shell has been redesigned from the ground up to provide a modern command-line experience, enhanced usability features, and a powerful scripting environment. The new MongoDB Shell has become the default shell for MongoDB. The new MongoDB Shell introduces syntax highlighting, intelligent auto-complete, contextual help, and useful error messages to create an intuitive and interactive experience.

- Version release adjustment

  Starting with the 5.0 release, MongoDB is released as two different release series: Rapid Releases and Major Releases. Rapid Releases are available for evaluation and development purposes. We recommend that you do not use Rapid Releases in a production environment.

MongoDB 4.4

MongoDB 4.4 addresses aspects that were frequently mentioned by users of earlier versions.

- Hidden Indexes

  Existing indexes are hidden to ensure that these indexes are not used in subsequent queries. You can check whether the deletion of a specified inefficient index compromises performance. If not, you can delete the inefficient index.

- Refinable Shard Keys

  One or more suffix fields are added to improve the distribution of existing documents on chunks. This prevents concurrent access to a single shard and alleviates pressure on servers.

- Compound Hashed Shard Keys

  A single hash field can be specified in a composite index to simplify the business logic.

- Hedged Reads

  For sharded clusters, a read request can be simultaneously sent to two replicas of the same shard. The results first received are used to recover the client. This reduces the request latency.

- Streaming Replication

  Operation logs of the primary database actively flow into the secondary database. Compared with the previous version where the secondary database is polled, this method saves nearly half of the round-trip time and improves the performance of data replication between the primary and secondary databases.

- Simultaneous Indexing

  The indexing processes of the primary and secondary databases are synchronized. This greatly reduces the latency generated by the primary and secondary databases in the indexing processes and ensures that the secondary database can access the most recent data in a timely manner.

- Mirrored Reads

  The primary node synchronizes a portion of read traffic to the secondary node for processing. The secondary node processes read traffic, which can reduce the access latency.

- Resumable Initial Sync

  MongoDB supports resumable upload during full synchronization between primary and secondary nodes. This prevents the need for a full synchronization from scratch in the event of a network disconnection.

- Time-Based Oplog Retention

  You can customize the retention period of operation logs. When operation logs are cleared, full synchronization from the primary node is required.

- Union

  The $unionWith stage is added to improve the query performance of MongoDB. This stage is similar to the `UNION ALL` statement in SQL.

- Custom Aggregation Expressions

  The $accumulator and $function operators are added to implement custom aggregation expressions and improve interface consistency and user experience.

Note For more information about the new features of MongoDB 4.4, see MongoDB 4.4功能概览.

MongoDB 4.2

MongoDB 4.2 uses a two-phase commit protocol to ensure the atomicity, consistency, isolation, durability (ACID) properties of transactions in sharded clusters. This greatly expands its business scenarios.

- Distributed transactions

  The two-phase commit method ensures the ACID feature of sharded cluster transactions, expands business scenarios, and achieves a leap from NoSQL to NewSQL.

- Repeatable reads

  The repeatable read feature provides the automatic retry capability in a poor-quality network environment. This reduces logic complexity on the service side and ensures continuity of your business.

- Wildcard indexes

  You can create a wildcard index for nondeterministic fields to overwrite multiple feature fields in a document for flexible management and usage.

- Field-level encryption

  Field-level encryption is supported at the driver layer and can be used to separately encrypt specified sensitive information such as accounts, passwords, prices, and mobile phone numbers. You can abandon full-database encryption and use field-level encryption to make business more flexible and secure.

- Materialized views

  Latest materialized views can cache computing results to make computing more efficient and logic less complex.

MongoDB 4.0

MongoDB 4.0 is better suited for finance and other scenarios that are dependent on transactions and use NoSQL features.

- Cross-document transactions

  MongoDB 4.0 is the first NoSQL database that supports cross-document transactions. It combines the speed, flexibility, features, and ACID guarantee of document models.

- 40% increase in migration speed

  Concurrent read and write operations enable new shard nodes to migrate data fast and bear service loads.

- Significantly improved read performance

  The transaction feature ensures that secondary nodes no longer block read requests due to log synchronization. The multi-node scaling feature is supported in all versions to significantly improve read capabilities.

MongoDB 3.4

MongoDB 3.4 offers higher performance and security than MongoDB 3.2.

Note MongoDB 3.2 is phrased out. For more information, see Notice: ApsaraDB for MongoDB has phased MongoDB 3.2 out and released MongoDB 4.2 since February 4.

- Faster synchronization between the primary and secondary nodes

  All indexes are created when data is synchronized. Only the _id index is created in earlier versions. During data synchronization, the secondary node continuously reads new operation log information to ensure that the local database of the secondary node has enough space to store temporary data.

- More efficient load balancing

  In earlier versions, mongos nodes are responsible for load balancing of sharded clusters. Multiple mongos nodes compete for a distributed lock. The node that obtains the lock performs load balancing tasks and migrates chunks between shard nodes. In MongoDB 3.4, the primary Configserver node is responsible for load balancing. This greatly improves the concurrency and efficiency of load balancing.

- More aggregation operations

  Many aggregation operators are added to MongoDB 3.4 to provide more powerful data analysis capabilities. For example, `bucket` can conveniently classify data. `$grahpLookup` supports more complex relational operations than `$lookup` in MongoDB 3.2. `$addFields` enables richer document operations such as summing some fields and saving them as a new field.

- Sharding zones

  The zone concept is introduced for sharded clusters to replace the current tag-aware sharding mechanism. The zone feature can allocate data to one or more specified shard nodes. This feature allows you to conveniently deploy sharded clusters across data centers.

- Collation

  In earlier versions, strings stored in documents are always compared byte-by-byte regardless of Chinese, English, uppercase, or lowercase. After collation is introduced, the string content can be interpreted or compared based on the used locale. Case-insensitive comparison is also supported.

- Read-only views

MongoDB 3.4 supports read-only views. MongoDB 3.4 virtualizes the data that meets a query condition into a special collection, on which you can perform further queries.

11.Instance specifications

11.1. Instance categories

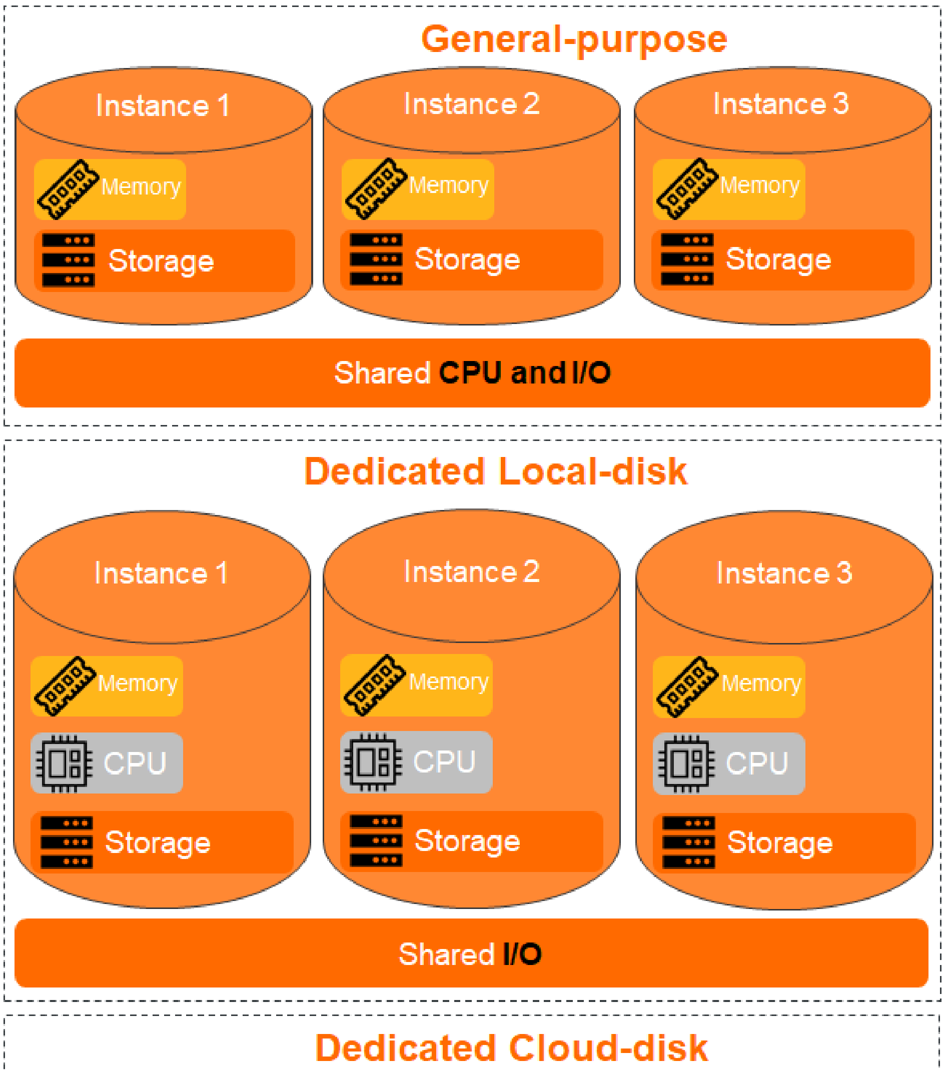
This topic describes general-purpose, dedicated-disk, and dedicated-host instance categories of ApsaraDB for MongoDB.

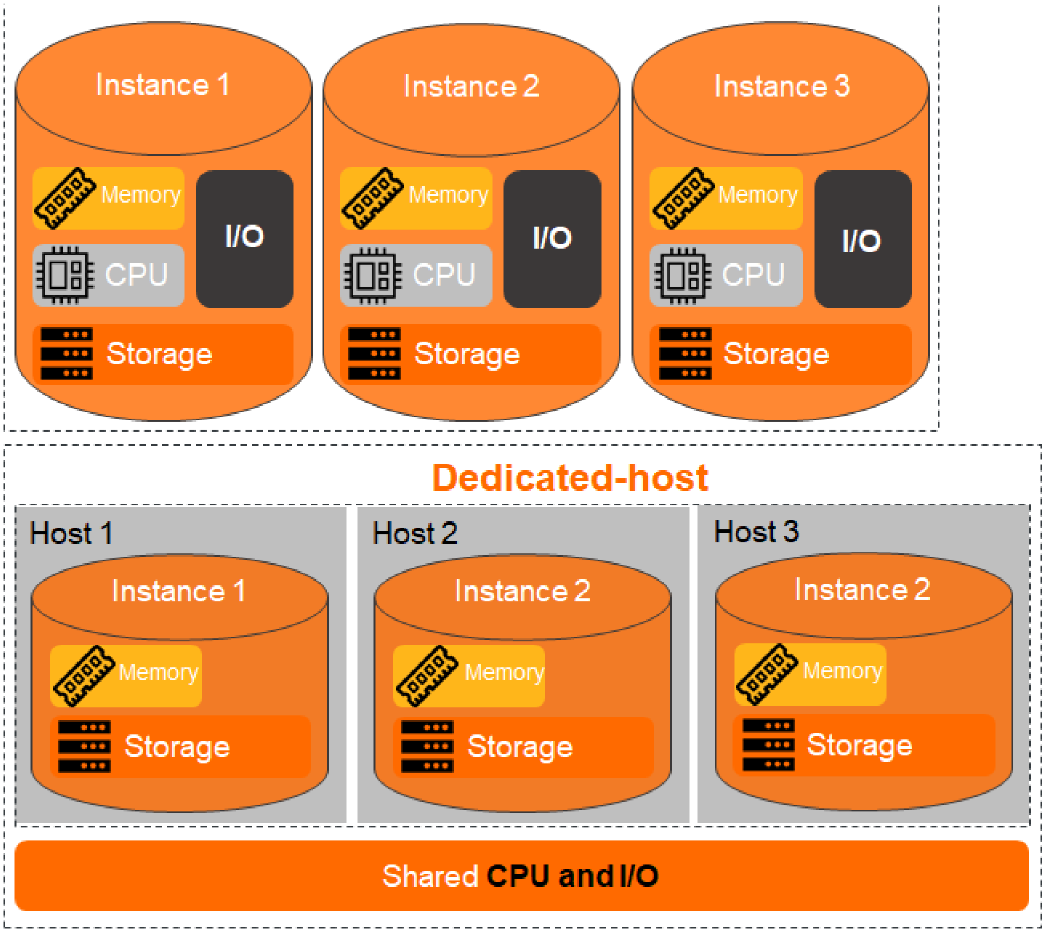
Categories

Category	Description	Scenario
General-purpose	• General-purpose instances on the same physical server occupy dedicated memory and storage but share CPU and I/O resources. • CPU resources are highly reused among shared instances that are deployed on the same physical server. This maximizes cost-effectiveness.	• Price-sensitive scenarios. • Scenarios that require a baseline level of performance stability.
Dedicated-disk	• Dedicated local-disk instances on the same physical server occupy dedicated memory, storage, and CPU resources but share I/O resources. • Dedicated cloud-disk instances occupy dedicated memory, storage, CPU, and I/O resources. Therefore, dedicated cloud-disk instances provide higher performance and stability and is less affected by other instances that are deployed on the same physical server than dedicated local-disk instances. The top configuration of this category is a dedicated-host instance, which exclusively occupies all resources on a physical server.	Business scenarios where databases are used as core systems, such as finance, e-commerce, government affairs, and large and medium-sized Internet services.

Category	Description	Scenario
Dedicated-host	• Dedicated-host instances occupy exclusive resources on the virtual or physical hosts where they are deployed. • You can log on to dedicated hosts and perform operations and maintenance (O&M). • You can create multiple instances on a host. Dedicated-host instances enable flexible resource scheduling and meet your requirements for regulatory compliance, high performance, and security. For more information, see What is ApsaraDB for MyBase?	Scenarios that require flexible resource scheduling and full permissions on hosts.

The following figure shows the differences between general-purpose instances, dedicated local-disk instances, dedicated cloud-disk instances, and dedicated-host instances.





Instance specifications

For more information about instance specifications such as the number of vCPUs, memory, storage capacity, maximum number of connections, and IOPS, see Instance types.

Pricing

For more information about the pricing of each instance type, see Billable items and pricing.

Change instance specifications

You can change the specifications of an instance. For more information about specific operations, see Overview.