ALIBABA CLOUD

Alibaba Cloud

ApsaraDB for RDS

RDS PostgreSQL Database

Document Version: 20220713

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.Overview of ApsaraDB RDS for PostgreSQL 13
2.Quotas and limits 15
3.Features 20
3.1. PostgreSQL 14 20
3.2. PostgreSQL 13 22
3.3. PostgreSQL 12 25
3.4. PostgreSQL 11 28
3.5. PostgreSQL 10 31
3.6. PostgreSQL 9.4 35
4.Specifications 38
4.1. Primary ApsaraDB RDS for PostgreSQL instance types 38
4.2. Read-only ApsaraDB RDS for PostgreSQL instance types 44
5.Billing 47
5.1. Switch an ApsaraDB RDS for PostgreSQL instance from pa... 47
5.2. Switch an ApsaraDB RDS for MySQL instance from subscri... 48
5.3. Manually renew an ApsaraDB RDS for PostgreSQL instance 49
5.4. Enable auto-renewal for an ApsaraDB RDS for PostgreSQL... 51
6.Plug-ins 55
6.1. Supported plug-ins 55
6.2. Query vertical industry-specific data 66
6.2.1. Use the TimescaleDB plug-in 66
6.2.2. Use the smlar plug-in 68
6.2.3. Use the PASE plug-in for efficient vector search 70
6.2.4. Use the roaringbitmap plug-in 78
6.2.5. Use the varbitx plug-in 83
6.2.6. Use the RDKit plug-in 86

6.3. Run cross-database queries 88
6.3.1. Use the oss_fdw plug-in to read and write foreign dat... 88
6.3.2. Use mysql_fdw to read data from and write data to a... 94
6.3.3. Use the log_fdw plug-in 97
6.3.4. Use the tds_fdw plug-in to query data of SQL Server i... 100
6.3.5. Use the oracle_fdw plug-in 102
6.4. Use the dblink and postgres_fdw plug-ins for cross-databa.. 105
6.5. Use the hll plug-in 107
6.6. Use the pg_cron plug-in to configure scheduled tasks 110
6.7. Use the PL/Proxy plug-in for horizontal sharding 112
6.8. Fuzzy query (PG_ bigm) 118
6.9. Use the wal2json plug-in 124
6.10. Use the ZomboDB plug-in to integrate with Elasticsearch 125
6.11. Use the sql_firewall plug-in 126
6.12. Use the pg_concurrency_control plug-in 129
6.13. Use the RUM plug-in 132
6.14. Use the zhparser plug-in 137
6.15. Use the pg_jieba plug-in to run full-text searches in Chin... 138
6.16. Use the fuzzystrmatch plug-in to compute similarity betw.. 139
6.17. Use the pg_hint_plan plug-in to customize query plans 142
6.18. Use pg_repack to clear tablespaces 147
6.19. Use the pglogical plug-in for logical streaming replication 149
6.20. Use the pgAudit plug-in to generate audit logs 151
6.21. Use the pldebugger plug-in to debug stored procedures 152
6.22. Using the index_adviser plug-in on an ApsaraDB RDS for... 155
6.23. Use HypoPG to create hypothetical indexes 160
6.24. Use the sequential-uuids plug-in to generate sequential ... 162
6.25. Use the MADlib plug-in 163

7.Quick start 166
7.1. General workflow to use ApsaraDB RDS for PostgreSQL 166
7.2. Create an ApsaraDB RDS for PostgreSQL instance 166
7.3. Set the whitelist 174
7.3.1. Configure an IP address whitelist for an ApsaraDB RDS... 174
7.3.2. Configure a security group for an ApsaraDB RDS for P... 179
7.3.3. Errors and FAQ about IP address whitelist settings in A.. 181
7.4. Create a database and an account on an ApsaraDB RDS f... 183
7.5. Connect to an ApsaraDB RDS for PostgreSQL instance 185
8.Development and O&M recommendations for ApsaraDB RDS fo... 191
9.Migration to Cloud 197
9.1. Use scenarios 197
9.2. Preparations for cloud migration 199
9.2.1. (Optional) Configure an ECS security group on a self-m... 199
9.2.2. Configure a self-managed PostgreSQL instance to liste... 203
9.2.3. Create an account for cloud migration on a self-mana... 204
9.2.4. Update the pg_hba.conf file of a self-managed Postgre.. 205
9.2.5. Configure the firewall of the server on which a self-ma.. 207
9.3. Use the cloud migration feature for an ApsaraDB RDS for... 209
9.4. Introduction to cloud migration assessment reports 215
9.5. Use scenarios on Cloud 223
9.5.1. Migrate data between ApsaraDB RDS for PostgreSQL in... 223
9.5.2. Migrate data between ApsaraDB RDS for PostgreSQL i... 232
9.5.3. Scale down an ApsaraDB RDS for PostgreSQL instance 241
10.Data Migration 246
10.1. Overview of data migration methods 246
10.2. Migrate user-created databases to ApsaraDB 246
10.2.1. Manually migrate data from a user-created PostgreSQ... 246

10.2.2. Migrate data from a self-managed PostgreSQL databa.. 250
10.3. Migrate third-party databases to ApsaraDB 258
10.3.1. Migrate incremental data from an Amazon RDS for Po.. 259
10.3.2. Migrate full data from an Amazon RDS for PostgreSQ... 268
10.4. Migrate data between ApsaraDB RDS for PostgreSQL inst... 276
10.4.1. Use DTS to migrate data between ApsaraDB RDS for ... 276
10.4.2. Use the cloud migration feature to migrate data betw.. 276
11.Manage DataConnectors 281
11.1. Configure one-way data synchronization between ApsaraD... 281
11.2. Configure two-way data synchronization between Apsara... 284
11.3. Synchronize data from a self-managed PostgreSQL datab... 288
11.4. Synchronize data from an ApsaraDB RDS for PostgreSQL .. 295
11.5. Synchronize data from an ApsaraDB RDS for PostgreSQL .. 299
12.Change tracking 306
12.1. Use DTS to track data changes from ApsaraDB RDS for P... 306
13.Version upgrade 307
13.1. Upgrade an ApsaraDB RDS for PostgreSQL instance from ... 307
13.2. Upgrade the major engine version of an ApsaraDB RDS f... 309
13.3. Introduction to the check report of a major engine versio.. 316
13.4. Update the minor engine version of an ApsaraDB RDS fo... 320
14.Instance 323
14.1. Create an ApsaraDB RDS for PostgreSQL instance 323
14.2. Restart an ApsaraDB RDS for PostgreSQL instance 330
14.3. Switch workloads over between primary and secondary A... 331
14.4. Set the maintenance window of an ApsaraDB RDS for Po.. 334
14.5. Configure automatic storage expansion for an ApsaraDB ... 335
14.6. Migrate an ApsaraDB RDS for PostgreSQL instance across... 337
14.7. Release or unsubscribe from an ApsaraDB RDS for Postgr... 342

14.8. Enable or disable the release protection feature for an A... 343
14.9. Manage the parameters of an ApsaraDB RDS for Postgre... 345
14.10. Set the protection level of an ApsaraDB RDS for Postgre.. 346
14.11. Manage ApsaraDB RDS for PostgreSQL instances in the ... 349
15.Babelfish for ApsaraDB RDS for PostgreSQL 351
15.1. Introduction to Babelfish 351
15.2. Enable Babelfish for an ApsaraDB RDS for PostgreSQL in... 355
15.3. Manage Babelfish accounts 358
15.4. Connect to an ApsaraDB RDS for PostgreSQL instance wi... 360
15.4.1. Use clients to connect to an ApsaraDB RDS for Postgr... 360
15.4.2. Use applications to connect to an ApsaraDB RDS for ... 365
15.5. View the version of Babelfish 376
15.6. Migrate the data of a SQL Server database to a databas... 378
15.7. Common operations and compatibility description 382
16.RDS for PostgreSQL read-only instances 389
16.1. Overview of read-only ApsaraDB RDS for PostgreSQL inst... 389
16.2. Create a read-only ApsaraDB RDS for PostgreSQL instanc... 391
17.Database proxy (read/write splitting) 396
17.1. What are database proxies? 396
17.2. Billing rules for the database proxy of an ApsaraDB RDS... 399
17.3. Use the database proxy feature 400
17.3.1. Enable and configure the database proxy feature for ... 400
17.3.2. Create a database proxy terminal for an ApsaraDB RD.. 408
17.3.3. Manage the database proxy endpoints of an ApsaraD... 412
17.3.4. View the proxy monitoring data of an ApsaraDB RDS .. 415
17.3.5. Change the number of proxy instances on an ApsaraD.. 416
17.3.6. Upgrade the database proxy version of an ApsaraDB ... 417
17.3.7. Disable the database proxy feature 418

17.4. Default read weights 418
18.Account 423
18.1. Create an account on an ApsaraDB RDS for PostgreSQL i... 423
18.2. Reset the password of an account on an ApsaraDB RDS ... 425
18.3. Authorize the service account of an RDS PostgreSQL inst... 426
18.4. Lock or delete an account from an ApsaraDB RDS for Po... 427
18.5. Account permissions 429
18.6. Connect an ApsaraDB RDS for PostgreSQL instance to a ... 430
18.7. System accounts of an ApsaraDB RDS for PostgreSQL inst... 443
19.Database connections 445
19.1. Connect to an ApsaraDB RDS for PostgreSQL instance 445
19.2. Apply for or release a public endpoint on an ApsaraDB ... 450
19.3. Use DMS to log on to an ApsaraDB RDS for PostgreSQL ... 452
19.4. View and change the internal and public endpoints and ... 454
20.Network, VPC, and VSwitch 455
20.1. Switch an ApsaraDB RDS for PostgreSQL instance to a di... 455
20.2. Change the network type of an ApsaraDB RDS for Postg... 456
20.3. Configure the hybrid access solution for an ApsaraDB RD... 459
21.Database 462
21.1. Create a database on an ApsaraDB RDS for PostgreSQL i... 462
21.2. Delete a database from an ApsaraDB RDS for PostgreSQL... 463
21.3. Change the time zone of an ApsaraDB RDS for PostgreS... 463
22.SQL translation 465
23.Monitoring and alerts 466
23.1. View the resource and engine metrics of an ApsaraDB RD... 466
23.2. Set the monitoring frequency of an ApsaraDB RDS for P... 467
23.3. View the Enhanced Monitoring metrics of an ApsaraDB R... 468
23.4. Manage the alert rules of an ApsaraDB RDS for PostgreS... 491

24.Data security 495
24.1. Switch an ApsaraDB RDS for PostgreSQL instance to the ... 495
24.2. Configure an IP address whitelist and security group for .. 496
24.2.1. Configure an IP address whitelist for an ApsaraDB RD... 496
24.2.2. Configure a security group for an ApsaraDB RDS for ... 501
24.2.3. Errors and FAQ about IP address whitelist settings in... 503
24.3. SSL encryption 504
24.3.1. Configure SSL encryption for an ApsaraDB RDS for Po... 505
24.3.2. Configure a custom certificate on an ApsaraDB RDS f... 510
24.3.3. Configure a client CA certificate on an ApsaraDB RDS... 516
24.3.4. Connect to an ApsaraDB RDS for PostgreSQL instance... 521
24.4. Configure disk encryption for an ApsaraDB RDS for Postg.. 529
24.5. Fully encrypted database 531
24.5.1. Overview 531
24.5.2. Configure the fully encrypted database feature on an... 532
25.Log and event history management 533
25.1. Use the SQL Audit feature on an ApsaraDB RDS for Post... 533
25.2. View logs 535
25.3. View the event history of an ApsaraDB RDS instance 536
26.Manage pending events 541
27.Backup 544
27.1. Backup fees for an ApsaraDB RDS for PostgreSQL instance 544
27.2. Back up an ApsaraDB RDS for PostgreSQL instance 544
27.3. Use the cross-region backup feature for an ApsaraDB RD... 549
27.4. View the free quota for backup storage of an ApsaraDB ... 560
27.5. Download the data backup files and log backup files of ... 562
27.6. Create a logical backup for an ApsaraDB RDS for Postgre... 563
27.7. Create a full backup of an ApsaraDB RDS for PostgreSQL... 568

28.Restoration ---- 570
28.1. Restore the data of an ApsaraDB RDS for PostgreSQL ins... ---- 570
28.2. Restore the data of an ApsaraDB RDS for PostgreSQL ins... ---- 573
28.3. Restore data from a logical backup file ---- 575
29.Performance Optimization and diagnosis ---- 579
29.1. Overview of DAS ---- 579
29.2. Diagnostics ---- 580
29.2.1. Use the session management feature for an ApsaraDB... ---- 580
29.2.2. Use the real-time monitoring feature for an ApsaraDB... ---- 580
29.2.3. Storage analysis ---- 580
29.2.4. Use the performance insight feature for an ApsaraDB... ---- 581
29.3. Use the performance trends feature for an ApsaraDB RD... ---- 581
29.4. Use the slow query log analysis feature for an ApsaraDB... ---- 582
29.5. Use the query governance feature for an ApsaraDB RDS ... ---- 582
29.6. Use the SQL Explorer and Audit feature on an ApsaraDB... ---- 584
29.7. Use the monitoring dashboard feature ---- 585
30.Tag ---- 590
30.1. Add tags to ApsaraDB RDS instances ---- 590
30.2. Remove tags from an ApsaraDB RDS for MySQL instance ---- 592
30.3. Use tags to filter ApsaraDB RDS for MySQL instances ---- 593
31.Best Practices ---- 595
31.1. Manage permissions in an ApsaraDB RDS for PostgeSQL i... ---- 595
31.2. Migrate data from a user-created PostgreSQL database to... ---- 600
31.3. Configure the collation of a database on an ApsaraDB R... ---- 601
31.4. Insert, update, and delete multiple data records at a tim... ---- 608
31.5. Locate SQL statements with the highest resource consum... ---- 610
31.6. Logical subscription ---- 614
31.7. Use event triggers to implement the DDL recycle bin, fire... ---- 616

31.8. Configure automatic failover and read/write splitting ---- 620
32.Application Solutions ---- 624
32.1. Real-time precision marketing (user selection) ---- 624
32.2. Image recognition, face recognition, similarity-based retri... ---- 631
32.3. Second-level flashback for real-time disaster recovery ---- 643
32.4. Use pgpool for read/write splitting in ApsaraDB RDS for... ---- 651
32.5. User preference recommendation system ---- 680
32.6. Use an ApsaraDB RDS for PostgreSQL instance as the re... ---- 689
32.7. Use ShardingSphere to develop ApsaraDB RDS for Postgr... ---- 699

1.Overview of ApsaraDB RDS for PostgreSQL

This topic provides an overview of ApsaraDB RDS for PostgreSQL and describes the related terms.

ApsaraDB for RDS is a stable, reliable, and scalable online database service. It is designed based on the Apsara Distributed File System and high-performance SSD storage media of Alibaba Cloud. It supports five database engines: MySQL, SQL Server, PostgreSQL, and MariaDB. It also provides a complete suite of solutions for various scenarios, such as disaster recovery, backup, restoration, monitoring, and migration. These solutions facilitate database operation and maintenance (O&M). For more information about the benefits of ApsaraDB for RDS, see Competitive advantages of ApsaraDB RDS instances over self-managed databases.

You can submit a if you require technical support. If your workloads are complex, you can purchase a support plan on the Alibaba Cloud After-Sales Support page. This allows you to seek advice from instant messaging (IM) enterprise groups, technical account managers (TAMs), and service managers.

For more information about ApsaraDB for RDS, visit the ApsaraDB RDS for MySQL product page.

Disclaimer

Some features or functions that are described in this document may be unavailable. For more information, see the specific terms and conditions in your commercial contract. This document serves as a user guide that is for reference only. No content in this document can constitute any expressed or implied warranty.

PostgreSQL

PostgreSQL is an advanced open source database that is compatible with SQL and supports various data types, such as JSON data, IP data, and geometry data. PostgreSQL supports basic features, such as transactions, subqueries, multi-version concurrency control (MVCC), and data integrity check. In addition to the basic features, PostgreSQL supports the high availability architecture and provides the backup and restoration feature to facilitate database operation and maintenance (O&M). PostgreSQL also provides the following advanced features and functions:

- ApsaraDB for MyBase dedicated clusters: An ApsaraDB for MyBase dedicated cluster consists of multiple hosts, such as ECS instances of the ecs.i2.xlarge instance type and ECS Bare Metal instances. You can run instances on these hosts. For more information, see What is ApsaraDB for MyBase?
- Fully encrypted databases: You can create a fully encrypted database on your RDS instance to store data that is uploaded from the user end. Data is encrypted before it is uploaded to the fully encrypted database. The fully encrypted database protects data from internal and external security threats and allows only authorized users to access the data. For more information, see Fully encrypted databases.
- Read-only RDS instances: If the primary RDS instance is overwhelmed by a large number of read requests, your workloads may be interrupted. In this situation, you can create one or more read-only RDS instances to offload read requests from the primary RDS instance. For more information, see Create a read-only ApsaraDB RDS for PostgreSQL instance. By creating one or more read-only RDS instances, you can scale up the read capability of your database system and increase the throughput of your application.

For more information about the features and functions that ApsaraDB RDS for PostgreSQL supports, see Features of ApsaraDB RDS for PostgreSQL.

Basic terms

- Instance: An RDS instance is a database process that consumes independent physical memory

resources. You can specify a specific memory size, disk capacity, and database type for an RDS instance. The performance of an RDS instance varies based on the specified memory size. After an RDS instance is created, you can change its specifications or delete the instance.

- Database: A database is a logical unit that is created on an RDS instance. One RDS instance can have multiple databases. Each database must have a unique name on the RDS instance where it is created.
- Region and zone: Each region is a physical data center. Each region contains a number of isolated locations that are known as zones. Each zone has an independent power supply and network. For more information, visit the Alibaba Cloud's Global Infrastructure page.

General terms

Term	Description
On-premises database	A database that is deployed in an on-premises data center or a database that is not deployed on an ApsaraDB for RDS instance.
ApsaraDB RDS for XX (XX represents one of the following database engines: MySQL, SQL Server, PostgreSQL, and MariaDB.)	ApsaraDB for RDS with a specific database engine. For example, ApsaraDB RDS for MySQL indicates an ApsaraDB for RDS instance that runs MySQL.

2.Quotas and limits

This topic describes the quotas and limits for ApsaraDB RDS for PostgreSQL. Before you create an ApsaraDB RDS for PostgreSQL instance, make sure that you understand the quotas and limits. This way, you can ensure the stability and security of your RDS instance.

Specifications and performance

Item	Specification	Description
Storage capacity	• RDS instances that use local SSDs: up to 6,000 GB. • RDS instances that use standard SSDs: up to 6,000 GB. • RDS instances that use enhanced SSDs (ESSDs): up to 32,000 GB.	The storage capacity of an RDS instance that uses local SSDs varies based on the instance type. This limit does not apply to RDS instances that use standard SSDs or ESSDs. For more information, see Primary ApsaraDB RDS instance types.
Number of connections	Up to 76,800.	The number of connections that can be established to an RDS instance varies based on the instance type. For more information, see Primary ApsaraDB RDS instance types.
IOPS	• RDS instances that use local SSDs: up to 50,000. • RDS instances that use standard SSDs or ESSDs: For more information, see Maximum IOPS for standard SSDs and ESSDs.	None.
Memory capacity	• RDS instances that use local SSDs: up to 512 GB. • RDS instances that use standard SSDs or ESSDs: up to 768 GB.	For RDS instances that use standard SSDs or ESSDs, the memory includes the memory that is occupied by the underlying operating system and the memory that is occupied by the RDS-related management services. Therefore, the available memory of an RDS instance may be less than the memory capacity that is supported by the instance type. • The underlying operating system occupies 500 MB to 700 MB of memory. • The RDS-related management services occupy approximately 500 MB of memory.

Quotas

Item	Quota

Read-only RDS instance	Read-only RDS instances are supported only for RDS instances that run PostgreSQL 10 or later. The read-only RDS instances that you create must reside in the same region as the primary RDS instance to which the read-only RDS instances are attached.
	• If the primary RDS instance uses local SSDs:
	  ◦ You can create up to five read-only RDS instances.
	  ◦ You can create read-only RDS instances only when the primary RDS instance is a dedicated instance that provides at least 8 cores and 32 GB of memory.
	• If the primary RDS instance uses standard SSDs or ESSDs:
	  ◦ You can create up to 32 read-only RDS instances.
	  ◦ Each read-only RDS instance runs based on a single-node architecture. In this architecture, no secondary RDS instances are provided as standbys for read-only RDS instances.
	For more information about read-only RDS instances, see Overview of read-only ApsaraDB RDS for PostgreSQL instances.

Tag	The key of a tag must be unique. You can add up to 20 tags to an RDS instance. You can add tags to up 50 RDS instances at a time. For more information, see Add tags to ApsaraDB RDS instances.
Free quota for backup storage	RDS instances that use standard SSDS or ESSDs support only snapshot backups. RDS instances that use local SSDs support only physical backups. If your backup storage exceeds the free quota, you are charged for your excess backup storage. Your excess backup storage is calculated by using the following formula: Excess backup storage = Size of data backup files + Size of log backup files - Free quota. Unit: GB. The obtained result is rounded up to the next integer. • RDS instances that use local SSDs: Free quota for the storage of physical backup files = 50% × Purchased storage capacity. • RDS instances that use standard SSDs or ESSDs: Free quota for the storage of snapshot backup files = 200% × Purchased storage capacity. For more information, see Back up an ApsaraDB RDS for PostgreSQL instance.
Backup retention period	The default retention period is 7 days, and the maximum retention period is 730 days.
Retention period of error logs	The retention period of slow log details is 30 days. For more information, seeView logs.
Retention period of slow log details	The retention period of slow log details is 30 days. For more information, seeView logs.

Limits on names

Item	Description

Item	Description
Instance name	• The name of an RDS instance must be 2 to 256 characters in length. • The name of an RDS instance can contain letters, digits, underscores (_), and hyphens (-). • The name of an RDS instance must start with a letter.
Username	• The username of the account must be 2 to 63 characters in length. • The username of the account can contain lowercase letters, digits, and underscores (_). • The username of the account must start with a lowercase letter and end with a lowercase letter or a digit. • The username of the account cannot be the same as the username of an existing account. • The username of the account cannot start with pg. • The username of the account cannot contain SQL keywords. For more information, see SQL Keywords.
Database name	• The name of a database can contain up to 63 characters in length. • The name of a database can contain lowercase letters, digits, underscores (_), and hyphens (-). • The name of a database must start with a lowercase letter and end with a lowercase letter or a digit. • The name of a database must be unique. • The username of a database cannot contain SQL keywords. For more information, see SQL keywords.

Limits on security

Item	Description
Password	The password of an account must meet the following requirements: • The password must be 8 to 32 characters in length. • The password must contain at least three of the following character types: uppercase letters, lowercase letters, digits, and special characters. The password can contain any of the following special characters: `! @ # $ % ^ & * ( ) _ + - =`
Port	By default, an RDS instance is connected over port 5432. You can change the port number based on your business requirements.
Instance parameter configuration	For security and stability reasons, some parameters cannot be reconfigured. You can reconfigure most of the instance parameters in the ApsaraDB RDS console or by using the ApsaraDB RDS API. For more information, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.
Disk encryption	You can enable disk encryption for an RDS instance only at the time when you purchase the instance. Disk encryption cannot be disabled after it is enabled. For more information, see Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance.

Item	Description
Number of security groups	You can configure up to 10 security groups for an RDS instance. • After you configure security groups for an RDS instance, all Elastic Compute Service (ECS) instances in the configured security groups can communicate with the RDS instance. • The security groups that you configure for an RDS instance must have the same network type as the RDS instance. This means that the network types of the RDS instance and the security groups that you want to configure must both be Virtual Private Cloud (VPC) or classic network. For more information, see Configure a security group for an ApsaraDB RDS for PostgreSQL instance.
Number of IP address whitelists	You can configure up to 50 IP address whitelists for an RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
Root account	You cannot create root accounts. ApsaraDB RDS for PostgreSQL does not provide superuser accounts such as root accounts.
Privileged account	You can create and manage privileged accounts in the ApsaraDB RDS console or by using the ApsaraDB RDS API. You can create multiple privileged accounts for an RDS instance. The privileged accounts of an RDS instance have the permissions to disconnect all standard accounts of the RDS instance. For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance.
Standard account	You can create and manage standard accounts in the ApsaraDB RDS console. You can also use the ApsaraDB RDS API or execute SQL statements to create and manage standard accounts. You must grant the permissions on specific databases to each standard account. A standard account of an RDS instance does not have the permissions to create, manage, or disconnect the other accounts of the RDS instance. For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance.

Limits on SQL commands

The limits on SQL commands in ApsaraDB RDS for PostgreSQL are the same as the limits on SQL commands in open source PostgreSQL. For more information, see SQL Commands and PostgreSQL Limits.

Other limits

Item	Description
Public endpoint	If you want to connect to an RDS instance by using a public endpoint, you must manually apply for a public endpoint for the RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.

Item	Description
Replication	ApsaraDB RDS for PostgreSQL provides a primary/secondary replication architecture in all RDS editions except RDS Basic Edition. In this architecture, a secondary RDS instance is provided as a standby for the primary RDS instance. Secondary RDS instances are invisible to you and cannot be accessed by your applications.
Instance restart	You can restart an RDS instance only in the ApsaraDB RDS console or by using the ApsaraDB RDS API.
Tablespace creation	You cannot create tablespaces.

3.Features

3.1. PostgreSQL 14

This topic provides an overview of the features supported by ApsaraDB RDS instances that run PostgreSQL 14. You can purchase an ApsaraDB RDS for PostgreSQL instance that provides the features that you need. You can also query the features that are provided by an existing ApsaraDB RDS for PostgreSQL instance. In the following tables, ticks (√) indicate that a feature is supported, and crosses (×) indicate that a feature is not supported.

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Data migration to the cloud	Migration to the cloud	✔	✔
Data migration, synchronization, and change tracking	Data migration	✔	✔
	Data synchronization	✔	✔
	Change tracking	✔	✔
Database engine version upgrade	Upgrade of the major engine version	□	□
	Update of the minor engine version	✔	✔
Instance management	Instance creation	✔	✔
	Specification change	✔	✔
	Instance restart	✔	✔
	Automatic or manual primary/secondary switchover	✔	□
	Maintenance window setting	✔	✔
	Automatic storage expansion	✔	✔
	Cross-zone instance migration	✔	□
	Instance release	✔	✔
	Release protection	✔	✔
	Parameter configuration	✔	✔
	Protection level configuration	✔	□
	Recycle bin	✔	✔

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Babelfish for RDS PostgreSQL	Babelfish for RDS PostgreSQL	✔	✔
Read-only instance management	Read-only instance	✔	□
Database proxy (read/write splitting)	Database proxy (read/write splitting)	✔	□
Account management	Account creation	✔	✔
	Password reset	✔	✔
	Connection to a self-managed AD domain	✔	✔
	Account locking or deletion	✔	✔
Connection management	Connection to an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Public endpoint application or release	✔	✔
	Viewing and change of the internal and public endpoints and port numbers	✔	✔
Network management	Network type change	□	□
	vSwitch change	✔	✔
Database management	Database creation	✔	✔
	Database deletion	✔	✔
	Time zone change	✔	✔
	Plug-ins	✔	✔
Monitoring and alerting	Check on the resource metrics and engine metrics	✔	✔
	Monitoring frequency setting	✔	✔
	Check on the Enhanced Monitoring metrics	✔	✔
	Alert management	✔	✔

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Security management	Switching to the enhanced whitelist mode	□	□
	Whitelist configuration	✔	✔
	SSL encryption	✔	✔
	Disk encryption	✔	✔
	Creation of fully encrypted instances	✔	□
Audit, logs, and historical events	SQL Audit	✔	✔
	Log management	✔	✔
	Event history	✔	✔
Backup	Data backup	✔	✔
	Cross-region backup	✔	✔
	Backup file download	□	□
	Backup deletion	□	□
Restoration	Data restoration	✔	✔
	Data restoration from a cross-region backup file	✔	✔
Performance optimization and diagnosis	Database Autonomy Service (DAS)	✔	□
Tag management	Tag creation	✔	✔
	Tag deletion	✔	✔

3.2. PostgreSQL 13

This topic provides an overview of the features supported by ApsaraDB RDS instances that run PostgreSQL 13. You can purchase an ApsaraDB RDS for PostgreSQL instance that provides the features that you need. You can also query the features that are provided by an existing ApsaraDB RDS for PostgreSQL instance. In the following tables, ticks (√) indicate that a feature is supported, and crosses (×) indicate that a feature is not supported.

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Data migration to the cloud	Migration to the cloud	✔	✔
Data migration, synchronization, and change tracking	Data migration	✔	✔
	Data synchronization	✔	✔
	Change tracking	✔	✔
Database engine version upgrade	Upgrade of the major engine version	✔	✔
	Update of the minor engine version	✔	✔
Instance management	Instance creation	✔	✔
	Specification change	✔	✔
	Instance restart	✔	✔
	Automatic or manual primary/secondary switchover	✔	□
	Maintenance window setting	✔	✔
	Automatic storage expansion	✔	✔
	Cross-zone instance migration	✔	□
	Instance release	✔	✔
	Release protection	✔	✔
	Parameter configuration	✔	✔
	Protection level configuration	✔	□
	Recycle bin	✔	✔
Babelfish for RDS PostgreSQL	Babelfish for RDS PostgreSQL	✔	✔
Read-only instance management	Read-only instance	✔	□
Database proxy (read/write splitting)	Database proxy (read/write splitting)	✔	□
	Account creation	✔	✔
	Password reset	✔	✔

Category Account management	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
	Connection to a self-managed AD domain	✔	✔
	Account locking or deletion	✔	✔
Connection management	Connection to an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Public endpoint application or release	✔	✔
	Viewing and change of the internal and public endpoints and port numbers	✔	✔
Network management	Network type change	□	□
	vSwitch change	✔	✔
Database management	Database creation	✔	✔
	Database deletion	✔	✔
	Time zone change	✔	✔
	Plug-ins	✔	✔
Monitoring and alerting	Check on the resource metrics and engine metrics	✔	✔
	Monitoring frequency setting	✔	✔
	Check on the Enhanced Monitoring metrics	✔	✔
	Alert management	✔	✔
Security management	Switching to the enhanced whitelist mode	□	□
	Whitelist configuration	✔	✔
	SSL encryption	✔	✔
	Disk encryption	✔	✔
	Creation of fully encrypted instances	✔	□
	SQL Audit	✔	✔
	Log management	✔	✔

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Audit, logs, and historical events			
	Event history	✔	✔
Backup	Data backup	✔	✔
	Cross-region backup	✔	✔
	Backup file download	□	□
	Backup deletion	□	□
Restoration	Data restoration	✔	✔
	Data restoration from a cross-region backup file	✔	✔
Performance optimization and diagnosis	Database Autonomy Service (DAS)	✔	□
Tag management	Tag creation	✔	✔
	Tag deletion	✔	✔

3.3. PostgreSQL 12

This topic provides an overview of the features supported by ApsaraDB RDS for PostgreSQL instances that run PostgreSQL 12. You can purchase an ApsaraDB RDS for PostgreSQL instance that provides the features that you need. You can also query the features that are provided by an existing ApsaraDB RDS for PostgreSQL instance. In the following tables, ticks (√) indicate that a feature is supported, and crosses (×) indicate that a feature is not supported.

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Data migration and synchronization	Migrate and synchronize the data of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Create an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Change the specifications of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Manage the parameters of an ApsaraDB RDS for PostgreSQL instance	✔	✔

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Instance management	Set the protection level of an ApsaraDB RDS for PostgreSQL instance	✔	□
	Migrate an ApsaraDB RDS for PostgreSQL instance across zones in the same region	✔	□
	Switch workloads over between primary and secondary ApsaraDB RDS for PostgreSQL instances	✔	□
	Upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Restart an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Set the maintenance window of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Release an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Manage ApsaraDB RDS for PostgreSQL instances in the recycle bin	✔	✔
Account management	Create an account on an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Reset the password of an account of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Lock an account of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Delete an account from an ApsaraDB RDS for PostgreSQL instance	✔	✔
Database management	Create a database on an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Delete a database from an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Use plug-ins on an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Connect to an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Configure endpoints	✔	✔

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Connection management	View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance	✔	✔
Monitoring and alerting	View the resource and engine metrics of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Set the monitoring frequency of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Manage the alert rules of an ApsaraDB RDS for PostgreSQL instance	✔	✔
Network management	Change the network type of an ApsaraDB RDS for PostgreSQL instance	□	□
	Switch an ApsaraDB RDS for PostgreSQL instance to a different vSwitch	✔	✔
Read-only instances	Create a read-only ApsaraDB RDS for PostgreSQL instance	✔	□
Security management	Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Switch an ApsaraDB RDS for PostgreSQL instance to the enhanced whitelist mode	□	□
	Create a fully encrypted ApsaraDB RDS for PostgreSQL instance	□	□
Log and event history management	Manage the logs of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	View the event history of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Back up an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance	✔	✔

Category Backup	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
	Provide a free quota for backup storage for an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Download the data backup files and log backup files of an ApsaraDB RDS for PostgreSQL instance	□	□
Restoration	Restore the data of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Restore the data of an ApsaraDB RDS for PostgreSQL instance from a cross-region backup file	✔	✔
Logical subscription	Logical subscription	✔	✔
Tag management	Add tags to ApsaraDB RDS instances	✔	✔
	Remove tags from an ApsaraDB RDS for MySQL instance	✔	✔
	Use tags to filter ApsaraDB RDS for MySQL instances	✔	✔

3.4. PostgreSQL 11

This topic provides an overview of the features supported by ApsaraDB RDS for PostgreSQL instances that run PostgreSQL 11. You can purchase an ApsaraDB RDS for PostgreSQL instance that provides the features that you need. You can also query the features that are provided by an existing ApsaraDB RDS for PostgreSQL instance. In the following tables, ticks (√) indicate that a feature is supported, and crosses (×) indicate that a feature is not supported.

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Data migration and synchronization	Migrate and synchronize the data of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Create an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Change the specifications of an ApsaraDB RDS for PostgreSQL instance	✔	✔

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Instance management	Manage the parameters of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Set the protection level of an ApsaraDB RDS for PostgreSQL instance	✔	□
	Migrate an ApsaraDB RDS for PostgreSQL instance across zones in the same region	✔	□
	Switch workloads over between primary and secondary ApsaraDB RDS for PostgreSQL instances	✔	□
	Upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Restart an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Set the maintenance window of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Release an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Manage ApsaraDB RDS for PostgreSQL instances in the recycle bin	✔	✔
Account management	Create an account on an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Reset the password of an account of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Lock an account of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Delete an account from an ApsaraDB RDS for PostgreSQL instance	✔	✔
Database management	Create a database on an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Delete a database from an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Use plug-ins on an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Connect to an ApsaraDB RDS for PostgreSQL instance	✔	✔

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Connection management	Configure endpoints	✔	✔
	View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance	✔	✔
Monitoring and alerting	View the resource and engine metrics of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Set the monitoring frequency of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Manage the alert rules of an ApsaraDB RDS for PostgreSQL instance	✔	✔
Network management	Change the network type of an ApsaraDB RDS for PostgreSQL instance	□	□
	Switch an ApsaraDB RDS for PostgreSQL instance to a different vSwitch	✔	✔
Read-only instances	Create a read-only ApsaraDB RDS for PostgreSQL instance	✔	□
Security management	Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Switch an ApsaraDB RDS for PostgreSQL instance to the enhanced whitelist mode	□	□
	Create a fully encrypted ApsaraDB RDS for PostgreSQL instance	✔	✔
Log and event history management	Manage the logs of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	View the event history of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Back up an ApsaraDB RDS for PostgreSQL instance	✔	✔

Category	Feature	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Backup	Use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Provide a free quota for backup storage for an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Download the data backup files and log backup files of an ApsaraDB RDS for PostgreSQL instance	□	□
Restoration	Restore the data of an ApsaraDB RDS for PostgreSQL instance	✔	✔
	Restore the data of an ApsaraDB RDS for PostgreSQL instance from a cross-region backup file	✔	✔
Logical subscription	Logical subscription	✔	✔
Tag management	Add tags to ApsaraDB RDS instances	✔	✔
	Remove tags from an ApsaraDB RDS for MySQL instance	✔	✔
	Use tags to filter ApsaraDB RDS for MySQL instances	✔	✔

3.5. PostgreSQL 10

This topic provides an overview of the features supported by ApsaraDB RDS for PostgreSQL instances that run PostgreSQL 10. You can purchase an ApsaraDB RDS for PostgreSQL instance that provides the features that you need. You can also query the features that are provided by an existing ApsaraDB RDS for PostgreSQL instance. In the following tables, ticks (√) indicate that a feature is supported, and crosses (×) indicate that a feature is not supported.

Category	Feature	RDS High-availability Edition (with local SSDs)	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
	Create an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Change the specifications of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔

Category	Feature	RDS High-availability Edition (with local SSDs)	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Instance management	Manage the parameters of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Set the protection level of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Migrate an ApsaraDB RDS for PostgreSQL instance across zones in the same region	✔	✔	✔
	Switch workloads over between primary and secondary ApsaraDB RDS for PostgreSQL instances	✔	✔	✔
	Restart an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Set the maintenance window of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Release an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Manage ApsaraDB RDS for PostgreSQL instances in the recycle bin	✔	✔	✔
Account management	Create an account on an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Reset the password of an account of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Lock an account of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Delete an account from an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Create a database on an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔

Category	Feature	RDS High-availability Edition (with local SSDs)	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Database management				
	Delete a database from an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Use plug-ins on an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
Connection management	Connect to an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Configure endpoints	✔	✔	✔
	View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
Monitoring and alerting	View the resource and engine metrics of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Set the monitoring frequency of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
	Manage the alert rules of an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔
Network management	Change the network type of an ApsaraDB RDS for PostgreSQL instance	✔	□	□
	Switch an ApsaraDB RDS for PostgreSQL instance to a different vSwitch	□	✔	✔
Read-only instances	Create a read-only ApsaraDB RDS for PostgreSQL instance	✔	✔	□
	Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance	✔	✔	✔

Category	Feature	RDS High-availability Edition (with local SSDs)	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Security management	Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance	✗	✓	✓
	Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance	✗	✓	✓
	Switch an ApsaraDB RDS for PostgreSQL instance to the enhanced whitelist mode	✓	✗	✗
	Create a fully encrypted ApsaraDB RDS for PostgreSQL instance	✗	✗	✗
Log and event history management	Manage the logs of an ApsaraDB RDS for PostgreSQL instance	✓	✓	✓
	View the event history of an ApsaraDB RDS for PostgreSQL instance	✓	✓	✓
Backup	Back up an ApsaraDB RDS for PostgreSQL instance	✓	✓	✓
	Use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance	✓	✓	✓
	Provide a free quota for backup storage for an ApsaraDB RDS for PostgreSQL instance	✓	✓	✓
	Download the data backup files and log backup files of an ApsaraDB RDS for PostgreSQL instance	✓	✗	✗
Restoration	Restore the data of an ApsaraDB RDS for PostgreSQL instance	✓	✓	✓
	Restore the data of an ApsaraDB RDS for PostgreSQL instance from a cross-region backup file	✓	✓	✓
Logical subscription	Logical subscription	✗	✓	✓
	Add tags to ApsaraDB RDS instances	✓	✓	✓

Category	Feature	RDS High-availability Edition (with local SSDs)	RDS High-availability Edition (with standard SSDs or ESSDs)	RDS Basic Edition (with standard SSDs or ESSDs)
Tag management	Remove tags from an ApsaraDB RDS for MySQL instance	✔	✔	✔
	Use tags to filter ApsaraDB RDS for MySQL instances	✔	✔	✔

3.6. PostgreSQL 9.4

This topic provides an overview of the features supported by ApsaraDB RDS for PostgreSQL instances that run PostgreSQL 9.4. You can purchase an ApsaraDB RDS for PostgreSQL instance that provides the features that you need. You can also query the features that are provided by an existing ApsaraDB RDS for PostgreSQL instance. In the following tables, ticks (√) indicate that a feature is supported, and crosses (×) indicate that a feature is not supported.

Category	Feature	RDS High-availability Edition (with local SSDs)
Instance management	Create an ApsaraDB RDS for PostgreSQL instance	✔
	Change the specifications of an ApsaraDB RDS for PostgreSQL instance	✔
	Manage the parameters of an ApsaraDB RDS for PostgreSQL instance	✔
	Set the protection level of an ApsaraDB RDS for PostgreSQL instance	□
	Migrate an ApsaraDB RDS for PostgreSQL instance across zones in the same region	✔
	Switch workloads over between primary and secondary ApsaraDB RDS for PostgreSQL instances	✔
	Restart an ApsaraDB RDS for PostgreSQL instance	✔
	Set the maintenance window of an ApsaraDB RDS for PostgreSQL instance	✔
	Upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance	✔
	Release an ApsaraDB RDS for PostgreSQL instance	✔

Category	Feature	RDS High-availability Edition (with local SSDs)
	Manage ApsaraDB RDS for PostgreSQL instances in the recycle bin	✔
Account management	Create an account on an ApsaraDB RDS for PostgreSQL instance	✔
	Reset the password of an account of an ApsaraDB RDS for PostgreSQL instance	✔
	Lock an account of an ApsaraDB RDS for PostgreSQL instance	✔
	Delete an account from an ApsaraDB RDS for PostgreSQL instance	✔
Database management	Create a database on an ApsaraDB RDS for PostgreSQL instance	✔
	Delete a database from an ApsaraDB RDS for PostgreSQL instance	✔
	Use plug-ins on an ApsaraDB RDS for PostgreSQL instance	✔
Connection management	Connect to an ApsaraDB RDS for PostgreSQL instance	✔
	Configure endpoints	✔
	View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance	✔
	Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance	✔
Monitoring and alerting	View the resource and engine metrics of an ApsaraDB RDS for PostgreSQL instance	✔
	Set the monitoring frequency of an ApsaraDB RDS for PostgreSQL instance	✔
	Manage the alert rules of an ApsaraDB RDS for PostgreSQL instance	✔
Network management	Change the network type of an ApsaraDB RDS for PostgreSQL instance	✔
	Switch an ApsaraDB RDS for PostgreSQL instance to a different vSwitch	□
Read-only instances	Create a read-only ApsaraDB RDS for PostgreSQL instance	□

Category	Feature	RDS High-availability Edition (with local SSDs)
Security management	Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance	✔
	Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance	✔
	Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance	✔
	Switch an ApsaraDB RDS for PostgreSQL instance to the enhanced whitelist mode	✔
	Create a fully encrypted ApsaraDB RDS for PostgreSQL instance	✔
Log and event history management	Manage the logs of an ApsaraDB RDS for PostgreSQL instance	✔
	View the event history of an ApsaraDB RDS for PostgreSQL instance	✔
Backup	Back up an ApsaraDB RDS for PostgreSQL instance	✔
	Use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance	✔
	Provide a free quota for backup storage for an ApsaraDB RDS for PostgreSQL instance	✔
	Download the data backup files and log backup files of an ApsaraDB RDS for PostgreSQL instance	✔
Restoration	Restore the data of an ApsaraDB RDS for PostgreSQL instance	✔
	Restore the data of an ApsaraDB RDS for PostgreSQL instance from a cross-region backup file	✔
Logical subscription	Logical subscription	✔
Tag management	Add tags to ApsaraDB RDS instances	✔
	Remove tags from an ApsaraDB RDS for MySQL instance	✔
	Use tags to filter ApsaraDB RDS for MySQL instances	✔

4.Specifications

4.1. Primary ApsaraDB RDS for PostgreSQL instance types

This topic provides an overview of primary ApsaraDB RDS for PostgreSQL instance types. The overview includes the most recent instance types, the earlier instance types, and the specifications for each instance type.

> **Note**
>
> For RDS instances that use standard SSDs or ESSDs, the memory includes the memory that is occupied by the underlying operating system and the memory that is occupied by the RDS-related management services. Therefore, the available memory of an RDS instance may be less than the memory capacity that is supported by the instance type.
>
> - The underlying operating system occupies 500 MB to 700 MB of memory.
> - The RDS-related management services occupy approximately 500 MB of memory.

Primary ApsaraDB RDS for PostgreSQL instances with local SSDs

RDS edition	PostgreSQL version	Instance family	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
RDS High-availability Edition	9.4 and 10	General-purpose instance	rds.pg.s2.large	2 cores, 4 GB	Unlimited	For more information, see the "IOPS" section in Primary instance types.	20 GB to 6,000 GB
			rds.pg.s3.large	4 cores, 8 GB			
			rds.pg.c1.large	8 cores, 16 GB			
			rds.pg.c1.xlarge	8 cores, 32 GB			
			rds.pg.c2.xlarge	16 cores, 64 GB			
			rds.pg.c2.2xlarge	16 cores, 128 GB			
		Dedicated instance (with a large memory	pg.x8.medium.2	2 cores, 16 GB	2,500	4,500	
			pg.x8.large.2	4 cores, 32 GB	5,000	9,000	
			pg.x8.xlarge.2	8 cores, 64 GB	10,000	18,000	
			pg.x8.2xlarge.2	16 cores, 128 GB	20,000	36,000	

RDS edition	PostgreSQL version	capacity) Instance family	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
		Dedicated host instance	rds.pg.st.h43	60 cores, 470 GB	64,000	50,000	

Primary ApsaraDB RDS for PostgreSQL instances with standard SSDs or ESSDs

RDS edition	PostgreSQL version	Instance family	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
RDS Basic Edition	10, 11, 12, 13, and 14	General-purpose instance	pg.n2.small.1	1 core, 2 GB	200	For more information, see the "IOPS" section in Primary instance types.	Standard SSD: 20 GB to 6,000 GB Enhanced SSD (ESSD) of PL1: 20 GB to 32,000 GB ESSD of PL2: 500 GB to 32,000 GB ESSD of PL3: 1,500 GB to 32,000 GB
			pg.n2.medium.1	2 cores, 4 GB	400		
			pg.n4.medium.1	2 cores, 8 GB	800		
			pg.n2.large.1	4 cores, 8 GB	800		
			pg.n4.large.1	4 cores, 16 GB	1,600		
			pg.n2.xlarge.1	8 cores, 16 GB	1,600		
			pg.n4.xlarge.1	8 cores, 32 GB	3,200		
			pg.n2.2xlarge.1	16 cores, 32 GB	3,200		
			pg.n4.2xlarge.1	16 cores, 64 GB	6,400		
			pg.n8.2xlarge.1	16 cores, 128 GB	10,000		
			pg.n4.4xlarge.1	32 cores, 128 GB	12,800		
			pg.n8.4xlarge.1	32 cores, 256 GB	20,000		
			pg.n4.8xlarge.1	64 cores, 256 GB	22,000		
			pg.n8.8xlarge.1	64 cores, 512 GB	48,000		

RDS edition	PostgreSQL version	Instance family	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
RDS High-availability Edition	10, 11, 12, 13, and 14	Dedicated instance	pg.x2.medium.2c	2 cores, 4 GB	400	For more information, see the "IOPS" section in Primary	Standard SSD: 20 GB to 6,000 GB ESSD of PL1: 20 GB to 32,000 GB ESSD of PL2: 500 GB to 32,000 GB
			pg.x4.medium.2c	2 cores, 8 GB	800		
			pg.x8.medium.2c	2 cores, 16 GB	1,600		
			pg.x2.large.2c	4 cores, 8 GB	800		
			pg.x4.large.2c	4 cores, 16 GB	1,600		
			pg.x8.large.2c	4 cores, 32 GB	3,200		
			pg.x2.xlarge.2c	8 cores, 16 GB	1,600		
			pg.x4.xlarge.2c	8 cores, 32 GB	3,200		
			pg.x8.xlarge.2c	8 cores, 64 GB	6,400		
			pg.x2.3large.2c	12 cores, 24 GB	2,400		
			pg.x4.3large.2c	12 cores, 48 GB	4,800		
			pg.x8.3large.2c	12 cores, 96 GB	9,600		
			pg.x2.2xlarge.2c	16 cores, 32 GB	3,200		
			pg.x4.2xlarge.2c	16 cores, 64 GB	6,400		
			pg.x8.2xlarge.2c	16 cores, 128 GB	12,800		
			pg.x2.3xlarge2c	24 cores, 48 GB	4,800		
			pg.x4.3xlarge.2c	24 cores, 96 GB	9,600		
			pg.x8.3xlarge.2c	24 cores, 192 GB	19,200		
			pg.x2.4xlarge.2c	32 cores, 64 GB	6,400		

RDS edition	PostgreSQL version	Instance family	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
			pg.x4.4xlarge.2c	32 cores, 128 GB	12,800		
			pg.x8.4xlarge.2c	32 cores, 256 GB	25,600		
			pg.x2.13large.2c	52 cores, 96 GB	9,600		
			pg.x4.13large.2c	52 cores, 192 GB	19,200		
			pg.x8.13large.2c	52 cores, 384 GB	38,400		
			pg.x2.8xlarge.2c	64 cores, 128 GB	12,800		
			pg.x4.8xlarge.2c	64 cores, 256 GB	25,600		
			pg.x8.8xlarge.2c	64 cores, 512 GB	51,200		
			pg.x2.13xlarge.2c	104 cores, 192 GB	19,200		
			pg.x4.13xlarge.2c	104 cores, 384 GB	38,400		
			pg.x8.13xlarge.2c	104 cores, 768 GB	76,800		

Primary ApsaraDB RDS for PostgreSQL instances with new general-purpose instance types

The new general-purpose instance types of ApsaraDB RDS for PostgreSQL provide better scalability and higher performance than the previous general-purpose instance types. In addition, the amount of time that is required to create an RDS instance and the amount of time that is required to change the specifications of an RDS instance are reduced.

> **Note** The new general-purpose instance types do not support the features that are described in the following topics:
>
> - Use the cloud migration feature for an ApsaraDB RDS for PostgreSQL instance
> - Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance
> - Connect an ApsaraDB RDS for PostgreSQL instance to a self-managed AD domain
> - Upgrade an ApsaraDB RDS for PostgreSQL instance from Basic Edition to High-availability Edition
> - Use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance

Before you can select a new general-purpose instance type for an RDS instance, make sure that the RDS instance meets the following requirements:

- The RDS instance runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, or PostgreSQL 14.
- The RDS instance uses ESSDs of PL1, ESSDs of PL2, or ESSDs of PL3.
- The new general-purpose instance types are available in the following Alibaba Cloud regions: Singapore (Singapore), China (Hangzhou), China (Beijing), China (Shanghai), China (Shenzhen), and China (Hong Kong). The new general-purpose instance types are to be released in the other Alibaba Cloud regions soon. If you want to create an RDS instance that uses a new general-purpose instance type, go to the ApsaraDB RDS console.
- If an RDS instance uses a new general-purpose instance type, you can change the instance type of the RDS instance only to a different new general-purpose instance type.
- If you want to create an RDS instance that uses a new general-purpose instance type, make sure that the AliyunServiceRoleForRdsPgsqlOnEcs role are assumed by ApsaraDB RDS for PostgreSQL. For more information, see Service-linked roles.

RDS edition	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
RDS Basic Edition	pg.n2.2c.1m	2 cores, 4 GB	400	For more information, see the "IOPS" section in Primary instance types.	ESSD of PL1: 20 GB to 32,000 GB ESSD of PL2: 500 GB to 32,000 GB ESSD of PL3: 1,500 GB to 32,000 GB
	pg.n4.2c.1m	2 cores, 8 GB	800		
	pg.n2.4c.1m	4 cores, 8 GB	800		
	pg.n4.4c.1m	4 cores, 16 GB	1,600		
	pg.n4.6c.1m	6 cores, 24 GB	2,400		
	pg.n4.8c.1m	8 cores, 32 GB	3,200		

RDS edition	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
RDS High-availability Edition	pg.n2.2c.2m	2 cores, 4 GB	400	For more information, see the "IOPS" section in Primary instance types.	ESSD of PL1: 20 GB to 32,000 GB ESSD of PL2: 500 GB to 32,000 GB ESSD of PL3: 1,500 GB to 32,000 GB
	pg.n4.2c.2m	2 cores, 8 GB	800		
	pg.n4.4c.2m	4 cores, 16 GB	1,600		
	pg.n4.6c.2m	6 cores, 24 GB	2,400		
	pg.n4.8c.2m	8 cores, 32 GB	3,200		
	pg.n4.12c.2m	12 cores, 48GB	4,800		

Phased-out instance types of ApsaraDB RDS for PostgreSQL

The following table describes the phased-out instance types of ApsaraDB RDS for PostgreSQL. These instance types are no longer available to new instances.

Instance type	CPU cores	Memory capacity	Maximum connections	Maximum IOPS
rds.pg.t1.small	1	1GB	100	600
pg.x8.4xlarge.2	32	256GB	20000	50000
pg.n1.micro.1	1	1GB	100	For more information, see the "IOPS" section in Primary instance types.
pg.gn5i-c2g1.large.1	2	8 GB	800	
pg.gn5i-c4g1.xlarge.1	4	16 GB	1,600	
pg.gn5i-c8g1.2xlarge.1	8	32 GB	3,200	
pg.gn5i-c16g1.4xlarge.1	16	64 GB	6,400	
pg.gn5i-c16g1.8xlarge.1	32	128 GB	12,800	
pg.gn5i-c28g1.14xlarge.1	56	224 GB	22,000	
pg.n2.small.2c	1	2	200	
pg.n2.medium.2c	2	4	400	
rds.pg.s1.small	1	2	200	

4.2. Read-only ApsaraDB RDS for PostgreSQL instance types

This topic provides an overview of read-only ApsaraDB RDS for PostgreSQL instance types. This overview includes the most recent instance types, the earlier instance types, and the specifications for each instance type.

> **Note** The subscription and pay-as-you-go billing methods are supported for read-only ApsaraDB RDS for PostgreSQL instances. For more information, see Read-only ApsaraDB RDS instance types. For more information about the prices, go to the buy page.

Read-only ApsaraDB RDS for PostgreSQL instances with local SSDs

Database engine version	Instance family	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
PostgreSQL 10	Dedicated instance family (with a large memory capacity)	pg.x8.xlarge.2	8 cores, 64 GB	10,000	18,000	20 GB to 6,000 GB
		pg.x8.2xlarge.2	16 cores, 128 GB	12,000	36,000	
	Dedicated instance family (with a large number of cores)	pg.x4.xlarge.2	8 cores, 32 GB	5,000	9,000	
		pg.x4.2xlarge.2	16 cores, 64 GB	10,000	18,000	
		pg.x4.4xlarge.2	32 cores, 128 GB	12,000	36,000	
	Dedicated host instance family	rds.pg.st.h43	60 cores, 470 GB	4,000	50,000	

Read-only ApsaraDB RDS for PostgreSQL instances with standard SSDs or enhanced SSDs (ESSD)

Database engine version	Instance family	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
PostgreSQL 14 PostgreSQL 13, PostgreSQL12, PostgreSQL11, and PostgreSQL 10	Dedicated instance family	pgro.x2.medium.1c	2 cores, 4 GB	400	For more information, see Maximum IOPS for standard SSDs and ESSDs.	50 GB to 32,000 GB
		pgro.x2.large.1c	4 cores, 8 GB	800		
		pgro.x2.xlarge.1c	8 cores, 16 GB	1,600		
		pgro.x2.3large.1c	12 cores, 24 GB	2,400		
		pgro.x2.2xlarge.1c	16 cores, 32 GB	3,200		
		pgro.x2.3xlarge.1c	24 cores, 48 GB	4,800		
		pgro.x2.4xlarge.1c	32 cores, 64 GB	6,400		
		pgro.x2.13large.1c	52 cores, 96 GB	9,600		
		pgro.x2.8xlarge.1c	64 cores, 128 GB	12,800		
		pgro.x2.13xlarge.1c	104 cores, 192 GB	19,200		
		pgro.x4.medium.1c	2 cores, 8 GB	800		
		pgro.x4.large.1c	4 cores, 16 GB	1,600		
		pgro.x4.xlarge.1c	8 cores, 32 GB	3,200		
		pgro.x4.3large.1c	12 cores, 48 GB	4,800		
		pgro.x4.2xlarge.1c	16 cores, 64 GB	6,400		
		pgro.x4.3xlarge.1c	24 cores, 96 GB	9,600		
		pgro.x4.4xlarge.1c	32 cores, 128 GB	12,800		
		pgro.x4.13large.1c	52 cores, 192 GB	19,200		
		pgro.x4.8xlarge.1c	64 cores, 256 GB	25,600		
		pgro.x4.13xlarge.1c	104 cores, 384 GB	38,400		
		pgro.x8.medium.1c	2 cores, 16 GB	1,600		
		pgro.x8.large.1c	4 cores, 32 GB	3,200		
		pgro.x8.xlarge.1c	8 cores, 64 GB	6,400		
		pgro.x8.3large.1c	12 cores, 96 GB	9,600		

Database engine version	Instance family	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
		pgro.x8.2xlarge.1c	16 cores, 128 GB	12,800		
		pgro.x8.3xlarge.1c	24 cores, 192 GB	19,200		
		pgro.x8.4xlarge.1c	32 cores, 256 GB	25,600		
		pgro.x8.13large.1c	52 cores, 384 GB	38,400		
		pgro.x8.8xlarge.1c	64 cores, 512 GB	51,200		
		pgro.x8.13xlarge.1c	104 cores, 768 GB	76,800		

Read-only ApsaraDB RDS for PostgreSQL instances with the new general-purpose instance types

Note The new general-purpose instance types of ApsaraDB RDS for PostgreSQL provide better scalability and higher performance than the previous general-purpose instance types. In addition, the amount of time that is required to create an RDS instance and the amount of time that is required to change the specifications of an RDS instance are reduced. If a primary RDS instance runs RDS High-availability Edition and uses a new general-purpose instance type, you can create read-only RDS instances with the new general-purpose instance types for the primary RDS instance. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types.

Database engine version	Instance family	Instance type	CPU and memory specifications	Maximum number of connections	Maximum IOPS	Storage capacity
PostgreSQL 14 PostgreSQL 13, PostgreSQL12, PostgreSQL11, and PostgreSQL 10	General-purpose instance family	pgro.n4.2c.1m	2 cores, 8 GB	800	For more information, see Maximum IOPS for standard SSDs and ESSDs.	20 GB to 32,000 GB
		pgro.n4.4c.1m	4 cores, 16 GB	1,600		
		pgro.n4.6c.1m	6 cores, 24 GB	2,400		
		pgro.n4.8c.1m	8 cores, 32 GB	3,200		
		pgro.n4.12c.1m	12 cores, 48 GB	4,600		

5.Billing

5.1. Switch an ApsaraDB RDS for PostgreSQL instance from pay-as-you-go to subscription

This topic describes how to switch an ApsaraDB RDS for PostgreSQL instance from pay-as-you-go to subscription.

Prerequisites

- The instance type of your RDS instance is not deprecated. For more information, see Primary instance types. If the instance type of your RDS instance is deprecated, you must change the instance type before you switch the instance from pay-as-you-go to subscription. For more information, see 变更配置.
- Your RDS instance uses the pay-as-you-go billing method.
- Your RDS instance is in the Running state.
- Your RDS instance does not have an unpaid subscription order.

> **Note** Read-only RDS instances that use standard or enhanced SSDs do not support the subscription billing method.

Impacts

A billing method change for your RDS instance does not affect the workloads on your RDS instance.

Precautions

- If your RDS instance has an unpaid subscription order, the order becomes invalid when you change the instance type. In this case, you must cancel the order in the Billing Management console. Then, you can change the billing method of your RDS instance again.

Procedure

1. Log on to the ApsaraDB RDS console. In the left-side navigation pane, click **Instances**. In the top navigation bar, select the region where your RDS instance resides.
2. Find your RDS instance and use one of the following methods to go to the **Switch to Subscription Billing** page:
   - Click **Switch to Subscription Billing** in the **Billing Method** column.
   - Click the ID of your RDS instance. In the **Status** section of the page that appears, click **Subscription Billing** on the right of **Billing Method**.
3. Configure the **Duration** parameter. Then, read and select Terms of Service.
4. Click **Pay Now**.

> **Note** ApsaraDB RDS generates a subscription order. You must pay for the order. If the order is not paid or canceled, you cannot purchase an RDS instance or change the billing method of your RDS instance from pay-as-you-go to subscription. You can pay for or cancel the order in the Billing Management console.

5. Complete the payment.

Related operations

Operation	Description
Change the billing method	Changes the billing method of an ApsaraDB RDS instance.

5.2. Switch an ApsaraDB RDS for MySQL instance from subscription to pay-as-you-go

This topic describes how to switch an ApsaraDB RDS for MySQL instance from the subscription billing method to the pay-as-you-go billing method based on your business requirements.

Prerequisites

- Your RDS instance uses the subscription billing method. For more information about the billing methods of ApsaraDB RDS, see Billable items, billing methods, and pricing.
- Your RDS instance is in the Running state.

  > **Note** If a subscription RDS instance is locked due to expiration, you must first renew the RDS instance. For more information about how to renew an RDS instance, see Manually renew an ApsaraDB RDS for MySQL instance.

- Your RDS instance does not use a phased-out instance type. For more information, see Primary instance types. If your RDS instance uses a phased-out instance type, you must change the instance type before you switch your RDS instance to the pay-as-you-go billing method.

Billing

After you switch your RDS instance to the pay-as-you-go billing method, a refund is returned based on the payment method that is used.

Refund = Fee actually paid - Fee for consumed resources

- The fee actually paid is the money that you paid and does not include the amount that is covered by coupons or vouchers.
- The fee for consumed resources is calculated based on the following formula: Fee for consumed resources = Daily fee x Consumed subscription duration x Discount for the consumed subscription duration. The daily fee is equal to the order-specific fee divided by 30.

> **Note** The consumed subscription duration is accurate to the day. The part that is less than one day is counted as one day.

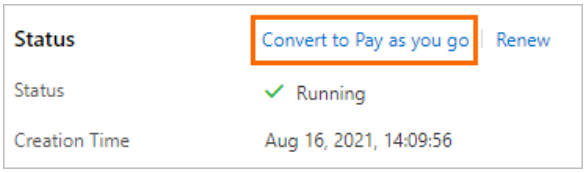
Impacts

When you switch your RDS instance to the pay-as-you-go billing method, the workloads on your RDS instance run as normal.

> **Note** The subscription billing method is more cost-effective than the pay-as-you-go billing method, and you are offered higher discounts for longer subscription periods. For long-term use, we recommend that you select the subscription billing method.

Procedure

1.
2. In the **Status** section of the **Basic Information** page, click **Convert to Pay as you go**.



3. Confirm the configuration of your RDS instance, read and select Terms of Service, click **Pay Now**, and then complete the payment.

Related operations

Operation	Description
Change the billing method	Changes the billing method of an ApsaraDB RDS instance.

5.3. Manually renew an ApsaraDB RDS for PostgreSQL instance

This topic describes how to manually renew an ApsaraDB RDS for PostgreSQL instance. If your RDS instance uses subscription billing, you must renew it before it expires. This allows you to prevent service interruptions and data loss.

For more information about the impacts caused by subscription expiration, see Unlock or rebuild an expired or overdue ApsaraDB for RDS instance.

> **Note** RDS instances that use the pay-as-you-go billing method do not expire and therefore do not require renewal.

You can manually renew your RDS instance before it expires or within 15 days after it expires.

Method 1: Renew an RDS instance in the ApsaraDB RDS console

Renew a single RDS instance

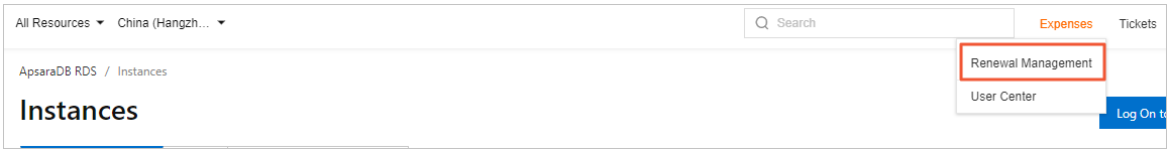
1.
2. In the **Status** section of the page that appears, click **Renew** on the right.
3. On the **Renew** page, configure the **Duration** parameter. You are offered lower prices for longer subscription periods.
4. Read and select Terms of Service, click **Pay Now**, and then complete the payment.

Renew multiple RDS instances at a time

1.
2. Select the RDS instances that you want to renew and click **Renew** below the instance list.
3. In the **Renew** dialog box, confirm the selected RDS instances and click **OK** to go to the **Renewal** page.
4. On the **Manual** tab, select the RDS instances and click **Batch Renew** in the lower part of the page.
5. Configure the **Duration** parameter of each RDS instance, click **Pay**, and then complete the payment.

Method 2: Renew the instance in the Billing Management console

1. Log on to the ApsaraDB RDS console.
2. In the top navigation bar, choose **Expenses > Renewal Management**.



3. On the **Manual** tab of the Renewal page, find the RDS instances that you want to renew. You can renew one or more RDS instances at a time.
   - **Renew a single RDS instance**
     a. Find the RDS instance that you want to renew and click **Renew** in the Actions column.

     > **Note** If the RDS instance is displayed on the **Auto** or **Nonrenewal** tab, you can click **Enable Manual Renewal** in the Actions column and then click **OK** in the message that appears to manually renew the RDS instance.

     b. On the page that appears, configure the Duration parameter, click **Pay Now**, and then complete the payment.
   - **Renew multiple RDS instances at a time**
     a. Select the RDS instances that you want to renew and click **Batch Renew** in the lower part of the page.
     b. Configure the **Duration** parameter of each RDS instance, click **Pay**, and then complete the payment.

Enable auto-renewal

If you enable auto-renewal for your RDS instance, you do not need to manually renew your RDS instance. If your Alibaba Cloud account has a sufficient balance, your RDS instance never expires. For more information, see Enable auto-renewal for an ApsaraDB RDS for PostgreSQL instance.

5.4. Enable auto-renewal for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to enable auto-renewal for an ApsaraDB RDS for PostgreSQL instance. If your RDS instance uses subscription billing, you can enable auto-renewal. This relieves the need to manually renew your RDS instance. Make sure that your Alibaba Cloud account has a sufficient balance, and your RDS instance will never expire.

If your RDS instance uses the subscription billing method, subscription instances can expire. If you do not renew your RDS instance before it expires, your service is interrupted and data may be lost. For more information, see Unlock or rebuild an expired or overdue ApsaraDB for RDS instance.

> **Note** RDS instances that use the pay-as-you-go billing method do not expire and therefore do not require renewal.

Precautions

- If you enable auto-renewal, the first time when the system deducts fees from your Alibaba Cloud account comes at 08:00:00 three days before your RDS instance expires. If the deduction fails, the system will attempt to deduct the fee every day for the next two days.

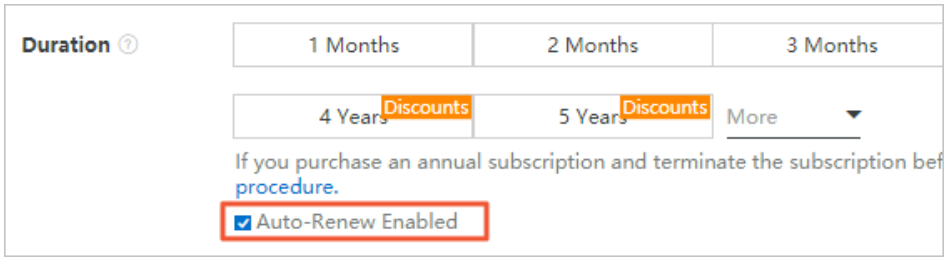
  > **Note** Make sure that the balance of your Alibaba Cloud account is sufficient. Otherwise, the renewal fails. If all the three automatic fee deduction attempts fail, you must manually renew your RDS instance in a timely manner to avoid service interruption and data loss.

- If you manually renew your RDS instance before the automatic fee deduction, the system will automatically renew the instance next time before the expiration.
- After you enable auto-renewal, it takes effect the next day. If your RDS instance is due to expire the next day, renew it manually to avoid service interruption. For more information, see Manually renew an ApsaraDB RDS for PostgreSQL instance.

Enable auto-renewal when you purchase an RDS instance

> **Note** If you select auto-renewal when you purchase an RDS instance, the system automatically renews the RDS instance based on the specified renewal cycle. The renewal cycle is one month or one year. For example, if you select auto-renewal when you purchase an RDS instance with a six-month subscription, the system automatically renews the RDS instance with a one-month subscription each time the instance is due to expire.

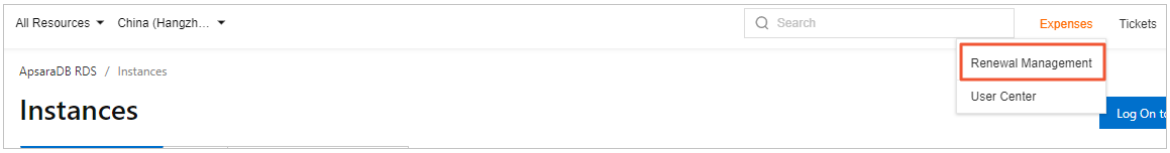
When you purchase a subscription RDS instance, select **Auto-Renew Enabled**.



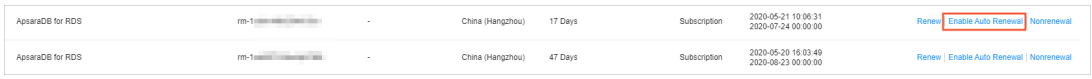
Enable auto-renewal after you purchase an RDS instance

> **Note** After you enable auto-renewal for a created RDS instance, the system automatically renews the RDS instance based on the selected renewal cycle. For example, if you select a three-month renewal cycle, you are charged for a three-month subscription in each renewal cycle.

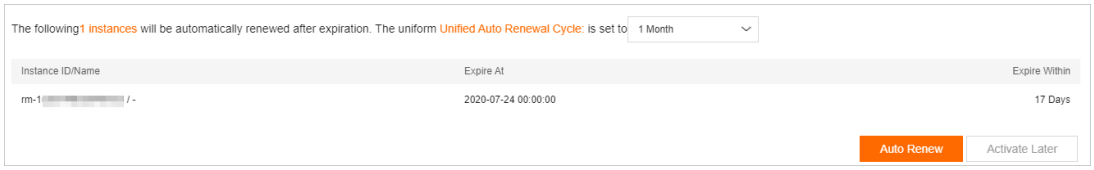
1. Log on to the ApsaraDB RDS console.
2. In the top navigation bar, choose **Expenses > Renewal Management**.



3. On the **Manual** or **Nonrenewal** tab, specify the filter conditions to find the RDS instance for which you want to enable auto-renewal. You can enable auto-renewal for one or more RDS instances at a time.
   - Enable auto-renewal for a single RDS instance.
     a. Find the RDS instance and in the Actions column click **Enable Auto Renewal**.

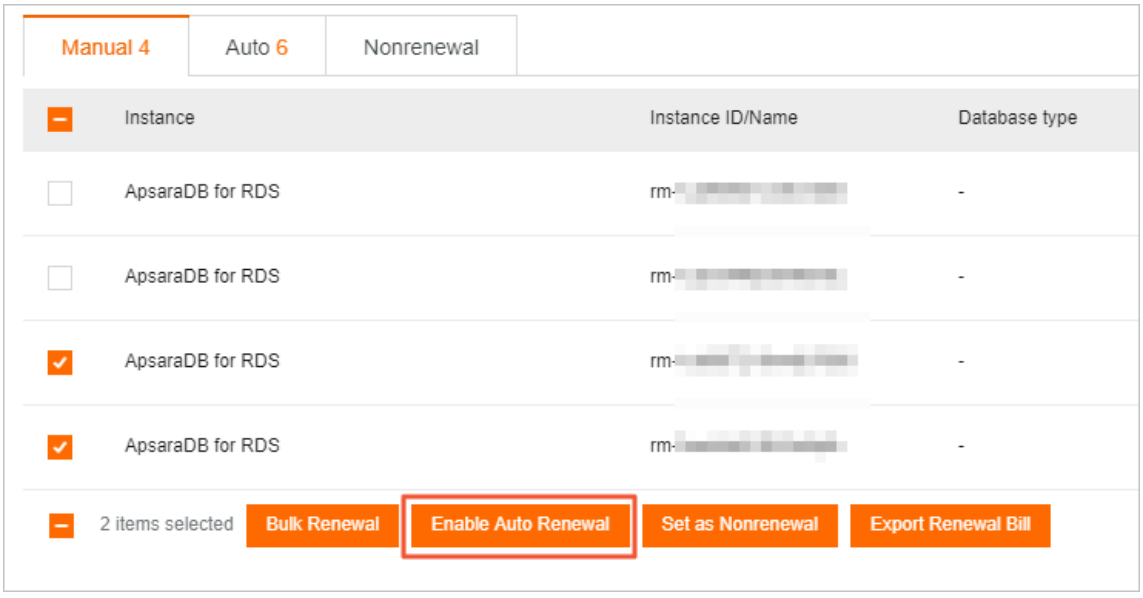


     b. In the dialog box that appears, specify the **Unified Auto Renewal Cycle** parameter and click **Auto Renew**.

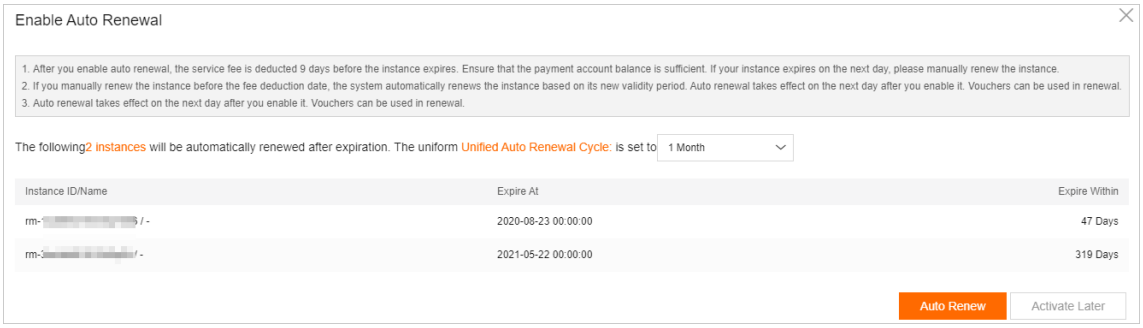


   - Enable auto-renewal for multiple RDS instances.

     Select the RDS instances and click **Enable Auto Renewal** below the instance list.

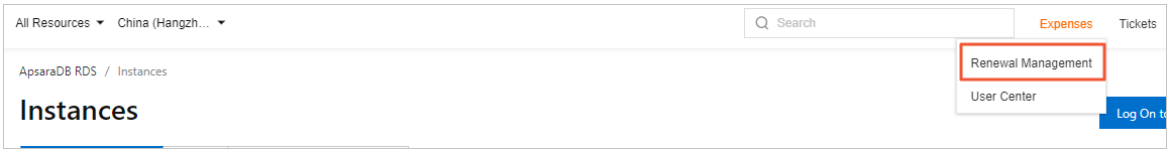


   - In the dialog box that appears, specify the **Unified Auto Renewal Cycle** parameter and click **Auto Renew**.

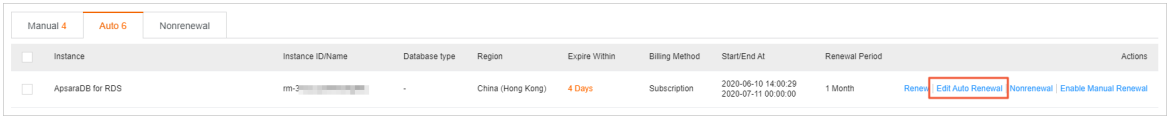


Change the auto-renewal cycle

1. Log on to the ApsaraDB RDS console.
2. In the top navigation bar, choose **Expenses > Renewal Management**.



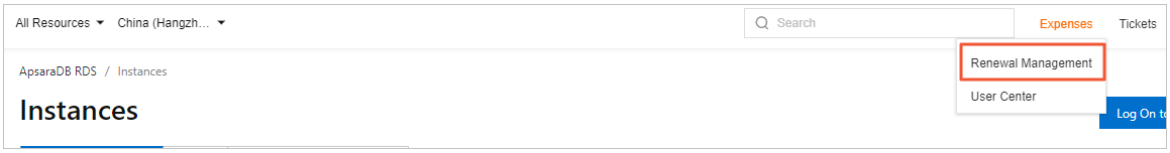
3. On the **Auto** tab, specify filter conditions to find the RDS instance for which you want to enable auto-renewal. Then, select the RDS instance and click **Edit Auto Renewal** in the Actions column.



4. In the dialog box that appears, change the auto-renewal cycle and click **OK**.

Disable auto-renewal

1. Log on to the ApsaraDB RDS console.
2. In the top navigation bar, choose **Expenses > Renewal Management**.



3. On the **Auto** tab, specify filter conditions to find the RDS instance for which you want to enable auto-renewal. Then, select the RDS instance and click **Enable Manual Renewal** in the Actions column.

4. In the message that appears, click **OK**.

Related operations

Operation	Description

Operation	Description
Create an instance	Creates an ApsaraDB RDS instance. **Note** You can call this operation to enable auto-renewal for an RDS instance that you want to create.
Manually renew an ApsaraDB for RDS instance	Renews an ApsaraDB RDS instance. **Note** You can call this operation to enable auto-renewal for a created RDS instance.

6.Plug-ins

6.1. Supported plug-ins

This topic provides an overview of the plug-ins and plug-in versions that are supported by ApsaraDB RDS for PostgreSQL instances.

Note

- If your RDS instance does not support a few plug-ins that are described in the following table, you must update the minor engine version of your RDS instance to the latest version. For more information, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.
- If you have requirements or suggestions for the plug-ins that are described in the following table, you can submit a ticket.
- The following table describes only common plug-ins. If you want to query the plug-ins that are supported by your RDS instance, you can run the `SELECT * FROM pg_available_extensions;` command.
- Before you use some plug-ins, you must add the names of the plug-ins to the value of the **shared_preload_libraries** parameter of your RDS instance. Otherwise, you cannot create the plug-ins.
  - The plug-ins include pg_stat_statements, auth_delay, passwordcheck, auto_explain, pg_pathman, orafce, pgaudit, pglogical, pg_bigm, pg_cron, pg_hint_plan, pg_prewarm, zhparser, timescaledb, pldebugger, and pg_jieba. If you want to use the pldebugger plug-in, you must add plugin_debugger to the value of the shared_preload_libraries parameter of your RDS instance.
  - For more information about how to configure the **shared_preload_libraries** parameter, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

Plug-in	14	13	12	11	10	9.4	Description
address_standardizer	3.1.4	3.1.4	3.1.4	3.1.4	3.1.4	2.5.4	This plug-in is used to standardize the names of geographic locations based on Postal Address Geocoder (PAGC). For more information, see Tuning the Standardizer.
address_standardizer_data_us	3.1.4	3.1.4	3.1.4	3.1.4	3.1.4	2.5.4	This plug-in is used to standardize the names of the geographic locations in the United States based on PAGC. For more information, see Tuning the Standardizer.

Plug-in	14	13	12	11	10	9.4	Description
aggs_for_arrays	Not supported	Not supported	Not supported	Not supported	1.3.1	Not supported	This plug-in provides an extension function that is used to compute the statistics of numeric arrays.
bloom	1.0	1.0	1.0	1.0	1.0	Not supported	This plug-in provides an index access method that is based on Bloom filters.
btree_gin	1.3	1.3	1.3	1.3	1.2	1.0	This plug-in provides sample GIN operator classes that are used to implement B-tree equivalent behavior for multiple data types and all enumerated data types.
btree_gist	1.6	1.5	1.5	1.5	1.5	1.0	This plug-in provides sample GiST operator classes that are used to implement B-tree equivalent behavior for multiple data types and all enumerated data types.
chkpass	Not supported	Not supported	Not supported	Not supported	1.0	1.0	This plug-in provides a data type that is used to store encrypted passwords.
citext	1.6	1.6	1.6	1.5	1.4	1.0	This plug-in provides a string type that is not case-sensitive.
cube	1.5	1.4	1.4	1.4	1.2	1.0	This plug-in provides a data type that is used to represent multidimensional cubes.
dblink	1.2	1.2	1.2	1.2	1.2	1.1	This plug-in is used to manage tables across databases.
decoderbufs	Not supported	0.1.0	0.1.0	0.1.0	0.1.0	Not supported	This plug-in is used to generate data that is compatible with the Debezium platform based on the Protocol Buffers protocol.

Plug-in	14	13	12	11	10	9.4	Description
dict_int	1.0	1.0	1.0	1.0	1.0	1.0	This plug-in provides a sample add-on dictionary template that is used to run full-text searches.
earthdistance	1.1	1.1	1.1	1.1	1.1	1.0	This plug-in provides two different methods that are used to calculate great-circle distances on the surface of the Earth.
encdb	1.1.9	1.1.9	1.1.9	1.1.9	1.1.9	Not supported	This plug-in is used to support fully encrypted databases.
fuzzystrmatch	1.1	1.1	1.1	1.1	1.1	1.0	This plug-in is used to calculate the similarity and distance between strings.
ganos_address_standardizer	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to standardize the names of geographical locations based on PAGC. For more information, see Tuning the Standardizer.
ganos_address_standardizer_data_us	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to standardize the names of the geographic locations in the United States based on PAGC. For more information, see Tuning the Standardizer.
ganos_geometry	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to compute and analyze spatial geometries.
ganos_geometry_sfcgal	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to compute and analyze spatial geometries.

Plug-in	14	13	12	11	10	9.4	Description
ganos_geometry_topology	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is an extension of the SFCGAL plug-in that is used to process spatial geometries.
ganos_networking	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to compute and analyze spatial network geometries.
ganos_pointcloud	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to store, compute, and analyze point clouds.
ganos_pointcloud_geometry	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to store, compute, and analyze point clouds.
ganos_raster	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to store, compute, and analyze spatial grids.
ganos_spatialref	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to compute and analyze spatial references.
ganos_tiger_geocoder	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to support the Topologically Integrated Geographic Encoding and Referencing (TIGER) data format that is used by the United States Census Bureau (USCB).
ganos_trajectory	4.6	4.6	4.6	4.6	4.6	Not supported	This plug-in is used to compute and analyze objects in the moving object detection (MOD) system of Ganos.

Plug-in	14	13	12	11	10	9.4	Description
hll	2.16	2.15	2.14	2.14	Not supported	Not supported	This plug-in is used to estimate business metrics, such as page views (PV) and unique visitors (UV), at fast speeds.
hstore	1.8	1.7	1.6	1.5	1.4	1.3	This plug-in is used to store key-value pairs within a single PostgreSQL value.
hypopg	1.3.1	1.3.1	1.3.1	1.3.1	1.3.1	Not supported	This plug-in is used to create virtual indexes.
imgsmlr	Not supported	Not supported	Not supported	Not supported	Not supported	1.0	This plug-in is used to search for similar images.
index_adviser	2.0	2.0	2.0	2.0	2.0	Not supported	This plug-in is used to recommend indexes.
intagg	1.1	1.1	1.1	1.1	1.1	1.0	This plug-in provides an integer aggregator and an enumerator.
intarray	1.5	1.3	1.2	1.2	1.2	1.0	This plug-in provides functions and operators that are used to manage null-free arrays of integers.
isn	1.2	1.2	1.2	1.2	1.1	1.0	This plug-in is used to validate input numbers and hyphenate output numbers based on a hard-coded list of prefixes.
ltree	1.2	1.2	1.1	1.1	1.1	1.0	This plug-in is used to label the data that is stored in a hierarchical tree structure.

Plug-in	14	13	12	11	10	9.4	Description
jsonbx	Not supported	Not supported	Not supported	Not supported	Not supported	1.0	This plug-in is an extension that is used in PostgreSQL 9.4 to support JSONB functions.
log_fdw	Not supported	Not supported	Not supported	1.0	Not supported	Not supported	This plug-in is used to query the logs.
madlib	Not supported	Not supported	1.18.0	1.18.0	Not supported	Not supported	This plug-in is an open source library that is used for machine learning and graph computing models.
mysql_fdw	Not supported	1.1	1.1	1.1	1.1	Not supported	This plug-in is used to read data from and write data to an ApsaraDB RDS for MySQL instance or a self-managed MySQL database.
oracle_fdw	Not supported	Not supported	1.1	Not supported	Not supported	Not supported	This plug-in is used to synchronize the update operations on the tables of an ApsaraDB RDS for PostgreSQL instance to the tables of an Oracle database.
orafce	3.17	Not supported	Not supported	3.8	3.6	3.6	This plug-in provides functions that are compatible with Oracle.
oss_fdw	Not supported	1.1	1.1	1.1	1.1	1.1	This plug-in is used to read data from and write data to an Object Storage Service (OSS) bucket.
pase	Not supported	Not supported	Not supported	0.0.1	Not supported	Not supported	This plug-in is used to efficiently search for vectors.

Plug-in	14	13	12	11	10	9.4	Description
pg_bigm	1.2	1.2	1.2	1.2	1.2	Not supported	This plug-in is used to create a 2-gram Generalized Inverted Index (GIN) that is used to accelerate full-text searches.
pg_buffercache	1.3	1.3	1.3	1.3	1.3	1.0	This plug-in is used to examine shared buffers in real time.
pg_concurrency_control	Not supported	Not supported	Not supported	1.0	1.0	1.0	This plug-in is used to control the concurrency of SQL statements.
pg_cron	1.1	1.1	1.1	1.1	1.1	Not supported	This plug-in is used to configure scheduled tasks.
pg_freespacemap	1.2	1.2	1.2	1.2	1.2	1.0	This plug-in is used to examine the free space map (FSM).
pg_jieba	Not supported	1.1.0	1.1.0	1.1.0	1.1.0	Not supported	This plug-in is used to segment Chinese texts.
pg_hint_plan	Not supported	1.3.7	1.3.7	1.3.7	1.3.0	1.1.3	This plug-in is used to add hints to SQL statements. The hints are used to change the execution plans of SQL statements.
pg_pathman	Not supported	1.5	1.5	1.5	1.5	Not supported	This plug-in is used to partition tables at high performance.
pg_prewarm	1.2	1.2	1.2	1.2	1.1	1.0	This plug-in is used to load data to the buffer of the operating system or PostgreSQL database engine.

Plug-in	14	13	12	11	10	9.4	Description
pg_repack	Not supported	1.4.6	1.4.6	1.4.6	1.4.6	Not supported	This plug-in is used to clear tablespaces that stay online.
pg_sphere	Not supported	Not supported	Not supported	Not supported	1.0	1.0	This plug-in provides spherical data types, functions, operators, and indexes for PostgreSQL.
pg_stat_statements	1.9	1.8	1.7	1.6	1.6	1.2	This plug-in is used to track the statistics of all SQL statements that are executed on a server.
pg_trgm	1.6	1.5	1.4	1.4	1.3	1.1	This plug-in provides functions and operators that are used to calculate the similarity between alphanumeric texts. This plug-in also provides index operator classes that are used to search for similar strings at fast speeds.
pgaudit	1.6.1	1.5	1.4.1	1.3.2	1.2.2	Not supported	This plug-in is used to generate audit logs. The audit logs contain details about sessions and objects.
pgcrypto	1.3	1.3	1.3	1.3	1.3	1.1	This plug-in provides cryptographic functions for PostgreSQL.
pglogical	2.4.0	2.4.0	2.4.0	2.4.0	2.4.0	Not supported	This plug-in provides the logical streaming replication feature by using a publish/subscribe pattern.
pgrouting	2.6.2	2.6.2	2.6.2	2.6.2	2.6.2	2.0.0	This plug-in is used to compute and analyze spatial network geometries.
pgrowlocks	1.2	1.2	1.2	1.2	1.2	1.1	This plug-in provides a function that is used to display the row lock information of a specified table.

Plug-in	14	13	12	11	10	9.4	Description
pgstattuple	1.5	1.5	1.5	1.5	1.5	1.2	This plug-in provides various functions that are used to obtain tuple-level statistics.
pldebugger	Not supported	1.1	1.1	1.1	1.1	Not supported	This plug-in is used to debug stored procedures.
plperl	1.0	1.0	1.0	1.0	1.0	1.0	This plug-in is used to support the Perl procedural language.
plpgsql	1.0	1.0	1.0	1.0	1.0	1.0	This plug-in is used to support the SQL procedural language.
plproxy	2.10.0	2.10.0	2.9.0	2.9.0	2.8.0	Not supported	This plug-in provides two modes that are used to access an ApsaraDB RDS for PostgreSQL instance: CLUSTER and CONNECT.
pltcl	1.0	1.0	1.0	1.0	1.0	1.0	This plug-in is used to support the TCL procedural language.
plv8	Not supported	2.3.15	2.3.15	2.3.15	2.3.15	1.4.2	This plug-in is a trusted JavaScript language extension.
postgis	3.1.4	3.1.4	3.1.4	3.1.4	3.1.4	2.5.4	This plug-in is an extension that is used to process and store spatial geographic information.
postgis_sfcgal	3.1.4	3.1.4	3.1.4	3.1.4	3.1.4	Not supported	This plug-in is an extension that is used to manage spatial geographic information in PostGIS.
postgis_tiger_geocoder	3.1.4	3.1.4	3.1.4	3.1.4	3.1.4	2.5.4	This plug-in is an extension that is used to manage PostGIS data in the TIGER format.

Plug-in	14	13	12	11	10	9.4	Description
postgis_topology	3.1.4	3.1.4	3.1.4	3.1.4	3.1.4	2.5.4	This plug-in is an extension that is used to manage topological objects in PostGIS.
postgres_fdw	1.1	1.0	1.0	1.0	1.0	1.0	This plug-in is used to manage tables across databases.
q3c	Not supported	Not supported	Not supported	Not supported	1.5.0	1.5.0	This plug-in is used to create spatial indexes on a sphere.
rdkit	Not supported	Not supported	3.8	Not supported	Not supported	3.4	This plug-in is used to support features such as molecular computing and search.
roaringbitmap	0.5	0.5	0.5	Not supported	Not supported	Not supported	This plug-in is used to compute bitmaps to increase query performance.
rum	Not supported	1.3	1.3	1.3	1.3	Not supported	This plug-in is used to efficiently run full-text searches.
sequential-uuids	1.0.2	1.0.2	1.0.2	1.0.2	1.0.2	Not supported	This plug-in is used to generate sequential UUIDs.
smlar	1.0	1.0	1.0	1.0	1.0	1.0	This plug-in is used to calculate the similarity between two arrays of the same data type.

Plug-in	14	13	12	11	10	9.4	Description
sql_firewall	Not supported	Not supported	0.8	0.8	0.8	Not supported	This plug-in is used as a firewall to protect databases from SQL injection attacks.
sslinfo	1.2	1.2	1.2	1.2	1.2	1.0	This plug-in is used to obtain information about the SSL certificate that is provided by the connected client.
tablefunc	1.0	1.0	1.0	1.0	1.0	1.0	This plug-in provides functions that are used to return tables.
tds_fdw	Not supported	Not supported	2.0.1	2.0.1	Not supported	Not supported	This plug-in is used to query data from an ApsaraDB RDS instance that does not run PostgreSQL.
timescaledb	Not supported	2.5.0	1.7.1	1.7.1	1.3.0	Not supported	This plug-in is used to support features such as the automatic sharding, efficient writes, retrieval, and near real-time aggregation of time series data.
tsearch2	Not supported	Not supported	Not supported	Not supported	Not supported	1.0	This plug-in provides a text search feature that is compatible with earlier tsearch2 versions.
tsm_system_rows	1.0	1.0	1.0	1.0	1.0	Not supported	This plug-in provides a table sampling method called SYSTEM_ROWS.
tsm_system_time	1.0	1.0	1.0	1.0	1.0	Not supported	This plug-in provides a table sampling method called SYSTEM_TIME.
unaccent	1.1	1.1	1.1	1.1	1.1	1.0	This plug-in provides a text search dictionary that is used to remove accent marks or diacritic signs from lexemes.

Plug-in	14	13	12	11	10	9.4	Description
uuid-ossp	1.1	1.1	1.1	1.1	1.1	1.0	This plug-in provides functions that use a standard algorithm to generate UUIDs.
varbitx	Not supported	Not supported	Not supported	1.0	1.0	1.0	This plug-in is used to support various BIT-type operations.
wal2json	2.3	2.3	2.3	2.2	2.2	Not supported	This plug-in is used to export logical log records as a file in the JSON format.
xml2	1.1	1.1	1.1	1.1	1.1	1.0	This plug-in provides XPath query and XSLT functionality.
zhparser	Not supported	1.0	1.0	1.0	1.0	1.0	This plug-in is used to support full-text searches in Chinese.
zombodb	Not supported	Not supported	Not supported	4.0	Not supported	Not supported	This plug-in provides text indexing and analytics features.

6.2. Query vertical industry-specific data

6.2.1. Use the TimescaleDB plug-in

This topic describes how to use the TimescaleDB plug-in on an ApsaraDB RDS for PostgreSQL instance. The TimescaleDB plug-in supports the automatic sharding, efficient writes, retrieval, and near real-time aggregation of time series data.

ApsaraDB RDS for PostgreSQL supports only the open source TimescaleDB plug-in and may not support specific advanced features due to license issues. For more information, see TimescaleDB.

Prerequisites

- Your RDS instance runs PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13.

> **Note** If your RDS instance runs PostgreSQL 13, the minor engine version of your RDS instance must be 20211130 or a later version. For more information about how to update the minor engine version, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

- Add timescaledb to the value of the shared_preload_libraries parameter.

  You can add **timescaledb** to the value of the **shared_preload_libraries** parameter in the ApsaraDB RDS console or by using the ApsaraDB RDS API. For more information, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

> **Note** If the TimescaleDB plug-in has been created, an error message similar to the following message may appear after you update the minor engine version of your RDS instance:
>
> ```
> ERROR:  could not access file "$libdir/timescaledb-1.3.0": No such file or directory
> ```
>
> To resolve the error, you must execute the following SQL statement on your RDS instance to update the TimescaleDB plug-in:
>
> ```
> alter extension timescaledb update;
> ```

Create the TimescaleDB plug-in

Use pgAdmin to connect to your RDS instance. For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance. Then, execute the following statement to create the TimescaleDB plug-in:

```
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
```

Create a hypertable

1. Create a standard table named conditions. Example:

   ```
   CREATE TABLE conditions (
     time        TIMESTAMPTZ       NOT NULL,
     location    TEXT              NOT NULL,
     temperature DOUBLE PRECISION  NULL,
     humidity    DOUBLE PRECISION  NULL
   );
   ```

2. Create a hypertable. Example:

   ```
   SELECT create_hypertable('conditions', 'time');
   ```

> **Note** For more information, see Create a Hypertable.

Insert data into a hypertable

You can execute standard SQL statements to insert data into a hypertable. Example:

```
INSERT INTO conditions(time, location, temperature, humidity)
  VALUES (NOW(), 'office', 70.0, 50.0);
```

You can also insert multiple rows of data into a hypertable at a time. Example:

```
INSERT INTO conditions
  VALUES
    (NOW(), 'office', 70.0, 50.0),
    (NOW(), 'basement', 66.5, 60.0),
    (NOW(), 'garage', 77.0, 65.2);
```

Retrieve data

You can run advanced SQL queries to retrieve data. Example:

```
--Collect data from the most recent 3 hours at a 15-minute interval and sort the data by time
and temperature.
SELECT time_bucket('15 minutes', time) AS fifteen_min,
    location, COUNT(*),
    MAX(temperature) AS max_temp,
    MAX(humidity) AS max_hum
  FROM conditions
  WHERE time > NOW() - interval '3 hours'
  GROUP BY fifteen_min, location
  ORDER BY fifteen_min DESC, max_temp DESC;
```

You can also use built-in functions to analyze and query data. Examples:

```
--Query the median.
SELECT percentile_cont(0.5)
  WITHIN GROUP (ORDER BY temperature)
  FROM conditions;
```

```
--Query the moving average.
SELECT time, AVG(temperature) OVER(ORDER BY time
      ROWS BETWEEN 9 PRECEDING AND CURRENT ROW)
    AS smooth_temp
  FROM conditions
  WHERE location = 'garage' and time > NOW() - interval '1 day'
  ORDER BY time DESC;
```

6.2.2. Use the smlar plug-in

This topic describes the smlar plug-in. This allows you to calculate the similarity between two arrays of the same data type.

Prerequisites

The instance runs one of the following PostgreSQL versions:

- PostgreSQL 14
- PostgreSQL 13
- PostgreSQL 12 (kernel version 20200421 and later)
- PostgreSQL 11 (kernel version 20200402 and later)

Context

The smlar plug-in provides multiple functions to calculate the similarity between two arrays of the same data type. It also provides parameters to control the similarity calculation methods. All built-in data types are supported.

Function description

- float4 smlar(anyarray, anyarray)

  Calculates the similarity between two arrays of the same data type.

- float4 smlar(anyarray, anyarray, bool useIntersect)

  Calculates the similarity between two arrays of composite data types. The composite data type is defined as follows:

  ```
  CREATE TYPE type_name AS (element_name anytype, weight_name FLOAT4);
  ```

  When the useIntersect parameter is set to true, only the parts that contain duplicate elements are calculated. When the useIntersect parameter is set to false, all elements are calculated.

- float4 smlar(anyarray a, anyarray b, text formula)

  Calculates the similarity between two arrays of the same data type. The arrays are specified by the formula parameter.

  The predefined variables for formula are described as follows:

  - N.i: The number of common elements in the two arrays.
  - N.a: The number of distinct elements in array a.
  - N.b: The number of distinct elements in array b.

- float4 set_smlar_limit(float4)

  Sets the smlar.threshold parameter.

- float4 show_smlar_limit()

  Displays the smlar.threshold parameter value.

- anyarray % anyarray

  Returns true if the similarity between arrays is greater than the smlar.threshold parameter value. Otherwise, returns false.

- text[] tsvector2textarray(tsvector)

  Converts the tsvector type to the text type.

- anyarray array_unique(anyarray)

  Sorts the elements (excluding duplicate elements) in an array.

- float4 inarray(anyarray, anyelement)

  Returns 1 if the anyelement parameter value exists in the anyarray parameter value. otherwise, returns 0.

- float4 inarray(anyarray, anyelement, float4, float4)

  Returns the third parameter value if anyelement exists in anyarray. Otherwise, returns the fourth parameter value.

For more information about parameter descriptions and supported data types, visit smlar.

Use smlar

- After you have connected to an instance, execute the following statement to create a smlar plug-in:

```
testdb=> create extension smlar;
```

- Execute the following statements to use basic functions of smlar:

```
testdb=> SELECT smlar('{1,4,6}'::int[], '{5,4,6}' );
  smlar
----------
 0.666667
(1 row)
testdb=> SELECT smlar('{1,4,6}'::int[], '{5,4,6}', 'N.i / sqrt(N.a * N.b)' );
  smlar
----------
 0.666667
(1 row)
```

- Execute the following statement to remove smlar:

```
testdb=> drop extension smlar;
```

6.2.3. Use the PASE plug-in for efficient vector search

This topic describes how to use the PostgreSQL ANN search extension (PASE) plug-in to search for vectors in ApsaraDB RDS for PostgreSQL.

Prerequisites

Your RDS instance runs PostgreSQL 11.

Background information

Representation learning is a typical artificial intelligence (AI) technology in the deep learning discipline. This technology has rapidly developed over the recent years and is used in various business scenarios such as advertising, face scan payment, image recognition, and speech recognition. This technology enables data to be embedded into high-dimensional vectors and allows you to query data by using the vector search approach.

PASE is a high-performance vector search index plug-in that is developed for PostgreSQL. PASE uses two well-developed, stable, and efficient approximate nearest neighbor (ANN) search algorithms, IVFFlat and Hierarchical Navigable Small World (HNSW), to query vectors from PostgreSQL databases at high speeds. PASE does not support the extraction or output of feature vectors. You must retrieve the feature vectors of the entities that you want to query. PASE only implements a similarity search among a large number of vectors that are identified based on the retrieved feature vectors.

Intended audience

This topic does not explain the terms related to machine learning in detail. Before you read this topic, you must understand the basics of machine learning and search technologies.

Precautions

- If an index on a table is bloated, you can obtain the size of the indexed data by executing the `select pg_relation_size('Index name');` statement. Then, compare the size of the index with the size of the data in the table. If the size of the indexed data is larger than the size of the data in the table and data queries slow down, you must rebuild the index on the table.
- An index on a table may be inaccurate after frequent data updates. If you require an accuracy level of 100%, you must rebuild the index on a regular basis.
- If you want to create an IVFFlat index on a table by using internal centroids, you must set the clustering_type parameter to 1 and insert some data into the table.

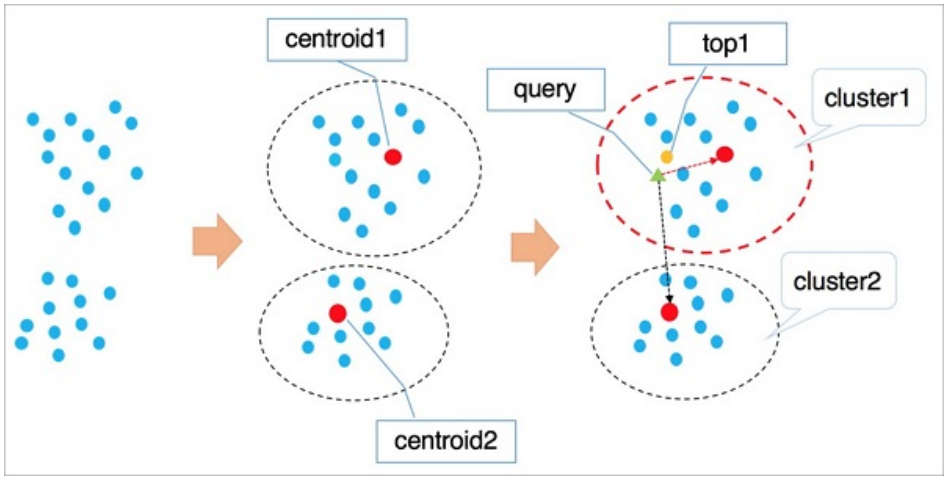
Algorithms used by PASE

- IVFFlat

  IVFFlat is a simplified version of the IVFADC algorithm. IVFFlat is suitable for business scenarios that require high precision but can tolerate up to 100 milliseconds taken for queries. IVFFlat has the following advantages compared with other algorithms:

  - If the vector to query is one of the candidate datasets, IVFFlat delivers 100% recall.
  - IVFFlat uses a simple structure to create indexes at fast speeds, and the created indexes occupy less storage space.
  - You can specify a centroid for clustering and can control precision by reconfiguring parameters.
  - You can control the accuracy of IVFFlat by reconfiguring its interpretable parameters.

  The following figure shows how IVFFlat works.



  The following procedure describes how IVFFlat works:

  i. IVFFlat uses a clustering algorithm such as k-means to divide vectors in the high-dimensional data space into clusters based on implicit clustering properties. Each cluster has a centroid.

  ii. IVFFlat traverses the centroids of all clusters to identify the n centroids that are nearest to the vector you want to query.

  iii. IVFFlat traverses and sorts all vectors in the clusters to which the identified n centroids belong. Then, IVFFlat obtains the nearest k vectors.

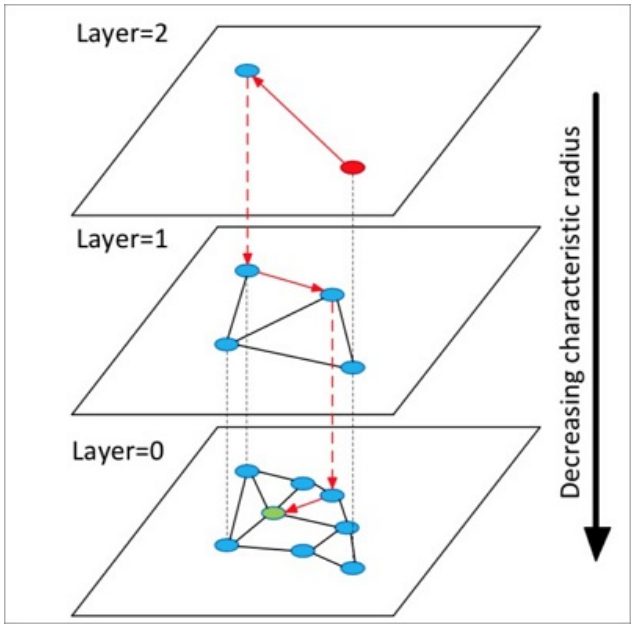
> **Note**
>
> - When IVFFlat attempts to identify the nearest n centroids, it skips the clusters that are located far away from the vector you want to query. This expedites the query. However, IVFFlat cannot ensure that all the similar k vectors are included in the clusters to which the identified n centroids belong. As a result, precision may decrease. You can use the variable n to control precision. A larger value of n indicates higher precision but more computing workloads.
> - In the first phase, IVFFlat works in the same way as IVFADC. The main differences between IVFFlat and IVFADC lie in the second phase. In the second phase, IVFADC uses product quantization to eliminate the need for traversal computing workloads. This expedites the query but decreases precision. IVFFlat implements brute-force search to ensure precision and allows you to control computing workloads.

- HNSW

  HNSW is a graph-based ANN algorithm that is suitable for queries among tens of millions or more vector datasets. Responses to these queries must be returned within 10 milliseconds or less.

  HNSW searches for similar vectors among proximity graph neighbors. If the data volume is large, HNSW significantly improves performance compared with other algorithms. However, HNSW requires the storage of proximity graph neighbors, which occupy storage space. In addition, after the precision of HNSW reaches a specific level, you cannot increase the precision of HNSW by reconfiguring parameters.

  The following figure shows how HNSW works.

  The following procedure describes how HNSW works:

  i. HNSW builds a hierarchical structure that consists of multiple layers, which are also called graphs. Each layer is a panorama and skips list of its lower layer.

  ii. HNSW randomly selects an element from the top layer to start a search.

  iii. HNSW identifies the neighbors of the selected element and adds the identified neighbors to a fixed-length dynamic list based on the distances of the identified neighbors to the selected element. HNSW continues to identify the neighbors of each neighbor that is included in the list and adds the identified neighbors to the list. Every time when HNSW adds a neighbor to the list, HNSW

re-sorts the neighbors in the list and retains only the first k neighbors. If the list changes, HNSW continues to search until the list reaches the final state. Then, HNSW uses the first element in the list as the start for a search in the lower layer.

iv. HNSW repeats the third step until it complete a search in the bottom layer.

> **Note** HNSW constructs a multi-layer structure by using the Navigable Small World (NSW) algorithm that is designed to construct single-layer structures. The employment of an approach for selecting proximity graph neighbors enables HNSW to deliver higher query speedup than clustering algorithms.

IVFFlat and HNSW each are suitable for specific business scenarios. For example, IVFFlat is suitable for image comparison at high precision, and HNSW is suitable for searches with recommended recall. More industry-leading algorithms will be integrated into PASE.

Procedure

1. Execute the following statement to enable the PASE plug-in:

```
CREATE EXTENSION pase;
```

2. Use one of the following construction methods to calculate vector similarity:
   - PASE-type-based construction

     Example:

```
SELECT ARRAY[2, 1, 1]::float4[] <?> pase(ARRAY[3, 1, 1]::float4[]) AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <?> pase(ARRAY[3, 1, 1]::float4[], 0) AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <?> pase(ARRAY[3, 1, 1]::float4[], 0, 1) AS distance;
```

     > **Note**
     > - <?> is a PASE-type operator, which is used to calculate the similarity between the vectors to the left and right of a specific element. The vector to the left must use the float4[] data type, and the vector to the right must use the PASE data type.
     > - The PASE data type is defined in the PASE plug-in and can contain up to three constructors. Take the `float4[], 0, 1` part in the preceding third as an example. The first parameter specifies the vector to the right with the float4[] data type. The second parameter does not serve a special purpose and can be set to 0. The third parameter specifies the similarity calculation method, where the value 0 represents the Euclidean distance method and the value 1 represents the dot product method. A dot product is also called an inner product.
     > - The vector to the left must have the same number of dimensions as the vector to the right. Otherwise, the system reports similarity calculation errors.

   - String-based construction

     Example:

```
SELECT ARRAY[2, 1, 1]::float4[] <?> '3,1,1'::pase AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <?> '3,1,1:0'::pase AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <?> '3,1,1:0:1'::pase AS distance;
```

> **Note** The string-based construction method differs from the PASE-type-based construction in the following aspect: The string-based construction method uses colons (:) to separate parameters. Take the `3,1,1:0:1` part in the preceding third statement as an example: The first parameter specifies the vector to the right. The second parameter does not serve a special purpose and can be set to 0. The third parameter specifies the similarity calculation method, where the value 0 represents the Euclidean distance method and the value 1 represents the dot product method. A dot product is also called an inner product.

3. Use IVFFlat or HNSW to create an index.

> **Note** If you use the Euclidean distance method to calculate vector similarity, the original vector does not need to be processed. If you use the dot product or the cosine method to calculate vector similarity, the original vector must be normalized. For example, if the original vector is $x_1, x_2, x_3, \ldots, x_n$, it must comply with the following formula:
>
> $x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2 = 1$. In this example, the dot product is the same as the cosine value.

- IVFFlat

  Example:

```
CREATE INDEX ivfflat_idx ON vectors_table
USING
  pase_ivfflat(vector)
WITH
  (clustering_type = 1, distance_type = 0, dimension = 256, base64_encoded = 0, cluster
ing_params = "10,100");
```

  The following table describes the parameters in the IVFFlat index.

Parameter	Description
clustering_type	The type of clustering operation that IVFFlat performs on vectors. This parameter is required. Valid values: ■ 0: external clustering. An external centroid file is loaded. This file is specified by the **clustering_params** parameter. ■ 1: internal clustering. The k-means clustering algorithm is used. This algorithm is specified by the **clustering_params** parameter. If you are using PASE for the first time, we recommend that you select internal clustering.

Parameter	Description
distance_type	The method that is used to calculate vector similarity. Default value: 0. Valid values: ■ 0: Euclidean distance. ■ 1: dot product. This method requires the normalization of vectors. The order of dot products is opposite to the order of Euclidean distances. ApsaraDB RDS for PostgreSQL supports only the Euclidean distance method. Dot products can be calculated only after vectors are normalized. For more information, see Appendix.
dimension	The number of dimensions. This parameter is required. The maximum value of this parameter is 512.
base64_encoded	Specifies whether to use Base64 encoding. Default value: 0. Valid value: ■ 0: uses the float4[] data type to represent the vector type. ■ 1: uses the Base64-encoded float[] data type to represent the vector type.
clustering_params	For external clustering, this parameter specifies the directory of the external centroid file that you want to use. For internal clustering, this parameter specifies the clustering algorithm that you want to use. The value of this parameter is in the following format `clustering_sample_ratio,k`. This parameter is required. ■ clustering_sample_ratio: the sampling fraction with 1000 as the denominator. The value of this field is an integer within the (0, 1000] range. For example, if you set this field to 1, the system samples data from the dynamic list based on the 1/1000 sampling ratio before it performs k-means clustering. A larger value indicates higher query accuracy but slower index creation. We recommend that the total number of data records sampled does not exceed 100,000. ■ k: the number of centroids. A larger value indicates higher query accuracy but slower index creation. We recommend that you set this field to a value within the [100, 1000] range.

- HNSW

  Example:

```
CREATE INDEX hnsw_idx ON vectors_table
USING
  pase_hnsw(vector)
WITH
  (dim = 256, base_nb_num = 16, ef_build = 40, ef_search = 200, base64_encoded = 0);
```

The following table describes the parameters in the HNSW index.

Parameter	Description
dim	The number of dimensions. This parameter is required. The maximum value of this parameter is 512.
base_nb_num	The number of neighbors that you want to identify for an element. This parameter is required. A larger value indicates higher query accuracy, but slower index creation and more storage space occupied. We recommend that you set this parameter to a value within the [16, 128] range.
ef_build	The heap length that you want to use during index creation. This parameter is required. A longer heap length indicates higher query accuracy but slower index creation. We recommend that you set this parameter to a value within the [40, 400] range.
ef_search	The heap length that you want to use during a query. This parameter is required. A longer heap length indicates higher query accuracy but lower query performance. You can specify this parameter when you initiate a query request. Default value: 200.
base64_encoded	Specifies whether to use Base64 encoding. Default value: 0. Valid values: ■ 0: uses the float4[] data type to represent the vector type. ■ 1: uses the Base64-encoded float[] data type to represent the vector type.

4. Use one of the following indexes to query a vector:
   - IVFFlat index

     Example:

     ```
     SELECT id, vector <#> '1,1,1'::pase as distance
     FROM vectors_ivfflat
     ORDER BY
     vector <#> '1,1,1:10:0'::pase
     ASC LIMIT 10;
     ```

     > **Note**
     >
     > ■ <#> is an operator that is used by IVFFlat indexes.
     >
     > ■ You must execute the ORDER BY statement to make an IVFFlat index take effect. An IVFFlat index allows vectors to be sorted in ascending order.
     >
     > ■ The PASE data type requires three parameters to specify a vector. These parameters are separated by colons (:). For example, `1,1,1:10:0` includes three parameters: The first parameter specifies the vector to query. The second parameter specifies the query efficiency of IVFFlat with a value range of (0, 1000], in which a larger value indicates higher query accuracy but lower query performance. The third parameter specifies the vector similarity calculation method, where the value 0 represents the Euclidean distance method and the value 1 represents the dot product method. A dot product is also called an inner product. The dot product method requires the normalization of vectors. The order of dot products is opposite to the order of Euclidean distances.

   - HNSW index

Example:

```
SELECT id, vector <?> '1,1,1'::pase as distance
FROM vectors_ivfflat
ORDER BY
vector <?> '1,1,1:100:0'::pase
ASC LIMIT 10;
```

> **Note**
>
> - <?> is an operator that is used by the HNSW index.
> - You must execute the ORDER BY statement to make an HNSW index take effect. An HNSW index allows vectors to be sorted in ascending order.
> - The PASE data type requires three parameters to specify a vector. These parameters are separated by colons (:). For example, `1,1,1:10:0` includes three parameters: The first parameter specifies the vector to query. The second parameter specifies the query efficiency of HNSW with a value range of (0, ∞), in which a larger value indicates higher query accuracy but lower query performance. We recommend that you set the second parameter to 40 and then test the value by small increases until you find the most suitable value for your business. The third parameter specifies the vector similarity calculation method, where the value 0 represents the Euclidean distance method and the value 1 represents the dot product method. A dot product is also called an inner product. The dot product method requires the normalization of vectors. The order of dot products is opposite to the order of Euclidean distances.

Appendix

- Calculate the dot product of a vector.

  For this example, use an HNSW index to create a function:

```
CREATE OR REPLACE FUNCTION inner_product_search(query_vector text, ef integer, k integer, t
able_name text) RETURNS TABLE (id integer, uid text, distance float4) AS $$
BEGIN
    RETURN QUERY EXECUTE format('
    select a.id, a.vector <?> pase(ARRAY[%s], %s, 1) AS distance from
    (SELECT id, vector FROM %s ORDER BY vector <?> pase(ARRAY[%s], %s, 0) ASC LIMIT %s) a
    ORDER BY distance DESC;', query_vector, ef,  table_name,  query_vector, ef, k);
END
$$
LANGUAGE plpgsql;
```

  > **Note** The dot product of a normalized vector is the same as its cosine value. Therefore, you can also follow this example to calculate the cosine value of a vector.

- Create an IVFFlat index from an external centroid file.

  This is an advanced feature. You must upload an external centroid file to the specified directory of the server and use this file to create an IVFFlat index. For more information, see the parameters of the IVFFlat index. This file is in the following format:

```
Number of dimensions|Number of centroids|Centroid vector dataset
```

Example:

```
3|2|1,1,1,2,2,2
```

References

- Product Quantization for Nearest Neighbor Search

  Herv´e J´egou, Matthijs Douze, Cordelia Schmid. Product quantization for nearest neighbor search.

- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs

  Yu.A.Malkov, D.A.Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs.

6.2.4. Use the roaringbitmap plug-in

This topic describes how to use the roaringbitmap plug-in provided by ApsaraDB RDS for PostgreSQL to improve query performance.

Prerequisites

Your RDS instance runs PostgreSQL 12, 13 or 14.

Context

The Roaring bitmap algorithm divides 32-bit integers into 2^{16} chunks. Each chunk stores the 16 most significant digits and uses a container to store the 16 least significant digits. A Roaring bitmap stores containers in a dynamic array as primary indexes. Two types of containers are available: array containers for sparse chunks and bitmap containers for dense chunks. An array container can store up to 4,096 integers. A bitmap container can store more than 4,096 integers.

Roaring bitmaps can use this storage structure to rapidly retrieve specific values. Additionally, Roaring bitmaps provide bitwise operations such as AND, OR, and XOR between the two types of containers. Therefore, Roaring bitmaps can deliver excellent storage and computing performance.

Procedure

1. Create a plug-in. Example:

   ```
   CREATE EXTENSION roaringbitmap;
   ```

2. Create a table with roaringbitmap data. Example:

   ```
   CREATE TABLE t1 (id integer, bitmap roaringbitmap);
   ```

3. Call the rb_build function to insert roaringbitmap data. Example:

   ```
   -- Set the bit value of an array to 1. INSERT INTO t1 SELECT 1,RB_BUILD(ARRAY[1,2,3,4,5,6
   ,7,8,9,200]);
   -- Set the bit values of multiple elements to 1 and aggregate the bit values into a Roari
   ng bitmap. INSERT INTO t1 SELECT 2,RB_BUILD_AGG(e) FROM GENERATE_SERIES(1,100) e;
   ```

4. Perform bitwise operations such as OR, AND, XOR, and ANDNOT. Example:

   ```
   SELECT RB_OR(a.bitmap,b.bitmap) FROM (SELECT bitmap FROM t1 WHERE id = 1) AS a,(SELECT bi
   tmap FROM t1 WHERE id = 2) AS b;
   ```

5. Perform bitwise aggregate operations such as OR, AND, XOR, and BUILD to generate a new Roaring bitmap. Example:

```
SELECT RB_OR_AGG(bitmap) FROM t1;
SELECT RB_AND_AGG(bitmap) FROM t1;
SELECT RB_XOR_AGG(bitmap) FROM t1;
SELECT RB_BUILD_AGG(e) FROM GENERATE_SERIES(1,100) e;
```

6. Calculate the cardinality of the Roaring bitmap. The cardinality is the number of bits that are set to 1 in the Roaring bitmap. Example:

```
SELECT RB_CARDINALITY(bitmap) FROM t1;
```

7. Obtain the subscripts of the bits that are set to 1. Example:

```
SELECT RB_ITERATE(bitmap) FROM t1 WHERE id = 1;
```

Bitmap calculation functions

Function	Input	Output	Description	Example
rb_build	integer[]	roaringbitmap	Creates a Roaring bitmap from an integer array.	`rb_build('{1,2,3,4,5}')`
rb_and	roaringbitmap,roaringbitmap	roaringbitmap	Performs an AND operation.	`rb_and(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_or	roaringbitmap,roaringbitmap	roaringbitmap	Performs an OR operation.	`rb_or(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_xor	roaringbitmap,roaringbitmap	roaringbitmap	Performs an XOR operation.	`rb_xor(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_andnot	roaringbitmap,roaringbitmap	roaringbitmap	Performs an ANDNOT operation.	`rb_andnot(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`

Function	Input	Output	Description	Example
rb_cardinality	roaringbitmap	integer	Calculates the cardinality.	`rb_cardinality(rb_build('{1,2,3,4,5}'))`
rb_and_cardinality	roaringbitmap,roaringbitmap	integer	Calculates the cardinality from an AND operation on two Roaring bitmaps.	`rb_and_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_or_cardinality	roaringbitmap,roaringbitmap	integer	Calculates the cardinality from an OR operation on two Roaring bitmaps.	`rb_or_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_xor_cardinality	roaringbitmap,roaringbitmap	integer	Calculates the cardinality from an XOR operation on two Roaring bitmaps.	`rb_xor_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_andnot_cardinality	roaringbitmap,roaringbitmap	integer	Calculates the cardinality from an ANDNOT operation on two Roaring bitmaps.	`rb_andnot_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_is_empty	roaringbitmap	boolean	Checks whether a Roaring bitmap is empty.	`rb_is_empty(rb_build('{1,2,3,4,5}'))`

Function	Input	Output	Description	Example
rb_equals	roaringbitmap,roaringbitmap	boolean	Checks whether two Roaring bitmaps are the same.	`rb_equals(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_intersect	roaringbitmap,roaringbitmap	boolean	Checks whether two Roaring bitmaps intersect.	`rb_intersect(rb_build('{1,2,3}'),rb_build('{3,4,5}'))`
rb_remove	roaringbitmap,integer	roaringbitmap	Removes an offset from a Roaring bitmap.	`rb_remove(rb_build('{1,2,3}'),3)`
rb_flip	roaringbitmap,integer,integer	roaringbitmap	Flips specific offsets in a Roaring bitmap.	`rb_flip(rb_build('{1,2,3}'),2,3)`
rb_minimum	roaringbitmap	integer	Returns the smallest offset in a Roaring bitmap. If the Roaring bitmap is empty, the value -1 is returned.	`rb_minimum(rb_build('{1,2,3}'))`
rb_maximum	roaringbitmap	integer	Returns the largest offset in a Roaring bitmap. If the Roaring bitmap is empty, the value 0 is returned.	`rb_maximum(rb_build('{1,2,3}'))`
rb_rank	roaringbitmap,integer	integer	Returns the number of elements that are smaller than or equal to a specified offset in a Roaring bitmap.	`rb_rank(rb_build('{1,2,3}'),3)`

Function	Input	Output	Description	Example
rb_iterate	roaringbitmap	setof integer	Returns a list of offsets from a Roaring bitmap.	`rb_iterate(rb_build('{1,2,3}'))`

Bitmap aggregate functions

Function	Input	Output	Description	Example
rb_build_agg	integer	roaringbitmap	Creates a Roaring bitmap from a group of offsets.	`rb_build_agg(1)`
rb_or_agg	roaringbitmap	roaringbitmap	Performs an OR aggregate operation.	`rb_or_agg(rb_build('{1,2,3}'))`
rb_and_agg	roaringbitmap	roaringbitmap	Performs an AND aggregate operation.	`rb_and_agg(rb_build('{1,2,3}'))`
rb_xor_agg	roaringbitmap	roaringbitmap	Performs an XOR aggregate operation.	`rb_xor_agg(rb_build('{1,2,3}'))`
rb_or_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an OR aggregate operation on two Roaring bitmaps.	`rb_or_cardinality_agg(rb_build('{1,2,3}'))`
rb_and_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an AND aggregate operation on two Roaring bitmaps.	`rb_and_cardinality_agg(rb_build('{1,2,3}'))`

Function	Input	Output	Description	Example
rb_xor_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an XOR aggregate operation on two Roaring bitmaps.	`rb_xor_cardinality_agg(rb_build('{1,2,3}'))`

6.2.5. Use the varbitx plug-in

The varbit plug-in that is provided in the PostgreSQL Community edition supports only simple BIT-type operation functions. The varbitx plug-in is an extension of the varbit plug-in. The varbitx plug-in is provided in ApsaraDB RDS for PostgreSQL to support more BIT-type operations in more scenarios. These scenarios include real-time user profile recommendation, access control advertising, and ticketing.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 11
- PostgreSQL 10

Functions supported

Function	Description
`get_bit (varbit a, int b, int c) returns varbit`	Obtains a specified number c of bits that start at position b and returns a VARBIT-type string. For example, `get_bit('111110000011', 3, 5)` returns 11000.
`set_bit_array (varbit a, int b, int c, int[] d) returns varbit`	Changes the values of the bits that are specified by the subscript array d to values b (0 or 1), and fills the bits that exceed the original length with values c (0 or 1). For example, `set_bit_array('111100001111', 0, 1, array[1,15])` returns 1011000011111110.
`bit_count (varbit a, int b, int c, int d) returns int`	Counts the number of values b (0 or 1) that start at position c among a specified number d of bits. The bits that are beyond the specified length are not counted. For example, `bit_count('1111000011110000', 1, 5, 4)` returns 1.
`bit_count (varbit a, int b) returns int`	Counts the total number of values b (0 or 1). For example, `bit_count('1111000011110000', 1)` returns 8.

Function	Description
`bit_fill (int a, int b) returns varbit`	Fills a specified number b of bits with values a (0 or 1). For example, `bit_fill(0,10)` returns 0000000000.
`bit_rand (int a, int b, float c) returns varbit`	Obtains a random percentage c of bits from a specified number a of bits, and randomly fills the obtained bits with values b (0 or 1). For example, `bit_rand(10, 1, 0.3)` may return 0101000001.
`bit_posite (varbit a, int b, boolean c) returns int[]`	Returns a subscript array. The subscript array indicates the positions of the bits whose values are b (0 or 1). Subscripts start at 0. If the value of c is true, subscripts are returned in a positive sequence. If the value of c is false, subscripts are returned in a negative sequence. For example, `bit_posite ('11110010011', 1, true)` returns [0,1,2,3,6,9,10], and `bit_posite ('11110010011', 1, false)` returns [10,9,6,3,2,1,0].
`bit_posite (varbit a, int b, int c, boolean d) returns int[]`	Returns a subscript array. The subscript array indicates the positions of the bits whose values are b (0 or 1). The number of subscripts in the subscript array is c. Subscripts start at 0. If the value of d is true, subscripts are returned in a positive sequence. If the value of d is false, subscripts are returned in a negative sequence. For example, `bit_posite ('11110010011', 1, 3, true)` returns [0,1,2], and `bit_posite ('11110010011', 1, 3, false)` returns [10,9,6].
`get_bit_array (varbit a, int b, int c, int d) returns int[]`	Obtains a specified number c of bits that start at position b, identifies the bits whose values are d (0 or 1) among the obtained bits, and returns a subscript array. The subscript array indicates the positions of the identified bits. For example, `get_bit_array('111110000011', 3, 5, 1)` returns [3,4] for the specified 11000 bit string.
`get_bit_array (varbit a, int b, int[] c) returns int[]`	Obtains the bits that are specified by a subscript array c, identifies the bits whose values are b (0 or 1) among the obtained bits, and returns a subscript array. The subscript array indicates the positions of the identified bits. The bits that are not included in the subscript array c are not counted. For example, `get_bit_array('111110000011', 1, array[1,5,6,7,10,11])` returns [1,10,11].

Function	Description
`set_bit_array (varbit a, int b, int c, int[] d, int e) returns varbit`	Changes the values of the bits that are specified by a subscript array d to values b (0 or 1), and fills the bits that are beyond the original length with values c (0 or 1). The number of bits that are returned is e. For example, `set_bit_array('111100001111', 1, 0, array[4,5,6,7], 2)` returns 111111001111.
`set_bit_array_record (varbit a, int b, int c, int[] d) returns (varbit,int[])`	Changes the values of the bits that are specified by a subscript array d to values b (0 or 1), and fills the bits that are beyond the original length with values c (0 or 1). This function not only returns a bit string but also a subscript array. The subscript array indicates the positions of the bits whose values are changed. For example, `set_bit_array_record('111100001111', 0, 1, array[1,15])` returns 1011000011111110 and [1,15].
`set_bit_array_record (varbit a, int b, int c, int[] d, int e) returns (varbit,int[])`	Changes the values of the bits that are specified by a subscript array d to values b (0 or 1), and fills the bits that are beyond the original length with values c (0 or 1). This function not only returns a bit string but also a subscript array. The subscript array indicates the positions of the bits whose values are changed. This function returns results immediately after it changes values for a total of e bits. For example, `set_bit_array_record('111100001111', 1, 0, array[1,4,5,6,7], 2)` returns 111111001111 and [4,5].
`bit_count_array (varbit a, int b, int[] c) returns int`	Counts the number of values b (0 or 1) among the bits that are specified by a subscript array c. For example, `bit_count_array('1111000011110000', 1, array[1,2,7,8])` returns 3.

Basic usage

- Create the varbitx plug-in.

```
CREATE EXTENSION varbitx;
```

- Delete the varbitx plug-in.

```
DROP EXTENSION varbitx;
```

- Call the functions that are supported by the varbitx plug-in.

  You can execute the `SELECT <function>` statement to call the functions.

- Execute the following statement to call the bit_count function:

```
select bit_count('1111000011110000', 1, 5, 4);
```

The following result is returned:

```
 bit_count
-----------
         1
(1 row)
```

- Execute the following statement to call the set_bit_array_record function:

```
select set_bit_array_record('111100001111', 1, 0, array[1,4,5,6,7], 2);
```

The following result is returned:

```
  set_bit_array_record
------------------------
 (111111001111,"{4,5}")
(1 row)
```

For more information about the functions and their descriptions, see Functions supported.

6.2.6. Use the RDKit plug-in

This topic describes how to use the RDKit plug-in of ApsaraDB RDS for PostgreSQL to implement functions such as molecular computing and search.

Prerequisites

Your RDS instance runs PostgreSQL 12.

Context

RDKit supports two data types: the mol data type that is used to describe molecular types, and the fp data type that is used to describe molecular fingerprints. It allows for comparison computing, similarity computing based on the Tanimoto and Dice coefficients, and GiST indexing.

For more information about the SQL statements that are supported by RDKit, visit RDKit SQL.

Precautions

- Input and output functions based on the mol data type comply with the simplified molecular input line entry specification (SMILES).
- Input and output functions based on the fp data type comply with the bytea format that is used to store binary data.

Create the RDKit plug-in

```
postgres=# create extension rdkit ;
CREATE EXTENSION
```

Default parameter settings

```
postgres=# show rdkit.tanimoto_threshold ;
 rdkit.tanimoto_threshold
--------------------------
 0.5
(1 row)
postgres=# show rdkit.dice_threshold;
 rdkit.dice_threshold
----------------------
 0.5
(1 row)
```

Indexes supported

- B-tree and hash indexes are supported for comparison computing operations that are based on the mol and fp data types. Examples:

```
CREATE INDEX molidx ON pgmol (mol);
CREATE INDEX molidx ON pgmol (fp);
```

- GiST indexes are supported for the following operations that are based on the mol and fp data types: "mol % mol", "mol # mol", "mol @> mol", "mol <@ mol", "fp % fp", and "fp # fp." Example:

```
CREATE INDEX molidx ON pgmol USING gist (mol);
```

Sample functions

- The tanimoto_sml function calculates the degree of similarity based on the Tanimoto coefficient.

```
postgres=# \df tanimoto_sml
                            List of functions
 Schema |     Name     | Result data type | Argument data types | Type
--------+--------------+------------------+---------------------+------
 public | tanimoto_sml | double precision | bfp, bfp            | func
 public | tanimoto_sml | double precision | sfp, sfp            | func
(2 rows)
```

- The dice_sml function calculates the degree of similarity based on the Dice coefficient.

```
postgres=# \df dice_sml
                          List of functions
 Schema |   Name    | Result data type | Argument data types | Type
--------+-----------+------------------+---------------------+------
 public | dice_sml | double precision | bfp, bfp            | func
 public | dice_sml | double precision | sfp, sfp            | func
(2 rows)
```

- If the second argument is a substructure of the first argument, the substruct function returns the TRUE value.

```
postgres=# \df substruct
                          List of functions
 Schema |   Name    | Result data type | Argument data types | Type
--------+-----------+------------------+---------------------+------
 public | substruct | boolean          | mol, mol            | func
 public | substruct | boolean          | mol, qmol           | func
 public | substruct | boolean          | reaction, reaction  | func
(3 rows)
```

Basic operations

- `mol % mol` and `fp % fp`

  If the degree of similarity that is calculated based on the Tanimoto coefficient is less than the value of the rdkit.tanimoto_threshold GUC variable, the TRUE value is returned.

- `mol # mol` and `fp # fp`

  If the degree of similarity that is calculated based on the Dice coefficient is less than the value of the rdkit.dice_threshold GUC variable, the TRUE value is returned.

- `mol @> mol`

  If the left operand contains the right operand, the TRUE value is returned.

- `mol <@ mol`

  If the right operand contains the left operand, the TRUE value is returned.

6.3. Run cross-database queries

6.3.1. Use the oss_fdw plug-in to read and write foreign data text files

This topic describes how to import data from an Object Storage Service (OSS) bucket into an ApsaraDB RDS for PostgreSQL instance by using the oss_fdw plug-in. This topic also describes how to export data from an ApsaraDB RDS for PostgreSQL instance to an OSS bucket by using the oss_fdw plug-in.

Prerequisites

Your RDS instance runs one of the following database engine versions:

- PostgreSQL 13
- PostgreSQL 12
- PostgreSQL 11
- PostgreSQL 10
- PostgreSQL 9.4

Examples

```
# Create an oss_fdw plug-in for your RDS instance.
create extension oss_fdw;
# Create an OSS server.
CREATE SERVER ossserver FOREIGN DATA WRAPPER oss_fdw OPTIONS
     (host 'oss-cn-hangzhou.aliyuncs.com' , id 'xxx', key 'xxx',bucket 'mybucket');
# Create a foreign OSS table named ossexample.
CREATE FOREIGN TABLE ossexample
    (date text, time text, open float,
     high float, low float, volume int)
     SERVER ossserver
     OPTIONS ( filepath 'osstest/example.csv', delimiter ',' ,
         format 'csv', encoding 'utf8', PARSE_ERRORS '100');
# Create a table named example on your RDS instance. The table is used to store the data that
is imported into your RDS instance.
create table example
        (date text, time text, open float,
         high float, low float, volume int);
# Import data from the ossexample table into the example table.
insert into example select * from ossexample;
# Use the oss_fdw plug-in to estimate the size of the ossexample table and formulate a query
plan.
explain insert into example select * from ossexample;
                            QUERY PLAN
------------------------------------------------------------------------
 Insert on example  (cost=0.00..1.60 rows=6 width=92)
   ->  Foreign Scan on ossexample  (cost=0.00..1.60 rows=6 width=92)
         Foreign OssFile: osstest/example.csv.0
         Foreign OssFile Size: 728
(4 rows)
# Export data from the example table to the ossexample table.
insert into ossexample select * from example;
explain insert into ossexample select * from example;
                          QUERY PLAN
---------------------------------------------------------------
 Insert on ossexample  (cost=0.00..16.60 rows=660 width=92)
   ->  Seq Scan on example  (cost=0.00..16.60 rows=660 width=92)
(2 rows)
```

For more information about the parameters in the preceding examples, see the following sections.

Parameters supported by the oss_fdw plug-in

Similar to other foreign data wrappers (FDWs), the oss_fdw plug-in encapsulates foreign data that is stored in OSS buckets. You can use the oss_fdw plug-in to read data from OSS buckets. This process is similar to the process of reading data tables. The oss_fdw plug-in provides some unique parameters that are used to access a specified OSS bucket and parse the OSS objects in the OSS bucket.

> **Note**
>
> - The oss_fdw plug-in can read and write data to OSS objects in the following formats: TEXT and CSV. These OSS objects include the TEXT objects and CSV objects that are compressed by using gzip.
> - The value of each parameter that is used by the oss_fdw plug-in must be enclosed in a pair of double quotation marks (""). In addition, the value of each parameter cannot contain unnecessary spaces.

Parameters in the CREATE SERVER statement

Parameter	Description
ossendpoint	The internal endpoint of the OSS bucket. The endpoint is also referred to as the host address.
id oss	The AccessKey ID of the account that is used to access OSS.
key oss	The AccessKey secret of the account that is used to access OSS.
bucket	The OSS bucket that stores the objects whose data you want to read or write. Before you set this parameter, you must create an account that is used to access OSS.

The following table describes the fault tolerance parameters that are provided by OSS. If network connectivity is poor, you can adjust the values of these parameters to ensure successful import and export.

Parameter	Description
oss_connect_timeout	The period of time after which the connection to OSS times out. Unit: seconds. Default value: 10.
oss_dns_cache_timeout	The period of time after which the cached Domain Name System (DNS) record times out. Unit: seconds. Default value: 60.
oss_speed_limit	The minimum transmission rate that can be tolerated. Unit: bit/s. Default value: 1024. The default value is equal to 1 Kbit/s.
oss_speed_time	The maximum period of time for which the minimum transmission rate can be tolerated. Unit: seconds. Default value: 15.

> **Note** You can retain the default values of the oss_speed_limit and oss_speed_time parameters. In this case, if the transmission rate remains less than 1 Kbit/s for 15 consecutive seconds, a time-out occurs.

Parameters in the CREATE FOREIGN TABLE statement

Parameter	Description

Parameter	Description
filepath	The object name that is used to match objects stored in the OSS bucket. The object name must contain an OSS path. • The object name does not contain an OSS bucket name. • The object name matches multiple objects that are stored in the OSS path. This allows you to import data from multiple objects into your RDS instance. • Only the data from the objects that are named in the following formats can be imported into your RDS instance: filepath and filepath.x. The values of the x variable must be consecutive integers that start from 1. For example, the OSS path stores five objects: filepath, filepath.1, filepath.2, filepath.3, and filepath.5. In this case, filepath, filepath.1, filepath.2, and filepath.3 are matched and imported, but filepath.5 cannot be matched or imported.
dir	The folder that is used to match objects stored in the OSS bucket. • The folder must end with a forward slash (/). • The data of all the objects in the folder are matched and imported into your RDS instance. However, these objects do not include the subfolders and the objects stored in the subfolders.
prefix	The prefix of the OSS path in the OSS bucket. This parameter does not support regular expressions. You can configure only one of the prefix, filepath, and dir parameters.
format	The format that is supported for the objects stored in the OSS bucket. Only the CSV format is supported.
encoding	The format that is used to encode data in the objects stored in the OSS bucket. Common encoding formats in PostgreSQL are supported. These supported formats include UTF-8.
parse_errors	The mode that is used to tolerate faults during the parsing process. If an error occurs during the parsing process, the entire row that encounters the error is ignored.
delimiter	The delimiter that is used to separate columns in the objects stored in the OSS bucket.
quote	The quote character that is supported for the objects stored in the OSS bucket.
escape	The escape character that is supported for the objects stored in the OSS bucket.
null	Populates an empty column by using null values. For example, you specify the null 'test' setting. In this case, if the test column is empty, it is populated by using null values.
force_not_null	Populates an empty column by using empty strings rather than null values. For example, you specify the `force_not_null 'id'` setting. In this case, if the ID column is empty, it is populated by using empty strings rather than null values.

Parameter	Description
compressiontype	The compression format that is used to read and write data to the objects stored in the OSS bucket. • none: The data is not compressed. This is the default value. • gzip: The data is compressed in the GZIP format.
compressionlevel	The compression level that is used to write data to the objects stored in the OSS bucket. Valid values: 1 to 9. Default value: 6.

Note

- The filepath and dir parameters are specified in the OPTIONS parameter.
- You must specify the filepath parameter or the dir parameter. Do not specify both parameters.
- If you export data from your RDS instance to the OSS bucket, you can specify only the dir parameter. You cannot specify the filepath parameter.

Parameters in the CREATE FOREIGN TABLE statement

- oss_flush_block_size: the buffer size of the data that can be written to the OSS bucket at a time. Valid values: 1 to 128. Unit: MB. Default value: 32.
- oss_file_max_size: the maximum amount of data that can be written to an object in the OSS bucket. If the amount of data that needs to be written reaches the maximum value, the data that remains is written to a new object. Valid values: 8 to 4000. Unit: MB. Default value: 1024.
- num_parallel_worker: the maximum number of threads that can run in parallel to compress the data written to the OSS bucket. Valid values: 1 to 8. Default value: 3.

Auxiliary functions

FUNCTION oss_fdw_list_file (relname text, schema text DEFAULT 'public')

- This function is used to obtain the name and size of the OSS object that the specified foreign table matches.
- The size is measured in bytes.

```
select * from oss_fdw_list_file('t_oss');
              name               |   size
---------------------------------+-----------
 oss_test/test.gz.1  | 739698350
 oss_test/test.gz.2  | 739413041
 oss_test/test.gz.3  | 739562048
(3 rows)
```

Auxiliary parameters

oss_fdw.rds_read_one_file: specifies the OSS object that a foreign table matches. This parameter is supported only when you import data from the OSS bucket into your RDS instance. If you specify this parameter, only the OSS object that the specified foreign table matches is imported.

Example: set oss_fdw.rds_read_one_file = 'oss_test/example16.csv.1';

```
set oss_fdw.rds_read_one_file = 'oss_test/test.gz.2';
select * from oss_fdw_list_file('t_oss');
              name              |   size
--------------------------------+-----------
  oss_test/test.gz.2  | 739413041
(1 rows)
```

Usage notes

- The oss_fdw plug-in is developed based on the PostgreSQL FOREIGN TABLE framework to manage foreign tables.
- Data import performance varies based on the available PostgreSQL and OSS resources. The PostgreSQL resources are CPU, I/O, and memory.
- To import data at high performance, make sure that your RDS instance and the OSS bucket reside in the same region. For more information, see OSS domain names.
- If the error "`ERROR: oss endpoint userendpoint not in aliyun white list`" is reported when SQL statements are read from the foreign table, we recommend that you use the public OSS endpoint that is provided for the specified region. For more information, see Regions and endpoints. If the error persists, submit a ticket.

Troubleshooting

If an import or export error occurs, the following error information is logged:

- code: the HTTP status code of the request that has failed.
- error_code: the error code that is returned by OSS.
- error_msg: the error message that is returned by OSS.
- req_id: the universally unique identifier (UUID) of the request that has failed. If you require assistance from OSS developers, you can submit a ticket that contains the req_id parameter of the failed request.

For more information about various errors, see the following documentation. You can handle time-out errors by reconfiguring the parameters related to the oss_ext plug-in.

- OSS documentation
- CREATE FOREIGN TABLE
- OSS error handling
- OSS error responses

Encryption of AccessKey ID and AccessKey secret

If you do not encrypt the values of the id and key parameters in the CREATE SERVER statement, other users can obtain your AccessKey pair in plaintext by executing the `select * from pg_foreign_server` statement. You can use symmetric encryption to encrypt the values of the id and key parameters. Use different keys for different RDS instances. This further protects your AccessKey pair. However, you cannot add data types as you can in Greenplum. This prevents incompatibility with earlier versions.

The following snippet provides the encrypted values of the id and key parameters:

```
postgres=# select * from pg_foreign_server ;
  srvname  | srvowner | srvfdw | srvtype | srvversion | srvacl |
srvoptions
-----------+----------+--------+---------+------------+--------+-----------------------------
-----------------------------------------------------------------------------------------------
----------
---------------------------------
 ossserver |       10 |  16390 |         |            |        | {host=oss-cn-hangzhou-zmf.al
iyuncs.com,id=MD5xxxxxxxx,key=MD5xxxxxxxx,bucket=067862}
```

Each encrypted value starts with an MD5 string. The total length divided by 8 is 3. After these encrypted values are exported, they will not be encrypted again. Take note that you cannot create an AccessKey ID or AccessKey secret that starts with an MD5 string.

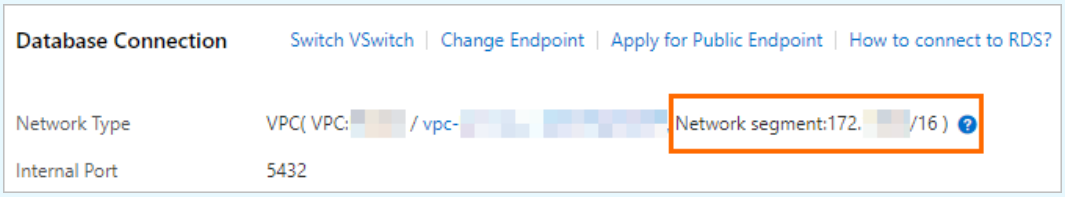
6.3.2. Use mysql_fdw to read data from and write data to a MySQL database

This topic describes how to use the mysql_fdw plug-in of ApsaraDB RDS for PostgreSQL to read data from and write data to a database on an ApsaraDB RDS for MySQL instance or a self-managed MySQL database.

Prerequisites

- The ApsaraDB RDS for PostgreSQL instance runs PostgreSQL 13, PostgreSQL 12, PostgreSQL 11, or PostgreSQL 10 with standard SSDs or enhanced SSDs (ESSDs).
- The CIDR block of the VPC to which the ApsaraDB RDS for PostgreSQL instance belongs is added to the IP address whitelist of the ApsaraDB RDS for MySQL instance or self-managed MySQL database. This way, the ApsaraDB RDS for PostgreSQL instance can communicate with the ApsaraDB RDS for MySQL instance or self-managed MySQL database. An example CIDR block is `172.xx.xx.xx/16`.

> **Note** You can view the CIDR block on the **Database Connection** page of the ApsaraDB RDS for PostgreSQL instance.



Context

PostgreSQL 9.6 and later versions support parallel computing. PostgreSQL 11 can complete join queries on up to 1 billion data records within seconds. A large number of users use PostgreSQL to build small-sized data warehouses and process highly concurrent access requests.

The mysql_fdw plug-in can establish a connection and synchronize data between an ApsaraDB RDS for PostgreSQL instance and a MySQL database.

Procedure

1. Create the mysql_fdw plug-in.

```
postgres=> create extension mysql_fdw;
CREATE EXTENSION
```

> **Note** Only privileged accounts are authorized to run the preceding command.

2. Define a MySQL server.

```
postgres=> CREATE SERVER <The name of the MySQL server>
postgres->      FOREIGN DATA WRAPPER mysql_fdw
postgres->      OPTIONS (host '<The endpoint of the MySQL server>', port '<The port numbe
r of the MySQL server>');
CREATE SERVER
```

> **Note** The value of `host` must be the internal endpoint of the MySQL server. The value of `port` must be the internal port number of the MySQL server.

Example:

```
postgres=> CREATE SERVER mysql_server
postgres->      FOREIGN DATA WRAPPER mysql_fdw
postgres->      OPTIONS (host 'rm-xxx.mysql.rds.aliyuncs.com', port '3306');
CREATE SERVER
```

3. Map the MySQL server to an account that is created on the ApsaraDB RDS for PostgreSQL instance. You can use the account to read data from and write data to the MySQL database, which resides on the MySQL server.

```
postgres=> CREATE USER MAPPING FOR <The username of the account to which the MySQL server
is mapped>
SERVER <The name of the MySQL server>
OPTIONS (username '<The username of the account that is used to connect to the MySQL data
base>', password '<The password of the preceding account>');
CREATE USER MAPPING
```

Example:

```
postgres=> CREATE USER MAPPING FOR pgtest
SERVER mysql_server
OPTIONS (username 'mysqltest', password 'Test1234!');
CREATE USER MAPPING
```

4. Create a foreign MySQL table by using the account that you mapped to the MySQL server in the previous step.

> **Note** The field names in the foreign MySQL table must be the same as the field names in the table in the MySQL database. You can choose to create only the fields that you want to query. For example, if the table in the MySQL database contains the ID, NAME, and AGE fields, you can create only the ID and NAME fields in the foreign MySQL table.

```
postgres=> CREATE FOREIGN TABLE <The name of the foreign MySQL table> (<The name of field
1> <The data type of field 1>,<The name of field 2> <The data type of field 2>...) server
<The name of the MySQL server> options (dbname '<The name of the MySQL database>', table_
name '<The name of the table in the MySQL database>');
CREATE FOREIGN TABLE
```

Example:

```
postgres=> CREATE FOREIGN TABLE ft_test (id1 int, name1 text) server mysql_server options
(dbname 'test123', table_name 'test');
CREATE FOREIGN TABLE
```

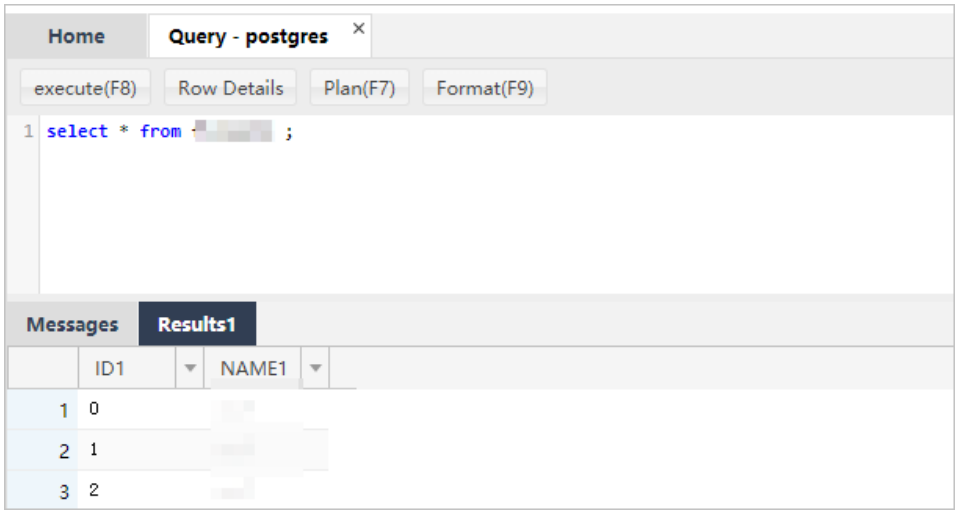
What to do next

You can use the foreign MySQL table to check the performance of the read and write operations on the MySQL database.

Note Data can be written to the table in the MySQL database only when the table is assigned a primary key. If the table is not assigned a primary key, the following error is returned:

```
ERROR:  first column of remote table must be unique for INSERT/UPDATE/DELETE operation.
```

```
postgres=> select * from ft_test ;
postgres=> insert into ft_test values (2,'abc');
INSERT 0 1
postgres=> insert into ft_test select generate_series(3,100),'abc';
INSERT 0 98
postgres=> select count(*) from ft_test ;
 count
-------
    99
(1 row)
```



View the execution plan to check how the requests sent by the ApsaraDB RDS for PostgreSQL instance to query data from the MySQL database are executed.

```
postgres=> explain verbose select count(*) from ft_test ;
                                QUERY PLAN
---------------------------------------------------------------------------
 Aggregate  (cost=1027.50..1027.51 rows=1 width=8)
   Output: count(*)
   ->  Foreign Scan on public.ft_test  (cost=25.00..1025.00 rows=1000 width=0)
         Output: id, info
         Remote server startup cost: 25
         Remote query: SELECT NULL FROM `test123`.`test`
(6 rows)
postgres=> explain verbose select id from ft_test where id=2;
                              QUERY PLAN
-----------------------------------------------------------------------
 Foreign Scan on public.ft_test  (cost=25.00..1025.00 rows=1000 width=4)
   Output: id
   Remote server startup cost: 25
   Remote query: SELECT `id` FROM `test123`.`test` WHERE ((`id` = 2))
(4 rows)
```

6.3.3. Use the log_fdw plug-in

This topic describes how to use the log_fdw plug-in to query the database logs of an RDS PostgreSQL instance.

Prerequisites

The RDS instance runs PostgreSQL 11.

Context

The log_fdw plug-in provides the following two functions:

- list_postgres_log_files(): lists all .csv log files.
- create_foreign_table_for_log_file(IN table_name text, IN log_server text, IN log_file text): creates a foreign table associated with a specific .csv log file.

Procedure

1. Create the log_fdw plug-in.

   ```
   postgres=> create extension log_fdw;
   CREATE EXTENSION
   ```

2. Create a definition for the log server.

   ```
   postgres=> create server <The name of the log server> foreign data wrapper log_fdw;
   ```

   Example:

   ```
   postgres=> create server log_server foreign data wrapper log_fdw;
   CREATE SERVER
   ```

3. Invoke the list_postgres_log_files() function to list all .csv log files.

```
postgres=> select * from list_postgres_log_files() order by 1;
            file_name             | file_size_bytes
----------------------------------+-----------------
 postgresql-2020-01-10_095546.csv |            3794
 postgresql-2020-01-10_100336.csv |          318318
 postgresql-2020-01-11_000000.csv |          198437
 postgresql-2020-01-11_083546.csv |            4775
 postgresql-2020-01-13_030618.csv |            3347
```

4. Invoke the create_foreign_table_for_log_file(IN table_name text, IN log_server text, IN log_file text) function to create a foreign table associated with a specific .csv log file.

```
postgres=> select create_foreign_table_for_log_file('<The name to use for the foreign tab
le>', '<The name of the log server>', '<The name of the .csv log file associated with the
foreign table>');
```

Example:

```
postgres=> select create_foreign_table_for_log_file('ft1', 'log_server', 'postgresql-2020
-01-13_030618.csv');
 create_foreign_table_for_log_file
-----------------------------------
 t
(1 row)
```

5. Query the foreign table to obtain the data of the .csv log file associated with it.

```
postgres=> select log_time, message from <The name of the foreign table to query> order b
y log_time desc limit 2;
```

Example:

```
postgres=> select log_time, message from ft1 order by log_time desc limit 2;
          log_time          |                              message
----------------------------+-------------------------------------------------------------
-------
 2020-01-13 03:35:00.003+00 | cron job 1 completed: INSERT 0 1 1
 2020-01-13 03:35:00+00     | cron job 1 starting: INSERT INTO cron_test VALUES ('Hello W
orld')
(2 rows)
```

Schema of a foreign table

```
postgres=> \d+ ft1
                                                        Foreign table "public.ft1"
         Column         |            Type             | Collation | Nullable | Default | FDW
options | Storage  | Stats target | Description
------------------------+-----------------------------+-----------+----------+---------+-----
--------+----------+--------------+-------------
 log_time               | timestamp(3) with time zone |           |          |         |
| plain    |              |
 user_name              | text                        |           |          |         |
| extended |              |
 database_name          | text                        |           |          |         |
| extended |              |
 process_id             | integer                     |           |          |         |
| plain    |              |
 connection_from        | text                        |           |          |         |
| extended |              |
 session_id             | text                        |           |          |         |
| extended |              |
 session_line_num       | bigint                      |           |          |         |
| plain    |              |
 command_tag            | text                        |           |          |         |
| extended |              |
 session_start_time     | timestamp with time zone    |           |          |         |
| plain    |              |
 virtual_transaction_id | text                        |           |          |         |
| extended |              |
 transaction_id         | bigint                      |           |          |         |
| plain    |              |
 error_severity         | text                        |           |          |         |
| extended |              |
 sql_state_code         | text                        |           |          |         |
| extended |              |
 message                | text                        |           |          |         |
| extended |              |
 detail                 | text                        |           |          |         |
| extended |              |
 hint                   | text                        |           |          |         |
| extended |              |
 internal_query         | text                        |           |          |         |
| extended |              |
 internal_query_pos     | integer                     |           |          |         |
| plain    |              |
 context                | text                        |           |          |         |
| extended |              |
 query                  | text                        |           |          |         |
| extended |              |
 query_pos              | integer                     |           |          |         |
| plain    |              |
 location               | text                        |           |          |         |
| extended |              |
 application_name       | text                        |           |          |         |
| extended |              |
Server: log_server
FDW options: (filename 'postgresql-2020-01-13_030618.csv')
```

6.3.4. Use the tds_fdw plug-in to query data of SQL Server instances

This topic describes how to use the tds_fdw plug-in provided by PostgreSQL to query data of SQL Server instances.

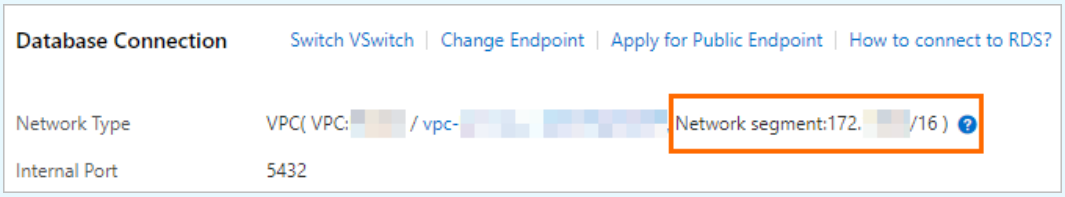
Prerequisites

- Your ApsraDB RDS for PostgreSQL instance runs one of the following database engine versions:
  - PostgreSQL 12 with a minor engine version of 20200421 or later
  - PostgreSQL 11 with a minor engine version of 20200402 or later

> **Note** To view the minor engine version of your RDS instance, you must log on to the ApsaraDB RDS console and go to the **Basic Information** page. In the **Configuration Information** section of the page, you can check whether the **Upgrade Kernel Version** button is displayed. If the button is displayed, you can click the button to view and update the minor engine version of your RDS instance. If the button is not displayed, your RDS instance runs the latest minor engine version. For more information, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

- You must add the CIDR block of the virtual private cloud (VPC) in which your RDS instance resides to an IP address whitelist of the SQL Server instance that you want to connect. Example of an CIDR block: `172.xx.xx.xx/16`.

> **Note** You can view the CIDR block of the VPC in which an RDS instance resides on the **Database Connection** page of the ApsaraDB RDS console.



Context

The tds_fdw plug-in is a Foreign Data Wrapper (FDW) provided by PostgreSQL that you can use to connect with foreign instances such as Sybase and Microsoft SQL Server instances. These instances use the Tabular Data Stream (TDS) protocol.

For more information, see postgres_fdw.

Create a tds_fdw plug-in

After you have connected to an SQL Server instance, execute the following statement to create a tds_fdw plug-in:

```
create extension tds_fdw;
```

Use the tds_fdw plug-in

1. Execute the following statement to create a server:

```
CREATE SERVER mssql_svr
  FOREIGN DATA WRAPPER tds_fdw
  OPTIONS (servername '<The endpoint used to connect to the SQL Server instance>', port '
<The port used to connect to the SQL Server instance>', database 'tds_fdw_test', tds_vers
ion '7.1');
```

> **Note** You must set the `servername` parameter to the internal endpoint used to connect to the SQL Server instance and the `port` parameter to the internal port used to connect to the SQL Server instance.

2. Create a foreign table. You can use one of the following methods to create a foreign table:
   - Specify the table_name parameter and execute the following statement to create a foreign table:

```
CREATE FOREIGN TABLE mssql_table (
 id integer,
 data varchar)
 SERVER mssql_svr
 OPTIONS (table_name 'dbo.mytable', row_estimate_method 'showplan_all');
```

   - Specify the schema_name and table_name parameters and execute the following statement to create a foreign table:

```
CREATE FOREIGN TABLE mssql_table (
 id integer,
 data varchar)
 SERVER mssql_svr
 OPTIONS (schema_name 'dbo', table_name 'mytable', row_estimate_method 'showplan_all');
```

   - Specify the query parameter and execute the following statement to create a foreign table:

```
CREATE FOREIGN TABLE mssql_table (
 id integer,
 data varchar)
 SERVER mssql_svr
 OPTIONS (query 'SELECT * FROM dbo.mytable', row_estimate_method 'showplan_all');
```

   - Specify a foreign column name and execute the following statement to create a foreign table:

```
CREATE FOREIGN TABLE mssql_table (
 id integer,
 col2 varchar OPTIONS (column_name 'data'))
 SERVER mssql_svr
 OPTIONS (schema_name 'dbo', table_name 'mytable', row_estimate_method 'showplan_all');
```

3. Execute the following statement to create a user mapping:

```
CREATE USER MAPPING FOR postgres
  SERVER mssql_svr
  OPTIONS (username 'sa', password '123456');
```

4. Execute the following statement to import a schema from a foreign table:

```
IMPORT FOREIGN SCHEMA dbo
  EXCEPT (mssql_table)
  FROM SERVER mssql_svr
  INTO public
  OPTIONS (import_default 'true');
```

6.3.5. Use the oracle_fdw plug-in

This topic describes how to use the oracle_fdw plug-in to connect to an Oracle database. You can also use this plug-in to synchronize data between tables in a PostgreSQL database and tables in an Oracle database.

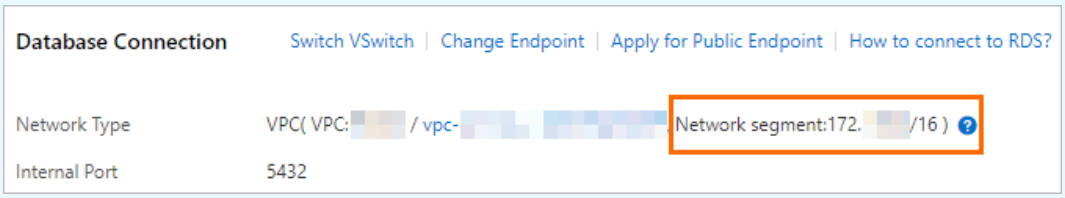
Prerequisites

- Your ApsaraDB RDS for PostgreSQL instance runs PostgreSQL 12 with the minor engine version of 20200421 or later.

  > **Note** You can execute the `SHOW rds_supported_extensions;` statement to check whether the current minor engine version of your RDS instance supports the oracle_fdw plug-in. If the current minor engine version does not support the oracle_fdw plug-in, you must first update the minor engine version.

- The Oracle client version is 11.2 or later.
- The Oracle server version is based on the Oracle client version. For more information, see Oracle documentation.
- You must add the CIDR block of the virtual private cloud (VPC) in which your RDS instance resides to an IP address whitelist of the Oracle database that you want to connect. Example of an CIDR block: `172.xx.xx.xx/16`.

  > **Note** You can view the CIDR block of the VPC in which an RDS instance resides on the **Database Connection** page of the ApsaraDB RDS console.

Context

The oracle_fdw plug-in is developed by PostgreSQL to manage foreign tables. The plug-in provides easy access to Oracle databases and allows you to synchronize data between PostgreSQL databases and Oracle databases.

For more information, see oracle_fdw.

Precautions

- If you want to execute the UPDATE or DELETE statements, you must set the key parameter to true for primary key columns when you create a foreign table. For more information, see the "Create a foreign table" section of this topic.

- The data types of columns in the foreign table must be identifiable and convertible for the oracle_fdw plug-in. For more information about the conversion rules supported by the oracle_fdw plug-in, see Data types.
- The oracle_fdw plug-in can push down the WHERE and ORDER BY clauses to Oracle databases.
- The oracle_fdw plug-in can push down JOIN operations to Oracle databases. Pushdown has the following limits:
  - Both tables for a JOIN operation must be defined in the same database mapping.
  - JOIN operations on three or more tables cannot be pushed down.
  - JOIN operations must be included in a SELECT statement.
  - Cross JOIN operations without JOIN conditions cannot be pushed down.
  - If a JOIN operation is pushed down, ORDER BY clauses are not pushed down.
- After PostGIS is installed, the oracle_fdw plug-in further supports the following spatial data types:
  - Point
  - Line
  - Polygon
  - MultiPoint
  - MultiLine
  - MultiPolygon

Procedure

1. Execute the following statement to create an oracle_fdw plug-in:

```
CREATE EXTENSION oracle_fdw;
```

2. Execute one of the following statements to create an Oracle database mapping:

```
CREATE SERVER <Server name>
FOREIGN DATA WRAPPER oracle_fdw
OPTIONS (dbserver '//<The internal endpoint that is used to connect to the Oracle datab
ase>:<The internal port that is used to connect to the Oracle database>/<The name of th
e Oracle database that you want to connect>');
```

```
CREATE SERVER oradb
FOREIGN DATA WRAPPER oracle_fdw
OPTIONS (host '<The internal endpoint that is used to connect to the Oracle database>',
port '<The internal port that is used to connect to the Oracle database>', dbname '<The
name of the Oracle database that you want to connect>');
```

3. Execute the following statement to create a user mapping:

```
CREATE USER MAPPING
FOR <The username used to log on to the PostgreSQL database> SERVER <The name of the user
mapping>
OPTIONS (user '<The username used to log on to the Oracle database>', password '<The pass
word used to log on to the Oracle database>');
```

> **Note** If you do not store the Oracle user credentials in the PostgreSQL database, set the user parameter to an empty string and provide external authorization credentials.

Example:

```
CREATE USER MAPPING
FOR pguser SERVER oradb
OPTIONS (user 'orauser', password 'orapwd');
```

4. Executee the following statement to create a foreign table:

```
CREATE FOREIGN TABLE oratab (
          id        integer OPTIONS (key 'true')  NOT NULL,
          text      character varying(30),
          floating  double precision  NOT NULL
       ) SERVER oradb OPTIONS (table 'ORATAB',
                               schema 'ORAUSER',
                               max_long '32767',
                               readonly 'false',
                               sample_percent, '100',
                               prefetch, '200');
```

> **Note** The schema of the foreign table must be consistent with that of the mapped Oracle table.

The following table describes the parameters in OPTIONS.

Parameter	Description
key	Specifies whether to set a column as a primary key column. Valid values: true and false. Default value: false. If you want to execute the UPDATE and DELETE statements, you must set the value to true for all primary key columns.
table	Required. The name of the Oracle table. The value must be in uppercase. You can also use an Oracle SQL statement to define the value of the table parameter. Example: `OPTIONS (table '(SELECT col FROM tab WHERE val = ''string'')')`. In this case, do not use the schema parameter.
schema	The Oracle username for accessing a table that does not belong to the currently connected user. The value must be in uppercase.
max_long	The maximum length of columns that have the LONG, LONG RAW, or XMLTYPE data type in the Oracle table. Valid values: 1 to 1073741823. Default value: 32767.
readonly	Specifies whether the Oracle table is read-only. If the value is true, you cannot execute the INSERT, UPDATE, or DELETE statements.
sample_percent	The percentage of Oracle table blocks that are randomly selected to calculate PostgreSQL table statistics. Valid values: 0.000001 to 100. Default value: 100.
prefetch	The number of rows that are fetched for a single round-trip transmission between PostgreSQL and Oracle during a foreign table scan. Valid values: 0 to 1024. Default value: 200. The value 0 indicates that the prefetch feature is disabled.

After you create the foreign table, you can use it to perform operations on the Oracle table. Basic SQL statements such as DELETE, INSERT, UPDATE, and SELECT are supported. Foreign table definitions can be imported. Sample statement:

```
IMPORT FOREIGN SCHEMA <ora_schema_name>
FROM SERVER <server_name>
INTO <schema_name>
OPTIONS (case 'lower');
```

Note You can set the case parameter to one of the following valid values:

- keep: uses the same object names as those in Oracle. In most cases, the names are in uppercase.
- lower: converts all object names to lowercase.
- smart: converts only the object names that are in all uppercase to lowercase.

Delete the oracle_fdw plug-in

Execute the following SQL statement to delete the oracle_fdw plug-in:

```
DROP EXTENSION oracle_fdw;
```

6.4. Use the dblink and postgres_fdw plug-ins for cross-database operations

This topic describes how to use the dblink and postgres_fdw plug-ins provided with PostgreSQL to manage tables across databases.

Context

ApsaraDB RDS for PostgreSQL instances that use standard SSDs or enhanced SSDs (ESSDs) support the dblink and postgres_fdw plug-ins. You can use the plug-ins to manage tables across databases on instances that reside in the same virtual private cloud (VPC). The instances include self-managed PostgreSQL instances.

If you want to purchase an ApsaraDB RDS for PostgreSQL instance that uses standard SSDs or ESSDs, go to the ApsaraDB RDS buy page.

Precautions

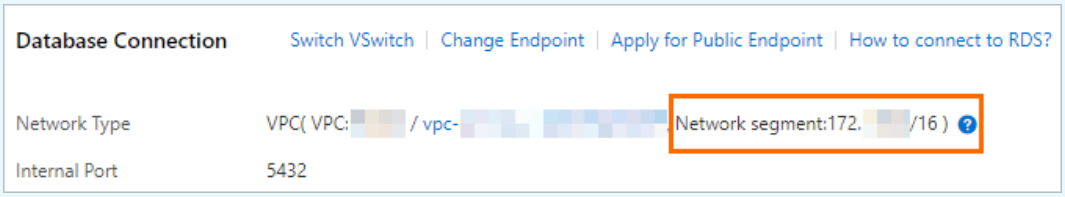
Take notes of the following items before you perform cross-database operations in ApsaraDB RDS for PostgreSQL instances that use standard SSDs or ESSDs:

- If a self-managed PostgreSQL instance resides on an Elastic Compute Service (ECS) instance, and the ECS instance and your RDS instance reside in the same VPC, you can directly perform cross-database operations.
- If you want to connect a self-managed PostgreSQL instance to an Oracle or MySQL instance that resides in different VPCs, you can use the oracle_fdw or mysql_fdw plug-in.
- If you want to manage tables across databases on the same RDS instance, you must set the host parameter to localhost and the port parameter to the local port that is obtained by executing the `sh`

`ow port` statement.

- You must add the CIDR block of the VPC where your RDS instance resides to an IP address whitelist of the destination instance that you want to connect. The CIDR block of the VPC follows the `172.xx.xx.xx/16` format.

  Note You can view the CIDR block of the VPC where an RDS instance resides on the **Database Connection** page of the ApsaraDB RDS console.



Use the dblink plug-in

1. Create the dblink plug-in.

```
create extension dblink;
```

2. Create a dblink connection to the destination RDS instance that resides in the same VPC as your RDS instance.

```
postgres=> select dblink_connect('<The name of the connection>', 'host=<The internal endp
oint used to connect to the destination RDS instance> port=<The internal port used to con
nect to the destination RDS instance> user=<The username used to log on to the database t
hat you want to manage on the destination RDS instance> password=<The password used to lo
g on to the database that you want to manage on the destination RDS instance> dbname=<The
name of the database that you want to manage on the destination RDS instance>');
postgres=> SELECT * FROM dblink('<The name of the connection>', '<The SQL statement to ex
ecute>') as <The name of the table to manage>(<The name of the column to manage> <The typ
e of the column to manage>);
```

   Example:

```
postgres=> select dblink_connect('a', 'host=pgm-bpxxxxx.pg.rds.aliyuncs.com port=3433 use
r=testuser2 password=passwd1234 dbname=postgres');
postgres=> select * from dblink('a','select * from products') as T(id int,name text,price
numeric);  //Query a table on the destination RDS instance.
```

For more information, see dblink.

Use the postgres_fdw plug-in

1. Create a database.

```
postgres=> create database <The name of the database>;  //Create a database.
postgres=> \c <The name of the created database>  //Switch to the database that you creat
ed.
```

   Example:

```
postgres=> create database db1;
CREATE DATABASE
postgres=> \c db1
```

2. Create the postgres_fdw plug-in.

```
db1=> create extension postgres_fdw;
```

3. Create a remote database server that can connect to the destination RDS instance that resides in the same VPC as your RDS instance.

```
db1=> CREATE SERVER <The name of the remote database server>

        FOREIGN DATA WRAPPER postgres_fdw
        OPTIONS (host '<The internal endpoint used to connect to the destination RDS inst
ance>,port '<The internal port used to connect to the destination RDS instance>', dbname
'<The name of the database that you want to manage on the destination RDS instance>');
db1=> CREATE USER MAPPING FOR <The username used to log on to your RDS instance>
        SERVER <The name of the created remote database server>
         OPTIONS (user '<The username used to log on to the database that you want to man
age on the destination RDS instance>', password '<The password used to log on to the data
base that you want to manage on the destination RDS instance>');
```

Example:

```
db1=> CREATE SERVER foreign_server1

        FOREIGN DATA WRAPPER postgres_fdw
        OPTIONS (host 'pgm-bpxxxxx.pg.rds.aliyuncs.com', port '3433', dbname 'postgres');

CREATE SERVER
db1=> CREATE USER MAPPING FOR testuser
        SERVER foreign_server1
        OPTIONS (user 'testuser2', password 'passwd1234');
CREATE USER MAPPING
```

4. Import an external table.

```
db1=> import foreign schema public from server foreign_server1 into <The name of the sche
ma used by the external table>;  //Import an external table.
db1=> select * from <The name of the schema used by the external table>.<The name of the
external table>      //Query a remote table.
```

Example:

```
db1=> import foreign schema public from server foreign_server1 into ft;
IMPORT FOREIGN SCHEMA
db1=> select * from ft.products;
```

For more information, see postgres_fdw.

6.5. Use the hll plug-in

This topic describes the use of the HyperLogLog data type supported by the hll plug-in to estimate page views (PV) and unique visitors (UV).

Prerequisites

The instance runs one of the following PostgreSQL versions:

- PostgreSQL 14
- PostgreSQL 13
- PostgreSQL 12 (kernel version 20200421 and later)
- PostgreSQL 11 (kernel version 20200402 and later)

> **Note** To view the kernel version, perform the following steps: Log on to the ApsaraDB for RDS console, find the target RDS instance, and navigate to the **Basic Information** page. Then, in the **Configuration Information** section, check whether the **Upgrade Minor Version** button exists. If the button exists, click it to view the kernel version. If the button does not exist, it indicates that you are already using the latest kernel version. For more information, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

Context

The hll plug-in supports an extendable, set-resembled data type HyperLogLog (hll) to estimate DISTINCT elements under a specified accuracy. For example, you can use 1,280 bytes of hll data to accurately estimate billions of DISTINCT elements. The hll plug-in is suitable for industries that need estimation analysis, such as Internet advertisement analysis to estimate PVs and UVs.

For more information about how to use the hll plug-in, visit postgresql-hll.

For more information about the detailed algorithm, visit HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm.

Create an hll plug-in

After you connect to an instance, execute the following statement to create an hll plug-in:

```
CREATE EXTENSION hll;
```

Basic operations

- Execute the following statement to create a table that contains hll fields:

  ```
  create table agg (id int primary key,userids hll);
  ```

- Execute the following statement to convert INT data to hll_hashval data:

  ```
  select 1::hll_hashval;
  ```

Basic operators

- The hll data type supports the following operators:
  - =
  - !=
  - <>
  - ||
  - #

  Examples:

```
select hll_add_agg(1::hll_hashval) = hll_add_agg(2::hll_hashval);
select hll_add_agg(1::hll_hashval) || hll_add_agg(2::hll_hashval);
select #hll_add_agg(1::hll_hashval);
```

- The hll_hashval data type supports the following operators:
  - =
  - !=
  - <>

  Examples:

```
select 1::hll_hashval = 2::hll_hashval;
select 1::hll_hashval <> 2::hll_hashval;
```

Basic functions

- The hll plug-in supports hash functions such as hll_hash_boolean, hll_hash_smallint, and hll_hash_bigint. Examples:

```
select hll_hash_boolean(true);
select hll_hash_integer(1);
```

- The hll plug-in supports the hll_add_agg function to convert the data type from INT to hll. Example:

```
select hll_add_agg(1::hll_hashval);
```

- The hll plus-in supports the hll_union function to perform UNION operations on hll data. Example:

```
select hll_union(hll_add_agg(1::hll_hashval),hll_add_agg(2::hll_hashval));
```

- The hll plus-in supports the hll_set_defaults function to set the accuracy. Example:

```
select hll_set_defaults(15,5,-1,1);
```

- The hll plug-in supports the hll_print function to display debug information. Example:

```
select hll_print(hll_add_agg(1::hll_hashval));
```

Example

```
create table access_date (acc_date date unique, userids hll);
insert into access_date select current_date, hll_add_agg(hll_hash_integer(user_id)) from gene
rate_series(1,10000) t(user_id);
insert into access_date select current_date-1, hll_add_agg(hll_hash_integer(user_id)) from ge
nerate_series(5000,20000) t(user_id);
insert into access_date select current_date-2, hll_add_agg(hll_hash_integer(user_id)) from ge
nerate_series(9000,40000) t(user_id);
postgres=# select #userids from access_date where acc_date=current_date;
     ? column?
------------------
 9725.85273370708
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-1;
     ? column?
------------------
 14968.6596883279
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-2;
     ? column?
------------------
 29361.5209149911
(1 row)
```

6.6. Use the pg_cron plug-in to configure scheduled tasks

This topic describes how to use the pg_cron plug-in to configure scheduled tasks for an ApsaraDB RDS for PostgreSQL instance.

Context

The pg_cron plug-in allows you to schedule tasks by using CRON. The pg_cron plug-in uses the same syntax as standard CRON expressions but can initiate PostgreSQL commands from databases.

Each scheduled task consists of the following parts:

- Schedule

  The schedule based on which ApsaraDB RDS uses the pg_cron plug-in to run the task that you configure. For example, the schedule specifies to run the task at an interval of 1 minute.

  The schedule follows the same syntax as standard CRON expressions. In the syntax, a wildcard (*) specifies to **run the task at any time**, and a number specifies to **run the task only within the time range that is specified by this number**.

  ```
  ┌──────────── Minute: 0 to 59
  │ ┌──────────── Hour: 0 to 23
  │ │ ┌──────────── Date: 1 to 31
  │ │ │ ┌──────────── Month: 1 to 12
  │ │ │ │ ┌──────────── Day of the week: 0 to 6 (The value 0 indicates Sunday.)
  │ │ │ │ │
  │ │ │ │ │
  │ │ │ │ │
  * * * * *
  ```

The following schedule specifies to run the task at Greenwich Mean Time (GMT) 03:30 each Saturday:

```
30 3 * * 6
```

- Task

  The task that you configured. Example: `select * from some_table`.

Precautions

- The pg_cron plug-in is supported for RDS instances that run PostgreSQL 10, PostgreSQL 11, or PostgreSQL 12.
- Scheduled tasks are run based on GMT.
- Scheduled tasks are stored in a default database named postgres. You can query scheduled tasks in other databases.
- A new version of the pg_cron plug-in is provided. If you used the pg_cron plug-in before you updated the minor AliPG version to 20201130, you can re-create the pg_cron plug-in. This way, you can use the new features that are provided in the new version of the pg_cron plug-in. After you re-create the pg_cron plug-in, the scheduled tasks that are created in the original version of the pg_cron plug-in are lost. For more information, see Release notes for AliPG.
- You must add pg_cron to the value of the shared_preload_libraries parameter of your RDS instance.

  For more information about how to add **pg_cron** to the value of the **shared_preload_libraries** parameter, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

Use the pg_cron plug-in

- Create the pg_cron plug-in.

  ```
  CREATE EXTENSION pg_cron;
  ```

  > **Note** Only privileged accounts are granted the permissions to run the preceding command.

- Delete the pg_cron plug-in.

  ```
  DROP EXTENSION pg_cron;
  ```

  > **Note** Only privileged accounts are granted the permissions to run the preceding command.

- Run a scheduled task.

  ```
  SELECT cron.schedule('<Schedule>', '<Task>')
  ```

  Examples:

```
-- Delete outdated data at GMT 03:30 each Saturday.
SELECT cron.schedule('30 3 * * 6', $$DELETE FROM events WHERE event_time < now() - interval
'1 week'$$);
----------
-- Clear the disk at GMT 10:00 each day.
SELECT cron.schedule('0 10 * * *', 'VACUUM');
----------
-- Execute a specified script at an interval of 1 minute.
SELECT cron.schedule('* * * * *', 'select 1;');
----------
-- Execute a specified script at the twenty-third minute of each hour.
SELECT cron.schedule('23 * * * *', 'select 1;');
----------
-- Execute a specified script on the fourth day of each month.
SELECT cron.schedule('* * 4 * *', 'select 1;');
```

- Run a scheduled task on a specified database.

```
SELECT cron.schedule('<Schedule>', '<Task>', '<Database>')
```

> **Note** If you do not specify a database, the scheduled task is run on the default database named postgres that is specified in the configuration file.

- Delete a scheduled task.

```
SELECT cron.unschedule(<The ID of the scheduled task>)
```

Example:

```
SELECT cron.unschedule(43);
```

- View all scheduled tasks.

```
SELECT * FROM cron.job;
```

6.7. Use the PL/Proxy plug-in for horizontal sharding

The PL/Proxy plug-in allows you to access your ApsaraDB RDS instance in CLUSTER or CONNECT mode.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 14
- PostgreSQL 13
- PostgreSQL 12 (with a minor engine version of 20200421 or later)
- PostgreSQL 11 (with a minor engine version of 20200402 or later)

> **Note** If you want to view the minor engine version, perform the following steps: Log on to the ApsaraDB RDS console, find your RDS instance and navigate to the **Basic Information** page. In the **Configuration Information** section of the page, check whether the **Upgrade Minor Version** button exists. If the button exists, you can click it to view and update the minor engine version. If the button does not exist, you are using the latest minor engine version. For more information, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

Context

The PL/Proxy plug-in supports the following modes:

- CLUSTER

  This mode supports horizontal sharding and SQL replication.

- CONNECT

  This mode allows ApsaraDB RDS to route SQL requests to specified databases.

For more information about how to use the PL/Proxy plug-in, visit PL/Proxy.

Precautions

- You can directly manage tables across RDS instances that reside in the same virtual private cloud (VPC).
- An Elastic Compute Service (ECS) instance that resides in the same VPC as your RDS instance can serve as a proxy to redirect access requests for your RDS instance. This allows you to manage tables across RDS instances that reside in different VPCs.
- The number of data nodes that are served by the proxy node must be 2 to the power of n.

Test environment

Select an RDS instance as the proxy node and another two RDS instances as the data nodes. The following table provides details about the three RDS instances.

IP address	Node type	Instance name	Username
100.xx.xx.136	Proxy node	postgres	postgres
100.xx.xx.72	Data node	pl_db0	postgres
11.xx.xx.9	Data node	pl_db1	postgres

Create a PL/Proxy plug-in

Execute the following statement to create a PL/Proxy plug-in:

```
create extension plproxy
```

Create a PL/Proxy cluster

> **Note** If you use the CONNECT mode, you can skip the operations that are described in this section.

1. Create a PL/Proxy cluster and specify the names, IP addresses, and ports of the RDS instances that

you want to connect as data nodes in the cluster. Example:

```
postgres=# CREATE SERVER cluster_srv1 FOREIGN DATA WRAPPER plproxy
postgres-# OPTIONS (
postgres(#         connection_lifetime '1800',
postgres(#         disable_binary '1',
postgres(#         p0 'dbname=pl_db0 host=100.xxx.xxx.72 port=5678',
postgres(#         p1 'dbname=pl_db1 host=11.xxx.xxx.9 port=5678'
postgres(#         );
CREATE SERVER
```

2. Grant the permissions on the created PL/Proxy cluster to the postgres user. Example:

```
postgres=# grant usage on FOREIGN server cluster_srv1 to postgres;
GRANT
```

3. Create a user mapping. Example:

```
postgres=> create user mapping for postgres server cluster_srv1 options (user 'postgres')
;
CREATE USER MAPPING
```

Create a test table

Create a test table named users on each data node. Example:

```
create table users(userid int, name text);
```

Test the CLUSTER mode

To test horizontal sharding, perform the following steps:

1. Create a function that is used to insert data on each data node. Example:

```
pl_db0=> CREATE OR REPLACE FUNCTION insert_user(i_id int, i_name text)
pl_db0-> RETURNS integer AS $$
pl_db0$>        INSERT INTO users (userid, name) VALUES ($1,$2);
pl_db0$>        SELECT 1;
pl_db0$> $$ LANGUAGE SQL;
CREATE FUNCTION
pl_db1=> CREATE OR REPLACE FUNCTION insert_user(i_id int, i_name text)
pl_db1-> RETURNS integer AS $$
pl_db1$>        INSERT INTO users (userid, name) VALUES ($1,$2);
pl_db1$>        SELECT 1;
pl_db1$> $$ LANGUAGE SQL;
CREATE FUNCTION
```

2. Create a function that is used to insert data on the proxy node. This function has the same name as the function that is used to insert data on each data node. Example:

```
postgres=> CREATE OR REPLACE FUNCTION insert_user(i_id int, i_name text)
postgres-> RETURNS integer AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ANY;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

3. Create a function that is used to read data on the proxy node. Example:

```
postgres=> CREATE OR REPLACE FUNCTION get_user_name()
postgres-> RETURNS TABLE(userid int, name text) AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ALL ;
postgres$> SELECT userid,name FROM users;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

4. Insert 10 test records on the proxy node. Example:

```
SELECT insert_user(1001, 'Sven');
SELECT insert_user(1002, 'Marko');
SELECT insert_user(1003, 'Steve');
SELECT insert_user(1004, 'lottu');
SELECT insert_user(1005, 'rax');
SELECT insert_user(1006, 'ak');
SELECT insert_user(1007, 'jack');
SELECT insert_user(1008, 'molica');
SELECT insert_user(1009, 'pg');
SELECT insert_user(1010, 'oracle');
```

5. View the data on each data node. The function that is used to insert data contains the RUN ON ANY statement. This statement randomly inserts data into either data node. Therefore, you may find the following data on the two data nodes:

```
pl_db0=> select * from users;
 userid |  name
--------+--------
   1001 | Sven
   1003 | Steve
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
   1008 | molica
   1009 | pg
(8 rows)
pl_db1=> select * from users;
 userid |  name
--------+--------
   1002 | Marko
   1010 | oracle
(2 rows)
```

> **Note** The query results indicate that the 10 data records are unevenly distributed between the two data nodes.

6. Invoke the function that is used to read data on the proxy node. This function contains the RUN ON ALL statement that reads data from both data nodes. Example:

```
postgres=> SELECT USERID,NAME FROM GET_USER_NAME();
 userid |  name
--------+--------
   1001 | Sven
   1003 | Steve
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
   1008 | molica
   1009 | pg
   1002 | Marko
   1010 | oracle
(10 rows)
```

To test SQL replication, perform the following steps:

1. Create a function that is used to truncate the users table on each node. Example:

```
pl_db0=> CREATE OR REPLACE FUNCTION trunc_user()
pl_db0-> RETURNS integer AS $$
pl_db0$>        truncate table users;
pl_db0$>        SELECT 1;
pl_db0$> $$ LANGUAGE SQL;
CREATE FUNCTION
pl_db1=> CREATE OR REPLACE FUNCTION trunc_user()
pl_db1-> RETURNS integer AS $$
pl_db1$>        truncate table users;
pl_db1$>        SELECT 1;
pl_db1$> $$ LANGUAGE SQL;
CREATE FUNCTION
postgres=> CREATE OR REPLACE FUNCTION trunc_user()
postgres-> RETURNS SETOF integer AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ALL;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

2. Invoke the function that is used to truncate data on the proxy node. Example:

```
postgres=> SELECT TRUNC_USER();
 trunc_user
------------
          1
          1
(2 rows)
```

3. Create a function that is used to insert data on the proxy node. Example:

```
postgres=> CREATE OR REPLACE FUNCTION insert_user_2(i_id int, i_name text)
postgres-> RETURNS SETOF integer AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ALL;
postgres$> TARGET insert_user;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

4. Insert four test records into the proxy node. Example:

```
SELECT insert_user_2(1004, 'lottu');
SELECT insert_user_2(1005, 'rax');
SELECT insert_user_2(1006, 'ak');
SELECT insert_user_2(1007, 'jack');
```

5. View the data on each data node. Example:

```
pl_db0=> select * from users;
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)
pl_db1=> select * from users;
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)
```

> **Note** The data is the same on each data node. This indicates that SQL replication is successful.

6. Query data on the proxy node. You need only to execute the RUN ON ANY statement that randomly reads data from either data node. Example:

```
postgres=> CREATE OR REPLACE FUNCTION get_user_name_2()
postgres-> RETURNS TABLE(userid int, name text) AS $$
postgres$>     CLUSTER 'cluster_srv1';
postgres$>     RUN ON ANY ;
postgres$> SELECT userid,name FROM users;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
postgres=> SELECT USERID,NAME FROM GET_USER_NAME_2();
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)
```

Test the CONNECT mode

When you use the CONNECT mode, you can access other RDS instances from the proxy node. Examples:

```
postgres=> CREATE OR REPLACE FUNCTION get_user_name_3()
postgres-> RETURNS TABLE(userid int, name text) AS $$
postgres$>     CONNECT 'dbname=pl_db0 host=100.81.137.72 port=56789';
postgres$> SELECT userid,name FROM users;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
postgres=> SELECT USERID,NAME FROM GET_USER_NAME_3();
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)
```

6.8. Fuzzy query (PG_ bigm)

The pg_bigm plug-in that is provided by ApsaraDB RDS for PostgreSQL supports full-text search. It allows you to create a 2-gram Generalized Inverted Index (GIN) index that is used to expedite full-text search queries.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 13
- PostgreSQL 12
- PostgreSQL 11
- PostgreSQL 10

Note If the pg_bigm plugin cannot be created, you must update the minor engine version of your RDS instance before you create the plug-in. For more information, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

Differences between pg_bigm and pg_trgm

The pg_trgm plug-in is also provided by ApsaraDB RDS for PostgreSQL. However, it uses a 3-gram model to implement full-text search. The pg_bigm plug-in is developed based on the pg_trgm plug-in. The following table describes the differences between the two plug-ins.

Functionality	pg_trgm	pg_bigm
Phrase matching model	3-gram	2-gram
Index types	GIN and GiST	GIN
Operators	`LIKE` | `ILIKE` | `~` | `~*`	`LIKE`
Non-alphabet full-text search	Not supported	Supported

Functionality	pg_trgm	pg_bigm
Full-text search with keywords that contain 1 to 2 characters	Slow	Fast
Similarity search	Supported	Supported
Maximum indexed column length	238,609,291 bytes (approximately equal to 228 MB)	107,374,180 bytes (approximately equal to 102 MB)

Precautions

- The length of the column on which you create a GIN index cannot exceed 107,374,180 bytes (approximately equal to 102 MB).
- If the data in your RDS instance is not encoded by using ASCII, we recommend that you change the encoding format to UTF8.

> **Note** To query the encoding format of your RDS instance, run the `select pg_encoding_to_char(encoding) from pg_database where datname = current_database();` command.

Basic operations

- Create the pg_bigm plug-in.

```
postgres=> create extension pg_bigm;
CREATE EXTENSION
```

- Create a GIN index.

```
postgres=> CREATE TABLE pg_tools (tool text, description text);
CREATE TABLE
postgres=> INSERT INTO pg_tools VALUES ('pg_hint_plan', 'Tool that allows a user to specify
an optimizer HINT to PostgreSQL');
INSERT 0 1
postgres=> INSERT INTO pg_tools VALUES ('pg_dbms_stats', 'Tool that allows a user to stabil
ize planner statistics in PostgreSQL');
INSERT 0 1
postgres=> INSERT INTO pg_tools VALUES ('pg_bigm', 'Tool that provides 2-gram full text sea
rch capability in PostgreSQL');
INSERT 0 1
postgres=> INSERT INTO pg_tools VALUES ('pg_trgm', 'Tool that provides 3-gram full text sea
rch capability in PostgreSQL');
INSERT 0 1
postgres=> CREATE INDEX pg_tools_idx ON pg_tools USING gin (description gin_bigm_ops);
CREATE INDEX
postgres=> CREATE INDEX pg_tools_multi_idx ON pg_tools USING gin (tool gin_bigm_ops, descri
ption gin_bigm_ops) WITH (FASTUPDATE = off);
CREATE INDEX
```

- Run a full-text search query.

```
postgres=> SELECT * FROM pg_tools WHERE description LIKE '%search%';
  tool   |                            description
---------+-----------------------------------------------------------------
 pg_bigm | Tool that provides 2-gram full text search capability in PostgreSQL
 pg_trgm | Tool that provides 3-gram full text search capability in PostgreSQL
(2 rows)
```

- Run a similarity search query by using the `=%` operator.

```
postgres=> SET pg_bigm.similarity_limit TO 0.2;
SET
postgres=> SELECT tool FROM pg_tools WHERE tool =% 'bigm';
  tool
---------
 pg_bigm
 pg_trgm
(2 rows)
```

- Delete the pg_bigm plug-in.

```
postgres=> drop extension pg_bigm;
DROP EXTENSION
```

Basic functions

- likequery
  - Purpose: This function is used to generate a string that can be identified based on the LIKE keyword.
  - Request parameters: This function contains one request parameter. The data type for this parameter is STRING.
  - Return value: This function returns a string that can be identified based on the LIKE keyword.
  - Implementation:
    - Add a percent sign (`%`) preceding and following the keyword.
    - Use a backward slash (`\`) to escape the percent sign (`%`).
  - Example:

```
postgres=> SELECT likequery('pg_bigm has improved the full text search performance by 200
%');
                            likequery
------------------------------------------------------------------
 %pg\_bigm has improved the full text search performance by 200\%%
(1 row)
postgres=> SELECT * FROM pg_tools WHERE description LIKE likequery('search');
  tool   |                            description
---------+-----------------------------------------------------------------
 pg_bigm | Tool that provides 2-gram full text search capability in PostgreSQL
 pg_trgm | Tool that provides 3-gram full text search capability in PostgreSQL
(2 rows)
```

- show_bigm
  - Purpose: This function is used to obtain all of the 2-gram elements that comprise a string.
  - Request parameters: This function contains one request parameter. The data type for this parameter is STRING.

- Return value: This parameter returns an array that consists of all the 2-gram elements of a string.
- Implementation:
  - Add a space preceding and following the string.
  - Identify all of the 2-gram elements in the string.
- Example:

```
postgres=> SELECT show_bigm('full text search');
                           show_bigm
-----------------------------------------------------------------
 {" f"," s"," t",ar,ch,ea,ex,fu,"h ","l ",ll,rc,se,"t ",te,ul,xt}
(1 row)
```

- bigm_similarity
  - Purpose: This function is used to obtain the similarity between two strings.
  - Request parameters: This function contains two request parameters. The data types for these parameters are STRING.
  - Return value: This function returns a floating-point number. The number indicates the similarity between the two strings.
  - Implementation:
    - Identify the 2-gram elements that are included in both the two strings.
    - The return value is within the range from 0 to 1. The value 0 indicates that the two strings are completely different. The value 1 indicates that the two strings are identical.

> **Note**
>
> - This function adds a space preceding and following each string. Therefore, the similarity between the `ABC` string and the `B` string is 0, and the similarity between the `ABC` string and the `A` string is 0.25.
> - This function supports case sensitivity. For example, it determines that the similarity between the `ABC` string and the `abc` string is 0.

- Example:

```
postgres=> SELECT bigm_similarity('full text search', 'text similarity search');
 bigm_similarity
-----------------
       0.5714286
(1 row)
postgres=> SELECT bigm_similarity('ABC', 'A');
 bigm_similarity
-----------------
            0.25
(1 row)
postgres=> SELECT bigm_similarity('ABC', 'B');
 bigm_similarity
-----------------
               0
(1 row)
postgres=> SELECT bigm_similarity('ABC', 'abc');
 bigm_similarity
-----------------
               0
(1 row)
```

- pg_gin_pending_stats
  - Purpose: This function is used to obtain the number of pages and the number of tuples in the pending list of a GIN index.
  - Request parameters: This function contains one parameter. This parameter specifies the name or OID of the GIN index.
  - Return value: This function returns two values: the number of pages in the pending list of the GIN index and the number of tuples in the pending list of the GIN index.

    > **Note** If you set the FASTUPDATE parameter to False for a GIN index, the GIN index does not have a pending list. In this case, this function returns two values 0.

  - Example:

```
postgres=> SELECT * FROM pg_gin_pending_stats('pg_tools_idx');
 pages | tuples
-------+--------
     0 |      0
(1 row)
```

Behavior control

- pg_bigm.last_update

  This parameter indicates the last date when the pg_bigm plug-in was updated. This parameter is read-only. You cannot reconfigure this parameter.

  Example:

```
SHOW pg_bigm.last_update;
```

- pg_bigm.enable_recheck

This parameter specifies whether to perform a recheck.

> **Note** We recommend that you retain the default value ON. This allows you to obtain accurate query results.

Example:

```
postgres=> CREATE TABLE tbl (doc text);
CREATE TABLE
postgres=> INSERT INTO tbl VALUES('He is awaiting trial');
INSERT 0 1
postgres=> INSERT INTO tbl VALUES('It was a trivial mistake');
INSERT 0 1
postgres=> CREATE INDEX tbl_idx ON tbl USING gin (doc gin_bigm_ops);
CREATE INDEX
postgres=> SET enable_seqscan TO off;
SET
postgres=> EXPLAIN ANALYZE SELECT * FROM tbl WHERE doc LIKE likequery('trial');
                                                    QUERY PLAN
------------------------------------------------------------------------------------------------
----------------------
 Bitmap Heap Scan on tbl  (cost=20.00..24.01 rows=1 width=32) (actual time=0.020..0.021 row
s=1 loops=1)
   Recheck Cond: (doc ~~ '%trial%'::text)
   Rows Removed by Index Recheck: 1
   Heap Blocks: exact=1
   ->  Bitmap Index Scan on tbl_idx  (cost=0.00..20.00 rows=1 width=0) (actual time=0.013..
0.013 rows=2 loops=1)
         Index Cond: (doc ~~ '%trial%'::text)
 Planning Time: 0.117 ms
 Execution Time: 0.043 ms
(8 rows)
postgres=>
postgres=> SELECT * FROM tbl WHERE doc LIKE likequery('trial');
         doc
----------------------
 He is awaiting trial
(1 row)
postgres=> SET pg_bigm.enable_recheck = off;
SET
postgres=> SELECT * FROM tbl WHERE doc LIKE likequery('trial');
           doc
--------------------------
 He is awaiting trial
 It was a trivial mistake
(2 rows)
```

- pg_bigm.gin_key_limit

  This parameter specifies the maximum number of 2-gram elements that can be used for a full-text search query. The default value is 0, which indicates that all 2-gram elements are used.

  > **Note** If the use of all 2-gram elements triggers a performance decrease, you can decrease the value of this parameter.

- pg_bigm.similarity_limit

  This parameter specifies the threshold for similarity. The tuples whose similarity exceeds the specified threshold are returned as similarity search results.

6.9. Use the wal2json plug-in

This topic describes how to use the wal2json plug-in provided by RDS PostgreSQL to export logical log records as a file in JSON format.

Prerequisites

- Your RDS instance runs PostgreSQL 10, 11, 12, 13 or 14.
- The wal_level parameter is set to logical. For more information, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

Context

wal2json is a logical decoding plug-in. It has access to tuples generated by INSERT and UPDATE statements and can parse log records produced by write-ahead logging (WAL).

The wal2json plug-in produces a JSON object for each transaction. All new and old tuples are available in the JSON object. In addition, there are options to include properties such as transaction timestamp, schema-qualified, data type, and transaction ID. You can obtain JSON objects by executing SQL statements. For more information, see Execute SQL statements to obtain JSON objects.

Execute SQL statements to obtain JSON objects

1. Use DMS to log on to an ApsaraDB RDS for PostgreSQL instance.
2. Execute the following statements to create a table and initialize the wal2json plug-in:

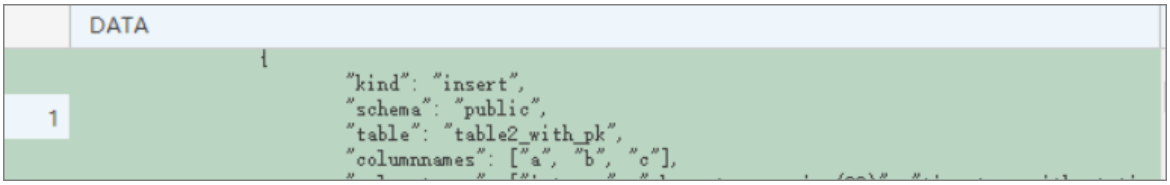
```
CREATE TABLE table2_with_pk (a SERIAL, b VARCHAR(30), c TIMESTAMP NOT NULL, PRIMARY KEY(a
, c));
CREATE TABLE table2_without_pk (a SERIAL, b NUMERIC(5,2), c TEXT);
SELECT 'init' FROM pg_create_logical_replication_slot('test_slot', 'wal2json');
```

3. Execute the following statements to change data:

```
BEGIN;
INSERT INTO table2_with_pk (b, c) VALUES('Backup and Restore', now());
INSERT INTO table2_with_pk (b, c) VALUES('Tuning', now());
INSERT INTO table2_with_pk (b, c) VALUES('Replication', now());
DELETE FROM table2_with_pk WHERE a < 3;
INSERT INTO table2_without_pk (b, c) VALUES(2.34, 'Tapir');
UPDATE table2_without_pk SET c = 'Anta' WHERE c = 'Tapir';
COMMIT;
```

4. Execute the following statement to produce logical log records in JSON format:

```
SELECT data FROM pg_logical_slot_get_changes('test_slot', NULL, NULL, 'pretty-print', '1'
);
```



Note If you want to stop producing logical log records and release the resources used, execute the following statement:

```
SELECT 'stop' FROM pg_drop_replication_slot('test_slot');
```

6.10. Use the ZomboDB plug-in to integrate with Elasticsearch

ZomboDB is a PostgreSQL extension plug-in. It supports the access methods that are provided by native PostgreSQL. It also provides powerful text search and analytics features by using Elasticsearch.

Prerequisites

Your ApsaraDB for RDS instance runs PostgreSQL 11.

Context

ZomboDB provides a full set of query languages to query relational data. You can also create ZomboDB indexes. In this case, ZomboDB takes over remote Elasticsearch indexes and ensures transactionally correct query results from text search.

ZomboDB allows you to use Elasticsearch without the need to handle synchronization or communication issues.

Create and delete the ZomboDB plug-in

- Create the plug-in

```
CREATE EXTENSION zombodb;
```

- Delete the plug-in

```
DROP EXTENSION zombodb;
```

Examples

1. Create a table.

```
CREATE TABLE products (
    id SERIAL8 NOT NULL PRIMARY KEY,
    name text NOT NULL,
    keywords varchar(64)[],
    short_summary text,
    long_description zdb.fulltext,
    price bigint,
    inventory_count integer,
    discontinued boolean default false,
    availability_date date
);
```

2. Create a ZomboDB index for the table.

```
CREATE INDEX idxproducts
          ON products
       USING zombodb ((products. *))
        WITH (url='localhost:9200/');
```

> **Note** The WITH clause is followed by an Elasticsearch endpoint, which points to a running Elasticsearch cluster.

3. Query data by using the ZomboDB index.

```
SELECT *
  FROM products
 WHERE products ==> '(keywords:(sports OR box) OR long_description:"wooden away"~5) AND p
rice:[1000 TO 20000]';
```

> **Note** For more information about the query syntax, see ZomboDB Documentation.

6.11. Use the sql_firewall plug-in

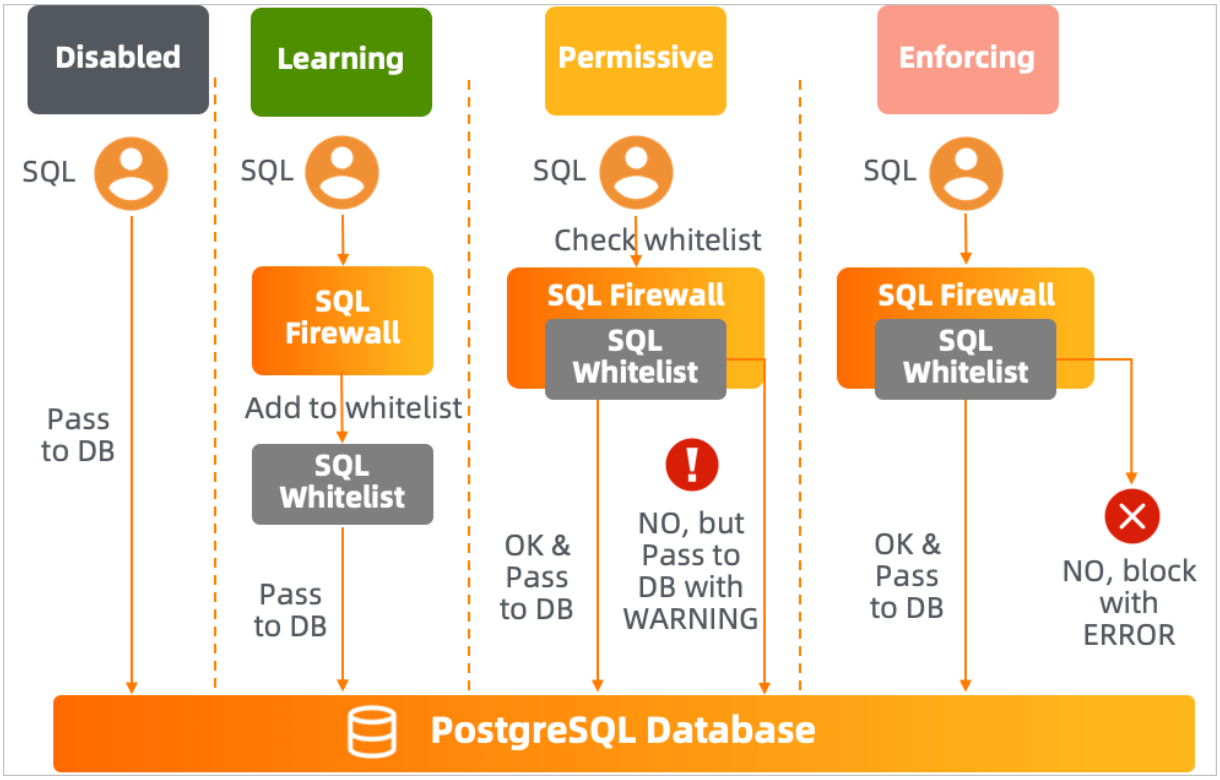
The sql_firewall plug-in is a database-level firewall that prevents against SQL injection. It learns defined SQL rules and stores the rules in your ApsaraDB RDS for PostgreSQL instance as a whitelist. User operations that do not comply with the rules are forbidden.

Prerequisites

Your RDS instance runs one of the following RDS editions:

- PostgreSQL 12
- PostgreSQL 11
- PostgreSQL 10

Learning, permissive, and enforcing modes



The sql_firewall plug-in supports the following modes:

- Learning: The plug-in records common SQL statements that are executed and adds them to a whitelist.
- Permissive: The plug-in checks SQL statements that will be executed. If the SQL statements are not in the whitelist, the plug-in executes the SQL statements but generates alerts.
- Enforcing: The plug-in checks SQL statements that will be executed. If the SQL statements are not in the whitelist, the plug-in does not execute the SQL statements and returns errors.

Procedure

1. Enable the learning mode of the sql_firewall plug-in. Wait for a specific period of time to ensure that the plug-in learns more SQL statements.
2. Switch the sql_firewall plug-in to the permissive mode. In this mode, the plug-in generates alerts for SQL statements that are not in the whitelist. You can check whether these SQL statements are high-risky statements based on your business requirements. If these statements are not high-risky statements, switch to the learning mode and add these SQL statements to the whitelist.
3. Switch the sql_firewall plug-in to the enforcing mode. In this mode, the plug-in does not execute SQL statements that are not in the whitelist.

Operations

- Create the plug-in

```
create extension sql_firewall;
```

- Delete the plug-in

```
drop extension sql_firewall;
```

- Switch the mode

In the ApsaraDB for RDS console, find the **sql_firewall.firewall** parameter. Modify the parameter value and restart your RDS instance. For more information, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

Valid values for the **sql_firewall.firewall** parameter:

- disable: disables the sql_firewall plug-in.
- learning: enables the learning mode.
- permissive: enables the permissive mode.
- enforcing: enables the enforcing mode.

- Functionality functions
  - sql_firewall_reset()

    This function clears the whitelist. You can call this function only if you are authorized with the rds_superuser role and the enforcing mode is disabled.
  - sql_firewall_stat_reset()

    This function deletes statistics. You can call this function only if you are authorized with the rds_superuser role and the enforcing mode is disabled.
- View functions
  - sql_firewall.sql_firewall_statements

    This function returns all SQL statements in the whitelist of your RDS instance. This function also returns the number of times that each SQL statement is executed.

    ```
    postgres=# select * from sql_firewall.sql_firewall_statements;
         userid |  queryid   |              query              | calls
        --------+------------+---------------------------------+-------
             10 | 3294787656 | select * from k1 where uid = ? ; |     4
        (1 row)
    ```
  - sql_firewall.sql_firewall_stat

    This function returns the number of alerts that are generated in permissive mode and the number of errors that are generated in enforcing mode. The first number is measured by sql_warning, and the second number is measured by sql_error.

    ```
    postgres=# select * from sql_firewall.sql_firewall_stat;
         sql_warning | sql_error
        -------------+-----------
                   2 |         1
        (1 row)
    ```

Examples

```
-- Permissive mode
    postgres=# select * from sql_firewall.sql_firewall_statements;
    WARNING:  Prohibited SQL statement
     userid |  queryid   |              query              | calls
    --------+------------+---------------------------------+-------
         10 | 3294787656 | select * from k1 where uid = 1; |     1
    (1 row)
    postgres=# select * from k1 where uid = 1;
     uid |    uname
    -----+-------------
       1 | Park Gyu-ri
    (1 row)
    postgres=# select * from k1 where uid = 3;
     uid |   uname
    -----+-----------
       3 | Goo Ha-ra
    (1 row)
    postgres=# select * from k1 where uid = 3 or 1 = 1;
    WARNING:  Prohibited SQL statement
     uid |     uname
    -----+----------------
       1 | Park Gyu-ri
       2 | Nicole Jung
       3 | Goo Ha-ra
       4 | Han Seung-yeon
       5 | Kang Ji-young
    (5 rows)
-- Enforcing mode
    postgres=# select * from k1 where uid = 3;
     uid |   uname
    -----+-----------
       3 | Goo Ha-ra
    (1 row)
    postgres=# select * from k1 where uid = 3 or 1 = 1;
    ERROR:  Prohibited SQL statement
    postgres=#
```

6.12. Use the pg_concurrency_control plug-in

ApsaraDB RDS for PostgreSQL provides a pg_concurrency_control plug-in to control concurrency of SQL statements.

Prerequisites

Your RDS instance runs PostgreSQL 10 or 11.

Parameters

Note The following parameters cannot be modified in the ApsaraDB RDS console. You can submit a to modify them.

Parameter	Default value	Description
pg_concurrency_control.query_concurrency	0	Sets the maximum number of concurrent jobs in SELECT SQL statements. Valid values: 0 to 1024. Default value: 0. The default value indicates that concurrency control is disabled for SELECT SQL statements.
pg_concurrency_control.bigquery_concurrency	0	Sets the maximum number of concurrent jobs in slow queries. Valid values: 0 to 1024. Default value: 0. The default value indicates that concurrency control is disabled for slow queries. You can specify a statement as a slow query by using `hint "/*+bigsql*/"`. Example: `/*+bigsql*/select * from test;` The `select * from test;` statement is a slow query.
pg_concurrency_control.transaction_concurrency	0	Sets the maximum number of concurrent jobs for transaction blocks. Valid values: 0 to 1024. Default value: 0. The default value indicates that concurrency control is disabled for transaction blocks.
pg_concurrency_control.autocommit_concurrency	0	Sets the maximum number of concurrent jobs in DML SQL statements. Valid values: 0 to 1024. Default value: 0. The default value indicates that concurrency control is disabled for DML SQL statements.
pg_concurrency_control.control_timeout	1s	Sets the maximum time to wait for a SELECT SQL statement, DML SQL statement, and transaction block. The minimum value is 30 ms, and the maximum value is 3s.
pg_concurrency_control.bigsql_control_timeout	1s	Sets the maximum time to wait for a slow query. The minimum value is 30 ms, and the maximum value is 3s.
pg_concurrency_control.timeout_action	TCC_break	Sets the action upon a timeout for a SELECT SQL statement, DML SQL statement, and transaction block. Valid values: • TCC_break: skips the statement in waiting and execute the statement that follows. • TCC_rollback: reports an error and rolls back the transaction. • TCC_wait: resets the timestamp after a timeout and continues to wait.

Parameter	Default value	Description
pg_concurrency_control.bigsql_timeout_action	TCC_wait	Sets the action upon a timeout for slow queries. Valid values: • TCC_break: skips the statement in waiting and execute the statement that follows. • TCC_rollback: reports an error and rolls back the transaction. • TCC_wait: resets the timestamp after a timeout and continues to wait.

Procedure

1. Run the following command to create the plug-in:

```
create extension pg_concurrency_control;
```

2. Set the number of concurrent jobs to a value that is greater than 0 to enable concurrency control in the plug-in.

   For example, set the pg_concurrency_control.query_concurrency parameter to 10 to enable concurrency control for SELECT SQL statements. The methods to enable concurrency control for other types of statements are similar.

   > **Note** The parameters cannot be modified in the ApsaraDB RDS console. You can submit a to modify them.

Example

Perform the following operations to enable concurrency control for custom SQL statements:

1. Run the following command to view information of the statement queue:

```
select * from pg_concurrency_control_status();
```

   The system displays information similar to the following output:

```
 autocommit_count | bigquery_count | query_count | transaction_count
------------------+----------------+-------------+-------------------
                0 |              0 |           0 |                 0
(1 row)
```

2. Set the pg_concurrency_control.query_concurrency parameter to a value that is greater than 0, for example, 10.
3. Execute a slow query.

```
/*+ bigsql */ select pg_sleep(10);
```

4. Run the following command to view information of the statement queue again:

```
select * from pg_concurrency_control_status();
```

   The system displays information similar to the following output:

```
 autocommit_count | bigquery_count | query_count | transaction_count
------------------+----------------+-------------+-------------------
                0 |              1 |           0 |                 0
(1 row)
```

> **Note** After the slow query is complete, the queue information is automatically cleared.

6.13. Use the RUM plug-in

This topic describes how to use the RUM plug-in of ApsaraDB RDS for PostgreSQL to run full-text searches.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 13
- PostgreSQL 12
- PostgreSQL 11
- PostgreSQL 10

Context

Generalized Inverted Index (GIN) allows you to run full-text searches by using the tsvector and tsquery data types. However, this may produce the following issues:

- Slow sorting

  ApsaraDB RDS for PostgreSQL can sort words only after it obtains the locations of the words. However, GIN does not store word locations. As a result, after ApsaraDB RDS for PostgreSQL runs a scan based on a GIN index, it must run another scan to retrieve the word locations.

- Slow queries for phrases

  GIN can search for phrases only after it obtains the locations of the phrases.

- Slow sorting of timestamps

  GIN does not store related information in indexes that contain morphemes. Therefore, an additional scan is required.

The RUM plug-in of ApsaraDB RDS for PostgreSQL is designed based on GIN. It allows you to store word or timestamp locations in RUM indexes.

However, the RUM plug-in requires more time than GIN to construct and insert indexes. This is because the RUM plug-in generates indexes based on write-ahead logging (WAL) logs and the generated RUM indexes contain more information than the keys that are used for encryption.

Universal operators

The RUM plug-in provides the following operators.

Operator	Data type	Description
tsvector <=> tsquery	float4	Returns the distances between the data objects of the tsvector type and those of the tsquery type.
timestamp <=> timestamp	float8	Returns the distance between two timestamps.

Operator	Data type	Description
timestamp <=| timestamp	float8	Returns only the distance to the left-side timestamp.
timestamp |=> timestamp	float8	Returns only the distance to the right-side timestamp.

Note The last three operators are also supported for the following data types: timestamptz, int2, int4, int8, float4, float8, money, and oid.

The following sections describe the functions that are provided by the RUM plug-in.

rum_tsvector_ops

- Supported data types: tsvector.
- Description: This function stores phrases of the tsvector data type along with the locations of the phrases. This function allows you to sort phrases by using the `<=>` operator. This function also allows you to search for phrases based on prefixes.
- Examples:
  i. Execute the following statements to create a table:

```
CREATE TABLE test_rum(t text, a tsvector);
CREATE TRIGGER tsvectorupdate
BEFORE UPDATE OR INSERT ON test_rum
FOR EACH ROW EXECUTE PROCEDURE tsvector_update_trigger('a', 'pg_catalog.english', 't');
INSERT INTO test_rum(t) VALUES ('The situation is most beautiful');
INSERT INTO test_rum(t) VALUES ('It is a beautiful');
INSERT INTO test_rum(t) VALUES ('It looks like a beautiful place');
```

  ii. Execute the following statement to create the RUM plug-in:

```
CREATE EXTENSION rum;
```

  iii. Execute the following statement to create an index:

```
CREATE INDEX rumidx ON test_rum USING rum (a rum_tsvector_ops);
```

  iv. Run the following two types of queries:

```
SELECT t, a <=> to_tsquery('english', 'beautiful | place') AS rank
    FROM test_rum
    WHERE a @@ to_tsquery('english', 'beautiful | place')
    ORDER BY a <=> to_tsquery('english', 'beautiful | place');
```

  The following result is returned:

```
                t                |  rank
---------------------------------+---------
 It looks like a beautiful place | 8.22467
 The situation is most beautiful | 16.4493
 It is a beautiful               | 16.4493
(3 rows)
```

```
SELECT t, a <=> to_tsquery('english', 'place | situation') AS rank
    FROM test_rum
    WHERE a @@ to_tsquery('english', 'place | situation')
    ORDER BY a <=> to_tsquery('english', 'place | situation');
```

The following result is returned:

```
                t                |  rank
---------------------------------+---------
 The situation is most beautiful | 16.4493
 It looks like a beautiful place | 16.4493
(2 rows)
```

rum_tsvector_hash_ops

- Supported data types: tsvector.
- Description: This function stores phrases of the tsvector data type along with the hash values and locations of the phrases. This function allows you to sort phrases by using the `<=>` operator. However, this function does not allow you to search for phrases based on prefixes.

rum_TYPE_ops

- Supported data types:
  - The `<`, `<=`, `=`, `>=`, `>`, and `<=>` operators support the following data types: int2, int4, int8, float4, float8, money, oid, time, timetz, date, interval, macaddr, inet, cidr, text, varchar, char, bytea, bit, varbit, numeric, timestamp, and timestamptz.
  - The `<=|` and `|=>` operators support the following data types: int2, int4, int8, float4, float8, money, oid, timestamp, and timestamptz.
- Description: This function can be used with the rum_tsvector_addon_ops, rum_tsvector_hash_addon_ops, and rum_anyarray_addon_ops functions.

rum_tsvector_addon_ops

- Supported data types: tsvector.
- Description: This function stores the word segmentation method that is used by the tsvector data type. This method also stores all the other word segmentation methods that are supported by the fields of the RUM plug-in.
- Examples:
  i. Execute the following statements to create a table with an index:

```
CREATE TABLE tsts (id int, t tsvector, d timestamp);
\copy tsts from 'rum/data/tsts.data'
CREATE INDEX tsts_idx ON tsts USING rum (t rum_tsvector_addon_ops, d)
    WITH (attach = 'd', to = 't');
```

  ii. Run the following query:

```
EXPLAIN (costs off)
    SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tsts WHERE t @@ 'wr&qh' ORDER BY d <
=> '2016-05-16 14:21:25' LIMIT 5;
```

The following result is returned:

```
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Limit
   ->  Index Scan using tsts_idx on tsts
         Index Cond: (t @@ '''wr'' & ''qh'''::tsquery)
         Order By: (d <=> 'Mon May 16 14:21:25 2016'::timestamp without time zone)
(4 rows)
```

iii. Run the following query:

```
SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tsts WHERE t @@ 'wr&qh' ORDER BY d <=> '
2016-05-16 14:21:25' LIMIT 5;
```

The following result is returned:

```
 id  |               d                |   ? column?
-----+--------------------------------+---------------
 355 | Mon May 16 14:21:22.326724 2016 |      2.673276
 354 | Mon May 16 13:21:22.326724 2016 |   3602.673276
 371 | Tue May 17 06:21:22.326724 2016 |  57597.326724
 406 | Wed May 18 17:21:22.326724 2016 | 183597.326724
 415 | Thu May 19 02:21:22.326724 2016 | 215997.326724
(5 rows)
```

> **Note** When the RUM plug-in creates an index on a table whose data is sorted based on referenced additional information, errors may occur. This is because the RUM plug-in uses a suffix tree that has the following limits: Both edges and suffixes must be of a fixed length, and suffixes cannot have child nodes.

rum_tsvector_hash_addon_ops

- Supported data types: tsvector.
- Description: This function stores the hash values of the word libraries that are used by the tsvector data type. This function also stores the fields of the RUM plug-in. However, this function does not support prefix-based searches.

rum_tsquery_ops

- Supported data types: tsquery.
- Description: This function stores the branches of query trees.
- Examples:
  i. Execute the following statements to create a table with an index:

```
CREATE TABLE test_array (i int2[]);
INSERT INTO test_array VALUES ('{}'), ('{0}'), ('{1,2,3,4}'), ('{1,2,3}'), ('{1,2}'), (
'{1}');
CREATE INDEX idx_array ON test_array USING rum (i rum_anyarray_ops);
```

ii. Run the following query:

```
SET enable_seqscan TO off;
EXPLAIN (COSTS OFF) SELECT * FROM test_array WHERE i && '{1}' ORDER BY i <=> '{1}' ASC;
```

The following result is returned:

```
                QUERY PLAN
------------------------------------------
 Index Scan using idx_array on test_array
   Index Cond: (i && '{1}'::smallint[])
   Order By: (i <=> '{1}'::smallint[])
(3 rows)
```

iii. Run the following query:

```
SELECT * FROM test_array WHERE i && '{1}' ORDER BY i <=> '{1}' ASC;
```

The following result is returned:

```
     i
-----------
 {1}
 {1,2}
 {1,2,3}
 {1,2,3,4}
(4 rows)
```

rum_anyarray_ops

- Supported data types: anyarray.
- Description: This function stores anyarray elements at lengths that are similar to the lengths of arrays. This function supports the following operators: `&&`, `@>`, `<@`, `=`, and `%`. This function also allows you to sort data by using the `<=>` operator.
- Examples:

  i. Execute the following statements to create a table with an index:

```
CREATE TABLE test_array (i int2[]);
INSERT INTO test_array VALUES ('{}'), ('{0}'), ('{1,2,3,4}'), ('{1,2,3}'), ('{1,2}'), (
'{1}');
CREATE INDEX idx_array ON test_array USING rum (i rum_anyarray_ops);
```

  ii. Run the following query:

```
SET enable_seqscan TO off;
EXPLAIN (COSTS OFF) SELECT * FROM test_array WHERE i && '{1}' ORDER BY i <=> '{1}' ASC;
```

  The following result is returned:

```
                QUERY PLAN
------------------------------------------
 Index Scan using idx_array on test_array
   Index Cond: (i && '{1}'::smallint[])
   Order By: (i <=> '{1}'::smallint[])
(3 rows)
```

  iii. Run the following query:

```
SELECT * FROM test_array WHERE i && '{1}' ORDER BY i <=> '{1}' ASC;
```

The following result is returned:

```
     i
-----------
 {1}
 {1,2}
 {1,2,3}
 {1,2,3,4}
(4 rows)
```

rum_anyarray_addon_ops

- Supported data types: anyarray.
- Description: This function stores anyarray elements. This function also stores all the elements that are supported for the fields of the RUM plug-in.

6.14. Use the zhparser plug-in

This topic describes how to use the zhparser plug-in to implement Chinese full-text searches in ApsaraDB RDS for PostgreSQL.

Context

Open source PostgreSQL provides a built-in parser plug-in that splits text into tokens. The parser plug-in is suitable for languages such as English. These languages support tokenization by punctuation or space. However, the Chinese language does not contain spaces and is length-variable. In addition, the tokenization of Chinese text is based on semantics. Therefore, the parser plug-in cannot be used for the tokenization of Chinese text.

The zhparser plug-in provided with open source PostgreSQL is used for the tokenization of Chinese text. After this plug-in is installed on your ApsaraDB RDS for PostgreSQL instance, you can implement Chinese full-text searches.

Enable the zhparser plug-in

Execute the following statements to enable the zhparser plug-in:

```
CREATE EXTENSION zhparser;
CREATE TEXT SEARCH CONFIGURATION testzhcfg (PARSER = zhparser);
ALTER TEXT SEARCH CONFIGURATION testzhcfg ADD MAPPING FOR n,v,a,i,e,l WITH simple;
--Optional parameter configuration
alter role all set zhparser.multi_short=on;
--Perform a simple test
SELECT * FROM ts_parse('zhparser', 'hello world! 2010年保障房建设在全国范围内获全面启动，从中央到地方纷纷加大了保障房的建设和投入力度。 2011年，保障房进入了更大规模的建设阶段。 住房城乡建设部党组书记、部长姜伟新去年底在全国住房城乡建设工作会议上表示，要继续推进保障性安居工程建设。') ;
SELECT to_tsvector('testzhcfg','“今年保障房新开工数量虽然有所下调，但实际的年度在建规模以及竣工规模会超以往年份，相对应的对资金的需求也会创历史纪录。” 陈国强说。 在他看来，与2011年相比，2012年的保障房建设在资金配套上的压力将更为严峻。') ;
SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

Execute the following statements to use the zhparser plug-in to run a full-text index:

```
--Create a full-text index for the name field of table T1
create index idx_t1 on t1 using gin (to_tsvector('zhcfg',upper(name) ));
--Use the full-text index
 select * from t1 where to_tsvector('zhcfg',upper(t1.name)) @@ to_tsquery('zhcfg','(防火)') ;
```

Customize a Chinese word segment dictionary

Execute the following statements to customize a Chinese word segment dictionary

```
-- The segmentation result
SELECT to_tsquery('testzhcfg', '保障房资金压力');
-- Insert a new word segment to the dictionary
insert into pg_ts_custom_word values ('保障房资');
-- Make the inserted word segment take effect
select zhprs_sync_dict_xdb();
-- End the connection
\c
-- Requery to obtain new segmentation results
SELECT to_tsquery('testzhcfg', '保障房资金压力');
```

Instructions to use custom word segments:

- A maximum of 1 million custom word segments can be added. If the number of word segments exceed the limit, the word segments outside the limit are not processed. Ensure that the number of word segments is within this range. The custom and default word segmentation dictionaries take effect at the same time.
- Each word segment can be a maximum of 128 bytes in length. The section after the 128th byte will be truncated.
- After adding, deleting, or changing word segments, execute the `select zhprs_sync_dict_xdb();` statement and re-establish a connection to make the operation take effect.

6.15. Use the pg_jieba plug-in to run full-text searches in Chinese

This topic describes how to run full-text searches in Chinese to query data from an ApsaraDB RDS for PostgreSQL instance by using the pg_jieba plug-in.

Prerequisites

- Your RDS instance runs one of the following database engine versions:
  - Major engine version: PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13.
  - Minor engine version: 20211130 or later. For more information about how to update the minor engine version, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.
- pg_jieba is added to the value of the shared_preload_libraries parameter of your RDS instance.

  For more information about how to add **pg_jieba** to the value of the **shared_preload_libraries** parameter, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

Procedure

- Create the pg_jieba plug-in.

```
CREATE EXTENSION pg_jieba;
```

> **Note** Only privileged accounts are authorized to run the preceding command.

- Delete the pg_jieba plug-in.

```
DROP EXTENSION pg_jieba;
```

> **Note** Only privileged accounts are authorized to run the preceding command.

- Example 1:

```
select * from to_tsvector('jiebacfg', '小明硕士毕业于中国科学院计算所，后在日本京都大学深造');
                                                to_tsvector
-----------------------------------------------------------------------------------------------------
-------------------
 '中国科学院':5 '于':4 '后':8 '在':9 '小明':1 '日本京都大学':10 '毕业':3 '深造':11 '硕士':2 '计算
所':6 '，':7
(1 row)
```

- Example 2:

```
select * from to_tsvector('jiebacfg', '李小福是创新办主任也是云计算方面的专家');
                                      to_tsvector
-----------------------------------------------------------------------------------------------
 '专家':11 '主任':5 '也':6 '云计算':8 '创新':3 '办':4 '方面':9 '是':2,7 '李小福':1 '的':10
(1 row)
```

6.16. Use the fuzzystrmatch plug-in to compute similarity between strings

ApsaraDB RDS for PostgreSQL provides the fuzzystrmatch plug-in. This plug-in supports the Soundex, Levenshtein, Metaphone, and Double Metaphone algorithms. You can use these algorithms to calculate the similarity and distance between strings.

Soundex

The Soundex algorithm converts similar-sounding words into the same code. However, this algorithm is unsuitable for non-English words.

The Soundex algorithm provides the following functions:

```
soundex(text) returns text
difference(text, text) returns int
```

- The soundex function converts a string into its Soundex code, such as A550.
- The difference function converts two strings into their Soundex codes. Then, the difference function reports the number of code matching positions between the two strings. A Soundex code consists of four characters. Therefore, the number of code matching positions ranges from 0 to 4. The value 0 indicates a zero match, and the value 4 indicates an exact match.

Examples:

```
SELECT soundex('hello world!') ;
SELECT soundex('Anne'), soundex('Andrew'), difference('Anne', 'Andrew');
SELECT soundex('Anne'), soundex('Margaret'), difference('Anne', 'Margaret');
CREATE TABLE s (nm text);
INSERT INTO s VALUES ('john');
INSERT INTO s VALUES ('joan');
INSERT INTO s VALUES ('wobbly');
INSERT INTO s VALUES ('jack');
SELECT * FROM s WHERE soundex(nm) = soundex('john');
SELECT * FROM s WHERE difference(s.nm, 'john') > 2;
```

Levenshtein

The Levenshtein algorithm calculates the Levenshtein distance between two strings.

The Levenshtein algorithm provides the following functions:

```
levenshtein(text source, text target, int ins_cost, int del_cost, int sub_cost) returns int
levenshtein(text source, text target) returns int
levenshtein_less_equal(text source, text target, int ins_cost, int del_cost, int sub_cost, in
t max_d) returns int
levenshtein_less_equal(text source, text target, int max_d) returns int
```

The following table describes the parameters that you must configure in the preceding functions.

Parameter	Description
source	The first string to compare. The string cannot be empty and can contain up to 255 characters in length.
target	The second string to compare. The string cannot be empty and can contain up to 255 characters in length.
ins_cost	The overhead that is required to insert characters.
del_cost	The overhead that is required to delete characters.
sub_cost	The overhead that is required to replace characters.
max_d	The maximum Levenshtein distance that is allowed between the two specified strings.

Note The levenshtein_less_equal function is an accelerated version of the levenshtein function. It is used only to calculate a short Levenshtein distance:

- If the actual distance is less than or equal to the value of the max_d parameter, the levenshtein_less_equal function returns the exact distance that is calculated.
- If the actual distance is greater than the value of the max_d parameter, the levenshtein_less_equal function returns a random distance that is greater than the value of the max_d parameter.
- If the value of the max_d parameter is negative, the levenshtein_less_equal and levenshtein functions return the same distance.

Examples:

```
SELECT levenshtein('GUMBO', 'GAMBOL');
SELECT levenshtein('GUMBO', 'GAMBOL', 2,1,1);
SELECT levenshtein_less_equal('extensive', 'exhaustive',2);
SELECT levenshtein_less_equal('extensive', 'exhaustive',4);
```

```
test=# SELECT levenshtein('GUMBO', 'GAMBOL');
 levenshtein
-------------
           2
(1 row)

test=# SELECT levenshtein('GUMBO', 'GAMBOL', 2,1,1);
 levenshtein
-------------
           3
(1 row)

test=# SELECT levenshtein_less_equal('extensive', 'exhaustive',2);
 levenshtein_less_equal
------------------------
                      3
(1 row)

test=# SELECT levenshtein_less_equal('extensive', 'exhaustive',4);
 levenshtein_less_equal
------------------------
                      4
(1 row)
```

Metaphone

The Metaphone algorithm works in the same way as the Soundex algorithm. The Metaphone algorithm constructs a representative code for each specified string. If two strings have the same representative code, the Metaphone algorithm considers them to be similar.

The Metaphone algorithm provides the following functions:

```
metaphone(text source, int max_output_length) returns text
```

The following table describes the parameters that you must configure in the preceding functions.

Parameter	Description
source	A string that is not empty. The string can contain up to 255 characters in length.
max_output_length	The maximum length of the Metaphone code that can be returned. If the Metaphone code exceeds the maximum length, the Metaphone algorithm truncates the Metaphone code to the maximum length.

Example:

```
SELECT metaphone('GUMBO', 4);
```

Double Metaphone

The Double Metaphone algorithm obtains two similar-sounding codes for a specified string. These codes include a primary code and a secondary code. In most cases, the two codes are the same. They may be slightly different when you specify a non-English word. The difference varies based on the pronunciation.

The Double Metaphone algorithm provides the following functions:

```
dmetaphone(text source) returns text
dmetaphone_alt(text source) returns text
```

Examples:

```
select dmetaphone('gumbo');
select dmetaphone_alt('gumbo');
```

6.17. Use the pg_hint_plan plug-in to customize query plans

ApsaraDB RDS for PostgreSQL provides the pg_hint_plan plug-in. You can use the plug-in to add hints to SQL statements. This allows you to change the execution plans of SQL statements on an ApsaraDB RDS for PostgreSQL instance.

Context

PostgreSQL uses a cost-based optimizer that works based on data statistics instead of static rules. The optimizer evaluates the cost of every possible execution plan for an SQL statement to your database system. This helps the optimizer select a final execution plan that has the lowest cost. However, the optimizer does not consider the possible internal relationships among data. Therefore, the final execution plan may not be perfect. You can use the pg_hint_plan plug-in to add hints to an SQL statement. These hints specify how you want to execute the SQL statement. This is another way to optimize the execution plans of SQL statements.

Basic usage

A hint starts with a forward slash, an asterisk, and a plus sign (`/*+`) and ends with an asterisk and a forward slash (`*/`). A hint consists of a hint name followed by parameters that are enclosed in a pair of parentheses. The parameters are separated by space characters. For readability purposes, you can separate each hint with a line break.

Example:

In this example, the HashJoin hint specifies that the SeqScan method is used to scan the pgbench_accounts table.

```
/*+
   HashJoin(a b)
   >SeqScan(a)
 */
EXPLAIN SELECT *
   FROM pgbench_branches b
   JOIN pgbench_accounts a ON b.bid = a.bid
  ORDER BY a.aid;
```

The following result is returned:

```
                                    QUERY PLAN
-------------------------------------------------------------------------------------------
 Sort  (cost=31465.84..31715.84 rows=100000 width=197)
   Sort Key: a.aid
   ->  Hash Join  (cost=1.02..4016.02 rows=100000 width=197)
         Hash Cond: (a.bid = b.bid)
         ->  Seq Scan on pgbench_accounts a  (cost=0.00..2640.00 rows=100000 width=97)
         ->  Hash  (cost=1.01..1.01 rows=1 width=100)
               ->  Seq Scan on pgbench_branches b  (cost=0.00..1.01 rows=1 width=100)
(7 rows)
```

Hint table

Hints can be used to optimize the execution plans of SQL statements. However, this is convenient only when SQL statements are editable. If SQL statements are not editable, you can place hints in a table named hint_plan.hints. The table consists of the following columns.

> **Note** By default, the user who creates the pg_hint_plan plug-in has the permissions on the hint table. The hints in the hint table take precedence over the hints that you add by using the pg_hint_plan plug-in.

Column	Description
id	The ID of the hint. The ID is unique and automatically generated.
norm_query_string	The pattern that matches the SQL statement to which you want to add the hint. The constants in the SQL statement must be replaced by wildcards (`?`). Space characters are crucial parts of the pattern.
application_name	The name of the application to which the hint is applied. If this parameter is left empty, the hint is applied to all applications.
hints	The comment that contains the hint. You do not need to include comment marks.

Example:

```
INSERT INTO hint_plan.hints(norm_query_string, application_name, hints)
    VALUES (
        'EXPLAIN (COSTS false) SELECT * FROM t1 WHERE t1.id = ? ;',
        '',
        'SeqScan(t1)'
    );
INSERT 0 1
postgres=# UPDATE hint_plan.hints
postgres-#    SET hints = 'IndexScan(t1)'
postgres-#  WHERE id = 1;
UPDATE 1
postgres=# DELETE FROM hint_plan.hints
postgres-#  WHERE id = 1;
DELETE 1
```

Hint types

Hints come in the following six types based on how they affect execution plans:

- Hints for scan methods

  This type of hint specifies the method that is used to scan the specified table. If the specified table has an alias, the pg_hint_plan plug-in identifies the table based on the alias. Supported scan methods include `SeqScan`, `IndexScan`, and `NoSeqScan`.

  The hints for scan methods are valid on ordinary tables, inherited tables, unlogged tables, temporary tables, and system tables. However, the hints for scan methods are invalid on external tables, table functions, statements in which the values of constants are specified, universal expressions, views, and subqueries.

  Example:

  ```
  /*+
      SeqScan(t1)
      IndexScan(t2 t2_pkey)
   */
  SELECT * FROM table1 t1 JOIN table table2 t2 ON (t1.key = t2.key);
  ```

- Hints for join methods

  This type of hint specifies the method that is used to join the specified tables.

  The hints for join methods are valid on ordinary tables, inherited tables, unlogged tables, temporary tables, external tables, system tables, table functions, statements in which the values of constants are specified, and universal expressions. However, the hints for join methods are invalid on views and subqueries.

- Hints for join order

  This type of hint specifies the order in which two or more tables are joined. You can use one of the following methods to specify a hint for join order:

  - Specify the order in which you want to join the specified tables, without restricting the direction at each join level.
  - Specify the order in which you want to join the specified tables, as well as the direction at each join level.

  Example:

```
/*+
    NestLoop(t1 t2)
    MergeJoin(t1 t2 t3)
    Leading(t1 t2 t3)
 */
SELECT * FROM table1 t1
    JOIN table table2 t2 ON (t1.key = t2.key)
    JOIN table table3 t3 ON (t2.key = t3.key);
```

- Hints for row number correction

  This type of hint corrects row number errors that are caused by the optimizer.

  Examples:

```
/*+ Rows(a b #10) */ SELECT... ;     //Sets the row number to 10.
/*+ Rows(a b +10) */ SELECT... ;     //Increases the row number by 10.
/*+ Rows(a b -10) */ SELECT... ;     //Decreases the row number by 10.
/*+ Rows(a b *10) */ SELECT... ;     //Increases the row number by 10 times.
```

- Hints for parallel execution

  This type of hint specifies the plan that is used to execute SQL statements in parallel.

  The hints for parallel execution are valid on ordinary tables, inherited tables, unlogged tables, and system tables. However, the hints for parallel execution are invalid on external tables, clauses in which the values of constants are specified, universal expressions, views, and subqueries. You can specify the internal tables of a view based on their real names or aliases.

  The following examples show how an SQL statement is executed in a different way on each table:

  - Example 1:

```
explain /*+ Parallel(c1 3 hard) Parallel(c2 5 hard) */
       SELECT c2.a FROM c1 JOIN c2 ON (c1.a = c2.a);
```

    The following result is returned:

```
                                QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=2.86..11406.38 rows=101 width=4)
   Hash Cond: (c1.a = c2.a)
   ->  Gather  (cost=0.00..7652.13 rows=1000101 width=4)
         Workers Planned: 3
         ->  Parallel Seq Scan on c1  (cost=0.00..7652.13 rows=322613 width=4)
   ->  Hash  (cost=1.59..1.59 rows=101 width=4)
         ->  Gather  (cost=0.00..1.59 rows=101 width=4)
               Workers Planned: 5
               ->  Parallel Seq Scan on c2  (cost=0.00..1.59 rows=59 width=4)
```

- Example 2:

```
EXPLAIN /*+ Parallel(tl 5 hard) */ SELECT sum(a) FROM tl;
```

The following result is returned:

```
                                  QUERY PLAN
-------------------------------------------------------------------------------------
 Finalize Aggregate  (cost=693.02..693.03 rows=1 width=8)
   ->  Gather  (cost=693.00..693.01 rows=5 width=8)
         Workers Planned: 5
         ->  Partial Aggregate  (cost=693.00..693.01 rows=1 width=8)
               ->  Parallel Seq Scan on tl  (cost=0.00..643.00 rows=20000 width=4)
```

- Hints for GUC parameter setting

  This type of hint temporarily changes the value of a GUC parameter. The values of GUC parameters in the execution plan help you achieve the expected effect. However, this does not apply if the specified hint conflicts with the execution plans of other SQL statements. If you set a GUC parameter more than once, the latest value takes effect.

  Example:

```
/*+ Set(random_page_cost 2.0) */
SELECT * FROM table1 t1 WHERE key = 'value';
```

The following table describes all of the supported hints.

Type	Hint	Description
Hints for scan methods	SeqScan(table)	Specifies a sequence scan.
	TidScan(table)	Specifies a TID scan.
	IndexScan(table[index...])	Specifies an index scan. You can specify an index.
	IndexOnlyScan(table[index...])	Specifies an index-only scan. You can specify an index.
	BitmapScan(table[index...])	Specifies a bitmap scan.
	NoSeqScan(table)	Prohibits a sequence scan.
	NoTidScan(table)	Prohibits a TID scan.
	NoIndexScan(table)	Prohibits an index scan.
	NoIndexOnlyScan(table)	Prohibits an index scan. Only tables are scanned.
	NoBitmapScan(table)	Prohibits a bitmap scan.
Hints for join	NestLoop(table table[table...])	Specifies a nested loop join.
	HashJoin(table table[table...])	Specifies a hash join.
	MergeJoin(table table[table...])	Specifies a merge join.

Hints for join methods Type	Hint	Description
	NoNestLoop(table table[table...])	Prohibits a nested loop join.
	NoHashJoin(table table[table...])	Prohibits a hash join.
	NoMergeJoin(table table[table...])	Prohibits a merge join.
Hints for join order	Leading(table table[table...])	Specifies the join order.
	Leading(<join pair>)	Specifies the join order and direction.
Hints for row number correction	Rows(table table[table...] correction)	Corrects the row number of the join result that is obtained from the specified tables. The following operators are supported: `#<n>`, `+ <n>`, `- <n>`, and `* <n>`. The `<n>` operator is supported by the strtod function.
Hints for parallel execution	Parallel(table <# of workers> [soft|hard])	Specifies or prohibits the parallel execution of the specified tables. The `<worker#>` parameter specifies the number of working programs that are required. The value 0 specifies to prohibit parallel execution. If the third parameter is set to soft, only the value of the max_parallel_workers_per_gather parameter is changed and the other parameters are specified by the optimizer. If the third parameter is set to hard, the values of all related parameters are changed. The third parameter is set to soft by default.
Hints for GUC parameter setting	Set(GUC-param value)	Specifies the value of a GUC parameter when the optimizer runs.

For more information, visit the official PostgreSQL website.

6.18. Use pg_repack to clear tablespaces

This topic describes how to clear the tablespaces of an ApsaraDB RDS for PostgreSQL instance by using the pg_repack plug-in. If you perform a large number of operations, such as UPDATE, on an entire table, a table bloat may occur. In this case, you can use this plug-in to remove the table bloat. When you use this plug-in to handle a table bloat issue, this plug-in does not need to acquire an exclusive lock on the bloated table. This plug-in is more lightweight than the CLUSTER and VACUUM FULL statements.

Prerequisites

The RDS instance must meet the following requirements:

- The major engine version of your RDS instance is PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13.
- The minor engine version of your RDS instance is 20210331 or later. For more information about how to view and update the minor engine version of an RDS instance, see Update the minor engine version of

an ApsaraDB RDS for PostgreSQL instance.

Precautions

- The pg_repack plug-in requires additional storage. A full-table repack requires the amount of free storage to be at least twice the size of the table that you want to repack.
- The pg_repack plug-in cannot remove bloats from temporary tables.
- The pg_repack plug-in cannot remove bloats from GiST indexes.
- When a table is being repacked by the pg_repack plug-in, you cannot execute DDL statements on the table. The pg_repack plug-in holds an ACCESS SHARE lock on the bloated table to prohibit the execution of DDL statements on the bloated table.
- The pg_repack plug-in consumes a large number of disk I/O resources to create tables and indexes. Before you use the pg_repack plug-in, you must evaluate whether the repack operation can interrupt your workloads. For example, if your RDS instance is equipped with enhanced SSDs (ESSDs) of PL1 and you want to repack a 100-GB table, the IOPS of the RDS instance can reach the maximum IOPS, which is 250 MB/s.

Features

The pg_repack plug-in supports full-table repack and index repack.

- The following procedure shows how the pg_repack plug-in repacks an entire table:
  i. Creates a logging table. The logging table is used to record the changes that are made to the original table during the repack process.
  ii. Creates triggers on the original table. The triggers are used to record the INSERT, UPDATE, and DELETE statements that are executed on the original table and insert the generated log records into the logging table during the repack process.
  iii. Creates a table. The new table contains the same rows and columns as the original table.
  iv. Creates indexes in the new table.
  v. Applies the data changes in the logging table to the new table.
  vi. Switches the original and new tables in the system catalog.
  vii. Deletes the original table.

  > **Note** The pg_repack plug-in holds an ACCESS EXCLUSIVE lock on the original table to prohibit operations on the original table in Steps i, ii, vi, and vii. In the other steps, the plug-in holds an ACCESS SHARE lock on the original table. The lock does not prohibit the execution of INSERT, UPDATE, and DELETE statements on the original table.

- The following procedure shows how the pg_repack plug-in repacks the indexes on a table:
  i. Concurrently creates indexes.
  ii. Switches the original and new indexes in the system catalog.
  iii. Deletes the original indexes.

Enable or disable the pg_repack plug-in

- Enable the pg_repack plug-in.

```
CREATE EXTENSION pg_repack;
```

- Disable the pg_repack plug-in.

```
DROP EXTENSION pg_repack;
```

Install the client utility of the pg_repack plug-in

You must install the client utility of the pg_repack plug-in. If your Elastic Compute Service (ECS) instance runs Alibaba Cloud Linux 3.2104 or a later version, run the following command to install the client utility:

1. Run the following command to install environment dependencies:

   ```
   yum install postgresql* redhat-rpm-config libpq* openssl-devel readline-devel -y
   ```

2. Run the following command to add environment variables:

   ```
   export PATH=$PATH:/usr/lib64/pgsql/postgresql-12/bin
   ```

3. Run the following commands to download the client utility, compile the client utility, and then install the client utility:

   ```
   wget https://github.com/reorg/pg_repack/archive/refs/tags/ver_1.4.6.tar.gz
   tar zxvf ver_1.4.6.tar.gz
   cd pg_repack-ver_1.4.6
   make && make install
   ```

Example

```
-- Check the pg_repack plug-in but do not perform the repack operation: --dry-run
$pg_repack --dry-run --no-superuser-check --echo --no-order -h endpoint-p port-d database1 -U
user --table schema1.table1
-- Check the pg_repack plug-in and perform the repack operation:
$pg_repack --no-superuser-check --echo --no-order -h endpoint-p port-d database1 -U user --ta
ble schema1.table1
```

FAQ

What do I do if the "ERROR: pg_repack failed with error: You must be a superuser to use pg_repack" message is returned?

Use the -k or --no-superuser-check option to skip the superuser permission check. This way, you can prevent this type of permission error.

References

For more information about the pg_repack plug-in, see Reorganize tables in PostgreSQL databases with minimal locks.

6.19. Use the pglogical plug-in for logical streaming replication

This topic describes how to use the pglogical plug-in of PostgreSQL to replicate the data of an ApsaraDB RDS for PostgreSQL instance. This plug-in uses a publish/subscribe pattern to implement logical streaming replication.

Prerequisites

- Your RDS instance runs one of the following PostgreSQL versions:
  - PostgreSQL 14
  - PostgreSQL 13
  - PostgreSQL 12
  - PostgreSQL 11
  - PostgreSQL 10
- pglogical is added to the value of the shared_preload_libraries parameter of your RDS instance.

  For more information about how to add **pglogical** to the value of the **shared_preload_libraries** parameter, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

Background information

The pglogical plug-in uses the publish/subscribe pattern to replicate data. The publish/subscribe pattern makes selective replication more efficient. The pglogical plug-in delivers higher replication speeds than Slony, Bucardo, and Londiste and supports cross-version upgrades. The pglogical plug-in is suitable for the following scenarios:

- Upgrade of major engine versions
- Replication of all data from a database on your RDS instance
- Replication of specified tables, rows, and columns by using replication sets
- Aggregation and merging of data from multiple upstream servers

Use the pglogical plug-in

- Enable the pglogical plug-in.

```
CREATE EXTENSION pglogical;
```

- Disable the pglogical plug-in.

```
DROP EXTENSION pglogical;
```

Configure logical streaming replication

1. Create a provider on the provider node.

   Note The value of the `host` parameter is fixed as 127.0.0.1, and the value of the `port` parameter is fixed as 3002.

```
SELECT pglogical.create_node(
    node_name := 'provider',
    dsn := 'host=127.0.0.1 port=3002 dbname=test user=provider_user password=provider_pas
s'
);
```

2. Configure a replication set.

   Add all tables in the public schema to the default replication set.

```
SELECT pglogical.replication_set_add_all_tables('default', ARRAY['public']);
```

> **Note**
>
> - A replication set is used to replicate only the specified tables and only the specified changes to these tables from the provider to a subscriber.
> - The default replication set is used to replicate all tables and all changes to these tables from the provider to a subscriber. For more information, see the pglogical documentation.

3. Create a subscriber on the subscriber node.

   > **Note** The value of the `host` parameter is fixed as 127.0.0.1, and the value of the `port` parameter is fixed as 3002.

   ```
   SELECT pglogical.create_node(
       node_name := 'subscriber',
       dsn := 'host=127.0.0.1 port=3002 dbname=test user=subscriber_user password=subscriber
   _pass'
   );
   ```

4. Create a subscription on the subscriber node.

   > **Note** The `host` parameter must be set to the private IP address of the provider, and the `port` parameter must be set to the internal port number of the provider.

   ```
   SELECT pglogical.create_subscription(
       subscription_name := 'subscription',
       provider_dsn := 'host=<The private IP address of the provider> port=<The internal por
   t number of the provider> dbname=test user=provider_user password=provider_pass'
   );
   ```

   After the subscription is created, the synchronization and replication process starts in the background.

References

For more information, see the pglogical documentation.

6.20. Use the pgAudit plug-in to generate audit logs

This topic describes how to use the pgAudit plug-in to generate audit logs for an ApsaraDB RDS for PostgreSQL instance in compliance with public service, financial, or ISO requirements. Audit logs help you analyze faults and operations on your RDS instance to obtain information about data queries.

Prerequisites

- Your RDS instance meets the following requirements:
  - The major engine version of your RDS instance is PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, or PostgreSQL 14.

  - The minor engine version of your RDS instance is 20210531 or later. For more information about how to view and update the minor engine version of an RDS instance, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

- pgaudit is added to the value of the shared_preload_libraries parameter of your RDS instance.

  For more information about how to add **pgaudit** to the value of the **shared_preload_libraries** parameter, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

Precautions

- The pgAudit plug-in may generate a large amount of audit log data. The amount of audit log data that is generated varies based on the configuration of the pgAudit plug-in. Before you use the pgAudit plug-in to audit objects, we recommend that you evaluate the objects to prevent the pgAudit plug-in from generating a large amount of audit log data. A large amount of audit log data can exhaust the storage capacity of your RDS instance.
- After an object is renamed, new audit log records that are generated by the pgAudit plug-in for the object are associated with the new name of the object.

Enable or disable the pgAudit plug-in

- Enable the pgAudit plug-in.

```
CREATE EXTENSION pgaudit;
```

- Disable the pgAudit plug-in.

```
DROP EXTENSION pgaudit;
```

References

For more information, see the pgAudit documentation.

6.21. Use the pldebugger plug-in to debug stored procedures

This topic describes how to debug the stored procedures of an ApsaraDB RDS for PostgreSQL instance by using the pldebugger plug-in.

Context

ApsaraDB RDS for PostgreSQL supports various stored procedure languages, such as PL/pgSQL, PL/Python, PL/Perl, and PL/Tcl. You can use these languages to create functions or stored procedures.

Prerequisites

- Your RDS instance runs one of the following database engine versions:
  - Major engine version: PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13.
  - Minor engine version: 20211130 or later. For more information about how to update the minor engine version, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

- plugin_debugger is added to the value of the shared_preload_libraries parameter of your RDS instance.

For more information about how to add **plugin_debugger** to the value of the **shared_preload_libraries** parameter, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

- The version of pgAdmin4 on your database client is 4.19 or later. You can download pgAdmin4 at pgAdmin4.

Procedure

- Enable the pldebugger plug-in.

```
CREATE EXTENSION pldbgapi;
```

> **Note** Only privileged accounts are authorized to execute the preceding statement.

- Disable the pldebugger plug-in.

```
DROP EXTENSION pldbgapi;
```

> **Note** Only privileged accounts are authorized to execute the preceding statement.

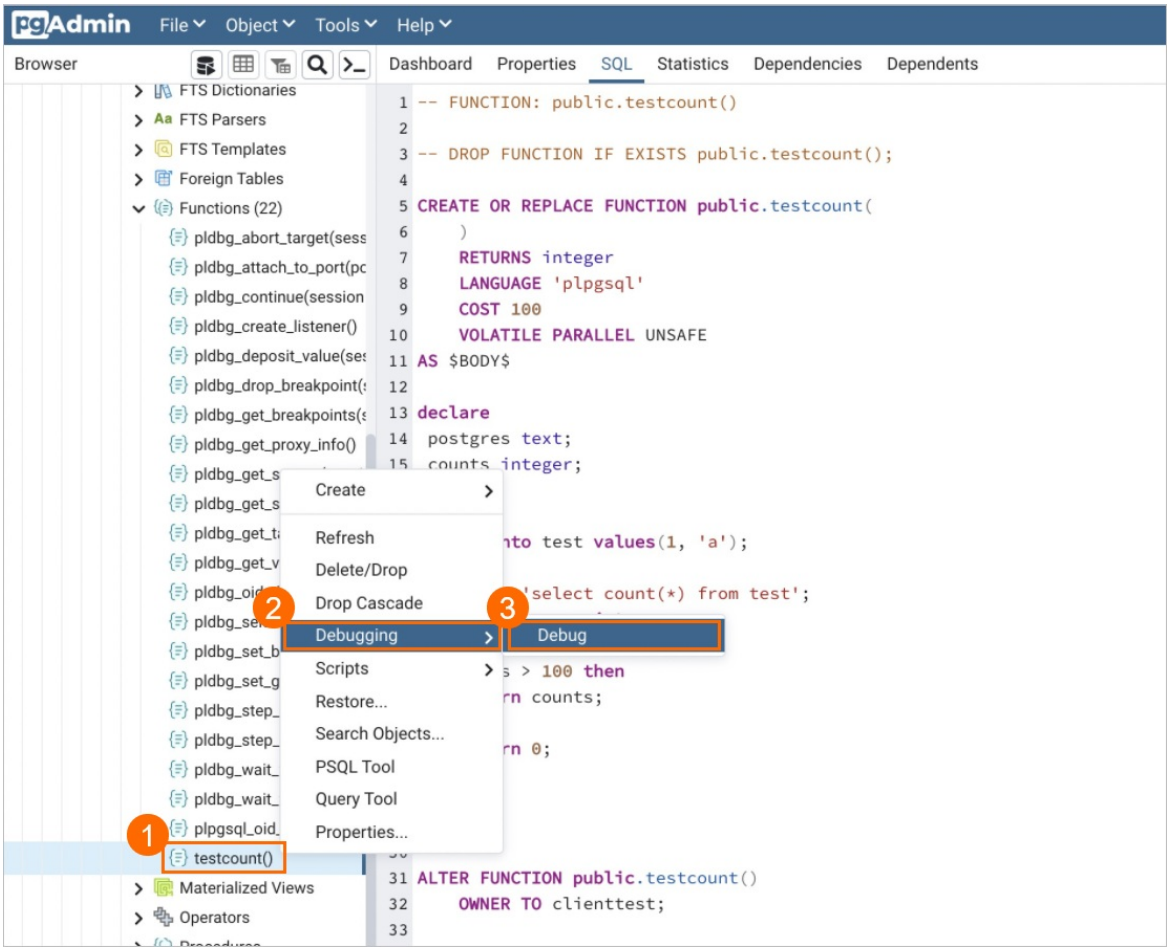
Examples

1. Use pgAdmin to connect to your RDS instance. For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance.
2. Create a database and a stored procedure that are used for testing.

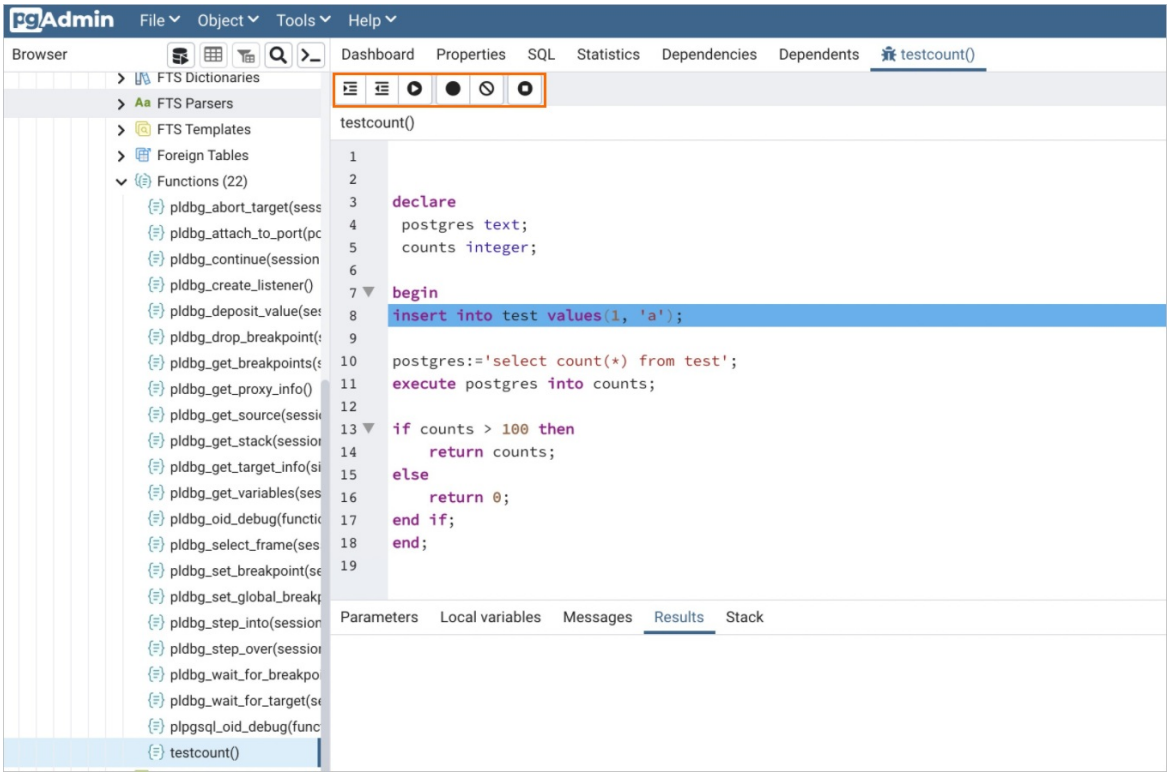
   Example:

```
CREATE TABLE test(
    id int,
    name VARCHAR(50));
CREATE OR REPLACE FUNCTION public.testcount()
RETURNS integer AS $$
DECLARE
    postgres text;
    counts integer;
BEGIN
INSERT INTO test VALUES(1, 'a');
postgres:='SELECT COUNT(*) FROM test';
EXECUTE postgres INTO counts;
IF counts > 100 THEN
    RETURN counts;
ELSE
    RETURN 0;
END IF;
END;
$$ language plpgsql;
```

3. Right-click the function that you want to debug and choose Debugging > Debug.



4. In the right-side debugging section of the page, perform step-by-step operations to debug the function. These operations include step into/over, continue, checkpointing, and stop. In the lower part of the page, you can view the local variables, debugging results, and function stack.



6.22. Using the index_adviser plug-in on an ApsaraDB RDS for PostgreSQL instance

This topic describes how to use the index_adviser plug-in on an ApsaraDB RDS for PostgreSQL instance. This plug-in helps you determine the columns on which you need to create indexes to improve query performance for specified workloads. This plug-in can recognize only single-column or composite B-tree indexes. This plug-in cannot recognize other types of indexes that can improve performance. For example, this plug-in cannot recognize GIN, GiST, or Hash indexes.

Prerequisites

The minor engine version of the RDS instance is 20220130 or later. For more information about how to view and update the minor engine version of an RDS instance, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

Components of the index_adviser plug-in

When you execute the statement that is used to create the index_adviser plug-in, the index_advisory table, show_index_advisory() function, and select_index_advisory view are also created.

Component	Description
index_advisory	A table that is created when the index_adviser plug-in is created. This table is used to record indexing suggestions.
show_index_advisory()	A PL/pgSQL function that interprets and displays the suggestions made during a specific session. The session is identified by its backend process ID.

Component	Description
select_index_advisory	A view that is created by the index_adviser plug-in based on the information stored in the index_advisory table during query analysis. The format of the view is the same as the format of the output of the show_index_advisory() function. The view contains all indexing suggestions for the specified session.

Usage

1. Create an index_adviser plug-in.

```
postgres=# create extension index_adviser;
CREATE EXTENSION
```

2. Load the index_adviser plug-in.

```
postgres=# LOAD 'index_adviser';
LOAD
```

> **Note** The preceding statement is valid only for the current session. If you want all sessions to load the index_adviser plug-in by default, you must configure the shared_preload_libraries parameter in the index_adviser plug-in and restart the RDS instance. However, this may affect the performance of the RDS instance. Perform the following configuration:
>
> ```
> shared_preload_libraries='index_adviser'
> ```

Examples

- Create a table

```
CREATE TABLE t( a INT, b INT );
INSERT INTO t SELECT s, 99999 - s FROM generate_series(0,99999) AS s;
ANALYZE t;
The table contains the following rows:
   a   |   b
-------+-------
     0 | 99999
     1 | 99998
     2 | 99997
     3 | 99996
       .
       .
       .
 99997 |     2
 99998 |     1
 99999 |     0
```

- Query the indexing suggestions for a single SQL statement.

- If you want to use the index_adviser plug-in to analyze a query and obtain the indexing suggestions but you do not want to execute the query, use the EXPLAIN keyword as the prefix of the SQL statement. Example:

```
postgres=# EXPLAIN SELECT * FROM t WHERE a < 10000;
                                  QUERY PLAN
-------------------------------------------------------------------------------
 Seq Scan on t  (cost=0.00..1693.00 rows=9983 width=8)
   Filter: (a < 10000)
 Result  (cost=0.00..0.00 rows=0 width=0)
   One-Time Filter: '** plan (using Index Adviser) **'::text
   ->  Index Scan using "<1>t_a_idx" on t  (cost=0.42..256.52 rows=9983 width=8)
         Index Cond: (a < 10000)
(6 rows)
```

```
postgres=# EXPLAIN SELECT * FROM t WHERE a = 100;
                                QUERY PLAN
---------------------------------------------------------------------------
 Seq Scan on t  (cost=0.00..1693.00 rows=1 width=8)
   Filter: (a = 100)
 Result  (cost=0.00..0.00 rows=0 width=0)
   One-Time Filter: '** plan (using Index Adviser) **'::text
   ->  Index Scan using "<1>t_a_idx" on t  (cost=0.42..2.64 rows=1 width=8)
         Index Cond: (a = 100)
(6 rows)
```

```
postgres=# EXPLAIN SELECT * FROM t WHERE b = 10000;
                                QUERY PLAN
---------------------------------------------------------------------------
 Seq Scan on t  (cost=0.00..1693.00 rows=1 width=8)
   Filter: (b = 10000)
 Result  (cost=0.00..0.00 rows=0 width=0)
   One-Time Filter: '** plan (using Index Adviser) **'::text
   ->  Index Scan using "<1>t_b_idx" on t  (cost=0.42..2.64 rows=1 width=8)
         Index Cond: (b = 10000)
(6 rows)
```

- You can use the PostgreSQL CLI to query indexing suggestions from the index_advisory table. Example:

```
postgres=# SELECT * FROM index_advisory;
 reloid | relname | attrs | benefit | original_cost | new_cost | index_size | backend_pid
|            timestamp
--------+---------+-------+---------+---------------+----------+------------+------------
-+-----------------------------------
  16438 | t       | {1}   | 1337.43 |          1693 |  355.575 |       2624 |       79370
| 18-JUN-21 08:55:51.492388 +00:00
  16438 | t       | {1}   | 1684.56 |          1693 |    8.435 |       2624 |       79370
| 18-JUN-21 08:59:00.319336 +00:00
  16438 | t       | {2}   | 1684.56 |          1693 |    8.435 |       2624 |       79370
| 18-JUN-21 08:59:07.814453 +00:00
(3 rows)
```

Field	Type	Description
reloid	oid	The OID of the table for the index.
relname	name	The name of the table for the index.
attrs	integer[]	The column to which the indexing suggestion is generated. The column is identified by a number.
benefit	real	The benefit of using the index to accelerate the query.
original_cost	real	The average amount of time that is required to execute the SQL statement before you use the index to accelerate the query.
new_cost	real	The average amount of time that is required to execute the SQL statement after you use the index to accelerate the query.
index_size	integer	The estimated index size in the disk page.
backend_pid	integer	The ID of the process that generated this suggestion.
timestamp	timestamp	The date and time when this suggestion was generated.

- If the SQL statement is not prefixed with the EXPLAIN keyword, the index_adviser plug-in analyzes the SQL statement when the query is being executed and records indexing suggestions.

> **Note** You cannot use the index_adviser plug-in in read-only transactions.

- Query the indexing suggestions for a specified workload.

- Obtain the indexing suggestions for a session by using the show_index_advisory() function.

  This function is used to obtain the indexing suggestions for a session. The session is identified by its backend process ID. You can call this function by specifying the process ID of the session.

```
SELECT show_index_advisory( pid );
```

> **Note** pid indicates the process ID of the current session. You can obtain the process ID by using the backend_pid parameter in the index_advisory table. You can also specify null as a passed value to return the result set for the current session.

```
postgres=# SELECT show_index_advisory(null);
                                                                                show_index_advisory
-------------------------------------------------------------------------------------------------------------------------------------------
 create index idx_t_a on public.t(a);/* size: 2624 KB, benefit: 3021.99, gain: 1.15167301457103, original_cost: 1693, new_cost: 182.005006313324 */
 create index idx_t_b on public.t(b);/* size: 2624 KB, benefit: 1684.56, gain: 0.641983590474943, original_cost: 1693, new_cost: 8.4350004196167 */
(2 rows)
```

> **Note** The following description shows the meaning of each row in the result set:
>
> - The SQL statement that is used to create an index from the indexing suggestions.
> - The estimated size of the index page.
> - The benefit of using the index to accelerate the query.
> - The gain of using the index. The following formula is used to calculate the gain of the index: Gain of using the index = Benefit of using the index/Consumed size of the index.
> - The average amount of time that is required to execute the SQL statement before you use the index to accelerate the query.
> - The average amount of time that is required to execute the SQL statement after you use the index to accelerate the query.

- Obtain the indexing suggestions for a session by using the select_index_advisory view.

  This view contains calculated metrics and **CREATE INDEX** statements and provides indexing suggestions for all sessions in the index_advisory table. The following example shows the indexing suggestions for Column a and Column b of Table t:

```
postgres=# SELECT * FROM select_index_advisory;
 backend_pid |                                                                                show_index_adv
isory
-------------+--------------------------------------------------------------------------------
-------------------------------------------------------------------------
       79370 | create index t_a_idx on public.t(a);/* size: 2624 KB, benefit: 3021.99, ga
in: 1.15167301457103, original_cost: 1693, new_cost: 182.005006313324 */
       79370 | create index t_b_idx on public.t(b);/* size: 2624 KB, benefit: 1684.56, ga
in: 0.641983590474943, original_cost: 1693, new_cost: 8.4350004196167 */
(2 rows)
```

  In each session, the results of all queries that benefit from the same indexing suggestion are combined into a set of metrics for the indexing suggestion. The metric is represented by a field named benefit and a field named gain. The following formula shows how to calculate the values of the two fields:

```
size = MAX(index size of all queries)
benefit = SUM(benefit of each query)
gain = SUM(benefit of each query) / MAX(index size of all queries)
```

  > **Note** If the indexing suggestions recommend that you create multiple indexes for a single SQL statement, the new_cost field of the index_advisory table records the cost after multiple indexes are created.

  The gain field is useful for comparing the advantages between different recommended indexes during the specified session. A larger value of the gain field indicates a higher benefit of the recommended index. The benefit can offset the disk space that the recommended index may consume.

6.23. Use HypoPG to create hypothetical indexes

This topic describes how to create hypothetical indexes by using the HypoPG plug-in of ApsaraDB RDS for PostgreSQL. You can use hypothetical indexes to check whether indexes can help increase query performance. Hypothetical indexes are not real indexes and do not consume resources such as CPU and disk resources.

Prerequisites

- The minor engine version of your RDS instance is updated to 20220130. For more information about how to update the minor engine version of an RDS instance, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.
- A privileged account is used to connect to your RDS instance. You can check the type of the account that you use on the **Accounts** page in the ApsaraDB RDS console. If the account is a standard account, you must create a privileged account and use the privileged account to connect to your RDS instance. For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance.

> **Note** Hypothetical indexes are valid only in the current session.

Enable or disable HypoPG

- Execute the following statement to enable HypoPG:

```
CREATE EXTENSION hypopg;
```

> **Note** Only privileged accounts are granted the permissions to execute the preceding statement.

- Execute the following statement to disable HypoPG:

```
DROP EXTENSION hypopg;
```

> **Note** Only privileged accounts are granted the permissions to execute the preceding statement.

Examples

1. Create a table and insert test data into the table.

```
create extension hypopg;
CREATE TABLE hypo (id integer, val text) ;
INSERT INTO hypo SELECT i, 'line ' || i FROM generate_series(1, 100000) i ;
VACUUM ANALYZE hypo ;
```

2. Check query performance of execution plans of SQL statements when no index is created on the table.

```
EXPLAIN SELECT val FROM hypo WHERE id = 1;
                       QUERY PLAN
-------------------------------------------------------------
 Seq Scan on hypo  (cost=0.00..1791.00 rows=1 width=14)
   Filter: (id = 1)
(2 rows)
```

3. Enable HypoPG and create a hypothetical index on the table.

```
SELECT * FROM hypopg_create_index('CREATE INDEX ON hypo (id)') ;
 indexrelid |      indexname
------------+----------------------
      18284 | <18284>btree_hypo_id
(1 row)
```

4. Check whether the hypothetical index increases query performance of the execution plans.

```
EXPLAIN SELECT val FROM hypo WHERE id = 1;
                                    QUERY PLAN
-------------------------------------------------------------------------------------
 Index Scan using <18284>btree_hypo_id on hypo  (cost=0.04..8.06 rows=1 width=10)
   Index Cond: (id = 1)
(2 rows)
```

5. Check the execution plans of the SQL statements that are executed. The hypothetical index that you created is not used in the execution plans of the SQL statements.

```
EXPLAIN ANALYZE SELECT val FROM hypo WHERE id = 1;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------
----------
 Seq Scan on hypo  (cost=0.00..1791.00 rows=1 width=10) (actual time=0.046..46.390 rows=1
loops=1)
   Filter: (id = 1)
   Rows Removed by Filter: 99999
 Planning time: 0.160 ms
 Execution time: 46.460 ms
(5 rows)
```

References

For more information about HypoPG, see Usage of HypoPG.

6.24. Use the sequential-uuids plug-in to generate sequential UUIDs

This topic describes how to use the sequential-uuids plug-in. ApsaraDB RDS for PostgreSQL provides this plug-in to generate UUIDs in a more sequential pattern.

Prerequisites

- Your RDS instance runs one of the following database engine versions:
  - The major engine version of your RDS instance is PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, or PostgreSQL 14.
  - The minor engine version of your RDS instance is 20220228 or later. For more information about how to update the minor engine version of an RDS instance, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.
- A privileged account is used to connect to your RDS instance. You can check the type of the account that you use on the **Accounts** page in the ApsaraDB RDS console. If the account is a standard account, you must create a privileged account and use the privileged account to connect to your RDS instance. For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance.

Enable or disable the plug-in

- Execute the following statement to enable the sequential-uuids plug-in:

```
CREATE EXTENSION sequential_uuids;
```

- Execute the following statement to disable the sequential-uuids plug-in:

```
DROP EXTENSION sequential_uuids;
```

Examples

This plug-in provides two functions to generate sequential UUIDs by using sequences or timestamps.

> **Note** For more information about how to use the plug-in and related parameters, see Sequential UUID generators.

- The uuid_sequence_nextval function

  Syntax:

```
uuid_sequence_nextval(sequence regclass, block_size int default 65536, block_count int defa
ult 65536)
```

  Sample statement:

```
CREATE SEQUENCE s;
SELECT uuid_sequence_nextval('s'::regclass, 256, 256);
```

  Sample result:

```
        uuid_sequence_nextval
--------------------------------------
 00cf26f7-ef7a-4746-8871-08b9c475713e
(1 row)
```

- The uuid_time_nextval function

  Syntax:

```
uuid_time_nextval(interval_length int default 60, interval_count int default 65536) RETURNS
uuid
```

  Sample statement:

```
SELECT uuid_time_nextval(1, 256);
```

  Sample result:

```
          uuid_time_nextval
---------------------------------------
 179f542c-978c-454c-9dee-5f4ba75288e1
(1 row)
```

6.25. Use the MADlib plug-in

This topic describes how to use the MADlib plug-in. MADlib is an open source library that runs machine learning and graph computing models in AliPG databases. In terms of machine learning, MADlib provides functions and stored procedures for mathematical operations. MADlib also provides a set of typical supervised and unsupervised algorithm libraries for machine learning.

Prerequisites

- Your ApsraDB RDS for PostgreSQL instance runs one of the following database engine versions:
  - The major engine version of your RDS instance is PostgreSQL 11 or PostgreSQL 12.
  - The minor engine version of your RDS instance is 20220228 or later. For more information about how to view and update the minor engine version of an RDS instance, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

- A privileged account is used to connect to your RDS instance. You can check the type of the account that you use on the **Accounts** page in the ApsaraDB RDS console. If the account is a standard account, you must create a privileged account and use the privileged account to connect to your RDS instance. For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance.

Background information

The machine learning module of MADlib solves the following issues:

- Classification and regression issues: MADlib provides a set of algorithms such as K-Nearest Neighbor (KKN), multilayer perceptron neural network, support vector machine (SVM), and decision tree to solve binary classification and regression issues. MADlib also provides a set of models such as least-squares regression, generalized linear model (GLM), logistic regression, and multinomial logistic regression to solve regression issues.
- Clustering issues: MADlib provides the K-means algorithm for clustering analysis.
- Correlation analysis: MADlib provides the Apriori algorithm for correlation analysis. The feature can help find unexpected correlations between products such as the correlation between diapers and beer.
- Analysis of time series data: MADlib provides autoregressive integrated moving average (ARIMA) models to predict future trends of time series data.
- Others: MADlib provides principal component analysis (PCA) to extract the main factors for data dimension reduction. MADlib provides a Latent Dirichlet Allocation (LDA) model for document classification and topic modeling.

MADlib also integrates a graph computing model to solve issues such as the shortest path, PageRank ranking, and social media issues on queries for the contacts of a specific user. The following table describes the algorithms related to graph computing models.

Type	Model or feature	Description
Shortest path	Shortest path among all vertices	Calculates the shortest path among all vertices and saves the result to a specific result table. This model queries the shortest path from a start vertex to an end vertex based on the result table.
	Shortest path between a specific vertex and all other vertices	Calculates the shortest path between a specific vertex and all other vertices and saves the result to a specific result table. This model queries the shortest path from a specific vertex to any other vertex based on the result table.
Breadth-first search (BFS)	BFS	Uses the BFS method to query vertices that are reachable from a specific source vertex.
HITS	HITS score	Queries the HITS scores of all vertices in a directed graph. The HITS scores include hub scores and authority scores.
Web page ranking	PageRank	Queries the PageRank values of all vertices in a directed graph.
Weakly connected component	Weakly connected component	Queries all weakly connected components in a directed graph.
	Average path length	Calculates the average shortest path length of graphs.

Type	Model or feature	Description
Measure	Proximity	Calculates the closeness centrality of all nodes in a graph.
	Graph diameter	Calculates the graph diameter.
	In-degree or out-degree	Calculates the in-degree and out-degree of all vertices.

Enable or disable the MADlib plug-in

- Execute the following statement to enable the MADlib plug-in:

  > **Note** Before you execute the following statement, you must execute the `CREATE EXTENSION plpythonu;` statement to create the plpythonu plug-in.

  ```
  CREATE EXTENSION madlib;
  ```

- Execute the following statement to disable the MADlib plug-in:

  ```
  DROP EXTENSION madlib;
  ```

References

For more information about the MADlib plug-in, see MADlib documentation.

7.Quick start

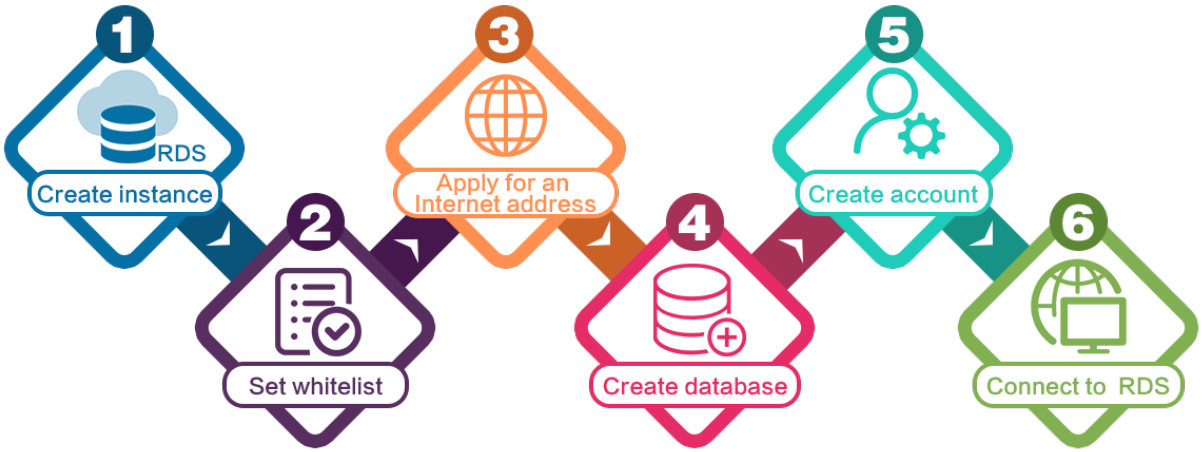
7.1. General workflow to use ApsaraDB RDS for PostgreSQL

This topic describes how to create and use an ApsaraDB RDS for PostgreSQL instance.

Quick start flowchart

If this is the first time that you use ApsaraDB RDS for PostgreSQL, we recommend that you familiarize yourself with the limits of ApsaraDB RDS for PostgreSQL. For more information, see Limits of ApsaraDB RDS for PostgreSQL.

The following flowchart shows the operations that you must perform before you use an ApsaraDB RDS for PostgreSQL instance.



1. Create an ApsaraDB RDS for PostgreSQL instance
2. Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance
3. Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance
4. Create a database and an account on an ApsaraDB RDS for PostgreSQL instance
5. Connect to an ApsaraDB RDS for PostgreSQL instance

7.2. Create an ApsaraDB RDS for PostgreSQL instance

This topic describes how to create an ApsaraDB RDS for PostgreSQL instance in the ApsaraDB RDS console. You can also create an ApsaraDB RDS for PostgreSQL instance by calling an API operation.

> **Note** You are offered a reduced price on your first purchase of an RDS instance. For more information, visit the ApsaraDB RDS promotion page.

Prerequisites

The AliyunRDSFullAccess policy is attached to the RAM user that you use to create an RDS instance. For more information, see Use RAM for resource authorization.

Procedure

1. Go to the ApsaraDB RDS buy page.
2. Configure the **Billing Method** parameter.

Billing method	Description	Benefit
Subscription	A subscription instance is an instance for which you pay an upfront fee. If you want to use an instance for a long period of time, we recommend that you select the **Subscription** billing method. If you select the subscription billing method, configure the **Duration** parameter in the lower part of the page.	In most cases, the subscription billing method is more cost-effective than the pay-as-you-go billing method for long-term usage. Alibaba Cloud provides lower prices for longer subscription periods.
Pay-As-You-Go	You are charged on an hourly basis for a pay-as-you-go instance based on your actual resource usage. If you want to use an instance for a short period of time, we recommend that you select the **Pay-As-You-Go** billing method. You can create a **pay-as-you-go** RDS instance. After you confirm that the new RDS instance meets your business requirements, you can change the billing method of the RDS instance from pay-as-you-go to **subscription**.	You can release a pay-as-you-go RDS instance based on your business requirements. The billing cycle of a pay-as-you-go RDS instance immediately stops after you release the instance.

> **Note** You can view the price in the lower-right corner of the page. The price is displayed only after you configure all required parameters.

3. Configure the following parameters.

Parameter	Description

Parameter	Description
Region	The region where the RDS instance resides. We recommend that you select the region of on which your application is deployed. If the RDS instance and the ECS instance reside in different regions, you cannot connect these instances over an internal network. In this case, these instances cannot deliver the optimal performance. the Elastic Compute Service (ECS) instance **Note** • After an RDS instance is created, you cannot change the region of the RDS instance. If you want to connect an ECS instance and an RDS instance over an internal network, make sure that the RDS instance and the ECS instance reside in the same region. • For more information about how to view the region of an ECS instance, see Get ready to use ApsaraDB RDS for MySQL. • If your application is deployed on an on-premises server or on-premises computer, we recommend that you select a region that is near your on-premises server or on-premises computer. This way, you can use the public endpoint of the RDS instance to connect to the RDS instance from your application.
Database Engine	The database engine and version that are run by the RDS instance. Select **PostgreSQL**. The supported PostgreSQL versions are 10, 11, 12, 13, and 14. **Notice** ApsaraDB RDS for PostgreSQL provides the Babelfish feature that is developed based on the Babelfish for PostgreSQL open source project. This feature enables your RDS instance to be compatible with Transact-SQL (T-SQL) statements. If you want to connect your SQL Server application or client to an RDS instance that runs PostgreSQL, we recommend that you select **Enable Babelfish** when you create the RDS instance. For more information, see Introduction to Babelfish.

Parameter	Description
Edition	◦ **Basic**: In RDS Basic Edition, the database system consists of only a primary RDS instance. RDS Basic Edition is cost-effective and suitable for learning and testing. **Note** RDS instances that run RDS Basic Edition require a long period of time to restart or recover from faults. ◦ **High-availability**: This is the recommended edition. In RDS High-availability Edition, the database system consists of a primary RDS instance and a secondary RDS instance. These instances work in the high availability architecture. RDS High-availability Edition is suitable for production environments and more than 80% of business scenarios. **Note** ◦ The available RDS editions vary based on the region and database engine that you select. ◦ For more information, see Overview of ApsaraDB RDS editions.
Storage Type	◦ **Local SSD**: A local SSD resides on the same host as the database engine. You can store data on local SSDs to reduce I/O latency. Local SSDs are supported only for RDS instances that run PostgreSQL 10. ◦ **ESSD**: Enhanced SSDs (ESSDs) come in three performance levels (PLs). ■ ESSD PL1: This is the basic PL of ESSDs. ■ ESSD PL2: An ESSD of PL2 delivers IOPS and throughput that are approximately twice the IOPS and throughput delivered by an ESSD of PL1. ■ ESSD PL3: An ESSD of PL3 delivers IOPS that is up to 20 times the IOPS delivered by an ESSD of PL1 and up to 11 times the throughput delivered by an ESSD of PL1. ESSDs of PL3 are suitable for business scenarios in which highly concurrent requests must be processed with high I/O performance and at low read and write latencies. ◦ **Standard SSD**: A standard SSD is an elastic block storage device that is designed based on the distributed storage architecture. You can store data on standard SSDs to separate computing from storage. **Note** ◦ The available storage types vary based on the instance type and RDS edition that you select. ◦ If you select the **ESSD** or **Standard SSD** storage type, you can select **Disk Encryption** to enhance the security of your data. For more information, see Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance. ◦ For more information about storage types, see Storage types.
Zone of Primary Node	Select a . zone

Parameter	Description
Deployment Method	◦ **Multi-zone Deployment**: This is the recommended deployment method. The primary RDS instance and the secondary RDS instance reside in different zones to provide zone-disaster recovery. ◦ **Single-zone Deployment**: The primary RDS instance and the secondary RDS instance reside in the same zone. **Note** ◦ No substantive differences exist between the zones in the same region. ◦ If the RDS instance resides in the same zone as the ECS instance on which your application is deployed, these instances can deliver optimal performance. If the RDS instance and the ECS instance reside in different zones in the same region, the performance of the RDS instance and the ECS instance is slightly lower than the performance of the RDS instance and the ECS instance that reside in the same zone. ◦ If you set the Edition parameter to **Basic**, only the **Single-zone Deployment** method is supported. ◦ If **Sold Out** appears in the upper-right corner of a zone name, this zone does not have sufficient resources. In this case, you must switch to another zone.
Zone of Secondary Node	If you set the **Deployment Method** parameter to **Multi-zone Deployment**, you must select the zone in which the secondary RDS instance resides.

Parameter	Description
Instance Type	The instance type of the RDS instance. Before you select an instance type, you must select an instance family. ○ **General-purpose**: A general-purpose RDS instance exclusively occupies the allocated memory and I/O resources. The RDS instance shares CPU and storage resources with the other general-purpose RDS instances deployed on the same server. ○ **Dedicated**: A dedicated instance exclusively occupies the allocated CPU, memory, storage, and I/O resources. Dedicated host instance types provide the highest specifications in the dedicated instance family. A dedicated host instance exclusively occupies all the CPU, memory, storage, and I/O resources on the physical host on which the RDS instance is deployed. ○ **General-purpose (New)**: The new general-purpose instance types provide better scalability and higher performance than the old general-purpose instance types. In addition, the period of time that is required to create an RDS instance and the period of time that is required to change the specifications of an RDS instance are reduced. The new general-purpose instance types are in development. Some features of ApsaraDB RDS are not supported for RDS instances that use the new general-purpose instance types. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types. **Note** ○ In a test environment, select an instance type that provides one or more cores. ○ In a production environment, select an instance type that provides four or more cores. For more information, see Primary ApsaraDB RDS instance types.
Capacity	The storage capacity that is provided to store data files, system files, binary log files, and transaction files in the RDS instance. The storage capacity varies based on the instance type and storage type that you select. You can adjust the storage capacity at a step size of 5 GB. **Note** ○ If you select the local SSD storage type, the storage capacity of the RDS instance may vary based on the instance type. If you select the standard SSD or ESSD storage type, the storage capacity of the RDS instance does not vary based on the instance type. For more information, see Primary ApsaraDB RDS instance types. ○ After an RDS instance is created, you can adjust the storage capacity of the RDS instance by changing the specifications or configuring automatic storage expansion. For more information, see 变更配置 or Configure automatic storage expansion for an ApsaraDB RDS for PostgreSQL instance.

4. In the lower-right corner of the page, click **Next: Instance Configuration**.
5. Configure the **VPC** and **VSwitch** parameters. We recommend that you select the same virtual private

cloud (VPC) in which your ECS instance resides for your RDS instance. If you select a different VPC for your RDS instance, you cannot connect your RDS instance and ECS instance over an internal network.

> **Note**
> - For more information about how to view the VPC in which your ECS instance resides, see Get ready to use ApsaraDB RDS for MySQL.
> - You can connect the RDS instance and the ECS instance over an internal network even if the instances use different vSwitches in the same VPC.

6. Configure more custom parameters. If you do not have special business requirements, you can retain the default values of these parameters.

Parameter	Description
Release Protection	Specifies whether to enable the release protection feature. The release protection feature is used to prevent a pay-as-you-go RDS instance from being released due to unintended operations. For more information, see Enable or disable the release protection feature for an ApsaraDB RDS for PostgreSQL instance.
Resource Group	The resource group to which the RDS instance belongs. You can retain the default resource group or select a custom resource group based on your business requirements.
Babelfish Migration Mode	The migration mode of the RDS instance after Babelfish is enabled. This parameter takes effect only when you select **Enable Babelfish** in the Basic Configurations step. ○ **single-db**: You can create only one SQL Server database on an RDS instance for which Babelfish is enabled and create a standard PostgreSQL schema for the database. ○ **multi-db**: You can create multiple SQL Server databases and create different PostgreSQL schemas for the databases. You must name the schemas in the <Database name>_<Schema name> format to prevent name conflicts. **Note** For more information, see Migration modes.
Initial Account	The username of the Babelfish management account. This parameter takes effect only when you select **Enable Babelfish** in the Basic Configurations step. The Babelfish management account is used to connect to the RDS instance over the TDS port. **Notice** This account is a privileged account and cannot be deleted after it is created. Username requirements: ○ The username must be 2 to 63 characters in length. ○ The username can contain lowercase letters, digits, and underscores (_). ○ The username must start with a lowercase letter and end with a lowercase letter or a digit. ○ The username cannot start with pg. ○ The username cannot contain SQL keywords. For more information, see SQL keywords.

Parameter	Description
Password	The password of the Babelfish management account. This parameter takes effect only when you select **Enable Babelfish** in the Basic Configurations step. **Note** You can change the password after the RDS instance is created. For more information, see Reset the password of an account on an ApsaraDB RDS for PostgreSQL instance. Password requirements: ○ The password must be 8 to 32 characters in length. ○ The password must contain at least three of the following character types: uppercase letters, lowercase letters, digits, and special characters. ○ The following special characters are supported: `! @ # $ % ^ & * ( ) _ + - =` .
Time Zone	The time zone of the RDS instance. **Note** ○ You can configure the time zone when you create a primary RDS instance. You cannot configure the time zone when you create a read-only RDS instance. Read-only RDS instances inherit the time zone of their primary RDS instance. ○ You can configure this parameter only when the RDS instance uses standard SSDs or ESSDs. ○ The time zone is not in UTC. For more information about time zones, see Common time zones for ApsaraDB RDS for MySQL instances and ApsaraDB RDS for PostgreSQL instances. ○ If you do not configure this parameter, the system assigns the default time zone of the region in which the RDS instance resides to the RDS instance. For more information about the mappings between regions and time zones, see Default time zones for ApsaraDB RDS for PostgreSQL instances.

7. In the lower-right corner of the page, click **Next: Confirm Order**.
8. Confirm the configuration of the RDS instance in the Parameters section, configure the **Purchase Plan** and **Duration** parameters, read and select **Terms of Service**, and then click **Pay Now**. You must configure the Duration parameter only if you select the subscription billing method for the RDS instance.

   Note If you select the subscription billing method for the RDS instance, we recommend that you select **Auto-Renew Enabled**. This way, you can prevent interruptions on your workloads even if you forget to renew the RDS instance.

9. View the RDS instance.

   Go to the Instances page. In the top navigation bar, select the region in which the RDS instance resides. Then, find the RDS instance based on the **Creation Time** parameter.

What to do next

Create a database and an account on an ApsaraDB RDS for PostgreSQL instance

FAQ

Why am I unable to find the RDS instance that I created?

Possible cause	Description	Suggestion
Incorrect region	The RDS instance does not reside in the region that you selected in the top navigation bar of the ApsaraDB RDS console.	In the top navigation bar, select the region in which the RDS instance resides.
Insufficient resources	The zone that you selected cannot provide sufficient resources. If the RDS instance cannot be created, you can go to the Orders page in the Billing Management console to view the refunded fee.	Select a different zone and try again.
RAM policies that do not allow users to create unencrypted RDS instances	RAM policies that do not allow users to create unencrypted RDS instances are attached to a RAM user. • If you use the credentials of the RAM user to create an RDS instance that uses local SSDs, the RDS instance cannot be created. When you create an RDS instance that uses local SSDs, you cannot enable disk encryption. • If you use the credentials of the RAM user to create an RDS instance that uses standard SSDs or ESSDs and you do not enable disk encryption for the RDS instance, the RDS instance cannot be created. For more information, see Use RAM policies to manage the permissions of RAM users on ApsaraDB RDS instances.	When you create an RDS instance, select the standard SSD or ESSD storage type, select Disk Encryption, set the Key parameter, and then try again.

References

- For more information about how to create an RDS instance by calling an API operation, see Create an instance.
- For more information about how to create an RDS instance that runs a different database engine, see the following topics:
  - Create an ApsaraDB RDS for MySQL instance
  - Create an ApsaraDB RDS for SQL Server instance
  - Create an ApsaraDB RDS for MariaDB TX instance

7.3. Set the whitelist

7.3.1. Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance. After an RDS instance is created, you must configure IP address whitelists or security groups for the RDS instance. Otherwise, the RDS instance is inaccessible.

For more information about how to configure an IP address whitelist for an RDS instance that runs a different database engine, see the following topics:

- Configure an IP address whitelist for an ApsaraDB RDS for MySQL instance
- Configure an IP address whitelist for an ApsaraDB RDS for SQL Server instance
- Configure an IP address whitelist for an ApsaraDB RDS for MariaDB TX instance

Scenarios

An IP address whitelist consists of IP addresses and CIDR blocks that are granted access to your RDS instance. You can configure IP address whitelists to provide high-level access control and security protection for your RDS instance. We recommend that you update the configured IP address whitelists on a regular basis.

You must configure IP address whitelists in the following scenarios:

- Scenario 1

  After your RDS instance is created, you must add the IP addresses of specific devices to an IP address whitelist of your RDS instance. This way, these devices can access your RDS instance.

- Scenario 2

  Your RDS instance cannot be connected. In this case, you must check the IP address whitelists of your RDS instance and modify the IP address whitelists that are incorrectly configured.

  The following table provides the IP address whitelist configurations in various connection scenarios.

  > **Note** A virtual private cloud (VPC) is an isolated network on Alibaba Cloud. VPCs provide higher security than the classic network. For more information, see What is a VPC?

Connection scenario	Network type	IP address whitelist configuration
	The ECS instance and your RDS instance reside in the same VPC. This is the recommended connection scenario.	Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.

Connection scenario	Network type	IP address whitelist configuration
Connect an Elastic Compute Service (ECS) instance to your RDS instance	The ECS instance and your RDS instance reside in different VPCs.	Instances in different VPCs cannot communicate with each other over internal networks. Perform the following operations: i. Migrate your RDS instance to the VPC where the ECS instance resides. **Note** This operation is supported only when the ECS instance and your RDS instance reside in the same region. If the ECS instance and your RDS instance reside in different regions, we recommend that you use Data Transmission Service (DTS) to migrate your RDS instance to the region where the ECS instance resides. This way, you can ensure the stability of your database service. . ii. Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.
	The ECS instance and your RDS instance reside in the classic network.	Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.
	The ECS instance resides in the classic network. Your RDS instance resides in a VPC.	Instances of different network types cannot communicate with each other over internal networks. Perform the following operations: i. Migrate the ECS instance from the classic network to the VPC where your RDS instance resides. For more information, see Migrate an ECS instance from the classic network to a VPC. **Note** This operation is supported only when the ECS instance and your RDS instance reside in the same region. If the ECS instance and your RDS instance reside in different regions, we recommend that you use DTS to migrate your RDS instance to the region where the ECS instance resides. This way, you can ensure the stability of your database service. . ii. Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.

Connection scenario	Network type	IP address whitelist configuration
	The ECS instance resides in a VPC. Your RDS instance resides in the classic network.	Instances of different network types cannot communicate with each other over internal networks. Perform the following operations: i. Migrate your RDS instance from the classic network to the VPC where the ECS instance resides. **Note** This operation is supported only when the ECS instance and your RDS instance reside in the same region. If the ECS instance and your RDS instance reside in different regions, we recommend that you use DTS to migrate your RDS instance to the region where the ECS instance resides. This way, you can ensure the stability of your database service. . ii. Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.
Connect a self-managed host outside the cloud to your RDS instance	None.	Add the public IP address of the self-managed host to an IP address whitelist of your RDS instance. **Note** ◦ The applications that run on the self-managed host connect to the public endpoint of your RDS instance. ◦ For more information about how to obtain the public IP address of the self-managed host, see Why am I unable to connect to my ApsaraDB RDS for MySQL or ApsaraDB RDS for MariaDB instance from a local server over the Internet?

Precautions

- A maximum of 50 IP address whitelists can be configured for each RDS instance.
- When you configure IP address whitelists, the workloads on your RDS instance are not interrupted.
- The IP address whitelist that is labeled default can be cleared but cannot be deleted.
- Do not modify or delete the IP address whitelists that are generated by other Alibaba Cloud services. If you delete the IP address whitelist that is generated by an Alibaba Cloud service, the Alibaba Cloud service cannot connect to your RDS instance. For example, the IP address whitelist labeled ali_dms_group is generated by Data Management (DMS), and the IP address whitelist labeled hdm_security_ips is generated by Database Autonomy Service (DAS).
- The IP address whitelist labeled default contains only the 127.0.0.1 IP address. This indicates that no IP addresses are granted access to your RDS instance.

Configure a standard IP address whitelist

In standard whitelist mode, ApsaraDB RDS does not distinguish between the classic network and VPCs. The IP addresses or CIDR blocks in a standard IP address whitelist are granted access to your RDS instance over both the classic network and VPCs.

1.
2. In the left-side navigation pane, click **Data Security**.
3. Click **Create Whitelist**. In the dialog box that appears, configure parameters such as **Whitelist Name** to create an IP address whitelist. Alternatively, click **Modify** to the right of an existing IP address whitelist to modify the IP address whitelist.
4. Enter the IP addresses or CIDR blocks that require access to your RDS instance. Then, click OK.

   > **Note**
   > - If you enter more than one IP address or CIDR block, you must separate these IP addresses or CIDR blocks with commas (,). Do not add spaces preceding or following the commas. Example: `192.168.0.1,172.16.213.9`.
   > - A maximum of 1,000 IP addresses and CIDR blocks can be configured for each RDS instance. If you want to enter a large number of IP addresses, we recommend that you merge the IP addresses into CIDR blocks, such as 10.10.10.0/24.

5. Optional. If the RDS instance is attached with a read-only RDS instance, you can use the **Synchronize whitelist to read-only instance** parameter to configure the RDS instance to synchronize IP address whitelists to the read-only RDS instance. ApsaraDB RDS supports the synchronization of IP address whitelists to multiple read-only RDS instances.
6. Optional. Click **Loading ECS Inner IP**. In the dialog box that appears, view the IP addresses of all ECS instances that are created within your Alibaba Cloud account. Then, add the IP addresses of the ECS instances that you want to connect to the IP address whitelist.

Configure an enhanced IP address whitelist

In enhanced whitelist mode, ApsaraDB RDS distinguishes between the classic network and VPCs. You must specify the network isolation mode of each enhanced IP address whitelist. For example, if the Network Isolation Mode parameter is set to Classic Network for an IP address whitelist, the IP addresses in the IP address whitelist are granted access to your RDS instance only over the classic network and you cannot connect to your RDS instance over VPCs from these IP addresses.

The enhanced whitelist mode is supported only for RDS instances that are equipped with local SSDs. If your RDS instance runs in enhanced whitelist mode, you can perform the following procedure to configure an enhanced IP address whitelist. For more information about how to switch the network isolation mode of an RDS instance from the standard whitelist mode to the enhanced whitelist mode, see Switch an ApsaraDB RDS for PostgreSQL instance to the enhanced whitelist mode.

1.
2. In the left-side navigation pane, click **Data Security**.
3. On the Whitelist Settings tab, create or modify an IP address whitelist.
   - Create an IP address whitelist
     a. Click **Create Whitelist**.
     b. Set the **Network Isolation Mode** parameter.
     c. Enter a name in the **Whitelist Name** field, add IP addresses or CIDR blocks, and then click OK.
   - Modify an IP address whitelist.

     Click **Modify** to the right of the IP address whitelist.
4. In the **Edit Whitelist** dialog box, add IP addresses or CIDR blocks and click **OK**.

> **Note**
>
> - If you enter more than one IP address or CIDR block, you must separate these IP addresses or CIDR blocks with commas (,). Do not add spaces preceding or following the commas. Example: `192.168.0.1,172.16.213.9`.
> - A maximum of 1,000 IP addresses and CIDR blocks can be configured for each RDS instance. If you want to enter a large number of IP addresses, we recommend that you merge the IP addresses into CIDR blocks, such as, 10.10.10.0/24.

5. Optional. If the RDS instance is attached with a read-only RDS instance, you can use the **Synchronize whitelist to read-only instance** parameter to configure the RDS instance to synchronize IP address whitelists to the read-only RDS instance. ApsaraDB RDS supports the synchronization of IP address whitelists to multiple read-only RDS instances.
6. Optional. Click **Loading ECS Inner IP**. In the dialog box that appears, view the IP addresses of all ECS instances that are created within your Alibaba Cloud account. Then, add the IP addresses of the ECS instances that you want to connect to the IP address whitelist.

What to do next

Create a database and an account on an ApsaraDB RDS for PostgreSQL instance

Related operations

Operation	Description
DescribeDBInstanceIPArrayList	Queries the IP address whitelists of an ApsaraDB RDS instance.
ModifySecurityIps	Modifies an IP address whitelist of an ApsaraDB RDS instance.

7.3.2. Configure a security group for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure a security group for an ApsaraDB RDS for MySQL instance. A security group is a virtual firewall that is used to control the inbound and outbound traffic of the Elastic Compute Service (ECS) instances in that security group. After you add a security group to your RDS instance, all the ECS instances in that security group can access the instance.

Scenarios

After your RDS instance is created, you must configure IP address whitelists or security groups for the instance. Otherwise, your RDS instance is inaccessible. For more information about how to configure an IP address whitelist, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

For more information about security groups, see Create a security group.

Precautions

- You can configure both IP address whitelists and security groups. All the IP addresses in the configured IP address whitelists and all the ECS instances in the configured security groups are granted access to your RDS instance.

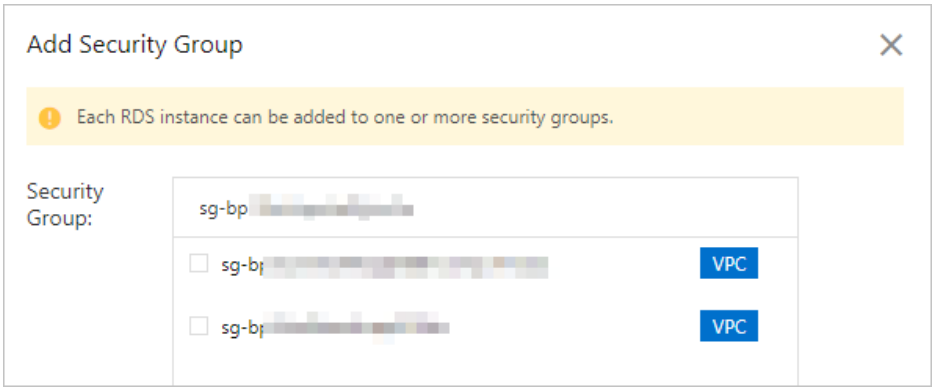
- A maximum of 10 security groups can be configured for each RDS instance.
- After the ECS instances in a configured security group are updated, the updates are automatically synchronized to the configured security group.
- You can configure only a security group that has the same network type as your RDS instance. In this case, the network types of your RDS instance and the security group that you want to configure must both be VPC or classic network.

> **Note** After you change the network type of your RDS instance, the security groups that you configured become invalid. You must configure the security groups of the specified network type again.

Procedure

1.
2. In the left-side navigation pane, click **Data Security**. On the page that appears, click the **Security Group** tab.
3. Click **Add Security Group**.

   > **Note** Security groups that are followed by a **VPC** tag contain ECS instances that reside in virtual private clouds (VPCs).

4. Select the security group that you want to add, and then click **OK**.

What to do next

Create a database and an account on an ApsaraDB RDS for PostgreSQL instance

Related operations

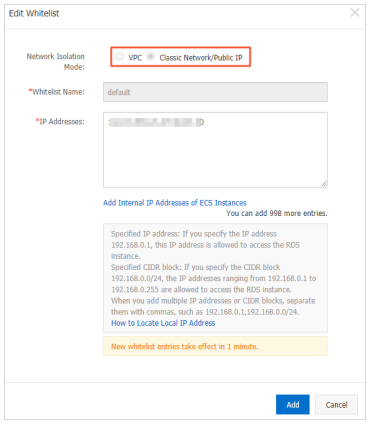
API	Description
DescribeSecurityGroupConfiguration	Queries details about the ECS security groups that are associated with an ApsaraDB RDS instance.
ModifySecurityGroupConfiguration	Modifies details about the ECS security groups that are associated with an ApsaraDB RDS instance.

7.3.3. Errors and FAQ about IP address whitelist settings in ApsaraDB RDS for PostgreSQL

This topic describes the common errors and provides answers to some commonly asked questions about the IP address whitelist settings of an ApsaraDB RDS for PostgreSQL instance.

Common errors

Error	Description	Solution
No IP address whitelists are configured. Your RDS instance has only one default IP address whitelist. The default IP address whitelist contains only the 127.0.0.1 IP address.	The 127.0.0.1 IP address indicates that no devices can access your RDS instance.	Add the IP addresses of the specified devices to an IP address whitelist.
The 0.0.0.0 entry is added to an IP address whitelist during a connectivity test.	The format of the 0.0.0.0 entry is invalid.	Change the 0.0.0.0 IP address to the 0.0.0.0/0 Classless Inter-Domain Routing (CIDR) block. **Notice** The 0.0.0.0/0 CIDR block indicates that all IP addresses are granted access to your RDS instance. We recommend that you add this CIDR block only for a connectivity test. When you run online workloads, do not add this CIDR block to an IP address whitelist.
The public IP addresses in a configured IP address whitelist are inaccessible.	• The public IP addresses dynamically change. • The tool or website that you use to query public IP addresses returns inaccurate results.	For more information, see How do I locate the IP address connected to an RDS for PostgreSQL instance?

Error	Description	Solution
The IP addresses of the specified devices are added to an enhanced IP address whitelist, and the network type of this whitelist differs from the network types of these devices.	In enhanced whitelist mode, ApsaraDB RDS distinguishes between the classic network and virtual private networks (VPCs).	Add the IP addresses to an IP address whitelist whose network type is the same as the network types of the devices. For example, if an IP address is added to an IP address whitelist of the VPC network type, you can connect to your RDS instance from the IP address only over a VPC. 

FAQ

- Can I configure both IP address whitelists and security groups for my RDS instance?

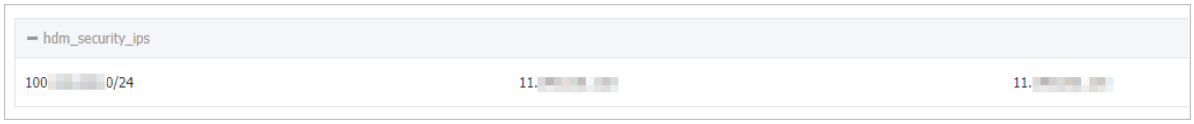
  Yes, you can configure both IP address whitelists and security groups for your RDS instance. All the IP addresses in the configured IP address whitelists and all the Elastic Compute Service (ECS) instances in the configured security groups are granted access to your RDS instance.

- After I configure an IP address whitelist for my RDS instance, does the IP address whitelist immediately take effect?

  After you configure an IP address whitelist for your RDS instance, the IP address whitelist requires about 1 minute to take effect.

- What are the IP address whitelists labeled **ali_dms_group** and **hdm_security_ips**?

  When you connect to your RDS instance from other Alibaba Cloud services, these services generate IP address whitelists upon your authorization. The generated IP address whitelists contain the IP addresses of the servers on which these services run. The IP address whitelist labeled ali_dms_group is generated by Data Management (DMS). The IP address whitelist labeled hdm_security_ips is generated by Database Autonomy Service (DAS). Do not modify or delete the IP address whitelists. If you modify or delete the IP address whitelists, these services cannot access your RDS instance. These services do not perform operations on your business data.

7.4. Create a database and an account on an ApsaraDB RDS for PostgreSQL instance

Before you can use an ApsaraDB RDS instance, you must create a database and an account on the instance. This topic describes how to create a database and an account on an ApsaraDB RDS for PostgreSQL instance.

Account types

ApsaraDB RDS for PostgreSQL instances support two types of accounts: privileged accounts and standard accounts. The following table describes these types of accounts.

Account type	Description
Privileged account	• You can create and manage privileged accounts in the ApsaraDB RDS console or by using the ApsaraDB RDS API. • You can create multiple privileged accounts for each RDS instance. The privileged accounts of an RDS instance have the permissions to manage all standard accounts and databases that are created on the instance. • A privileged account allows you to manage permissions at fine-grained levels based on your business requirements. For example, you can grant each standard account the permissions to query specific tables. • A privileged account has the permissions to log off all standard accounts on the instance on which the privileged account is created. **Note** • The first privileged account that you create is the owner of the default public schema of a standard system database named template1. • By default, the **CREATE DATABASE** statement creates a database by replicating the template1 system database. The owners of all databases that are created by using this statement from the template1 system database are the first privileged account. • The comment of the first privileged account starts with "template1 public schema owner."
Standard account	• You can create and manage standard accounts in the ApsaraDB RDS console, by using the ApsaraDB RDS API, or by executing SQL statements. • You can create multiple standard accounts for each RDS instance. • You must grant the permissions on specified databases to standard accounts. • You cannot use a standard account to create, manage, or log off other accounts from the instance on which the standard account is created.

Precautions

- You can create multiple privileged accounts and standard accounts in the ApsaraDB RDS console. You

can also create and manage standard accounts by using SQL statements.

- Before you can migrate data from an on-premises database to an RDS instance, you must create a database and an account on the RDS instance. Make sure that the created database has the same properties as the on-premises database. In addition, make sure that the created account has the same permissions on the created database as the account that is authorized to manage the on-premises database.
- We recommend that you follow the principle of least privilege (PoLP) and grant the read and write permissions to accounts based on your business requirements. You can create multiple accounts and grant each account only the permissions to access the data of specified databases. If an account does not need to write data to a database, we recommend that you grant only the read permissions on the database to the account.
- For security purposes, we recommend that you specify strong passwords for accounts and change the passwords on a regular basis.

Create a database

Create an account

1.
2. In the left-side navigation pane, click **Accounts**.
3. Click **Create Account**.
4. Configure the following parameters.

Parameter	Description
Database Account:	◦ The username of the account must be 2 to 63 characters in length. ◦ The username of the account can contain lowercase letters, digits, and underscores (_). ◦ The username of the account must start with a lowercase letter and end with a lowercase letter or a digit. ◦ The username of the account cannot be the same as the username of an existing account. ◦ The username of the account cannot start with pg. ◦ The username of the account cannot contain SQL keywords. For more information, see SQL Keywords.
Account Type:	Specify the type of the account. Two types of accounts are supported: privileged accounts and standard accounts. ◦ A privileged account has all operation permissions on all databases. ◦ Standard accounts have all operation permissions only on their authorized databases. **Note** The operation permissions include SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, and TRIGGER.

Parameter	Description
Password:	○ The password of the account must be 8 to 32 characters in length. ○ The password of the account must contain at least three of the following character types: uppercase letters, lowercase letters, digits, and special characters. ○ The password of the account can contain any of the following special characters: ! @ # $ % ^ & * ( ) _ + - =
Confirm Password:	Enter the password of the account again.
Description	Enter the description of the account.

5. Click **OK**.

FAQ

After I create accounts on my primary RDS instance, can I manage the accounts on the read-only RDS instances that are attached to my primary RDS instance?

No, although the accounts that are created on your primary RDS instance are synchronized to the read-only RDS instances, you cannot manage the accounts on the read-only RDS instances. The accounts have only the read permissions and do not have the write permissions on the read-only RDS instances.

Related operations

Operation	Description
Create a database account	Creates an account on an ApsaraDB RDS instance.

7.5. Connect to an ApsaraDB RDS for PostgreSQL instance

This topic describes how to connect to an ApsaraDB RDS for PostgreSQL instance. You can connect to an RDS instance by using Data Management (DMS), a command-line tool, pgAdmin, or an application.

Prerequisites

The operations that are described in the following topics are complete:

- Create an ApsaraDB RDS for PostgreSQL instance
- Create an account on an ApsaraDB RDS for PostgreSQL instance
- Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance
- If you connect to your RDS instance over an internal network from an Elastic Compute Service (ECS) instance, the following requirements are met:
  - The ECS instance and the RDS instance belong to the same Alibaba Cloud account.
  - The ECS instance and the RDS instance reside in the same region.
  - The ECS instance and the RDS instance reside in the same virtual private cloud (VPC).

- The private IP address of the ECS instance is added to an IP address whitelist of the RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

Use DMS to connect to an RDS instance

Log on to the ApsaraDB RDS console, find the RDS instance to which you want to connect, and go to the **Basic Information** page. In the upper-right corner of the page, click **Log On to Database**. Then, enter the required information to log on to the RDS instance.

For more information, see Use DMS to log on to an ApsaraDB RDS for PostgreSQL instance.

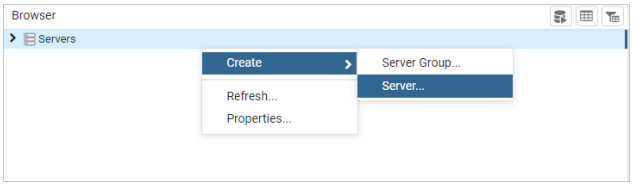
Use pgAdmin to connect to an RDS instance

When you download the PostgreSQL software package from the PostgreSQL official website and install PostgreSQL, **pgAdmin 4** is automatically downloaded and installed. You can also download the pgAdmin software package from the PostgreSQL official website.

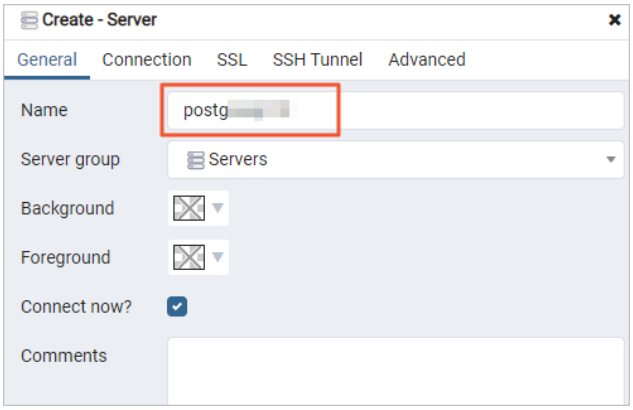
1. Start **pgAdmin 4**.

   > **Note** If you use pgAdmin that is in a later version than version 4 and you use pgAdmin for the first time, you must specify a master password to protect your saved logon credentials such as passwords.

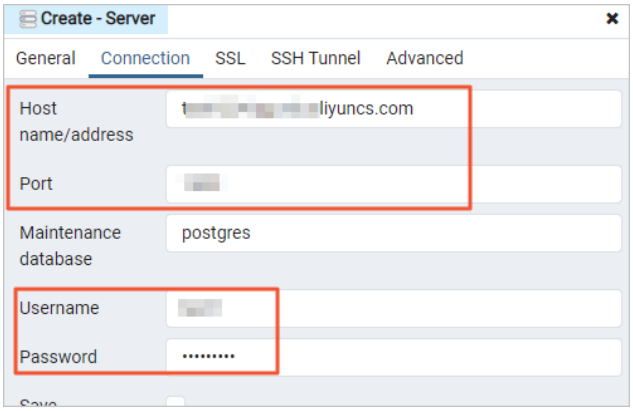
2. Right-click **Servers** and choose **Create > Server...**.



3. On the **General** tab of the Create - Server dialog box, enter the name of the server on which pgAdmin is installed.



4. Click the **Connection** tab and enter the information that is used to connect to the RDS instance.

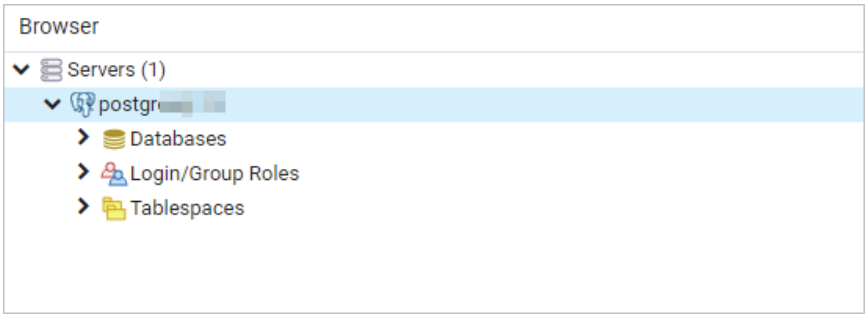


Parameter	Description
Hostname/address	Enter the endpoint of the RDS instance. If you want to connect to the RDS instance over an internal network, enter the internal endpoint of the RDS instance. If you want to connect to the RDS instance over the Internet, enter the public endpoint of the RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Port	Enter the port number that is associated with the specified endpoint.
Username	Enter the username of the account that is used to log on to the RDS instance. For more information about how to create an account for an RDS instance, see Create a database and an account on an ApsaraDB RDS for PostgreSQL instance.
Password	Enter the password of the account that is used to log on to the RDS instance.

5. Click **Save**.

   If the information that you enter is correct, the page that is shown in the following figure appears, which indicates that the connection to the RDS instance is successful.

   > **Notice** The postgres database is a default system database. Do not perform operations on this database.

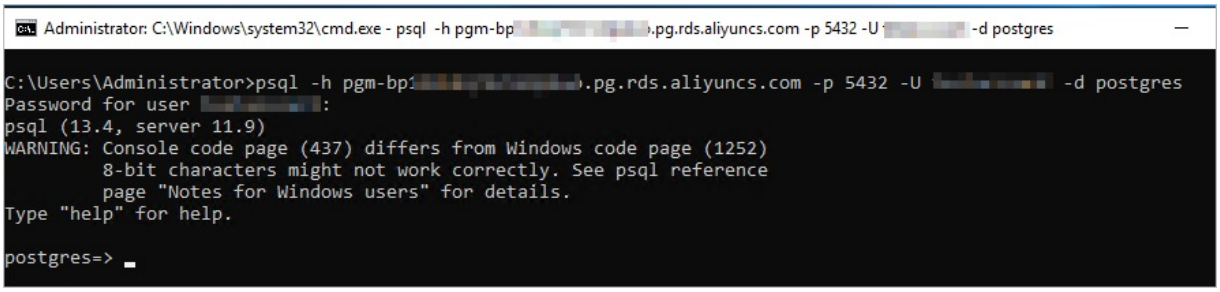


Use a command-line tool to connect to an RDS instance

When you download the PostgreSQL software package from the PostgreSQL official website and install PostgreSQL, **a PostgreSQL command-line tool** is automatically downloaded and installed.

Run the following command in the command-line tool to connect to a database of the RDS instance:

```
psql -h <Endpoint> -U <Username> -p <Port number> -d <Database name>
```



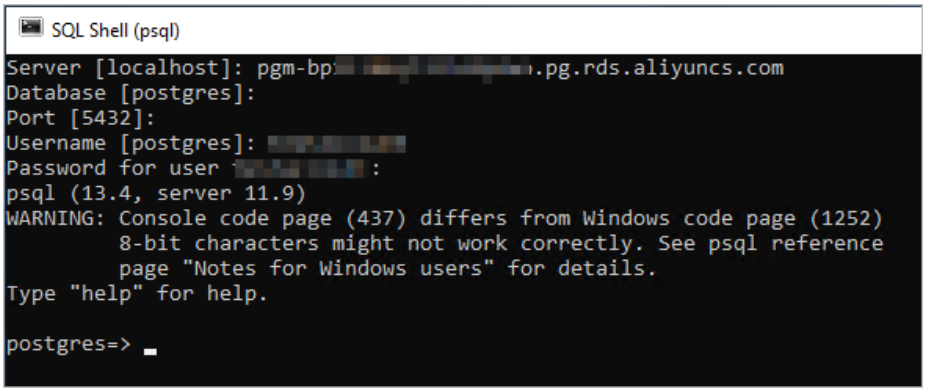
The following table provides details about how to obtain the values of the parameters from the ApsaraDB RDS console.

Parameter	How to obtain
Endpoint.	The endpoint that is used to connect to the RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Username	The username of the account that is used to log on to the RDS instance. You can obtain the username from the **Accounts** page. For more information about how to create an account, see Create an account on an ApsaraDB RDS for PostgreSQL instance.
Port number	The port number that is used to connect to the RDS instance. The default port number is 5432. If you have modified the port number, you can obtain the new port number from the **Database Connection** page. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Database name	The name of the database that you want to connect in the RDS instance. The postgres database is a default system database. Do not perform operations on this database. You can obtain the name of the database that you want to connect from the **Databases Connection** page. For more information about how to create a database, see Create a database on an ApsaraDB RDS for PostgreSQL instance.

Use SQL Shell (psql) to connect to an RDS instance

When you download the PostgreSQL software package from the PostgreSQL official website and install PostgreSQL, **SQL Shell (psql)** is automatically downloaded and installed.

Open the **Start** menu on your computer and click the **SQL Shell (psql)**. Then, enter the required parameters to connect to the RDS instance.



The following table provides details about how to obtain the values of the parameters from the ApsaraDB RDS console.

Parameter	How to obtain
Server	The endpoint that is used to connect to the RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Database	The name of the database that you want to connect in the RDS instance. If you do not specify this parameter, the default value is postgres. The postgres database is a default system database. Do not perform operations on this database. You can obtain the name of the database that you want to connect from the **Databases Connection** page. For more information about how to create a database, see Create a database on an ApsaraDB RDS for PostgreSQL instance.
Port	The port number that is used to connect to the RDS instance. The default port number is 5432. If you have modified the port number, you can obtain the new port number from the **Database Connection** page. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Username	The username of the account that is used to log on to the RDS instance. You can obtain the username from the **Accounts** page. For more information about how to create an account, see Create an account on an ApsaraDB RDS for PostgreSQL instance.

Use an application to connect to an RDS instance

> **Note** In this topic, a Maven project is connected to the RDS instance by using the Java Database Connectivity (JDBC). This connection method is similar to other programming languages.

1. Add dependencies to the pom.xml file:

```
<dependency>
  <groupId>postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>8.2-504.jdbc3</version>
</dependency>
```

2. The following code snippet provides an example on how to use the JDBC to connect to the RDS instance:

```
public class DatabaseConnection
{
    public static void main( String[] args ){
        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        //Endpoint
        String hostname = "pgm-bp1i3kkq7321o9****.pg.rds.aliyuncs.com";
        //Port number
        int port = 5432;
        //Database name
        String dbname = "postgres";
        //Username
        String username = "username";
        //Password
        String password = "password";
        String dbUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbname + "?bi
naryTransfer=true";
        Connection dbConnection;
        try {
            dbConnection = DriverManager.getConnection(dbUrl, username, password);
            Statement statement = dbConnection.createStatement();
            //SQL statement that you want to execute
            String selectSql = "SELECT * FROM information_schema.sql_features LIMIT 10";
            ResultSet resultSet = statement.executeQuery(selectSql);
            while (resultSet.next()) {
                System.out.println(resultSet.getString("feature_name"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Configure SSL encryption for an RDS instance

You can configure SSL encryption for an RDS instance. SSL encryption is used to encrypt the connections to the RDS instance and protect the data that is transmitted over the connections. For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance over SSL.

FAQ

How do I use Function Compute to obtain data from my RDS instance?

You can install third-party dependencies on Function Compute. Then, you can use these built-in dependencies to obtain data from ApsaraDB RDS. For more information, see Install third-party dependencies.

8.Development and O&M recommendations for ApsaraDB RDS for PostgreSQL

This topic describes the development and O&M recommendations that can help you increase the security compliance, stability, and performance of your ApsaraDB RDS for PostgreSQL instance.

Connection pooling

- We recommend that you store SQL statements in PreparedStatement objects. This way, hard parses are not required, which reduces CPU resource consumption and increases the performance of your RDS instance.
- We recommend that you close idle connections. This way, you can reduce memory usage, improve the efficiency of `GetSnapshotData()`, and increase database performance.
- We recommend that you enable the connection pool feature on your application to prevent the resources from being consumed by short-lived connections and prevent performance deterioration. If your application does not support the connection pool feature, we recommend that you configure a connection pool between your application and your RDS instance. For example, you can use PgBouncer or Pgpool-II as a connection pool.
- We recommend that you configure the following parameters for connection pooling:
  - `minimumIdle`: specifies the minimum number of idle connections in connection pools. We recommend that you set this parameter to 1 to reduce idle connections.

    > **Note** The maxIdle parameter is removed from the configurations of most connection pools. If the maxIdle parameter is available, we recommend that you set this parameter to 1.

  - `maxLifetime`: specifies the maximum time-to-live (TTL) of each connection in connection pools. We recommend that you set this parameter to 60 minutes. This way, you can reduce the probability of out of memory (OOM) errors that occur due to frequent connections to RelCache.
  - `maximumPoolSize`: specifies the maximum number of connections that are allowed in each connection pool. We recommend that you set this parameter to 15. A connection pool that supports up to 15 connections is suitable for most business scenarios. If the number of cached connections in a connection pool is small and your RDS instance processes only the workloads from the connections, you can set the maximumPoolSize parameter to a value greater than 15 on the database clients.

> **Note**
>
> We recommend that you use the following configurations for connection pooling:
>
> - We recommend that you use the following configuration for the HikariCP connection pool, which is the recommended connection pool in Java environments:
>
>   ```
>   minimumIdle=1, maximumPoolSize=15, idleTimeout=600000 (10 minutes), maxLifetime=3600000 (60 minutes)
>   ```
>
> - We recommend that you use the following configuration for the GORM connection pool, which is the recommended connection pool in GO environments:
>
>   ```
>   sqlDB.SetMaxIdleConns(1), sqlDB.SetMaxOpenConns(15), sqlDB.SetConnMaxLifetime(time.Hour)
>   ```
>
> - We recommend that you use the following configuration for the Druid connection pool, which is used in Java environments:
>
>   ```
>   initialSize=1, minIdle=1, maxIdle=1, maxActive=15, testOnBorrow=false, testOnReturn=false, testWhileIdle=true,minEvictableIdleTimeMillis=600000 (10 minutes), maxEvictableIdleTimeMillis=900000 (15 minutes), timeBetweenEvictionRunsMillis=60000 (1 minutes), maxWait=6000 (6 seconds).
>   ```
>
> The preceding configurations do not include PreparedStatement objects. You must configure PreparedStatement objects based on your business requirements.

Performance and stability

- We recommend that you do not create more than 5,000 tables in a single database and make sure that the number of subpartitions for a single partitioned table does not exceed 128. If you partition tables by time, we recommend that you use a granularity of month or year. We recommend that you do not partition tables by day. This way, you can reduce the memory consumption for establishing frequent connections to RelCache.
- We recommend that you use **CREATE INDEX CONCURRENTLY** to create indexes for online workloads. This way, you can prevent the DML **INSERT**, **UPDATE**, and **DELETE** operations that are performed in other sessions on the table for which indexes are created from being blocked.
- We recommend that you use **REINDEX CONCURRENTLY** to re-create indexes for RDS instances that run PostgreSQL 12 or a later version. For RDS instances run PostgreSQL 11 or an earlier version, we recommend that you use **CONCURRENTLY** to create indexes before you delete the original indexes.
- Do not frequently create or delete temporary tables. This way, you can reduce the consumption of system table resources. Proceed with caution when you use **ON COMMIT DROP**. In most cases, you can use the **WITH** clause instead of creating temporary tables.
- Compared with the previous versions, PostgreSQL 13 is optimized to improve partitioned tables, HashAggregate operations for GROUP BY clauses, and parallel queries. We recommend that you upgrade the major engine version of your RDS instance to PostgreSQL 13. For more information, see Upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance.
- If you no longer use the cursor feature, we recommend that you disable this feature.
- We recommend that you execute the **TRUNCATE** statement rather than the **DELETE** statement on tables to improve the performance of your RDS instance.
- PostgreSQL supports the execution and rollback of DDL transactions. We recommend that you encapsulate DDL statements in transactions. This way, you can roll back DDL statements based on your

business requirements. Take note that you must encapsulate DDL statements in transactions of appropriate lengths. If the transactions are long, the read operations on the objects that are being accessed by these transactions may be blocked for a long period of time.

- If you want to write a large amount of data to your RDS instance, we recommend that you run the copy command or execute the `INSERT INTO table VALUES (),(),...();` statement to increase the writing speed.

Minor engine version

- If you want to use the Replication Slot feature, we recommend that you update the minor engine version of your RDS instance to 20201230 or later. In 20201230 and later minor engine versions, you can enable the Logical Replication Slot Failover feature and configure an alert rule for the Maximum Replication Slot Latency metric to prevent logical subscriptions from being delayed or interrupted. If logical subscriptions are delayed or interrupted, logical replication slots are lost and write-ahead logging (WAL) records may pile up. For more information, see Logical Replication Slot Failover and Manage the alert rules of an ApsaraDB RDS for PostgreSQL instance.
- If you enable the audit log feature or the Performance Insight feature, we recommend that you update the minor engine version of your RDS instance to 20211031 or later.

> **Note** We recommend that you set the `log_statement` parameter to all. This way, you can improve the performance of your RDS instance by approximately four times in scenarios in which more than 50 active connections are established. However, if the minor engine version of your RDS instance is earlier than 20211031, the CPU utilization of your RDS instance abruptly increases after you set the `log_statement` parameter to all.

Monitoring and alerting

- We recommend that you turn on the Initiative Alert switch for your RDS instance in the ApsaraDB RDS console to enable the default alert rules that are provided by the monitoring and alerting feature. For more information, see Manage the alert rules of an ApsaraDB RDS for PostgreSQL instance.

Troubleshooting

- For more information about how to identify the SQL statements that consume the most CPU, memory, or I/O resources, see Locate SQL statements with the highest resource consumption.

Design

Permission design

- We recommend that you manage permissions at the schema level or the role level and create the following two roles for your RDS instance in compliance with the principle of least privilege (PoLP): one role with the read and write permissions and one role with only the read permissions. For more information, see Manage permissions in an ApsaraDB RDS for PostgeSQL instance.
- If you enable read/write splitting at the application layer, we recommend that you follow PoLP and use the read-only role for read-only database clients.

Table design

- The data types that are defined for the fields of the schema in your RDS instance must be the same as the data types that are defined in your application. In addition, the same rules must be used to check fields for all tables. This way, you can prevent errors and make sure that you can use indexes.
- If you want to delete historical data on a regular basis, we recommend that you partition tables by time. We also recommend that you execute the `DROP` or `TRUNCATE` statement on tables to delete

data. We recommend that you do not execute the `DELETE` statement on tables to delete data.

- If you want to frequently update a table, we recommend that you set the `FILLFACTOR` parameter of the table to 85 and reserve 15% of the available storage per page when you create the table. The reserved storage is used to update the hot data in the table.

```
CREATE TABLE test123(id int, info text) WITH(FILLFACTOR=85);
```

- We recommend that the names of temporary tables start with `tmp_`. We also recommend that the names of child tables end with the rule based on which the parent table of the child tables is partitioned. For example, if the name of a parent table that is partitioned by year is tbl, the names of the child tables of the parent table can be tbl_2016 and tbl_2017.

Index design

- A B-tree index can contain fields whose total size is up to 2,000 bytes. If the total size of the fields exceeds 2,000 bytes, a new index is required. We recommend that you create a function index, such as a hash index. If you do not create a function index, we recommend that you use an analyzer to analyze the data before you create an index on the data.
- Data, such as streaming data, time fields, and auto-increment fields, may be stored in a linear order. In most cases, range queries are run to query these types of data. We recommend that you create `BRIN` indexes to reduce the size per index and speed up data insertion.

```
CREATE INDEX idx ON tbl using BRIN(id);
```

- We recommend that you do not run full table scans, except when you want to scan and analyze a large amount of data. ApsaraDB RDS for PostgreSQL supports indexes of most data types.

  The following types of indexes are supported: B-tree, Hash, GIN, GiST, SP-GiST, BRIN, RUM, Bloom, and PASE. Among these types of indexes, RUN, Bloom, and PASE are extended indexes.

- We recommend that the names of primary key indexes start with `pk_`, the names of unique indexes start with `uk_`, and the names of regular indexes start with `idx_`.

Data type design and character set design

- We recommend that you select a suitable data type for the data you want to write. If you want to write numeric data or the data that you want to write can be stored in tree structures, we recommend that you do not select the string data type.

  A suitable data type increases query efficiency.

  ApsaraDB RDS for PostgreSQL supports the following data types: Numeric, Floating-Point, Monetary, String, Character, Binary, Date/Time, Boolean, Enumerated, Geometry, Network Address, Bit String, Text Search, UUID, XML, JSON, Array, Composite, Range, Object identifier, row number, large object, ltree structure, Data Cube, geography, H-Store, pg_trgm module, PostGIS, and HyperLogLog. PostGIS includes data types such as point, line segment, surface, path, latitude, longitude, raster, and topology. HyperLogLog is a fixed-size, set-like data structure that is used to count distinct values at a tunable precision.

- We recommend that you set LC_COLLATE to C rather than UTF8. The UTF8 character set collation is inferior to the C character set collation. In addition, if you use the UTF8 character set collation, you must specify the text_pattern_ops operator class for indexes to support LIKE queries.

Stored procedure design

- If the business logic is lengthy, we recommend that you reduce the number of interactions between your application and your RDS instance. We recommend that you use stored procedures, such as stored procedures that are based on PL/pgSQL, or built-in functions. PL/pgSQL is a procedural programming language that is supported by PostgreSQL and is used to process complex business logic. PostgreSQL

supports the following common functions and complex functions: analytic functions, aggregate functions, window functions, mathematical functions, and geometric functions.

Data query

- We recommend that you do not replace `COUNT(column_name)` or `COUNT(constants)` with `COUNT(*)`. `COUNT(*)` is a standard function that is defined in SQL-92 to count the number of rows. COUNT(*) counts in NULL values when it calculates the actual number of rows, whereas `COUNT(column_name)` does not count in NULL values.
- If `COUNT(DISTINCT)` is used, the names of the multiple columns that you want to specify must be enclosed in a pair of parentheses (). Example: `COUNT( (col1,col2,col3) )`. `COUNT(DISTINCT)` counts in all NULL values. Therefore, COUNT(DISTINCT) produces the same result as `COUNT(*)`.
- We recommend that you do not use `SELECT * FROM t`. Replace the wildcard (`*`) with an array of fields that you require. This way, ApsaraDB RDS returns only the fields that you specify and does not return the fields that you do not require.
- We recommend that you prevent ApsaraDB RDS from returning large amounts of data to database clients, except for extract, transform, and load (ETL) operations. If the amount of data that is returned for a query is abnormally large, check whether the execution plan of the query is optimal.
- If you want to perform range queries, we recommend that you use the Range data type and GiST indexes to improve query performance.
- If your application frequently initiates queries for which a large number of results are returned, we recommend that you aggregate all results of such a query into a result set. For example, if the number of results that are returned for a query reaches 100, we recommend that you aggregate the 100 results of the query into a result set. In addition, if your application frequently accesses the results in the result set by ID, we recommend that you aggregate the results by ID on a regular basis. A small number of results returned indicates shorter response time.

Instance management

- We recommend that you enable the SQL Explorer and Audit feature for your RDS instance. This feature allows you to query and export the information about the SQL statements that are executed on your RDS instance. The information includes the databases on which the SQL statements are executed, the status of the SQL statements, and the execution durations of the SQL statements. You can use this feature to diagnose the health status of the SQL statements, troubleshoot performance issues, and analyze business traffic. For more information, see Use the SQL Explorer and Audit feature on an ApsaraDB RDS for PostgreSQL instance.
- If you want to monitor and record the activities within your Alibaba Cloud account, we recommend that you use ActionTrail. The activities that you can monitor and record include access to and use of cloud products and services by using the Alibaba Cloud Management Console, open API, and developer tools. ActionTrail records these actions as events. You can download the events from the ActionTrail console or configure ActionTrail to deliver the events to Log Service Logstores or Object Storage Service (OSS) buckets. Then, you can perform operations, such as action analysis, security analysis, resource change tracking, or compliance audit based on the events. For more information, see What is ActionTrail?.
- DDL operations must be reviewed before they are performed. Make sure that you execute DDL operations during off-peak hours.
- Before you commit the transactions that are run to delete or modify data, we recommend that you execute the `SELECT` statement to confirm the transactions. This way, you can prevent accidental operations. If you want to update only one row based on your business logic, add `LIMIT 1`.
- If you want to perform DDL operations or other similar operations that may acquire locks on specific objects, we recommend that you configure a lock wait mechanism to prevent these operations from

blocking queries on the locked objects. Such operations include `VACUUM FULL` and `CREATE INDEX`.

```
begin;
SET local lock_timeout = '10s';
-- DDL query;
end;
```

- You can execute the **EXPLAIN ANALYZE** statement to view the execution plan of a query. The EXPLAIN ANALYZE statement and the **EXPLAIN** statement work in a similar way. However, the EXPLAIN ANALYZE statement may involve data changes. If the execution plan of a query involves operations such as DML **UPDATE**, **INSERT**, or **DELETE** operations that cause data changes, you must execute the **EXPLAIN ANALYZE** statement in the transaction, and roll the transaction back after the statement is executed.

  ```
  begin;
  EXPLAIN (ANALYZE) <DML(UPDATE/INSERT/DELETE) SQL>;
  rollback;
  ```

- If you want to delete data or update a large amount of data, we recommend that you divide the data into batches and delete or update each batch of data in an independent transaction. We recommend that you do not delete or update all data in one transaction. If you delete or update all data in one transaction, a large amount of junk data is generated.

9.Migration to Cloud

9.1. Use scenarios

ApsaraDB RDS for PostgreSQL provides the cloud migration feature. This feature uses physical streaming replication to accelerate and simplify cloud migration in various business scenarios. You can use this feature to migrate the data of a self-managed PostgreSQL instance that is deployed on an Alibaba Cloud Elastic Compute Service (ECS) instance or in a data center to an ApsaraDB RDS for PostgreSQL instance. You can also use this feature to migrate the backup files of an ApsaraDB RDS for PostgreSQL instance across regions or accounts. This topic describes the scenarios in which you can use the cloud migration feature.

The following table describes the scenarios.

Scenarios	Source instance	Destination instance	Migration link	References
• Migration to the cloud Migrate the data of a self-managed PostgreSQL instance to an ApsaraDB RDS for PostgreSQL instance. • Read capability expansion in the cloud Use an ApsaraDB RDS for PostgreSQL instance to offload read requests from a self-managed PostgreSQL instance. • Disaster recovery in the cloud Use an ApsaraDB RDS for PostgreSQL instance to run as a hot standby for a self-managed PostgreSQL instance. If the self-managed PostgreSQL instance fails, you can manually switch your workloads over from the self-managed PostgreSQL instance to the ApsaraDB RDS for PostgreSQL instance. • Internet-based data migration Migrate the data of a PostgreSQL instance that is connected by using a public IP address or provided by a third-party cloud service provider to an ApsaraDB RDS for PostgreSQL instance. **Note** A PostgreSQL instance that is provided by a third-party cloud service provider can be a self-managed PostgreSQL instance or a managed PostgreSQL instance, such as a Google Cloud SQL instance or an Amazon RDS for PostgreSQL instance.	• Self-managed PostgreSQL instance that is deployed on an Alibaba Cloud ECS instance • Self-managed PostgreSQL instance that is deployed in a data center • PostgreSQL instance that is connected by using a public IP address or provided by a third-party cloud service provider	ApsaraDB RDS for PostgreSQL instance	VPC and Internet • If the self-managed PostgreSQL instance is deployed on an ECS instance, the ECS instance and the ApsaraDB RDS for PostgreSQL instance must reside in the same virtual private cloud (VPC). If the ECS instance and the ApsaraDB RDS for PostgreSQL instance reside in different VPCs, you must use Cloud Enterprise Network (CEN) to connect the VPCs. For more information, see What is CEN? • If the self-managed PostgreSQL instance is deployed in a data center, you must use CEN, VPN Gateway, Express Connect, or Smart Access Gateway to connect the data center and the ApsaraDB RDS for PostgreSQL instance over an internal network. • If the source PostgreSQL instance is connected by using a public IP address or provided by a third-party cloud service provider, you can migrate data from the source PostgreSQL instance to your ApsaraDB RDS for PostgreSQL instance over the Internet.	Use the cloud migration feature for an ApsaraDB RDS for PostgreSQL instance

Scenarios	Source instance	Destination instance	Migration link	References
• Cross-region migration Migrate the data of an ApsaraDB RDS for PostgreSQL instance to another ApsaraDB RDS for PostgreSQL instance that resides in a different region. • Geo-disaster recovery Add ApsaraDB RDS for PostgreSQL instances that reside in different regions to a group to improve disaster recovery capabilities.	ApsaraDB RDS for PostgreSQL instance	ApsaraDB RDS for PostgreSQL instance that is created within the same Alibaba Cloud account but resides in a different region	VPC You must use CEN to connect the ApsaraDB RDS for PostgreSQL instances over an internal network.	Migrate data between ApsaraDB RDS for PostgreSQL instances that reside in different regions
Cross-account migration Migrate the data of an ApsaraDB RDS for PostgreSQL instance to another ApsaraDB RDS for PostgreSQL instance that is created within a different Alibaba Cloud account.	ApsaraDB RDS for PostgreSQL instance	ApsaraDB RDS for PostgreSQL instance that is created within a different Alibaba Cloud account	VPC You must use CEN to connect the ApsaraDB RDS for PostgreSQL instances over an internal network.	Migrate data between ApsaraDB RDS for PostgreSQL instances within different accounts
Instance configuration downgrade Migrate the data of an ApsaraDB RDS for PostgreSQL instance to another ApsaraDB RDS for PostgreSQL instance that has fewer storage resources in full and incremental synchronization modes. Then, interchange the endpoints of the RDS instances.	ApsaraDB RDS for PostgreSQL instance	ApsaraDB RDS for PostgreSQL instance	VPC The source and destination ApsaraDB RDS for PostgreSQL instances must reside in the same VPC.	Scale down an ApsaraDB RDS for PostgreSQL instance

9.2. Preparations for cloud migration

9.2.1. (Optional) Configure an ECS security group on a self-managed PostgreSQL instance

The cloud migration feature of ApsaraDB RDS for PostgreSQL allows you to migrate the data of a self-managed PostgreSQL instance that is deployed on an Elastic Compute Service (ECS) instance to an ApsaraDB RDS for PostgreSQL instance. This topic describes how to configure an ECS security group on a self-managed PostgreSQL instance before a cloud migration to allow an ApsaraDB RDS for PostgreSQL instance to access the self-managed PostgreSQL instance.

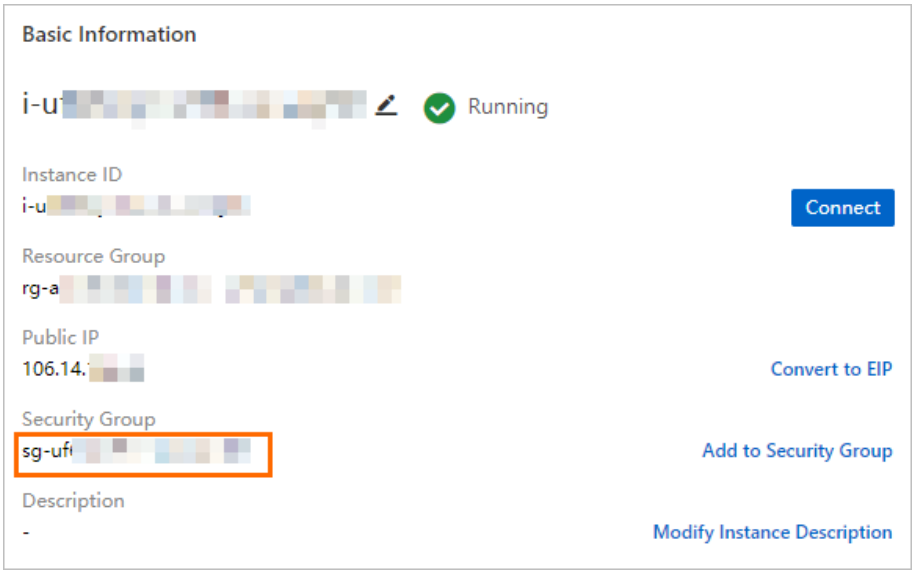
Prerequisites

If you want to migrate the data of a self-managed PostgreSQL instance that is deployed on an ECS instance to an ApsaraDB RDS for PostgreSQL instance, you must perform the configurations described in this topic. The ECS instance on which the self-managed PostgreSQL instance is deployed must meet the following requirements:

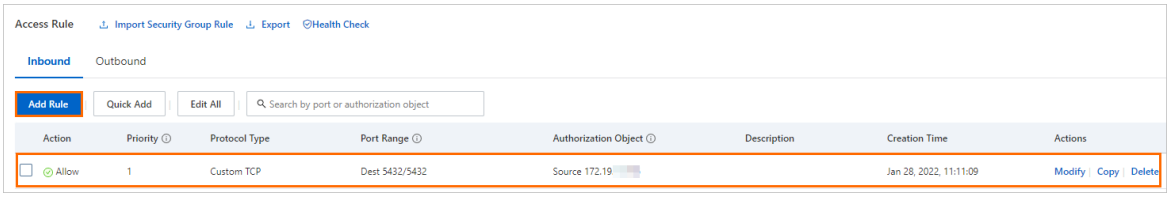
- Network requirements:
  - The ECS instance and the RDS instance reside in the same virtual private cloud (VPC).
  - If you want to migrate data over the Internet, the ECS instance is assigned a public IP address.
- The self-managed PostgreSQL instance runs as expected on the ECS instance.

Procedure

1. Log on to the ECS console.
2. In the left-side navigation pane, choose **Instances & Images > Instances**.
3. In the top navigation bar, select the region where the ECS instance resides.
4. Find the ECS instance and click the instance ID.
5. In the **Basic Information** section of the **Instance Details** tab, click the security group link below the Security Group.



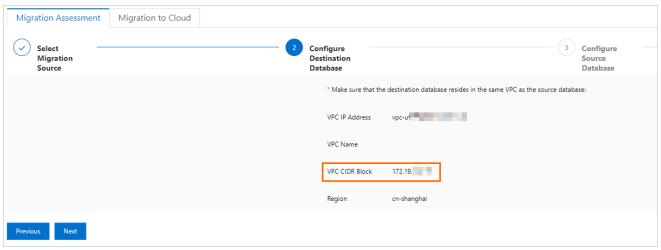
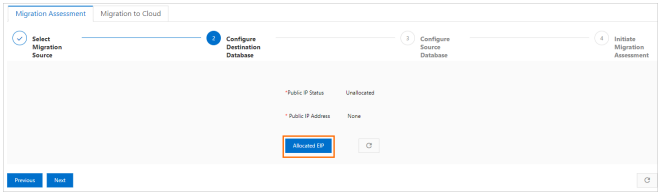
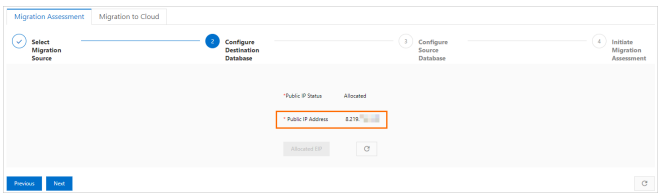
6. In the **Access Rule** section of the **Security Group Rules** page, click **Add Rule** on the **Inbound** tab. Then, add a security group rule.



The following table describes the parameters in a security group rule.

Parameter	Description
Protocol Type	Set the value to TCP.

Parameter	Description
Port Range	Specify the port that is used to connect to the self-managed PostgreSQL instance on the ECS instance. You can run the `netstat -a grep PGSQL` command to query the port.

Parameter	Description
Authorization Object	◦ If you want to migrate data over an internal network, enter the value of the **VPC CIDR Block** parameter. To view the VPC CIDR block, perform the following operations: a. b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab. c. In the **Select Migration Source** step of the configuration wizard, select **Self-managed ECS-based PostgreSQL Database or ApsaraDB RDS for PostgreSQL Instance** and click **Next**. d. In the **Configure Destination Database** step of the configuration wizard, view the value of the **VPC CIDR Block** parameter.  ◦ If you want to migrate data over the Internet, enter the value of the **Public IP Address** parameter. To view the Public IP address, perform the following operations: a. b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab. c. In the **Select Migration Source** step of the configuration wizard, select **PostgreSQL migration with public network address (including migrating from other cloud vendors)** and click **Next**. d. In the **Configure Destination Database** step, click **Allocated EIP**.  e. Refresh the page and check the value of the **Public IP Address** parameter. 

What to do next

Configure a self-managed PostgreSQL instance to listen to remote connections

9.2.2. Configure a self-managed PostgreSQL instance to listen to remote connections

The cloud migration feature of ApsaraDB RDS for PostgreSQL allows you to migrate the data of a self-managed PostgreSQL instance from an Elastic Compute Service (ECS) instance or a data center to an ApsaraDB RDS for PostgreSQL instance. This topic describes how to configure the postgresql.conf file of the self-managed PostgreSQL instance before a cloud migration to allow remote connections to the self-managed PostgreSQL instance.

Procedure

> **Note** In this topic, the self-managed PostgreSQL instance and the ApsaraDB RDS for PostgreSQL instance run PostgreSQL 13 in the CentOS 7 operating system.

1. Connect to the self-managed PostgreSQL instance to check whether the self-managed PostgreSQL instance listens to remote connections.

   ```
   SHOW listen_addresses;
   ```

   The following or similar command output is displayed:

   ```
    listen_addresses
   ------------------
    *
   (1 row)
   ```

   - If the return result is `*`, you do not need to configure the postgresql.conf file. You can proceed to create an account that is used for a cloud migration. For more information, see Create an account for cloud migration on a self-managed PostgreSQL instance.
   - If the return result is not `*`, go to Step 2.

2. Stop the PostgreSQL database service.

   > **Note** Only the postgres user can run the following command. You can run the `su - postgres` command to switch to the postgres user.

   ```
   /usr/pgsql-13/bin/pg_ctl stop -m fast
   ```

3. Find the **postgresql.conf** file.

   > **Note** Only the root user can run the following command.

   ```
   find / -name postgresql.conf
   ```

   The following or similar command output is displayed:

   ```
   /var/lib/pgsql/13/data/postgresql.conf
   ```

4. Open the directory in which the **postgresql.conf** file is stored.

```
cd /var/lib/pgsql/13/data/
```

5. Run the `vim postgresql.conf` command to enable the edit mode. Then, change the value of the listen_addresses parameter in the **postgresql.conf** file to `*`.

```
listen_addresses = '*'
```

> **Note** The listen_addresses parameter is commented out by default. After you modify the postgresql.conf file, you must delete the number sign (`#`) at the beginning of the line in which the parameter resides.

6. Press **ECS** and enter `:wq` to save the postgresql.conf file and exit.
7. Start the PostgreSQL database service.

> **Note** Only the postgres user can run the following command. You can run the `su - postgres` command to switch to the postgres user.

```
/usr/pgsql-13/bin/pg_ctl start
```

What to do next

Create an account for cloud migration on a self-managed PostgreSQL instance

9.2.3. Create an account for cloud migration on a self-managed PostgreSQL instance

The cloud migration feature of ApsaraDB RDS for PostgreSQL instance allows you to migrate the data of a self-managed PostgreSQL instance to an Elastic Compute Service (ECS) instance or a data center to an ApsaraDB RDS for PostgreSQL instance. This topic describes how to create an account for data migration on a self-managed PostgreSQL instance.

Procedure

The account that you create must have the CREATE ROLE, REPLICATION, and pg_monitor permissions. If an account that has these permissions is created on the self-managed PostgreSQL instance, you can proceed to update the pg_hba.conf file. For more information, see Update the pg_hba.conf file of a self-managed PostgreSQL instance.

1. Connect to the self-managed PostgreSQL instance and create an account that is used for cloud migration. In the following example, an account named migratetest is created:

```
CREATE USER migratetest CREATEROLE REPLICATION LOGIN PASSWORD '123456';
```

> **Note** The password of the account in the preceding command is an example. You can specify a custom password.

2. Grant the pg_monitor permission to the migratetest account.

```
GRANT pg_monitor TO migratetest;
```

What to do next

Update the pg_hba.conf file of a self-managed PostgreSQL instance

9.2.4. Update the pg_hba.conf file of a self-managed PostgreSQL instance

The cloud migration feature of ApsaraDB RDS for PostgreSQL allows you to migrate data from a self-managed PostgreSQL instance that is deployed on an Elastic Compute Service (ECS) instance or in a data center to an ApsaraDB RDS for PostgreSQL instance. This topic describes how to update the pg_hba.conf file of the self-managed PostgreSQL instance before a cloud migration to grant access from the CIDR block of the virtual private cloud (VPC) to which the ApsaraDB RDS for PostgreSQL instance belongs.

Procedure

> **Note** In this topic, the self-managed PostgreSQL instance and the ApsaraDB RDS for PostgreSQL instance run PostgreSQL 13 in the CentOS 7 operating system.

1. Log on to the server on which the self-managed PostgreSQL instance resides.
2. Find the **pg_hba.conf** file.

   > **Note** Only the root user can run the following command.

   ```
   find / -name pg_hba.conf
   ```

   The following or similar command output is displayed:

   ```
   /var/lib/pgsql/13/data/pg_hba.conf
   ```

3. Go to the directory in which the **pg_hba.conf** file is stored.

   ```
   cd /var/lib/pgsql/13/data/
   ```

4. Run the `vim pg_hba.conf` command to enable the edit mode and add the following content to the **pg_hba.conf** file.

   ```
   # Migrate data over an internal network.
   host  all       migratetest  172.21.XX.XX/16  md5
   host  replication  migratetest  172.21.XX.XX/16  md5
   # Migrate data over the Internet.
   host  all       migratetest  121.41.XX.XX/32  md5
   host  replication  migratetest  121.41.XX.XX/32  md5
   ```

   The content contains the following parameters:

   - `migratetest`: the account that is used to migrate data. For more information, see Create an account for cloud migration on a self-managed PostgreSQL instance.
   - `172.21.XX.XX/16` or `121.41.XX.XX/32`: the VPC CIDR block or the public IP address of the RDS instance.

- To migrate data over an internal network, you must configure the **VPC CIDR block** of the RDS instance.

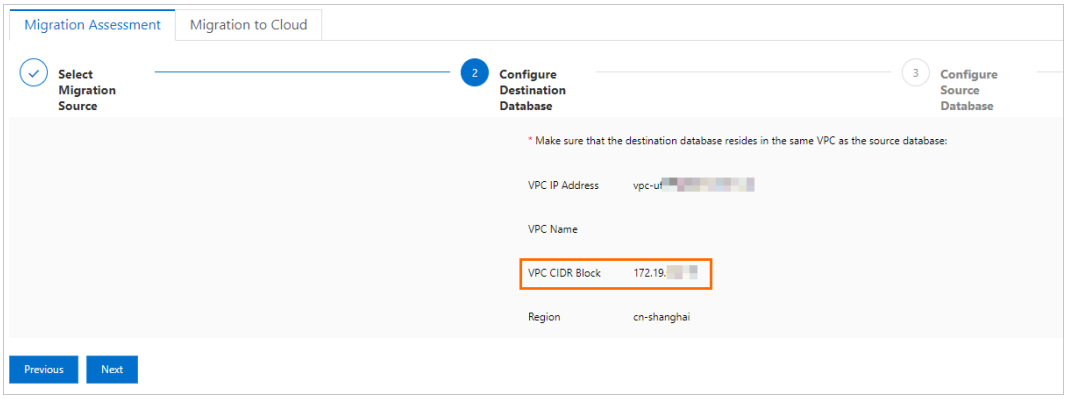
  To view the VPC CIDR block, perform the following operations:

  a.

  b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.

  c. In the **Select Migration Source** step of the configuration wizard, select **Self-managed ECS-based PostgreSQL Database or ApsaraDB RDS for PostgreSQL Instance** or **Self-managed PostgreSQL database in a data center (within the same VPC as the destination database)** and click **Next**.

  d. In the **Configure Destination Database** step of the configuration wizard, view the value of the **VPC CIDR block** parameter.



- If you want to migrate data over the Internet, enter the value of the **Public IP Address** parameter.

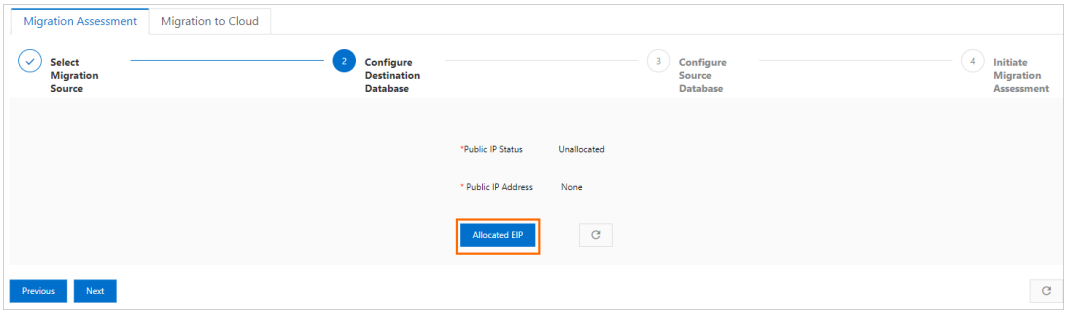
  To view the Public IP address, perform the following operations:

  a.

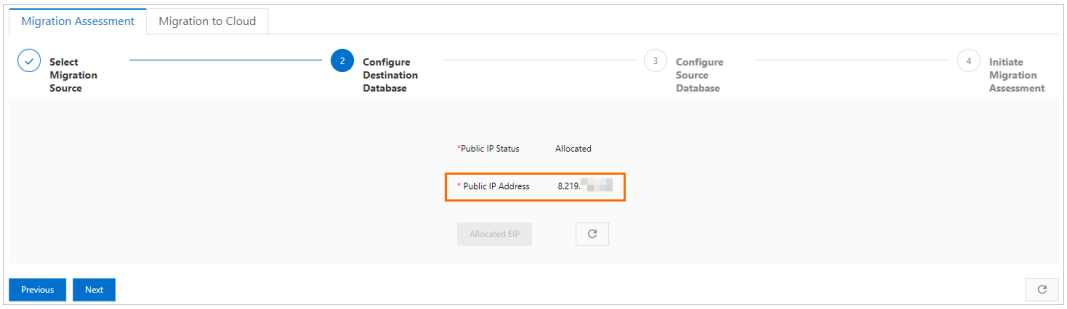
  b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.

  c. In the **Select Migration Source** step of the configuration wizard, select **PostgreSQL migration with public network address (including migrating from other cloud vendors)** and click **Next**.

  d. In the **Configure Destination Database** step, click **Allocated EIP**.



  e. Refresh the page and check the value of the **Public IP Address** parameter.



5. Connect to the self-managed PostgreSQL instance and reload its configuration.

```
SELECT pg_reload_conf();
```

Sample output:

```
 pg_reload_conf
----------------
 t
(1 row)
```

What to do next

Configure the firewall of the server on which a self-managed PostgreSQL instance resides

9.2.5. Configure the firewall of the server on which a self-managed PostgreSQL instance resides

The cloud migration feature of ApsaraDB RDS for PostgreSQL allows you to migrate the data of a self-managed PostgreSQL instance from an Elastic Compute Service (ECS) instance or a data center to an ApsaraDB RDS for PostgreSQL instance. This topic describes how to configure the firewall of the server on which a self-managed PostgreSQL instance resides to allow access to the port of the self-managed PostgreSQL instance before a cloud migration.

Procedure

Note In this topic, the server on which the self-managed PostgreSQL instance resides runs CentOS. For more information about how to configure the firewall of a server that runs a different operating system, see the related official documentation.

CentOS 7

1. Connect to the server on which the self-managed PostgreSQL instance resides.
2. View the ports that are opened.

```
firewall-cmd --list-ports
```

3. Configure the port of the self-managed PostgreSQL instance to allow access to the port.

```
firewall-cmd --zone=public --add-port=5432/tcp --permanent
```

4. Restart the firewall of the server.

```
firewall-cmd --reload
```

CentOS 6 or earlier versions

1. Connect to the server on which the self-managed PostgreSQL instance resides.
2. View the ports that are opened.

```
/etc/init.d/iptables status
```

3. Open port 5432.

```
/sbin/iptables -I INPUT -p tcp --dport 85432 -j ACCEPT
```

4. Restart the firewall of the server.

```
service iptables restart
```

Note You can also disable the firewall of the server before a cloud migration. In this case, you do not need to configure the firewall.

- CentOS 7:

```
systemctl stop firewalld.service
```

- CentOS 6 or earlier versions:

```
service iptables stop
```

What to do next

Use the cloud migration feature for an ApsaraDB RDS for PostgreSQL instance

9.3. Use the cloud migration feature for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to use the cloud migration feature to migrate data from a self-managed PostgreSQL instance that resides on an Elastic Compute Service (ECS) instance or in a data center to an ApsaraDB RDS for PostgreSQL instance. The cloud migration feature of ApsaraDB RDS for PostgreSQL uses physical streaming replication to help you migrate data to the cloud in an easy-to-use, efficient manner and at a high speed with no downtime. This feature is suitable in all scenarios.

Prerequisites

- The RDS instance meets the following requirements:
  - The RDS instance and the self-managed PostgreSQL instance run the same PostgreSQL version, which can be PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, or PostgreSQL 14.

    > **Note** If you want to migrate data from an instance that runs PostgreSQL 10 to an RDS instance that runs PostgreSQL 13, you must use the cloud migration feature to migrate the data from the source PostgreSQL instance to an RDS instance that runs PostgreSQL 10 and then upgrade the major engine version of the RDS instance to PostgreSQL 13 based on the descriptions in Upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance.

  - The RDS instance is a primary instance. Read-only RDS instances do not support cloud migration.
  - The RDS instance is equipped with standard SSDs or enhanced SSDs (ESSDs).
  - The RDS instance is empty. The available storage of the RDS instance is greater than or equal to the size of the data in the self-managed PostgreSQL instance.
  - The RDS instance does not use a new general-purpose instance type.

    > **Note** The new general-purpose instance types provide better scalability and performance and reduce the time to create an RDS instance or change the specifications of an RDS instance. The new general-purpose instance types do not support the cloud migration feature. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types.

- The self-managed PostgreSQL instance meets the following requirements:

- Networking

Migration source	Network configuration
Self-managed ECS-based PostgreSQL Database or ApsaraDB RDS for PostgreSQL Instance	If the self-managed PostgreSQL instance resides on an ECS instance, the ECS instance and the destination RDS instance must reside in the same virtual private cloud (VPC). If the source instance is an RDS instance, the source RDS instance and the destination RDS instance must reside in the same VPC. If the source instance and the destination RDS instance reside in different VPCs, you must use Cloud Enterprise Network (CEN) to connect the VPCs. For more information, see What is CEN?
Self-managed PostgreSQL database in a data center (within the same VPC as the destination database)	The data center must be able to communicate with the VPC to which the destination RDS instance belongs. For more information, see Connect a data center to a VPC.
PostgreSQL migration with public network address (including migrating from other cloud vendors)	The source PostgreSQL instance is assigned with a public IP address.

> **Note** The source PostgreSQL instance from other cloud service providers can be a self-managed PostgreSQL instance in the cloud or a managed PostgreSQL instance, such as a Google Cloud SQL instance or an Amazon RDS for PostgreSQL instance.

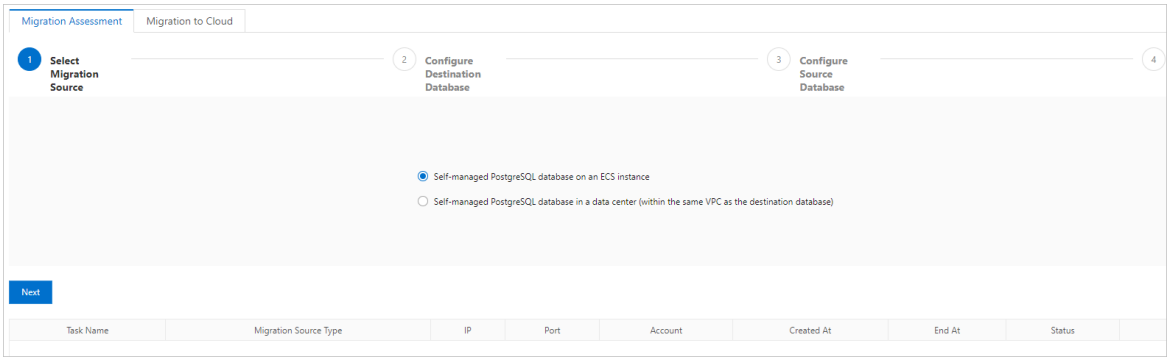
- If the self-managed PostgreSQL instance resides on an ECS instance, an ECS security group is configured. For more information, see (Optional) Configure an ECS security group on a self-managed PostgreSQL instance.
- The configurations that are described in Configure a self-managed PostgreSQL instance to listen to remote connections are complete.
- The configurations that are described in Create an account for cloud migration on a self-managed PostgreSQL instance are complete.
- The configurations that are described in Update the pg_hba.conf file of a self-managed PostgreSQL instance are complete.
- The configurations that are described in Configure the firewall of the server on which a self-managed PostgreSQL instance resides are complete.

Usage notes

During the cloud migration, you can read data from and write data to the self-managed PostgreSQL instance. However, do not perform operations such as migration, restart, or specification changes on the self-managed PostgreSQL instance.

Step 1: Evaluate whether the data migration is allowed

1.
2. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.



3. In the **Select Migration Source** step of the configuration wizard, select a migration source and click **Next**.
4. In the **Configure Destination Database** step of the configuration wizard, click **Next**.
5. In the **Configure Source Database** step of the configuration wizard, select all listed items and click **Next**. You must complete the preparations that are described in the listed items before you start the cloud migration.

   > **Note** For more information about the items that are listed in the **Configure Source Database** step of the configuration wizard, see (Optional) Configure an ECS security group on a self-managed PostgreSQL instance, Create an account for cloud migration on a self-managed PostgreSQL instance, and Update the pg_hba.conf file of a self-managed PostgreSQL instance.

6. In the **Initiate Migration Assessment** step of the configuration wizard, configure the information about the self-managed PostgreSQL instance.

Parameter	Description
Migration Task Name	The name of the cloud migration task. ApsaraDB RDS automatically generates a name for the cloud migration task. You do not need to modify the generated name.
Source VPC/DNS IP or **Source Public/DNS IP**	○ **Source VPC/DNS IP** ■ If the self-managed PostgreSQL instance resides on an ECS instance, enter the **private IP address** of the ECS instance. For more information about how to obtain the private IP address of an ECS instance, see View IP addresses. ■ If the self-managed PostgreSQL instance resides in a data center, enter a private IP address of the data center. ○ **Source Public/DNS IP** If the self-managed PostgreSQL instance resides on a device that is connected over the Internet, enter the public IP address of the device.
Source Port	The port that is used to connect to the self-managed PostgreSQL instance. You can run the `netstat -a | grep PGSQL` command to view the port.

Parameter	Description
Username	The username of the account that is used to connect to the self-managed PostgreSQL instance. Enter `migratetest`, which is the username of the account that you created in the **Configure Source Database** step.
Password	The password of the account that is used to connect to the self-managed PostgreSQL instance. Enter `123456`, which is the password of the account that you created in the **Configure Source Database** step.

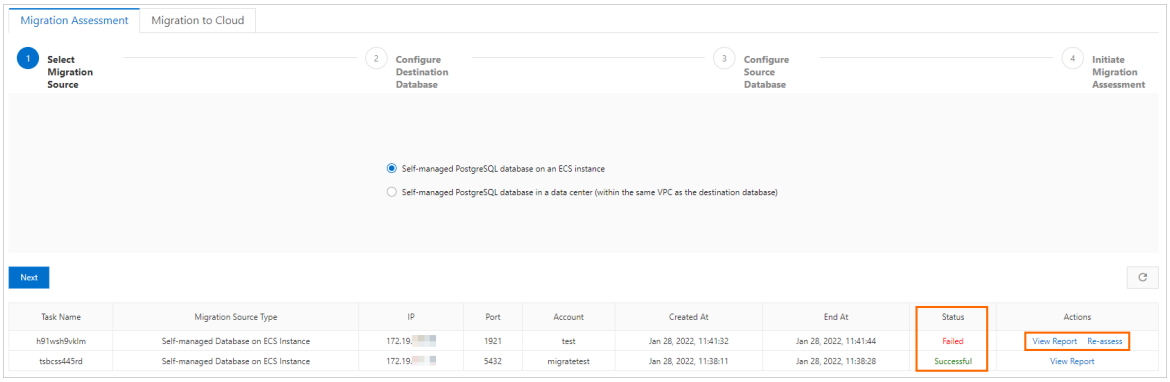
7. Click **Create Migration Assessment Task**.

> **Note** During the cloud migration assessment, the status of the destination RDS instance changes to **Maintaining Instance**.

After the cloud migration assessment is complete, you can view the status of the cloud migration assessment task on the **Migration Assessment** tab.

- If the value in the Status column of the cloud migration assessment task is **Successful**, you can start the cloud migration. For more information, see Step 2: Start the cloud migration.
- If the value in the Status column of the cloud migration assessment task is **Failed**, you can click **View Report** in the Actions column to view and handle the reported errors. For more information about common errors, see Introduction to cloud migration assessment reports.

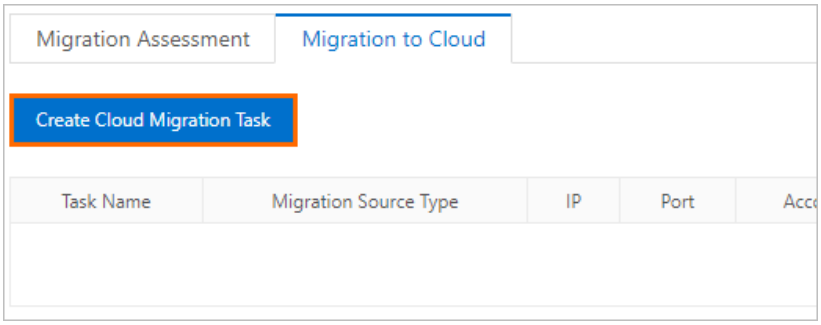
After you handle the reported errors, you can click **Re-access** in the Actions column to run the cloud migration assessment task again.



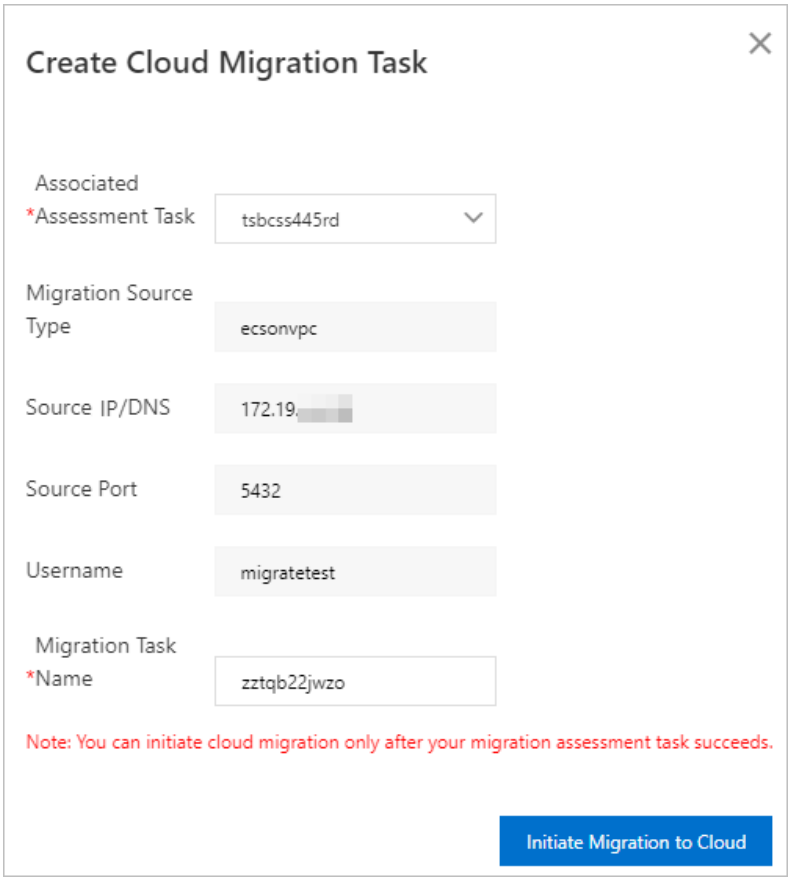
Step 2: Start the cloud migration

> **Note** You can start the cloud migration only when the status of the cloud migration assessment task indicates a success.

1.
2. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration to Cloud** tab. On the tab that appears, click Create Cloud Migration Task.



3. In the **Create Cloud Migration Task** dialog box, select the cloud migration assessment task whose status indicates a success in Step 1: Evaluate whether the data migration is allowed from the **Associated Assessment Task** drop-down list.



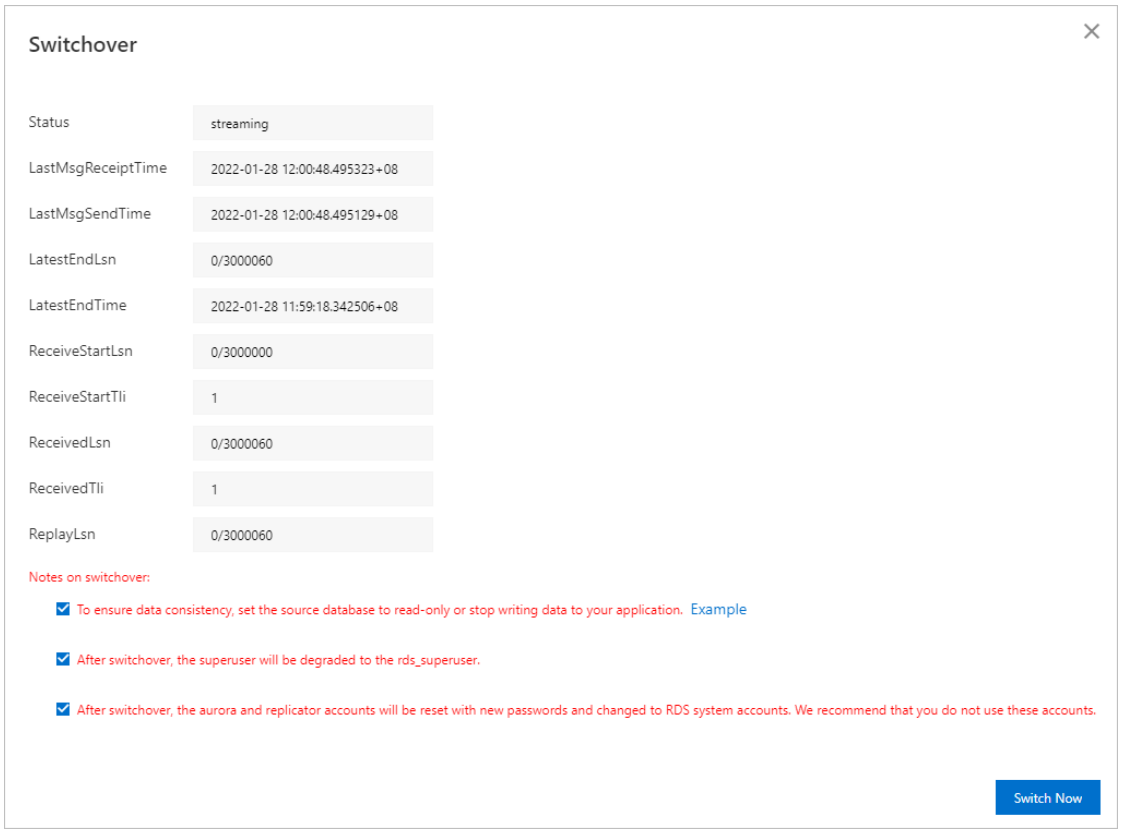
> **Note** After you select a cloud migration assessment task from the **Associated Assessment Task** drop-down list, ApsaraDB RDS automatically obtains the values of the **Migration Source Type**, **Source IP/DNS**, **Source Port**, and **Username** parameters. You do not need to manually configure these parameters.

4. Click **Initiate Migration to Cloud**. ApsaraDB RDS automatically starts the cloud migration task.

> **Notice** During the cloud migration, the status of the destination RDS instance changes to **Migrating Data In**. You can read data from and write data to the self-managed PostgreSQL instance. However, do not perform operations such as migration, restart, or specification changes on the self-managed PostgreSQL instance.

5. Switch the workloads of the self-managed PostgreSQL instance to the destination RDS instance.
   i. Click the link in the **Cloud Migration Phase** column of the cloud migration task to view the task progress.

   ii. When the cloud migration task enters the phase of **Incremental Data Synchronization**, click **Switchover** in the **Actions** column of the cloud migration task to switch the workloads of the self-managed PostgreSQL instance to the destination RDS instance.
   iii. In the **Switchover** dialog box, configure the self-managed PostgreSQL instance to process only read requests. Alternatively, stop the connected application from writing data to the self-managed PostgreSQL instance.



> **Note** To configure the self-managed PostgreSQL instance to process only read requests, you must run the following command by using the credentials of a superuser account:

```
-- Sets a database to read-only.
ALTER SYSTEM SET default_transaction_read_only=on;
-- Reloads the parameter configuration for the modification to take effect.
SELECT pg_reload_conf();
-- Terminates all existing sessions.
SELECT pg_terminate_backend(pid) FROM pg_stat_activity
WHERE usename not in ('replicator', 'monitor', 'pgsql', 'aurora') AND pid != pg_ba
ckend_pid();
```

   iv. Select all check boxes and click **Switch Now**. Then, wait until the cloud migration is complete.

Task Name	Migration Source Type	IP	Port	Account	Created At	Switched At	Last Modified At	Cloud Migration Phase	Data Synchronization Progress	Task Status	Actions
zztqb22jwzo	Self-managed Database on ECS Instance	172.19	5432	migratetest	Jan 28, 2022, 11:51:56	Jan 28, 2022, 12:05:25	Jan 28, 2022, 12:06:08	Migration Completed	/	Successful	View Logs

9.4. Introduction to cloud migration assessment reports

This topic describes the assessment report of a cloud migration from a self-managed PostgreSQL instance to an ApsaraDB RDS for PostgreSQL instance. This topic also describes the common errors in the report and the solutions to the errors.

If the **status** of a cloud migration assessment report is **failure**, you can click **View report** in the **operation** column to view the details about the report. The report contains the following details:

- Check rds empty (Check whether the databases in the ApsaraDB RDS for PostgreSQL instance are empty)
- Check source connectivity (Check whether communication between the self-managed PostgreSQL instance and the ApsaraDB RDS for PostgreSQL instance is normal)
- Check source version (Check the major engine version of the self-managed PostgreSQL instance)
- Check source glibc version (Check the GNU C Library version of the self-managed PostgreSQL instance)
- Check disk size (Check whether the available storage of the ApsaraDB RDS for PostgreSQL instance is sufficient)
- Check wal keep size (Check the value of the wal_keep_size parameter)
- Check spec params (Check the settings of specifications-related parameters)
- Check rds user (Check whether the system accounts of the ApsaraDB RDS for PostgreSQL instance are used in the self-managed PostgreSQL instance)
- Check extensions (Check the compatibility of plug-ins)
- Check triggers (Check whether triggers are compatible)
- Check user functions (Check user-defined functions)

> **Note** For more information about a cloud migration to an ApsaraDB RDS for PostgreSQL instance, see Use the cloud migration feature for an ApsaraDB RDS for PostgreSQL instance.

Check rds empty (Check whether the databases in the ApsaraDB RDS for PostgreSQL instance are empty)

Check item:

Check rds databases

Common error:

```
error:postgres not empty, check if any table exists
```

Description:

Databases are created in the ApsaraDB RDS for PostgreSQL instance, and the databases contain data.

Solution:

Delete all databases except the template0, template1, and postgres databases from the ApsaraDB RDS for PostgreSQL instance. In addition, delete all tables except the ha_health_check table from the postgres database.

Check source connectivity (Check whether communication between the self-managed PostgreSQL instance and the ApsaraDB RDS for PostgreSQL instance is normal)

- **Check item 1:**

  Check ip connectable

  Common error:

  ```
  error:XX.XX.XX.XX is unapproachable
  ```

  Description:

  The IP address of the server on which the self-managed PostgreSQL instance resides or the IP addresses of the DNS servers are inaccessible.

  Solution:

  - If the self-managed PostgreSQL instance resides on an Elastic Compute Service (ECS) instance, enter the **private IP address** of the ECS instance when you configure the source database information. For more information about how to obtain the private IP address of an ECS instance , see View IP addresses.
  - If the self-managed PostgreSQL instance resides in a data center, enter the IP addresses of the DNS servers of the host on which the self-managed PostgreSQL instance resides when you configure the source database information.

- **Check item 2:**

  Check port connectable

  Common error:

  ```
  error:5432 is unapproachable
  ```

  Description:

  - The self-managed PostgreSQL instance is not configured to listen to remote connections.
  - The firewall that is configured for the self-managed PostgreSQL instance does not allow access to the port of the self-managed PostgreSQL instance.

  Solution:

  - Add the following content to the **postgresql.conf** file of the self-managed PostgreSQL instance: `listen_addresses = '*'` . For more information, see Configure a self-managed PostgreSQL instance to listen to remote connections.
  - Configure the firewall to allow access to port 5432. Alternatively, disable the firewall before you start the cloud migration. For more information, see Configure the firewall of the server on which a self-managed PostgreSQL instance resides.

- **Check item 3:**

  Check database connectable

  Common error:

  ```
  error:cannot connect to source database by migratetest:123456
  ```

  Description:

- The password that you entered is incorrect.
- The configuration in the **pg_hba.conf** file of the self-managed PostgreSQL instance is incorrect.

Solution:

- Check whether you can connect to the self-managed PostgreSQL instance by using the username and password that you entered. If the connection fails, you can update the password. For example, if you use the migratetest account, run the following command to update the password of the migratetest account:

```
ALTER USER migratetest WITH PASSWORD '123456';
```

- Modify the **pg_hba.conf** file of the self-managed PostgreSQL instance. For example, if you use the migratetest account, add the following content to the file:

```
host  all       migratetest  <The CIDR block of the VPC to which the ApsaraDB RDS for PostgreSQL instance belongs>  md5
```

> **Note** For more information, see Update the pg_hba.conf file of a self-managed PostgreSQL instance.

- **Check item 4:**

  Check account replication privilege

  Common error:

```
error:migratetest has no replication privilege
```

  Description:

  - The account that you use does not have the REPLICATION permission.
  - The configuration in the **pg_hba.conf** file of the self-managed PostgreSQL instance is incorrect.

  Solution:

  - Grant the REPLICATION permission to the account that you use. For example, if you use the migratetest account, run the following command to grant the REPLICATION permission to the migratetest account:

```
ALTER ROLE migratetest REPLICATION;
```

  - Modify the **pg_hba.conf** file of the self-managed PostgreSQL instance. For example, if you use the migratetest account, add the following content to the file:

```
host  replication  migratetest  <The CIDR block of the VPC to which the ApsaraDB RDS for PostgreSQL instance belongs>  md5
```

> **Note** For more information, see Update the pg_hba.conf file of a self-managed PostgreSQL instance.

- **Check item 5:**

  Check account createrole privilege

  Common error:

```
error:migratetest has no createrole privilege
```

Description:

The account that you use does not have the CREATEROLE permission.

Solution:

Grant the CREATEROLE permission to the account that you use. For example, if you use the migratetest account, run the following command to grant the CREATEROLE permission to the migratetest account:

```
ALTER ROLE migratetest CREATEROLE;
```

- **Check item 6:**

  Check account monitor privilege

  Common error:

  ```
  error:migratetest should be a member of pg_monitor to monitor replication status
  ```

  Description:

  The account that you use does not have the pg_monitor permission. pg_stat_wal_receiver

  > **Note** The pg_monitor permission is used to query system views such as pg_stat_replication and pg_stat_wal_receiver and obtain the information about the replication link.

  Solution:

  Grant the pg_monitor permission to the account that you use. For example, if you use the migratetest account, run the following command to grant the pg_monitor permission to the migratetest account:

  ```
  GRANT pg_monitor TO migratetest;
  ```

Check source version (Check the major engine version of the self-managed PostgreSQL instance)

Check item:

Check major version consistent

Common error:

```
error:version mismatch, source version:10, current version:13.0
```

Description:

The self-managed PostgreSQL instance and the ApsaraDB RDS for PostgreSQL instance run different major engine versions.

Solution:

Purchase an ApsaraDB RDS for PostgreSQL instance that runs the same major engine version as the self-managed PostgreSQL instance.

Check source glibc version (Check the GNU C Library version of the self-managed PostgreSQL instance)

Check item:

Check source glibc version compatible

Common error:

```
warning:source glibc version is not compatible with rds pg
```

Description:

The GNU C Library version of the self-managed PostgreSQL instance is incompatible with the GNU C Library version of the ApsaraDB RDS for PostgreSQL instance.

> **Note** Version 2.28 of the GNU C Library uses a few different collations to sort character sets in UTF-8 encoding. If the GNU C Library version of the self-managed PostgreSQL instance is incompatible with the GNU C Library version of the ApsaraDB RDS for PostgreSQL instance, the character sets in the ApsaraDB RDS for PostgreSQL instance may be in an unexpected order after the cloud migration.

Solution:

Perform the following steps:

1. Check the collations of tables.

   ```
   begin;
   create temp table testcollation(id varchar(20) collate "en_US.utf8") on commit drop;
   insert into testcollation values('-1'),('1');
   select id='1' from testcollation order by id limit 1;
   rollback;
   ```

   The following results can be returned:

   - If true is returned, no further actions are required. The cloud migration does not have risks.
   - If false is returned, proceed with the subsequent steps.

2. Check the collations of databases.

   ```
   SELECT datname,datcollate FROM pg_database where datcollate NOT IN ('C', 'POSIX');
   ```

   The following results can be returned:

   - If no result is returned, no further actions are required. The cloud migration does not have risks.
   - If a result is returned, proceed with the subsequent steps.

3. Check all databases to find indexes whose collations are not `C` or `POSIX`.

```
WITH result AS (
    WITH defcoll AS (
        SELECT datcollate AS coll
        FROM pg_database
        WHERE datname = current_database()
    )
    SELECT indrelid::regclass::text relname, indexrelid::regclass::text indexname,
        CASE WHEN c.collname = 'default'
            THEN defcoll.coll
            ELSE c.collname
        END AS collation
    FROM (SELECT indexrelid, indrelid, indcollation[i] coll FROM pg_index, generate_subsc
ripts(indcollation, 1) g(i)) s
        JOIN pg_collation c ON coll=c.oid
        CROSS JOIN defcoll
    WHERE collprovider IN ('d', 'c') AND collname NOT IN ('C', 'POSIX')
)
SELECT result.relname, result.indexname, result.collation FROM result WHERE result.collat
ion NOT IN ('C', 'POSIX');
```

The following results can be returned:

- If no result is returned, no further actions are required. The cloud migration does not have risks.
- If a result is returned, the cloud migration has risks.

Check disk size (Check whether the available storage of the ApsaraDB RDS for PostgreSQL instance is sufficient)

Check item:

Check disk size enough

Common error:

```
error:source_db_size > disk_size * 0.95
```

Description:

The used storage of the self-managed PostgreSQL instance is greater than 95% of the available storage of the ApsaraDB RDS for PostgreSQL instance. This means that the available storage of the ApsaraDB RDS for PostgreSQL instance is insufficient.

Solution:

1. Run the following command to view the used storage of the self-managed PostgreSQL instance:

   ```
   SELECT SUM(pg_database_size(pg_database.datname))/1024/1024 AS size FROM pg_database;
   ```

   > **Note** The used storage that is returned is measured in the unit of MB.

2. Calculate the available storage that the ApsaraDB RDS for PostgreSQL instance must provide to ensure a successful cloud migration.

   For example, if the used storage of the self-managed PostgreSQL instance is 100 GB, the available storage of the ApsaraDB RDS for PostgreSQL instance must be at least 110 GB.

3. Change the specifications of the ApsaraDB RDS for PostgreSQL instance to expand the storage

capacity. For more information, see 变更配置.

Check wal keep size (Check the value of the wal_keep_size parameter)

Check item:

Check wal keep size large enough

Common error:

```
warning:wal_keep_size X MB is too small. Try to set wal_keep_segments or wal_keep_size large enough ensure pg_basebackup success
```

Description:

The value of the wal_keep_size or wal_keep_segments parameter is small.

Solution:

- If the self-managed PostgreSQL instance runs PostgreSQL 13 or a later version, increase the value of the wal_keep_size parameter for the ApsaraDB RDS for PostgreSQL instance. This way, you can increase the success rate of the full backup and incremental backup during the cloud migration.
- If the self-managed PostgreSQL instance runs a version earlier than PostgreSQL 13, increase the value of the wal_keep_segments parameter for the ApsaraDB RDS for PostgreSQL instance. This way, you can increase the success rate of the full backup and incremental backup during the cloud migration.

> **Note** If you use a PostgreSQL version that is earlier than PostgreSQL 13, the value of the wal_keep_size parameter is equal to the value of the wal_keep_segments parameter multiplied by the value of the wal_segment_size parameter.

Check spec params (Check the settings of specifications-related parameters)

Check item:

Check if spec params too large

Common error:

```
error:max_connections too large, value=XXX
error:max_prepared_transactions too large, value=XXX
```

Description:

The values of the max_connections and max_prepared_transactions parameters for the self-managed PostgreSQL instance are greater than 100 times the values of these parameters for the ApsaraDB RDS for PostgreSQL instance. In this case, the ApsaraDB RDS for PostgreSQL instance may fail to start during the establishment of a replication link.

Solution:

Decrease the values of the max_connections and max_prepared_transaction parameters for the self-managed PostgreSQL instance.

> **Note** After you reconfigure the max_connections and max_prepared_transaction parameters for the self-managed PostgreSQL instance, you must restart the self-managed PostgreSQL instance.

Check rds user (Check whether the system accounts of the ApsaraDB RDS for PostgreSQL instance are used in the self-managed PostgreSQL instance)

Check item:

Check if rds system user is occupied

Common error:

```
warning:Check if rds system user is occupied ...XXX will be reused in rds
```

Description:

The system accounts `aurora`, `replicator`, and `pgxxx` of the ApsaraDB RDS for PostgreSQL instance are used in the self-managed PostgreSQL instance.

Solution:

Make sure that you do not use the preceding accounts in the self-managed PostgreSQL instance.

Check extensions (Check the compatibility of plug-ins)

- **Check item 1:**

  Check source supported extensions

  Common error:

  ```
  error:Check source supported extensions XXX not supported
  ```

  Description:

  The plug-ins of the ApsaraDB RDS for PostgreSQL instance are incompatible with the plug-ins of the self-managed PostgreSQL instance.

  Solution:

  Delete the incompatible plug-ins from the self-managed PostgreSQL instance.

- **Check item 2:**

  Check source extensions with higher version

  Common error:

  ```
  error:Check source extensions with higher version XXX
  ```

  Description:

  The versions of specific plug-ins in the self-managed PostgreSQL instance are later than the versions of these plug-ins in the ApsaraDB RDS for PostgreSQL instance.

  Solution:

  Reinstall these plug-ins in the self-managed PostgreSQL instance and make sure that the versions of these plug-ins in the self-managed PostgreSQL instance are the same as the versions of these plug-ins in the ApsaraDB RDS for PostgreSQL instance.

- **Check item 3:**

  Check source extensions with lower version

Common error:

```
warning:Check source extensions with lower version XXX
```

Description:

The versions of specific plug-ins in the self-managed PostgreSQL instance are earlier than the versions of these plug-ins in the ApsaraDB RDS for PostgreSQL instance.

Solution:

No actions are required. The versions of the plug-ins are automatically updated after the cloud migration.

Check triggers (Check whether triggers are compatible)

Check item:

Check triggers compatible

Common error:

```
Check triggers compatible XXX
```

Description:

ApsaraDB RDS does not report errors for this check item. You can ignore the message that is displayed.

Solution:

N/A.

Check user functions (Check user-defined functions)

Check item:

Check user functions compatible

Common error:

```
Check user functions compatible XXX
```

Description:

ApsaraDB RDS does not report errors for this check item. You can ignore the message that is displayed.

Solution:

N/A.

9.5. Use scenarios on Cloud

9.5.1. Migrate data between ApsaraDB RDS for PostgreSQL instances that reside in different regions

ApsaraDB RDS for PostgreSQL provides the cloud migration feature. You can use the feature to migrate the data of a self-managed PostgreSQL instance that is deployed on an Alibaba Cloud Elastic Compute Service (ECS) instance or in a data center to an ApsaraDB RDS for PostgreSQL instance. You can also use the feature to migrate data between ApsaraDB RDS for PostgreSQL instances that reside in different regions. This topic describes how to use the cloud migration feature to migrate data between ApsaraDB RDS for PostgreSQL instances that reside in different regions.

Prerequisites

The following requirements are met:

- The source RDS instance and the destination RDS instance run the same major engine version. The supported major engine versions are PostgreSQL 10, PostgreSQL11, PostgreSQL12, PostgreSQL13, and PostgreSQL 14.
- The destination RDS instance is a primary instance. Read-only RDS instances do not support cloud migration.
- The destination RDS instance is equipped with standard SSDs or enhance SSDs (ESSDs).
- The destination RDS instance is empty. The available storage of the destination RDS instance is greater than or equal to the size of the data in the source RDS instance.
- The CIDR block of the virtual private cloud (VPC) to which the source RDS belongs is different from the CIDR block of the VPC to which the destination RDS instance belongs.

Precautions

When you migrate data between the RDS instances that reside in different regions, you must use Cloud Enterprise Network (CEN) to enable cross-region communication between the source RDS instance and the destination RDS instance over an internal network. You are charged for cross-region communication that is based on CEN. For more information, see What is CEN? and Billing.

Procedure

> **Note** In this example, the source RDS instance resides in the China (Beijing) region, and the destination RDS instance resides in the China (Hangzhou) region.

1. Create a CEN instance and configure the CEN instance to enable cross-region communication between the source RDS instance and the destination RDS instance over an internal network.

i. Create a CEN instance.

a. Log on to the CEN console.

b. On the **Instances** page, click **Create CEN Instance**.

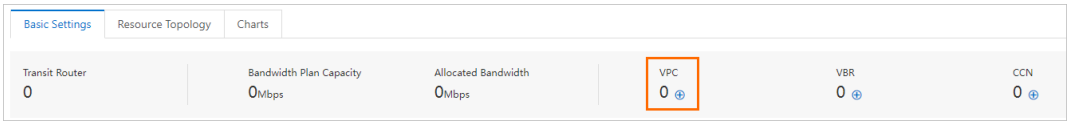
c. In the **Create CEN Instance** dialog box, configure the following parameters and click **OK**.

Parameter	Description
Name	Enter a name for the CEN instance. The name must be 2 to 128 characters in length and can contain letters, digits, hyphens (-), and underscores (_). The name must start with a letter.
Description	Enter a description for the CEN instance. The description must be 2 to 256 characters in length and cannot start with `http://` or `https://`. You can leave this parameter empty.

ii. Add the VPC of the source RDS instance and the VPC of the destination RDS instance to the CEN instance.

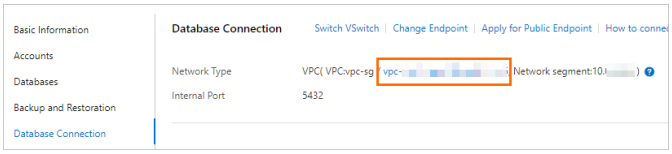
a. On the **Instances** page, click the ID of the CEN instance.

b. On the **Basic Settings** tab of the details page, click the ⊕ icon to the right of the number that is displayed below VPC.

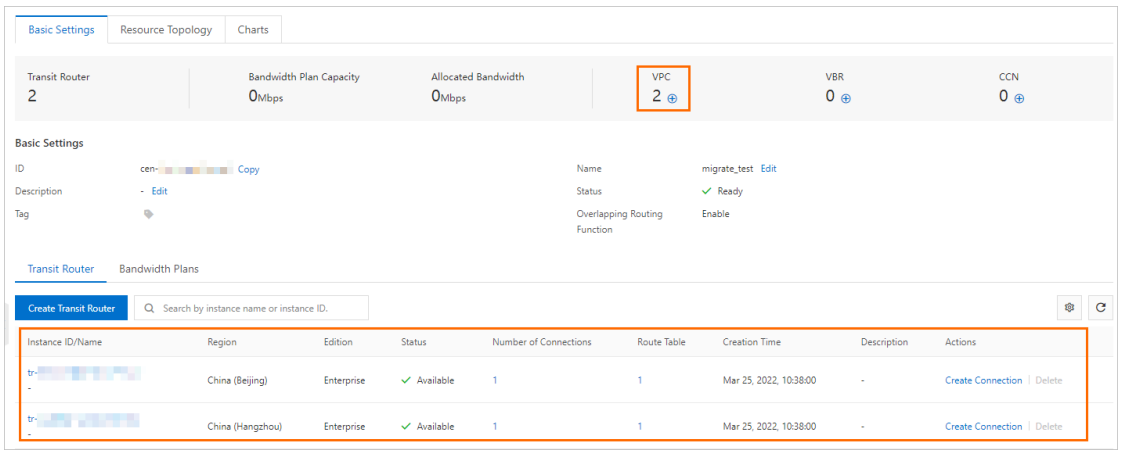


c. On the **Connection with Peer Network Instance** page, configure the following parameters and click **OK**.

> **Note** You must separately add the VPC of the source RDS instance and the VPC of the destination RDS instance to the CEN instance. This example shows how to add the VPC of the source RDS instance to the CEN instance. You can use the same method to add the VPC of the destination RDS instance to the CEN instance. Take note that the values of the **Region** and **Networks** parameters for the source RDS instance and the destination RDS instance are different.

Parameter	Description
Network Type	Retain the default value **VPC**.
Region	Select the region where the source RDS instance resides. In this example, select **China (Beijing)**.
Transit Router	Select the primary zone and secondary zone of the transit router. By default, no transit routers are available in the selected region. CEN automatically creates a transit router.
Resource Owner ID	Select **Your Account**.
Attachment Name	Enter a custom name.
Networks	Select the ID of the VPC to which the source RDS instance belongs. You can view the VPC ID on the Database Connection page of the source RDS instance in the ApsaraDB RDS console.  After you specify the VPC, select the vSwitches that are associated with the primary zone and secondary zone of the transit router.

After you add the VPC of the source RDS instance and the VPC of the destination RDS instance to the CEN instance, the page that is shown in the following figure is displayed.



iii. Purchase a bandwidth plan for the CEN instance.

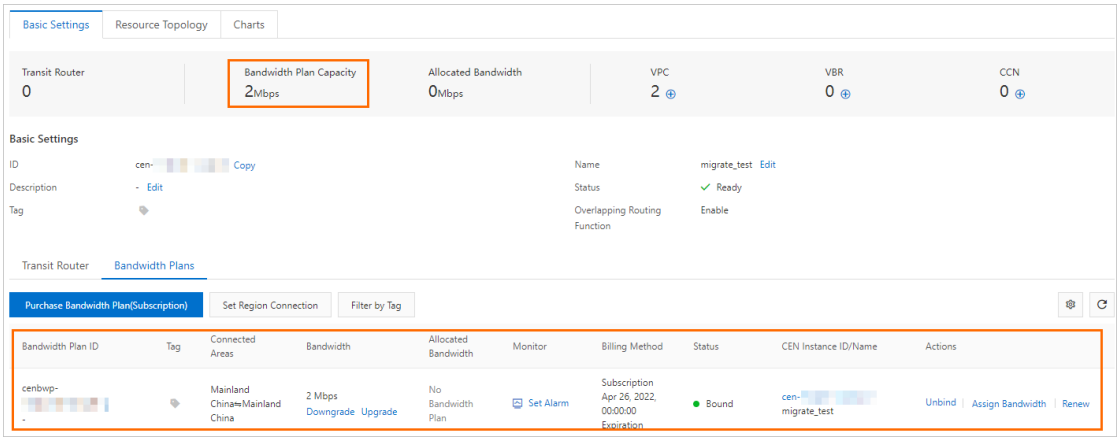
a. On the **Instances** page, click the ID of the CEN instance.

b. On the **Basic Settings** tab of the details page, click the **Bandwidth Plans** tab. On the Bandwidth Plans tab, click **Purchase Bandwidth Plan (Subscription)**.

c. On the page that appears, configure the following parameters and click **Buy Now**. Then, complete the payment.

Parameter	Description
CEN ID	Retain the CEN instance that is displayed. After you purchase a bandwidth plan, the bandwidth plan is automatically associated with the CEN instance.
Area A	Select an area for which you want to enable cross-region communication. In this example, select **Mainland China**. **Note** After you purchase a bandwidth plan, you cannot change the areas that you selected for the bandwidth plan.
Area B	Select the other area for which you want to enable cross-region communication. In this example, select **Mainland China**.
Billing Method	Select a billing method for the bandwidth plan. Default value: **Pay by Bandwidth**.
Bandwidth	Specify the bandwidth that is provided by the bandwidth plan. Unit: Mbit/s.
Bandwidth_package_name	Enter a name for the bandwidth plan.
Order time	Specify a subscription period for the bandwidth plan. You can select **Auto-renewal** to enable auto-renewal for the bandwidth plan.

After you purchase a bandwidth plan, the page that is shown in the following figure is displayed.



iv. Configure the bandwidth for cross-region communication.

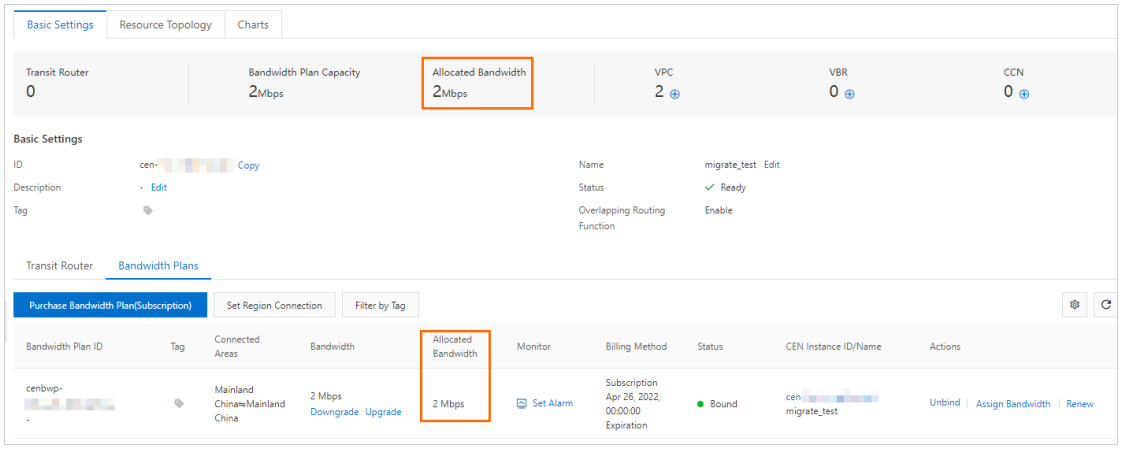
a. On the **Instances** page, click the ID of the CEN instance.

b. On the **Basic Settings** tab of the details page, click the **Bandwidth Plans** tab. On the Bandwidth Plans tab, click **Set Region Connection**.

c. On the **Connection with Peer Network Instance** page, configure the following parameters and click **OK**.

Parameter	Description
Network Type	Select **Cross-region**.
Region	Select the region where the source RDS instance resides. In this example, select **China (Beijing)**.
Transit Router	CEN automatically identifies the transit router in the region where the source RDS instance resides.
Peer Region	Select the region where the destination RDS instance resides. In this example, select **China (Hangzhou)**.
Transit Router	CEN automatically identifies the transit router in the region where the destination RDS instance resides.
Bandwidth Plan	Select the bandwidth plan that is associated with the CEN instance.
Bandwidth	Enter the bandwidth that is provided by the bandwidth plan. Unit: Mbit/s.

After you configure the bandwidth for cross-region communication, the page that is shown in the following figure is displayed.



2. Configure the source RDS instance.

i. Add the CIDR block of the VPC to which the destination RDS instance belongs to an IP address whitelist of the source RDS instance.

For more information see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance. You need to set the **IP Addresses** parameter to the CIDR block of the VPC to which the destination RDS instance belongs.

To view the CIDR block of the VPC to which the destination RDS instance belongs, perform the following operations:

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.

c. In the **Select Migration Source** step of the configuration wizard, select **Self-managed ECS-based PostgreSQL Database or ApsaraDB RDS for PostgreSQL Instance** and click **Next**.

d. In the **Configure Destination Database** step of the configuration wizard, view the value of the **VPC CIDR Block** parameter.

ii. Create a privileged account for the source RDS instance.

For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance. When you create a privileged account, you must set the **Account Type** parameter to Privileged Account.

> **Note** The privileged account is used to migrate data and must have the CREATE ROLE, REPLICATION, and pg_monitor permissions. If you have a privileged account, skip this step.

3. Configure the destination RDS instance.

i. Perform a cloud migration assessment.

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.

c. In the **Select Migration Source** step of the configuration wizard, select **Self-managed ECS-based PostgreSQL Database or ApsaraDB RDS for PostgreSQL Instance** and click **Next**.

d. In the **Configure Destination Database** step of the configuration wizard, click **Next**.

e. In the **Configure Source Database** step of the configuration wizard, select all listed items and click **Next**. Before you start the cloud migration, you must complete the preparations that are described in the listed items.

f. In the **Initiate Migration Assessment** step of the configuration wizard, configure the information about the source RDS instance.

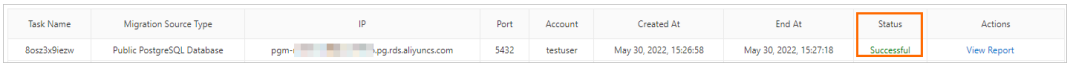
Parameter	Description
Migration Task Name	Enter a name for the cloud migration task. ApsaraDB RDS automatically generates a name for the cloud migration task. You do not need to modify the generated name.
Source VPC/DNS IP	Enter the internal endpoint of the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Source Port	Enter the internal port number of the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Username	Enter the username of the privileged account of the source RDS instance.
Password	Enter the password of the privileged account of the source RDS instance.

g. Click **Create Migration Assessment Task**.

> **Note** During the cloud migration assessment, the status of the destination RDS instance indicates that **the instance is in maintenance**.

After the cloud migration assessment is complete, you can view the status of the cloud migration assessment task on the **Migration Assessment** tab.

- If the status of the cloud migration assessment task indicates a **success**, you can start the cloud migration.
- If the status of the cloud migration assessment task indicates a **failure**, you can click **View Report** in the Actions column to view and handle the reported errors. For more information about common errors, see Introduction to cloud migration assessment reports.



ii. Migrate data to the destination RDS instance.

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration to Cloud** tab. On the Migration to Cloud tab, click Create Cloud Migration Task.

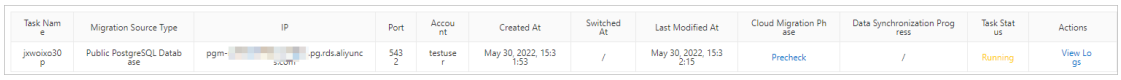
c. In the **Create Cloud Migration Task** dialog box, select the cloud migration assessment task whose status indicates a success from the **Associated Assessment Task** drop-down list.

> **Note** After you select a cloud migration assessment task from the **Associated Assessment Task** drop-down list, ApsaraDB RDS automatically determines the values of the **Migration Source Type**, **Source IP/DNS**, **Source Port**, and **Username** parameters. You do not need to configure these parameters.

d. Click **Initiate Migration to Cloud**. ApsaraDB RDS automatically starts the cloud migration task.

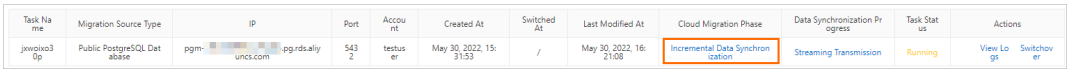
> **Notice** During the cloud migration, the status of the destination RDS instance changes to **Migrating Data In**. You can read data from and write data to the source RDS instance. However, do not migrate data from or to the source RDS instance, restart the source RDS instance, or change the specifications of the source RDS instance.

After you start the cloud migration task, the page that is shown in the following figure is displayed.



iii. Switch the workloads of the source RDS instance over to the destination RDS instance.

a. Click the link in the **Cloud Migration Phase** column to view the progress of the cloud migration task.



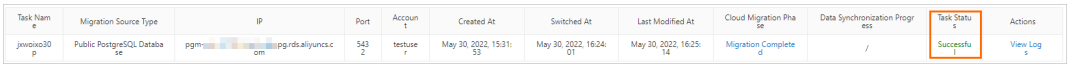
b. If the value in the Cloud Migration Phase column is **Incremental Data Synchronization**, click **Switchover** in the **Actions** column of the cloud migration task to switch the workloads of the source RDS instance over to the destination RDS instance.

c. In the **Switchover** dialog box, configure the source RDS instance to process only read requests. Otherwise, stop the connected application from writing data to the source RDS instance.

> **Note** You can also use the privileged account of the source RDS instance to execute the following statements to configure the source RDS instance to process only read requests:
>
> ```
> ALTER SYSTEM SET default_transaction_read_only=on;
> SELECT pg_reload_conf();
> SELECT pg_terminate_backend(pid) FROM pg_stat_activity
> WHERE usename not in ('replicator', 'monitor', 'pgsql', 'aurora') AND pid != p
> g_backend_pid();
> ```

d. Select all check boxes and click **Switch Now**. Then, wait until the cloud migration is complete.

After the workloads of the source RDS instance are switched over to the destination RDS instance, the page that is shown in the following figure is displayed.



9.5.2. Migrate data between ApsaraDB RDS for PostgreSQL instances within different accounts

ApsaraDB RDS for PostgreSQL provides the cloud migration feature. You can use this feature to migrate the data of a self-managed PostgreSQL instance that is deployed on an Alibaba Cloud Elastic Compute Service (ECS) instance or in a data center to an ApsaraDB RDS for PostgreSQL instance. You can also use this feature to migrate data between ApsaraDB RDS for PostgreSQL instances that are created within different accounts. This topic describes how to use the cloud migration feature to migrate data between ApsaraDB RDS for PostgreSQL instances that are created within different accounts.

Prerequisites

- The following requirements are met:
  - The source RDS instance and the destination RDS instance run the same major engine version. The supported major engine versions are PostgreSQL 10, PostgreSQL11, PostgreSQL12, PostgreSQL13, and PostgreSQL 14.
  - The destination RDS instance is a primary instance. Read-only RDS instances do not support cloud migration.
  - The destination RDS instance is equipped with standard SSDs or enhance SSDs (ESSDs).

- The destination RDS instance is empty. The available storage of the destination RDS instance is greater than or equal to the size of the data in the source RDS instance.
- The CIDR block of the virtual private cloud (VPC) to which the source RDS belongs is different from the CIDR block of the VPC to which the destination RDS instance belongs.

- The permissions on transit routers are granted to the network instances. For more information, see Grant permissions to another Alibaba Cloud account.

Precautions

When you migrate data between ApsaraDB RDS for PostgreSQL instances within different accounts, you must use Cloud Enterprise Network (CEN) to enable cross-region communication between the source RDS instance and the destination RDS instance over an internal network. You are charged for CEN-based cross-region communication. For more information, see What is CEN? and Billing.

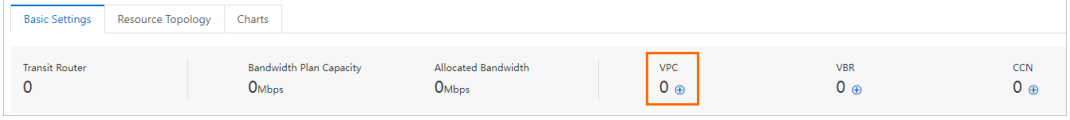
Procedure

> **Note** In this example, the source RDS instance belongs to account A, and the destination RDS instance belongs to account B. The following operations describe how to migrate the data of the source RDS instance to the destination RDS instance. A CEN instance is created under account B.

1. Configure the CEN instance to enable cross-region communication between the source RDS instance and the destination RDS instance over an internal network.
   i. Create a CEN instance.
      a. Log on to the CEN console.
      b. On the **Instances** page, click **Create CEN Instance**.
      c. In the **Create CEN Instance** dialog box, configure the following parameters and click **OK**.

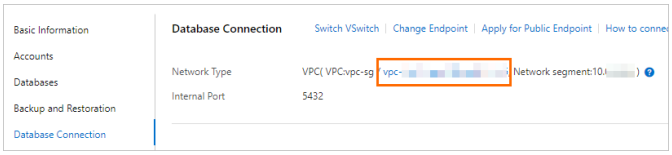
Parameter	Description
Name	Enter a name for the CEN instance. The name must be 2 to 128 characters in length and can contain letters, digits, hyphens (-), and underscores (_). The name must start with a letter.
Description	Enter a description for the CEN instance. The description must be 2 to 256 characters in length and cannot start with `http://` or `https://`. You can leave this parameter empty.

   ii. Add the virtual private cloud (VPC) of the source RDS instance and the VPC of the destination RDS instance to the CEN instance.
      a. On the **Instances** page, click the ID of the CEN instance.
      b. On the **Basic Settings** tab of the details page, click the ⊕ icon to the right of the number that is displayed below VPC.

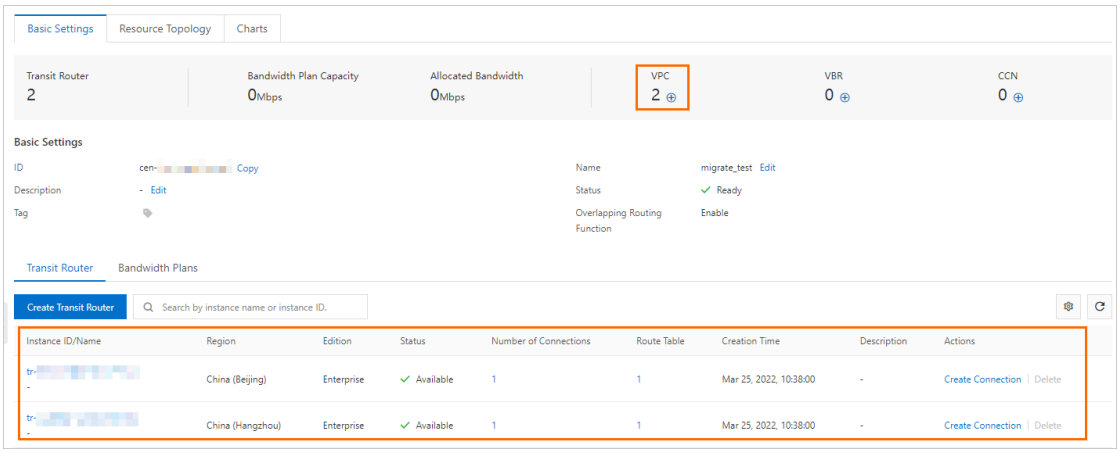


c. On the **Connection with Peer Network Instance** page, configure the following parameters and click **OK**.

> **Note** You must separately add the VPC of the source RDS instance and the VPC of the destination RDS instance to the CEN instance. This example shows how to add the VPC of the source RDS instance to the CEN instance. You can use the same method to add the VPC of the destination RDS instance to the CEN instance. Take note that the values of the **Region** and **Networks** parameters for the source RDS instance and the destination RDS instance are different.

Parameter	Description
Instance Type	Retain the default value **VPC**.
Region	Select the region where the source instance resides.
Transit Router	By default, no transit routers are available in the selected region. CEN automatically creates a transit router. Select the primary zone and secondary zone of the transit router.
Resource Owner ID	Select **Different Account** and enter the UID of the source RDS instance. **Note** Move the pointer over the user picture in the upper-right corner of the page and go to the Basic Information page to check the account ID of the source RDS instance.
Attachment Name	Enter a custom name.
Networks	Select the ID of the VPC to which the source RDS instance belongs. You can view the VPC ID on the Database Connection page of the source RDS instance in the ApsaraDB RDS console.  After you specify the VPC, select the vSwitches that are associated with the primary zone and secondary zone of the transit router.

After you add the VPC of the source RDS instance and the VPC of the destination RDS instance to the CEN instance, the page that is shown in the following figure is displayed.



iii. Optional. Purchase a bandwidth plan for the CEN instance.

> **Note** If the source RDS instance and the destination RDS instance reside in different regions, for example, the source RDS instance resides in China (Beijing) and the destination RDS instance resides in China (Hangzhou), you must perform this step. If the two RDS instances reside in the same region, you can skip this step and go to Step 2.

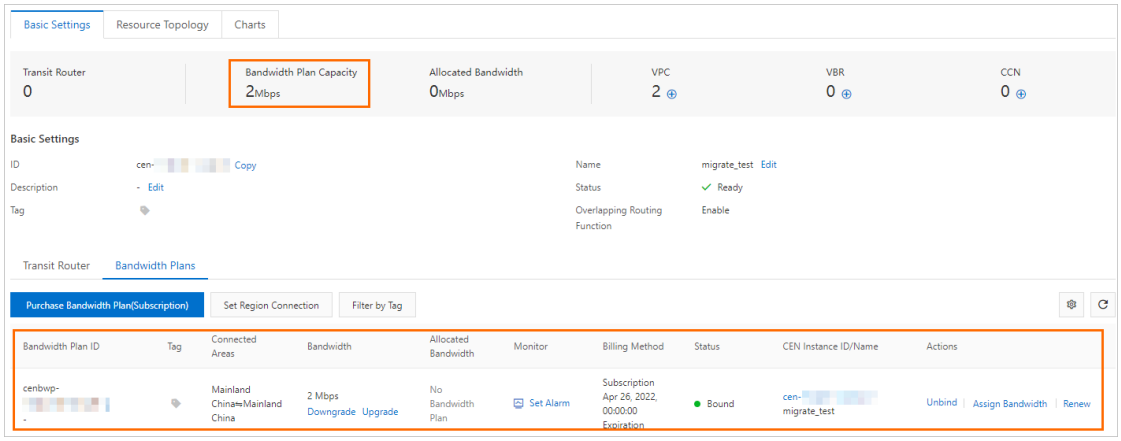
a. On the **Instances** page, click the ID of the CEN instance.

b. On the **Basic Settings** tab of the details page, click the **Bandwidth Plans** tab. On the Bandwidth Plans tab, click **Purchase Bandwidth Plan (Subscription)**.

c. On the page that appears, configure the following parameters and click **Buy Now**. Then, complete the payment.

Parameter	Description
CEN ID	Retain the CEN instance that is displayed. After you purchase a bandwidth plan, the bandwidth plan is automatically associated with the CEN instance.
Area A	Select an area for which you want to enable cross-region communication. In this example, select **Mainland China**. **Note** After you purchase a bandwidth plan, you cannot change the areas that you selected for the bandwidth plan.
Area B	Select the other area for which you want to enable cross-region communication. In this example, select **Mainland China**.
Billing Method	Select a billing method for the bandwidth plan. Default value: **Pay by Bandwidth**.
Bandwidth	Specify the bandwidth that is provided by the bandwidth plan. Unit: Mbit/s.
Bandwidth_package_name	Enter a name for the bandwidth plan.
Order time	Specify a subscription period for the bandwidth plan. You can select **Auto-renewal** to enable auto-renewal for the bandwidth plan.

After you purchase a bandwidth plan, the page that is shown in the following figure is displayed.



iv. Optional. Configure the bandwidth for cross-account communication.

> **Note** If the source RDS instance and the destination RDS instance reside in different regions, for example, the source RDS instance resides in China (Beijing) and the destination RDS instance resides in China (Hangzhou), you must perform this step. If the two RDS instances reside in the same region, you can skip this step and go to Step 2.

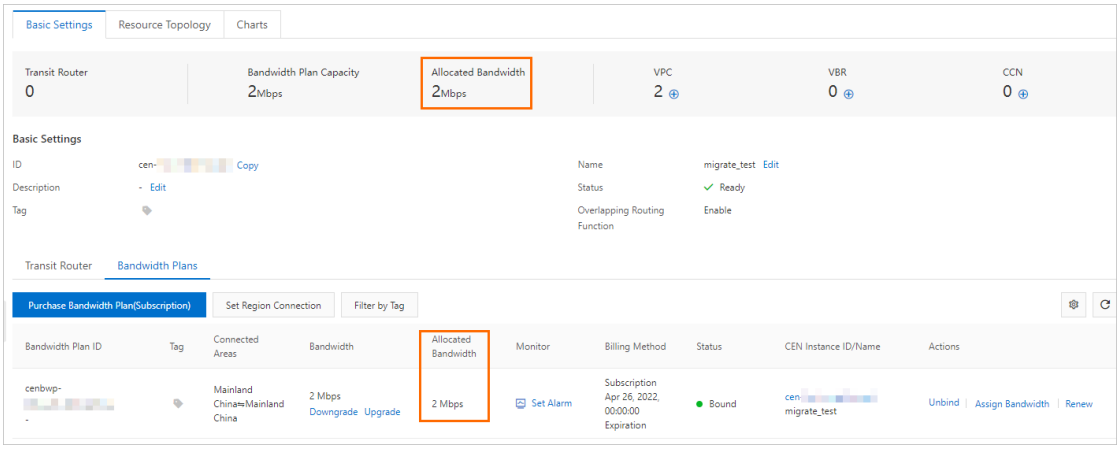
a. On the **Instances** page, click the ID of the CEN instance.

b. On the **Basic Settings** tab of the details page, click the **Bandwidth Plans** tab. On the Bandwidth Plans tab, click **Set Region Connection**.

c. On the **Connection with Peer Network Instance** page, configure the following parameters and click **OK**.

Parameter	Description
Network Type	Select **Cross-region**.
Region	Select the region where the source RDS instance resides. In this example, select **China (Beijing)**.
Transit Router	CEN automatically identifies the transit router in the region where the source RDS instance resides.
Peer Region	Select the region where the destination RDS instance resides. In this example, select **China (Hangzhou)**.
Transit Router	CEN automatically identifies the transit router in the region where the destination RDS instance resides.
Bandwidth Plan	Select the bandwidth plan that is associated with the CEN instance.
Bandwidth	Enter the bandwidth that is provided by the bandwidth plan. Unit: Mbit/s.

After you configure the bandwidth for cross-account communication, the page that is shown in the following figure is displayed.



2. Configure the source RDS instance.

i. Add the CIDR block of the VPC to which the destination RDS instance belongs to an IP address whitelist of the source RDS instance.

For more information see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance. You need to set the **IP Addresses** parameter to the CIDR block of the VPC to which the destination RDS instance belongs.

To view the CIDR block of the VPC to which the destination RDS instance belongs, perform the following operations:

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.

c. In the **Select Migration Source** step of the configuration wizard, select **Self-managed ECS-based PostgreSQL Database or ApsaraDB RDS for PostgreSQL Instance** and click **Next**.

d. In the **Configure Destination Database** step of the configuration wizard, view the value of the **VPC CIDR Block** parameter.

ii. Create a privileged account for the source RDS instance.

For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance. When you create a privileged account, you must set the **Account Type** parameter to Privileged Account.

> **Note** The privileged account is used to migrate data and must have the CREATE ROLE, REPLICATION, and pg_monitor permissions. If you have a privileged account, skip this step.

3. Configure the destination RDS instance.

i. Perform a cloud migration assessment.

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.

c. In the **Select Migration Source** step of the configuration wizard, select **Self-managed ECS-based PostgreSQL Database or ApsaraDB RDS for PostgreSQL Instance** and click **Next**.

d. In the **Configure Destination Database** step of the configuration wizard, click **Next**.

e. In the **Configure Source Database** step of the configuration wizard, select all listed items and click **Next**. Before you start the cloud migration, you must complete the preparations that are described in the listed items.

f. In the **Initiate Migration Assessment** step of the configuration wizard, configure the information about the source RDS instance.

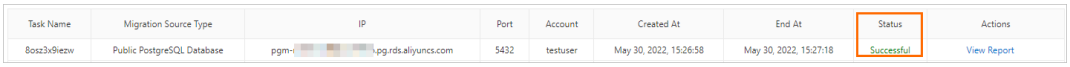
Parameter	Description
Migration Task Name	Enter a name for the cloud migration task. ApsaraDB RDS automatically generates a name for the cloud migration task. You do not need to modify the generated name.
Source VPC/DNS IP	Enter the internal endpoint of the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Source Port	Enter the internal port number of the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Username	Enter the username of the privileged account of the source RDS instance.
Password	Enter the password of the privileged account of the source RDS instance.

g. Click **Create Migration Assessment Task**.

> **Note** During the cloud migration assessment, the status of the destination RDS instance indicates that **the instance is in maintenance**.

After the cloud migration assessment is complete, you can view the status of the cloud migration assessment task on the **Migration Assessment** tab.

- If the status of the cloud migration assessment task indicates a **success**, you can start the cloud migration.
- If the status of the cloud migration assessment task indicates a **failure**, you can click **View Report** in the Actions column to view and handle the reported errors. For more information about common errors, see Introduction to cloud migration assessment reports.



ii. Migrate data to the destination RDS instance.

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration to Cloud** tab. On the Migration to Cloud tab, click Create Cloud Migration Task.

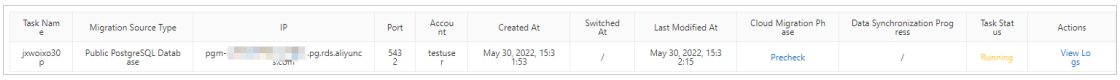
c. In the **Create Cloud Migration Task** dialog box, select the cloud migration assessment task whose status indicates a success from the **Associated Assessment Task** drop-down list.

> **Note** After you select a cloud migration assessment task from the **Associated Assessment Task** drop-down list, ApsaraDB RDS automatically determines the values of the **Migration Source Type**, **Source IP/DNS**, **Source Port**, and **Username** parameters. You do not need to configure these parameters.

d. Click **Initiate Migration to Cloud**. ApsaraDB RDS automatically starts the cloud migration task.

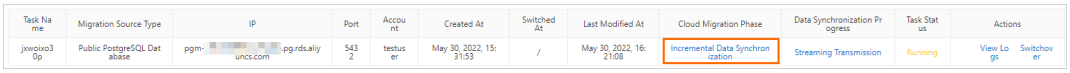
> **Notice** During the cloud migration, the status of the destination RDS instance changes to **Migrating Data In**. You can read data from and write data to the source RDS instance. However, do not migrate data from or to the source RDS instance, restart the source RDS instance, or change the specifications of the source RDS instance.

After you start the cloud migration task, the page that is shown in the following figure is displayed.



iii. Switch the workloads of the source RDS instance over to the destination RDS instance.

a. Click the link in the **Cloud Migration Phase** column to view the progress of the cloud migration task.



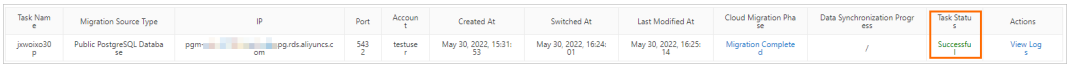
b. If the value in the Cloud Migration Phase column is **Incremental Data Synchronization**, click **Switchover** in the **Actions** column of the cloud migration task to switch the workloads of the source RDS instance over to the destination RDS instance.

c. In the **Switchover** dialog box, configure the source RDS instance to process only read requests. Otherwise, stop the connected application from writing data to the source RDS instance.

> **Note** You can also use the privileged account of the source RDS instance to execute the following statements to configure the source RDS instance to process only read requests:
>
> ```
> ALTER SYSTEM SET default_transaction_read_only=on;
> SELECT pg_reload_conf();
> SELECT pg_terminate_backend(pid) FROM pg_stat_activity
> WHERE usename not in ('replicator', 'monitor', 'pgsql', 'aurora') AND pid != p
> g_backend_pid();
> ```

d. Select all check boxes and click **Switch Now**. Then, wait until the cloud migration is complete.

After the workloads of the source RDS instance are switched over to the destination RDS instance, the page that is shown in the following figure is displayed.



9.5.3. Scale down an ApsaraDB RDS for PostgreSQL instance

ApsaraDB RDS for PostgreSQL provides the cloud migration feature. You can use the feature to migrate the data of a self-managed PostgreSQL instance that is deployed on an Alibaba Cloud Elastic Compute Service (ECS) instance or in a data center to an ApsaraDB RDS for PostgreSQL instance. You can also use the feature to reduce the storage capacity of an ApsaraDB RDS for PostgreSQL instance. This topic describes how to use the cloud migration feature to scale down an ApsaraDB RDS for PostgreSQL instance.

Context

When you change the specifications of your RDS instance, you cannot reduce the storage capacity of the RDS instance. You can use the cloud migration feature to migrate the data of the original RDS instance to a new RDS instance whose storage capacity is less than the storage capacity of the original RDS instance based on your business requirements. This way, you can reduce the storage capacity that you purchased.

The cloud migration feature has the following benefits:

- Data is migrated by using physical streaming replication in an easy-to-use and efficient manner and at a high speed with minimum downtime.

- Data is migrated over an internal network, which is secure and free of charge.
- Cloud migration has a few limits. Specifically, the destination RDS instance is a primary instance that uses standard SSDs or enhanced SSDs (ESSDs) and runs the same major engine version as the source RDS instance.
- The destination RDS instance can provide the same performance and statistics as the source RDS instance.

Prerequisites

The destination RDS instance is created and meets the following conditions:

- The source RDS instance and the destination RDS instance run the same major engine version. The supported major engine versions are PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, and PostgreSQL 14.
- The destination RDS instance is a primary instance. Read-only RDS instances do not support cloud migration.
- The destination RDS instance is equipped with standard SSDs or ESSDs.
- The destination RDS instance is empty.
- The storage capacity of the destination RDS instance is planned before cloud migration. We recommend that you reserve more than 20% of the storage capacity.

  For example, if the occupied storage of the source RDS instance is 200 GB, the storage capacity of the destination RDS instance must be at least 250 GB based on the following formula:

  ```
  200/(1 - 20%) = 250 GB
  ```

Procedure

> **Note** This topic describes how to migrate data between two ApsaraDB RDS for PostgreSQL instances that reside in the same virtual private cloud (VPC) in the China (Hangzhou) region. The source RDS instance has a storage capacity of 500 GB, and 200 GB of storage is used. The destination RDS instance has a storage capacity of 250 GB.

1. Configure the source RDS instance.
   i. Add the CIDR block of the VPC to which the destination RDS instance belongs to an IP address whitelist of the source RDS instance.

      For more information see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance. You need to set the **IP Addresses** parameter to the CIDR block of the VPC to which the destination RDS instance belongs.

      To view the CIDR block of the VPC to which the destination RDS instance belongs, perform the following operations:

      a.
      b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.
      c. In the **Select Migration Source** step of the configuration wizard, select **Self-managed ECS-based PostgreSQL Database or ApsaraDB RDS for PostgreSQL Instance** and click **Next**.
      d. In the **Configure Destination Database** step of the configuration wizard, view the value of the **VPC CIDR Block** parameter.

ii. Create a privileged account for the source RDS instance.

For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance. When you create a privileged account, you must set the **Account Type** parameter to Privileged Account.

> **Note** The privileged account is used to migrate data and must have the CREATE ROLE, REPLICATION, and pg_monitor permissions. If you have a privileged account, skip this step.

2. Configure the destination RDS instance.

i. Perform a cloud migration assessment.

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.

c. In the **Select Migration Source** step of the configuration wizard, select **Self-managed ECS-based PostgreSQL Database or ApsaraDB RDS for PostgreSQL Instance** and click **Next**.

d. In the **Configure Destination Database** step of the configuration wizard, click **Next**.

e. In the **Configure Source Database** step of the configuration wizard, select all listed items and click **Next**. Before you start the cloud migration, you must complete the preparations that are described in the listed items.

f. In the **Initiate Migration Assessment** step of the configuration wizard, configure the information about the source RDS instance.

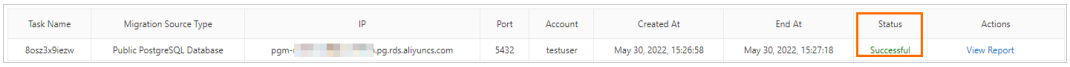
Parameter	Description
Migration Task Name	Enter a name for the cloud migration task. ApsaraDB RDS automatically generates a name for the cloud migration task. You do not need to modify the generated name.
Source VPC/DNS IP	Enter the internal endpoint of the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Source Port	Enter the internal port number of the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Username	Enter the username of the privileged account of the source RDS instance.
Password	Enter the password of the privileged account of the source RDS instance.

g. Click **Create Migration Assessment Task**.

> **Note** During the cloud migration assessment, the status of the destination RDS instance indicates that **the instance is in maintenance**.

After the cloud migration assessment is complete, you can view the status of the cloud migration assessment task on the **Migration Assessment** tab.

- If the status of the cloud migration assessment task indicates a **success**, you can start the cloud migration.
- If the status of the cloud migration assessment task indicates a **failure**, you can click **View Report** in the Actions column to view and handle the reported errors. For more information about common errors, see Introduction to cloud migration assessment reports.



ii. Migrate data to the destination RDS instance.

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration to Cloud** tab. On the Migration to Cloud tab, click Create Cloud Migration Task.

c. In the **Create Cloud Migration Task** dialog box, select the cloud migration assessment task whose status indicates a success from the **Associated Assessment Task** drop-down list.

> **Note** After you select a cloud migration assessment task from the **Associated Assessment Task** drop-down list, ApsaraDB RDS automatically determines the values of the **Migration Source Type**, **Source IP/DNS**, **Source Port**, and **Username** parameters. You do not need to configure these parameters.

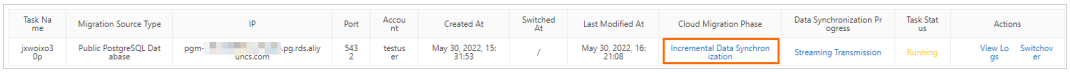
d. Click **Initiate Migration to Cloud**. ApsaraDB RDS automatically starts the cloud migration task.

> **Notice** During the cloud migration, the status of the destination RDS instance changes to **Migrating Data In**. You can read data from and write data to the source RDS instance. However, do not migrate data from or to the source RDS instance, restart the source RDS instance, or change the specifications of the source RDS instance.

After you start the cloud migration task, the page that is shown in the following figure is displayed.

iii. Switch the workloads of the source RDS instance over to the destination RDS instance.

a. Click the link in the **Cloud Migration Phase** column to view the progress of the cloud migration task.



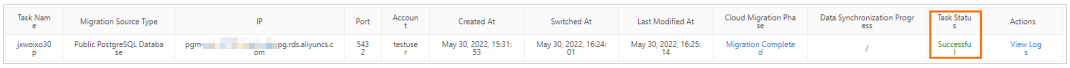
b. If the value in the Cloud Migration Phase column is **Incremental Data Synchronization**, click **Switchover** in the **Actions** column of the cloud migration task to switch the workloads of the source RDS instance over to the destination RDS instance.

c. In the **Switchover** dialog box, configure the source RDS instance to process only read requests. Otherwise, stop the connected application from writing data to the source RDS instance.

> **Note** You can also use the privileged account of the source RDS instance to execute the following statements to configure the source RDS instance to process only read requests:
>
> ```
> ALTER SYSTEM SET default_transaction_read_only=on;
> SELECT pg_reload_conf();
> SELECT pg_terminate_backend(pid) FROM pg_stat_activity
> WHERE usename not in ('replicator', 'monitor', 'pgsql', 'aurora') AND pid != p
> g_backend_pid();
> ```

d. Select all check boxes and click **Switch Now**. Then, wait until the cloud migration is complete.

After the workloads of the source RDS instance are switched over to the destination RDS instance, the page that is shown in the following figure is displayed.



What to do next

To connect your applications to the destination RDS instance without the need to modify business code after the scale-down, perform the following steps on the destination RDS instance:

1. Modify the whitelist settings of the destination RDS instance to ensure that the whitelists of the destination RDS instance are the same as the whitelists of the source RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
2. Change the endpoints that are used to connect to the destination RDS instance to ensure that these endpoints are the same as the endpoints that are used to connect to the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.

   For example, if the endpoint that is used to connect to the source RDS instance is `pgm-aaa.pg.rds.aliyuncs.com` and the endpoint that is used to connect to the destination RDS instance is `pgm-bbb.pg.rds.aliyuncs.com`, perform the following steps to change the endpoints:

   i. Change the endpoint that is used to connect to the source RDS instance from `pgm-aaa.pg.rds.aliyuncs.com` to `pgm-ccc.pg.rds.aliyuncs.com`.

   ii. Change the endpoint that is used to connect to the destination RDS instance from `pgm-bbb.pg.rds.aliyuncs.com` to `pgm-aaa.pg.rds.aliyuncs.com`.

10.Data Migration

10.1. Overview of data migration methods

This topic describes the methods that you can use to migrate data among self-managed data centers, third-party clouds, and ApsaraDB RDS with no downtime.

Scenario	References
Migrate data from a PostgreSQL database in a self-managed data center to an ApsaraDB RDS for PostgreSQL instance	• Manually migrate data from a user-created PostgreSQL database hosted on ECS to an ApsaraDB RDS for PostgreSQL database • Migrate data from a self-managed PostgreSQL database to an ApsaraDB RDS for PostgreSQL instance
Migrate data from a PostgreSQL database on a third-party cloud to an ApsaraDB RDS for PostgreSQL instance	• Migrate incremental data from an Amazon RDS for PostgreSQL instance to an ApsaraDB RDS for PostgreSQL instance • Migrate full data from an Amazon RDS for PostgreSQL instance to an ApsaraDB RDS for PostgreSQL instance
Migrate data between ApsaraDB RDS for PostgreSQL instances	• Use DTS to migrate data between ApsaraDB RDS for PostgreSQL instances • Use the cloud migration feature to migrate data between ApsaraDB RDS for PostgreSQL instances
Migrate data from an ApsaraDB RDS for PostgreSQL instance to an on-premises PostgreSQL database	• Use DTS to migrate data between ApsaraDB RDS for PostgreSQL instances • Use the cloud migration feature to migrate data between ApsaraDB RDS for PostgreSQL instances

10.2. Migrate user-created databases to ApsaraDB

10.2.1. Manually migrate data from a user-created PostgreSQL database hosted on ECS to an ApsaraDB RDS for PostgreSQL database

This topic describes how to migrate data from a source user-created PostgreSQL database to a destination database on your ApsaraDB RDS for PostgreSQL instance. The source user-created database is hosted on an Elastic Compute Service (ECS) instance.

Prerequisites

An ECS instance that runs a Linux operating system is created. In addition, the ECS instance can connect to your RDS instance.

Procedure

1. Install the PostgreSQL Community edition on the ECS instance.

   > **Note** The pg_dump plug-in must run the same PostgreSQL version as the source user-created database. The pg_restore plug-in must run the same PostgreSQL version as your RDS instance. In this topic, PostgreSQL 12 is used as an example. You can select a different PostgreSQL version based on your business requirements.

   ```
   --Install the RPM Package Manager (RPM).
   #yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
   --Install the PostgreSQL Community edition.
   #yum install -y postgresql12-*
   ```

2. Configure the required environment variables. This allows you to make sure that a suitable PostgreSQL version is used for the data export and import.

   ```
   #su - postgres
   #vi .bash_profile
   --Append the required content.
   export PS1="$USER@`/bin/hostname -s`-> "
   export LANG=en_US.utf8
   export PGHOME=/usr/pgsql-12
   export LD_LIBRARY_PATH=$PGHOME/lib:/lib64:/usr/lib64:/usr/local/lib64:/lib:/usr/lib:/usr/local/lib:$LD_LIBRARY_PATH
   export DATE=`date +"%Y%m%d%H%M"`
   export PATH=$PGHOME/bin:$PATH:.
   export MANPATH=$PGHOME/share/man:$MANPATH
   alias rm='rm -i'
   alias ll='ls -lh'
   unalias vi
   ```

3. Export all of the user data from the source user-created database.

   > **Note** The migration of the user data must be performed before the migration of the other data. Otherwise, the migration may fail due to issues related to object permissions or owners.

```
#pg_dumpall -g -h 127.0.0.1 -p 5432 -U postgres
--
-- PostgreSQL database cluster dump
--
SET default_transaction_read_only = off;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
--
-- Roles
--
CREATE ROLE postgres;
ALTER ROLE postgres WITH SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION BYPASSRL
S PASSWORD 'md5d5df0dxxxxxxxc88a541fec598f';
--
-- PostgreSQL database cluster dump complete
--
```

4. Modify the settings of the roles-related commands from the preceding step. Then, run the modified commands to import the user data from the source user-created database into your RDS instance. For example, you must replace SUPERUSER in the preceding step with rds_SUPERUSER in this step.

```
#CREATE ROLE postgres;
#ALTER ROLE postgres WITH rds_SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION BYP
ASSRLS PASSWORD 'md5d5df0dxxxxxxxc88a541fec598f';
```

```
1 CREATE ROLE postgres;
2 ALTER ROLE postgres WITH rds_SUPERUSER INHERIT CREATEROLE CREATEDB LOGIN REPLICATION BYPASSRLS PASSWORD 'md5d5df0d43cc619177e8c88a541fec598f';
```

5. Connect to your RDS instance and create the destination database.

> **Note** The destination database must use the same encoding format as the source user-created database. The UTF8 encoding format is used in this example.

```
#create database db1 with template template0 encoding 'UTF8' lc_ctype 'en_US.utf8' lc_col
late 'C';
```

6. Optional. If the preceding method is unsuitable for your workloads, use one of the following three methods to export and import data:
   - Method 1: online migration

     This method is used if the source user-created database can connect to your RDS instance.

a. Configure a password file. The passwords in the password file are in the following format: `hostname:port:dbname:username:password`.

> **Note** The values of the dbname and username parameters must be in lower case. This does not apply if the database name and username that you specified each are in upper case and are included in a pair of double quotation Marks ("). The metadata of a PostgreSQL database is stored in lower case by default.

```
#vi ~/.pgpass
pgm-xxx.pg.rds.aliyuncs.com:1921:db1:Username:Password
127.0.0.1:5432:postgres:Username:Password
#chmod 400 ~/.pgpass
```

b. Migrate data by using a pipeline.

```
#nohup pg_dump -F p -h 127.0.0.1 -p 5432 -U postgres -d postgres --no-tablespaces |
time psql -h pgm-bpxxxxx.pg.rds.aliyuncs.com -p 1921 -U postgres --single-transacti
on db1 > ./pg.dump.log 2>&1 &
```

> **Note** If errors occur, you can view the error logs in the pg.dump.log file. After you fix the errors, you can immediately resume the data import. This applies if you have configured the `--single-transaction` option.

- Method 2: offline migration

This method is used if the user-created source database cannot connect to your RDS instance. In this case, you can use the pg_dump plug-in to export data from the source user-created database as files. Then, after you copy the files to a host that can connect to your RDS instance, you can use the pg_restore plug-in to import the files into your RDS instance.

> **Note** The pg_dump plug-in that is used to export data must run the same PostgreSQL version as the source user-created database. The pg_restore plug-in that is used to import data must run the same PostgreSQL version as your RDS instance.

a. Export data from the source user-created database as files.

```
#nohup pg_dump -F c -h 127.0.0.1 -p 5432 -U postgres -d postgres --no-tablespaces -
f ./pg.dump > ./pg.dump.log 2>&1 &
#ll pg.dump
-rw-rw-r-- 1 digoal digoal 4.2M Aug 31 10:17 pg.dump
```

> **Note** Wait until the export is complete. Then, check that no errors are found in the pg.dump.log file.

b. Import the files into your RDS instance.

```
#pg_restore -h pgm-bpxxxxx.pg.rds.aliyuncs.com -p 1921 -U postgres -d db1 --no-tablespaces --single-transaction  pg1.dump  >./pg1.restore.log
```

> **Note** If you have specified an MD5-encrypted password for the specified user, errors may occur. We recommend that you reset the password in the ApsaraDB for RDS console. For more information, see Reset the password of an account on an ApsaraDB RDS for PostgreSQL instance.

Wait until the import is complete. If errors occur, you can view the error logs in the pg.dump.log file. After you fix the errors, you can immediately resume the data import. This applies if you have configured the `--single-transaction` option.

- Method 3: parallelism-based accelerated offline migration

  This method is similar to Method 2. However, this method allows you to specify the option that is used to enable parallelism when you run the pg_restore plug-in to import data.

```
nohup pg_restore -U postgres -d db1 --no-tablespaces -j 4 /tmp/pg.dump  >./pg.restore.log 2>&1 &
```

> **Note** The `-j` parameter that is used to enable parallelism cannot be used at the same time as the `--single-transaction` parameter.

10.2.2. Migrate data from a self-managed PostgreSQL database to an ApsaraDB RDS for PostgreSQL instance

This topic describes how to migrate data from a self-managed PostgreSQL database to an ApsaraDB RDS for PostgreSQL instance by using Data Transmission Service (DTS). DTS supports schema migration, full data migration, and incremental data migration. When you migrate data from a self-managed PostgreSQL database to Alibaba Cloud, you can select all of the supported migration types to ensure service continuity.

Prerequisites

- A self-managed PostgreSQL database and an instance are created. For information about how to create the destination instance, see Create an ApsaraDB RDS for PostgreSQL instance. For information about the supported versions, see Overview of data migration scenarios.

  > **Note**

- The available storage space of the destination instance is larger than the total size of the data in the self-managed PostgreSQL database.

Limits

Category	Description
Limits on the source database	• The server to which the source database belongs must have sufficient outbound bandwidth. Otherwise, the data migration speed decreases. • The tables to migrate must have PRIMARY KEY or UNIQUE constraints and all fields must be unique. Otherwise, the destination database may contain duplicate data records. The name of the source database cannot contain hyphens (-). Example: dts-testdata. • If you select tables as objects to migrate and you need to edit tables (such as renaming tables or columns) in the destination database, up to 1,000 tables can be migrated in a single data migration task. If you run a task to migrate more than 1,000 tables, a request error occurs. In this case, we recommend that you split the tables and configure multiple tasks to migrate the tables, or configure a task to migrate the entire database. • If you need to migrate incremental data, you must make sure that the following requirements are met: ◦ The value of the wal_level parameter must be set to *logical*. ◦ For an incremental data migration, the WAL logs of the source database must be stored for more than 24 hours. For a full data and incremental data migration, the WAL logs of the source database must be stored for at least seven days. After full data migration is complete, you can set the retention period to more than 24 hours. Otherwise, DTS may fail to obtain the WAL logs and the task may fail. In exceptional circumstances, data inconsistency or loss may occur. Make sure that you set the retention period of WAL logs in accordance with the preceding requirements. Otherwise, the Service Level Agreement (SLA) of DTS does not ensure service reliability or performance. • Limits on operations: ◦ If you perform a primary/secondary switchover on the self-managed PostgreSQL database, the data migration task fails. ◦ During schema migration and full data migration, do not perform DDL operations to change the schemas of databases or tables. Otherwise, the data migration task fails.
	• Data may be inconsistent between the primary and secondary nodes of the source database due to migration latency. Therefore, you must use the primary node as the data source when you migrate data. • A data migration task can migrate data from only a single database. To migrate data from multiple databases, you must create a data migration task for each database. • During incremental data migration, if you select a schema as the object to migrate, take note of the following limits: If you create a table in the schema or execute the RENAME statement to rename the table, you must execute the `ALTER TABLE schema.table REPLICA IDENTITY FULL;` statement before you write data to the table. **Note** Replace the `schema` and `table` in the preceding sample statement with your actual schema name and table name. • DTS does not check the validity of metadata such as sequences. You must manually check the validity of metadata. • After your workloads are switched to the destination database, newly written sequences do not increment from the maximum value of the sequences in the source

Category	Description
Other limits	database. Therefore, you must query the maximum value of the sequences in the source database before you switch your workloads to the destination database. Then, you must specify the queried maximum value as the starting value of the sequences in the destination database. You can execute the following statements to query the maximum value of the sequences in the source database: `do language plpgsql $$ declare nsp name; rel name; val int8; begin for nsp,rel in select nspname,relname from pg_class t2 , pg_namespace t3 where t2.relnamespace=t3.oid and t2.relkind='S' loop execute format($_$select last_value from %I.%I$_$, nsp, rel) into val; raise notice '%', format($_$select setval('%I.%I'::regclass, %s);$_$, nsp, rel, val+1); end loop; end; $$;` • DTS creates the following temporary tables in the source database to obtain the DDL statements of incremental data, the schemas of incremental tables, and the heartbeat information. During data migration, do not delete temporary tables in the source database. Otherwise, the data migration task may fail. After the DTS instance is released, temporary tables are automatically deleted. `public.DTS_PG_CLASS` , `public.DTS_PG_ATTRIBUTE` , `public.DTS_PG_TYPE` , `public.DTS_PG_ENUM` , `public.DTS_POSTGRES_HEARTBEAT` , `public.DTS_DDL_COMMAND` , and `public.DTS_ARGS_SESSION` . • To ensure that the latency of incremental data migration is accurate, DTS creates a heartbeat table named `dts_postgres_heartbeat` in the source database. • During incremental data migration, DTS creates a replication slot for the source database. The replication slot is prefixed with `dts_sync_` . DTS automatically clears historical replication slots every 90 minutes to reduce storage usage. **Note** If the data migration task is released or fails, DTS automatically clears the replication slot. If a primary/secondary switchover is performed on the source ApsaraDB RDS for PostgreSQL instance, you must log on to the secondary instance to clear the replication slot.  • Before you migrate data, evaluate the impact of data migration on the performance of the source and destination databases. We recommend that you migrate data during off-peak hours. During full data migration, DTS uses read and write resources of the source and destination databases. This may increase the loads of the database servers.

Category	Description
	• During full data migration, concurrent INSERT operations cause fragmentation in the tables of the destination database. After full data migration is complete, the size of tablespace used by the destination database is larger than that of the source database. • You must make sure that the precision settings for columns of the FLOAT or DOUBLE data type meets your business requirements. DTS uses the `ROUND(COLUMN,PRECISION)` function to retrieve values from columns of the FLOAT or DOUBLE data type. If you do not specify a precision, DTS sets the precision for the FLOAT data type to 38 digits and the precision for the DOUBLE data type to 308 digits. • DTS attempts to resume data migration tasks that failed within the last seven days. Before you switch workloads to the destination instance, stop or release the data migration task. You can also execute the `REVOKE` statement to revoke the write permissions from the accounts used by DTS to access the destination instance. Otherwise, the data in the source database overwrites the data in the destination instance after the task is resumed.

Migration types

- Schema migration

  DTS migrates the schemas of required objects from the source database to the destination database.

- Full data migration

  DTS migrates historical data of required objects from the source database to the destination database.

- Incremental data migration

  After full data migration is completed, DTS migrates incremental data from the source database to the destination database. Incremental data migration ensures service continuity when you migrate data between self-managed databases.

SQL operations that can be migrated

Operation type	SQL statements
DML	INSERT, UPDATE, and DELETE

Permissions required for database accounts

Database	Schema migration	Full data migration	Incremental data migration
Self-managed PostgreSQL database	The USAGE permission on pg_catalog	The SELECT permission on the objects to be migrated	The permissions of the superuser role
instance	The CREATE and USAGE permissions on the objects to be migrated	The permissions of the schema owner	The permissions of the schema owner

For information about how to create and authorize a database account, see the following topics:

- Self-managed PostgreSQL database: CREATE USER and GRANT
- instance: Create an account on an ApsaraDB RDS for PostgreSQL instance

Before you begin

If the version of the self-managed PostgreSQL database is 10.1 to 13, you must perform the following operations before you configure a data migration task.

1. Log on to the server where the self-managed PostgreSQL database resides.
2. Set the value of the `wal_level` parameter in the *postgresql.conf* configuration file to `logical`.

```
# - Settings -

wal_level = logical                     # minimal, replica, or logical
                                        # (change requires restart)
```

3. Add the CIDR blocks of DTS servers to the *pg_hba.conf* configuration file of the self-managed PostgreSQL database. Add only the CIDR blocks of the DTS servers that reside in the same region as the destination database. For more information, see Add the CIDR blocks of DTS servers to the security settings of on-premises databases.

> **Note** For more information, see The pg_hba.conf File. Skip this step if you have set the IP address in the pg_hba.conf file to `0.0.0.0/0`.

```
# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             0.0.0.0/0               trust
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

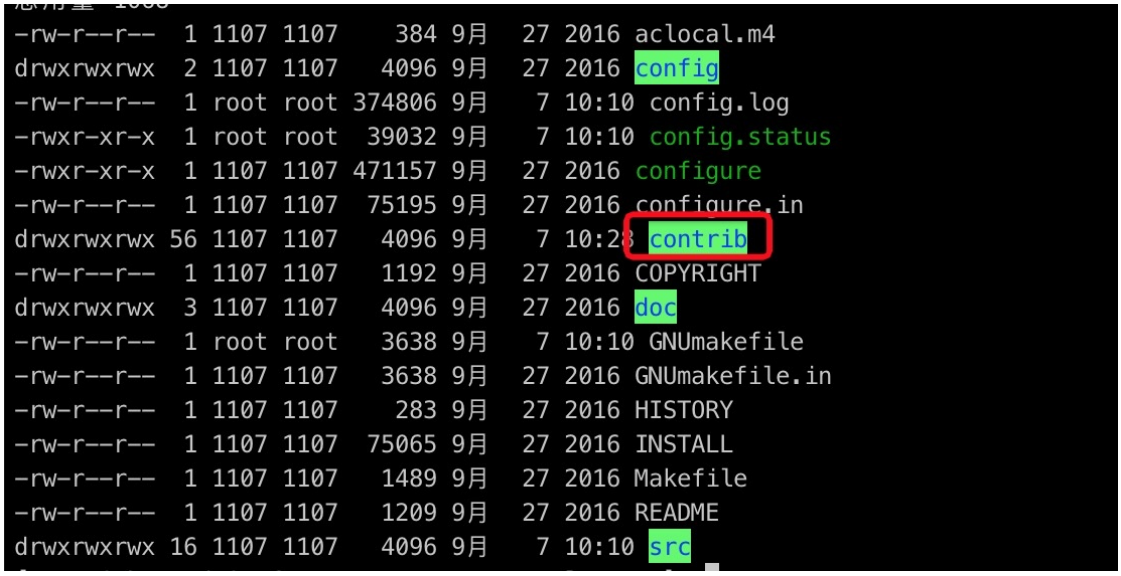
4. Create a database and schema in the destination instance based on the database and schema information of the objects to be migrated. The schema name of the source and destination databases must be the same. For more information, see Create a database on an ApsaraDB RDS for PostgreSQL instance and Appendix: User and schema management.

If the version of the self-managed PostgreSQL database is 9.4.8 to 10.0, you must perform the following operations before you configure a data migration task.

1. Download the PostgreSQL source code from the official website, and compile and install the source code.
   i. Download the source code from the PostgreSQL official website based on the version of the self-managed PostgreSQL database.
   ii. Run the `./configure`, `make`, and `make install` commands to in sequence to configure, compile, and install the source code.

> **Notice**
> - When you compile and install PostgreSQL, the operating system version of PostgreSQL must be consistent with the GNU Compiler Collection (GCC) version.
> - If an error occurs when you run the `./configure` command, you can adjust the command based on the error message. For example, if the error message is `readline library not found. Use --without-readline to disable readline support.`, you can change the command to `./configure --without-readline`.
> - If you use other methods to install PostgreSQL, you must compile the ali_decoding plug-in in a test environment that has the same OS version and GCC version.

2. Download the ali_decoding plug-in provided by DTS, and compile and install the plug-in.
   i. Download ali_decoding.
   ii. Copy the ali_decoding directory to the contrib directory of PostgreSQL (compiled and installed).



   iii. Go to the ali_decoding directory and replace the content of the Makefile file with the following script:

```
# contrib/ali_decoding/Makefile
MODULE_big = ali_decoding
MODULES = ali_decoding
OBJS    = ali_decoding.o
DATA = ali_decoding--0.0.1.sql ali_decoding--unpackaged--0.0.1.sql
EXTENSION = ali_decoding
NAME = ali_decoding
#subdir = contrib/ali_decoding
#top_builddir = ../..
#include $(top_builddir)/src/Makefile.global
#include $(top_srcdir)/contrib/contrib-global.mk
#PG_CONFIG = /usr/pgsql-9.6/bin/pg_config
#pgsql_lib_dir := $(shell $(PG_CONFIG) --libdir)
#PGXS := $(shell $(PG_CONFIG) --pgxs)
#include $(PGXS)
# Run the following commands to install the source code.
ifdef USE_PGXS
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
else
subdir = contrib/ali_decoding
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/contrib/contrib-global.mk
endif
```

   iv. Go to the ali_decoding directory, run the `make` and `make install` commands in sequence to compile ali_decoding and obtain the files required to install ali_decoding.

v. Copy the following files to the specified location.

```
/usr/bin/install -c -m 755  ali_decoding.so '/usr/local/pgsql/lib/ali_decoding.so'
/usr/bin/install -c -m 644 ./ali_decoding.control '/usr/local/pgsql/share/extension/'
/usr/bin/install -c -m 644 ./ali_decoding--0.0.1.sql ./ali_decoding--unpackaged--0.0.1.sql  '/usr/local/pgsql/share/extension/'
/usr/bin/install -c -m 755  ali_decoding.so '/usr/local/pgsql/lib/'
```

3. Create a database and schema in the destination instance based on the database and schema information of the objects to be migrated. The schema name of the source and destination databases must be the same. For more information, see Create a database on an ApsaraDB RDS for PostgreSQL instance and Appendix: User and schema management.

Procedure

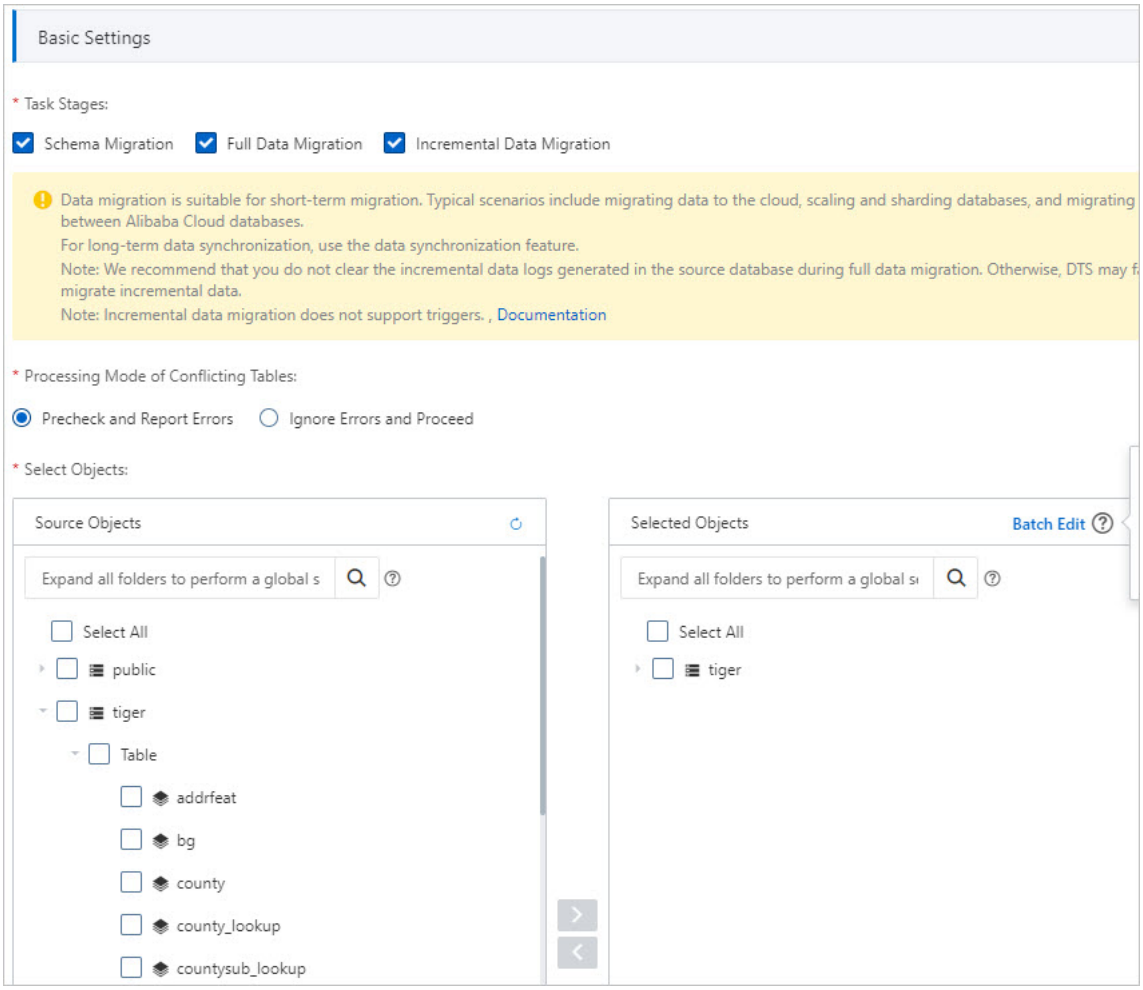
1.
2. In the upper-left corner of the page, select the region in which the data migration instance resides.
3. Warning

Section	Parameter	Description
N/A		
		Select **PostgreSQL**.
		Select an access method based on the deployment of the source database. In this example, select **Cloud Enterprise Network (CEN)**. **Note** If your source database is a self-managed database, you must deploy the network environment for the database. For more information, see Preparation overview.
		Select the region where the self-managed PostgreSQL database resides.
	CEN Instance ID	Select the ID of the CEN instance that hosts the self-managed PostgreSQL database.
	Connected VPC	Select the virtual private cloud (VPC) that is connected to the self-managed PostgreSQL database.
	IP Address	Enter the server IP address of the self-managed PostgreSQL database.
	Port Number	Enter the service port number of the self-managed SQL Server database. The default port number is **1433**.
	Database Name	Enter the name of the self-managed PostgreSQL database.
		Enter the account that is used to log on to the self-managed PostgreSQL database. For information about the permissions that are required for the account, see Permissions required for database accounts.
		Select **PostgreSQL**.

Section	Parameter	Description
		Select **Alibaba Cloud Instance**.
		Select the region where the destination instance resides.
	Instance ID	Select the ID of the destination instance.
	Database Name	Enter the name of the destination database in the instance.
		Enter the database account of the destination instance. For information about the permissions that are required for the account, see Permissions required for database accounts.

4.

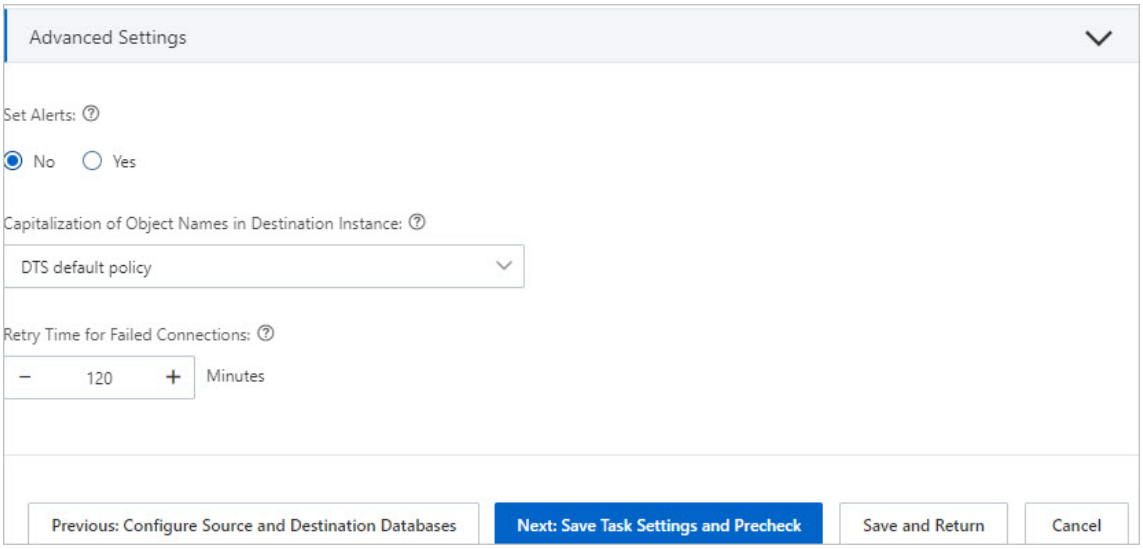
5. ○ **Basic Settings**



Parameter	Description

Parameter	Description
	Note
	In the **Selected Objects** section, right-click an object. In the dialog box that appears, select the SQL operations that you want to migrate. For more information, see SQL operations that can be migrated.

- **Advanced Settings**



Parameter	Description

6.
7.
8.
9.
10.

10.3. Migrate third-party databases to ApsaraDB

10.3.1. Migrate incremental data from an Amazon RDS for PostgreSQL instance to an ApsaraDB RDS for PostgreSQL instance

This topic describes how to migrate incremental data from an Amazon RDS for PostgreSQL instance to an ApsaraDB RDS for PostgreSQL instance by using Data Transmission Service (DTS). DTS supports schema migration, full data migration, and incremental data migration. You can select all of the supported migration types to ensure service continuity.

Prerequisites

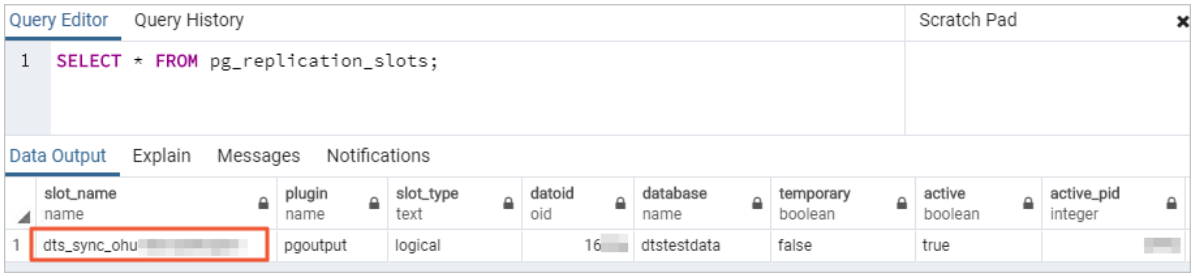
- The version of the Amazon RDS for PostgreSQL instance is 10.4 to 12.
- The **Public accessibility** option of the Amazon RDS for PostgreSQL instance is set to **Yes**. This ensures that DTS can access the instance over the Internet.
- The value of the rds.logical_replication parameter is set to *1*. This ensures that DTS can read incremental data from the Amazon RDS for PostgreSQL instance.
- An ApsaraDB RDS for PostgreSQL instance is created. For more information, see Create an ApsaraDB RDS for PostgreSQL instance.

  > **Note**
  > - The version of the ApsaraDB RDS for PostgreSQL instance is 10 or 11. To migrate data between different database versions, create a pay-as-you-go instance to verify compatibility.
  > - The available storage space of the ApsaraDB RDS for PostgreSQL instance must be larger than the total size of the data in the Amazon RDS for PostgreSQL instance.

Precautions

- DTS uses read and write resources of the source and destination databases during full data migration. This may increase the loads of the database servers. If the database performance is unfavorable, the specification is low, or the data volume is large, database services may become unavailable. For example, DTS occupies a large amount of read and write resources in the following cases: a large number of slow SQL queries are performed on the source database, the tables have no primary keys, or a deadlock occurs in the destination database. Before you migrate data, evaluate the impact of data migration on the performance of the source and destination databases. We recommend that you migrate data during off-peak hours. For example, you can migrate data when the CPU utilization of the source and destination databases is less than 30%.
- The objects to be migrated must have PRIMARY KEY or UNIQUE constraints and all fields must be unique. Otherwise, the destination database may contain duplicate data records and data migration may fail.
- A single data migration task can migrate data from only one database. To migrate data from multiple databases, you must create a data migration task for each database.
- In this scenario, DTS can migrate only data manipulation language (DML) operations, such as INSERT, DELETE, and UPDATE.
- During data migration, DTS creates a replication slot for the Amazon RDS for PostgreSQL instance. The replication slot is prefixed with `dts_sync_`. DTS automatically clears historical replication slots every 90 minutes to reduce storage usage.

> **Note** If the data migration task is released or fails, DTS automatically clears the replication slot. If a primary/secondary switchover is performed on the Amazon RDS for PostgreSQL instance, you must log on to the secondary database to clear the replication slot.



- If a data migration task fails, DTS automatically resumes the task. Before you switch your workloads to the destination instance, stop or release the data migration task. Otherwise, the data in the source instance will overwrite the data in the destination instance after the task is resumed.

Billing

Migration type	Task configuration fee	Internet traffic fee
Schema migration and full data migration	Free of charge.	Charged only when data is migrated from Alibaba Cloud over the Internet. For more information, see Pricing.
Incremental data migration	Charged. For more information, see Pricing.	

Migration types

- Schema migration

  DTS migrates the schemas of the required objects to the ApsaraDB RDS for PostgreSQL instance. DTS supports schema migration for the following types of objects: table, trigger, view, sequence, function, user-defined type, rule, domain, operation, and aggregate.

  > **Note** DTS does not migrate functions that are written in the C programming language.

- Full data migration

  DTS migrates historical data of the required objects from the Amazon RDS for PostgreSQL instance to the ApsaraDB RDS for PostgreSQL instance.

- Incremental data migration

  After full data migration is complete, DTS synchronizes incremental data from the Amazon RDS for PostgreSQL instance to the ApsaraDB RDS for PostgreSQL instance. Incremental data migration allows you to ensure service continuity when you migrate data between PostgreSQL databases.

Permissions required for database accounts

Database	Schema migration	Full data migration	Incremental data migration

Database	Schema migration	Full data migration	Incremental data migration
Amazon RDS for PostgreSQL	The USAGE permission on pg_catalog	The SELECT permission on the objects to be migrated	The rds_superuser permission
ApsaraDB RDS for PostgreSQL	The CREATE and USAGE permissions on the objects to be migrated	The permissions of the schema owner	The permissions of the schema owner

Data migration process

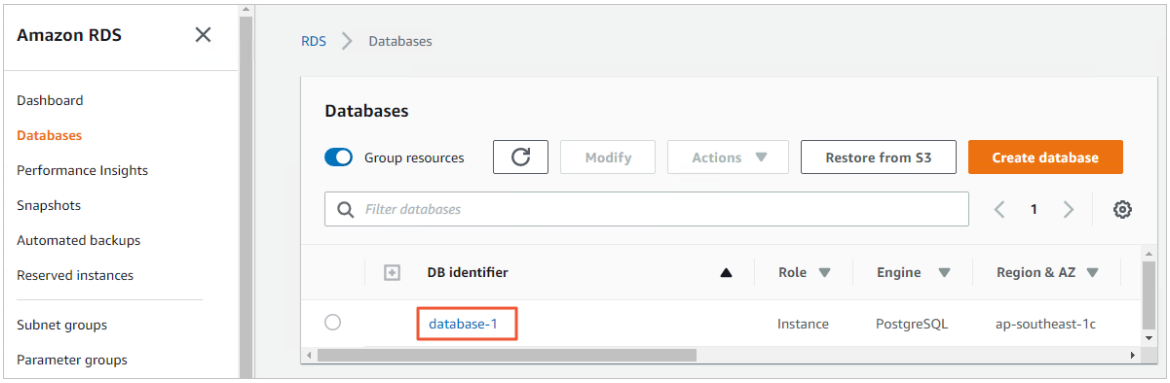
To prevent data migration failures caused by dependencies between objects, DTS migrates the schemas and data of the source PostgreSQL database in the following order:

1. Migrate the schemas of tables, views, sequences, functions, user-defined types, rules, domains, operations, and aggregates.
2. Perform full data migration.
3. Migrate the schemas of triggers and foreign keys.
4. Perform incremental data migration.

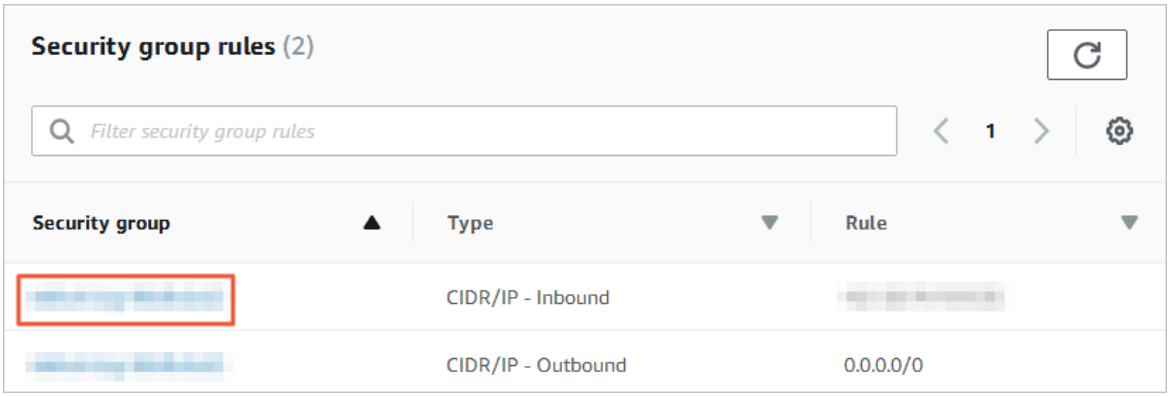
> **Note** Before incremental data migration, do not perform data definition language (DDL) operations on the objects in the Amazon RDS for PostgreSQL instance. Otherwise, the objects may fail to be migrated.

Before you begin

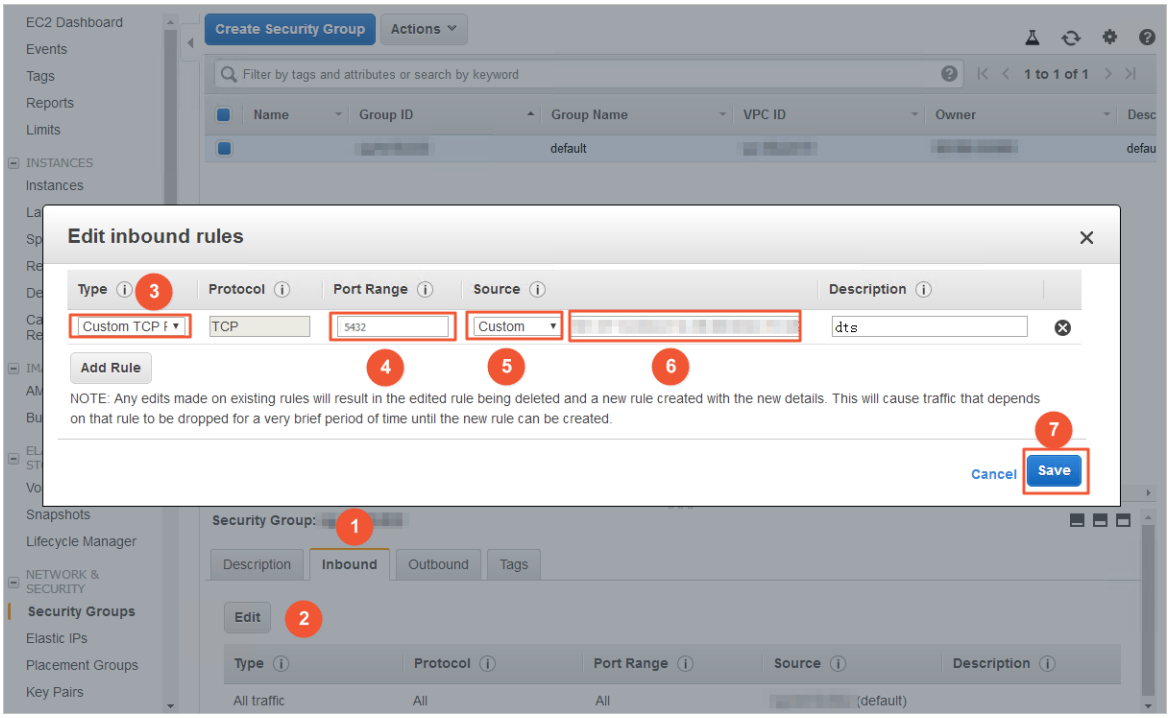
1. Log on to the Amazon RDS Management Console.
2. In the upper-right corner of the page, select the region where the destination instance resides.
3. In the left-side navigation pane, click **Databases**. On the page that appears, click the ID of the destination database. The **Basic Information** page appears.



4. In the **Security group rules** section, click the name of the security group corresponding to the existing inbound rule.



5. On the **Security Groups** page, click the Inbound tab in the Security Group section. On the Inbound tab, click Edit to add the CIDR blocks of DTS servers in the corresponding region to the inbound rule. For more information, see Add the CIDR blocks of DTS servers to the security settings of on-premises databases.

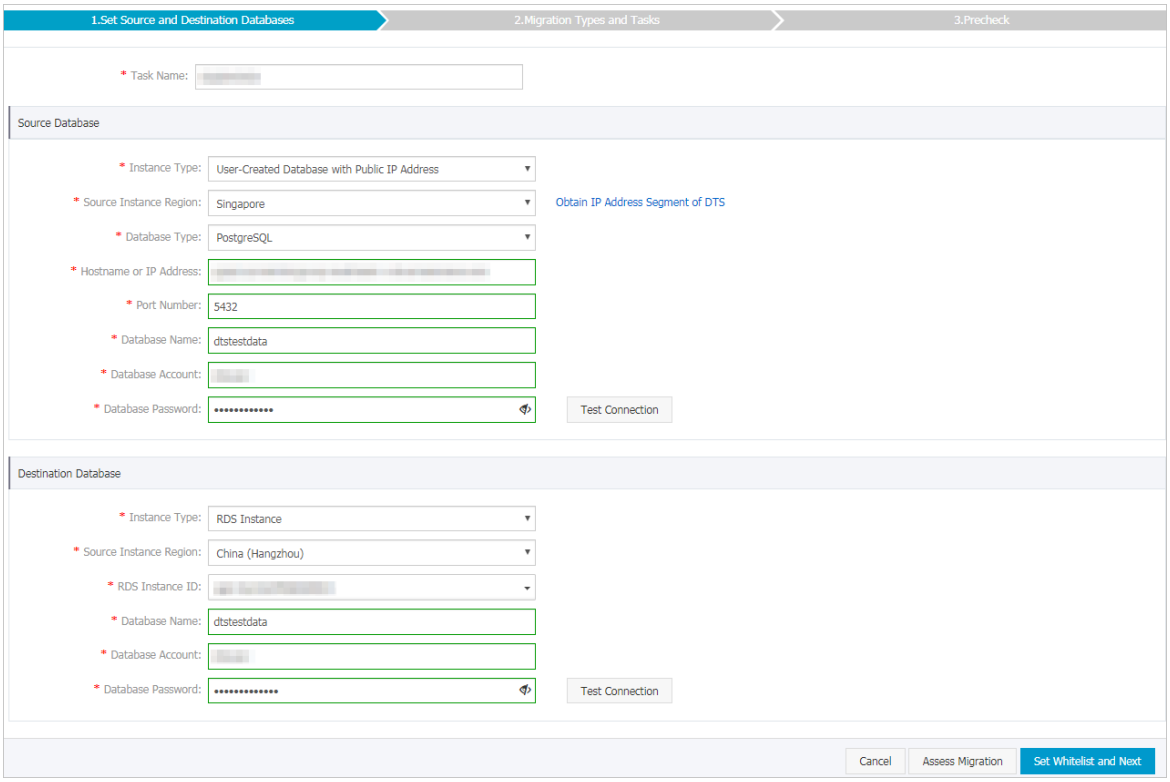


> **Note**
>
> - You need to add only the CIDR blocks of DTS servers that reside in the same region as the destination database. For example, the source database resides in the Singapore (Singapore) region and the destination database resides in the China (Hangzhou) region. You need to add only the CIDR blocks of DTS servers that reside in the China (Hangzhou) region.
> - You can add all of the required CIDR blocks to the inbound rule at a time.

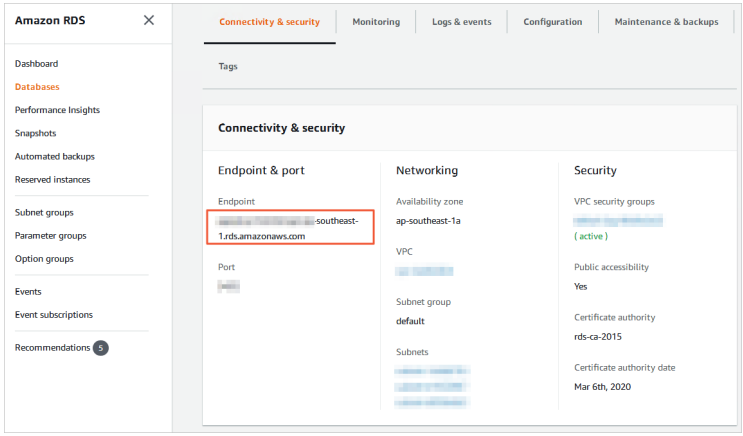
Procedure

1. Log on to the DTS console.
2. In the left-side navigation pane, click **Data Migration**.
3. At the top of the **Migration Tasks** page, select the region where the destination cluster resides.

4. In the upper-right corner of the page, click **Create Migration Task**.
5. Configure the source and destination databases.



Section	Parameter	Description
N/A	Task Name	DTS automatically generates a task name. We recommend that you specify an informative name for easy identification. You do not need to use a unique task name.
	Instance Type	Select **User-Created Database with Public IP Address**.
	Instance Region	Select the region where the source instance resides. If you select **User-Created Database with Public IP Address** as the instance type, you do not need to specify the Instance Region parameter.
	Database Type	Select **PostgreSQL**.

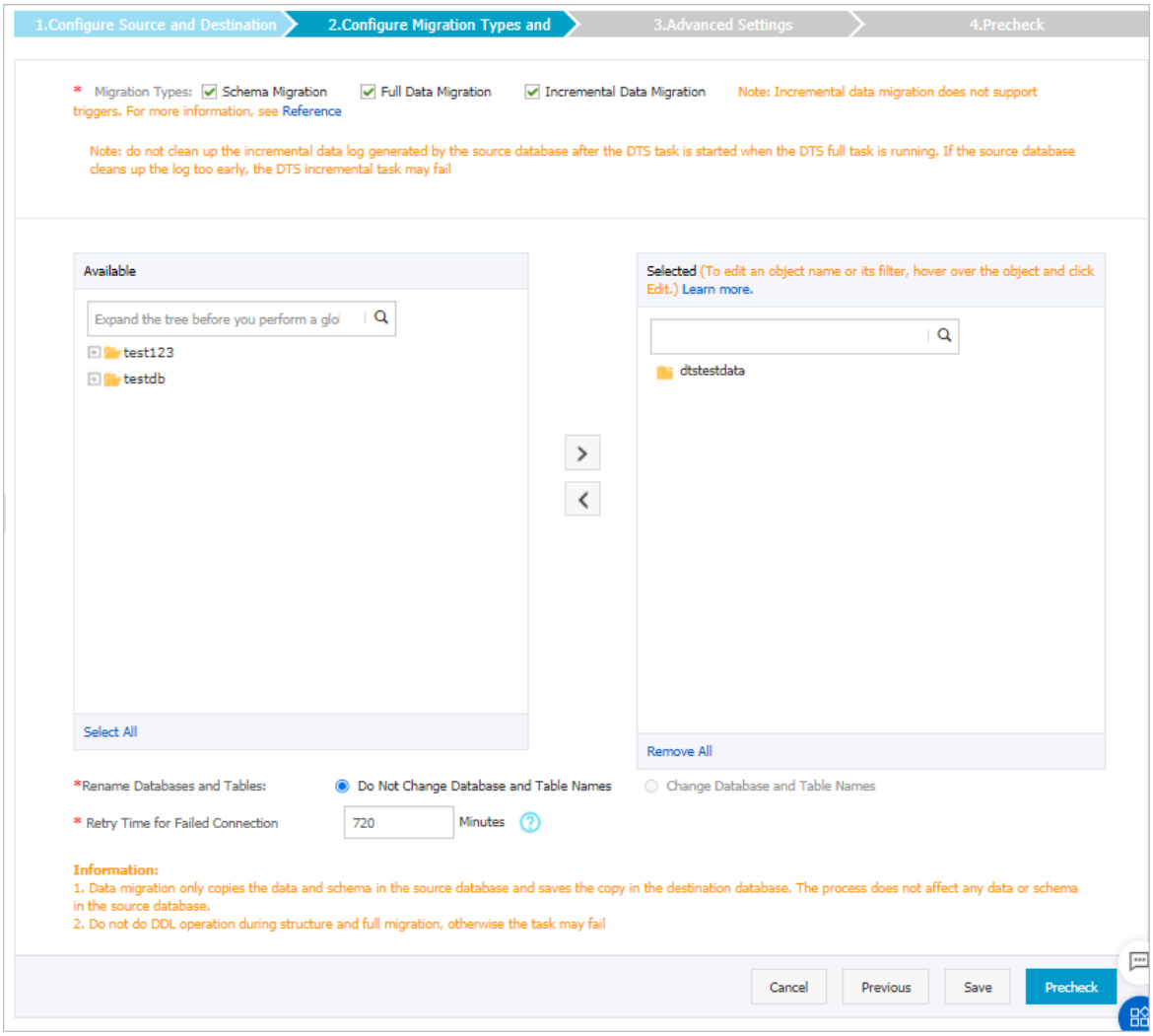
Section	Parameter	Description
Source Database	Hostname or IP Address	Enter the endpoint that is used to access the Amazon RDS for PostgreSQL instance. **Note** You can obtain the endpoint on the **Basic Information** page of the Amazon RDS for PostgreSQL instance. 
	Port Number	Enter the service port number of the Amazon RDS for PostgreSQL instance. The default port number is **5432**.
	Database Name	Enter the name of the source database in the Amazon RDS for PostgreSQL instance.
	Database Account	Enter the database account of the Amazon RDS for PostgreSQL instance. For information about the permissions that are required for the account, see Permissions required for database accounts.
	Database Password	Enter the password of the database account. **Note** After you specify the source database parameters, click **Test Connectivity** next to **Database Password** to verify whether the specified parameters are valid. If the specified parameters are valid, the **Passed** message appears. If the **Failed** message appears, click **Check** next to **Failed**. Modify the source database parameters based on the check results.
	Instance Type	Select **RDS Instance**.
	Instance Region	Select the region where the ApsaraDB RDS for PostgreSQL instance resides.
	RDS Instance ID	Select the ID of the ApsaraDB RDS for PostgreSQL instance.

Section	Parameter	Description
Destination Database	Database Name	Enter the name of the destination database in the ApsaraDB RDS for PostgreSQL instance. The name can be different from the name of the source database in the Amazon RDS for PostgreSQL instance. **Note** Before you configure the data migration task, you must create a database in the ApsaraDB RDS for PostgreSQL instance. For more information, see Create a database on an ApsaraDB RDS for PostgreSQL instance.
	Database Account	Enter the database account of the ApsaraDB RDS for PostgreSQL instance. For information about the permissions that are required for the account, see Permissions required for database accounts.
	Database Password	Enter the password of the database account. **Note** After you specify the destination database parameters, click **Test Connectivity** next to **Database Password** to verify whether the specified parameters are valid. If the specified parameters are valid, the **Passed** message appears. If the **Failed** message appears, click **Check** next to **Failed**. Modify the destination database parameters based on the check results.

6. In the lower-right corner of the page, click **Set Whitelist and Next**.

   Note DTS adds the CIDR blocks of DTS servers to the whitelist of the ApsaraDB RDS for PostgreSQL instance. This ensures that DTS servers can connect to the ApsaraDB RDS for PostgreSQL instance.

7. Select the migration types and the objects to be migrated.



Setting	Description
Select the migration types	Select **Schema Migration**, **Full Data Migration**, and **Incremental Data Migration**.
Select the objects to be migrated	Select one or more objects from the **Available** section and click the icon to add the objects to the **Selected** section. **Note** ◦ You can select columns, tables, or schemas as the objects to be migrated. ◦ By default, after an object is migrated to the destination instance, the name of the object remains unchanged. You can use the object name mapping feature to rename the objects that are migrated to the ApsaraDB RDS for PostgreSQL instance. For more information, see Object name mapping. ◦ If you use the object name mapping feature to rename an object, other objects that are dependent on the object may fail to be migrated.

Setting	Description
Specify whether to rename objects	You can use the object name mapping feature to rename the objects that are migrated to the destination instance. For more information, see Object name mapping.
Specify the retry time for failed connections to the source or destination database	By default, if DTS fails to connect to the source or destination database, DTS retries within the next 12 hours. You can specify the retry time based on your needs. If DTS reconnects to the source and destination databases within the specified time, DTS resumes the data migration task. Otherwise, the data migration task fails. **Note** When DTS retries a connection, you are charged for the DTS instance. We recommend that you specify the retry time based on your business needs. You can also release the DTS instance at your earliest opportunity after the source and destination instances are released.

8. In the lower-right corner of the page, click **Precheck**.

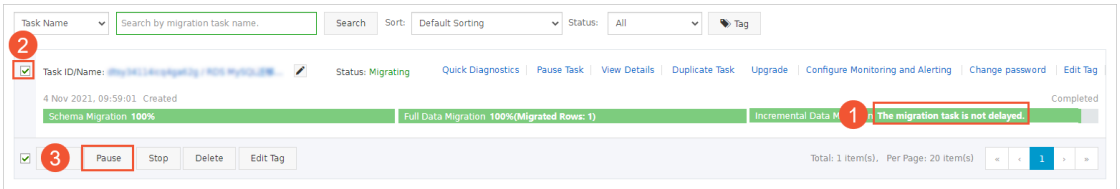
   Note
   - Before you can start the data migration task, a precheck is performed. You can start the data migration task only after the task passes the precheck.
   - If the task fails to pass the precheck, you can click the icon next to each failed item to view details.
     - You can troubleshoot the issues based on the causes and run a precheck again.
     - If you do not need to troubleshoot the issues, you can ignore failed items and run a precheck again.

9. After the task passes the precheck, click **Next**.
10. In the **Confirm Settings** dialog box, specify the **Channel Specification** parameter and select **Data Transmission Service (Pay-As-You-Go) Service Terms**.
11. Click **Buy and Start** to start the data migration task.

    Note A task does not automatically stop during incremental data migration. You must manually stop the task. We recommend that you select an appropriate time to manually stop the data migration task. For example, you can stop the task during off-peak hours or before you switch your workloads to the destination instance.

    i. Wait until **Incremental Data Migration** and **The migration task is not delayed** appear in the progress bar of the migration task. Then, stop writing data to the source database for a few minutes. The delay time of **incremental data migration** may be displayed in the progress bar.

ii. Wait until the status of **incremental data migration** changes to **The migration task is not delayed** again. Then, manually stop the migration task.



12. Switch your workloads to the ApsaraDB RDS for PostgreSQL instance.

10.3.2. Migrate full data from an Amazon RDS for PostgreSQL instance to an ApsaraDB RDS for PostgreSQL instance

This topic describes how to migrate full data from an Amazon RDS for PostgreSQL instance to an ApsaraDB RDS for PostgreSQL instance by using Data Transmission Service (DTS).

Prerequisites

- An Amazon RDS for PostgreSQL instance is created and the database version is 9.4, 9.5, 9.6, or 10.0.
- The **Public accessibility** option of the Amazon RDS for PostgreSQL instance is set to **Yes**. This ensures that DTS can access the instance over the Internet.
- An ApsaraDB RDS for PostgreSQL instance is created. For more information, see Create an ApsaraDB RDS for PostgreSQL instance.

  > **Note** The database version of the ApsaraDB RDS for PostgreSQL instance is 9.4 or 10.0.

- The available storage space of the ApsaraDB RDS for PostgreSQL instance is larger than the total size of the data in the Amazon RDS for PostgreSQL instance.

Precautions

- DTS uses read and write resources of the source and destination databases during full data migration. This may increase the loads of the database servers. If the database performance is unfavorable, the specification is low, or the data volume is large, database services may become unavailable. For example, DTS occupies a large amount of read and write resources in the following cases: a large number of slow SQL queries are performed on the source database, the tables have no primary keys, or a deadlock occurs in the destination database. Before you migrate data, evaluate the impact of data migration on the performance of the source and destination databases. We recommend that you migrate data during off-peak hours. For example, you can migrate data when the CPU utilization of the source and destination databases is less than 30%.
- You cannot use DTS to migrate incremental data from an Amazon RDS for PostgreSQL instance to an ApsaraDB RDS for PostgreSQL instance. Before you start the data migration task, you must stop the services that run on the Amazon RDS for PostgreSQL instance. To ensure data consistency, we recommend that you do not write data to the Amazon RDS for PostgreSQL instance during data migration.
- A single data migration task can migrate data from only one database. To migrate data from multiple databases, you must create a data migration task for each database.
- Functions that are written in the C programming language cannot be migrated.

- The tables to be migrated in the source database must have PRIMARY KEY or UNIQUE constraints and all fields must be unique. Otherwise, the destination database may contain duplicate data records.
- If a data migration task fails, DTS automatically resumes the task. Before you switch your workloads to the destination instance, stop or release the data migration task. Otherwise, the data in the source instance will overwrite the data in the destination instance after the task is resumed.
- To ensure that the data migration task runs as expected, you can perform a primary/secondary switchover only on a V11 ApsaraDB RDS for PostgreSQL instance. In this case, you must set the `rds_failover_slot_mode` parameter to `sync`. For more information, see Logical Replication Slot Failover.

  Warning If you perform a primary/secondary switchover on a self-managed PostgreSQL database or an ApsaraDB RDS for PostgreSQL instance of a version other than V11, the data migration task stops.

Billing

Migration type	Task configuration fee	Internet traffic fee
Schema migration and full data migration	Free of charge.	Charged only when data is migrated from Alibaba Cloud over the Internet. For more information, see Pricing.

Migration types

- Schema migration

  DTS migrates the schemas of the required objects to the ApsaraDB RDS for PostgreSQL instance. DTS supports schema migration for the following types of objects: table, trigger, view, sequence, function, user-defined type, rule, domain, operation, and aggregate.

- Full data migration

  DTS migrates historical data of the required objects from the Amazon RDS for PostgreSQL instance to the ApsaraDB RDS for PostgreSQL instance.

Permissions required for database accounts

Database	Schema migration	Full data migration
Amazon RDS for PostgreSQL	The USAGE permission on pg_catalog	The SELECT permission on the objects to be migrated
ApsaraDB RDS for PostgreSQL	The CREATE and USAGE permissions on the objects to be migrated	The permissions of the schema owner

Process of full data migration

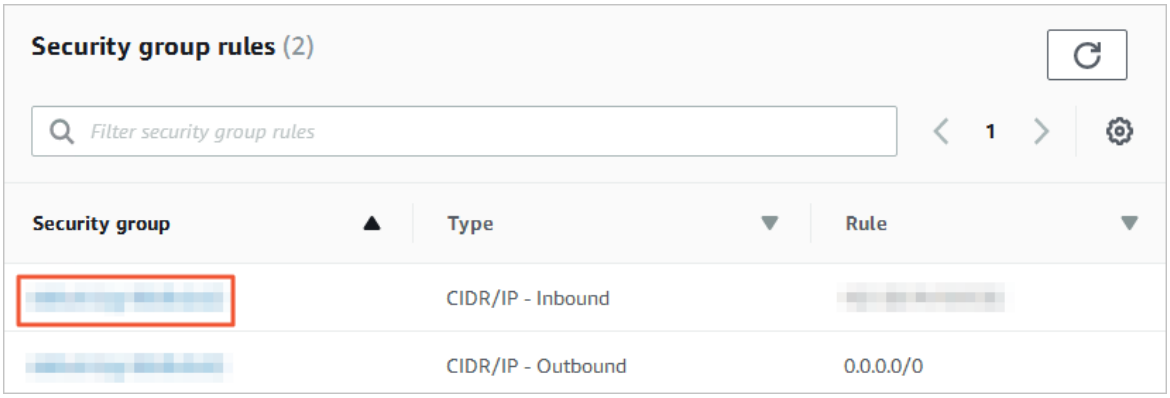
To prevent data migration failures caused by dependencies between objects, DTS migrates the schemas and data of the source PostgreSQL database in the following order:

1. Migrate the schemas of tables, views, sequences, functions, user-defined types, rules, domains, operations, and aggregates.
2. Perform full data migration.

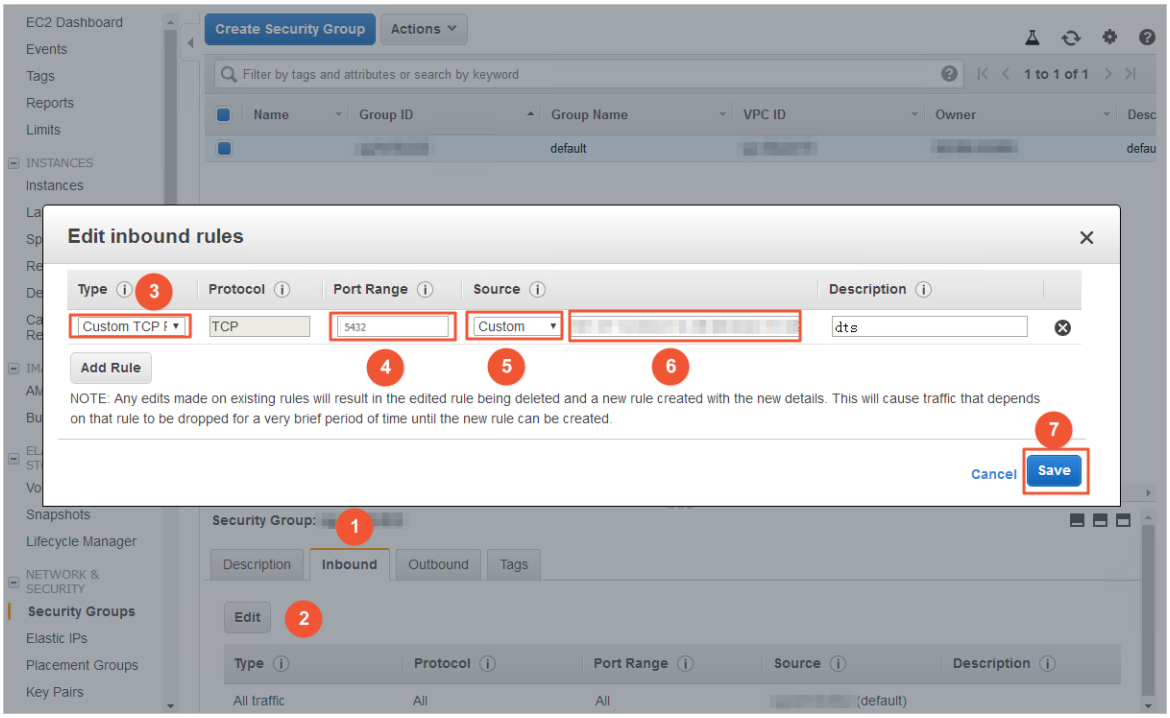
3. Migrate the schemas of triggers and foreign keys.

Preparation 1: Edit the inbound rule of the Amazon RDS for PostgreSQL instance

1. Log on to the Amazon RDS Management Console.
2. Go to the **Basic Information** page of the Amazon RDS for PostgreSQL instance.
3. In the **Security group rules** section, click the name of the security group corresponding to the existing inbound rule.



4. On the **Security Groups** page, click the Inbound tab in the Security Group section. On the Inbound tab, click Edit to add the CIDR blocks of DTS servers in the corresponding region to the inbound rule. For more information, see Add the CIDR blocks of DTS servers to the security settings of on-premises databases.



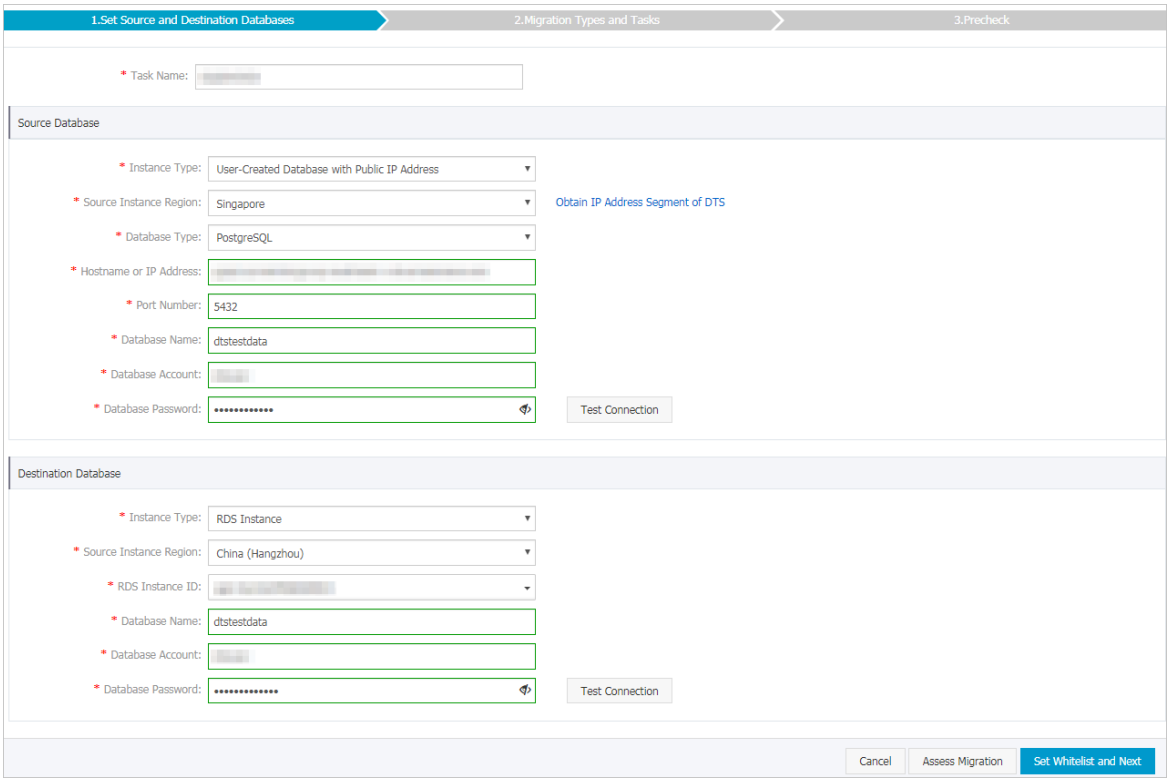
> **Note**
>
> - You need to add only the CIDR blocks of DTS servers that reside in the same region as the destination database. For example, the source database resides in the Singapore (Singapore) region and the destination database resides in the China (Hangzhou) region. You need to add only the CIDR blocks of DTS servers that reside in the China (Hangzhou) region.
> - You can add all of the required CIDR blocks to the inbound rule at a time.

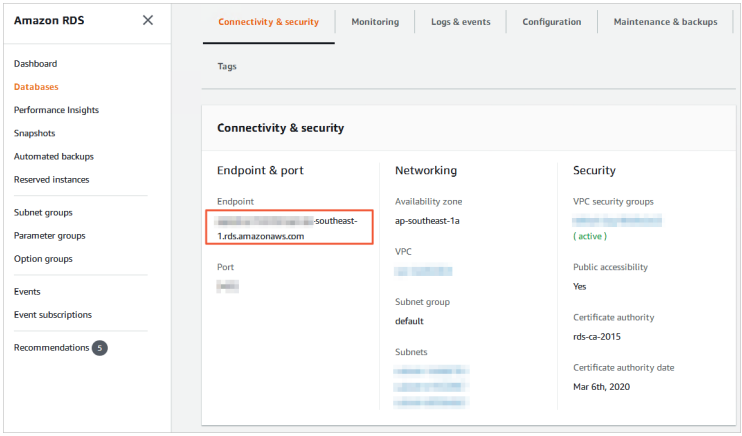
Preparation 2: Create a database and schema in the destination RDS instance

Create a database and schema in the destination RDS instance based on the database and schema information of the objects to be migrated. The schema name of the source and destination databases must be the same. For more information, see Create a database on an ApsaraDB RDS for PostgreSQL instance and Appendix: User and schema management.

Procedure

1. Log on to the DTS console.
2. In the left-side navigation pane, click **Data Migration**.
3. At the top of the **Migration Tasks** page, select the region where the destination cluster resides.
4. In the upper-right corner of the page, click **Create Migration Task**.
5. Configure the source and destination databases.



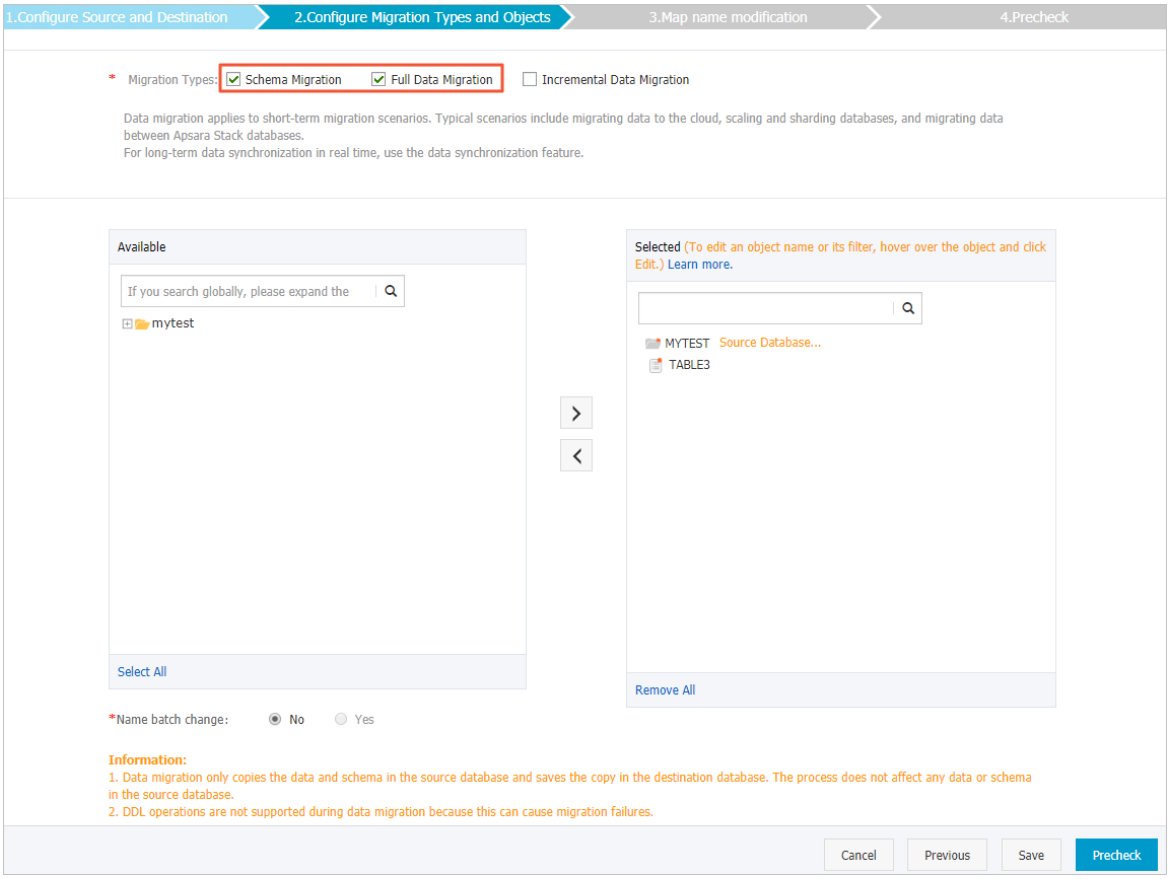
Section	Parameter	Description
N/A	Task Name	DTS automatically generates a task name. We recommend that you specify an informative name to identify the task. You do not need to use a unique task name.
Source Database	Instance Type	Select **User-Created Database with Public IP Address**.
	Instance Region	Select the region where the source instance resides. If you select **User-Created Database with Public IP Address** as the instance type, you do not need to specify the Instance Region parameter.
	Database Type	Select **PostgreSQL**.
	Hostname or IP Address	Enter the endpoint that is used to access the Amazon RDS for PostgreSQL instance. **Note** You can find the endpoint on the **Basic Information** page of the Amazon RDS for PostgreSQL instance. 
	Port Number	Enter the service port number of the Amazon RDS for PostgreSQL instance. The default port number is **5432**.
	Database Name	Enter the name of the source database in the Amazon RDS for PostgreSQL instance.
	Database Account	Enter the database account of the Amazon RDS for PostgreSQL instance. For information about the permissions that are required for the account, see Permissions required for database accounts.
	Database Password	Enter the password of the database account. **Note** After you specify the source database parameters, click **Test Connectivity** next to **Database Password** to verify whether the specified parameters are valid. If the specified parameters are valid, the **Passed** message appears. If the **Failed** message appears, click **Check** next to **Failed**. Modify the source database parameters based on the check results.

Section	Parameter	Description
Destination Database	Instance Type	Select **RDS Instance**.
	Instance Region	Select the region where the ApsaraDB RDS for PostgreSQL instance resides.
	RDS Instance ID	Select the ID of the ApsaraDB RDS for PostgreSQL instance.
	Database Name	Enter the name of the destination database in the ApsaraDB RDS for PostgreSQL instance. The name can be different from the name of the source database in the Amazon RDS for PostgreSQL instance. **Note** Before you configure the data migration task, you must create a database and schema in the ApsaraDB RDS for PostgreSQL instance. For more information, see Preparation 2: Create a database and schema in the destination RDS instance.
	Database Account	Enter the database account of the ApsaraDB RDS for PostgreSQL instance. For information about the permissions that are required for the account, see Permissions required for database accounts.
	Database Password	Enter the password of the database account. **Note** After you specify the destination database parameters, click **Test Connectivity** next to **Database Password** to verify whether the specified parameters are valid. If the specified parameters are valid, the **Passed** message appears. If the **Failed** message appears, click **Check** next to **Failed**. Modify the destination database parameters based on the check results.

6. In the lower-right corner of the page, click **Set Whitelist and Next**.

   Note DTS adds the CIDR blocks of DTS servers to the whitelist of the ApsaraDB RDS for PostgreSQL instance. This ensures that DTS servers can connect to the ApsaraDB RDS for PostgreSQL instance.

7. Select the migration types and the objects to be migrated.



Setting	Description
Select the migration types	Select **Schema Migration** and **Full Data Migration**. In this scenario, **Incremental Data Migration** is not supported. **Note** To ensure data consistency, we recommend that you do not write data to the Amazon RDS for PostgreSQL instance during data migration.
Select the objects to be migrated	Select one or more objects from the **Available** section and click the icon to move the objects to the **Selected** section. **Note** - You can select columns, tables, or databases as the objects to be migrated. - After an object is migrated to the destination database, the name of the object remains unchanged. You can use the object name mapping feature to rename the objects that are migrated to the destination database. For more information, see Object name mapping. - If you use the object name mapping feature to rename an object, other objects that are dependent on the object may fail to be migrated.

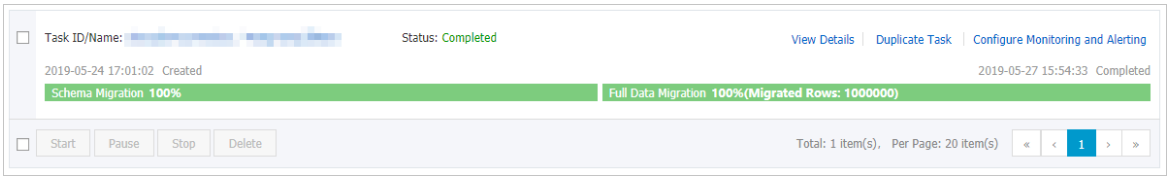
Setting	Description
Specify whether to rename objects	You can use the object name mapping feature to rename the objects that are migrated to the destination instance. For more information, see Object name mapping.
Specify the retry time for failed connections to the source or destination database	By default, if DTS fails to connect to the source or destination database, DTS retries within the next 12 hours. You can specify the retry time based on your needs. If DTS reconnects to the source and destination databases within the specified time, DTS resumes the data migration task. Otherwise, the data migration task fails. **Note** When DTS retries a connection, you are charged for the DTS instance. We recommend that you specify the retry time based on your business needs. You can also release the DTS instance at your earliest opportunity after the source and destination instances are released.

8. In the lower-right corner of the page, click **Precheck**.

   Note
   - Before you can start the data migration task, a precheck is performed. You can start the data migration task only after the task passes the precheck.
   - If the task fails to pass the precheck, you can click the icon next to each failed item to view details.
     - You can troubleshoot the issues based on the causes and run a precheck again.
     - If you do not need to troubleshoot the issues, you can ignore failed items and run a precheck again.

9. After the task passes the precheck, click **Next**.
10. In the **Confirm Settings** dialog box, specify the **Channel Specification** parameter and select **Data Transmission Service (Pay-As-You-Go) Service Terms**.
11. Click **Buy and Start** to start the data migration task.

    Note We recommend that you do not manually stop the task during full data migration. Otherwise, the data migrated to the destination database will be incomplete. You can wait until the data migration task automatically stops.



12. Switch your workloads to the ApsaraDB RDS for PostgreSQL instance.

10.4. Migrate data between ApsaraDB RDS for PostgreSQL instances

10.4.1. Use DTS to migrate data between ApsaraDB RDS for PostgreSQL instances

10.4.2. Use the cloud migration feature to migrate data between ApsaraDB RDS for PostgreSQL instances

ApsaraDB RDS for PostgreSQL supports the cloud migration feature. You can use this feature to migrate data between ApsaraDB RDS for PostgreSQL instances.

Prerequisites

The destination RDS instance is created and meets the following conditions:

- The destination RDS instance runs the same major engine version as the source RDS instance. The supported major engine versions are PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, and PostgreSQL 14.
- The destination RDS instance is a primary instance. Read-only RDS instances do not support cloud migration.
- The destination RDS instance is equipped with standard SSDs or enhance SSDs (ESSDs).
- The destination RDS instance is empty. The available storage of the destination RDS instance is greater than or equal to the size of the data in the source RDS instance.

Procedure

> **Note** The following procedure shows how to migrate data between RDS instances that reside in different virtual private clouds (VPCs) in the China (Hangzhou) region.

1. Configure the source RDS instance.

i. Create an IP address whitelist.

For more information see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance. You must set the **IP Addresses** parameter to the value of the **Public IP Address** parameter of the destination RDS instance.

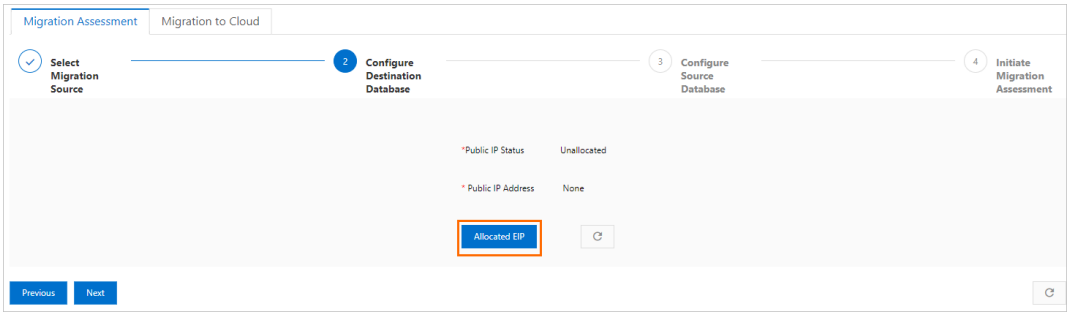
You can perform the following operations to view the value of the **Public IP Address** parameter of the destination RDS instance:

a.

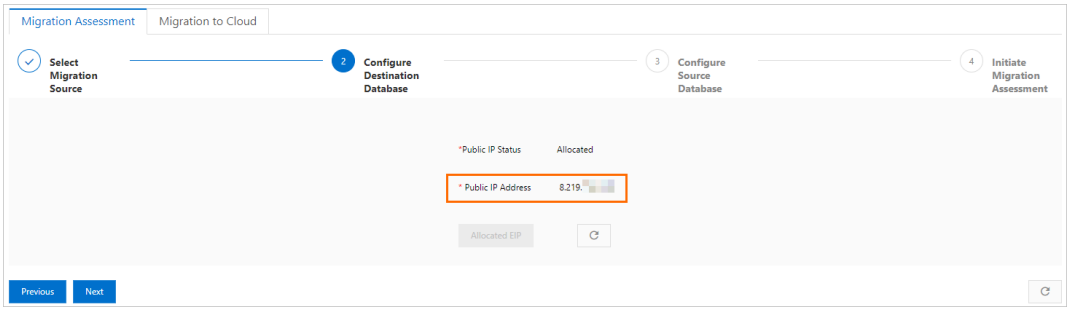
b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.

c. In the **Select Migration Source** step of the configuration wizard, select **PostgreSQL migration with public network address (including migrating from other cloud vendors)** and click **Next**.

d. In the **Configure Destination Database** step, click **Allocated EIP**.



e. Refresh the page and check the value of the **Public IP Address** parameter.



ii. Create a privileged account.

For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance. When you create a privileged account, you must set the **Account Type** parameter to Privileged Account.

> **Note** The privileged account is used to migrate data and must have the CREATE ROLE, REPLICATION, and pg_monitor permissions. If you have a privileged account, skip this step.

2. Configure the destination RDS instance.

i. Perform a cloud migration assessment.

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration Assessment** tab.

c. In the **Select Migration Source** step of the configuration wizard, select **PostgreSQL migration with public network address (including migrating from other cloud vendors)** and click **Next**.

d. In the **Configure Destination Database** step of the configuration wizard, click **Next**.

e. In the **Configure Source Database** step of the configuration wizard, select all listed items and click **Next**. Before you start the cloud migration, you must complete the preparations that are described in the listed items.

f. In the **Initiate Migration Assessment** step of the configuration wizard, configure the information about the source RDS instance.

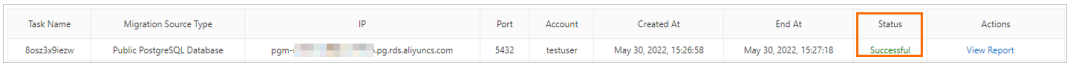
Parameter	Description
Migration Task Name	The name of the cloud migration task. ApsaraDB RDS automatically generates a name for the cloud migration task. You do not need to modify the generated name.
Source Public/DNS IP	The public endpoint of the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Source Port	The port number of the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Username	The username of the privileged account of the source RDS instance.
Password	The password of the privileged account of the source RDS instance.

g. Click **Create Migration Assessment Task**.

> **Note** During the cloud migration assessment, the status of the destination RDS instance changes to **Maintaining Instance**.

After the cloud migration assessment is complete, you can view the status of the cloud migration assessment task on the **Migration Assessment** tab.

- If the status of the cloud migration assessment task indicates a **success**, you can start the cloud migration.
- If the status of the cloud migration assessment task indicates a **failure**, you can click **View Report** in the Actions column to view and handle the reported errors. For more information about common errors, see Introduction to cloud migration assessment reports.



ii. Migrate data to the destination RDS instance.

a.

b. In the left-side navigation pane, click **Migrate to Cloud**. On the page that appears, click the **Migration to Cloud** tab. On the Migration to Cloud tab, click Create Cloud Migration Task.

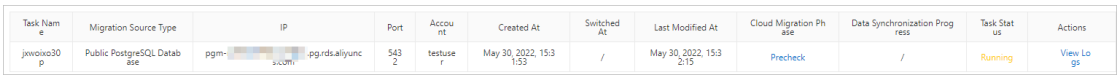
c. In the **Create Cloud Migration Task** dialog box, select the cloud migration assessment task whose status indicates a success from the **Associated Assessment Task** drop-down list.

> **Note** After you select a cloud migration assessment task from the **Associated Assessment Task** drop-down list, ApsaraDB RDS automatically determines the values of the **Migration Source Type**, **Source IP/DNS**, **Source Port**, and **Username** parameters. You do not need to configure these parameters.

d. Click **Initiate Migration to Cloud**. ApsaraDB RDS automatically starts the cloud migration task.

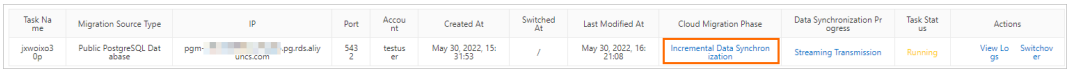
> **Notice** During the cloud migration, the status of the destination RDS instance changes to **Migrating Data In**. You can read data from and write data to the source RDS instance. However, do not migrate data from or to the source RDS instance, restart the source RDS instance, or change the specifications of the source RDS instance.

After you start the cloud migration task, the page that is shown in the following figure is displayed.



iii. Switch the workloads of the source RDS instance over to the destination RDS instance.

a. Click the link in the **Cloud Migration Phase** column to view the progress of the cloud migration task.



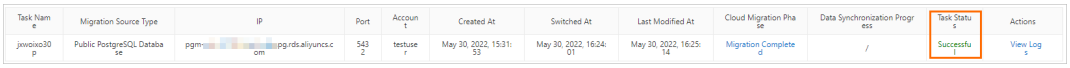
b. If the value in the Cloud Migration Phase column is **Incremental Data Synchronization**, click **Switchover** in the **Actions** column of the cloud migration task to switch the workloads of the source RDS instance over to the destination RDS instance.

c. In the **Switchover** dialog box, set the source RDS instance to read-only. Otherwise, stop the connected application from writing data to the source RDS instance.

> **Note** You can also use the privileged account of the source RDS instance to execute the following statements to set the source RDS instance to read-only:
>
> ```
> -- Sets the database to read-only.
> ALTER SYSTEM SET default_transaction_read_only=on;
> -- Reloads the parameter configuration for the modification to take effect.
> SELECT pg_reload_conf();
> -- Terminates all existing sessions.
> SELECT pg_terminate_backend(pid) FROM pg_stat_activity
> WHERE usename not in ('replicator', 'monitor', 'pgsql', 'aurora') AND pid != p
> g_backend_pid();
> ```

d. Select all check boxes and click **Switch Now**. Then, wait until the cloud migration is complete.

After the workloads of the source RDS instance are switched over to the destination RDS instance, the page that is shown in the following figure is displayed.



What to do next

To connect your applications to the destination RDS instance without the need to modify business code after the migration, perform the following operations on the destination RDS instance:

1. Modify the whitelist settings of the destination RDS instance to ensure that the whitelists of the destination RDS instance are the same as the whitelists of the source RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
2. Change the endpoints that are used to connect to the destination RDS instance to ensure that these endpoints are the same as the endpoints that are used to connect to the source RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.

   For example, if the endpoint that is used to connect to the source RDS instance is `pgm-aaa.pg.rds.aliyuncs.com` and the endpoint that is used to connect to the destination RDS instance is `pgm-bbb.pg.rds.aliyuncs.com`, perform the following steps to change the endpoints:

   i. Change the endpoint that is used to connect to the source RDS instance from `pgm-aaa.pg.rds.aliyuncs.com` to `pgm-ccc.pg.rds.aliyuncs.com`.

   ii. Change the endpoint that is used to connect to the destination RDS instance from `pgm-bbb.pg.rds.aliyuncs.com` to `pgm-aaa.pg.rds.aliyuncs.com`.

11.Manage DataConnectors

11.1. Configure one-way data synchronization between ApsaraDB RDS for PostgreSQL instances

This topic describes how to configure one-way data synchronization between instances in the Data Transmission Service (DTS) console.

Prerequisites

- The source and destination ApsaraDB RDS for PostgreSQL instances are created. For more information, see Create an ApsaraDB RDS for PostgreSQL instance.

  > **Note** To ensure compatibility, the version of the destination database must be the same as or later than the version of the source database.
  >
  > If the version of the destination database is earlier than the version of the source database, database compatibility issues may occur.

- The available storage space of the destination instance is larger than the total size of the data in the source instance.

Limits

Supported synchronization topologies

- One-way one-to-one synchronization
- One-way one-to-many synchronization
- One-way cascade synchronization
- One-way many-to-one synchronization

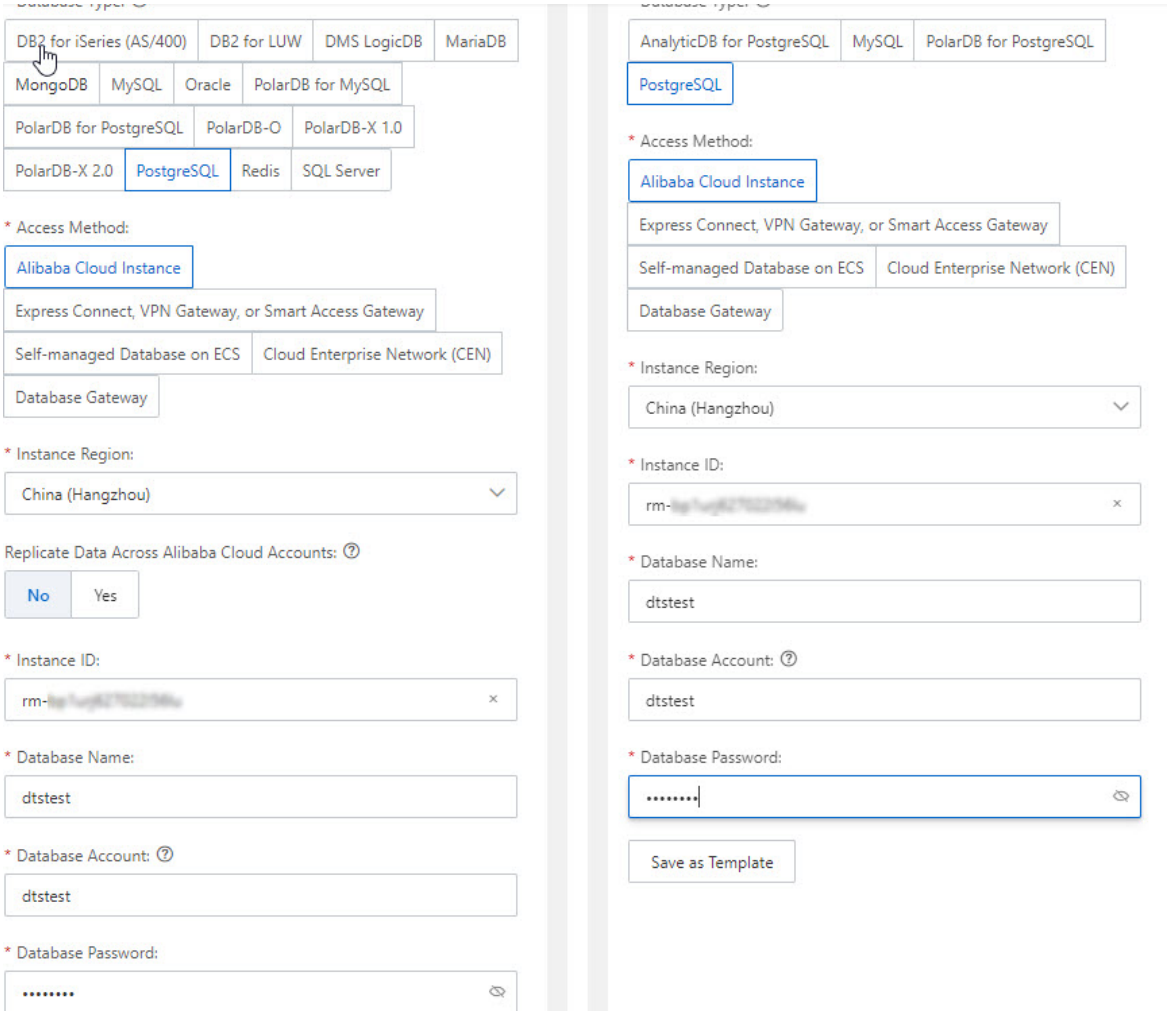
For more information, see Synchronization topologies.

SQL operations that can be synchronized

Operation type	SQL statements
DML	INSERT, UPDATE, and DELETE

Procedure

1.
2.
3. **Warning**



Section	Parameter	Description
N/A		
		Select **PostgreSQL**.
		Select **Alibaba Cloud Instance**.
		Select the region where the source instance resides.
	Instance ID	Select the ID of the source instance.
	Database Name	Enter the name of the source database in the instance.

Section	Parameter	Description
		Enter the database account of the source instance. A privileged account has the required permissions. **Note** If the version of the source ApsaraDB RDS for PostgreSQL instance is 9.4 and you synchronize only DML operations, the database account must have the REPLICATION permission.
		Select **PostgreSQL**.
		Select **Alibaba Cloud Instance**.
		Select the region where the destination instance resides.
	Instance ID	Select the ID of the destination instance.
	Database Name	Enter the name of the destination database in the instance.
		Enter the database account of the destination instance. A privileged account has the required permissions.

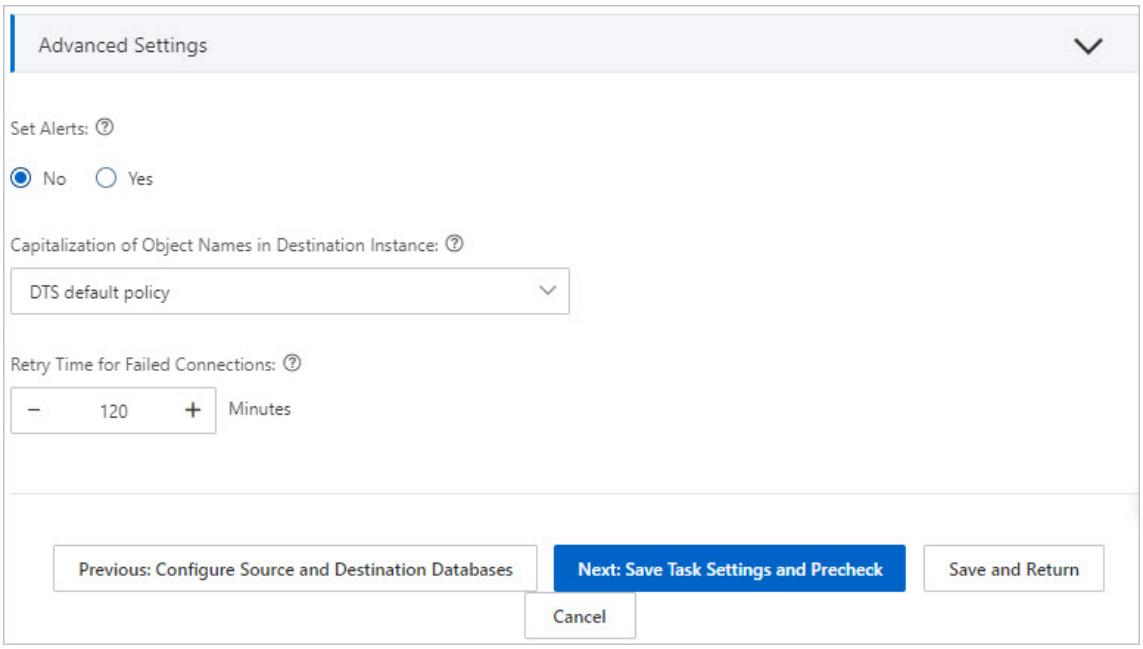
4.

5. ○ **Basic Settings**

Parameter	Description
Synchronization Topology	Select **One-way Synchronization**.
	Note

Parameter	Description
	In the **Selected Objects** section, right-click an object. In the dialog box that appears, select the SQL operations that you want to synchronize. For more information, see SQL operations that can be synchronized.

- **Advanced Settings**



Parameter	Description

6.
7.
8.
9.
10.

11.2. Configure two-way data synchronization between ApsaraDB RDS for PostgreSQL instances

Data Transmission Service (DTS) supports two-way data synchronization between two PostgreSQL databases, such as databases on an instance and self-managed PostgreSQL databases. This feature is applicable to scenarios such as active geo-redundancy (unit-based) and geo-disaster recovery. This topic describes how to configure two-way data synchronization between instances. You can also follow the procedure to configure data synchronization tasks for self-managed PostgreSQL databases.

Prerequisites

- The source and destination instances are created. For more information, see Create an ApsaraDB RDS for PostgreSQL instance.
- The wal_level parameter is set to logical for the source and destination instances. For more information, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.

Limits

Note By default, DTS disables FOREIGN KEY constraints for the destination database in a data synchronization task. Therefore, the cascade and delete operations of the source database are not synchronized to the destination database.

Supported synchronization topologies

DTS supports two-way data synchronization only between two PostgreSQL databases. DTS does not support two-way data synchronization between multiple PostgreSQL databases.

Conflict detection

Procedure

1. Purchase an instance for two-way data synchronization. For more information, see Purchase a DTS instance.

   Notice On the buy page, set both Source Instance and Destination Instance to **PostgreSQL** and set Synchronization Topology to **Two-way Synchronization**.

2.
3.
4. In the upper-left corner of the page, select the region where the destination instance resides.
5. Find the data synchronization instance and click **Configure Task** in the Actions column of the first data synchronization task.
6. **Warning**

Section	Parameter	Description
N/A		
		Select **PostgreSQL**.
		Select **Alibaba Cloud Instance**.

Section	Parameter	Description
Source Database		The source region that you selected on the buy page. You cannot change the value of this parameter.
	Instance ID	Select the ID of the source instance.
	Database Name	Enter the name of the source database in the instance.
		Enter the privileged account of the instance. The account must be the owner of the database. For more information about how to create an account on an ApsaraDB RDS for PostgreSQL instance and grant permissions to this account, see Create an account on an ApsaraDB RDS for PostgreSQL instance and Create a database on an ApsaraDB RDS for PostgreSQL instance.
Destination Database		Select **PostgreSQL**.
		Select **Alibaba Cloud Instance**.
		The destination region that you selected on the buy page. You cannot change the value of this parameter.
	Instance ID	Select the ID of the destination instance.
	Database Name	Enter the name of the destination database in the instance.
		Enter the privileged account of the instance. The account must be the owner of the database. For more information about how to create an account on an ApsaraDB RDS for PostgreSQL instance and grant permissions to this account, see Create an account on an ApsaraDB RDS for PostgreSQL instance and Create a database on an ApsaraDB RDS for PostgreSQL instance.

7.

8. ○ **Basic Settings**

Parameter	Description
	If you encounter the conflicts described in Conflict detection, select a conflict resolution policy based on your business needs.
	Note

Parameter	Description
	In the **Selected Objects** section, right-click an object. In the dialog box that appears, select the SQL operations that you want to synchronize.

- **Advanced Settings**

Parameter	Description
	This parameter is set to **DTS default policy**. In this scenario, you cannot change the capitalization of object names in the destination database. For two-way data synchronization between PostgreSQL databases, all object names in the destination database are in lowercase. For more information, see Specify the capitalization of object names in the destination instance.

9.
10. Wait until the **Success Rate** becomes **100%**. Then, click **Back**.
11. The data synchronization task in the forward direction starts. You can view the progress of the task in the task list.
12. Wait until **initial synchronization** is completed and the data synchronization task in the forward direction is in the **Running** state. You can view the status of the data synchronization task on the Data Synchronization page.
13. Find the data synchronization task in the reverse direction and click **Configure Task**.
14. On the **Create Task** page, **configure the source and destination databases**, as described in Step 5.

    Notice When you configure the data synchronization task in the reverse direction, you must select the correct source and destination instances. The source instance in the reverse direction corresponds to the destination instance in the forward direction. The destination instance in the reverse direction corresponds to the source instance in the forward direction. You must also make sure that the parameter settings such as the database name, account, and password are consistent.

15. , as described in Step 7. We recommend that you specify the same settings. The following table provides additional description.

    Basic Settings

Parameter	Description
	When DTS checks for conflicting tables in the reverse direction, the tables that have been synchronized to the destination instance are ignored.

Parameter	Description
	We recommend that you select the same objects for the forward and reverse directions. You can also add or delete objects based on your business requirements.
	We recommend that you do not use this feature when you configure the task in the reverse direction. Otherwise, data inconsistency may occur.

16.
17. Wait until the **Success Rate** becomes **100%**. Then, click **Back**.
18. After the second data synchronization task is configured, wait until both tasks are in the Running state. The two-way data synchronization tasks are configured.

11.3. Synchronize data from a self-managed PostgreSQL database to an ApsaraDB RDS for PostgreSQL instance

This topic describes how to synchronize data from a self-managed PostgreSQL database to an ApsaraDB RDS for PostgreSQL instance by using Data Transmission Service (DTS).

Prerequisites

- A self-managed PostgreSQL database and an instance are created. For information about how to create the destination instance, see Create an ApsaraDB RDS for PostgreSQL instance. For information about the supported database versions, see Overview of data synchronization scenarios.

  > **Note**

- The available storage space of the destination instance is larger than the total size of the data in the self-managed PostgreSQL database.

Limits

> **Note** By default, DTS disables FOREIGN KEY constraints for the destination database in a data synchronization task. Therefore, the cascade and delete operations of the source database are not synchronized to the destination database.

SQL operations that can be synchronized

Permissions required for database accounts

Database	Required permissions	References
Self-managed PostgreSQL database	The permissions of the superuser role	CREATE USER and GRANT

Database	Required permissions	References
instance	The permissions of the schema owner	instance: Create an account on an ApsaraDB RDS for PostgreSQL instance

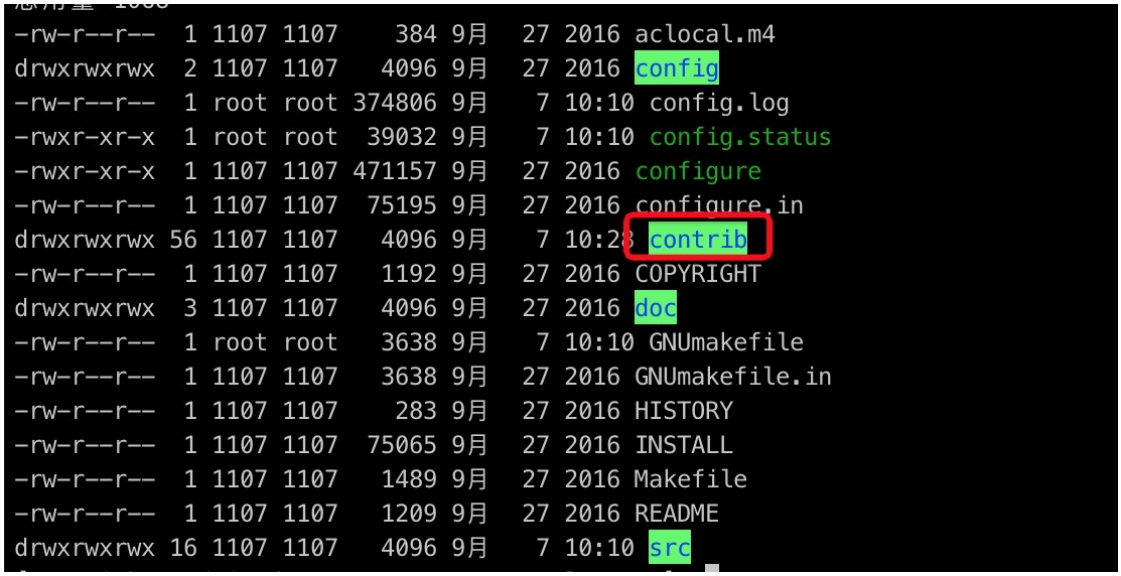
Before you begin

If the version of the self-managed PostgreSQL database is 9.4.8 to 10.0, you must perform the following operations before you configure a data synchronization task.

1. Download the PostgreSQL source code from the official website, and compile and install the source code.
   i. Download the source code from the PostgreSQL official website based on the version of the self-managed PostgreSQL database.
   ii. Run the `./configure`, `make`, and `make install` commands in sequence to configure, compile, and install the source code.

   > **Notice**
   > - When you compile and install PostgreSQL, the operating system version of PostgreSQL must be consistent with the GNU Compiler Collection (GCC) version.
   > - If an error occurs when you run the `./configure` command, you can adjust the command based on the error message. For example, if the error message is `readline library not found. Use --without-readline to disable readline support.`, you can change the command to `./configure --without-readline`.
   > - If you use other methods to install PostgreSQL, you must compile the ali_decoding plug-in in a test environment that has the same OS version and GCC version.

2. Download the ali_decoding plug-in provided by DTS, and compile and install the plug-in.
   i. Download ali_decoding.
   ii. Copy the ali_decoding directory to the contrib directory of PostgreSQL (compiled and installed).



iii. Go to the ali_decoding directory and replace the content of the Makefile file with the following script:

```
# contrib/ali_decoding/Makefile
MODULE_big = ali_decoding
MODULES = ali_decoding
OBJS    = ali_decoding.o
DATA = ali_decoding--0.0.1.sql ali_decoding--unpackaged--0.0.1.sql
EXTENSION = ali_decoding
NAME = ali_decoding
#subdir = contrib/ali_decoding
#top_builddir = ../..
#include $(top_builddir)/src/Makefile.global
#include $(top_srcdir)/contrib/contrib-global.mk
#PG_CONFIG = /usr/pgsql-9.6/bin/pg_config
#pgsql_lib_dir := $(shell $(PG_CONFIG) --libdir)
#PGXS := $(shell $(PG_CONFIG) --pgxs)
#include $(PGXS)
# Run the following commands to install the source code.
ifdef USE_PGXS
PG_CONFIG = pg_config
PGXS := $(shell $(PG_CONFIG) --pgxs)
include $(PGXS)
else
subdir = contrib/ali_decoding
top_builddir = ../..
include $(top_builddir)/src/Makefile.global
include $(top_srcdir)/contrib/contrib-global.mk
endif
```

iv. Go to the ali_decoding directory, run the `make` and `make install` commands in sequence to compile ali_decoding and obtain the files required to install ali_decoding.

v. Copy the following files to the specified location.

```
/usr/bin/install -c -m 755  ali_decoding.so '/usr/local/pgsql/lib/ali_decoding.so'
/usr/bin/install -c -m 644 ./ali_decoding.control '/usr/local/pgsql/share/extension/'
/usr/bin/install -c -m 644 ./ali_decoding--0.0.1.sql ./ali_decoding--unpackaged--0.0.1.sql  '/usr/local/pgsql/share/extension/'
/usr/bin/install -c -m 755  ali_decoding.so '/usr/local/pgsql/lib/'
```

3. Create a database and schema in the destination instance based on the database and schema information of the objects to be synchronized. The schema name of the source and destination databases must be the same. For more information, see Create a database on an ApsaraDB RDS for PostgreSQL instance and Appendix: User and schema management.

If the version of the self-managed PostgreSQL database is 10.1 to 13, you must perform the following operations before you configure a data synchronization task.

1. Log on to the server where the self-managed PostgreSQL database resides.
2. Set the value of the `wal_level` parameter in the *postgresql.conf* configuration file to `logical`.

```
# - Settings -

wal_level = logical                     # minimal, replica, or logical
                                        # (change requires restart)
```

3. Add the CIDR blocks of DTS servers to the *pg_hba.conf* configuration file of the self-managed PostgreSQL database. Add only the CIDR blocks of the DTS servers that reside in the same region as the destination database. For more information, see Add the CIDR blocks of DTS servers to the security

settings of on-premises databases.

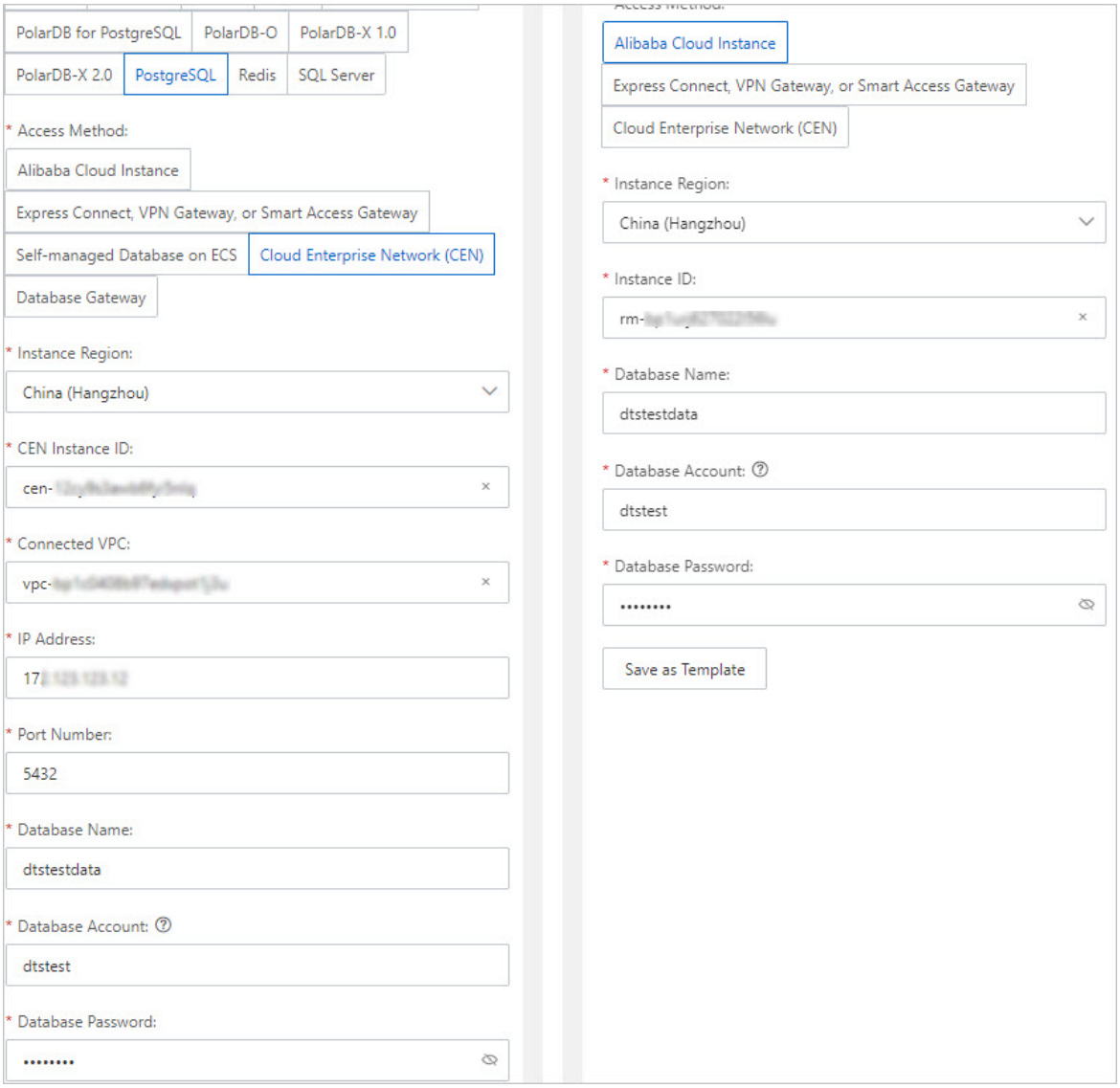
> **Note** For more information, see The pg_hba.conf File. Skip this step if you have set the IP address in the pg_hba.conf file to `0.0.0.0/0`.

```
# "local" is for Unix domain socket connections only
local   all             all                                     peer
# IPv4 local connections:
host    all             all             0.0.0.0/0               trust
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

4. Create a database and schema in the destination instance based on the database and schema information of the objects to be synchronized. The schema name of the source and destination databases must be the same. For more information, see Create a database on an ApsaraDB RDS for PostgreSQL instance and Appendix: User and schema management.

Procedure

1.
2.
3. > **Warning**

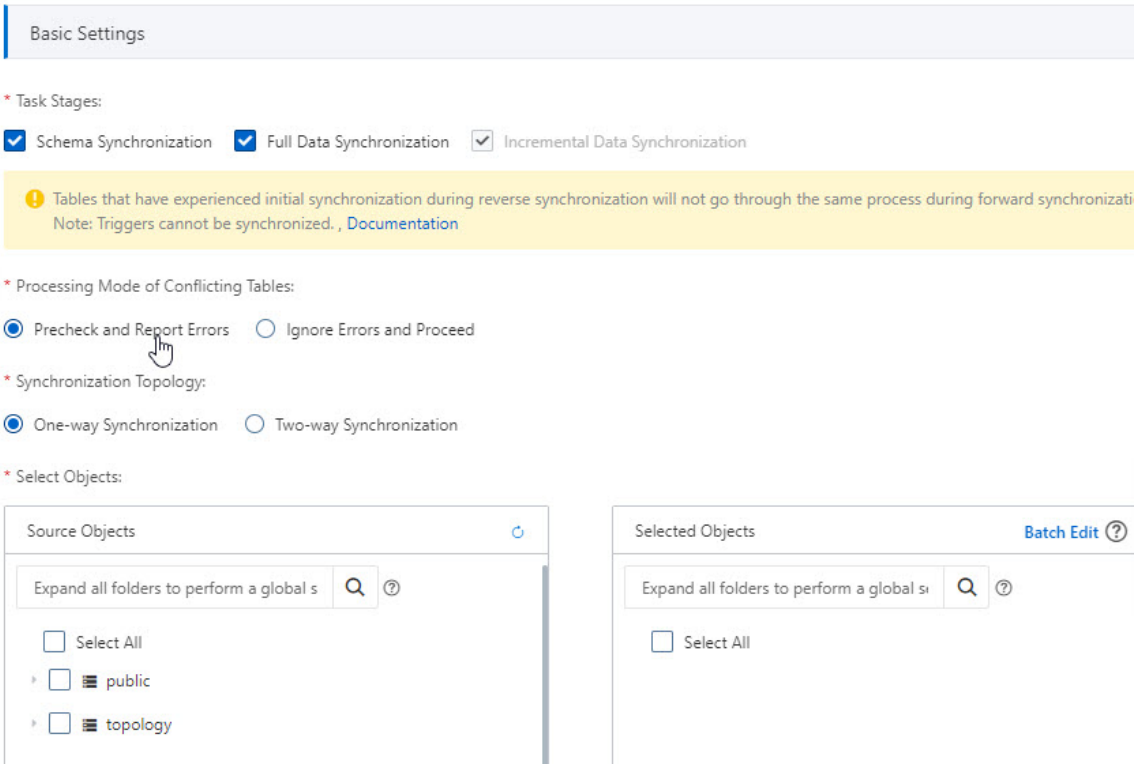


Section	Parameter	Description
N/A		
		Select **PostgreSQL**.
		Select **Cloud Enterprise Network (CEN)**.
		Select the region where the self-managed PostgreSQL database resides.
	CEN Instance ID	Select the ID of the CEN instance that hosts the self-managed PostgreSQL database.
	Connected VPC	Select the virtual private cloud (VPC) that is connected to the self-managed PostgreSQL database.
	IP Address	Enter the server IP address of the self-managed PostgreSQL database.

Section	Parameter	Description
	Port Number	Enter the service port number of the self-managed PostgreSQL database. The default port number is **5432**.
	Database Name	Enter the name of the self-managed PostgreSQL database.
		Enter the account that is used to log on to the self-managed PostgreSQL database. For information about the permissions that are required for the account, see Permissions required for database accounts.
		Select **PostgreSQL**.
		Select **Alibaba Cloud Instance**.
		Select the region in which the destination instance resides.
	Instance ID	Select the ID of the destination instance.
	Database Name	Enter the name of the destination database in the destination instance.
		Enter the database account of the destination instance. For information about the permissions that are required for the account, see Permissions required for database accounts.

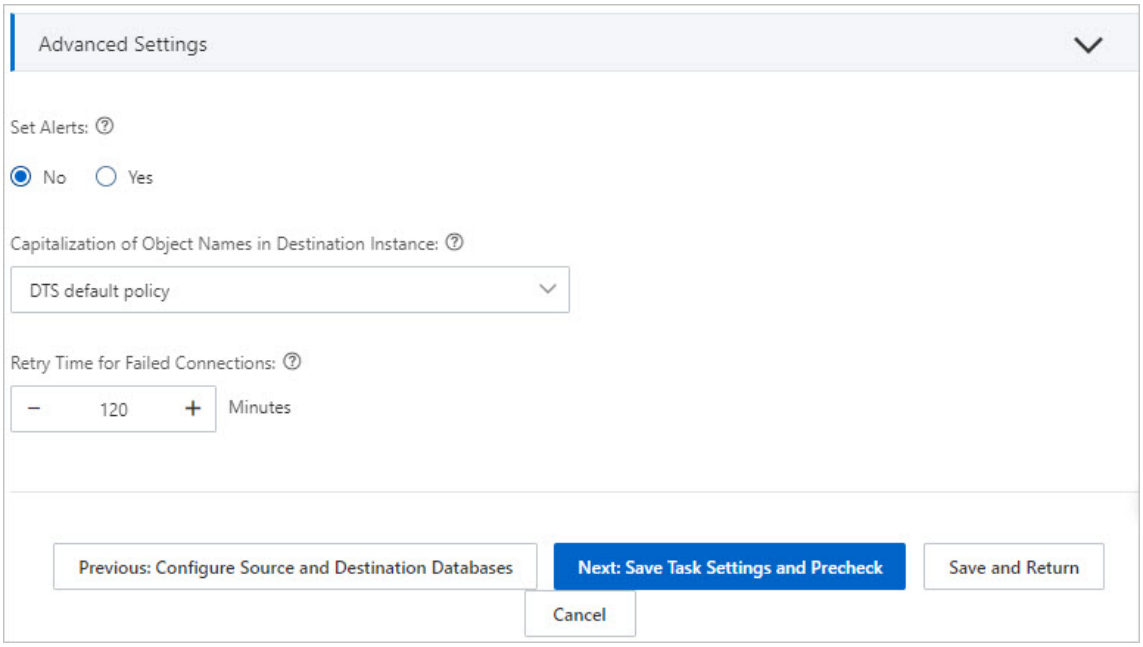
4.

5. ○ **Basic Settings**



Parameter	Description
Synchronization Topology	Select **One-way Synchronization**.
	Note
	In the **Selected Objects** section, right-click an object. In the dialog box that appears, select the SQL operations that you want to synchronize. For more information, see SQL operations that can be synchronized.

- **Advanced Settings**



Parameter	Description

6.
7.
8.
9.
10.

11.4. Synchronize data from an ApsaraDB RDS for PostgreSQL instance to an ApsaraDB RDS for MySQL instance

This topic describes how to synchronize data from an instance to an instance by using Data Transmission Service (DTS).

Prerequisites

- An instance is created. It is the source instance. For more information, see Create an ApsaraDB RDS for PostgreSQL instance.
- An instance is created. It is the destination instance. For more information, see Create an ApsaraDB RDS for MySQL instance.

- The available storage space of the destination instance is larger than the total size of the data in the source instance.

Limits

> **Note** By default, DTS disables FOREIGN KEY constraints for the destination database in a data synchronization task. Therefore, the cascade and delete operations of the source database are not synchronized to the destination database.

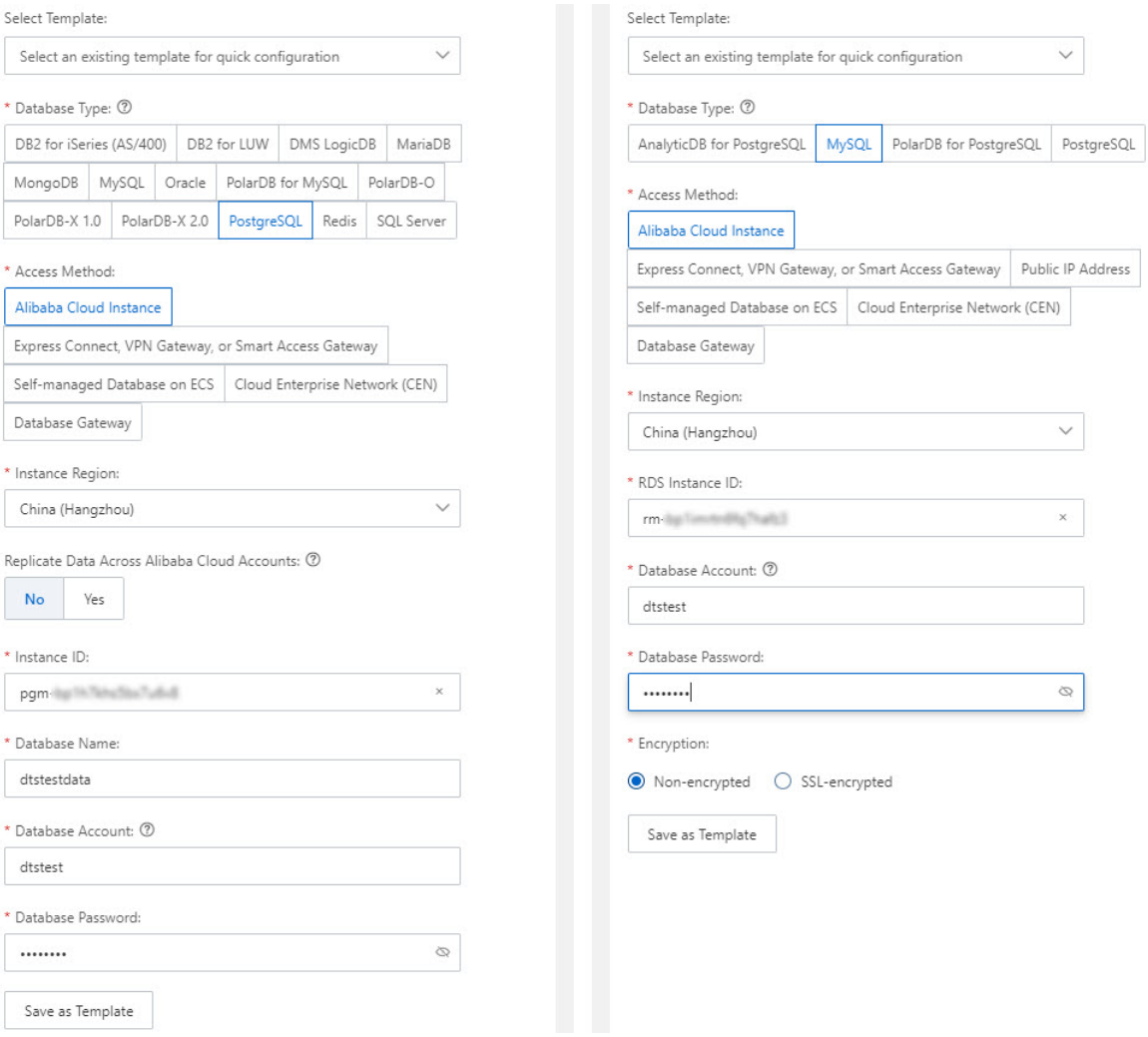
Supported synchronization topologies

- One-way one-to-one synchronization
- One-way one-to-many synchronization
- One-way cascade synchronization
- One-way many-to-one synchronization

For more information about synchronization topologies, see Synchronization topologies.

Procedure

1.
2.
3.
4.
5. **Warning**



Section	Parameter	Description
N/A		
		Select **PostgreSQL**.
		Select **Alibaba Cloud Instance**.
		Select the region in which the source instance resides.
	Instance ID	Select the ID of the source instance.
	Database Name	Enter the name of the source database in the instance.

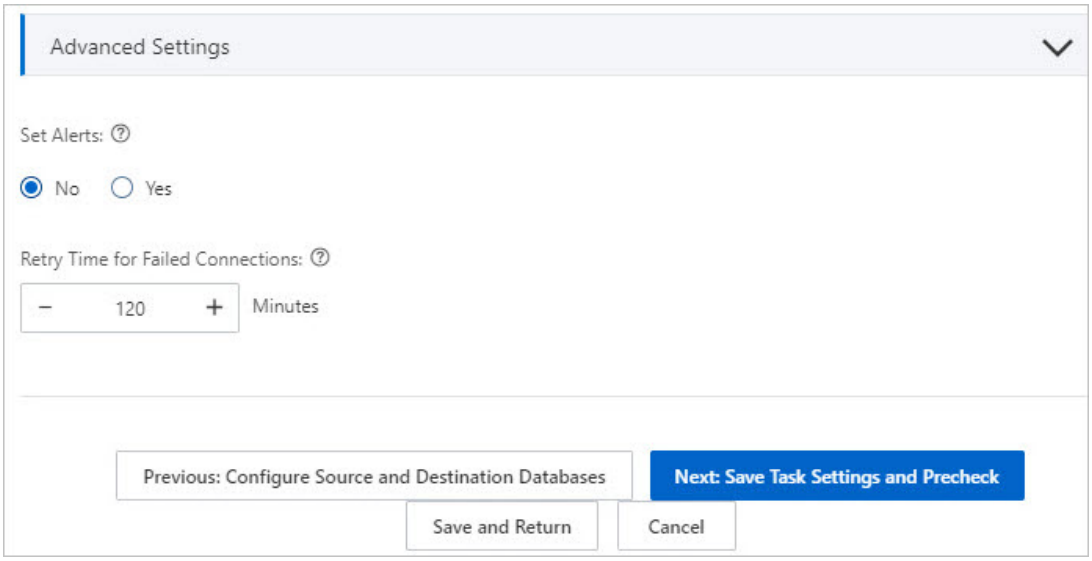
Section	Parameter	Description
		Enter the privileged account of the instance. The account must be the owner of the database. For more information about how to create an account on an ApsaraDB RDS for PostgreSQL instance and grant permissions to this account, see Create an account on an ApsaraDB RDS for PostgreSQL instance and Create a database on an ApsaraDB RDS for PostgreSQL instance. **Note** If the version of the source ApsaraDB RDS for PostgreSQL instance is 9.4 and you synchronize only DML operations, the database account must have the REPLICATION permission.
		Select **MySQL**.
		Select **Alibaba Cloud Instance**.
		Select the region where the destination instance resides.
	RDS Instance ID	Select the ID of the destination instance.
		Enter the database account of the destination instance. The account must have the read and write permissions on the destination database.

6.

7. ○ **Basic Settings**

Parameter	Description
	Note
	In the **Selected Objects** section, right-click an object. In the dialog box that appears, select the DML operations that you want to synchronize.

○ **Advanced Settings**



Parameter	Description

8.
9.
10.
11.
12.

11.5. Synchronize data from an ApsaraDB RDS for PostgreSQL instance to an AnalyticDB for PostgreSQL instance

This topic describes how to synchronize data from an ApsaraDB RDS for PostgreSQL instance to an instance by using Data Transmission Service (DTS). The data synchronization feature provided by DTS allows you to transfer and analyze data with ease.

Prerequisites

- The tables to be synchronized from the ApsaraDB RDS for PostgreSQL instance contain primary keys.
- The destination instance is created. For more information, see Create an AnalyticDB for PostgreSQL instance.

Precautions

- A single data synchronization task can synchronize data from only one database. To synchronize data from multiple databases, you must create a data synchronization task for each database.

- During data synchronization, new tables that are created in the source database can also be synchronized. However, to ensure data consistency, you must execute the following statement on the new tables before they can be synchronized:

  ```
  ALTER TABLE schema.table REPLICA IDENTITY FULL;
  ```

- To ensure that the data synchronization task runs as expected, you can perform primary/secondary switchover only on an ApsaraDB RDS for PostgreSQL instance V11. In this case, you must set the `rds_failover_slot_mode` parameter to `sync`. For more information, see Logical Replication Slot Failover.

  > **Warning** If you perform primary/secondary switchover on a self-managed PostgreSQL database or an ApsaraDB RDS for PostgreSQL instance of other versions, the data synchronization task stops.

Limits

- Initial schema synchronization is not supported. DTS does not synchronize the schemas of the required objects from the source database to the destination database.
- You can select only tables as the objects to be synchronized.
- You cannot synchronize the following types of data: BIT, VARBIT, GEOMETRY, ARRAY, UUID, TSQUERY, TSVECTOR, and TXID_SNAPSHOT.
- If you perform a DDL operation on an object to be synchronized in the source database during data synchronization, you must perform the operation in the destination database. Then, you must restart the data synchronization task.

SQL operations that can be synchronized

INSERT, UPDATE, and DELETE

Before you begin

1. Change the value of the `wal_level` parameter for the source RDS instance.

   > **Warning** After you change the value of the `wal_level` parameter, you must restart the instance to apply the change. We recommend that you evaluate the impact on your business and change the parameter setting during off-peak hours.

   i. Log on to the ApsaraDB RDS console.
   ii. In the top navigation bar, select the region where the RDS instance resides.
   iii. Find the RDS instance and click its ID.
   iv. In the left-side navigation pane, click **Parameters**.
   v. On the **Parameters** page, find the `wal_level` parameter and change the parameter value to `logical`.

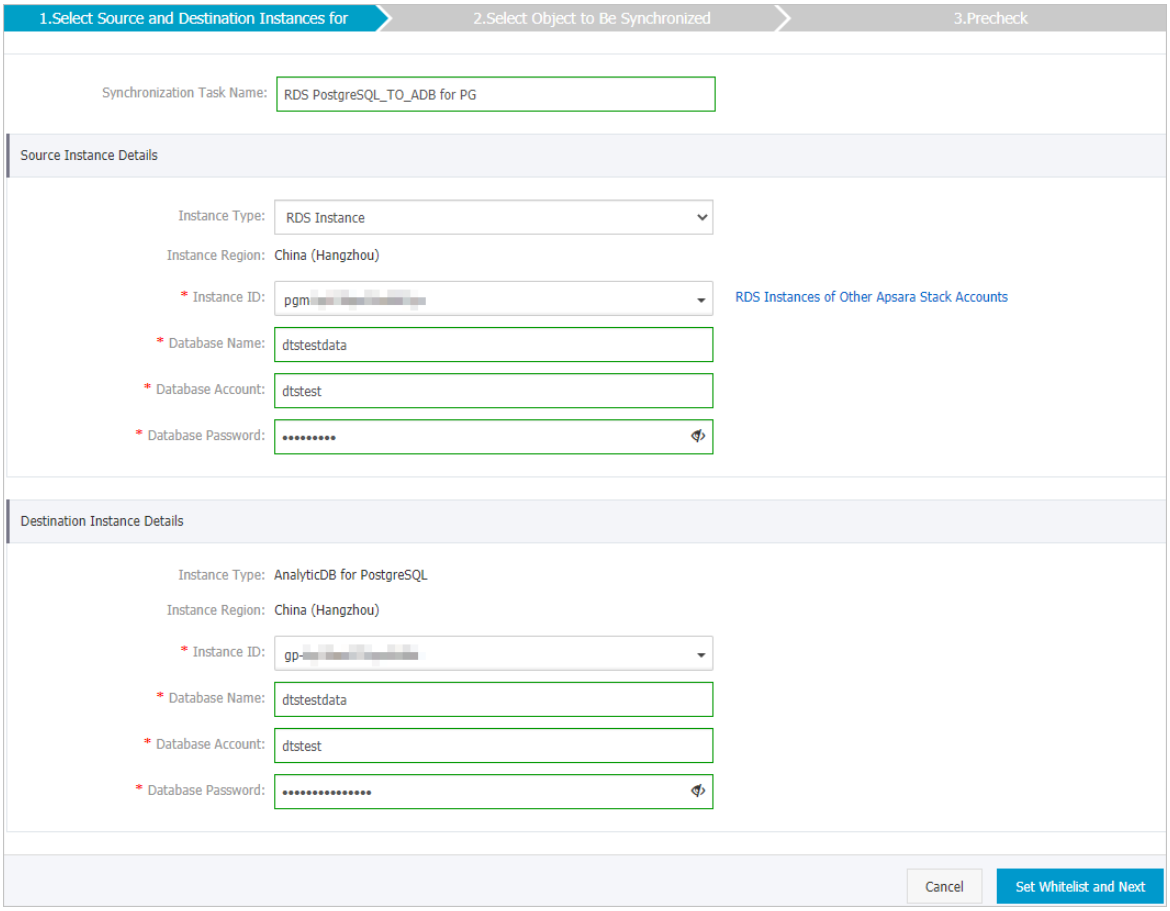
2. Create a database, schema, and table in the destination instance based on the schema of the objects to be synchronized. For more information, see SQL statements.

Procedure

1. Purchase a data synchronization instance. For more information, see Purchase procedure.

> **Note** On the buy page, set Source Instance to **PostgreSQL**, set Target Instance to **AnalyticDB for PostgreSQL**, and set Synchronization Topology to **One-Way Synchronization**.

2. Log on to the DTS console.
3. In the left-side navigation pane, click **Data Synchronization**.
4. At the top of the **Synchronization Tasks** page, select the region where the destination instance resides.
5. Find the data synchronization instance and click **Configure Synchronization Channel** in the Actions column.
6. Configure the source and destination instances.



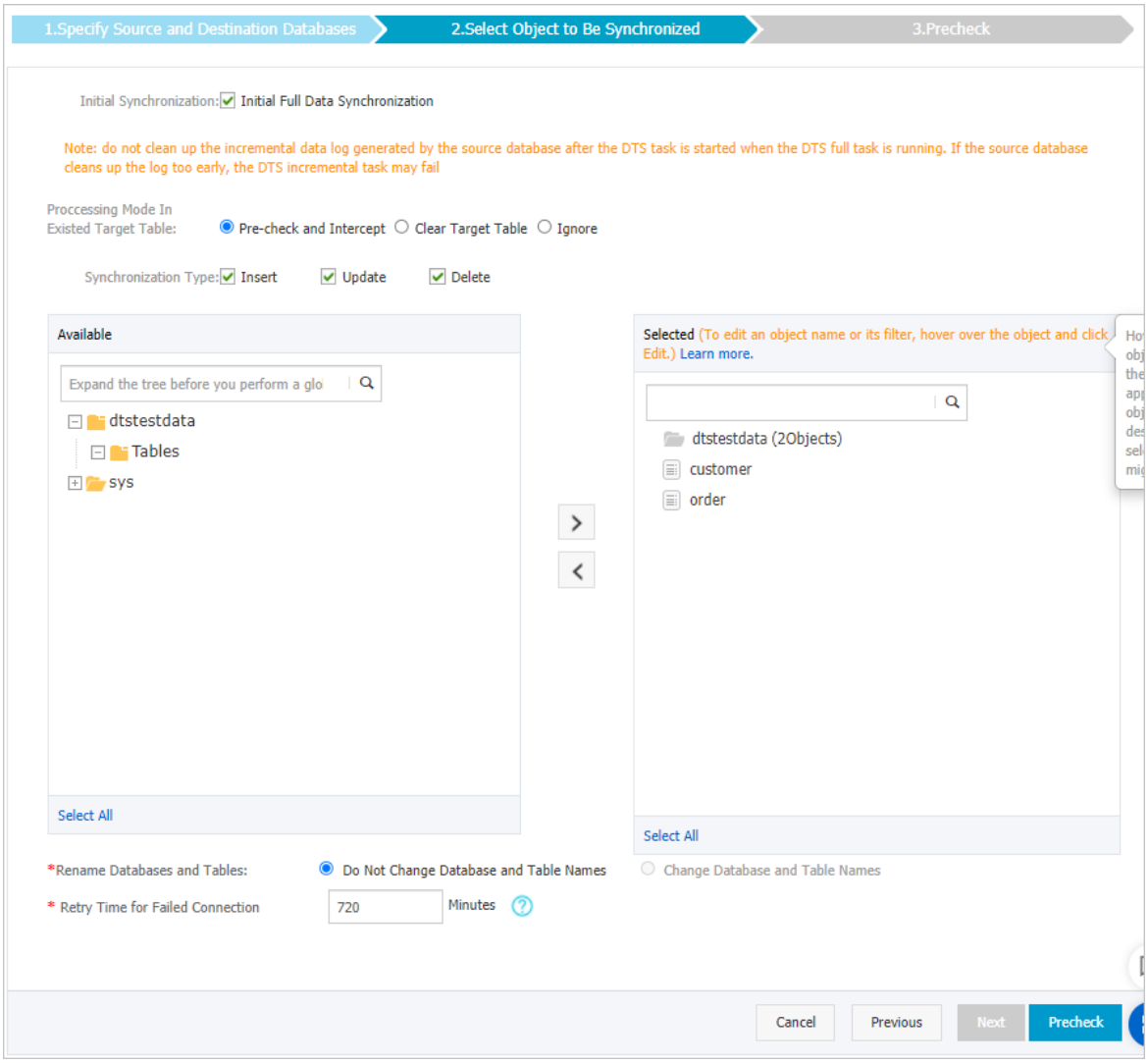
Section	Parameter	Description
N/A	Synchronization Task Name	DTS automatically generates a task name. We recommend that you specify an informative name for easy identification. You do not need to use a unique task name.
	Instance Type	Select **RDS Instance**.
	Instance Region	The source region that you selected on the buy page. You cannot change the value of this parameter.
	Instance ID	Select the ID of the ApsaraDB RDS for PostgreSQL instance.

Section	Parameter	Description
Source Instance Details	Database Name	Enter the name of the source database.
	Database Account	Enter the database account of the ApsaraDB RDS for PostgreSQL instance. The database account must have the SUPERUSER permission. **Note** If the source database runs on an ApsaraDB RDS for PostgreSQL instance V9.4 and you synchronize only DML operations, the database account must have the REPLICATION permission.
	Database Password	Enter the password of the database account.
Destination Instance Details	Instance Type	This parameter is set to **AnalyticDB for PostgreSQL** and cannot be changed.
	Instance Region	The destination region that you selected on the buy page. You cannot change the value of this parameter.
	Instance ID	Select the ID of the instance.
	Database Name	Enter the name of the destination database. **Note** The database must exist in the instance. Otherwise, you must create a database.
	Database Account	Enter the **initial account** of the instance. For more information, see Create a database account. **Note** You can also enter an account that has the RDS_SUPERUSER permission. For more information, see Manage users and permissions.
	Database Password	Enter the password of the database account.

7. In the lower-right corner of the page, click **Set Whitelist and Next**.

   Note DTS adds the CIDR blocks of DTS servers to the whitelists of the ApsaraDB RDS for PostgreSQ and instances. This ensures that DTS servers can connect to the source and destination instances.

8. Select the synchronization policy and the objects to be synchronized.



Setting	Parameter	Description
	Initial Synchronization	**Initial Full Data Synchronization** is selected by default. After the precheck, DTS synchronizes historical data of the required objects from the source instance to the destination instance. The data is the basis for subsequent incremental synchronization.

Setting	Parameter	Description
Select the synchronization policy	Select the processing mode of conflicting tables	◦ **Clear Target Table** Skips the **Schema Name Conflict** item during the precheck. Clears the data in the destination table before initial full data synchronization. If you want to synchronize your business data after testing the data synchronization task, you can select this mode. ◦ **Ignore** Skips the **Schema Name Conflict** item during the precheck. Adds data to the existing data during initial full data synchronization. If you want to synchronize data from multiple tables to one table, you can select this mode.
	Synchronization Type	Select the types of operations that you want to synchronize based on your business requirements. **Note** The **Alter Table** operation is not supported. ◦ **Insert** ◦ **Update** ◦ **Delete** ◦ **AlterTable**
Select the objects to be synchronized	N/A	Select one or more tables from the **Available** section and click the › icon to move the tables to the **Selected** section. **Note** ◦ You can select only tables as the objects to be synchronized. ◦ You can use the object name mapping feature to rename the columns that are synchronized to the destination database. For more information, see Rename an object to be synchronized.
Rename Databases and Tables	N/A	You can use the object name mapping feature to rename the objects that are synchronized to the destination instance. For more information, see Object name mapping.

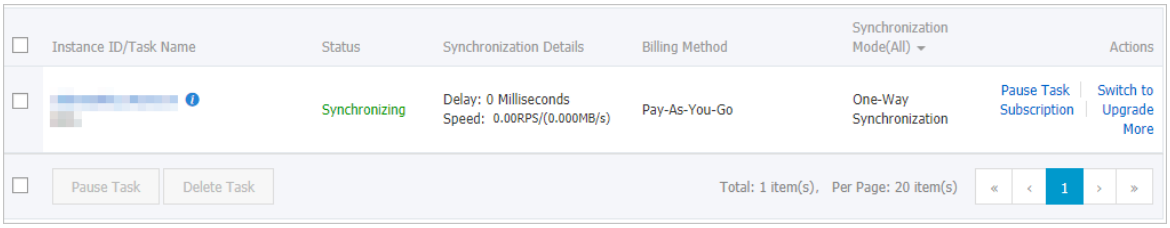
Setting	Parameter	Description
Retry Time for Failed Connections	N/A	By default, if DTS fails to connect to the source or destination database, DTS retries within the next 720 minutes (12 hours). You can specify the retry time based on your needs. If DTS reconnects to the source and destination databases within the specified time, DTS resumes the data synchronization task. Otherwise, the data synchronization task fails. **Note** When DTS retries a connection, you are charged for the DTS instance. We recommend that you specify the retry time based on your business needs. You can also release the DTS instance at your earliest opportunity after the source and destination instances are released.

9. In the lower-right corner of the page, click **Precheck**.

   Note
   - Before you can start the data synchronization task, DTS performs a precheck. You can start the data synchronization task only after the task passes the precheck.
   - If the task fails to pass the precheck, you can click the icon next to each failed item to view details.
     - After you troubleshoot the issues based on the causes, you can run a precheck again.
     - If you do not need to troubleshoot the issues, you can ignore failed items and run a precheck again.

10. Close the **Precheck** dialog box after the following message is displayed: **The precheck is passed.** Then, the data synchronization task starts.
11. Wait until initial synchronization is completed and the data synchronization task enters the **Synchronizing** state.

    You can view the state of the data synchronization task on the **Synchronization Tasks** page.



12.Change tracking

12.1. Use DTS to track data changes from ApsaraDB RDS for PostgreSQL instances

13.Version upgrade

13.1. Upgrade an ApsaraDB RDS for PostgreSQL instance from Basic Edition to High-availability Edition

This topic describes how to upgrade an ApsaraDB RDS for PostgreSQL instance from Basic Edition to High-availability Edition. The upgrade increases the reliability of your database service.

Prerequisites

The RDS instance must meet the following requirements:

- The major engine version of the RDS instance is PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, or PostgreSQL 14.
- The RDS instance runs RDS Basic Edition.

  > **Note** You can log on to the ApsaraDB RDS console and go to the **Basic Information** page of your RDS instance to view the RDS edition of your RDS instance.

- The RDS instance does not use a new general-purpose instance type.

  > **Note** The new general-purpose instance types provide better scalability and performance and reduce the time to create an RDS instance or change the specifications of an RDS instance. The new general-purpose instance types do not support upgrades from Basic Edition to High-availability Edition. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types.

Context

High-availability Edition is a widely used edition. If you use this edition, your database system consists of a primary RDS instance and a secondary RDS instance. These instances work in a high-availability architecture. High-availability Edition is suitable for more than 80% of business scenarios, such as Internet, IoT, online retail, logistics, and gaming.

For more information, see High-availability Edition.

Billing

For more information about the upgrade fees, see Specification change fees.

Impacts

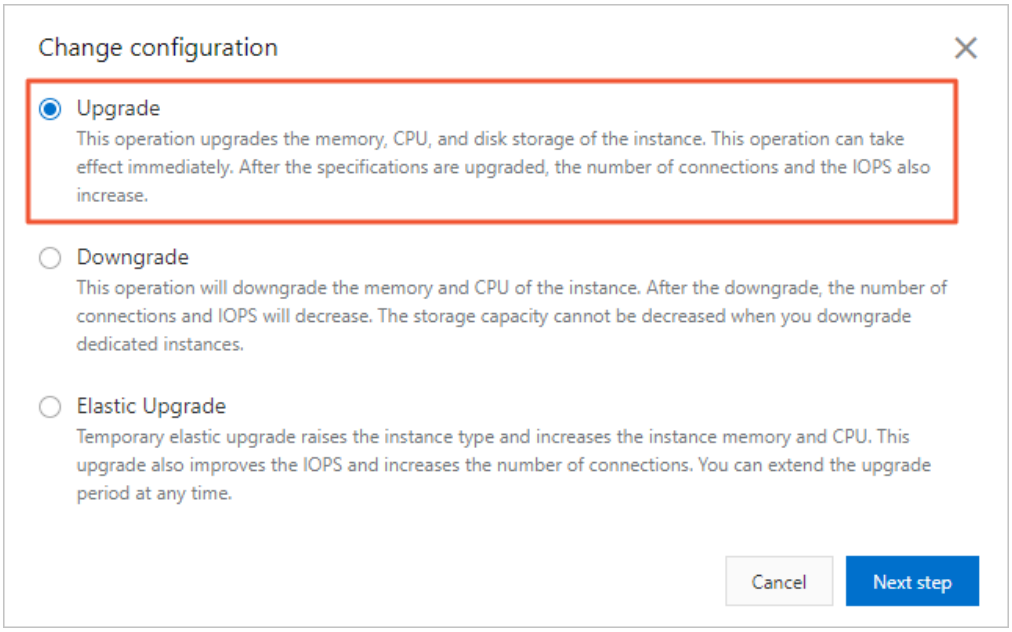
- The upgrade may cause a migration of underlying data. After the migration is complete, ApsaraDB RDS switches your workloads over to High-availability Edition at the specified switching time. During the switchover, a transient connection of about 30 seconds occurs. Make sure that your application is

configured to automatically reconnect to your RDS instance.

- After the upgrade is complete, you cannot downgrade the RDS edition of your RDS instance to RDS Basic Edition.

Procedure

1.
2. On the **Basic Information** page, click **Change Specifications**.
3. In the dialog box that appears, select **Upgrade** and click **Next step**. This step is required only for subscription RDS instances.



4. Configure the following parameters.

Parameter	Description
Edition	Select **High-availability**.
Instance Type	Select an instance type. Each instance type supports a specified number of cores, memory capacity, maximum number of connections, and maximum IOPS. For more information, see Primary ApsaraDB RDS instance types.
Capacity	Specify the storage capacity of your RDS instance. The storage capacity can only be increased but cannot be decreased.
Switching Time	Specify the time when you want to switch your workloads over to High-availability Edition. Valid values: ○ Switch Immediately After Data Migration ○ Switch Within Maintenance Window

5. Read and select Terms of Service, click **Pay Now**, and then complete the payment.

Related operations

Operation	Description
Change the specifications of an ApsaraDB RDS instance	Changes the specifications of an ApsaraDB RDS instance.

13.2. Upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance. For example, you can upgrade the major engine version from PostgreSQL 10 to PostgreSQL 11.

The PostgreSQL community releases a new major engine version every year and provides five years of support for each version. Each new version includes improvements in terms of functionality and performance compared with previous versions. The PostgreSQL community does not provide support for versions that were released five years ago or earlier. Therefore, the performance risks and security risks of these versions increase.

> **Notice** During an upgrade, ApsaraDB RDS retains the original RDS instance and creates an RDS instance that runs the new major engine version. You start to be **charged** for the new RDS instance based on the **pay-as-you-go** billing method after the instance is created. The new RDS instance does not carry over the discounts that are offered for the original RDS instance. You can decide whether to upgrade the major engine version of the original RDS instance based on your business requirements.

ApsaraDB RDS for PostgreSQL provides a major engine version upgrade feature. This feature can increase the performance and functionality of your database service and mitigate the risks that an upgrade may cause. To upgrade the major engine version of the original RDS instance by using this feature, you must perform the following steps:

1. Perform an upgrade check.

   Check whether the original RDS instance supports major engine version upgrades. Then, view the check report that is generated. If the check result in the upgrade check report is Fail, you cannot upgrade the major engine version.

2. Upgrade the major engine version.
   - Select the **No cutting** configuration method. ApsaraDB RDS creates an RDS instance that runs the new major engine version to test whether the new major engine version is compatible with your workloads.
   - After the new major engine version passes the compatibility test, select the **Cutover** configuration method. ApsaraDB RDS creates an RDS instance that runs the new major version. After the data of the original RDS instance is migrated to the new RDS instance, ApsaraDB RDS switches your workloads over to the new RDS instance.

Precautions

- The original RDS instance must meet the following requirements:
  - The original RDS instance runs PostgreSQL 13, PostgreSQL 12, PostgreSQL 11, PostgreSQL 10, or PostgreSQL 9.4.

- The original RDS instance runs RDS High-availability Edition or RDS Basic Edition. For more information, see High-availability Edition and RDS Basic Edition.
- The original RDS instance resides in a virtual private cloud (VPC).

  If the original RDS instance resides in the classic network, you must change the network type of the original RDS instance to VPC before you perform an upgrade. When you perform the network type change, do not select **Reserve original classic network endpoint**. For more information about how to view or change the network type of an RDS instance, see Change the network type of an ApsaraDB RDS for PostgreSQL instance.

  > **Note** If you select **Reserve original classic network endpoint**, you must wait until the retention period of the endpoint ends before you can perform the upgrade task.

- The original RDS instance is a primary instance. The original RDS instance cannot be a read-only instance and cannot be created in a dedicated cluster. For more information, see Overview of read-only ApsaraDB RDS for PostgreSQL instances and What is ApsaraDB for MyBase?
- The ID of the original RDS instance does not start with `pg-cn`.

- An upgrade has the following impacts:
  - If you select the Cutover configuration method, ApsaraDB RDS needs to switch your workloads over to a new RDS instance during the upgrade process. The original RDS instance processes only read requests and a transient connection that lasts a few minutes occurs during the switchover. Therefore, we recommend that you perform an upgrade during off-peak hours. If you select the No cutting configuration method, the original RDS instance is not affected.

    > **Note**
    > - The duration of the transient connection varies based on the amount of data and the intervals at which the DNS cache is refreshed. You can switch to a different vSwitch. Then, you can estimate the cache refresh interval of your database client based on the duration of the transient connection. For more information, see Switch an ApsaraDB RDS for PostgreSQL instance to a different vSwitch.
    > - The time that is required for an upgrade varies based on the amount of data. However, the amount of data does not affect the time that is required for a switchover. The time that is required for a switchover varies based on the number of objects that are defined in the original RDS instance. These objects include the defined tables, defined indexes, defined views, user-defined functions, and defined sequences. For example, the time that is required for a switchover when 100 tables with 1 GB of data in total are defined is the same as the time that is required for a switchover when 100 tables with 10 TB of data in total are defined.

  - If the original RDS instance uses a parameter that is not supported by the new major engine version, the parameter is deleted from the new RDS instance. If the value of a parameter in the previous major engine version is not within the value range that is supported in the new major engine version, ApsaraDB RDS sets the parameter to the default value that is specified in the new major engine version.
  - The new RDS instance does not carry over the **name**, tags, alert rules, and backup data of the original RDS instance. For more information, see Add tags to ApsaraDB RDS for PostgreSQL instances, Configure an alert rule on an ApsaraDB RDS for PostgreSQL instance, and Back up an ApsaraDB RDS for PostgreSQL instance.

- If the original RDS instance is the source RDS instance or destination RDS instance of a migration task that is created in the Data Transmission Service (DTS) console, you must re-create the migration task after you perform an upgrade. For more information about how to create a migration task in the DTS console, see What is DTS?
- After an upgrade is complete, the read-only RDS instances and logical replication slots that you created in the original RDS instance remain attached to the original RDS instance rather than the new RDS instance. You must create read-only RDS instances and logical replication slots on the new RDS instance after the upgrade.

- If a read-only RDS instance is attached to the original RDS instance, you must perform the following operations before and after you perform an upgrade:
  i. On your application, change the endpoint of the read-only RDS instance to the endpoint of the original RDS instance.

     > **Note** For service stability purposes, we recommend that you modify the endpoint configuration on your application during off-peak hours.

  ii. Delete the read-only RDS instance.
  iii. Upgrade the major engine version of the original RDS instance. For more information, see Procedure.
  iv. After the upgrade is complete, create a read-only RDS instance on the new RDS instance. For more information, see Create a read-only ApsaraDB RDS for PostgreSQL instance.
  v. On your application, change the endpoint of the original RDS instance to the endpoint of the new read-only RDS instance.

Highlights

- **Cross-version upgrade**: You can upgrade the major engine version of the original RDS instance to a later version. For example, you can upgrade the major engine version from PostgreSQL 10 to PostgreSQL 13.
- **Upgrade trial**: You can use the **No cutting** configuration method to verify the feasibility of an upgrade without interruptions to your workloads on the original RDS instance.
- **Smooth upgrade**:
  - **No application modification**: You can use the **Cutover** configuration method to perform an upgrade. In this case, the new RDS instance is connected by using the same endpoint as the original RDS instance. You do not need to modify the endpoint configuration on your application.
  - **No downtime**: An upgrade does not cause downtime on the original RDS instance. This mitigates the risks of service interruption. However, the original RDS instance processes only read requests during the upgrade process. In addition, ApsaraDB RDS performs the upgrade by using RDS instance cloning. In this case, the original RDS instance is not affected even if the upgrade fails.
  - **Reserved instance configuration**:
    - The new RDS instance carries over the IP address whitelists, parameter settings, and plug-ins of the original RDS instance. However, this does not apply to the plug-ins or parameters that are not supported in the new major engine version.
    - If the original RDS instance supports fully encrypted databases, the new RDS instance also supports fully encrypted databases and carries over the key that is used on the original RDS instance to encrypt data. For more information, see Create a fully encrypted database on an ApsaraDB RDS for PostgreSQL instance.

Billing

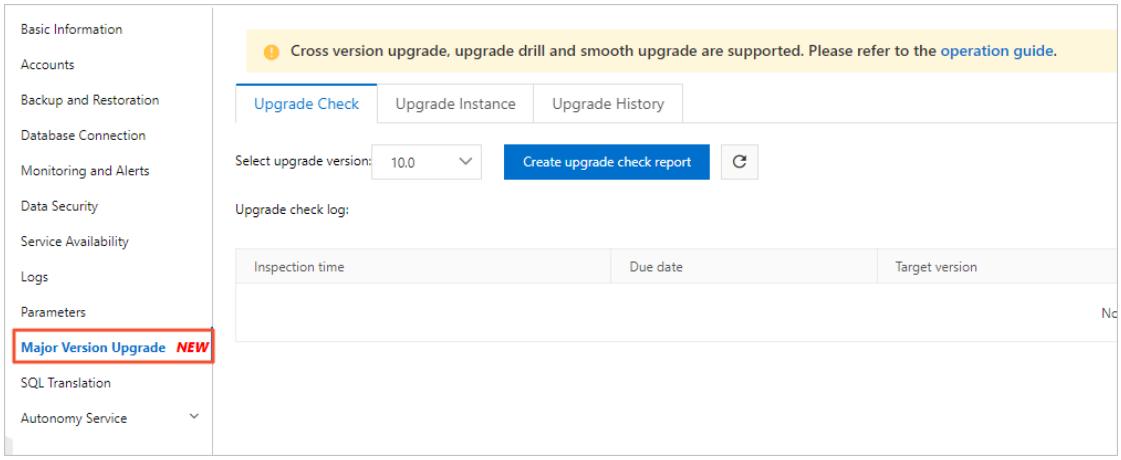
You are charged for the new RDS instance based on the pay-as-you-go billing method. After you verify that your workloads run as expected on the new RDS instance, you can change the billing method of the new RDS instance to subscription. Then, you can release the original RDS instance. For more information, see Switch an ApsaraDB RDS for PostgreSQL instance from pay-as-you-go to subscription and Release or unsubscribe from an ApsaraDB RDS for PostgreSQL instance.

Procedure

1. If a read-only RDS instance is attached to the original RDS instance, perform the following steps before you perform an upgrade:
   i. On your application, change the endpoint of the read-only RDS instance to the endpoint of the original RDS instance.

      > **Note** For service stability purposes, we recommend that you modify the endpoint configuration on your application during off-peak hours.

   ii. Delete the read-only RDS instance.
2. Go to the **Major Version Upgrade** page.
   i.
   ii. In the left-side navigation pane, click **Major Version Upgrade**.



> **Note**
> - If the **Major Version Upgrade** menu item cannot be found, you can check whether your RDS instance meets all requirements that are described in the "Precautions" section of this topic.
> - The major engine version upgrade feature is available only in the new ApsaraDB RDS console.

3. On the **Upgrade Check** tab, select the new major engine version and click **Create upgrade check report**. This process requires a few minutes.

> **Note**
>
> - Before you perform an upgrade, you must make sure that your RDS instance passes the upgrade check. This helps ensure that the upgrade is successful.
> - If you execute the `CREATE EXTENSION` statement on your RDS instance after the new major engine version passes the upgrade check, you must perform an upgrade check again.

If the upgrade check report indicates that your RDS instance fails the upgrade check, you can click **View Information** in the Report Content column to view the details about the report. For more information, see Introduction to the check report of a major engine version upgrade for an ApsaraDB RDS for PostgreSQL instance.

4. Click the **Upgrade Instance** tab, read the warning that is displayed, select the new major engine version, and then click **Continue to create upgrade instance**.
5. In the message that appears, read the tips and click **OK**.

Parameter	Description
Storage Type	Select the type of storage media that is used for the new RDS instance. The major version upgrade feature is based on SSD snapshots. You can select a storage type based on the following conditions: ○ If the original RDS instance uses standard SSDs, you can select standard SSDs. ○ If the original RDS instance uses ESSDs, you can select ESSDs of PL1, ESSDs of PL2, or ESSDs of PL3. ○ If the original RDS instance uses local SSDs, you can select ESSDs of PL1, ESSDs of PL2, or ESSDs of PL3.
The Available Zone **For Available Zone** **Primary Instance Switch** **Standby Instance Switch**	Specify the zones and vSwitches of the new RDS instance and its secondary RDS instance. The zones that you specify can be different from the zones of the original RDS instance.

Parameter	Description
Cutover configuration	Specify whether ApsaraDB RDS automatically switches your workloads over to the new RDS instance after the data of the original RDS instance is migrated to the new RDS instance. ◦ **No cutting**: ApsaraDB RDS does not automatically switch your workloads over to the new RDS instance. Before you perform an upgrade, we recommend that you set this parameter to false to test whether the new major engine version is compatible with your workloads. Before you perform an upgrade, we recommend that you select the **No cutting** configuration method to perform a compatibility test. If the new major engine version passes the compatibility test, you can release the new RDS instance. Then, you can select the **Cutover** configuration method to perform an upgrade. For more information, see Release or unsubscribe from an ApsaraDB RDS for PostgreSQL instance and the "Procedure" section of this topic. **Note** ■ The data migration does not interrupt your workloads on the original instance. ■ If you select the **No cutting** configuration method, you must update the endpoint configuration on your application after the data is migrated to the new RDS instance. This update requires that you replace the endpoint of the original RDS instance with the endpoint of the new RDS instance. For more information about how to view the endpoint of an RDS instance, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance. ◦ **Cutover**: ApsaraDB RDS automatically switches your workloads over to the new RDS instance. After the switchover is complete, you do not need to update the endpoint configuration on your application. Your application automatically connects to the new RDS instance. This configuration method is used to perform an upgrade after you verify that the new major engine version is compatible with your workloads. **Note** ■ After the switchover is complete, you cannot roll your workloads back to the original RDS instance. Proceed with caution. ■ During the switchover, the original RDS instance processes only read requests. You must perform the switchover during off-peak hours. ■ If read-only RDS instances are attached to the original RDS instance, you cannot select the **Cutover** configuration method. In this case, you can upgrade the major engine version of the original RDS instance only by using the **No cutting** configuration method. In addition, the read-only RDS instances that are attached to the original RDS instance cannot be cloned. After the upgrade is complete, you must create read-only RDS instances that run the new major engine version for the new RDS instance. For more information, see Create a read-only ApsaraDB RDS for PostgreSQL instance.

Parameter	Description
Cutover time	Specify the time when ApsaraDB RDS switches your workloads over to the new RDS instance after the data is migrated to the new RDS instance. ○ **Immediately**: After the data is migrated to the new RDS instance, ApsaraDB RDS immediately switches your workloads over to the new RDS instance. ○ **Instance operation and maintenance time**: ApsaraDB RDS switches your workloads over to the new RDS instance during the planned maintenance window. For more information, see Set the maintenance window of an ApsaraDB RDS for PostgreSQL instance. **Note** This parameter is available only when you set the **Cutover configuration** parameter to **Cutover**.
Statistical information collection mode	The time at which ApsaraDB RDS collects the statistics of the new RDS instance. ○ **Collection before cutting**: This option ensures the stability of your database service. If the original RDS instance contains a large amount of data, the upgrade may require a long period of time. ○ **Collection after cutting**: This option accelerates the upgrade process. If you access tables for which no statistics are generated, the query plans that you specify may be inaccurately executed. In addition, your database service may be unavailable during peak hours. **Note** If you select the **No cutting** configuration method, the **Collection before cutting** option specifies that ApsaraDB RDS collects the statistics of the new RDS instance before the new RDS instance starts to process read and write requests. The **Collection after cutting** option specifies that ApsaraDB RDS collects the statistics of the new RDS instance after the new RDS instance starts to process read and write requests.
Capacity	Specify the storage capacity of the new RDS instance.
Instance Type	Specify the instance type of the new RDS instance. For more information about the instance types that are supported, see Primary ApsaraDB RDS instance types.

6. Click **Create now**.

   The status of the original RDS instance changes to **EngineVersionUpgrading**. In addition, you can find a new RDS instance on the Instances page. The new RDS instance is in the **Creating** state. When the new RDS instance is created and the upgrade is complete, the statuses of the original RDS instance and new RDS instance change to **Running**. The time that is required for the upgrade varies based on the amount of data. You may need to wait a few minutes for the upgrade to finish.

   Note After an upgrade task is created, you cannot modify or delete the task. If you want to delete an upgrade task, you must submit a .

7. If a read-only RDS instance is attached to the original RDS instance and you deleted the read-only RDS instance in Step 1, perform the following steps after the upgrade:

   i. Create a read-only RDS instance for the new RDS instance. For more information, see Create a read-only ApsaraDB RDS for PostgreSQL instance.

ii. On your application, change the endpoint of the original RDS instance to the endpoint of the new read-only RDS instance. For more information, see Step 1.

What to do next

1. After you verify that your workloads run as expected on the new RDS instance, change the billing method of the new RDS instance to subscription. For more information, see Switch an ApsaraDB RDS for PostgreSQL instance from pay-as-you-go to subscription.
2. Release the original RDS instance. For more information, see Release or unsubscribe from an ApsaraDB RDS for PostgreSQL instance.
3. Create read-only RDS instances for the new RDS instance based on your business requirements. The new RDS instance does not carry over the read-only RDS instances of the original RDS instance. For more information, see Create a read-only ApsaraDB RDS for PostgreSQL instance.

Related operations

Operation	Description
UpgradeDBInstanceMajorVersionPrecheck	Checks the compatibility between a new major engine version and an ApsaraDB RDS for PostgreSQL instance before an upgrade
DescribeUpgradeMajorVersionPrecheckTask	Queries the check report for a major engine version upgrade to an ApsaraDB RDS for PostgreSQL instance
UpgradeDBInstanceMajorVersion	Upgrades the major engine version of an ApsaraDB RDS for PostgreSQL instance
DescribeUpgradeMajorVersionTask	Queries the tasks that are created to upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance

What's next

13.3. Introduction to the check report of a major engine version upgrade for an ApsaraDB RDS for PostgreSQL instance

This topic describes the check report of a major engine version upgrade for an ApsaraDB RDS for PostgreSQL instance. This topic also describes the common errors that are included in the report and the solutions to these errors.

If the report indicates that your RDS instance fails the upgrade check, you can log on to the ApsaraDB RDS console, go to the Major Version Upgrade page, and then click **View Information** in the Report Content column to view the details about the failure. The report contains the following check items:

- user_check_report
- pg_upgrade_internal.log

- pg_upgrade_server.log
- loadable_libraries.txt
- tables_with_oids.txt

> **Note** For more information about major engine version upgrades, see Upgrade the major engine version of an ApsaraDB RDS for PostgreSQL instance.

user_check_report

This item is used to check whether a superuser account is created in the background or an invalid encryption method is configured for a standard account.

Error message	Possible cause	Solution
invalid superuser: ["user_01"]	A superuser account is found. If you use this account to connect to your RDS instance, the read-only attribute that is configured for your RDS instance is invalid. As a result, the data of your RDS instance changes after you perform an upgrade.	To delete the superuser account, you must submit a to contact Alibaba Cloud technical support.
invalid user: ["user_02"]	A standard account is in an abnormal state. You cannot use this account to establish a connection after you perform an upgrade.	Reset the password of the abnormal account.

pg_upgrade_internal.log

This item is used to check whether the new major engine version is compatible with the plug-ins on your RDS instance.

Error message	Possible cause	Solution
A list of problem libraries is in the file: loadable_libraries.txt	Plug-ins that are incompatible with the new major engine version are found in the loadable_libraries.txt file.	Check the plug-ins that are listed in the loadable_libraries.txt file and evaluate whether some plug-ins need to be deleted. If a plug-in needs to be deleted, we recommend that you delete the plug-in before an upgrade. Before you delete a plug-in, make sure that your RDS instance can run as expected without the plug-in. For more information, see Supported plug-ins.

Error message	Possible cause	Solution
A list of tables with the problem is in the file: tables_with_oids.txt	Some tables are created with the `WITH OIDS` clause specified. This clause is not supported in PostgreSQL 12 or later versions.	• Solution 1: Upgrade the major engine version of your RDS instance to PostgreSQL 11. PostgreSQL 11 supports the `WITH OIDS` clause. This solution is recommended. • Solution 2: Check the tables that are listed in the tables_with_oids.txt file and evaluate whether the business code depends on the objects that are specified in the WITH OIDS clause. If the business code does not depend on the specified objects, execute the following statement: `ALTER TABLE {table_name} SET WITHOUT OIDS;`

pg_upgrade_server.log

This item is used to check the types of logs that are enabled for your RDS instance.

loadable_libraries.txt

This item is used to check the libraries that are incompatible with the new major engine version. You can identify incompatible plug-ins based on these libraries.

Error message	Possible cause	Solution
could not load library "$libdir/pgrouting-2.6.2": ERROR: could not access file "$libdir/pgrouting-2.6.2": No such file or directory	The pgrouting plug-in is incompatible with the new major engine version.	Check the plug-ins that are listed in the loadable_libraries.txt file and evaluate whether some plug-ins need to be deleted. If a plug-in needs to be deleted, we recommend that you delete the plug-in before an upgrade. Before you delete a plug-in, make sure that your RDS instance can run as expected without the plug-in. For more information, see Supported plug-ins.
could not load library "$libdir/jsonbx": ERROR: could not access file "$libdir/jsonbx": No such file or directory	Some JSON data types are not supported in PostgreSQL 9.4. To support all JSON data types, you must enable the jsonbx plug-in. PostgreSQL 10 and later versions support all JSON data types. If your RDS instance runs PostgreSQL 10 or a later version, you do not need to enable the jsonbx plug-in.	Check the functions that are used by the jsonbx plug-in in the new major engine version and evaluate whether the plug-in needs to be deleted. If the plug-in needs to be deleted, we recommend that you delete the plug-in before an upgrade. Before you delete the plug-in, make sure that your RDS instance can run as expected without the plug-in. For more information, see Use differences of jsonbx plug-in functions.

Error message	Possible cause	Solution

• could not load library "$libdir/postgis-2.5": ERROR: could not access file "$libdir/postgis-2.5": No such file or directory
• could not load library "$libdir/postgis_topology-2.2": ERROR: could not access file "$libdir/postgis_topology-2.2": No such file or directory
	• The version of the postgis plug-in that is used is outdated and is incompatible with the specified libraries in the new major engine version. As a result, your RDS instance failed the upgrade check.
	• The version of the postgis_topology plug-in that is used is outdated and is incompatible with the specified libraries in the new major engine version. As a result, your RDS instance failed the upgrade check.
		Notice The postgis plug-in varies in different database engine versions. For example, this plug-in reports different parsing errors that are related to the WKT format in different database engine versions. Before you update this plug-in, we recommend that you clone your RDS instance. You can use the cloned RDS instance to test the compatibility of this plug-in with the new major engine version. After you verify that this plug-in is compatible with the new engine version, you can update this plug-in in your original RDS instance to the new major engine version. For more information, see Back up an ApsaraDB RDS for PostgreSQL instance and Restore the data of an ApsaraDB RDS for PostgreSQL instance.
		
		Perform the following steps:
		1. Update the minor engine version of your RDS instance. For more information, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.
		2. Update the plug-in that reports errors.
		◦ postgis
		`ALTER EXTENSION postgis UPDATE;`
		◦ postgis_topology
		`ALTER EXTENSION postgis_topology UPDATE;`
		3. Run the `\dx` command to query the version of the postgis plug-in. Make sure that the version of the postgis plug-in is 2.5.4 or later.
		4. Perform an upgrade check again.

Error message	Possible cause	Solution
		Notice If the postgis plug-in, postgis_topology plug-in, or pgrouting plug-in is installed on your RDS instance, take note of the following limits: ◦ If your RDS instance runs PostgreSQL 9.4, you can upgrade the major engine version of your RDS instance only to PostgreSQL 10 or PostgreSQL 11. ◦ If your RDS instance runs PostgreSQL 10, you can upgrade the major engine version of your RDS instance only to PostgreSQL 11. ◦ If your RDS instance runs PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13, you cannot upgrade the major engine version of your RDS instance.

Some functions that are used by the jsonbx plug-in may return different results in different PostgreSQL versions. The following table describes the differences in the results. Before you perform an upgrade, you must evaluate whether the jsonbx plug-in needs to be deleted.

Function	Return result in PostgreSQL 9.4	Return result in PostgreSQL 10 and later versions
select '{"a":1, "b":2, "c":3}'::jsonb - 2;	{"a": 1, "b": 2}	ERROR: cannot delete from object using integer index
select jsonb_delete('{"a":1, "b":2, "c":3}'::jsonb, '{b}'::text[]);	{"a": 1, "c": 3}	ERROR: function jsonb_delete(jsonb, text[]) does not exist
select '{"a":{"c":1, "d":2}, "b":3}'::jsonb - '{a, c}'::text[];	{"a": {"d": 2}, "b": 3}	null

tables_with_oids.txt

This item is used to display the tables that are created with the `WITH OIDS` clause.

13.4. Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to update the minor engine version of an ApsaraDB RDS for PostgreSQL instance. You can update the minor engine version of your RDS instance to improve database performance, use new features, and fix bugs.

For more information about the features that are supported by different minor engine versions, see Release notes for AliPG.

Precautions

- When you update the minor engine version of your RDS instance, the instance may encounter a transient connection of about 30 seconds. We recommend that you update the minor engine version during off-peak hours. Otherwise, make sure that your application is configured to automatically reconnect to your RDS instance.
- After you update the minor engine version of your RDS instance, you cannot downgrade the instance to the previously used version.
- When you perform operations such as upgrading your RDS instance, ApsaraDB RDS updates the instance to the latest minor engine version.

Update the minor engine version of an RDS instance equipped with local SSDs

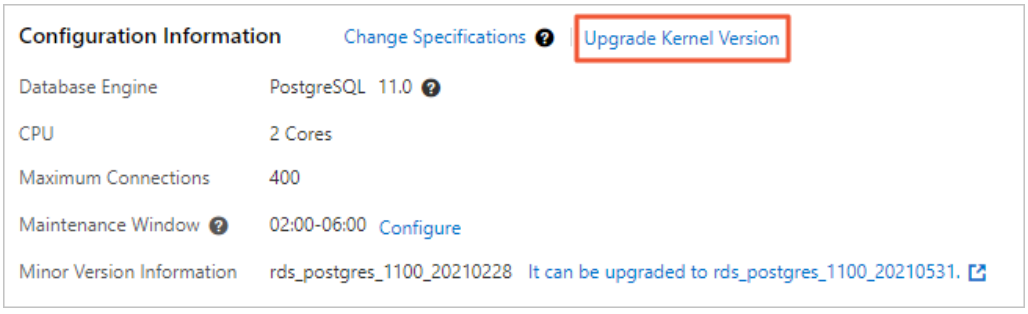
If your RDS instance is equipped with local SSDs, you cannot manually update the minor engine version of the instance. In this case, you can restart your RDS instance. During the restart process, ApsaraDB RDS updates the minor engine version of your RDS instance to the latest version. For more information, see Restart an ApsaraDB RDS for PostgreSQL instance.

> **Note** If your RDS instance is a primary RDS instance and is attached with read-only RDS instances, you must restart all the read-only RDS instances one by one before you restart the primary RDS instance. If you restart only the primary RDS instance, ApsaraDB RDS does not update the minor engine versions of the read-only RDS instances.

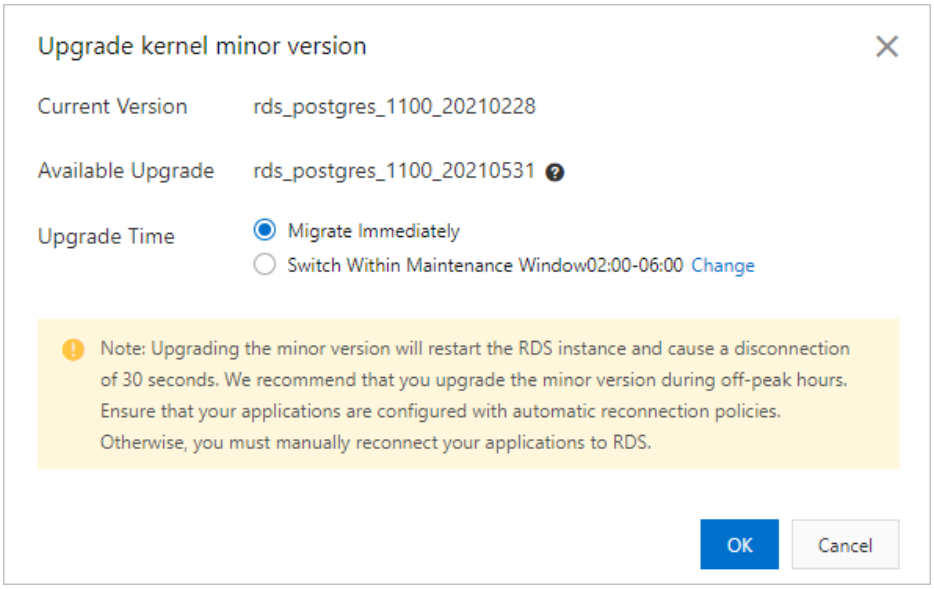
Update the minor engine version of an RDS instance equipped with standard SSDs or enhanced SSDs (ESSDs)

> **Note**
>
> If your RDS instance is a primary RDS instance and is attached with read-only RDS instances, you can use one of the following two methods to update the minor engine versions of these instances:
>
> - Update the minor engine version of the primary RDS instance. ApsaraDB RDS immediately updates the minor engine versions of the read-only instances all at once.
> - Update the minor engine versions of the read-only RDS instances one by one. Then, update the minor engine version of the primary RDS instance. This method is suitable if you do not want to update the minor engine versions of the read-only instances all at once.

1.
2. In the **Configuration Information** section of the page, click **Upgrade Minor Engine Version**.



3. In the dialog box that appears, select **Upgrade Time** and click **OK**.



View the minor engine version of an RDS instance

You can use one of the following two methods to view the minor engine version of your RDS instance:

- Log on to the ApsaraDB RDS console and go to the **Basic Information** page of the RDS instance.

  > **Note** This method is supported only for RDS instances that are equipped with standard SSDs or ESSDs.

- Connect to the RDS instance and run the `show rds_release_date;` command. For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance.

  > **Note** This method is supported for RDS instances that are equipped with local SSDs, standard SSDs, or ESSDs.

Related operations

Operation	Description
Update minor engine version	Updates the minor engine version of an ApsaraDB RDS instance.

14.Instance

14.1. Create an ApsaraDB RDS for PostgreSQL instance

This topic describes how to create an ApsaraDB RDS for PostgreSQL instance in the ApsaraDB RDS console. You can also create an ApsaraDB RDS for PostgreSQL instance by calling an API operation.

> **Note** You are offered a reduced price on your first purchase of an RDS instance. For more information, visit the ApsaraDB RDS promotion page.

Prerequisites

The AliyunRDSFullAccess policy is attached to the RAM user that you use to create an RDS instance. For more information, see Use RAM for resource authorization.

Procedure

1. Go to the ApsaraDB RDS buy page.
2. Configure the **Billing Method** parameter.

Billing method	Description	Benefit
Subscription	A subscription instance is an instance for which you pay an upfront fee. If you want to use an instance for a long period of time, we recommend that you select the **Subscription** billing method. If you select the subscription billing method, configure the **Duration** parameter in the lower part of the page.	In most cases, the subscription billing method is more cost-effective than the pay-as-you-go billing method for long-term usage. Alibaba Cloud provides lower prices for longer subscription periods.
Pay-As-You-Go	You are charged on an hourly basis for a pay-as-you-go instance based on your actual resource usage. If you want to use an instance for a short period of time, we recommend that you select the **Pay-As-You-Go** billing method. You can create a **pay-as-you-go** RDS instance. After you confirm that the new RDS instance meets your business requirements, you can change the billing method of the RDS instance from pay-as-you-go to **subscription**.	You can release a pay-as-you-go RDS instance based on your business requirements. The billing cycle of a pay-as-you-go RDS instance immediately stops after you release the instance.

> **Note** You can view the price in the lower-right corner of the page. The price is displayed only after you configure all required parameters.

3. Configure the following parameters.

Parameter	Description
Region	The region where the RDS instance resides. We recommend that you select the region of on which your application is deployed. If the RDS instance and the ECS instance reside in different regions, you cannot connect these instances over an internal network. In this case, these instances cannot deliver the optimal performance. the Elastic Compute Service (ECS) instance **Note** ◦ After an RDS instance is created, you cannot change the region of the RDS instance. If you want to connect an ECS instance and an RDS instance over an internal network, make sure that the RDS instance and the ECS instance reside in the same region. ◦ For more information about how to view the region of an ECS instance, see Get ready to use ApsaraDB RDS for MySQL. ◦ If your application is deployed on an on-premises server or on-premises computer, we recommend that you select a region that is near your on-premises server or on-premises computer. This way, you can use the public endpoint of the RDS instance to connect to the RDS instance from your application.
Database Engine	The database engine and version that are run by the RDS instance. Select **PostgreSQL**. The supported PostgreSQL versions are 10, 11, 12, 13, and 14. **Notice** ApsaraDB RDS for PostgreSQL provides the Babelfish feature that is developed based on the Babelfish for PostgreSQL open source project. This feature enables your RDS instance to be compatible with Transact-SQL (T-SQL) statements. If you want to connect your SQL Server application or client to an RDS instance that runs PostgreSQL, we recommend that you select **Enable Babelfish** when you create the RDS instance. For more information, see Introduction to Babelfish.
Edition	◦ **Basic**: In RDS Basic Edition, the database system consists of only a primary RDS instance. RDS Basic Edition is cost-effective and suitable for learning and testing. **Note** RDS instances that run RDS Basic Edition require a long period of time to restart or recover from faults. ◦ **High-availability**: This is the recommended edition. In RDS High-availability Edition, the database system consists of a primary RDS instance and a secondary RDS instance. These instances work in the high availability architecture. RDS High-availability Edition is suitable for production environments and more than 80% of business scenarios. **Note** ◦ The available RDS editions vary based on the region and database engine that you select. ◦ For more information, see Overview of ApsaraDB RDS editions.

Parameter	Description
Storage Type	◦ **Local SSD**: A local SSD resides on the same host as the database engine. You can store data on local SSDs to reduce I/O latency. Local SSDs are supported only for RDS instances that run PostgreSQL 10. ◦ **ESSD**: Enhanced SSDs (ESSDs) come in three performance levels (PLs). ■ ESSD PL1: This is the basic PL of ESSDs. ■ ESSD PL2: An ESSD of PL2 delivers IOPS and throughput that are approximately twice the IOPS and throughput delivered by an ESSD of PL1. ■ ESSD PL3: An ESSD of PL3 delivers IOPS that is up to 20 times the IOPS delivered by an ESSD of PL1 and up to 11 times the throughput delivered by an ESSD of PL1. ESSDs of PL3 are suitable for business scenarios in which highly concurrent requests must be processed with high I/O performance and at low read and write latencies. ◦ **Standard SSD**: A standard SSD is an elastic block storage device that is designed based on the distributed storage architecture. You can store data on standard SSDs to separate computing from storage. **Note** ◦ The available storage types vary based on the instance type and RDS edition that you select. ◦ If you select the **ESSD** or **Standard SSD** storage type, you can select **Disk Encryption** to enhance the security of your data. For more information, see Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance. ◦ For more information about storage types, see Storage types.
Zone of Primary Node	Select a . zone

Parameter	Description
Deployment Method	◦ **Multi-zone Deployment**: This is the recommended deployment method. The primary RDS instance and the secondary RDS instance reside in different zones to provide zone-disaster recovery. ◦ **Single-zone Deployment**: The primary RDS instance and the secondary RDS instance reside in the same zone. **Note** ◦ No substantive differences exist between the zones in the same region. ◦ If the RDS instance resides in the same zone as the ECS instance on which your application is deployed, these instances can deliver optimal performance. If the RDS instance and the ECS instance reside in different zones in the same region, the performance of the RDS instance and the ECS instance is slightly lower than the performance of the RDS instance and the ECS instance that reside in the same zone. ◦ If you set the Edition parameter to **Basic**, only the **Single-zone Deployment** method is supported. ◦ If **Sold Out** appears in the upper-right corner of a zone name, this zone does not have sufficient resources. In this case, you must switch to another zone.
Zone of Secondary Node	If you set the **Deployment Method** parameter to **Multi-zone Deployment**, you must select the zone in which the secondary RDS instance resides.
Instance Type	The instance type of the RDS instance. Before you select an instance type, you must select an instance family. ◦ **General-purpose**: A general-purpose RDS instance exclusively occupies the allocated memory and I/O resources. The RDS instance shares CPU and storage resources with the other general-purpose RDS instances deployed on the same server. ◦ **Dedicated**: A dedicated instance exclusively occupies the allocated CPU, memory, storage, and I/O resources. Dedicated host instance types provide the highest specifications in the dedicated instance family. A dedicated host instance exclusively occupies all the CPU, memory, storage, and I/O resources on the physical host on which the RDS instance is deployed. ◦ **General-purpose (New)**: The new general-purpose instance types provide better scalability and higher performance than the old general-purpose instance types. In addition, the period of time that is required to create an RDS instance and the period of time that is required to change the specifications of an RDS instance are reduced. The new general-purpose instance types are in development. Some features of ApsaraDB RDS are not supported for RDS instances that use the new general-purpose instance types. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types. **Note** ◦ In a test environment, select an instance type that provides one or more cores. ◦ In a production environment, select an instance type that provides four or more cores. For more information, see Primary ApsaraDB RDS instance types.

Parameter	Description
Capacity	The storage capacity that is provided to store data files, system files, binary log files, and transaction files in the RDS instance. The storage capacity varies based on the instance type and storage type that you select. You can adjust the storage capacity at a step size of 5 GB. **Note** ◦ If you select the local SSD storage type, the storage capacity of the RDS instance may vary based on the instance type. If you select the standard SSD or ESSD storage type, the storage capacity of the RDS instance does not vary based on the instance type. For more information, see Primary ApsaraDB RDS instance types. ◦ After an RDS instance is created, you can adjust the storage capacity of the RDS instance by changing the specifications or configuring automatic storage expansion. For more information, see 変更配置 or Configure automatic storage expansion for an ApsaraDB RDS for PostgreSQL instance.

4. In the lower-right corner of the page, click **Next: Instance Configuration**.
5. Configure the **VPC** and **VSwitch** parameters. We recommend that you select the same virtual private cloud (VPC) in which your ECS instance resides for your RDS instance. If you select a different VPC for your RDS instance, you cannot connect your RDS instance and ECS instance over an internal network.

   Note
   - For more information about how to view the VPC in which your ECS instance resides, see Get ready to use ApsaraDB RDS for MySQL.
   - You can connect the RDS instance and the ECS instance over an internal network even if the instances use different vSwitches in the same VPC.

6. Configure more custom parameters. If you do not have special business requirements, you can retain the default values of these parameters.

Parameter	Description
Release Protection	Specifies whether to enable the release protection feature. The release protection feature is used to prevent a pay-as-you-go RDS instance from being released due to unintended operations. For more information, see Enable or disable the release protection feature for an ApsaraDB RDS for PostgreSQL instance.
Resource Group	The resource group to which the RDS instance belongs. You can retain the default resource group or select a custom resource group based on your business requirements.

Parameter	Description
Babelfish Migration Mode	The migration mode of the RDS instance after Babelfish is enabled. This parameter takes effect only when you select **Enable Babelfish** in the Basic Configurations step. ○ **single-db**: You can create only one SQL Server database on an RDS instance for which Babelfish is enabled and create a standard PostgreSQL schema for the database. ○ **multi-db**: You can create multiple SQL Server databases and create different PostgreSQL schemas for the databases. You must name the schemas in the <Database name>_<Schema name> format to prevent name conflicts. **Note** For more information, see Migration modes.
Initial Account	The username of the Babelfish management account. This parameter takes effect only when you select **Enable Babelfish** in the Basic Configurations step. The Babelfish management account is used to connect to the RDS instance over the TDS port. **Notice** This account is a privileged account and cannot be deleted after it is created. Username requirements: ○ The username must be 2 to 63 characters in length. ○ The username can contain lowercase letters, digits, and underscores (_). ○ The username must start with a lowercase letter and end with a lowercase letter or a digit. ○ The username cannot start with pg. ○ The username cannot contain SQL keywords. For more information, see SQL keywords.
Password	The password of the Babelfish management account. This parameter takes effect only when you select **Enable Babelfish** in the Basic Configurations step. **Note** You can change the password after the RDS instance is created. For more information, see Reset the password of an account on an ApsaraDB RDS for PostgreSQL instance. Password requirements: ○ The password must be 8 to 32 characters in length. ○ The password must contain at least three of the following character types: uppercase letters, lowercase letters, digits, and special characters. ○ The following special characters are supported: `! @ # $ % ^ & * ( ) _ + - =` .

Parameter	Description
Time Zone	The time zone of the RDS instance. **Note** ◦ You can configure the time zone when you create a primary RDS instance. You cannot configure the time zone when you create a read-only RDS instance. Read-only RDS instances inherit the time zone of their primary RDS instance. ◦ You can configure this parameter only when the RDS instance uses standard SSDs or ESSDs. ◦ The time zone is not in UTC. For more information about time zones, see Common time zones for ApsaraDB RDS for MySQL instances and ApsaraDB RDS for PostgreSQL instances. ◦ If you do not configure this parameter, the system assigns the default time zone of the region in which the RDS instance resides to the RDS instance. For more information about the mappings between regions and time zones, see Default time zones for ApsaraDB RDS for PostgreSQL instances.

7. In the lower-right corner of the page, click **Next: Confirm Order**.
8. Confirm the configuration of the RDS instance in the Parameters section, configure the **Purchase Plan** and **Duration** parameters, read and select **Terms of Service**, and then click **Pay Now**. You must configure the Duration parameter only if you select the subscription billing method for the RDS instance.

   Note If you select the subscription billing method for the RDS instance, we recommend that you select **Auto-Renew Enabled**. This way, you can prevent interruptions on your workloads even if you forget to renew the RDS instance.

9. View the RDS instance.

   Go to the Instances page. In the top navigation bar, select the region in which the RDS instance resides. Then, find the RDS instance based on the **Creation Time** parameter.

What to do next

Create a database and an account on an ApsaraDB RDS for PostgreSQL instance

FAQ

Why am I unable to find the RDS instance that I created?

Possible cause	Description	Suggestion
Incorrect region	The RDS instance does not reside in the region that you selected in the top navigation bar of the ApsaraDB RDS console.	In the top navigation bar, select the region in which the RDS instance resides.
Insufficient resources	The zone that you selected cannot provide sufficient resources. If the RDS instance cannot be created, you can go to the Orders page in the Billing Management console to view the refunded fee.	Select a different zone and try again.
RAM policies that do not allow users to create unencrypted RDS instances	RAM policies that do not allow users to create unencrypted RDS instances are attached to a RAM user. • If you use the credentials of the RAM user to create an RDS instance that uses local SSDs, the RDS instance cannot be created. When you create an RDS instance that uses local SSDs, you cannot enable disk encryption. • If you use the credentials of the RAM user to create an RDS instance that uses standard SSDs or ESSDs and you do not enable disk encryption for the RDS instance, the RDS instance cannot be created. For more information, see Use RAM policies to manage the permissions of RAM users on ApsaraDB RDS instances.	When you create an RDS instance, select the standard SSD or ESSD storage type, select Disk Encryption, set the Key parameter, and then try again.

References

- For more information about how to create an RDS instance by calling an API operation, see Create an instance.
- For more information about how to create an RDS instance that runs a different database engine, see the following topics:
  - Create an ApsaraDB RDS for MySQL instance
  - Create an ApsaraDB RDS for SQL Server instance
  - Create an ApsaraDB RDS for MariaDB TX instance

14.2. Restart an ApsaraDB RDS for PostgreSQL instance

This topic describes how to manually restart an ApsaraDB RDS instance. If the number of connections to your RDS instance exceeds the specified threshold or if your RDS instance encounters performance issues, you can manually restart the instance.

Impacts

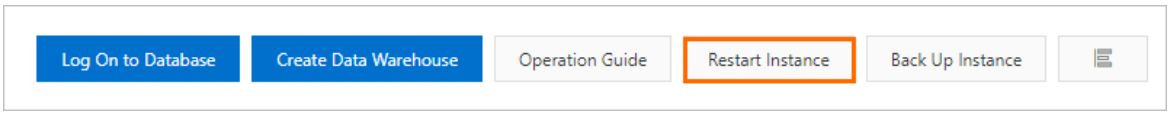
- During the restart process, your RDS instance encounters a transient connection of about 30 seconds. Before you restart your RDS instance, we recommend that you make suitable arrangements for your workloads. Proceed with caution.

> **Note** If you are using the RDS Basic Edition, your RDS instance does not have a secondary RDS instance as a hot standby. In this case, if your RDS instance exits unexpectedly, your database service becomes unavailable. If you change the specifications or upgrade the database engine version of your RDS instance, your database service also becomes unavailable. The unavailability may last for a long period of time. If you require high service availability, we recommend that you do not select the RDS Basic Edition. For example, you can select the RDS High-availability Edition. Alternatively, you can upgrade your RDS instance from the RDS Basic Edition to the RDS High-availability Edition. The upgrade is supported only when your RDS instance meets the specified requirements. For more information, see Upgrade an RDS instance to the High-availability Edition.

- After your RDS instance is restarted, ApsaraDB RDS updates your RDS instance to the latest minor engine version. This applies if your RDS instance uses local SSDs. For more information, see Updates the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

Procedure

1.
2. In the upper-right corner of the Basic Information page, click **Restart Instance**.



3. In the message that appears, click **OK**.

Related operations

Operation	Description
Restart an ApsaraDB for RDS instance	Restarts an ApsaraDB RDS instance.

14.3. Switch workloads over between primary and secondary ApsaraDB RDS for PostgreSQL instances

ApsaraDB RDS for MySQL provides the primary/secondary switchover feature to ensure high availability. If the primary RDS instance of your database system fails, ApsaraDB RDS automatically switches your workloads over from the primary RDS instance to the secondary RDS instance to ensure service availability. After the primary/secondary switchover is complete, the secondary RDS instance serves as the primary RDS instance. The endpoint that is used to connect to your database system remains unchanged. Your application can automatically connect to the new primary RDS instance by using the endpoint. You can also manually switch your workloads over between the primary RDS instance and the secondary RDS instance.

Prerequisites

The primary RDS instance runs RDS High-availability Edition.

> **Note**
>
> - Primary RDS instances that run RDS Basic Edition do not have secondary RDS instances as standbys. Therefore, these primary RDS instances do not support primary/secondary switchovers.
> - You can disable automatic primary/secondary switchovers for a short period of time only for primary RDS instances that run RDS High-availability Edition with standard SSDs or enhanced SSDs (ESSDs).

Context

- Automatic primary/secondary switchover: By default, the automatic primary/secondary switchover feature is enabled. If the primary RDS instance fails, ApsaraDB RDS automatically switches your workloads over to the secondary RDS instance. For more information about the causes of primary/secondary switchovers, see Reasons for primary/secondary switchovers.
- Manual primary/secondary switchover: You can manually switch your workloads over between the primary RDS instance and the secondary RDS instance even if the automatic primary/secondary switchover feature is enabled. You can perform manual primary/secondary switchovers for disaster recovery drills. You can also perform manual primary/secondary switchovers if you use the multi-zone deployment method and want to connect your application to the RDS instance in the zone that is closest to your application.

> **Note** Data is synchronized between the primary RDS instance and the secondary RDS instance in real time. You can access only the primary RDS instance. The secondary RDS instance runs only as a standby.

Impacts

- Transient connections may occur during a primary/secondary switchover. Make sure that your application is configured to automatically reconnect to your database system.
- After a primary/secondary switchover, the read-only RDS instances that are attached to the primary RDS instance must re-establish the connections that are used to replicate data to and synchronize incremental data from the primary RDS instance. As a result, the data on the read-only RDS instances shows latencies of a few minutes.
- A primary/secondary switchover does not cause changes to the endpoints that are used to connect to your database system.

Perform a manual primary/secondary switchover

1.
2. In the left-side navigation pane, click **Service Availability**.
3. In the **Availability Information** section of the page that appears, click **Switch Primary/Secondary Instance**.
4. Specify the point in time at which you want to perform a switchover. Then, click **OK**.

> **Note** During a primary/secondary switchover, you cannot perform a number of operations. For example, you cannot manage databases and accounts or change the network type. We recommend that you select **Switch Within Maintenance Window**.

Disable automatic primary/secondary switchovers for a short period of time

By default, the automatic primary/secondary switchover feature is enabled. If the primary RDS instance fails, ApsaraDB RDS automatically switches your workloads over from the primary RDS instance to the secondary RDS instance. You can disable the automatic primary/secondary switchover feature in the following situations:

- A large-scale sales promotion during which you do not want a primary/secondary switchover to affect system availability
- An important application upgrade during which you do not want a primary/secondary switchover to cause unexpected issues
- A major event during which you do not want a primary/secondary switchover to affect system stability

1.
2. In the left-side navigation pane, click **Service Availability**.
3. In the **Availability Information** section of the page that appears, click **Configure Primary/Secondary Switchover**.

   > **Note** If **Configure Primary/Secondary Switchover** is not displayed, you must check whether the RDS instance runs RDS High-availability Edition.

4. Select **Disable Temporarily**, specify the **Deadline** parameter, and then click **OK**.

   > **Note**
   > - When the point in time that is specified by the **Deadline** parameter arrives, automatic primary/secondary switchovers are automatically enabled.
   > - If you do not specify the Deadline parameter, automatic primary/secondary switchovers are disabled for one day. You can set the Deadline parameter to 23:59:59 seven days later at most.

After you disable the automatic primary/secondary switchover feature, you can go to the **Service Availability** page to check the deadline after which the automatic primary/secondary switchover feature can be automatically enabled.

FAQ

- Can I access the secondary RDS instance of my database system?

  No, you cannot access the secondary RDS instance of your database system. You can access only the primary RDS instance of your database system. The secondary RDS instance runs only as a standby.

- Do I need to manually switch my workloads over from the secondary RDS instance to the primary RDS instance after a primary/secondary switchover?

  No, you do not need to manually switch your workloads over from the secondary RDS instance to the primary RDS instance after a primary/secondary switchover. The data in the primary RDS instance is the same as the data in the secondary RDS instance. After a primary/secondary switchover, the secondary RDS instance serves as the new primary RDS instance. No additional operations are required.

- Each time a primary/secondary switchover is performed, my RDS instance does not run as expected 10 minutes after the primary/secondary switchover is complete. What are the possible causes? How do I handle the issue?

If an exception on your RDS instance triggers a primary/secondary switchover to ensure high availability, your application may fail to identify and respond to the changes to the connections. If no timeout periods are specified for socket connections, your application waits for the database to return the results. In most cases, your application is disconnected after hundreds of seconds. During this period, some connections to the database cannot work as expected, and a large number of SQL statements fail to be executed. To avoid invalid connections, we recommend that you configure the **connectTimeout** and **socketTimeout** parameters to prevent your application from waiting for a long period of time due to network errors. This reduces the time required to recover from failures.

You must configure these parameters based on your workloads and usage modes. For online transactions, we recommend that you set **connectTimeout** to 1 to 2 seconds and **socketTimeout** to 60 to 90 seconds. This configuration is for reference only.

Related operations

Operation	Description
Switch services between a primary ApsaraDB for RDS instance and its secondary instance	Switches workloads over between primary and secondary ApsaraDB RDS instances.
Enable or disable automatic primary/secondary switchover	Enables or disables the automatic primary/secondary switchover feature for an ApsaraDB RDS instance.
Query settings of automatic primary/secondary switchover	Queries the settings of the automatic primary/secondary switchover feature for an ApsaraDB RDS instance.

14.4. Set the maintenance window of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to set the maintenance window of an ApsaraDB RDS for PostgreSQL instance. The backend system performs maintenance on the RDS instance during the maintenance window. This ensures the stability of the RDS instance. The default maintenance window spans from 02:00:00 to 06:00:00 UTC+8. We recommend that you set the maintenance window to an off-peak hour. This allows you to avoid interruptions to your workloads.

Precautions

- Before the maintenance starts, ApsaraDB RDS sends emails to the contacts that are associated with your Alibaba Cloud account. We recommend that you check your email box on a regular basis to obtain up-to-date information.
- When the maintenance window arrives, your RDS instance enters the **Maintaining Instance** state. This ensures a smooth maintenance process. Database access and query operations such as performance monitoring are not affected while the instance is in this state. However, except for account and database management and IP address whitelist configuration, modification operations such as upgrade, downgrade, and restart are temporarily unavailable.
- During the maintenance window, one or two transient connections may occur. Make sure that your application is configured to automatically reconnect to your RDS instance.

Procedure

1.
2. In the **Configuration Information** section, click **Configure** next to **Maintenance Window**.
3. Select an appropriate maintenance window and click OK to save the setting.

> **Note** The time zone of the maintenance window is the same as that of the computer that you use to log on to the ApsaraDB RDS console.

Related operations

Operation	Description
Modify the maintenance time	Modifies the maintenance window of an ApsaraDB RDS instance.

14.5. Configure automatic storage expansion for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure automatic storage expansion for an ApsaraDB RDS for PostgreSQL instance.

Prerequisites

- The RDS instance runs PostgreSQL with standard SSDs or enhanced SSDs (ESSDs) and the instance is in the Running state.
- The balance in your Alibaba Cloud account is sufficient for the expansion.

Precautions

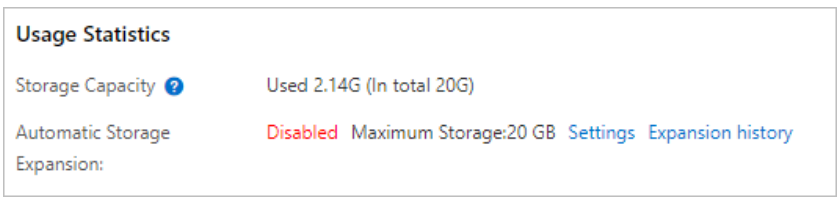
After the storage capacity of your RDS instance is expanded, you cannot reduce the storage capacity.

Billing

The billing rules for automatic storage expansion are the same as the billing rules for manual storage expansion of an RDS instance. For more information, see Specification change fees.

Procedure

1.
2. In the **Usage Statistics** section of the page that appears, click **Settings** to the right of **Automatic Storage Expansion**.



3. Configure the following parameters.

Parameter	Requirement
Automatic Resource Scalability	The switch that is used to enable or disable automatic storage expansion.
Available Storage Threshold<=	The threshold based on which ApsaraDB RDS triggers an automatic storage expansion. The threshold is expressed as a percentage. If the storage usage reaches the threshold, ApsaraDB RDS increases the storage capacity of the RDS instance. **Note** The maximum amount of storage that you can increase is the larger value of the following values: ○ 5 GB ○ 15% of the current storage capacity of the RDS instance
Maximum Storage	The maximum storage capacity that is allowed by automatic storage expansion. The value of this parameter must be greater than or equal to the current storage capacity of the RDS instance. ○ If the RDS instance uses ESSDs, the maximum value of this parameter can be set to 32000 GB. ○ If the RDS instance uses SSDs, the maximum value of this parameter can be set to 6000 GB.

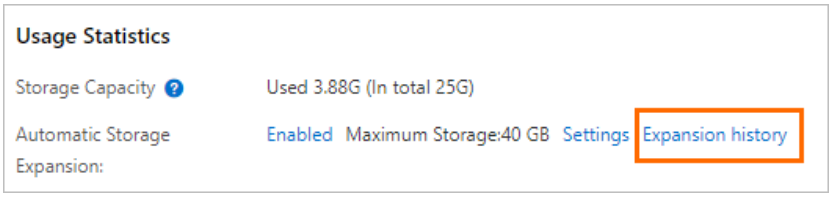
4. Click **Confirm**.

View a storage expansion record

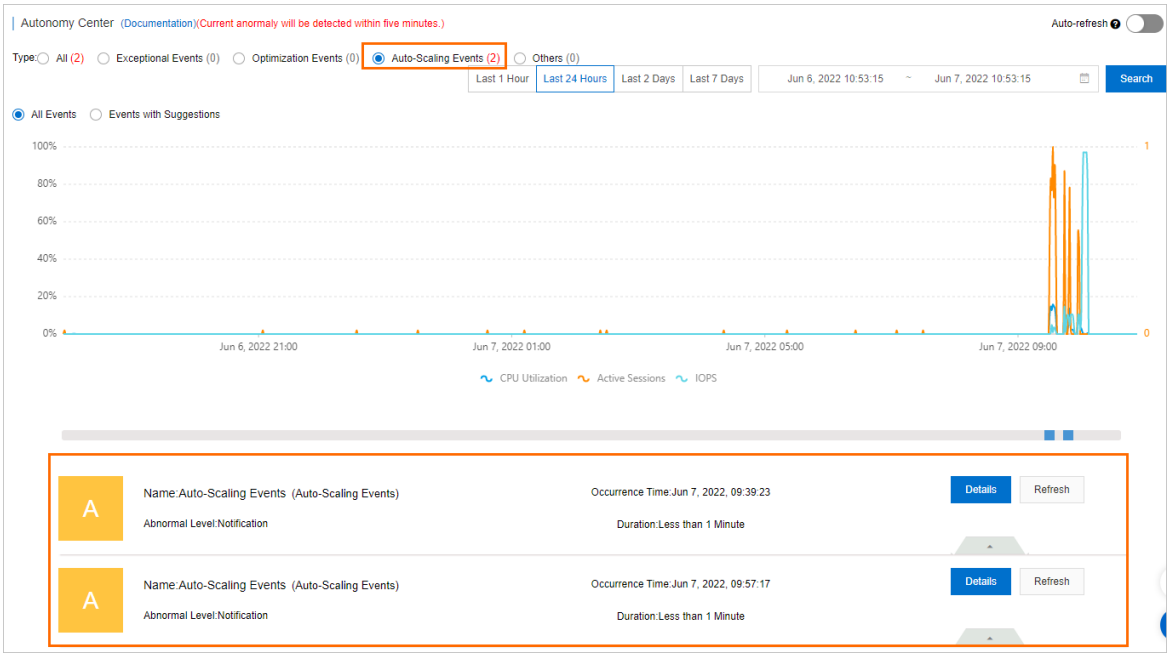
Note Storage expansion records of RDS instances that run RDS Basic Edition with standard SSDs or ESSDs are not viewable.

1.

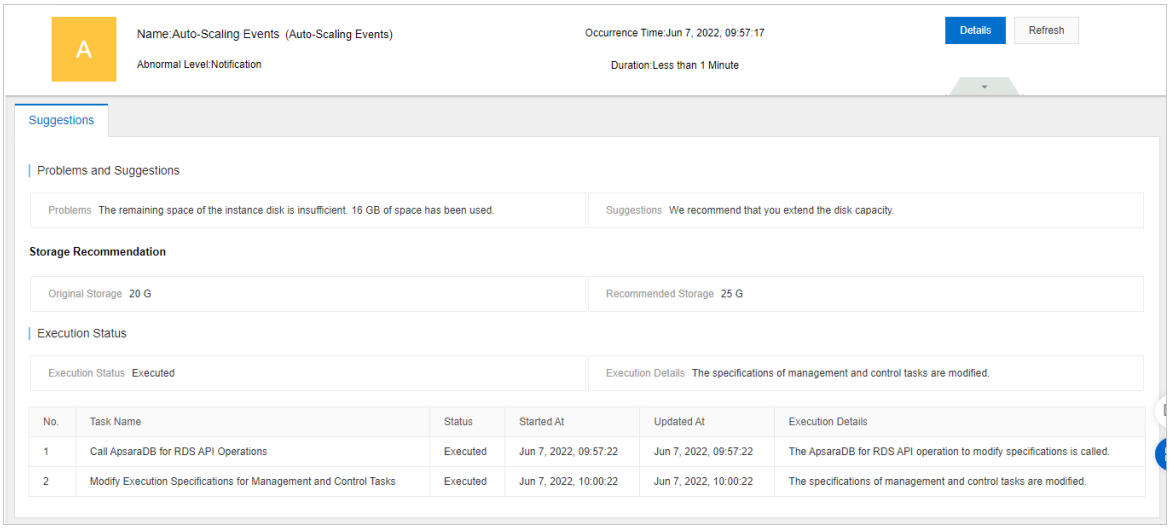
2. In the **Usage Statistics** section, click **Expansion history** to the right of **Automatic Storage Expansion** to go to the **Database Autonomy Service (DAS)** console.



3. On the **Autonomy Center** page, set **Type** to **Auto-Scaling Events** to view the storage expansion records.



4. Click **Details** of the required scaling event to view the details of the storage expansion record.



Related operations

Operation	Description
ModifyDasInstanceConfig	Configures automatic storage expansion for an ApsaraDB RDS instance.

14.6. Migrate an ApsaraDB RDS for PostgreSQL instance across zones in the same region

This topic describes how to migrate an ApsaraDB RDS for PostgreSQL instance across zones in the same region. After your RDS instance is migrated, its attributes, configuration, and endpoints remain unchanged. The amount of time that is required to complete the migration varies based on the amount of data that needs to be migrated. In most cases, the migration requires a few hours.

Prerequisites

- The storage type, PostgreSQL version, RDS edition, and instance family of your RDS instance meet the requirements for cross-zone migration. For more information, see the following table.

Storage type	PostgreSQL version	RDS edition	Instance family	Description
Standard SSD or ESSD	PostgreSQL 14, PostgreSQL 13, PostgreSQL 12, PostgreSQL 11, or PostgreSQL 10	High-availability Edition	Dedicated instance family	The migration is supported for all RDS instances that use one of the dedicated instance types.
			General-purpose instance family	The migration is supported only for RDS instances that use one of the following instance types: ○ pg.n2.medium.2c ○ pg.n2.small.2c
		Basic Edition	General-purpose instance family	The migration is not supported.
Local SSD	RDS PostgreSQL 10	High-availability Edition	General-purpose instance family	The migration is supported for RDS instances that use local SSDs.
	RDS PostgreSQL 9.4	High-availability Edition	Dedicated instance family	
			General-purpose instance family	

- The region to which your RDS instance belongs consists of multiple zones. For more information about the regions and zones of Alibaba Cloud, see Regions and zones.
- Your RDS instance is a primary RDS instance, and no read-only RDS instances are attached to your RDS instance.

Precautions

- If your database system uses standard SSDs or enhanced SSDs (ESSDs), you cannot retain the primary RDS instance in the original zone and migrate only the secondary RDS instance across zones within the same region. Examples:

  - **The primary RDS instance resides in Zone A of the Singapore (Singapore) region and the secondary RDS instance resides in Zone B of the Singapore (Singapore) region.** You cannot retain **the primary RDS instance in Zone A of the Singapore (Singapore) region and migrate only the secondary RDS instance to Zone C of the Singapore (Singapore) region.**
  - **The primary RDS instance resides in Zone A of the Singapore (Singapore) region and the secondary RDS instance resides in Zone B of the Singapore (Singapore) region.** You can migrate **the primary RDS instance to Zone C of the Singapore (Singapore) region and the secondary RDS instance to Zone A of the Singapore (Singapore) region.**
- If your database system uses local SSDs, you can migrate only the primary RDS instance across zones in the same region.

Fees

You are not charged for the migration. This applies even if you migrate your RDS instance from one zone to multiple zones.

Impacts

- During the migration, your database service may be unavailable for a short period of time. Make sure that your application is configured to automatically reconnect to your RDS instance.
- The migration causes changes to the virtual IP addresses (VIPs) of your RDS instance. We recommend that you use an endpoint rather than an IP address of your RDS instance to connect your application to your RDS instance.
- After the migration, you must immediately delete the cached DNS records from the database client. If the database client runs on a JVM, we recommend that you set the time-to-live (TTL) in the JVM configuration to 60 seconds or less. In this case, if the virtual IP address that is bound to the in-use endpoint of your RDS instance changes, your application can query the related DNS records again to obtain the new virtual IP address. Then, your application can connect to the new virtual IP address.

  > **Note** The following TTL-setting methods are provided for reference:
  >
  > - Set the TTL for all JVM-based applications: Set the networkaddress.cache.ttl parameter in the *$JAVA_HOME/jre/lib/security/java.security* file to 60.
  > - Set the TTL only for local applications: Specify the `networkaddress.cache.ttl java.security.Security.setProperty("networkaddress.cache.ttl" , "60");` setting in the initialized code of the local applications. This must be completed before you call the `InetAddress.getByName()` function for the first time or before you establish a network connection.
- If your RDS instance has an ongoing Data Transmission Service (DTS) task, you must restart the DTS task after the migration is complete.

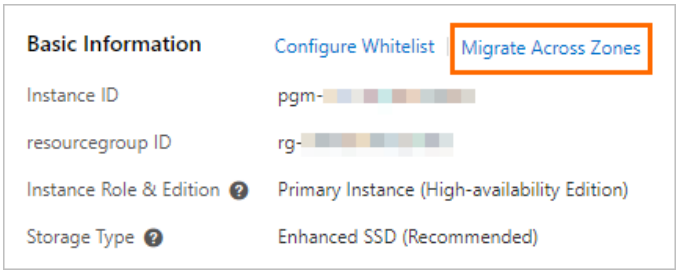
Migration scenarios

Migration scenario	Description
Migration from one zone to another zone	The original zone where your RDS instance resides cannot ensure service performance due to issues such as heavy loads.

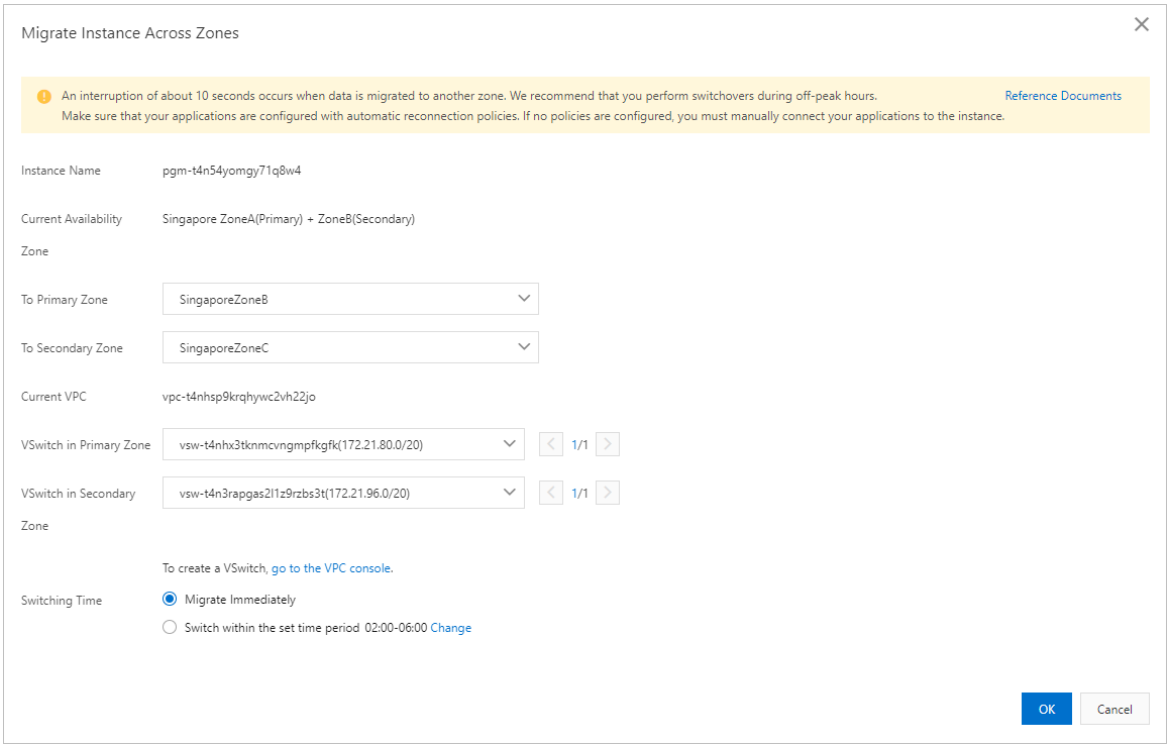
Migration scenario	Description
Migration from one zone to multiple zones	You want to implement disaster recovery across data centers. After the migration is complete, your RDS instance and its secondary RDS instance reside in different zones. The multi-zone deployment method delivers higher disaster recovery capabilities than the single-zone deployment method. If you select the single-zone deployment method, your database system can withstand server and rack failures. If you select the multi-zone deployment method, your database system can withstand data center failures.
Migration from multiple zones to one zone	You want to use specific features that are supported only for the single-zone deployment method.

Procedure

1.
2. In the **Basic Information** section of the page that appears, click **Migrate Across Zones**.



3. In the dialog box that appears, select the primary zone, secondary zone, and vSwitches, and click **OK**.



Parameters

<table>
<tr><th>Parameter</th><th>Description</th></tr>
<tr><td>To Primary Zone</td><td rowspan="2">Select a new Primary Zone and a new Secondary Zone.
If your database system uses local SSDs, you can migrate only the primary RDS instance across zones in the same region. You cannot migrate the secondary RDS instance.
If your database system uses standard SSDs or ESSDs, you cannot retain the primary RDS instance in the original zone and migrate only the secondary RDS instance across zones within the same region.
Note For example, the primary RDS instance resides in Zone A of the Singapore (Singapore) region and the secondary RDS instance resides in Zone B of the Singapore (Singapore) region. In this case, if you want to retain the primary RDS instance in Zone A of the Singapore (Singapore) region and migrate the secondary RDS instance to Zone C of the Singapore (Singapore) region, you can perform the following operations:<ol type="a">
Migrate the primary RDS instance to Zone C of the Singapore (Singapore) region and the secondary RDS instance to Zone A of the Singapore (Singapore) region.
Perform a primary/secondary switchover. After the switchover is complete, the original secondary RDS instance that resides in Zone A of the Singapore (Singapore) region becomes the new primary RDS instance, and the original primary RDS instance that resides in Zone C of the Singapore (Singapore) region becomes the new secondary RDS instance. For more information, see Switch workloads over between primary and secondary ApsaraDB RDS for PostgreSQL instances.</td></tr>
<tr><td>To Secondary Zone</td></tr>
<tr><td>vSwitch in Primary Zone</td><td rowspan="2">Select vSwitches for the primary zone and the secondary zone. If no vSwitches are available in the destination zones, create vSwitch in the destination zones. For more information, see Create a vSwitch.</td></tr>
<tr><td>vSwitch in Secondary Zone</td></tr>
</table>

Parameter	Description
Switching Time	○ **Migrate Immediately**: The migration is immediately started after you configure the migration. ○ **Switch within the set time period**: Click **Change** to select a time period during which you want to perform the migration.

After you click **OK**, ApsaraDB RDS starts to copy the data of your RDS instance to the destination zones. This process does not interrupt the workloads on your RDS instance. After all data is copied to the destination zones, ApsaraDB RDS switches the workloads over to the destination zones at the specified time period. You can set the switching time to **Migrate Immediately** or **Switch within the set time period**.

> **Note**
>
> ○ The migration triggers a transient connection. Make sure that your application is configured to automatically reconnect to your RDS instance. Otherwise, you must manually reconnect your application to your RDS instance after the migration.
>
> ○ If the DNS records cached on the database client are not immediately updated after the migration, some workloads may be switched over to the destination zones 10 minutes later. As a result, your RDS instance encounters another transient connection. If the database client runs on a JVM, we recommend that you set the time-to-live (TTL) in the JVM configuration to 60 seconds or less. In this case, if the virtual IP address that is bound to the in-use endpoint of your RDS instance changes, your application can query the related DNS records again to obtain the new virtual IP address. Then, your application can connect to the new virtual IP address. For more information, see the "Impacts" section of this topic.

Related operations

Operation	Description
Migrate an instance across zones	Migrates an ApsaraDB RDS instance across zones in the same region.

14.7. Release or unsubscribe from an ApsaraDB RDS for PostgreSQL instance

This topic describes how to manually release a pay-as-you-go-billed ApsaraDB RDS for PostgreSQL instance or unsubscribe from a subscription-billed ApsaraDB RDS for PostgreSQL instance.

Precautions

After you release or unsubscribe from an RDS instance, the RDS instance and its data are immediately deleted. Before you release or unsubscribe from an RDS instance, we recommend that you back up the RDS instance and download the required backup file.

Release a pay-as-you-go-billed RDS instance

14.8. Enable or disable the release protection feature for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to enable or disable the release protection feature for an ApsaraDB RDS for PostgreSQL instance. If your RDS instance uses the pay-as-you-go billing method and runs critical workloads, you can enable the release protection feature for pay-as-you-go RDS instances. This feature prevents your RDS instance from being manually released due to unintended operations or lack of communication among team members.

Prerequisites

The RDS instance is a pay-as-you-go instance.

Precautions

The release protection feature cannot prevent the automatic release of RDS instances in normal scenarios such as the following scenarios:

- A payment in your account is overdue for more than 15 days.
- The RDS instance does not comply with the applicable security compliance policies.

Benefits of release protection

If you release an RDS instance for which the release protection feature is enabled, the following result is returned:

- If you release the RDS instance in the ApsaraDB RDS console, the "`The instance cannot be released because release protection has been enabled. Disable release protection first`" message is displayed.
- If you call the DeleteDBInstnace operation to release the RDS instance, the error code `OperationDenied.DeletionProtection` is returned.

Enable the release protection feature when you create an RDS instance

This section describes how to configure release protection settings when you create an RDS instance. For more information about how to create an RDS instance, see Create an ApsaraDB RDS for PostgreSQL instance.

1.
2. On the **Instances** page, click **Create Instance**.
3. In the **Basic Configurations** step, set **Billing Method** to **Pay-As-You-Go** and complete the remaining configurations. Click **Next: Instance Configuration**.
4. In the **Instance Configurations** step, select **Prevent release through the console or API by mistake** and complete the remaining configurations. Click **Next: Confirm Order**.
5. Complete the remaining configurations until the RDS instance is created.

Note When you can call the CreateDBInsance or CloneDBInstance operation to create an RDS instance, you can enable or disable the release protection feature for the RDS instance by setting the DeletionProtection parameter.

Modify release protection settings

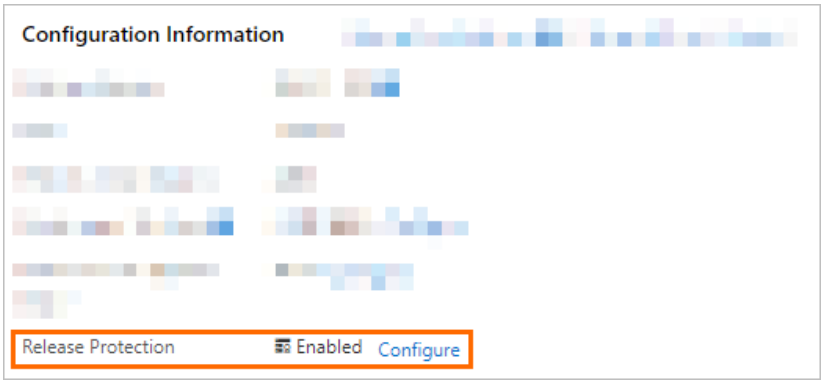
You can also enable or disable the release protection feature for an RDS instance by modifying the settings of the RDS instance.

1.
2. On the **Instances** page, find the RDS instance whose release protection settings you want to modify. In the **Actions** column, click **More** and select **Change Instance Release Protection Settings**.
3. In the Change Release Protection Setting dialog box, turn on or turn off **Release Protection**.
4. Click **OK**.

Note You can also call the ModifyDBInstanceDeletionProtection operation to enable or disable the release protection feature for an RDS instance.

Check whether the release protection feature is enabled

1.
2. On the **Basic Information** page, view the **Release Protection** section of the **Configuration Information** section.



Related operations

Operation	Description
Create an instance	Creates an ApsaraDB RDS instance.
Restore data to a new ApsaraDB RDS instance	Restores the data of an ApsaraDB RDS instance to a new instance. The new instance is also called a cloned instance.
Enable or disable the release protection feature	Enables or disables the release protection feature for an ApsaraDB RDS instance.

14.9. Manage the parameters of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to manage the parameters of an ApsaraDB RDS for PostgreSQL instance by using the ApsaraDB RDS console or the ApsaraDB RDS API. You can view the values of these parameters, reconfigure these parameters, and view the parameter reconfiguration history.

Reconfigure parameters

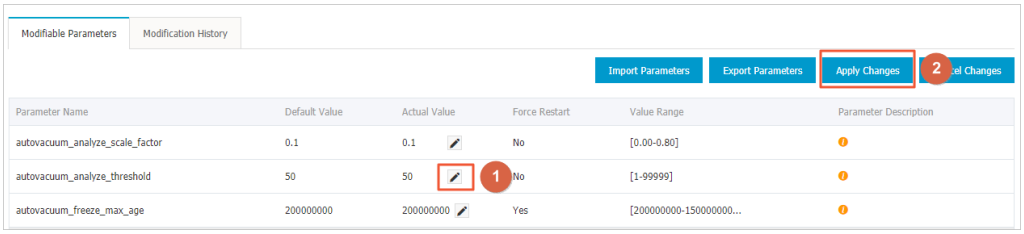
Precautions

- The reconfiguration of some parameters triggers a restart of your RDS instance. After you reconfigure these parameters and click **Apply Changes**, your RDS instance immediately restarts. To check whether the reconfiguration of a parameter triggers a restart, you need to log on to the ApsaraDB RDS console, go to the **Editable Parameters** tab, and then view the value in the **Force Restart** column for the parameter. If the value is Yes, the reconfiguration of the parameter triggers a restart. If the value is No, the reconfiguration of the parameter does not trigger a restart. The restart causes a disconnection of your application from your RDS instance. Proceed with caution. Before the restart is triggered, we recommend that you plan suitable service arrangements.
- The new values of the reconfigured parameters must be within the value ranges that are provided in the **Value Range** column on the **Editable Parameters** tab in the ApsaraDB RDS console.

Procedure

1.
2. In the left-side navigation pane, click **Parameters**.
3. On the **Editable Parameters** tab, reconfigure the parameters of your RDS instance based on your business requirements.
   - If you want to reconfigure a single parameter, perform the following steps:
     a. Find the parameter and in the Running Parameter Value column click the

        icon.
     b. Enter a new value and click **OK**.
     c. Click **Apply Changes**.
     d. In the message that appears, click **OK**.



   - If you want to reconfigure multiple parameters at a time, perform the following steps:
     a. Click **Export Parameters** to download the parameter settings of your RDS instance as a file to your computer.
     b. Open the file and reconfigure the parameters.
     c. Click **Import Parameters**.

d. In the **Import Parameters** dialog box, paste the parameters and their new values that you have copied from the file. Then, click **OK**.

e. Confirm the values of the reconfigured parameters and click **Apply Changes**.

View the parameter reconfiguration history

1.
2. In the left-side navigation pane, click **Parameters**.
3. Click the **Edit History** tab.
4. Select a time range and click **OK**.

Related operations

API	Description
Modify parameters of an ApsaraDB for RDS instance	Reconfigures the parameters of an ApsaraDB RDS instance.
Query the parameter template of an ApsaraDB for RDS instance	Queries the parameter templates that are available to an ApsaraDB RDS instance.
Query parameter configurations	Queries the parameter settings of an ApsaraDB RDS instance.

Parameters

For more information about the parameters that are supported for PostgreSQL, see the official PostgreSQL documentation.

14.10. Set the protection level of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to set the protection level of an ApsaraDB RDS for PostgreSQL instance based on your business requirements. A proper protection level increases the availability or performance of your RDS instance.

Prerequisites

Your RDS instance runs RDS High-availability Edition with standard SSDs or enhanced SSDs (ESSDs).

Context

You can configure the following parameters to set the protection level of your RDS instance: **synchronous_commit**, **rds_sync_replication_timeout**, and **synchronous_standby_names**.

The following table describes the protection levels that are provided for ApsaraDB RDS for PostgreSQL instances:

Protection level	Data replication mode	Description	Parameter configuration
Optimum performance	Asynchronous mode	This is the default protection level. This protection level delivers the highest response speed but medium data persistence.	Set the **synchronous_commit** parameter to **off**. **Note** If you set the **synchronous_commit** parameter to **off**, the **rds_sync_replication_timeout** parameter is invalid.
Optimum protection	Synchronous mode	This protection level delivers high data persistence but medium response speed.	• Set the **synchronous_commit** parameter to **remote_write**. • Set the **rds_sync_replication_timeout** parameter to **0**. • Set the **synchronous_standby_names** parameter to **standby1**. **Warning** The default value of the **synchronous_standby_names** parameter is **standby1**. If you have a self-managed secondary RDS instance, we recommend that you do not name the self-managed secondary RDS instance as **standby1**. If you name the self-managed secondary RDS instance as standby1, your RDS instance may replicate data to the self-managed secondary RDS instance in synchronous or semi-synchronous mode. As a result, data loss may occur in the event of a primary/secondary switchover.

Protection level	Data replication mode	Description	Parameter configuration
High availability	Semi-synchronous mode	This protection level balances data persistence and response speed.	• Set the **synchronous_commit** parameter to **remote_write**. • Set the **rds_sync_replication_timeout** parameter to a value that is greater than **0**. **Note** The **rds_sync_replication_timeout** parameter specifies the data synchronization timeout period in milliseconds. The value of this parameter ranges from 0 to 300000. We recommended that you set this parameter to 1000. If a synchronization times out, the high availability protection level is decreased to the optimum performance protection level, which makes your RDS instance run in asynchronous mode. After data is synchronized, the optimum performance protection level is increased to the high availability protection level, which makes your RDS instance run in semi-synchronous mode. • Set the **synchronous_standby_names** parameter to **standby1**. **Warning** The default value of the **synchronous_standby_names** parameter is **standby1**. If you have a self-managed secondary RDS instance, we recommend that you do not name the self-managed secondary RDS instance as **standby1**. If you name the self-managed secondary RDS instance as standby1, your RDS instance may replicate data to the self-managed secondary RDS instance in synchronous or semi-synchronous mode. As a result, data loss may occur in the event of a primary/secondary switchover.

Procedure

1.
2. In the left-side navigation pane, click **Parameters**.
3. On the **Editable Parameters** tab, find the **synchronous_commit**, **rds_sync_replication_timeout**, and **synchronous_standby_names** parameters.
4. In the Running Parameter Value column for each parameter, click the

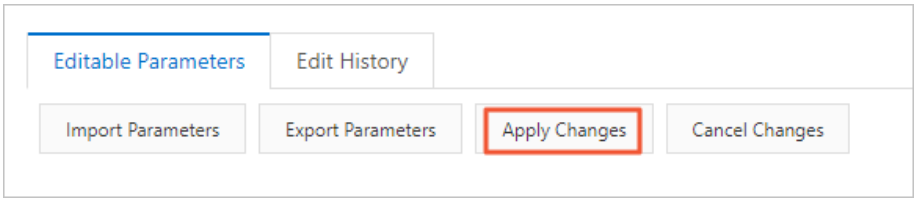
   

   icon next to the original value. In the dialog box that appears, enter the new value and click **OK**.

   > **Note** For more information about how to configure the parameters, see the "Background information" section of this topic.

5. In the upper-left corner of the page, click **Apply Changes**.

   > **Warning** After you submit the change to the value of the **rds_sync_replication_timeout** parameter, the RDS instance restarts. The restarts causes your application to disconnect from the RDS instance. We recommend that you reconfigure this parameter during off-peak hours.



14.11. Manage ApsaraDB RDS for PostgreSQL instances in the recycle bin

This topic describes how to manage ApsaraDB RDS for PostgreSQL instances in the recycle bin. You can unlock, rebuild, or destroy the RDS instances in the recycle bin.

Background information

Expired and overdue ApsaraDB RDS for PostgreSQL instances are moved to the recycle bin.

> **Notice** If you manually release a pay-as-you-go ApsaraDB RDS for PostgreSQL instance in the ApsaraDB RDS console by using the ApsaraDB RDS API, SDK, or Terraform, the RDS instance is not moved to the recycle bin and cannot be retrieved. Proceed with caution.

Rebuild an RDS instance

After a subscription RDS instance expires, it is retained for a specified period of time. After the specified retention period elapses, ApsaraDB RDS releases the RDS instance. However, the data backup files of the RDS instance can still be retained for eight more days. During the eight-day retention period, you can restore the data of the RDS instance from a data backup file to a new RDS instance by using the instance rebuild feature. For more information, see Unlock or rebuild an expired or overdue ApsaraDB RDS instance.

1. Log on to the ApsaraDB RDS console.
2. In the left-side navigation pane, click **Locked Instances**. In the top navigation bar, select the region where the RDS instance resides.

3. Find the RDS instance and click **Recreate Instance**.

By default, ApsaraDB RDS creates an RDS instance that has the same specifications in the same zone as the original RDS instance. You can also create an RDS instance that has different specifications in a different zone than the original RDS instance.

Destroy an RDS instance

If an RDS instance is locked due to expiration or overdue payments, you can destroy the RDS instance in the recycle bin.

> **Warning** After you destroy an RDS instance, all backup files except the cross-region backup files of the RDS instance are destroyed. The backup files that are destroyed include regular data backup files, archived backup files, and log backup files. Proceed with caution. For more information, see Use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance.

1. Log on to the ApsaraDB RDS console.
2. In the left-side navigation pane, click **Locked Instances**. In the top navigation bar, select the region where the RDS instance resides.
3. Find the RDS instance and click **Destroy**.

15.Babelfish for ApsaraDB RDS for PostgreSQL

15.1. Introduction to Babelfish

This topic describes the features, architecture, scenarios, and usage notes of Babelfish for ApsaraDB RDS for PostgreSQL instances.

Definition

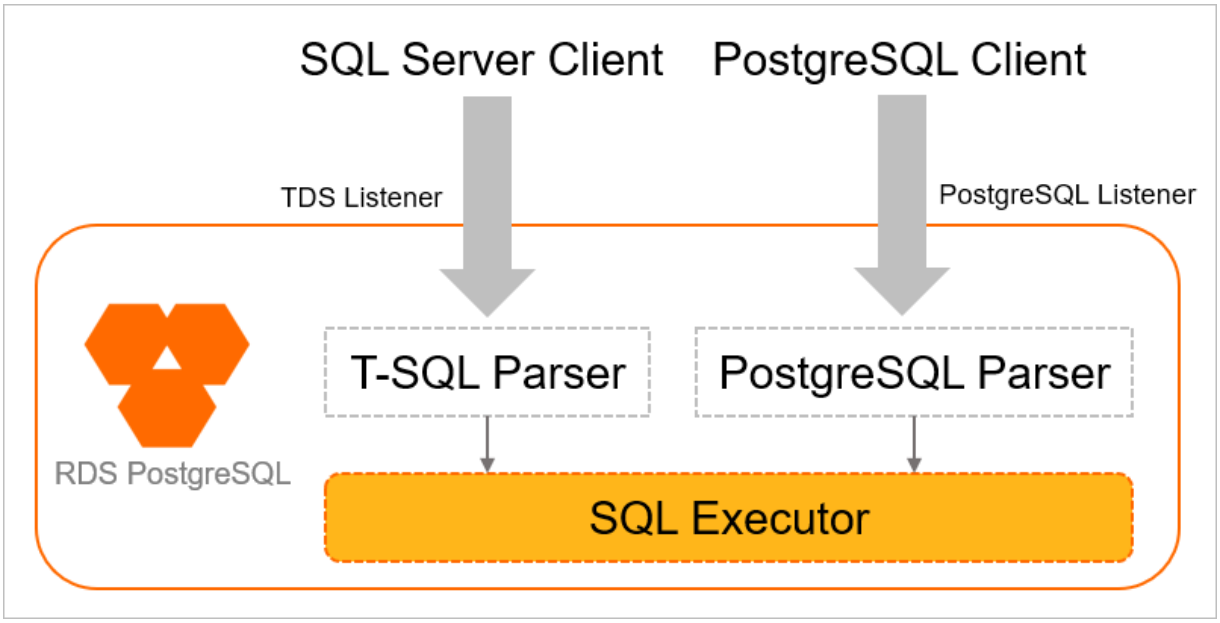
ApsaraDB RDS for PostgreSQL provides the Babelfish feature that is developed based on the Babelfish for PostgreSQL open source project. You can enable Babelfish when you create an ApsaraDB RDS for the PostgreSQL instance. After you enable Babelfish, your RDS instance can query and process data from both Microsoft SQL Server databases and PostgreSQL databases. This way, your RDS instance can parse and execute Transact-SQL (T-SQL) statements.

Babelfish supports **Tabular Data Stream (TDS), the SQL Server wire-protocol** and **T-SQL, the Microsoft SQL Server query language**. You can migrate the database of your application from an SQL Server instance to an RDS instance for which Babelfish is enabled by modifying only a few lines of code. You do not need to switch database drivers or rewrite SQL statements.

Scenarios

- You want to reduce the license costs of SQL Server.
- You want to migrate an SQL Server database to a PostgreSQL database. However, you do not want to spend a long period of time or efforts in modifying the application code.
- You want to use the open source plug-in libraries of PostgreSQL, such as the PostGIS plug-in for spatio-temporal data analysis and the TimescaleDB plug-in for time series data analysis.
- You want to reduce the costs by using a single RDS instance to process data from both PostgreSQL databases and SQL Server databases.

Architecture



Architecture description:

- An RDS instance for which Babelfish is enabled uses the following listeners to monitor requests from SQL Server clients and PostgreSQL clients over TCP ports:
  - TDS listener: receives requests over the TDS port for SQL Server. The default port number is 1433.

    > **Note** Babelfish supports TDS 7.1 or later. Microsoft SQL Server 2000 and later versions are supported.

  - PostgreSQL listener: receives requests over the port for PostgreSQL. The default port number is 5432.

    > **Note** RDS instances that run PostgreSQL 13 or 14 are supported.

- When an RDS instance for which Babelfish is enabled receives a request from the TDS listener, the RDS instance forwards the request to the T-SQL parser. Then, the T-SQL parser converts the T-SQL statements of SQL Server to an execution plan that can be processed by PostgreSQL.
- When an RDS instance for which Babelfish is enabled receives a request from the PostgreSQL listener, the RDS instance forwards the request to the PostgreSQL parser to generate an execution plan.
- The SQL executor of PostgreSQL processes and executes all plans.

If you enable Babelfish for your RDS instance, the RDS instance incorporates the capabilities of both the PostgreSQL database engine and the SQL Server database engine. Your RDS instance can process requests from SQL Server clients and PostgreSQL clients. This helps reduce costs and enhance processing capabilities.

Migration modes

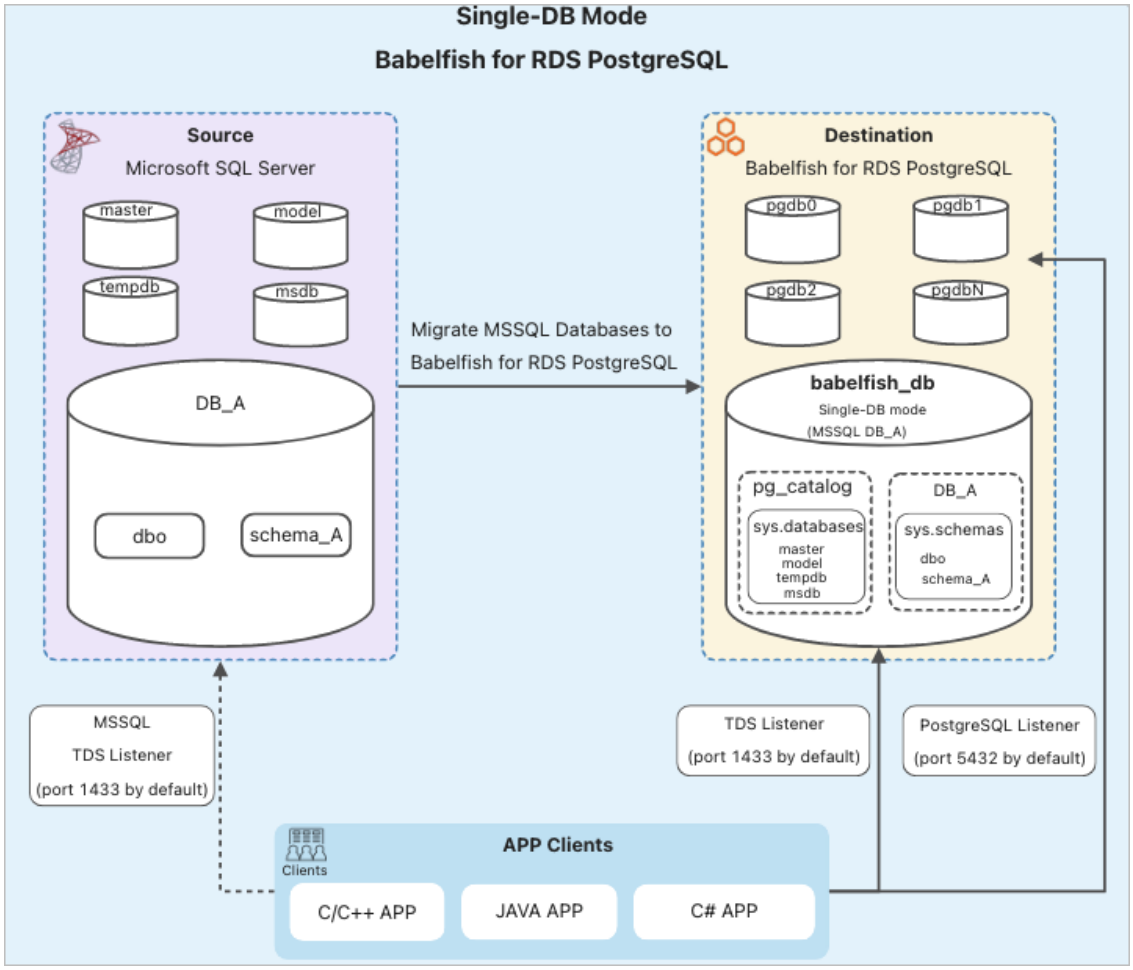
A PostgreSQL database named `babelfish_db` is provisioned for an RDS instance for which Babelfish is enabled. All migrated SQL Server objects and schemas are stored in this PostgreSQL database.

> **Note** If you connect your application to your RDS instance over the TDS port, the `babelfish_db` database is invisible.

You can use the **Single-DB** mode or **Multi-DB** mode. The mode that you choose to use affects the names of schemas in SQL Server databases and those in the `babelfish_db` database.

Single-DB mode

- **Architecture**



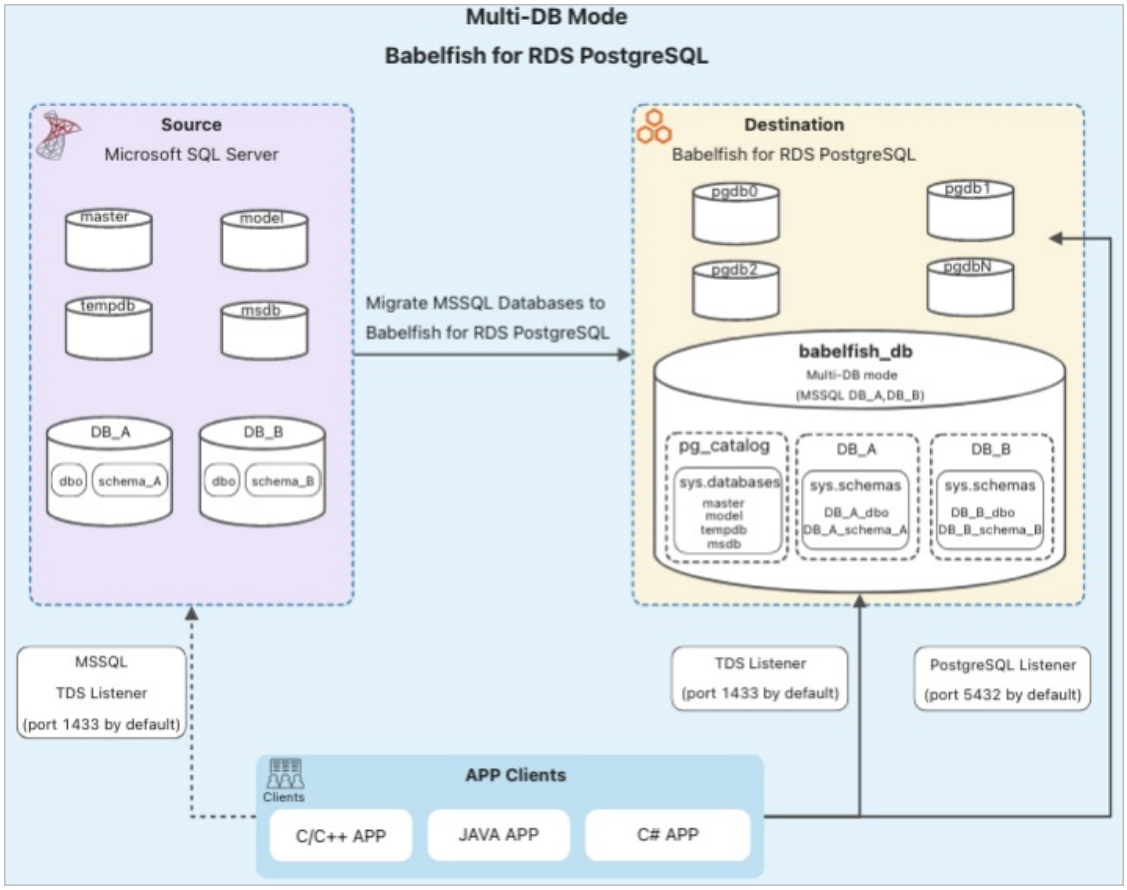
- **Description**: In this mode, the schema names of the user database in the babelfish_db database are the same as the schema names in SQL Server databases.

  For example, you create a database named DB_A over the TDS port and then create a schema named schema_A in the DB_A database.

  - The schemas named dbo and schema_A reside in the babelfish_db database of PostgreSQL.
  - The schemas named dbo and schema_A reside in the DB_A database of SQL Server.

Multi-DB mode

- **Architecture**



- **Description**: In this mode, the schema names of the user database in PostgreSQL are named in the format of `dbname_schemaname`. In SQL Server, the schema names remain unchanged.

  For example, you create a database named DB_A over the TDS port and then create a schema named schema_A in the DB_A database.

  - The schemas named DB_A_dbo and DB_A_schema_A reside in the babelfish_db database of PostgreSQL.
  - The schemas named dbo and schema_A reside in the DB_A database of SQL Server.

Pricing

The pricing of an RDS instance for which Babelfish is enabled is the same as the pricing of a regular RDS instance. No additional fees are charged. The price varies based on the instance configurations, such as the region, instance type, and storage type. For more information, visit the ApsaraDB RDS buy page.

Usage notes

- You can enable Babelfish for an RDS instance only when you create the instance. You must use the following parameter settings to create the RDS instance:
  - Database Engine: PostgreSQL 13 or 14.
  - Instance Type: a new general-purpose instance type. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types.
- Babelfish cannot be enabled for an existing RDS instance that runs PostgreSQL 13 or 14 and uses a new general-purpose instance type.
- After Babelfish is enabled, you cannot disable it.

References

- Enable Babelfish for an ApsaraDB RDS for PostgreSQL instance
- Manage Babelfish accounts
- Use clients to connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled
- Use applications to connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled

15.2. Enable Babelfish for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to enable Babelfish for an ApsaraDB RDS for PostgreSQL instance.

Usage notes

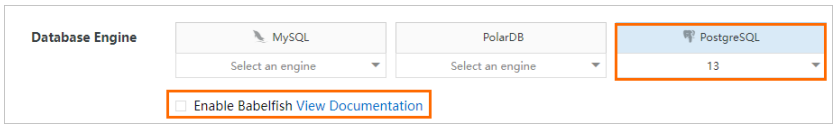
- You can enable Babelfish for an ApsaraDB RDS for PostgreSQL instance only when you create the RDS instance. The RDS instance must use the following parameter settings:
  - Database Engine: PostgreSQL 13 or 14.
  - Instance Type: a new general-purpose instance type. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types.
- You cannot enable Babelfish for existing RDS instances that run PostgreSQL 13 or 14 and use new general-purpose instance types.
- You cannot disable Babelfish after it is enabled.

Enable Babelfish for an RDS instance

1.
2. Click **Create Instance** to go to the ApsaraDB RDS buy page.
3. Configure the parameters.

   This topic describes the key parameters that are used to enable Babelfish for an RDS instance when you create the RDS instance. You must configure the other parameters in the same manner as you configure the parameters for a regular RDS instance. For more information, see Create an ApsaraDB RDS for PostgreSQL instance.

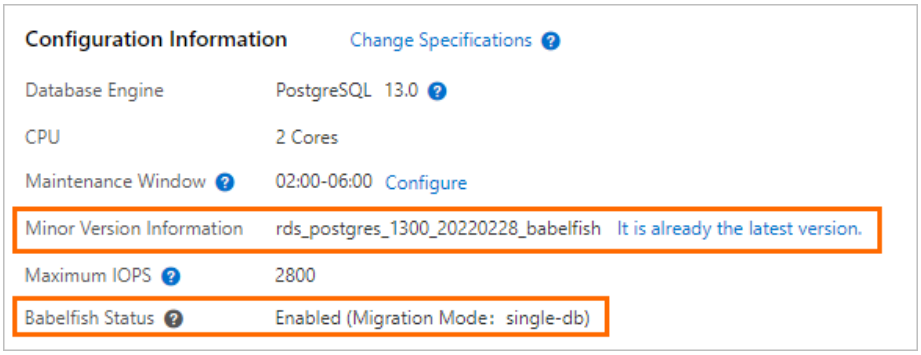
   The following table describes the key parameters.

Parameter	Description
Database Engine	The database engine of the RDS instance. Set the value to PostgreSQL 13 or 14. Select **Enable Babelfish**. 
Instance Type	The instance type of the RDS instance. Only new general-purpose instance types support Babelfish. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types.

Parameter	Description
Babelfish Migration Mode	The migration mode after you enable Babelfish. This parameter takes effect only if you select **Enable Babelfish** in the Basic Configurations step. ○ **single-db**: You can create only one SQL Server database on an RDS instance for which Babelfish is enabled and create a standard PostgreSQL schema for the database. ○ **multi-db**: You can create multiple SQL Server databases and create different PostgreSQL schemas for the databases. You must name the schemas in the <Database name>_<Schema name> format to prevent name conflicts. For more information about how to select a migration mode, see Migration modes.
Initial Account	The username of the management account that can be used to manage Babelfish or PostgreSQL. **Notice** This account is a privileged account. You cannot delete the account after it is created. Username requirements: ○ The username must be 2 to 63 characters in length. ○ The username can contain lowercase letters, digits, and underscores (_). ○ The username must start with a lowercase letter and end with a lowercase letter or a digit. ○ The username cannot start with pg. ○ The username cannot contain SQL keywords. For more information, see SQL keywords.
Password	The password of the management account. **Note** You can change the password after the RDS instance is created. For more information, see Reset the password of an account on an ApsaraDB RDS for PostgreSQL instance. Password requirements: ○ The password must be 8 to 32 characters in length. ○ The password must contain at least three of the following character types: uppercase letters, lowercase letters, digits, and special characters. ○ The following special characters are supported: `! @ # $ % ^ & * ( ) _ + - =` .

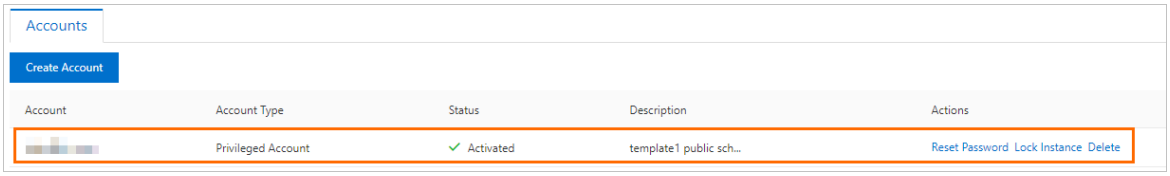
View the status of Babelfish

1.
2. In the left-side navigation pane, click **Basic Information**.
3. In the **Configuration Information** section, view the values of **Minor Version Information** and **Babelfish Status**.



View the Babelfish management account

1.
2. In the left-side navigation pane, click **Accounts**.
3. On the **Accounts** tab, view the Babelfish management account that is initialized when you created the RDS instance.



View the endpoints and TDS port

Prerequisites

A whitelist is created for the RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

Procedure

1.
2. In the left-side navigation pane, click **Database Connection**.

   In the Database Connection section of the page that appears, you can view the internal and public endpoints and Tabular Data Stream (TDS) port.

> **Note**
>
> - If you want to use a public endpoint, click **Apply for Public Endpoint**. For more information, see Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.
> - If you want to change the TDS port number for Babelfish, click **Change Endpoint**. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.

Related operations

Operation	Description
CreateDBInstance	Creates an instance.
DescribeDBInstanceAttribute	Queries the details of an instance.
AllocateInstancePublicConnection	Applies for a public endpoint for an instance.
ModifyDBInstanceConnectionString	Changes the endpoint of an instance.
DescribeDBInstanceNetInfo	Queries the endpoints of an instance.

15.3. Manage Babelfish accounts

When you create an ApsaraDB RDS for PostgreSQL instance and enable Babelfish for the RDS instance, the management account is initialized and can be used to manage Babelfish or PostgreSQL. After the RDS instance is created, you can manually create a Babelfish management account. You can also create standard accounts after you connect your application to the RDS instance over the Tabular Data Stream (TDS) port.

Create a Babelfish management account

1.
2. In the left-side navigation pane, click **Accounts**.
3. Click **Create Account**. In the panel that appears, set the Account Type parameter to **Privileged Account**.

> **Note**
>
> - This topic describes how to create a Babelfish management account and how to grant logon permissions to the account. The parameters that are used to create a Babelfish management account are the same as the parameters that are used to create an account for the RDS instance. For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance.
> - In this example, a Babelfish management account named **babelfish_user** is created.

4. Run the following command to log on to the RDS instance by using the **babelfish_user** account:

```
psql -h <Endpoint of the RDS instance> -p 5432 -U babelfish_user -d babelfish_db
```

> **Note** For more information about how to obtain the endpoint of the RDS instance, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.

5. Run the following command to grant permissions to the **babelfish_user** account to connect to the RDS instance over the TDS port:

```
call sys.babel_initialize_logins('babelfish_user');
GRANT sysadmin to babelfish_user;
```

> **Note** The sysadmin account in PostgreSQL is used in a similar manner to the SA account in SQL Server.

Create a Babelfish standard account

After you connect your application to the RDS instance over the TDS port, you can create a standard account.

1. Connect your application to the RDS instance over the TDS port.

   > **Note** If you want to connect your application to the RDS instance over the TDS port, an SQL Server client is required. For more information about how to download an SQL Server client and how to configure connection parameters, see Use clients to connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled. In this example, sqlcmd is used.

   ```
   sqlcmd -S pgm-****.pg.rds.aliyuncs.com,1433 -U babelfish_user
   ```

2. Execute the following statements to create a standard account:

   ```
   -- Creates the login test_babelfish with password 'Test123456!'.
   CREATE LOGIN test_babelfish
       WITH PASSWORD = 'Test123456!';
   GO
   -- Creates a database user for the login created above.
   CREATE USER test_babelfish FOR LOGIN test_babelfish;
   GO
   ```

   > **Note** The method that is used to create standard accounts in this step is an example. For information about more methods, see CREATE USER (Transact-SQL).

3. View the information about the account that you want to use to connect to the RDS instance over the TDS port.

   > **Note** You cannot create, view, modify, or delete the accounts that are connected over the TDS port in the ApsaraDB RDS console.

   ```
   SELECT name
   FROM sys.server_principals;
   GO
   ```

   The following result is returned:

   ```
   1> SELECT name FROM sys.server_principals;
   2> GO
   name
   ----------------------------------------
   alicloud_rds_admin
   babelfish_user
   test_babelfish

   (3 rows affected)
   ```

Related operations

Operation	Description
CreateAccount	Creates a database account.

15.4. Connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled

15.4.1. Use clients to connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled

This topic describes how to use clients, such as TSQL (FreeTDS), sqlcmd, SQL Server Management Studio (SSMS), and Azure Data Studio, to connect to an ApsaraDB RDS for PostgreSQL instance for which Babelfish is enabled over the Tabular Data Stream (TDS) port.

If you want to connect to the RDS instance over the PostgreSQL port, see Connect to an ApsaraDB RDS for PostgreSQL instance.

Prerequisites

- An ApsaraDB RDS for PostgreSQL instance for which Babelfish is enabled is created. For more information, see Enable Babelfish for an ApsaraDB RDS for PostgreSQL instance.
- A Babelfish account is created. For more information, see Manage Babelfish accounts.
- A whitelist is configured to allow the server on which the client resides to access the RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
- The endpoints and TDS port of the RDS instance are obtained. For more information, see View the endpoints and TDS port.

Use TSQL (FreeTDS) to connect to the RDS instance

1. Run the following command to install TSQL (FreeTDS). In this example, CentOS 7 is used.

   ```
   yum install -y freetds
   ```

2. Run the following command to connect to the RDS instance:

   ```
   tsql -S pgm-****.pg.rds.aliyuncs.com -p 1433 -U babelfish_user
   ```

   The following table describes the parameters.

Parameter	Example	Description
-S	pgm-****.pg.rds.aliyuncs.com	The endpoint that is used to connect to the RDS instance.
-p	1433	The TDS port number.

Parameter	Example	Description
-U	babelfish_user	The username of the Babelfish account.

3. Perform a simple SQL query. For more information, see Common operations and compatibility description.

```
SELECT name FROM sys.databases;
GO
```

Use sqlcmd to connect to the RDS instance

1. Download and install sqlcmd. For more information, see Download and install sqlcmd in the official SQL Server documentation.
2. Run the following command to connect to the RDS instance:

```
sqlcmd -S pgm-****.pg.rds.aliyuncs.com,1433 -U babelfish_user
```

The following table describes the parameters.

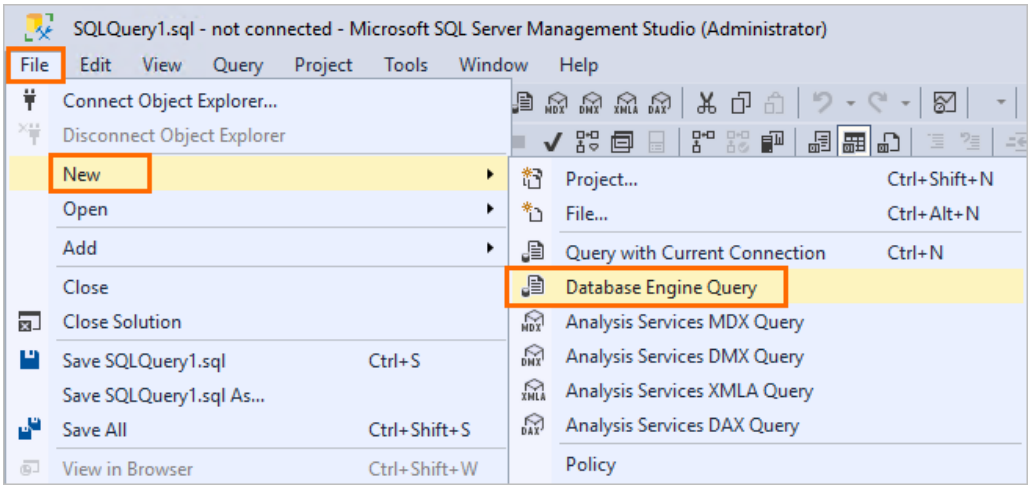
Parameter	Example	Description
-S	pgm-****.pg.rds.aliyuncs.com,1433	The endpoint and port number that are used to connect to the RDS instance. The value is in the format of `Endpoint,Port number`.
-U	babelfish_user	The username of the Babelfish account.

3. Perform a simple SQL query. For more information, see Common operations and compatibility description.

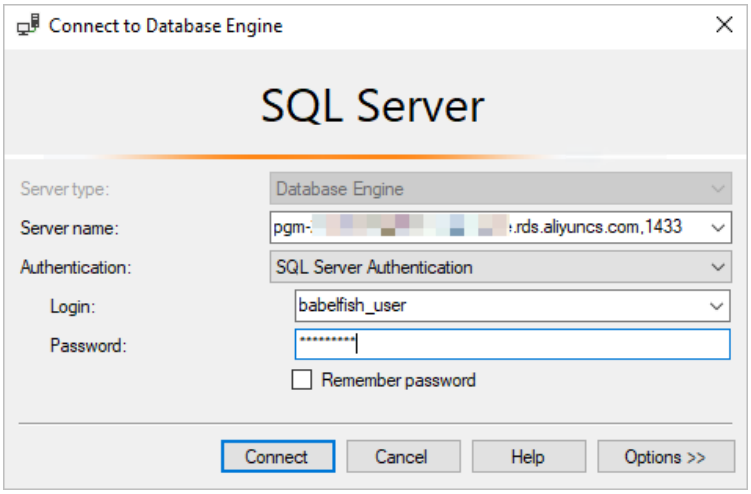
```
SELECT name FROM sys.databases;
GO
```

Use SSMS to connect to the RDS instance

1. Download and install SSMS. For more information, see Download SSMS in the official SQL Server documentation.
2. Open the SSMS client. In the menu bar, choose **File > New > Database Engine Query**.



3. Configure the parameters and click **Connect**.



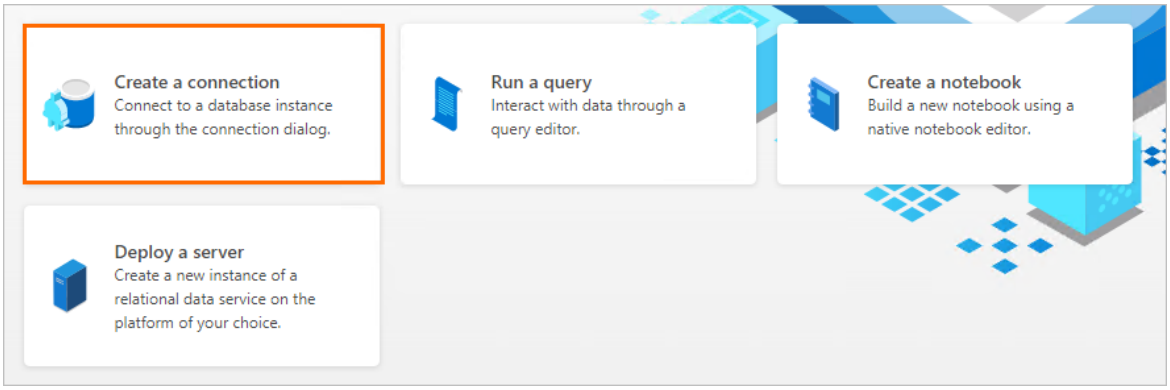
The following table describes the parameters.

Parameter	Example	Description
Server type	Database Engine	The value of this parameter is fixed as Database Engine.
Server name	pgm-****.pg.rds.aliyuncs.com,1433	The endpoint and port number that are used to connect to the RDS instance. The value is in the format of `Endpoint,Port number`.
Authentication	SQL Server Authentication	The value of this parameter is fixed as SQL Server Authentication.
Login	babelfish_user	The username and password of the Babelfish account.
Password	babelfish_pwd	

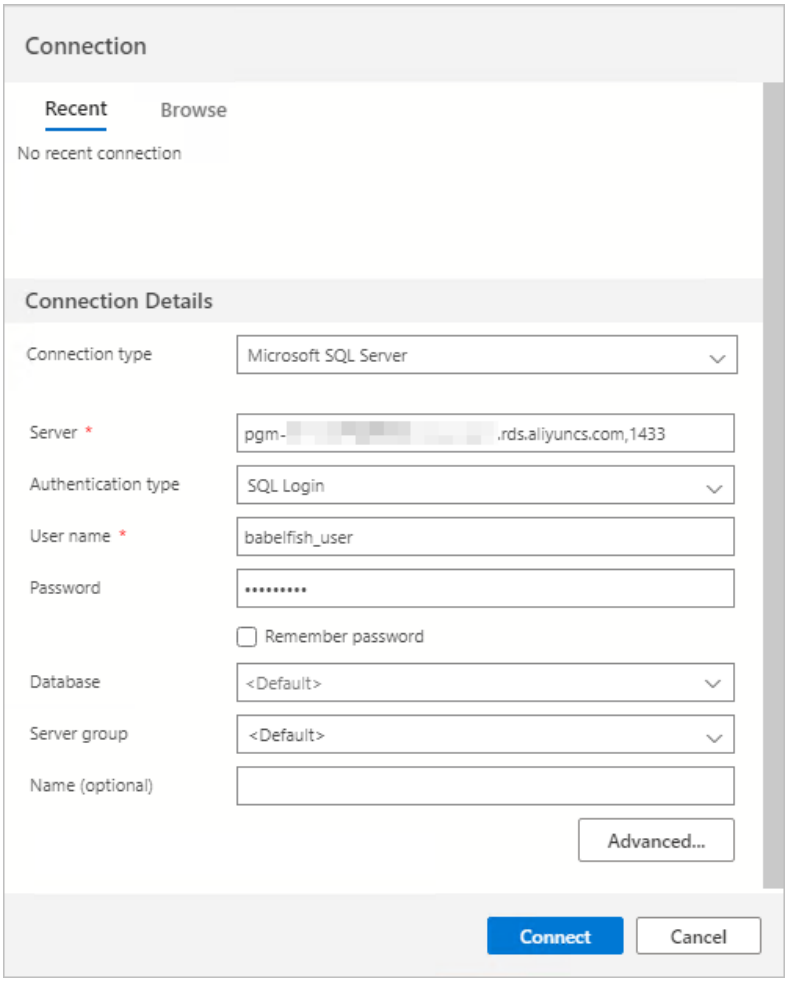
> **Note** If `Specified case is not valid. (Micosoft.SqlServer.ConnectionInfo)` is displayed during the connection, handle the issue based on FAQ.

Use Azure Data Studio to connect to the RDS instance

1. Download and install Azure Data Studio. For more information, see Download and install Azure Data Studio in the official SQL Server documentation.
2. Open Azure Data Studio and click **Create a connection**.



3. Configure the parameters and click **Connect**.



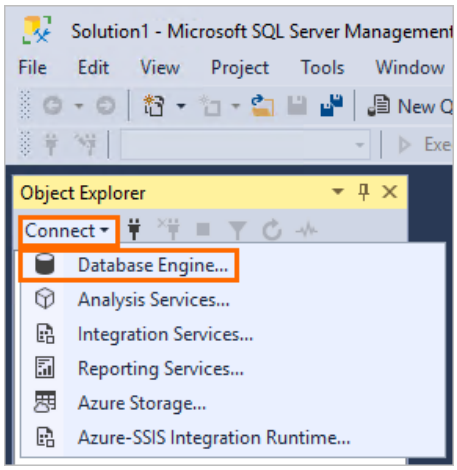
The following table describes the parameters.

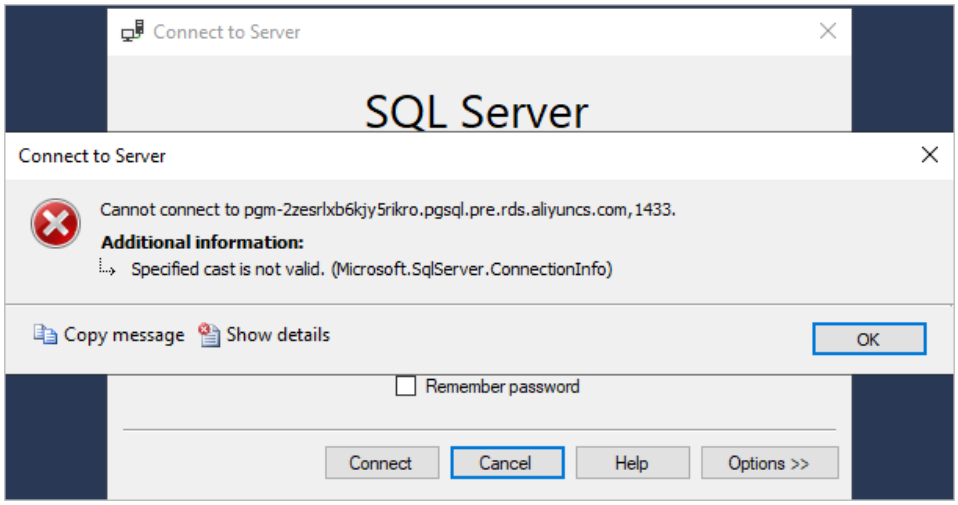
Parameter	Example	Description
Connection type	Microsoft SQL Server	The value of this parameter is fixed as Microsoft SQL Server.
Server	pgm-****.pg.rds.aliyuncs.com,1433	The endpoint and port number that are used to connect to the RDS instance. The value is in the format of `Endpoint,Port number`.
Authentication type	SQL Login	The value of this parameter is fixed as SQL Login.
User name	babelfish_user	The username and password of the Babelfish account.
Password	babelfish_pwd	

FAQ

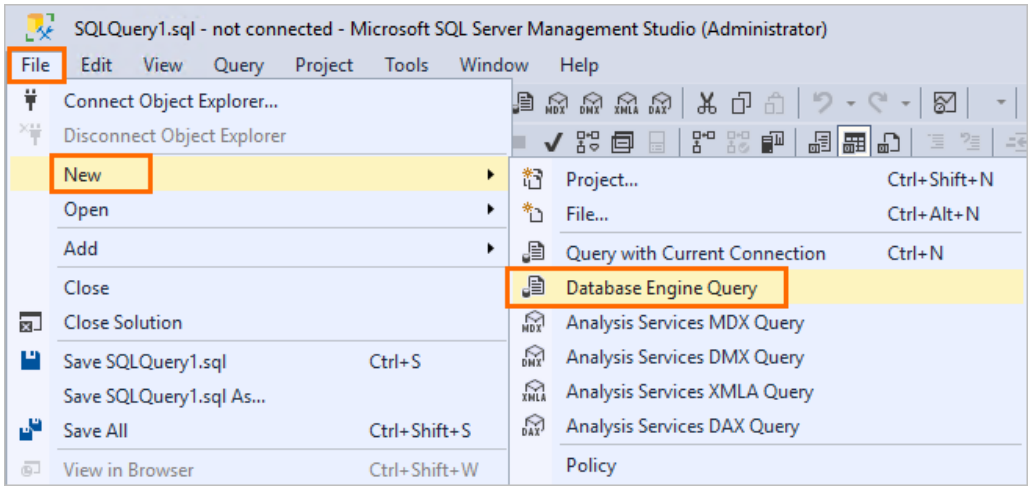
What do I do if the `Specified case is not valid. (Micosoft.SqlServer.ConnectionInfo)` error message is displayed when I use SSMS to connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled?

- Cause: Babelfish is not fully compatible with all system tables of SQL Server. You cannot create a connection by using **Object Explorer**.





- Solution: In the menu bar, choose **File > New > Database Engine Query** to create a connection. For more information, see Use SSMS to connect to the RDS instance.



15.4.2. Use applications to connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled

This topic describes how to use a C#, Java, Python, or C application to connect to an ApsaraDB RDS for PostgreSQL instance for which Babelfish is enabled over the Tabular Data Stream (TDS) port.

Prerequisites

- An ApsaraDB RDS for PostgreSQL instance for which Babelfish is enabled is created. For more information, see Enable Babelfish for an ApsaraDB RDS for PostgreSQL instance.
- A Babelfish account is created. For more information, see Manage Babelfish accounts.
- A whitelist is configured to allow the server on which your client resides to access the RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
- The endpoints and TDS port of the RDS instance are obtained. For more information, see View the endpoints and TDS port.

Preparations

1. Connect to the RDS instance. For more information, see Use clients to connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled.
2. Create a test database.

```
create database sqlserverdb;
```

> **Note** If you use the **Single-DB** migration mode, you can create only one database. If you have created a database, you cannot create another database. For more information about migration modes, see View the status of Babelfish.

3. Create a test table.

```
USE sqlserverdb
GO
CREATE TABLE dbo.tb_test(
    id int not null IDENTITY(1,1) PRIMARY KEY,
    name varchar(50))
GO
```

Use a C# application to connect to the RDS instance

In this example, the C# application is configured in a Windows Server system. For more information about how to configure the C# application in other operating systems, see the tutorial in official .NET documentation.

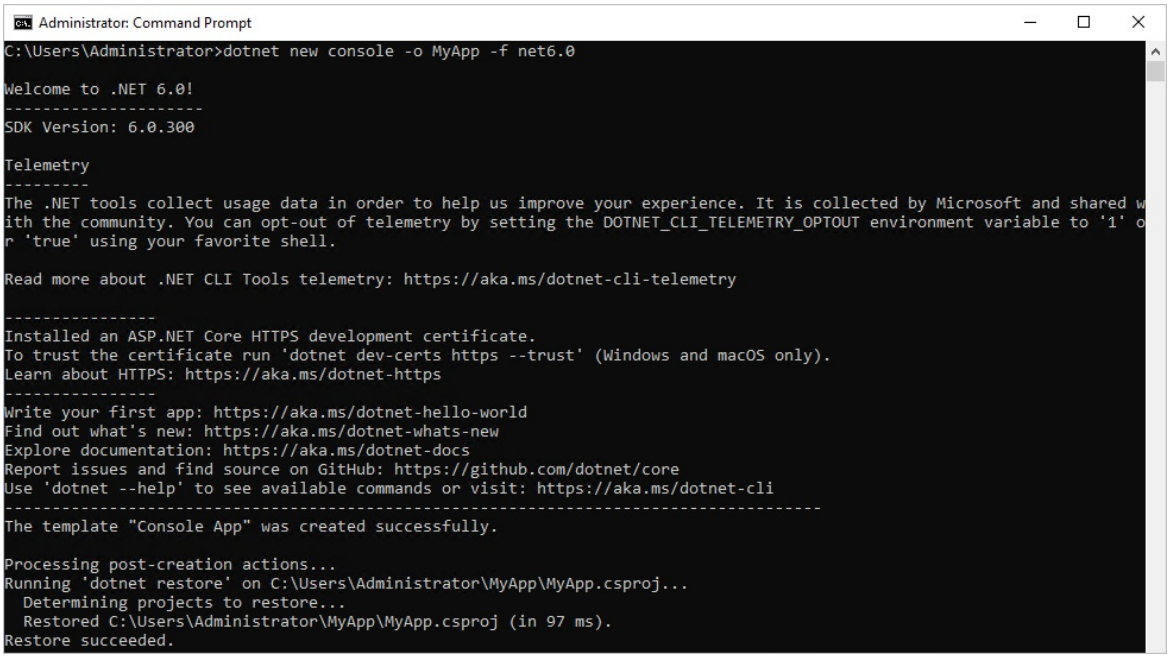
Environment preparations

.NET 6 SDK (64-bit) is installed. For more information, see Download and install.

Procedure

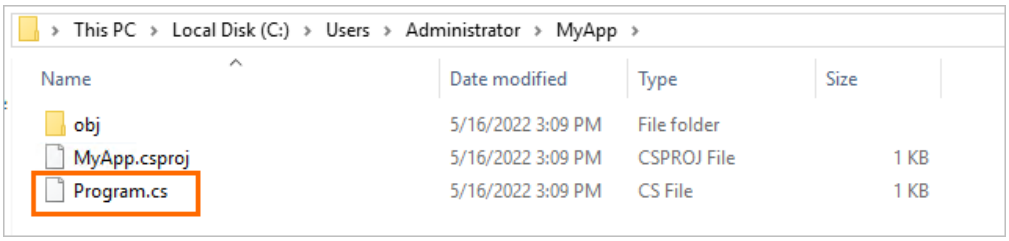
1. In the Windows Server desktop, press **Win+Q** to open the search box, enter **cmd**, and then press **Enter** to open Command Prompt. Then, run the following command to create a project:

```
dotnet new console -o <The name of the project> -f net6.0
```



2. Go to the directory in which the project resides and edit the **Program.cs** file.

   You can obtain the directory from the command output in Step 1. In this example, the *C:\Users\Administrator\MyApp* directory is used.



3. Copy the following sample code and paste the code into the **Program.cs** file.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
namespace sample
{
    class Program
    {
        static void Main(string[] args)
        {
            // Setting up MSSQL Credentials
            SqlConnection con;
            //Configure the endpoint, the TDS port, the username and password of the Babelfish account, and the name of the database that are used to connect to the RDS instance.

            string conString = "Server=" + @"pgm-****.pg.rds.aliyuncs.com,1433" + ";" +
                               "User id=" + "babelfish_user" + ";" +
                               "Password=" + "babelfish_pwd" + ";" +
                               "Database=" + "sqlserverdb" + ";" +
                               "MultipleActiveResultSets=true;";
```

```
            con = new SqlConnection(conString);
            SqlCommand cmd = new SqlCommand();
            // Creating MSSQL Connection
            try
            {
                con.Open();
                Console.WriteLine("Connection established\n") ;
            }
            catch
            {
                Console.WriteLine("Can not connect to database!\nPlease check credentials
!");
                Environment.Exit(1);
            }
            string sqlQuery = "";
            // Select values example
            select_all(con);
            // Transaction example
            // Insert values into sample table
            cmd = con.CreateCommand();
            SqlTransaction transaction = con.BeginTransaction("SampleTransaction");
            try
            {
                sqlQuery = "insert into dbo.tb_test(name) values(@name)";
                cmd.CommandType = System.Data.CommandType.Text;
                cmd.CommandText = sqlQuery;
                cmd.Transaction = transaction;
                cmd.Parameters.AddWithValue("@name", "A");
                cmd.ExecuteNonQuery();
                cmd.Parameters.Clear();
                cmd.Parameters.AddWithValue("@name", "B");
                cmd.ExecuteNonQuery();
                cmd.Parameters.Clear();
                cmd.Parameters.AddWithValue("@name", "C");
                cmd.ExecuteNonQuery();
                cmd.Parameters.Clear();
                cmd.Parameters.AddWithValue("@name", "D");
                cmd.ExecuteNonQuery();
                transaction.Commit();
                Console.WriteLine("\nInsert successful!\n");
            }
            catch
            {
                transaction.Rollback();
                Console.WriteLine("\nInsert failed!\n");
            }
            select_all(con);
            // Removing inserted values
            sqlQuery = "delete from dbo.tb_test";
            cmd = con.CreateCommand();
            cmd.CommandText = sqlQuery;
            int row_count = cmd.ExecuteNonQuery();
            // Select metadata
            // Select row count from delete
            Console.WriteLine("\nDeleted rows: " + row_count + "\n");
```

```
            // Select column names from table
            sqlQuery = "SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS WHERE TABLE_NA
ME = 'dbo.tb_test'";
            cmd = con.CreateCommand();
            cmd.CommandText = sqlQuery;
            SqlDataReader reader = cmd.ExecuteReader();
            string value = "";
            while (reader.Read())
            {
                value += reader.GetValue(0) + " ";
            }
            Console.WriteLine(value);
            reader.Close();
            // Closing connection
            con.Close();
            Console.WriteLine("\nConnection closed!");
        }
        private static void select_all(SqlConnection con)
        {
            string sqlQuery = "select id,name from dbo.tb_test order by id";
            SqlCommand cmd = con.CreateCommand();
            cmd.CommandText = sqlQuery;
            SqlDataReader reader = cmd.ExecuteReader();
            while (reader.Read())
            {
                string value = "";
                for (int i = 0; i != reader.FieldCount; i++)
                {
                    value += reader.GetValue(i) + " ";
                }
                Console.WriteLine(value);
            }
            reader.Close();
        }
    }
}
```

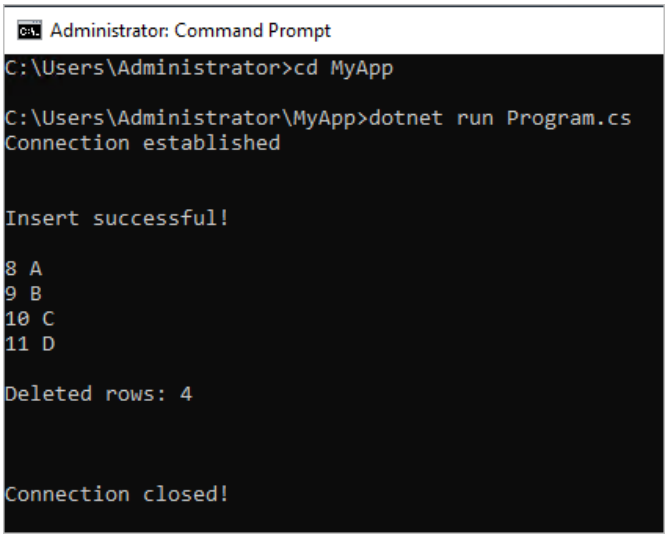
4. Go to the directory in which the project resides and add the following dependencies to the **MyApp.csproj** file:

```
<ItemGroup>
    <PackageReference Include="System.Data.SqlClient" Version="4.8.3" />
</ItemGroup>
```

5. Open Command Prompt and switch to the directory in which the project resides. Then, run the following command to run the C# application:

```
cd MyAPP
dotnet run Program.cs
```

Command output:



Use a Java application to connect to the RDS instance

In this example, your Java project is set up by using Maven.

Environment preparations

Java 1.8 or later is installed.

Procedure

1. Add the following dependencies to the pom.xml file of your Maven project:

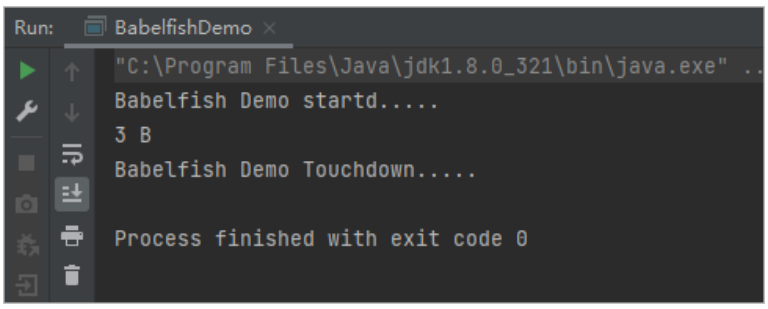
```
<dependency>
  <groupId>com.microsoft.sqlserver</groupId>
  <artifactId>mssql-jdbc</artifactId>
  <version>9.4.0.jre8</version>
</dependency>
```

2. Connect to the RDS instance over Java Database Connectivity (JDBC).

```
public class BabelfishDemo {
    public static Connection getRdsPgConnection(){
        //The endpoint that is used to connect to the RDS instance.
        String rdsPgConnStr = "pgm-****.pg.rds.aliyuncs.com";
        //The TDS port.
        String rdsPgPort = "1433";
        //The name of the database.
        String databaseName = "sqlserverdb";
        //The username of the Babelfish account.
        String userName = "babelfish_user";
        //The password of the Babelfish account.
        String password = "babelfish_pwd";
        String connectionUrl = String.format("jdbc:sqlserver://%s:%s;databaseName=%s;user
=%s;password=%s;connectTimeout=600;socketTimeout=600", rdsPgConnStr, rdsPgPort, databaseN
ame, userName, password);
        Connection connection = null;
        try{
            connection = DriverManager.getConnection(connectionUrl);
        }
```

```
        catch (Exception exception) {
            exception.printStackTrace();
        }
        return connection;
    }
    public static void insertRecord(String name, Connection dbConnObj){
        try{
            PreparedStatement stmt = dbConnObj.prepareStatement("delete from dbo.tb_test;
insert into dbo.tb_test(name) values(?)");
            stmt.setString(1, name);
            stmt.execute();
        } catch (Exception exception){
            exception.printStackTrace();
        }
    }
    public static void queryDataRecords(Connection dbConnObj){
        try (Statement stmt = dbConnObj.createStatement()) {
            String SQL = "select * from dbo.tb_test order by id;";
            ResultSet rs = stmt.executeQuery(SQL);
            // Iterate through the data in the result set and display it.
            while (rs.next()) {
                System.out.println(rs.getString("id") + " " + rs.getString("name"));
            }
        }
        // Handle any errors that may have occurred.
        catch (SQLException e) {
            e.printStackTrace();
        }
    }
    public static void main(String[] args) {
        System.out.println("Babelfish Demo startd.....");
        // get connection
        Connection dbConnObj = getRdsPgConnection();
        // write record to db
        insertRecord("B", dbConnObj);
        // query data
        queryDataRecords(dbConnObj);
        System.out.println("Babelfish Demo Touchdown.....");
    }
}
```

Command output:



Use a Python application to connect to the RDS instance

In this example, the Python application is configured in CentOS 7.9.

Environment preparations

The required dependencies are installed.

```
yum install gcc gcc-c++ -y
wget https://packages.microsoft.com/config/centos/7/packages-microsoft-prod.rpm
rpm -ivh packages-microsoft-prod.rpm
yum install msodbcsql17.x86_64 -y
pip3 install pyodbc
```

Procedure

1. Run the **vim** command to create a file. For example, you can run the following command to create a file named **main01.py**.

   ```
   vim main01.py
   ```

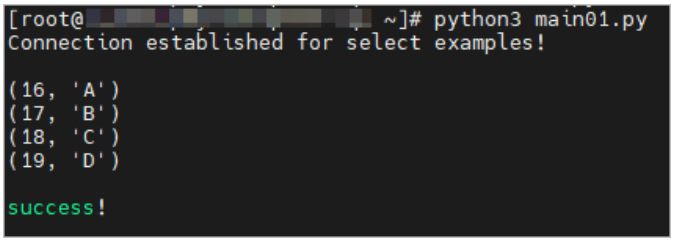
2. Enter **i** to enter the insert mode. Copy the following sample code and paste the code into the **main01.py** file.

   ```
   import sys
   import os
   import pyodbc
   # Configure the endpoint, the TDS port, the username and password of the Babelfish accoun
   t, and the name of the database that are used to connect to the RDS instance.
   server = 'pgm-*****.pg.rds.aliyuncs.com,1433'
   database = 'sqlserverdb'
   username = 'babelfish_user'
   password = 'babelfish_pwd'
   # Trying to establish connection
   connection, cursor = None, None
   try:
       connection = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};SERVER='+server+'
   ;DATABASE='+database+';UID='+username+';PWD='+ password)
       cursor = connection.cursor()
       print("Connection established for select examples!\n")
   except pyodbc.ProgrammingError:
       print("Cannot connect to the database!\nPlease check credentials!")
       exit(1)
   sql = "insert into dbo.tb_test(name) values('A'),('B'),('C'),('D')"
   cursor.execute(sql)
   # Select values
   cursor.execute("select id,name from dbo.tb_test order by id")
   for row in cursor.fetchall():
       print(row)
   sql = "delete from dbo.tb_test"
   cursor.execute(sql)
   cursor.close()
   connection.close()
   print("\nsuccess!\n")
   ```

3. Press **Esc** to exit the insert mode and enter **:wq** to save and close the file.
4. Run the following command to run the Python application:

   ```
   python3 main01.py
   ```

Command output:



Use a C application to connect to the RDS instance

In this example, the C application is configured in CentOS 7.9.

Environment preparations

The required dependencies are installed.

```
yum install freetds freetds-devel unixODBC-devel -y
```

Procedure

1. Run the **vim** command to create a file. For example, you can run the following command to create a file named **main01.c**:

   ```
   vim main01.c
   ```

2. Enter **i** to enter the insert mode. Copy the following sample code and paste the code into the **main01.c** file.

   ```
   #include <stdio.h>
   #include <stdlib.h>
   #include <unistd.h>
   #include <sys/param.h>
   #include <sybfront.h>
   #include <sybdb.h>
   #include <syberror.h>
   #define  DBNAME    "sqlserverdb"     // The name of the database.
   #define  UID       "babelfish_user"  // The username of the Babelfish account.
   #define  PWD       "babelfish_pwd"   // The password of the Babelfish account.
   #define  DBSERVER  "pgm-*****.pg.rds.aliyuncs.com"   // The endpoint that is used to conn
   ect to the RDS instance.
   #define  TDSPORT   1433
   /* handler from messages from the server */
   static int
   msg_handler(DBPROCESS* dbproc, DBINT msgno, int msgstate, int severity,
       char *msgtext, char *srvname, char *procname, int line)
   {
       /* regular errors are handled by the error handler */
       if      (severity < 11)
               fprintf(stderr, "Server message (severity %d): %s\n", severity, msgtext);
       return 0;
   }
   /* error handler */
   static int err_handler(DBPROCESS* dbproc, int severity, int dberr, int oserr, char *dberr
   str, char *oserrstr)
   {
   ```

```
    fprintf(stderr, "Server error %d: %s\n", dberr, dberrstr);
    if     (oserr != 0)
            fprintf(stderr, "Caused by system error %d: %s\n", oserr, oserrstr);
    return INT_CANCEL;
}
int main(void)
{
      LOGINREC     *login;
      DBPROCESS     *dbconn;
      char         hostname[MAXHOSTNAMELEN];
      int         max_len = MAXHOSTNAMELEN;
      DBCHAR         accession[10];
      DBCHAR         examdesc[10];
      DBCHAR         examcode[255];
    char         portstr[20];
    int         rc;
    char        sql[65535];
      if (dbinit() == FAIL)
    {
            fprintf(stderr, "Could not init db.\n");
            return 1;
      }
      /* Allocate a login params structure */
    if     ((login = dblogin()) == FAIL)
    {
        fprintf(stderr, "Could not initialize dblogin() structure.\n");
            return 2;
      }
    /* Initialize the login params in the structure */
    DBSETLUSER(login, UID);
    DBSETLPWD(login, PWD);
    if     (gethostname(hostname, max_len) == 0)
    {
            DBSETLHOST(login, hostname);
        fprintf(stderr, "setting login hostname: %s\n", hostname);
    }
    /* the port can only be set via environment variable */
    rc = snprintf(portstr, 20, "TDSPORT=%d", TDSPORT);
    if     (rc < 0 || rc >= 20)
    {
            fprintf(stderr, "error composing string for environment variable TDSPORT\n");
            return 0;
    }
    if     (putenv(portstr) != 0)
    {
            fprintf(stderr, "error setting TDSPORT environment variable\n");
            return 0;
    }
    /* install error handler */
    dberrhandle(err_handler);
    dbmsghandle(msg_handler);
    /* Now connect to the DB Server */
    if     ((dbconn = dbopen(login, DBSERVER)) == NULL)
    {
        fprintf(stderr, "Could not connect to DB Server: %s\n", DBSERVER);
            return 3;
```

```
        return 3;
}
/* Use database which you want to operate */
if  (dbuse(dbconn, DBNAME) == FAIL)
{
    fprintf(stderr, "Could not use database: %s\n", DBNAME);
        return 4;
}
/* Prepare sql */
snprintf(sql, 65535, "insert into dbo.tb_test(name) values('A'),('B'),('C'),('D')");
if  (dbcmd(dbconn, sql) == FAIL)
{
    fprintf(stderr, "Could not prepare sql: %s\n", sql);
        return 5;
}
/* Execute sql */
if  (dbsqlexec(dbconn) == FAIL)
{
       fprintf(stderr, "Could not execute sql: %s\n", sql);
        return 6;
}
/* Judge sql execute result */
if  (dbresults(dbconn) != SUCCEED)
{
       fprintf(stderr, "Could not execute sql: %s\n", sql);
        return 7;
}
/* Prepare sql */
snprintf(sql, 65535, "select id,name from dbo.tb_test order by id");
if  (dbcmd(dbconn, sql) == FAIL)
{
    fprintf(stderr, "Could not prepare sql: %s\n", sql);
        return 8;
}
/* Execute sql */
if  (dbsqlexec(dbconn) == FAIL)
{
       fprintf(stderr, "Could not execute sql: %s\n", sql);
        return 9;
}
/* Fetch sql execute result */
int             retcode;
char            id[65535];
char            name[65535];
if  ((retcode = dbresults(dbconn)) != NO_MORE_RESULTS && retcode == SUCCEED)
{
    dbbind(dbconn, 1, CHARBIND, (DBCHAR)0, (BYTE*)id);
    dbbind(dbconn, 2, CHARBIND, (DBCHAR)0, (BYTE*)name);
    while (dbnextrow(dbconn) != NO_MORE_ROWS)
    {
        printf("id: %s, name: %s\n", id, name);
    }
}  else
{
    fprintf(stderr, "Could not fetch result for sql: %s\n", sql);
        return 10;
```

```
    }
    /* Prepare sql */
    snprintf(sql, 65535, "delete from dbo.tb_test");
    if  (dbcmd(dbconn, sql) == FAIL)
    {
        fprintf(stderr, "Could not prepare sql: %s\n", sql);
            return 11;
    }
    /* Execute sql */
    if  (dbsqlexec(dbconn) == FAIL)
    {
           fprintf(stderr, "Could not execute sql: %s\n", sql);
            return 12;
    }
    /* Judge sql execute result */
    if  (dbresults(dbconn) != SUCCEED)
    {
           fprintf(stderr, "Could not execute sql: %s\n", sql);
            return 13;
    }
    /* Close the connection */
    dbclose(dbconn);
    printf("success\n");
    return 0;
}
```

3. Press **Esc** to exit the insert mode and enter **:wq** to save and close the file.
4. Run the following command to run the C application:

```
gcc main01.c -lsybdb  -o main01
./main01
```

Command output:

```
[root@                    ~]# gcc main01.c -lsybdb  -o main01
[root@                    ~]# ./main01
setting login hostname:
Server message (severity 0): Changed database context to 'master'.
Server message (severity 10): Changed language setting to 'us_english'
Server message (severity 0): Changed database context to 'sqlserverdb'.
id: 12, name: A
id: 13, name: B
id: 14, name: C
id: 15, name: D
success
```

15.5. View the version of Babelfish

This topic describes how to view the version of Babelfish that is enabled for an ApsaraDB RDS for PostgreSQL instance. You can connect your application to the RDS instance over the Tabular Data Stream (TDS) port or PostgreSQL port to view the version.

Prerequisites

- An ApsaraDB RDS for PostgreSQL instance for which Babelfish is enabled is created. For more information, see Enable Babelfish for an ApsaraDB RDS for PostgreSQL instance.
- A Babelfish account is created. For more information, see Manage Babelfish accounts.
- A whitelist is configured to allow the server on which your client resides to access the RDS instance. For

more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

- The endpoints and TDS port of the RDS instance are obtained. For more information, see View the endpoints and TDS port.
- sqlcmd and psql are installed.

> **Note** In this topic, the sqlcmd command-line tool is used to connect to the RDS instance over the TDS port, and psql is used to connect to the RDS instance over the PostgreSQL port. For more information, see Use clients to connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled and Connect to an ApsaraDB RDS for PostgreSQL instance.

Connect to the RDS instance over the TDS port to view the Babelfish version

1. Run the following command to connect to the RDS instance over the TDS port:

```
sqlcmd -S pgm-****.pg.rds.aliyuncs.com,1433 -U babelfish_user
```

2. Execute the following statement to check whether the connection to the RDS instance is established over the TDS port:

```
SELECT CAST(serverproperty('babelfish') AS BIT) AS is_run_on_babelfish;
GO
```

Sample output:

```
is_run_on_babelfish
-------------------
                  1
(1 rows affected)
```

3. Execute the following statement to view the version of Babelfish:

```
SELECT CAST(serverproperty('babelfishversion') AS VARCHAR(5)) as babelfish_version;
GO
```

Sample output:

```
babelfish_version
-----------------
1.2.0
(1 rows affected)
```

4. Execute the following statement to view the version details of Babelfish:

```
SELECT @@version as version;
GO
```

Sample output:

```
version
------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------
Babelfish for PostgreSQL with SQL Server Compatibility - 12.0.2000.8
PostgreSQL 13.6
(1 rows affected)
```

Connect to the RDS instance over the PostgreSQL port to view the Babelfish version

1. Run the following command to connect to the RDS instance over the PostgreSQL port:

   ```
   psql -h pgm-****.pg.rds.aliyuncs.com -p 5432 -U babelfish_user -d babelfish_db
   ```

2. Execute the following statement to view the version of Babelfish:

   ```
   SELECT
       version() as postgresql_version,
       sys.version() as babelfish_compatibility,
       sys.SERVERPROPERTY('BabelfishVersion') as babelfish_Version;
   ```

   Sample output:

   ```
    postgresql_version |                      babelfish_compatibility
   | babelfish_version
   -------------------+-------------------------------------------------------------------------
   --+-------------------
    PostgreSQL 13.6    | Babelfish for PostgreSQL with SQL Server Compatibility - 12.0.2000.
   8+| 1.2.0
                      | PostgreSQL 13.6
   |
   (1 row)
   ```

15.6. Migrate the data of a SQL Server database to a database on an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled

This topic describes how to migrate table objects and data of a SQL Server database to a database on an ApsaraDB RDS for PostgreSQL instance for which Babelfish is enabled.

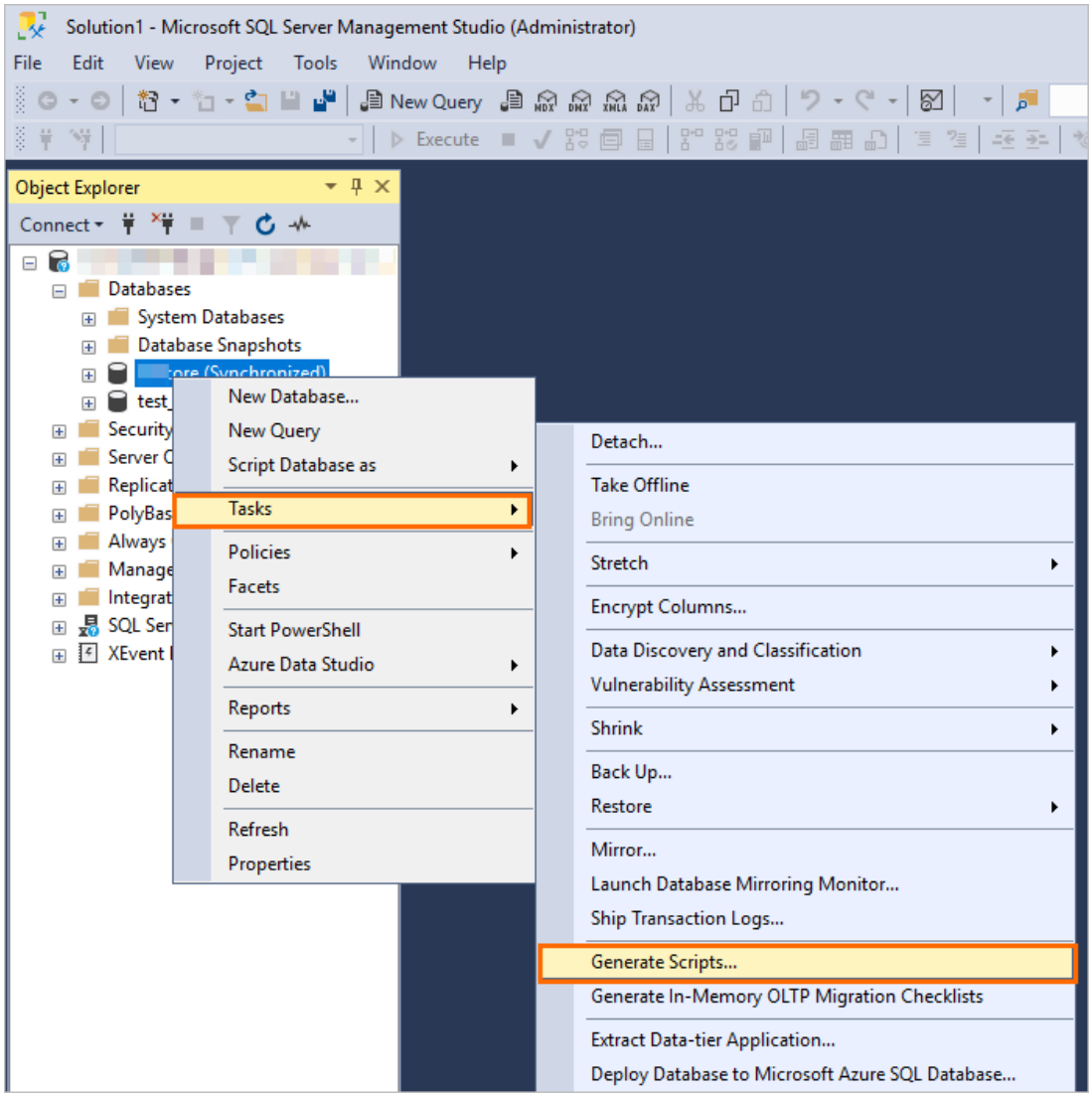
Prerequisites

- An ApsaraDB RDS for PostgreSQL instance for which Babelfish is enabled is created. For more information, see Enable Babelfish for an ApsaraDB RDS for PostgreSQL instance.
- A Babelfish account is created. For more information, see Manage Babelfish accounts.
- A whitelist is configured to allow the server on which your client resides to access the RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

- The endpoints and Tabular Data Stream (TDS) port of the RDS instance are obtained. For more information, see View the endpoints and TDS port.

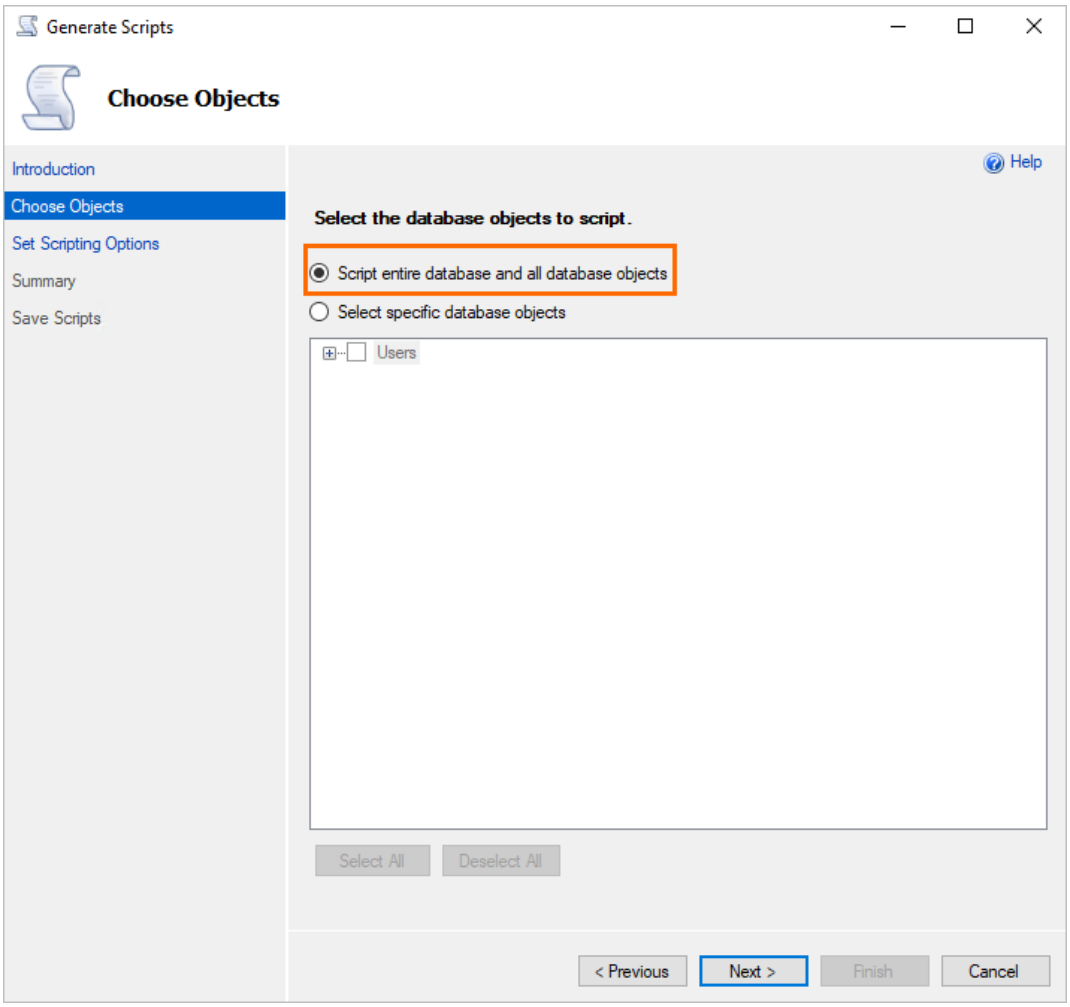
Procedure

1. Export the table objects and data of the SQL Server database.
   i. Connect to the SQL Server database by using SQL Server Management Studio (SSMS). For more information, see Quickstart: Connect and query a SQL Server instance using SQL Server Management Studio (SSMS).
   ii. In the left-side navigation pane of **Object Explorer**, right-click the SQL Server database that you want to migrate. Then, choose **Tasks > Generate Scripts**.

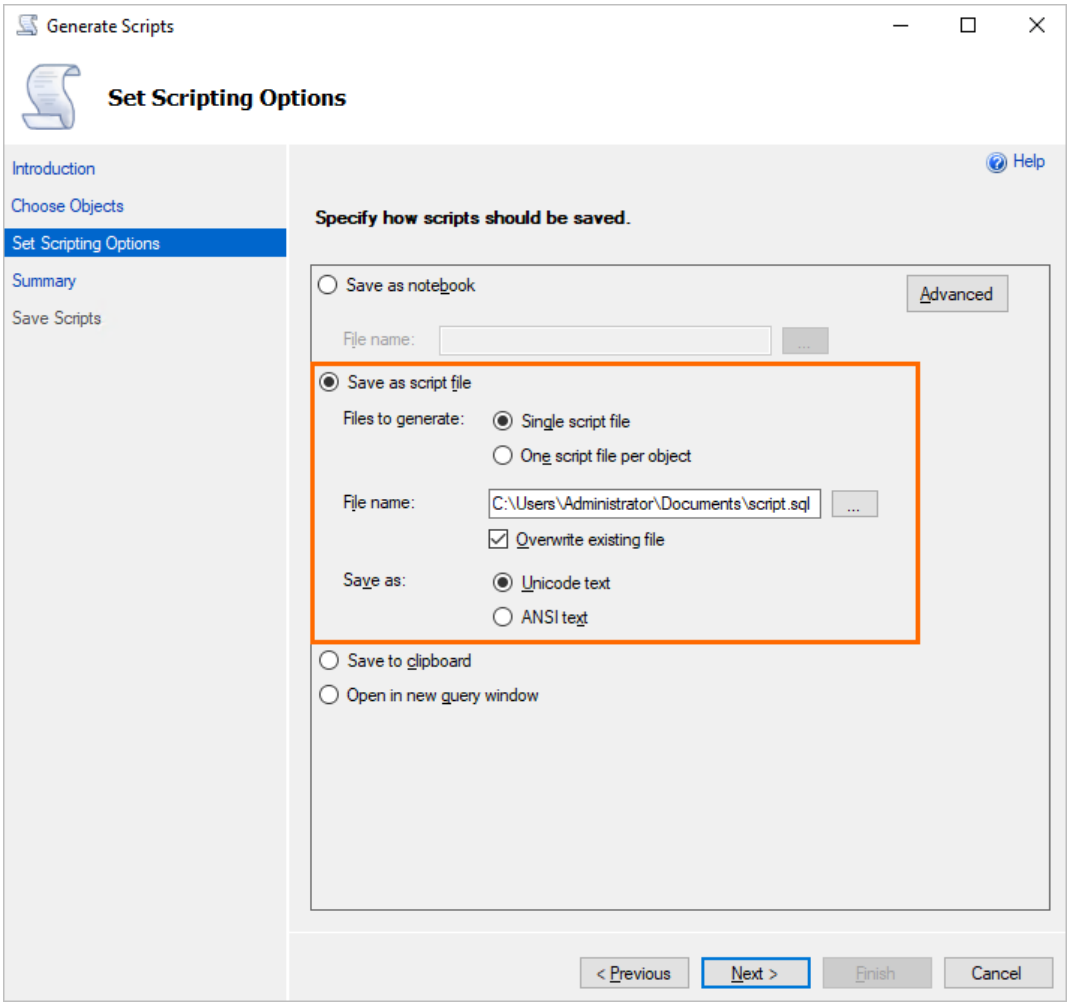


   iii. In the **Generate Scripts** configuration wizard, configure the parameters and export the script to a `.sql` file.
      a. Click **Next** in the **Introduction** step.

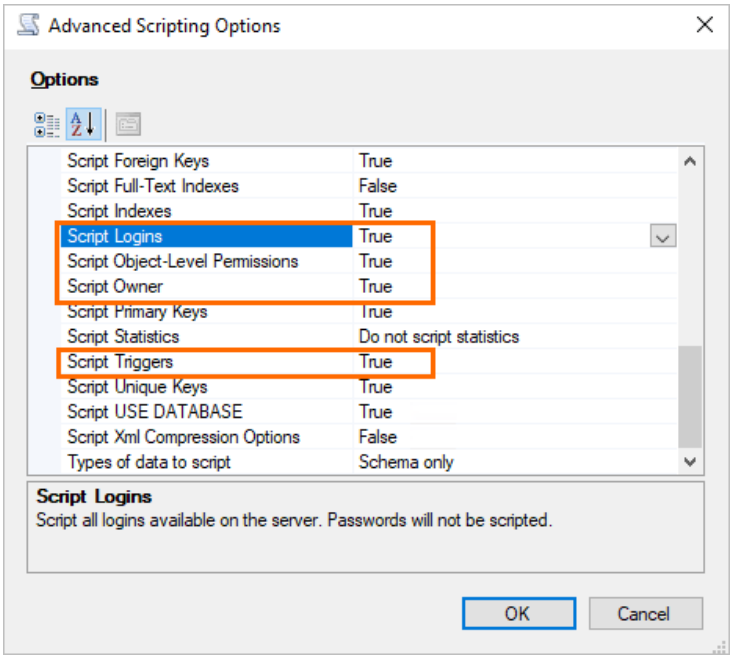
b. In the **Choose Objects** step, select **Script entire database and all database objects** and click **Next**.



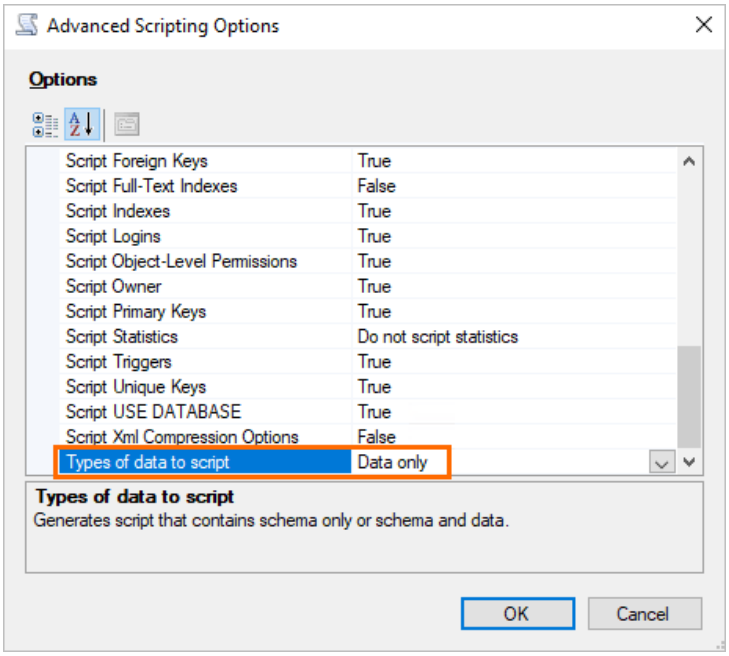
c. In the **Set Scripting Options** step, select **Save as script file** and configure the **File name** parameter. Then, click **Next**.



- If you want to export the data definition language (DDL) statements of the SQL Server database, perform the following operations: In the Set Scripting Options step, click **Advanced**. In the Advanced Scripting Options dialog box, configure the parameters based on the following figure:



- If you want to export the data manipulation language (DML) statements of the SQL Server database, perform the following operations: In the Set Scripting Options step, click **Advanced**. In the Advanced Scripting Options dialog box, set **Type of data to script** to **Data only**.



d. In the **Summary** step, click **Next**.

e. In the **Save Scripts** step, click **Confirm**.

After the configuration, you can obtain the `.sql` file from the path that you specified in the **Set Scripting Options** step.

2. Run Babelfish Compass to check how Babelfish supports T-SQL statements and make required modifications based on the results.

> **Note** For more information about Babelfish Compass, see Babelfish Compass official documentation.

3. Execute the modified T-SQL statements on the RDS instance for which Babelfish is enabled.

   i. Connect to the RDS instance over the TDS port. For more information, see Use clients to connect to an ApsaraDB RDS for PostgreSQL instance with Babelfish enabled.

   ii. Execute the modified T-SQL statements on the RDS instance.

15.7. Common operations and compatibility description

You can connect to an ApsaraDB RDS for PostgreSQL instance for which Babelfish is enabled over the Tabular Data Stream (TDS) port. This topic describes the common operations and compatibility of SQL statements of the RDS instance after the connection.

Common operations

Category		
System query	Query the version of a database.	`SELECT @@version;`
	Query the information of a database.	`SELECT * FROM sys.databases;`
Database-related operations	Create a database.	`CREATE DATABASE testdb;` **Note** If you use the **Single-DB** migration mode, you can create only one database. If you have created a database, you cannot create another database.
	Query a database.	`SELECT * FROM sys.databases WHERE name = 'testdb';`

	Switch to a different database.	`USE testdb`
		`GO`
		`SELECT db_name();`

	Delete a database.	`DROP DATABASE testdb;`
	Create a schema.	`CREATE SCHEMA sch_demo;`

Category		
Schema-related operations	View a schema.	`SELECT * FROM sys.schemas AS sch WHERE sch.name = 'sch_demo';`

	Use a schema to create a table.	`CREATE TABLE sch_demo.tb_demo(id int);`
		`SELECT`
		`  sch.name AS schema_name,`
		`  tb.name AS table_name`
		`FROM sys.tables AS tb`
		`  INNER JOIN  sys.schemas AS sch`
		`  ON tb.schema_id = sch.schema_id`
		`WHERE tb.name = 'tb_demo';`

	Delete a schema.	**Note** If a table is created by using a schema, you must delete the table before you delete the schema.
		`DROP TABLE sch_demo.tb_demo;`
		`GO`
		`DROP SCHEMA sch_demo;`
		`GO`

	Create a table.	`USE testdb`
		`GO`
		`CREATE TABLE dbo.tb_test(`
		`  id int not null IDENTITY(1,1) PRIMARY KEY,`
		`  name varchar(50))`
		`GO`

Category		

Table-related operations	Query a table.	`SELECT sche.name AS schema_name, tb.name AS table_name`
		`FROM sys.tables AS tb`
		`  INNER JOIN sys.schemas AS sche`
		`  ON tb.schema_id = sche.schema_id`
		`WHERE tb.name = 'tb_test';`
		`GO`

	Create a field.	`ALTER TABLE dbo.tb_test ADD col_added bigint null;` `GO`
	Modify a field in a table.	`ALTER TABLE dbo.tb_test ALTER column col_added varchar(50);` `GO`
	Delete a field from a table.	`ALTER TABLE dbo.tb_test DROP column col_added;` `GO`
	Create an index.	`CREATE INDEX ix_tb_test_name ON tb_test(name);` `GO`
	Delete an index.	`DROP INDEX ix_tb_test_name ON tb_test;` `GO`

Category		

Database-related operations	INSERT	`INSERT INTO dbo.tb_test`
		`SELECT 'A' UNION ALL`
		`SELECT 'B';`
		`GO`

	SELECT	`SELECT * FROM dbo.tb_test;`

	UPDATE	`UPDATE TOP(1) dbo.tb_test`
		`SET name = 'A_updated';`
		`GO`

	DELETE	`DELETE TOP(1) FROM dbo.tb_test;`
		`GO`
		`SELECT * FROM dbo.tb_test;`

Category		

Operations related to a stored procedure	Create a stored procedure.	`USE testdb`
		`GO`
		`CREATE PROC dbo.UP_getDemoData(`
		`  @id int`
		`)`
		`AS`
		`BEGIN`
		`  SET NOCOUNT ON`
		`  SELECT *`
		`  FROM dbo.tb_test`
		`  WHERE id = @id`
		`END;`
		`GO`

	View a stored procedure.	`SELECT *`
		`FROM sys.procedures`
		`WHERE name = 'up_getdemodata';`

	Execute a stored procedure.	`EXEC dbo.UP_getDemoData @id = 7;` `GO`

	Delete a stored procedure.	`USE testdb`
		`GO`
		`DROP PROC dbo.UP_getDemoData`
		`GO`

Compatibility

> **Note** This section provides only common incompatibility scenarios. For more information, see Babelfish for PostgreSQL official documentation.

The following SQL statements are not supported for an RDS instance for which Babelfish is enabled:

- View the schema of a table. Example:

```
EXEC sp_help 'dbo.tb_test'
```

- When you modify a field in a table, set the default value to null. Example:

```
ALTER TABLE dbo.tb_test ALTER column col_added varchar(50) null;
GO
```

- Recreate an index. We recommend that you delete an index and then create an index. Example:

```
ALTER INDEX ix_tb_test_name ON tb_test REBUILD;
GO
```

- Modify a stored procedure. We recommend that you delete a stored procedure and then create a stored procedure. Example:

```
USE testdb
GO
ALTER PROC dbo.UP_getDemoData(
    @id int
)
AS
BEGIN
    SET NOCOUNT ON
    SELECT *
    FROM dbo.tb_test
    WHERE id >= @id
END;
GO
```

- Query an execution plan. Example:

```
SET showplan_xml ON
SELECT * from tb_test;
```

16.RDS for PostgreSQL read-only instances

16.1. Overview of read-only ApsaraDB RDS for PostgreSQL instances

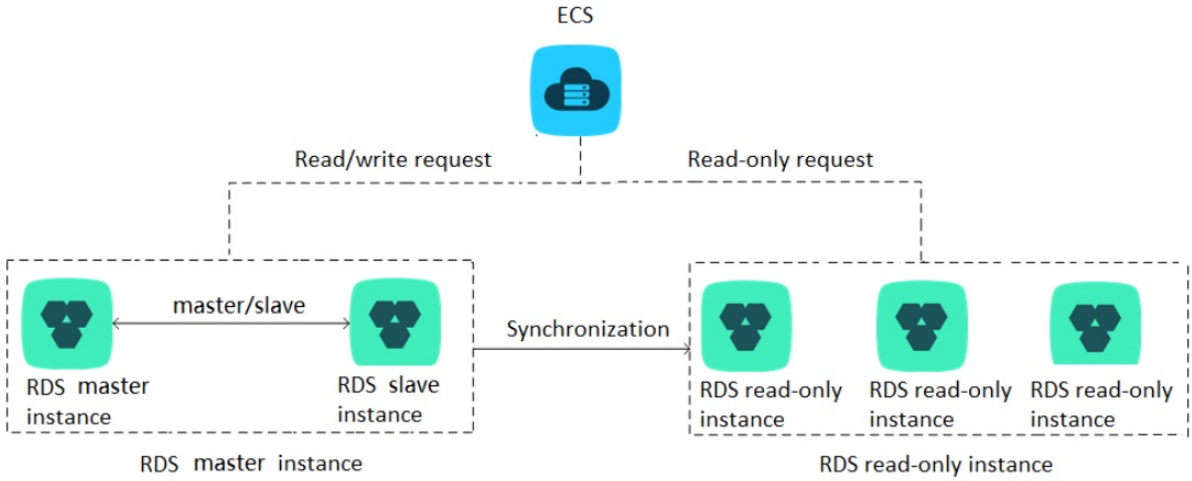
This topic provides an overview of read-only ApsaraDB RDS for PostgreSQL instances. If your database system receives a large number of read requests and a small number of write requests, a single primary RDS instance may fail to process the read requests at high speeds. This may even interrupt your workloads. In this case, you can create one or more read-only RDS instances to offload read requests from the primary RDS instance. This increases the read capability of your database system and the throughput of your application.

Background information

When a read-only RDS instance is being created, ApsaraDB RDS replicates data from the secondary RDS instance to the read-only RDS instance. After the read-only RDS instance is created, it has the same data as the primary RDS instance. In addition, after the data on the primary RDS instance is updated, ApsaraDB RDS immediately synchronizes the updates to all read-only RDS instances that are attached to the primary RDS instance.

> **Note**
>
> - If the primary RDS instance is equipped with local SSDs, you can create up to 5 read-only RDS instances. If the primary RDS instance is equipped with standard SSDs or enhanced SSDs (ESSDs), you can create up to 32 read-only RDS instances.
> - If the primary RDS instance is equipped with local SSDs, its read-only RDS instances run in a high-availability architecture.
> - If the primary RDS instance is equipped with standard SSDs or ESSDs, its read-only RDS instances run in a single-node architecture. In this architecture, a read-only RDS instance does not have a secondary RDS instance as a standby. For availability purposes, we recommend that you purchase more than one read-only RDS instance. This allows you to perform failovers among the created read-only RDS instances by using libpq or Java Database Connectivity (JDBC). For more information, see Configure automatic failover and read/write splitting.

The following figure shows the topology of the primary RDS instance and its read-only RDS instances.



Billing

Read-only RDS instances support the subscription billing method and the pay-as-you-go billing method. For more information about the fee for a subscription read-only RDS instance, go to the ApsaraDB RDS buy page. For more information about the fee for a pay-as-you-go read-only RDS instance, see Read-only ApsaraDB RDS instance types.

Highlights

- Region and zone: The primary RDS instance and its read-only RDS instances can reside in the same zone or in different zones of the same region.
- Specifications and storage capacity: If the primary RDS instance is equipped with local SSDs, the specifications and storage capacity of a read-only RDS instance must be higher than or equal to the specifications and storage of the primary RDS instance.
- Network type: The network type of a read-only RDS instance can be different from the network type of the primary RDS instance. For more information, see Change the network type of an ApsaraDB RDS for PostgreSQL instance.
- Account and database management: The accounts and databases on a read-only RDS instance are synchronized from the primary RDS instance. You cannot manage the accounts or databases on a read-only RDS instance.
- IP address whitelist management: When a read-only RDS instance is being created, ApsaraDB RDS replicates the IP address whitelists of the primary RDS instance to the read-only RDS instance. However, the IP address whitelists on the read-only RDS instance are independent of the IP address whitelists on the primary RDS instance. For more information about how to modify the IP address whitelists of a read-only RDS instance, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
- Monitoring and alerting: You can monitor the performance metrics of a read-only RDS instance. These metrics include the disk usage, IOPS, number of connections, and CPU utilization. The monitoring data of these metrics are provided in charts.

Precautions

- If the primary RDS instance is equipped with local SSDs, you can create up to 5 read-only RDS instances. If the primary RDS instance is equipped with standard SSDs or ESSDs, you can create up to 32 read-only RDS instances.
- Backups are created on the primary RDS instance. Read-only RDS instances do not support backup settings or manual backups.

- You cannot migrate data to read-only RDS instances.
- You cannot create or delete databases on read-only RDS instances.
- You cannot create or delete accounts, grant permissions to accounts, or change the passwords of accounts on read-only RDS instances.
- If the memory capacity that you specify for the primary RDS instance during a specification change is greater than the memory capacity of a read-only RDS instance, the read-only RDS instance restarts.
- If a read-only RDS instance encounters unexpected errors such as failures to replicate database engine settings, ApsaraDB RDS rebuilds the read-only RDS instance.

FAQ

After I create accounts on my primary RDS instance, can I manage the accounts on the read-only RDS instances?

No, the accounts that are created on your primary RDS instance are synchronized to its read-only RDS instances. You cannot manage the accounts on the read-only RDS instances. The accounts have only the read permissions on the read-only RDS instances.

16.2. Create a read-only ApsaraDB RDS for PostgreSQL instance

This topic describes how to create a read-only RDS instance for a primary ApsaraDB RDS for PostgreSQL instance. Read-only RDS instances are used to process a large number of read requests. You can increase the read capability of your database system and the throughput of your application by adding read-only RDS instances to your database system. Each read-only RDS instance is a replica of the primary RDS instance and has the same data as the primary RDS instance. Data updates on the primary RDS instance are automatically synchronized to all read-only RDS instances that are attached to the primary RDS instance.

For more information about read-only RDS instances, see Overview of read-only ApsaraDB RDS for PostgreSQL instances.

Prerequisites

- The primary RDS instance runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, or PostgreSQL 14.
- If the primary RDS instance uses local SSDs, it must be a dedicated RDS instance that provides at least 8 cores and 32 GB of memory.
- The primary RDS instance runs RDS High-availability Edition. Before you create a read-only RDS instance, you must view the RDS edition of the primary RDS instance on the **Basic Information** page of the primary RDS instance in the ApsaraDB RDS console. If the primary RDS instance runs RDS Basic Edition, you can click **Change Specifications** on the page to upgrade the RDS edition of the primary RDS instance to High-availability Edition before you create a read-only RDS instance. For more information, see 变更配置.

Precautions

- You can create only read-only RDS instances that belong to the same instance family as the primary RDS instance. For example, if the primary RDS instance belongs to a general-purpose instance family, you can create only read-only RDS instances of the general-purpose instance family. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types and Read-only ApsaraDB RDS for PostgreSQL instance types.

- You can create read-only RDS instances for the primary RDS instance. You cannot convert existing RDS instances to read-only RDS instances.
- When you create a read-only RDS instance, ApsaraDB RDS replicates data from the secondary RDS instance to the read-only RDS instance. This prevents interruptions to your workloads on the primary RDS instance.
- A read-only RDS instance does not inherit the parameter settings of the primary RDS instance. Default parameter settings are generated for the read-only RDS instance. You can modify the default parameter settings in the ApsaraDB RDS console.

  Notice Read-only RDS instances that are attached to the primary RDS instance of a new general-purpose instance type inherit the parameter settings of the primary RDS instance. For more information about read-only RDS instances that use new general-purpose instance types, see Read-only ApsaraDB RDS for PostgreSQL instances with the new general-purpose instance types.

- If the primary RDS instance uses local SSDs, the specifications and storage capacity of its read-only RDS instances cannot be lower than the specifications and storage capacity of the primary RDS instance.
- If the primary RDS instance uses standard SSDs or enhanced SSDs (ESSDs), we recommend that you make sure the specifications of its read-only RDS instances are the same as the specifications of the primary RDS instance or are higher than or equal to 50% of the specifications of the primary RDS instance. This helps prevent replication latency and out of memory (OOM) errors that are caused by a significant performance difference between the primary RDS instance and its read-only RDS instances.
- If the primary RDS instance uses local SSDs, you can create up to 5 read-only RDS instances. If the primary RDS instance uses standard SSDs or ESSDs, you can create up to 32 read-only RDS instances.
- If the primary RDS instance uses local SSDs, its read-only RDS instances run based on a high-availability architecture. If the primary RDS instance uses standard SSDs or ESSDs, its read-only RDS instances run based on a single-node architecture.

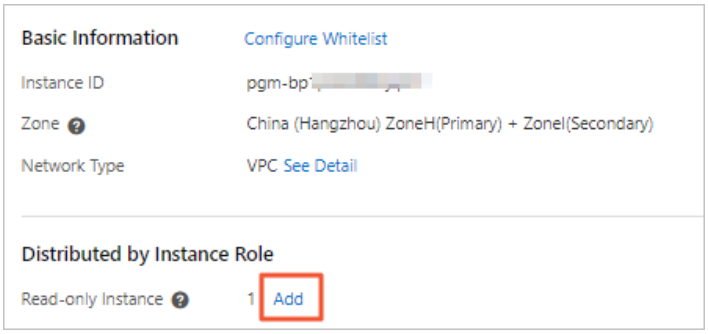
  Note In the single-node architecture, a read-only RDS instance does not have a secondary RDS instance as a standby. For availability purposes, we recommend that you purchase multiple read-only RDS instances. You can use the libpq or Java Database Connectivity (JDBC) API to implement failovers among the read-only RDS instances that you create. For more information, see Configure automatic failover and read/write splitting.

- You are charged for the read-only RDS instances that you create based on the subscription billing method or the pay-as-you-go billing method. For more information about the fee for a subscription read-only RDS instance, visit the ApsaraDB RDS buy page. For more information about the fee for a pay-as-you-go read-only RDS instance, see Read-only ApsaraDB RDS instance types.

Create a read-only RDS instance

1.
2. In the **Distributed by Instance Role** section of the page that appears, click **Add** to the right of **Read-only Instance**.

   Note If you are using the original ApsaraDB RDS console, click **Create Read-only Instance** in the Distributed by Instance Role section of the Basic Information page.



3. Configure the following parameters.

Parameter	Description
Billing Method	◦ **Subscription**: A subscription instance is an instance for which you pay an upfront fee. If you want to use the read-only RDS instance for a long period of time, we recommend that you select the subscription billing method. If you select the subscription billing method, you must specify a subscription period. ◦ **Pay-As-You-Go**: A pay-as-you-go instance is an instance for which you are charged per hour based on your resource usage. If you want to use the read-only RDS instance for a short period of time, we recommend that you select the pay-as-you-go billing method. You can create a pay-as-you-go read-only RDS instance. After you confirm that the created read-only RDS instance meets your business requirements, you can change the billing method of the read-only RDS instance from pay-as-you-go to subscription.
Zone	The zone to which the read-only RDS instance belongs. Each zone is an independent physical location within a region. Zones in the same region do not have substantial differences. The multi-zone deployment method supports zone-disaster recovery.
Instance Type	◦ **General-purpose (Entry-level)**: allows you to select a general-purpose instance type. A general-purpose instance exclusively occupies the allocated memory and I/O resources. However, it shares CPU and storage resources with other general-purpose instances that are deployed on the same host. ◦ **Dedicated (Enterprise-level)**: allows you to select a dedicated instance type or a dedicated host instance type. A dedicated RDS instance exclusively occupies the allocated CPU, memory, storage, and I/O resources. Dedicated host instance types provide the highest specifications in the dedicated instance family. A dedicated host instance exclusively occupies all the CPU, memory, storage, and I/O resources on the host on which the instance is deployed. **Note** Each instance type supports a specific number of cores, memory capacity, maximum number of connections, and maximum IOPS. For more information, see Read-only ApsaraDB RDS for PostgreSQL instance types.

Parameter	Description
Capacity	The maximum amount of storage that is provisioned to store data files, system files, WAL files, and transaction files in the read-only RDS instance. You can adjust the storage capacity at a step size of 5 GB. **Note** A dedicated RDS instance that uses local SSDs exclusively occupies the allocated resources, and its storage capacity varies based on the instance type. For more information, see Read-only ApsaraDB RDS for PostgreSQL instance types.

4. Click **Next: Instance Configuration** and configure the following parameters.

Parameter	Description
Network Type	○ **Classic Network**: the traditional type of network. ○ **VPC**: the recommended type of network. A virtual private cloud (VPC) is an isolated virtual network that provides higher security and higher performance than the classic network. If you select the VPC network type, you must configure the **VPC** and **VSwitch of Primary Node** parameters. If you set the **Deployment Method** parameter to **Multi-zone deployment** in the previous step, you must also configure the **VSwitch of Secondary Node** parameter. **Note** The network type of the RDS instance must be the same as the network type of the Elastic Compute Service (ECS) instance that you want to connect. If both the RDS instance and the ECS instance reside in VPCs, these instances must reside in the same VPC. If the RDS instance and the ECS instance reside in different VPCs, these instances cannot communicate over an internal network.

5. Click **Next: Confirm Order**.
6. Read and select Terms of Service, click **Pay Now**, and then complete the payment.

The amount of time that is required to create a read-only RDS instance varies based on the storage type and storage capacity of the primary RDS instance. When a read-only RDS instance is being created, your workloads on the primary RDS instance are not affected.

- If the primary RDS instance uses local SSDs, the amount of time that is required to create a read-only RDS instance is approximately 10 minutes plus the amount of time that is required for a full backup.
- If the primary RDS instance uses standard SSDs, the amount of time that is required to create a read-only RDS instance is approximately 20 minutes plus the amount of time that is required for a full backup.
- If the primary RDS instance uses ESSDs, the amount of time required to create a read-only RDS instance is approximately 20 minutes.

View a read-only RDS instance

- To view a read-only RDS instance on the Instances page, perform the following steps:

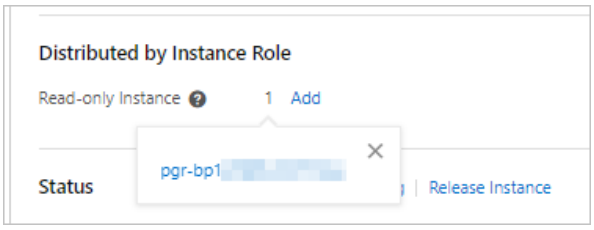
  i.

ii. Find the read-only RDS instance and click the instance ID.

- To view a read-only RDS instance on the Basic Information page of the primary RDS instance, perform the following steps:

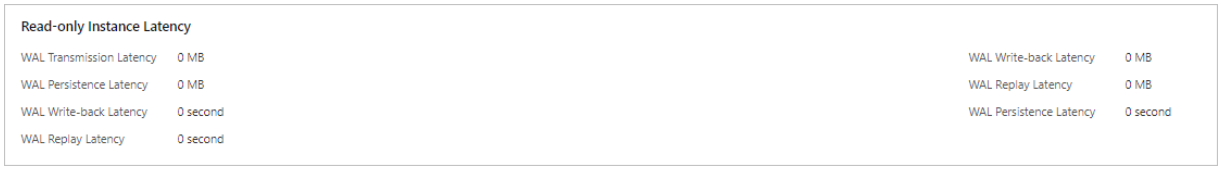
  i.

  ii. On the **Basic Information** page of the primary RDS instance, move the pointer over the number of read-only RDS instances and click the ID of the read-only RDS instance that you want to view.



View the data replication latency for a read-only RDS instance

A read-only RDS instance may synchronize data from the primary RDS instance at a specific latency. You can view the latency on the Basic Information page of the read-only RDS instance.



Related operations

Operation	Description
Create a read-only instance	Creates a read-only ApsaraDB RDS instance.

17.Database proxy (read/write splitting)

17.1. What are database proxies?

The database proxy feature of ApsaraDB RDS for PostgreSQL provides advanced capabilities such as automatic read/write splitting. A database proxy resides between your database system and your application and receives all requests from your application. A database proxy is easy to use and maintain, and provides high availability and high performance. This topic describes the database proxy feature of ApsaraDB RDS for PostgreSQL.

Scenarios

- The primary RDS instance is heavily loaded due to a large number of requests that are encapsulated in transactions.
- The primary RDS instance is heavily loaded due to an excessively large number of connections.
- You need to process read-only workloads and workloads that need to be isolated from multiple applications.

> **Note** For example, your database system consists of one primary RDS instance and four read-only RDS instances, and you have two applications, Application A and Application B. Application A initiates only read requests, and Application B initiates both read and write requests. In this case, you can bind two read-only RDS instances to Proxy Terminal A that has the Read-only attribute and bind the other two read-only RDS instances to Proxy Terminal B that has the Read/Write attribute. This way, Application A and Application B are physically isolated from each other in your database system.

Terms

- proxy terminal

  Proxy terminals are the core of database proxies. You can create custom endpoints for proxy terminals. Each ApsaraDB RDS for PostgreSQL instance supports up to seven proxy terminals. You can modify the read and write attributes of each proxy terminal based on your business requirements.

- read/write splitting

  Read/write splitting indicates that proxy terminals are used to automatically forward read and write requests.

  If your database system receives a large number of read requests and a small number of write requests, the primary RDS instance may fail to efficiently process read requests and your workloads may be interrupted. The read/write splitting feature allows ApsaraDB RDS for PostgreSQL to forward write requests to the primary RDS instance and read requests to the read-only RDS instances. This reduces the loads on the primary RDS instance.

- transaction splitting

  The transaction splitting feature is automatically enabled for a database proxy. This feature allows ApsaraDB RDS for PostgreSQL to forward the read requests prior to write operations in a transaction to the read-only RDS instances of your database system. This reduces the loads on the primary RDS instance.

> **Note**
> - Explicit transactions cannot be split. These explicit transactions include the transactions that are started by executing the BEGIN or START statement.
> - If the transaction splitting feature is enabled, global consistency cannot be ensured. Before you use this feature, we recommend that you evaluate whether this feature is suitable for your workloads.
> - The transaction splitting feature cannot be disabled. If you want to disable this feature, submit a . If the transaction splitting feature is disabled, transaction requests are forwarded to the primary RDS instance.

Benefits of read/write splitting

- Unified endpoint to facilitate maintenance

  If you do not enable the read/write splitting feature, you can perform read/write splitting only after you add the endpoints of the primary RDS instance and read-only RDS instances to your application.

  If you enable the read/write splitting feature, you can use a database proxy endpoint to implement read/write splitting. You need to only add the database proxy endpoint to your application. After your application is connected to the database proxy endpoint, your database system can forward read and write requests to the primary RDS instance and read-only RDS instances based on the read weights of the instances. This reduces maintenance costs.

  You can also create read-only RDS instances to increase the read capability of your database system without the need to modify the configuration data on your application.

- Native links to improve performance and reduce maintenance costs

  If you build your own proxy layer in the cloud to implement read/write splitting, data must be parsed and forwarded by multiple components before the data reaches your database system. As a result, response latencies increase. The read/write splitting feature is embedded in the ApsaraDB RDS ecosystem to reduce response latencies, increase processing speeds, and reduce maintenance costs.

- Instance-level health checks to ensure high availability

  If the read/write splitting feature is enabled, ApsaraDB RDS for PostgreSQL automatically checks the health status of the primary RDS instance and read-only RDS instances. If a read-only RDS instance unexpectedly exits or the data replication latency of the read-only RDS instance exceeds the specified threshold, ApsaraDB RDS for PostgreSQL stops forwarding read requests to the instance. ApsaraDB RDS for PostgreSQL forwards the read requests that are destined for the faulty RDS instance to other healthy RDS instances in your database system. This ensures service availability even if a read-only RDS instance fails. After the faulty read-only RDS instance recovers, ApsaraDB RDS for PostgreSQL resumes forwarding read requests to the instance.

  > **Note** To mitigate the impacts of single points of failure (SPOFs), we recommend that you create at least two read-only RDS instances.

- Configurable read weights and thresholds to ensure suitability in various scenarios

  You can specify the read weights of the primary RDS instance and read-only RDS instances. You can also specify the latency threshold for data replication to the read-only RDS instances.

Request forwarding types

Forwarding destination	Request type
Primary RDS instance	• Requests that are used to execute INSERT, UPDATE, DELETE, and SELECT FOR UPDATE statements • All requests that are used to execute DDL statements, such as the DDL statements that are used to create databases or tables, delete databases or tables, and change schemas or permissions. • All requests that are encapsulated in transactions • Requests that are used to call user-defined functions • Requests that are used to run stored procedures. • Requests for multi-statements Note If you execute multi-statements or call stored procedures, all subsequent requests over the current connection are forwarded to the primary RDS instance. To perform read/write splitting again, you must close the current connection and establish a new connection. • Requests that involve temporary tables • Read and write requests for system tables • Write operations that are stored in PreparedStatement objects
Primary RDS instance or read-only RDS instances	• Requests that are used to execute SELECT statements that are not encapsulated in transactions • Requests prior to the first write operation in the transaction after the transaction splitting feature is enabled • Read operations that are stored in PreparedStatement objects • System functions, such as pg_sleep, that can be safely called on read-only RDS instances
Primary RDS instance and read-only RDS instances	• All requests that are used to reconfigure system variables • Parsing operations that are stored in PreparedStatement objects • BEGIN, START, END, ROLLBACK, and COMMIT • CANCEL

Usage notes

- Database proxy instances, read-only RDS instances, and primary RDS instances are separately billed. For more information about the billing rules, see Billing rules for the database proxy of an ApsaraDB RDS for PostgreSQL instance, Pricing, and Billable items, billing methods, and pricing.
- When you change the specifications of the primary RDS instance or a read-only RDS instance, a transient connection may occur.
- If you create or restart a read-only RDS instance after you enable the database proxy feature, only the requests that are sent over new connections are forwarded to the new or restarted read-only RDS instance.
- If a database proxy endpoint is used to perform read/write splitting, the read consistency of the requests in a session cannot be ensured. If you want to ensure the read consistency of these requests, you must update your AliPG version to 20220228 or later and then submit a . For more information

about how to update your AliPG version, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.

- If you use a database proxy endpoint, you must add `/*force_master*/ /*force_slave*/` when you view session variables to check the configurations on the primary RDS instance and read-only RDS instances.
- The database proxy feature uses the 1:N connection model. After your application initiates a connection request, the database proxy replicates the established connection to the primary RDS instance and all the read-only RDS instances. The maximum number of connections that are allowed for the database proxy is not limited. The maximum number of connections varies based on the specifications of the primary RDS instance and read-only RDS instances. After the database proxy receives a request from your application, the database proxy establishes a connection to each of the primary RDS instance and read-only RDS instances. After you enable the database proxy feature, we recommend that you specify the same maximum number of connections for the primary RDS instance and read-only RDS instances. If the maximum numbers for the primary RDS instance and read-only RDS instances are different, the maximum number of connections that are allowed for the database proxy is subject to the minimum number of connections among these instances.
- If the primary RDS instance is locked, the enabled proxy instances are not released but can process only read requests.
- If the primary RDS instance is released, the enabled proxy instances are automatically released. You are no longer charged for the database proxy feature.

Enable the database proxy feature

For more information, see Enable and configure the database proxy feature for an ApsaraDB RDS for PostgreSQL instance

17.2. Billing rules for the database proxy of an ApsaraDB RDS for PostgreSQL instance

This topic describes the billing rules for the database proxy of an ApsaraDB RDS for PostgreSQL instance.

Precautions

- Database proxy instances, read-only RDS instances, and primary RDS instances are separately billed.
- If the primary RDS instance is locked, the enabled proxy instances are not released but can process only read requests.
- If the primary RDS instance is released, the enabled proxy instances are automatically released. You are no longer charged for the database proxy feature.
- The database proxy feature of ApsaraDB RDS for PostgreSQL is rolled out in phases. The feature is available in the following regions and zones:
  - China (Hong Kong): Hongkong Zone B
  - China (Hangzhou): Hangzhou Zone G, Hangzhou Zone H, Hangzhou Zone I, and Hangzhou Zone J
  - China (Beijing): Beijing Zone F, Beijing Zone G, Beijing Zone H, and Beijing Zone I
  - China (Zhangjiakou): Zhangjiakou Zone A, Zhangjiakou Zone B, and Zhangjiakou Zone C
  - China (Shanghai): Shanghai Zone B, Shanghai Zone E, Shanghai Zone F, and Shanghai Zone G
  - China (Shenzhen): Shenzhen Zone D and Shenzhen Zone E

- Indonesia (Jakarta): Jakarta Zone A
- Singapore (Singapore): Singapore Zone A, Singapore Zone B, and Singapore Zone C

> **Note** If the preceding zones cannot meet your requirements, submit a .

Billing

Database proxy instances support only the pay-as-you-go billing method. The following table describes the prices of proxy instances in different Alibaba Cloud regions.

Region	Price (USD per proxy-hour)
UAE (Dubai)	0.377
Australia (Sydney)	0.273
Germany (Frankfurt)	0.243
China (Zhangjiakou) and China (Ulanqab)	0.120
Malaysia (Kuala Lumpur)	0.253
US (Virginia)	0.237
US (Silicon Valley)	0.284
Japan (Tokyo)	0.288
Singapore (Singapore) and Indonesia (Jakarta)	0.271
India (Mumbai)	0.231
UK (London)	0.280
China (Hong Kong)	0.297
Other regions	0.173

17.3. Use the database proxy feature

17.3.1. Enable and configure the database proxy feature for an ApsaraDB RDS for PostgreSQL instance

The database proxy feature of ApsaraDB RDS for PostgreSQL provides advanced capabilities such as read/write splitting. This topic describes how to enable and configure the database proxy feature for an ApsaraDB RDS for PostgreSQL instance.

Prerequisites

Your RDS instance meets the following conditions:

- The major engine version of the instance is PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, or PostgreSQL 14.
- The instance uses standard SSDs or enhanced SSDs (ESSDs).
- The instance runs RDS High-availability Edition.
- The instance is a primary RDS instance.
- The database proxy feature of ApsaraDB RDS for PostgreSQL is rolled out in phases. The feature is available in the following regions and zones:
  - China (Hong Kong): Hongkong Zone B
  - China (Hangzhou): Hangzhou Zone G, Hangzhou Zone H, Hangzhou Zone I, and Hangzhou Zone J
  - China (Beijing): Beijing Zone F, Beijing Zone G, Beijing Zone H, and Beijing Zone I
  - China (Zhangjiakou): Zhangjiakou Zone A, Zhangjiakou Zone B, and Zhangjiakou Zone C
  - China (Shanghai): Shanghai Zone B, Shanghai Zone E, Shanghai Zone F, and Shanghai Zone G
  - China (Shenzhen): Shenzhen Zone D and Shenzhen Zone E
  - Indonesia (Jakarta): Jakarta Zone A
  - Singapore (Singapore): Singapore Zone A, Singapore Zone B, and Singapore Zone C

> **Note** If the preceding zones cannot meet your requirements, submit a .

Billing

For more information, see Billing rules for the database proxy of an ApsaraDB RDS for PostgreSQL instance.

Precautions

- After the database proxy feature is enabled, we recommend that you do not migrate the primary RDS instance across zones. If you migrate the primary RDS instance across zones, the primary RDS instance and its proxy instances are in different zones. This increases access latency and slows down responses.
- A read-only RDS instance is created for your RDS instance. If no read-only RDS instances exist, you can enable the database proxy feature but cannot configure proxy terminals for your RDS instance. For more information, see Create a read-only ApsaraDB RDS for PostgreSQL instance.

Procedure

Step 1: Enable the database proxy feature

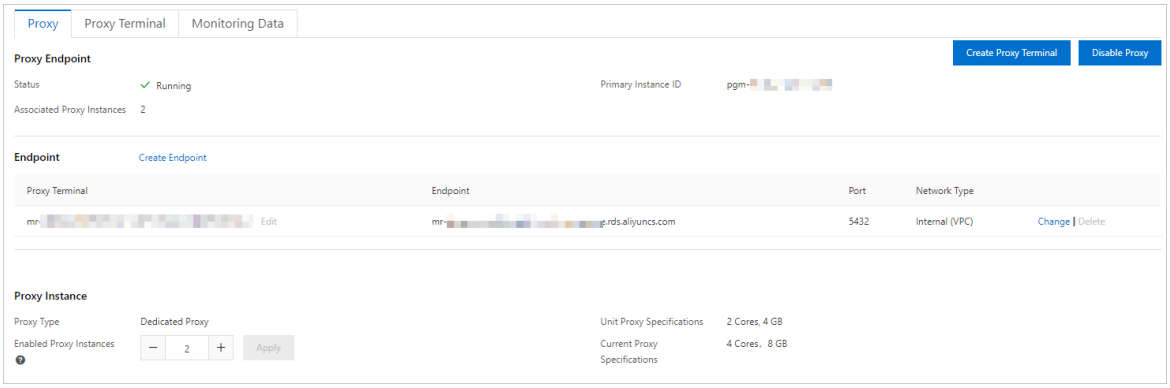
1.
2. In the left-side navigation pane, click **Database Proxy**.
3. Optional. The first time you enable the database proxy feature, click **Authorize** to the right of **SLR Authorization**. In the dialog box that appears, click **OK**. Alibaba Cloud automatically creates a service-linked role named **AliyunServiceRoleForRdsProxyOnEcs**. Then, the database proxy feature can use this role to bind elastic network interfaces (ENIs) to users and establish network connections.

   > **Note** For more information about the service-linked role AliyunServiceRoleForRdsProxyOnEcs, see Service-linked roles.

4. On the page that appears, click **Enable Proxy**. In the dialog box that appears, configure the **Proxies** parameter to specify the number of proxy instances and click **Enable**.

> **Note**
>
> - We recommend that you set the Proxies parameter to `one-eighth of the total number of cores that are configured for your primary RDS instance and its read-only RDS instances`. If the result is not an integer, you must round up the result to the nearest integer. Up to 60 proxy instances are supported.
>
>   For example, if your primary RDS instance has eight cores and its read-only RDS instance has four cores, we recommend that you specify the Proxies parameter based on the following calculation: `(8 + 4)/8 = 1.5`. The result 1.5 is rounded up to 2.
> - The number of proxy instances is used to calculate the performance of the database proxy. The number does not represent the actual number of proxy instances. If you set the Proxies parameter to a large value, the database proxy can process more requests of your RDS instance.
>
>   For example, if a single proxy instance is configured with `2 cores and 4 GB of memory` and you enable two proxy instances, the database proxy of your RDS instance can provide `4 cores and 8 GB of memory`.

After you enable the database proxy feature, you can view the basic information about the database proxy on the **Proxy** tab.



Basic information

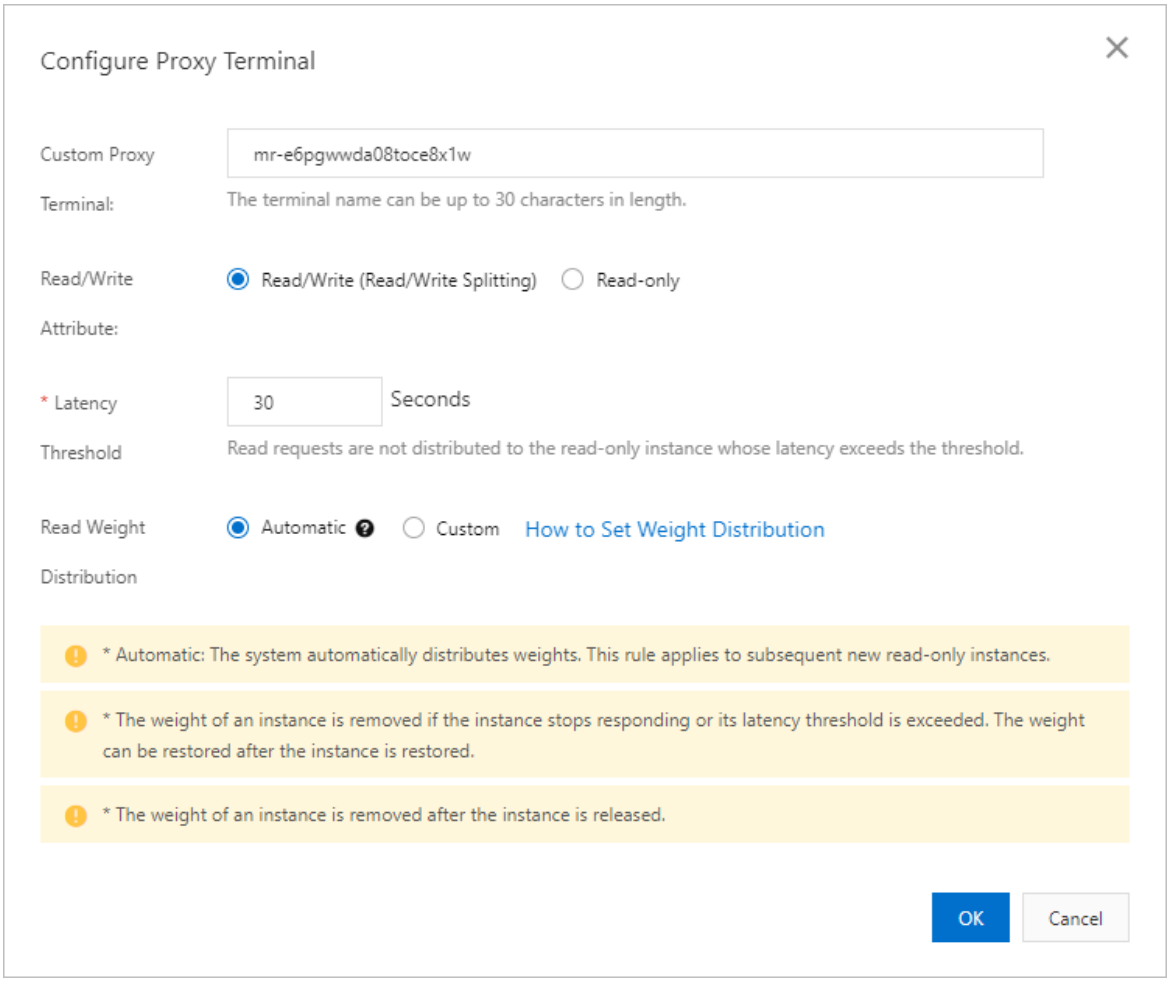
Section	Parameter	Description
Proxy Endpoint	**Status**	The status of the database proxy.
	Primary Instance ID	The ID of the RDS instance.
	Associated Proxy Instances	The number of proxy instances that are associated with the database proxy. You can increase the processing capability of the database proxy by enabling more proxy instances.
	Proxy Terminal	The name of the proxy terminal. You can create multiple proxy endpoints for each proxy terminal. For more information, see Create a database proxy endpoint.

Section	Parameter	Description
Endpoint	**Endpoint**	The endpoint that is used to connect to the database proxy. The database proxy provides a default proxy endpoint to which the proxy terminal feature is bound. You can create, modify, or delete a proxy endpoint. For more information, see Manage the database proxy endpoints of an ApsaraDB RDS for PostgreSQL instance.
	Port	The port number that is bound to the proxy endpoint. **Note** To change a port number, find the proxy endpoint to which the port number is bound and click **Change** on the right. A valid port number ranges from 1000 to 5999.
	Network Type	The network type of the proxy endpoint. You cannot change the network type of a proxy endpoint.
Proxy Instance	**Proxy Type**	Only **Dedicated Proxy** is supported.
	Unit Proxy Specifications	The specifications of each proxy instance. Only the specifications of `2 cores and 4 GB of memory` are supported.
	Enabled Proxy Instances	The number of enabled proxy instances.
	Current Proxy Specifications	The specifications of the current database proxy. The value is calculated based on the values of the **Unit Proxy Specifications** and **Enabled Proxy Instances** parameters. For example, if the value of the Enabled Proxy Instances parameter is 2, the specifications of the current database proxy are `4 cores and 8 GB of memory`.

Step 2: Configure a proxy terminal

Before you can use the advanced capabilities that are provided by the database proxy feature, you must configure a proxy terminal for your RDS instance.

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. On the page that appears, click the **Proxy Terminal** tab. Then, click **Configure Proxy Terminal**.
4. In the dialog box that appears, configure the following parameters and click **OK**.



Parameter	Description
Custom Proxy Terminal	The name of the proxy terminal. The name can be up to 30 characters in length.

Parameter	Description
Read/Write Attribute	The read and write attributes of the proxy terminal. For more information about the read and write attributes, see Processing logic based on the read and write attributes. ○ **Read/Write (Read/Write Splitting)**: The proxy terminal connects to the primary RDS instance and the read-only RDS instances, and can receive write requests. This is the default attribute. If you select this option, your primary RDS instance can receive write requests, and you can specify a read weight for the primary RDS instance. If you select this option, make sure that the proxy terminal is associated with at least one primary RDS instance and one read-only RDS instance. All write requests are routed to the primary RDS instance. ○ **Read-only**: The proxy terminal connects only to the read-only RDS instances and cannot receive write requests. You cannot specify a read weight for your primary RDS instance. If you select this option, make sure that the proxy terminal is associated with at least one read-only RDS instance. The proxy terminal does not route requests to the primary RDS instance. If you select Read-only for a proxy terminal, the proxy terminal assigns connections to the associated read-only RDS instances based on the round-robin algorithm. Each database client is assigned only one connection to one read-only RDS instance. The proxy terminal does not assign the connection to the primary RDS instance. The total number of available connections is the sum of connections that are established to all the read-only RDS instances.
Latency Threshold	The maximum latency that is allowed for the read-only RDS instances to replicate data from the primary RDS instance. If the latency at which a read-only RDS instance replicates data from the primary RDS instance exceeds the value of this parameter, ApsaraDB RDS no longer routes read requests to the read-only RDS instance regardless of the read weight of the read-only RDS instance. Valid values: 0 to 3600. Unit: seconds. The read-only RDS instances may replicate data from the primary RDS instance at a specific latency. The latency varies based on the status of the SQL statements that are executed. We recommend that you set this parameter to a value that is greater than or equal to 30. **Note** This parameter appears only when you set the Read/Write Attribute parameter to **Read/Write (Read/Write Splitting)**.

Parameter	Description
Read Weight Distribution	The method that is used to assign read weights. A higher read weight indicates more read requests that must be processed. For example, three read-only RDS instances are attached to the primary RDS instance, the read weight of the primary RDS instance is 0, and the read weights of the three read-only RDS instances are 100, 200, and 200. In this case, the primary RDS instance processes only write requests, and the three read-only RDS instances process all read requests based on the 1:2:2 ratio. ◦ **Automatic**: ApsaraDB RDS automatically assigns a read weight to each RDS instance in your database system based on the specifications of each RDS instance. After you create a read-only RDS instance, ApsaraDB RDS automatically assigns a read weight to the read-only RDS instance and adds the read-only RDS instance to the read/write splitting link. For more information, see Default read weights. ◦ **Custom**: You must manually specify a read weight for each RDS instance in your database system based on your business requirements. Valid values: 0 to 10000. By default, the read weight of a read-only RDS is 0. After you create a read-only RDS instance, you must manually specify a read weight for the read-only RDS instance based on your business requirements. **Note** ◦ You cannot specify weights for read-only RDS instances that have the data replication latency specified. ◦ After you reconfigure this parameter, the new read weights immediately take effect and no transient connections occur. In addition, the existing connections remain open. Only the requests that are sent over new connections are routed based on the new weights. ◦ After the RDS instances are released, the read weights automatically become invalid. ◦ If your RDS instance fails or the data replication latency exceeds the specified threshold, the read weights automatically become invalid. After your RDS instance runs as normal, the read weights become valid again.

After you configure a proxy terminal, you must add the specified endpoint of the proxy terminal to your application. Then, ApsaraDB RDS can route write requests to the primary RDS instance and read requests to the read-only RDS instances based on the read weights of these instances.

Processing logic based on the read and write attributes

Read and write attributes	Method to specify read weights	Weight of a primary RDS instance	Normal case	After the last read-only RDS instance is deleted	After all read-only RDS instances are faulty
Read-only	**Automatic** or **Custom**	You cannot specify a read weight for your primary RDS instance.	• Primary RDS instance: does not process read or write requests. • Proxy terminal: processes only read requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.
Read and write	Automatic	A weight greater than 0 For more information, see Default read weights.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.
	Custom	A weight greater than 0	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.
		A weight equal to 0	• Primary RDS instance: processes only write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.

Related operations

Operation	Description

Operation	Description
ModifyDBProxy	Enables or disables the database proxy feature.
DescribeDBProxy	Queries the details of a database proxy.
ModifyDBProxyEndpoint	Configures a database proxy terminal.

17.3.2. Create a database proxy terminal for an ApsaraDB RDS for PostgreSQL instance

The database proxy feature allows you to create multiple proxy terminals. This topic describes how to create proxy terminals for an ApsaraDB RDS for PostgreSQL instance.

Usage notes

Each RDS instance supports up to seven proxy terminals. You can create multiple proxy terminals to apply different read and write policies to different clients.

> **Note** For example, your database system consists of one primary RDS instance and four read-only RDS instances, and you have two applications, Application A and Application B. Application A initiates only read requests, and Application B initiates both read and write requests. In this case, you can bind two read-only RDS instances to Proxy Terminal A that has the Read-only attribute and bind the other two read-only RDS instances to Proxy Terminal B that has the Read/Write attribute. This way, Application A and Application B are physically isolated from each other in your database system.

Prerequisites

- The database proxy feature is enabled. For more information, see Enable and configure the database proxy feature for an ApsaraDB RDS for PostgreSQL instance.
- Multiple proxy instances are enabled. The number of proxy instances that you enabled is greater than the number of proxy terminals that you created. For more information, see Change the number of proxy instances on an ApsaraDB RDS for PostgreSQL instance.

Procedure

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. In the upper-right corner of the page, click **Create Proxy Terminal**.
4. In the dialog box that appears, configure the following parameters and click **OK**.

Parameter	Description
Custom Proxy Terminal	The name of the proxy terminal. The name can be up to 30 characters in length.

Parameter	Description
Read/Write Attribute	The read and write attributes of the proxy terminal. For more information about the read and write attributes, see the "Processing logic based on the read and write attributes" section of this topic. ○ **Read/Write (Read/Write Splitting)**: The proxy terminal connects to the primary RDS instance and the read-only RDS instances, and can receive write requests. This is the default attribute. If you select this option, your primary RDS instance can receive write requests, and you can specify a read weight for the primary RDS instance. If you select this option, make sure that the proxy terminal is associated with at least one primary RDS instance and one read-only RDS instance. All write requests are routed to the primary RDS instance. ○ **Read-only**: The proxy terminal connects only to the read-only RDS instances and cannot receive write requests. You cannot specify a read weight for your primary RDS instance. If you select this option, make sure that the proxy terminal is associated with at least one read-only RDS instance. The proxy terminal does not route requests to the primary RDS instance. If you select Read-only for a proxy terminal, the proxy terminal assigns connections to the associated read-only RDS instances based on the round-robin algorithm. Each database client is assigned only one connection to one read-only RDS instance. The proxy terminal does not assign the connection to the primary RDS instance. The total number of available connections is the sum of connections that are established to all the read-only RDS instances.
Latency Threshold	The maximum latency that is allowed for the read-only RDS instances to replicate data from the primary RDS instance. If the latency at which a read-only RDS instance replicates data from the primary RDS instance exceeds the value of this parameter, ApsaraDB RDS no longer routes read requests to the read-only RDS instance regardless of the read weight of the read-only RDS instance. Valid values: 0 to 3600. Unit: seconds. The read-only RDS instances may replicate data from the primary RDS instance at a specific latency. The latency varies based on the status of the SQL statements that are executed. We recommend that you set this parameter to a value that is greater than or equal to 30. **Note** This parameter appears only when you set the Read/Write Attribute parameter to **Read/Write (Read/Write Splitting)**.

Parameter	Description
Read Weight Distribution	The method that is used to assign read weights. A higher read weight indicates more read requests that must be processed. For example, three read-only RDS instances are attached to the primary RDS instance, the read weight of the primary RDS instance is 0, and the read weights of the three read-only RDS instances are 100, 200, and 200. In this case, the primary RDS instance processes only write requests, and the three read-only RDS instances process all read requests based on the 1:2:2 ratio. ○ **Automatic**: ApsaraDB RDS automatically assigns a read weight to each RDS instance in your database system based on the specifications of the RDS instance. After you create a read-only RDS instance, ApsaraDB RDS automatically assigns a read weight to the read-only RDS instance and adds the read-only RDS instance to the read/write splitting link. For more information, see Default read weights. ○ **Custom**: You must manually specify a read weight for each RDS instance in your database system based on your business requirements. Valid values: 0 to 10000. By default, the read weight of a read-only RDS is 0. After you create a read-only RDS instance, you must manually specify a read weight for the read-only RDS instance based on your business requirements. **Note** ○ You cannot specify weights for read-only RDS instances that have the data replication latency specified. ○ After you reconfigure this parameter, the new read weights immediately take effect and no transient connections occur. In addition, the existing connections remain open. Only the requests that are sent over new connections are routed based on the new weights. ○ After the RDS instances are released, the read weights automatically become invalid. ○ If your RDS instance fails or the data replication latency exceeds the specified threshold, the read weights automatically become invalid. After your RDS instance runs as normal, the read weights become valid again.

Delete a proxy terminal

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. On the page that appears, click the **Proxy Terminal** tab, find the proxy terminal that you want to delete, and then click **Delete Proxy Terminal**.

Processing logic based on the read and write attributes

Read and write attributes	Method to specify read weights	Weight of a primary RDS instance	Normal case	After the last read-only RDS instance is deleted	After all read-only RDS instances are faulty
Read-only	**Automatic** or **Custom**	You cannot specify a read weight for your primary RDS instance.	• Primary RDS instance: does not process read or write requests. • Proxy terminal: processes only read requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.
Read and write	Automatic	A weight greater than 0 For more information, see Default read weights.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.
	Custom	A weight greater than 0	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.
		A weight equal to 0	• Primary RDS instance: processes only write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.	• Primary RDS instance: processes read and write requests. • Proxy terminal: processes read and write requests.

Related operations

Operation	Description
DescribeDBProxy	Queries the details of a database proxy.

Operation	Description
ModifyDBProxyEndpoint	Creates, modifies, or deletes a database proxy terminal.
DescribeDBProxyEndpoint	Queries information about a database proxy terminal.

17.3.3. Manage the database proxy endpoints of an ApsaraDB RDS for PostgreSQL instance

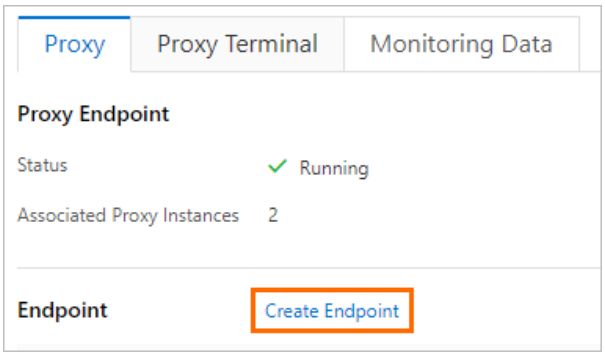
This topic describes how to manage the database proxy endpoints of an ApsaraDB RDS for PostgreSQL instance. After the database proxy feature is enabled, a default database proxy endpoint is generated. The proxy terminal feature is bound to this endpoint. You can create, modify, or delete a database proxy endpoint.

Prerequisites

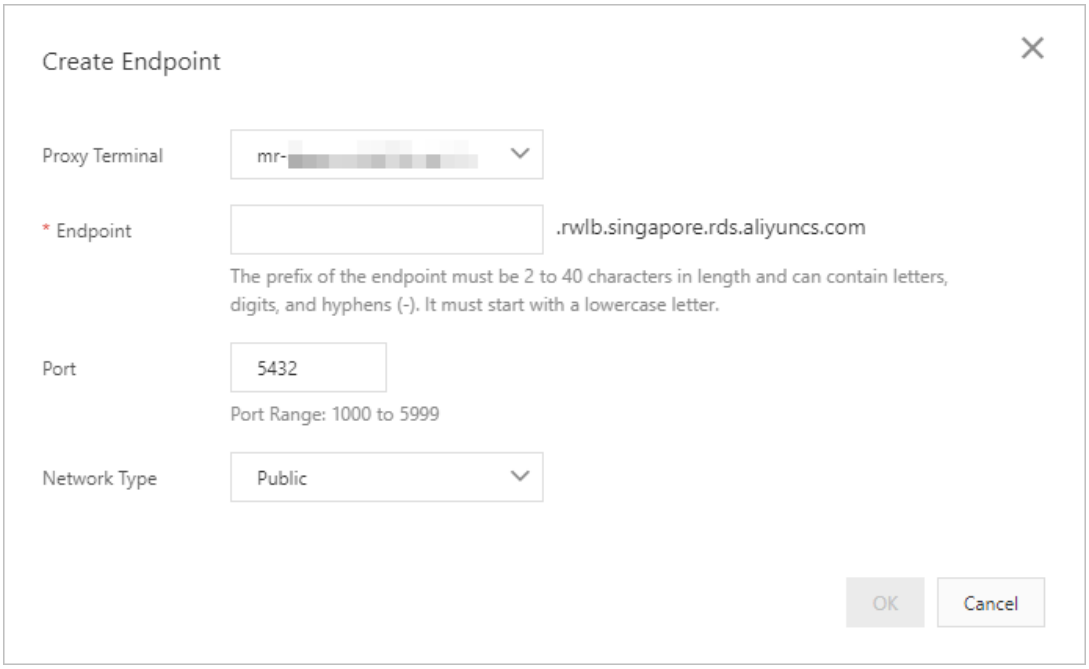
The database proxy feature is enabled. For more information, see Enable and configure the database proxy feature for an ApsaraDB RDS for PostgreSQL instance.

Create a database proxy endpoint

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. In the **Endpoint** section of the **Proxy** tab, click **Create Endpoint**.



4. Configure the **Proxy Terminal**, **Endpoint**, **Port**, and **Network Type** parameters. Then, click **OK**.



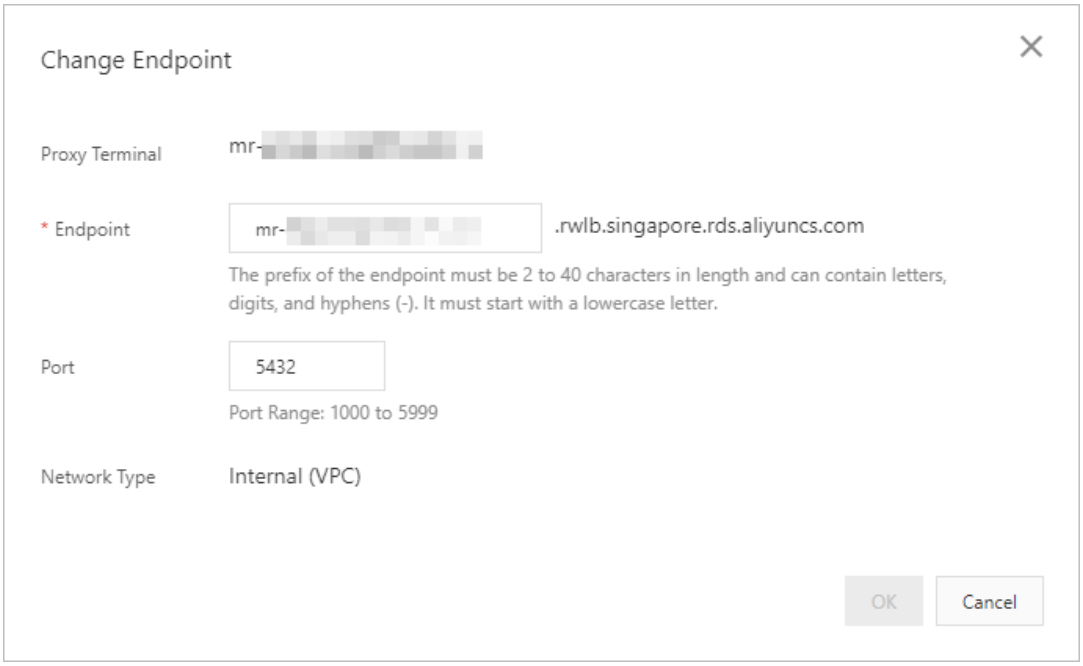
Note

- You can create only one database proxy endpoint for which the **Network Type** parameter is set to **Public**.
- The prefix of a database proxy endpoint must be 1 to 40 characters in length and can contain letters, digits, and hyphens (-). The prefix must start with a lowercase letter.
- The port number that is bound to a database proxy endpoint must be within the range of 1000 to 5999.
- After a database proxy endpoint is created, you cannot change the value of the **Network Type** parameter for the endpoint.

Modify a database proxy endpoint

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. In the **Endpoint** section of the **Proxy** tab, find the database proxy endpoint that you want to modify and click **Change** to the right of the endpoint.

4. Change the values of the **Endpoint** and **Port** parameters and click **OK**.

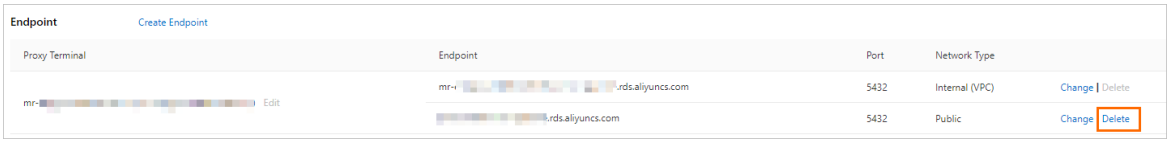


> **Note**
>
> - The prefix of a database proxy endpoint must be 1 to 40 characters in length and can contain letters, digits, and hyphens (-). The prefix must start with a lowercase letter.
> - The port number that is bound to a database proxy endpoint must be within the range of 1000 to 5999.
> - You cannot change the value of the **Network Type** parameter for the endpoint.

Delete a database proxy endpoint

> **Note** You cannot delete a database proxy endpoint whose network type is **Internal (VPC)**.

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. Find the database proxy endpoint that you want to delete and click **Delete** to the right of the endpoint. In the message that appears, click **OK**.



> **Note** You can delete only the public endpoint that you created.

Related operations

Operation	Description
DescribeDBProxy	Queries the details of a database proxy.

Operation	Description
CreateDBProxyEndpointAddress	Creates a database proxy endpoint.
ModifyDBProxyEndpointAddress	Modifies a database proxy endpoint.
DeleteDBProxyEndpointAddress	Deletes a database proxy endpoint.

17.3.4. View the proxy monitoring data of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to view the proxy monitoring data of an ApsaraDB RDS for PostgreSQL instance. You can check the performance metrics of the enabled proxy instances, obtain information about the loads on the enabled proxy instances, and change the number of proxy instances based on the monitoring data.

Prerequisites

The database proxy feature is enabled. For more information, see Enable and configure the database proxy feature for an ApsaraDB RDS for PostgreSQL instance.

Procedure

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. On the page that appears, click the **Monitoring Data** tab.
4. Specify a time range from which you want to view the metrics of the proxy instance.

   The following metrics are provided: **Connections per Second**, **Current Connections**, **Outbound Traffic**, **Inbound Traffic**, **Queries per Second (QPS)**, **Memory Usage**, and **CPU Utilization**.

   The monitoring frequency varies based on the time range that you specify for the query.

   - If the time range is less than or equal to 1 hour, the time granularity is 5 seconds.
   - If the time range is greater than 1 hour and less than or equal to 2 hours, the time granularity is 10 seconds.
   - If the time range is greater than 2 hours and less than or equal to 6 hours, the time granularity is 30 seconds.
   - If the time range is greater than 6 hours and less than or equal to 12 hours, the time granularity is 1 minute.
   - If the time range is greater than 12 hours and less than or equal to 1 day, the time granularity is 2 minutes.
   - If the time range is greater than 1 day and less than or equal to 5 days, the time granularity is 10 minutes.
   - If the time range is greater than 5 days and less than or equal to 15 days, the time granularity is 30 minutes.
   - If the time range is greater than 15 days and less than or equal to 30 days, the time granularity is 1 hour.

Related operations

Operation	Description
DescribeDBProxyPerformance	Queries the performance data of a proxy instance.

17.3.5. Change the number of proxy instances on an ApsaraDB RDS for PostgreSQL instance

This topic describes how to change the number of proxy instances on an ApsaraDB RDS for PostgreSQL instance based on monitoring data and business planning. This helps improve proxy performance.

Prerequisites

The database proxy feature is enabled. For more information, see Enable and configure the database proxy feature for an ApsaraDB RDS for PostgreSQL instance.

Precautions

When you change the number of proxy instances on an RDS instance, a transient connection occurs on your application. Make sure that your application is configured to automatically reconnect to your RDS instance.

Procedure

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. In the **Proxy Instance** section of the **Proxy** tab, change the value of the **Enabled Proxy Instances** parameter. Then, click **Apply**.

   Note

   - We recommend that you set the Proxies parameter to `one-eighth of the total number of cores that are configured for your primary RDS instance and its read-only RDS instances`. If the result is not an integer, you must round up the result to the nearest integer. Up to 60 proxy instances are supported.

     For example, if your primary RDS instance has eight cores and its read-only RDS instance has four cores, we recommend that you specify the Proxies parameter based on the following calculation: `(8 + 4)/8 = 1.5`. The result 1.5 is rounded up to 2.
   - The number of proxy instances is used to calculate the performance of the database proxy. The number does not represent the actual number of proxy instances. If you set the Proxies parameter to a large value, the database proxy can process more requests of your RDS instance.

     For example, if a single proxy instance is configured with `2 cores and 4 GB of memory` and you enable two proxy instances, the database proxy of your RDS instance can provide `4 cores and 8 GB of memory`.
4. In the dialog box that appears, configure the **Applied At** parameter and click **OK**.

Related operations

Operation	Description
ModifyDBProxyInstance	Changes the number of proxy instances.
DescribeDBProxy	Queries the details of a database proxy.

17.3.6. Upgrade the database proxy version of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to upgrade the database proxy version of an ApsaraDB RDS for PostgreSQL instance.

Prerequisites

The database proxy feature is enabled. For more information, see Enable and configure the database proxy feature for an ApsaraDB RDS for PostgreSQL instance.

Precautions

When you upgrade the database proxy version of an RDS instance, the proxy instances that you enabled on the RDS instance are restarted. During the restart process, a transient connection occurs for approximately 30 seconds. The point in time when the proxy instances restart varies based on the value of the **Upgrade Time** parameter. You can select **Upgrade Now** or **Upgrade Within Maintenance Window** for the parameter. We recommend that you upgrade the database proxy version of your RDS instance during off-peak hours. If you want to upgrade the database proxy version of your RDS instance during peak hours, make sure that your application is configured to automatically reconnect to your RDS instance.

Procedure

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. On the Proxy tab of the page that appears, click **Upgrade Database Proxy Version**.
4. In the dialog box that appears, configure the **Upgrade Time** parameter and click **OK**.

> **Note**
>
> - You can select **Upgrade Now** or **Upgrade Within Maintenance Window** for the Upgrade Time parameter. For more information about the maintenance window, see Set the maintenance window of an ApsaraDB RDS for PostgreSQL instance.
> - After you set the **Upgrade Time** parameter to **Upgrade Within Maintenance Window** and click **OK**, the RDS instance generates an upgrade plan based on the current maintenance window. If you modify the maintenance window, the upgrade plan that is generated is not affected. If you want to cancel or modify the upgrade plan that is generated, submit a .

Related operations

Operation	Description
UpgradeDBProxyInstanceKernelVersion	Upgrades the database proxy version.

17.3.7. Disable the database proxy feature

This topic describes how to disable the database proxy feature for an ApsaraDB RDS for PostgreSQL instance.

Prerequisites

The database proxy feature is enabled. For more information, see Enable and configure the database proxy feature for an ApsaraDB RDS for PostgreSQL instance.

Precautions

- When you disable the database proxy feature, the proxy terminal feature is disabled.
- If you disable the database proxy feature, database proxy endpoints become unavailable. The endpoints of your primary RDS instance are not affected.
- If you enable the database proxy feature after you disable the feature, the database proxy endpoints change.

Procedure

1.
2. In the left-side navigation pane, click **Database Proxy**.
3. In the upper-right corner of the page that appears, click **Disable Proxy**.
4. In the message that appears, click **OK**.

Related operations

Operation	Description
ModifyDBProxy	Enables or disables the database proxy feature.
DescribeDBProxy	Queries the details of a database proxy.

17.4. Default read weights

This topic describes the default read weights for ApsaraDB RDS for PostgreSQL instances that use different specifications. If you set **Read Weight Distribution** to **Automatic** when you create a database proxy terminal, you can check the default read weights for RDS instances that use different specifications in this topic.

Primary RDS instances

Instance type	CPU and memory specifications	Read weight
pg.n2.small.1	1 core, 2 GB	100
pg.n2.medium.1	2 cores, 4 GB	200

Instance type	CPU and memory specifications	Read weight
pg.n4.medium.1	2 cores, 8 GB	200
pg.x2.medium.2c	2 cores, 4 GB	200
pg.x4.medium.2c	2 cores, 8 GB	200
pg.x4t.medium.2c	2 cores, 8 GB	200
pg.x8.medium.2c	2 cores, 16 GB	200
pg.x8t.medium.2c	2 cores, 16 GB	200
pg.n2.large.1	4 cores, 8 GB	400
pg.n4.large.1	4 cores, 16 GB	400
pg.x2.large.2c	4 cores, 8 GB	400
pg.x2t.large.2c	4 cores, 8 GB	400
pg.x4.large.2c	4 cores, 16 GB	400
pg.x4t.large.2c	4 cores, 16 GB	400
pg.x8.large.2c	4 cores, 32 GB	400
pg.x8t.large.2c	4 cores, 32 GB	400
pg.n2.xlarge.1	8 cores, 16 GB	800
pg.n4.xlarge.1	8 cores, 32 GB	800
pg.x2.xlarge.2c	8 cores, 16 GB	800
pg.x2t.xlarge.2c	8 cores, 16 GB	800
pg.x4.xlarge.2c	8 cores, 32 GB	800
pg.x4t.xlarge.2c	8 cores, 32 GB	800
pg.x8.xlarge.2c	8 cores, 64 GB	800
pg.x8t.xlarge.2c	8 cores, 64 GB	800
pg.x2.3large.2c	12 cores, 24 GB	1,200
pg.x2t.3large.2c	12 cores, 24 GB	1,200
pg.x4.3large.2c	12 cores, 48 GB	1,200
pg.x4t.3large.2c	12 cores, 48 GB	1,200
pg.x8.3large.2c	12 cores, 96 GB	1,200

Instance type	CPU and memory specifications	Read weight
pg.x8t.3large.2c	12 cores, 96 GB	1,200
pg.n2.2xlarge.1	16 cores, 32 GB	1,600
pg.n4.2xlarge.1	16 cores, 64 GB	1,600
pg.n8.2xlarge.1	16 cores, 128 GB	1,600
pg.x2.2xlarge.2c	16 cores, 32 GB	1,600
pg.x2t.2xlarge.2c	16 cores, 32 GB	1,600
pg.x4.2xlarge.2c	16 cores, 64 GB	1,600
pg.x4t.2xlarge.2c	16 cores, 64 GB	1,600
pg.x8.2xlarge.2c	16 cores, 128 GB	1,600
pg.x8t.2xlarge.2c	16 cores, 128 GB	1,600
pg.x2.3xlarge2c	24 cores, 48 GB	2,400
pg.x2t.3xlarge.2c	24 cores, 48 GB	2,400
pg.x4.3xlarge.2c	24 cores, 96 GB	2,400
pg.x4t.3xlarge.2c	24 cores, 96 GB	2,400
pg.x8.3xlarge.2c	24 cores, 192 GB	2,400
pg.x8t.3xlarge.2c	24 cores, 192 GB	2,400
pg.n4.4xlarge.1	32 cores, 128 GB	3,200
pg.n8.4xlarge.1	32 cores, 256 GB	3,200
pg.x2.4xlarge.2c	32 cores, 64 GB	3,200
pg.x2t.4xlarge.2c	32 cores, 64 GB	3,200
pg.x4.4xlarge.2c	32 cores, 128 GB	3,200
pg.x4t.4xlarge.2c	32 cores, 128 GB	3,200
pg.x8.4xlarge.2c	32 cores, 256 GB	3,200
pg.x8t.4xlarge.2c	32 cores, 256 GB	3,200
pg.x2.13large.2c	52 cores, 104 GB	5,200
pg.x4.13large.2c	52 cores, 192 GB	5,200
pg.x8.13large.2c	52 cores, 384 GB	5,200

Instance type	CPU and memory specifications	Read weight
pg.n4.8xlarge.1	56 cores, 256 GB	5,600
pg.n8.8xlarge.1	56 cores, 512 GB	5,600
pg.x2.8xlarge.2c	64 cores, 128 GB	6,400
pg.x2t.8xlarge.2c	64 cores, 128 GB	6,400
pg.x4.8xlarge.2c	64 cores, 256 GB	6,400
pg.x4t.8xlarge.2c	64 cores, 256 GB	6,400
pg.x8.8xlarge.2c	64 cores, 512 GB	6,400
pg.x8t.8xlarge.2c	64 cores, 512 GB	6,400
pg.x2.13xlarge.2c	104 cores, 192 GB	10,400
pg.x4.13xlarge.2c	104 cores, 384 GB	10,400
pg.x8.13xlarge.2c	104 cores, 768 GB	10,400

Read-only RDS instances

Instance type	CPU and memory specifications	Read weight
pgro.x2.medium.1c	2 cores, 4 GB	200
pgro.x4.medium.1c	2 cores, 8 GB	200
pgro.x8.medium.1c	2 cores, 16 GB	200
pgro.x2.large.1c	4 cores, 8 GB	400
pgro.x4.large.1c	4 cores, 16 GB	400
pgro.x8.large.1c	4 cores, 32 GB	400
pgro.x2.xlarge.1c	8 cores, 16 GB	800
pgro.x4.xlarge.1c	8 cores, 32 GB	800
pgro.x8.xlarge.1c	8 cores, 64 GB	800
pgro.x2.3large.1c	12 cores, 24 GB	1,200
pgro.x4.3large.1c	12 cores, 48 GB	1,200
pgro.x8.3large.1c	12 cores, 96 GB	1,200
pgro.x2.2xlarge.1c	16 cores, 32 GB	1,600
pgro.x4.2xlarge.1c	16 cores, 64 GB	1,600

Instance type	CPU and memory specifications	Read weight
pgro.x8.2xlarge.1c	16 cores, 128 GB	1,600
pgro.x2.3xlarge.1c	24 cores, 48 GB	2,400
pgro.x4.3xlarge.1c	24 cores, 96 GB	2,400
pgro.x8.3xlarge.1c	24 cores, 192 GB	2,400
pgro.x2.4xlarge.1c	32 cores, 64 GB	3,200
pgro.x4.4xlarge.1c	32 cores, 128 GB	3,200
pgro.x8.4xlarge.1c	32 cores, 256 GB	3,200
pgro.x2.13large.1c	52 cores, 104 GB	5,200
pgro.x4.13large.1c	52 cores, 192 GB	5,200
pgro.x8.13large.1c	52 cores, 384 GB	5,200
pgro.x2.8xlarge.1c	64 cores, 128 GB	6,400
pgro.x4.8xlarge.1c	64 cores, 256 GB	6,400
pgro.x8.8xlarge.1c	64 cores, 512 GB	6,400
pgro.x2.13xlarge.1c	104 cores, 192 GB	10,400
pgro.x4.13xlarge.1c	104 cores, 384 GB	10,400
pgro.x8.13xlarge.1c	104 cores, 768 GB	10,400

18.Account

18.1. Create an account on an ApsaraDB RDS for PostgreSQL instance

This topic describes how to create an account on an ApsaraDB RDS for PostgreSQL instance.

Account types

ApsaraDB RDS for PostgreSQL instances support two types of accounts: privileged accounts and standard accounts. The following table describes these types of accounts.

Account type	Description
Privileged account	• You can create and manage privileged accounts in the ApsaraDB RDS console or by using the ApsaraDB RDS API. • You can create multiple privileged accounts for each RDS instance. The privileged accounts of an RDS instance have the permissions to manage all standard accounts and databases that are created on the instance. • A privileged account allows you to manage permissions at fine-grained levels based on your business requirements. For example, you can grant each standard account the permissions to query specific tables. • A privileged account has the permissions to log off all standard accounts on the instance on which the privileged account is created. **Note** • The first privileged account that you create is the owner of the default public schema of a standard system database named template1. • By default, the **CREATE DATABASE** statement creates a database by replicating the template1 system database. The owners of all databases that are created by using this statement from the template1 system database are the first privileged account. • The comment of the first privileged account starts with "template1 public schema owner."
Standard account	• You can create and manage standard accounts in the ApsaraDB RDS console, by using the ApsaraDB RDS API, or by executing SQL statements. • You can create multiple standard accounts for each RDS instance. • You must grant the permissions on specified databases to standard accounts. • You cannot use a standard account to create, manage, or log off other accounts from the instance on which the standard account is created.

Precautions

- You can create multiple privileged accounts and standard accounts in the ApsaraDB RDS console. You can also create and manage standard accounts by using SQL statements.

- Before you can migrate data from an on-premises database to an RDS instance, you must create a database and an account on the RDS instance. Make sure that the created database has the same properties as the on-premises database. In addition, make sure that the created account has the same permissions on the created database as the account that is authorized to manage the on-premises database.
- We recommend that you follow the principle of least privilege (PoLP) and grant the read and write permissions to accounts based on your business requirements. You can create multiple accounts and grant each account only the permissions to access the data of specified databases. If an account does not need to write data to a database, we recommend that you grant only the read permissions on the database to the account.
- For security purposes, we recommend that you specify strong passwords for accounts and change the passwords on a regular basis.

Procedure

1.
2. In the left-side navigation pane, click **Accounts**.
3. Click **Create Account**.
4. Configure the following parameters.

Parameter	Description
Database Account:	◦ The username of the account must be 2 to 63 characters in length. ◦ The username of the account can contain lowercase letters, digits, and underscores (_). ◦ The username of the account must start with a lowercase letter and end with a lowercase letter or a digit. ◦ The username of the account cannot be the same as the username of an existing account. ◦ The username of the account cannot start with pg. ◦ The username of the account cannot contain SQL keywords. For more information, see SQL Keywords.
Account Type:	Specify the type of the account. Two types of accounts are supported: privileged accounts and standard accounts. ◦ A privileged account has all operation permissions on all databases. ◦ Standard accounts have all operation permissions only on their authorized databases. **Note** The operation permissions include SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, and TRIGGER.

Parameter	Description
Password:	○ The password of the account must be 8 to 32 characters in length. ○ The password of the account must contain at least three of the following character types: uppercase letters, lowercase letters, digits, and special characters. ○ The password of the account can contain any of the following special characters: ! @ # $ % ^ & * ( ) _ + - =
Confirm Password:	Enter the password of the account again.
Description	Enter the description of the account.

5. Click **OK**.

Related operations

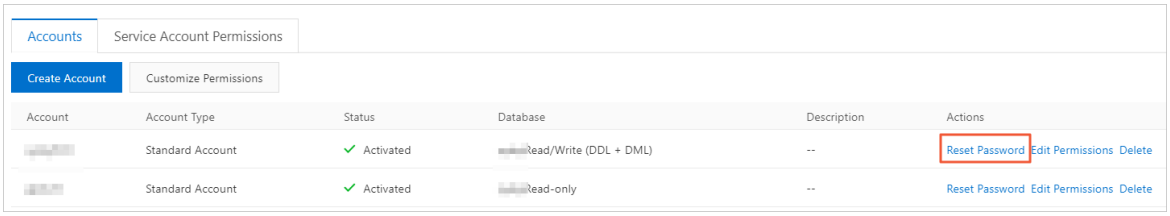
Operation	Description
CreateAccount	Creates an account that is used to manage the databases of an ApsaraDB RDS instance.

18.2. Reset the password of an account on an ApsaraDB RDS for PostgreSQL instance

This topic describes how to reset the password of an account on your ApsaraDB RDS for PostgreSQL instance. You can reset the password if the password is lost.

Procedure

1.
2. In the left-side navigation pane, click **Accounts**.
3. Find the account whose password you want to reset, and click **Reset Password** in the Actions column.



4. In the dialog box that appears, specify a new password, confirm the new password, and then click **Create**.

> **Note** The password must meet the following requirements:
>
> - The password must be 8 to 32 characters in length.
> - The password must contain at least three of the following character types: uppercase letters, lowercase letters, digits, and special characters.
> - The password can contain any of the following characters:
>
>   ! @ # $ % ^ & * ( ) _ + - =

Related operations

Operation	Description
ResetAccountPassword	Resets the password of an account on an ApsaraDB RDS instance.

18.3. Authorize the service account of an RDS PostgreSQL instance

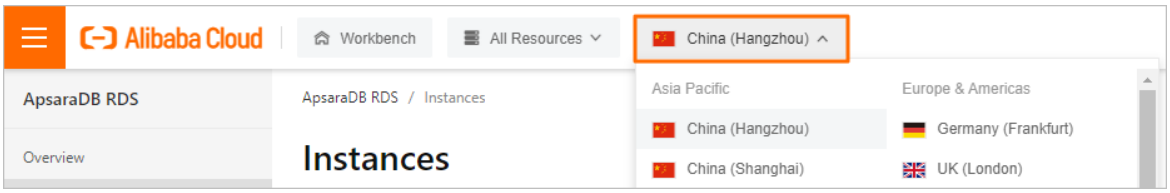
When you seek help from Alibaba Cloud technical support to locate problems that occurred in your RDS PostgreSQL instance, you may need to grant permissions to the service account of your instance. The service account is used by Alibaba Cloud technical support to perform operations on the databases in your instance. After the specified expiration time, the system deletes the service account.

Prerequisites

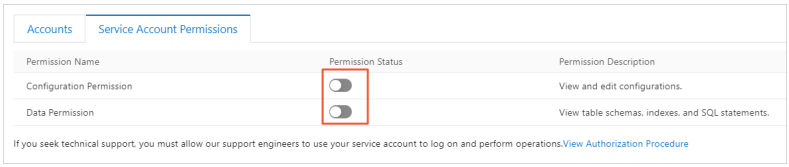
Your RDS PostgreSQL instance is equipped with standard SSDs or enhanced SSDs (ESSDs).

Procedure

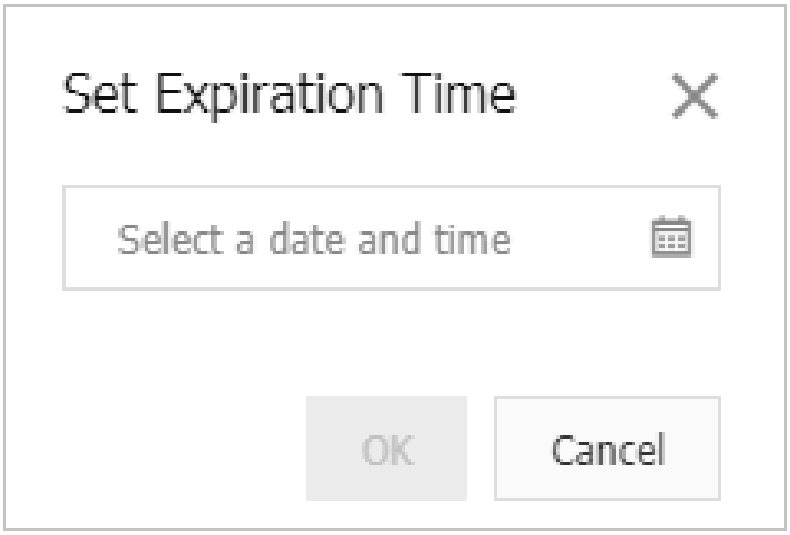
1. Log on to the ApsaraDB for RDS console.
2. In the top navigation bar, select the region where the target RDS instance resides.



3. Find the target RDS instance and click its ID.
4. In the left-side navigation pane, click **Accounts**.
5. Click the **Service Account Permissions** tab, find the permissions you want to grant to the service account, and click the slider in the **Permission Status** column.
   - For problems related to IP address whitelists or database parameters, you can grant only the Configuration Permission to the service account.
   - For database performance deterioration caused by application defects, you must grant the Data Permission to the service account.

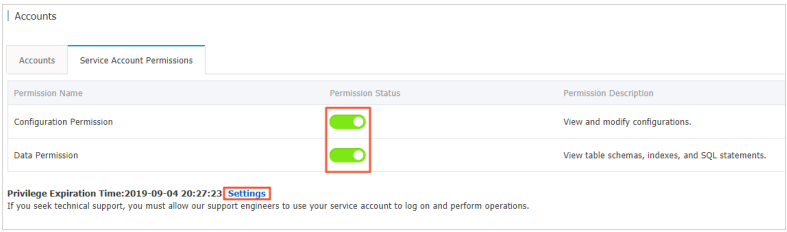


6. In the dialog box that appears, specify an expiration time and click **OK**.



Revoke the permissions or change the expiration time of the service account

After you grant permissions to the service account, you can revoke the permissions or change the account expiration time on the **Service Account Permissions** tab in the ApsaraDB for RDS console.



18.4. Lock or delete an account from an ApsaraDB RDS for PostgreSQL instance

This topic describes how to lock or delete an account from an ApsaraDB RDS for PostgreSQL instance.

Lock an account in the ApsaraDB RDS console

> **Note** After an account is locked, you cannot use the account to log on to the RDS instance.

1.
2. In the left-side navigation pane, click **Accounts**.
3. Find the account that you want to lock, and click **Lock Instance** in the **Actions** column. In the

message that appears, click OK.

> **Note** Wait a few minutes. When the **Status** column displays **Locking**, the account is locked. If you want to use the account again, click **Unlock** in the **Actions** column.

Delete an account in the ApsaraDB RDS console

> **Note** If the account that you want to delete is granted permissions on databases, tables, or other objects, the "`Some objects depend on account`" message is displayed when you attempt to delete the account. You can delete the account only after you remove the permissions of the account.

1.
2. In the left-side navigation pane, click **Accounts**.
3. Find the account that you want to delete, and click **Delete** in the **Actions** column.
4. In the message that appears, click **OK**.

Delete a standard account by using SQL statements

> **Note** If the account that you want to delete is granted permissions on databases, tables, or other objects, the "`ERROR: current user cannot be dropped`" message is displayed when you attempt to delete the account. You can delete the account only after you remove the permissions of the account.

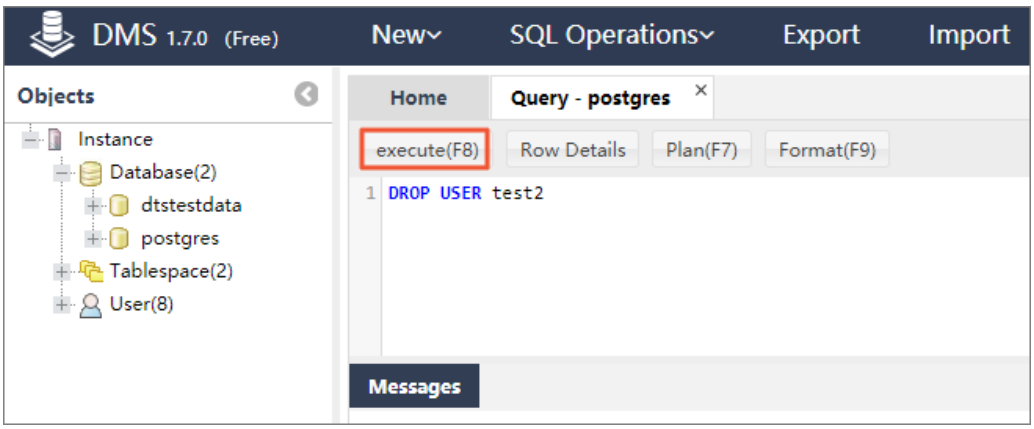
1. Use Alibaba Cloud Data Management (DMS) to log on to the RDS instance. For more information, see Use DMS to log on to an ApsaraDB RDS for PostgreSQL instance.
2. In the top navigation bar, click **SQL Console**.
3. Execute the following statement to grant the permissions of the account to another account:

```
REASSIGN OWNED BY <The username of the account that you want to delete> TO <The username of another account>;
```

4. Execute the following SQL statement to delete the account:

```
DROP USER <The username of the account that you want to delete>;
```

5. Click **execute**.



Related operations

Operation	Description
Delete an account	Deletes an account from an ApsaraDB RDS instance.
Lock account	Locks an account of an ApsaraDB RDS instance.
Unlock account	Unlocks an account of an ApsaraDB RDS instance.

18.5. Account permissions

This topic describes the permissions of the privileged account and standard account that are created on an ApsaraDB RDS for PostgreSQL instance.

Permission	Privileged account	Standard account
CREATEDB	✔	□
CREATEROLE	✔	□
REPLICATION	✔	□
SELECT	✔	✔
INSERT	✔	✔
UPDATE	✔	✔
DELETE	✔	✔
TRUNCATE	✔	✔
REFERENCES	✔	✔
TRIGGER	✔	✔
CREATE	✔	✔
CONNECT	✔	✔
TEMPORARY	✔	✔
EXECUTE	✔	✔
USAGE	✔	✔

Note In the preceding table, ticks (√) indicate that an account has the permission and crosses (×) indicate that an account does not have the permission.

18.6. Connect an ApsaraDB RDS for PostgreSQL instance to a self-managed AD domain

This topic describes how to configure an Active Directory (AD) domain controller on an Elastic Compute Service (ECS) instance and connect an ApsaraDB RDS for PostgreSQL instance to a self-managed AD domain.

Context

AD is a directory service that is provided by Microsoft. A directory is a hierarchical structure that stores information about the objects on the same LAN. An enterprise can store data, such as computer accounts, user accounts, and groups, in a directory. This way, the enterprise can improve the security of the data and manage the data in a more convenient manner.

You can connect your RDS instance to a self-managed AD domain. This way, you can manage your enterprise in a centralized manner and can configure IP address whitelists at the database level and the user level to improve the security of your data.

> **Note** You can modify the information of the AD domain controller in the *pg_hba.conf* file of your RDS instance or add the information of the AD domain controller to the *pg_hba.conf* file of your RDS instance. You can configure the AD domain controller and the pg_hba.conf file in the ApsaraDB RDS console. For more information, see Introduction of pg_hba.conf file.

Prerequisites

- The following requirements are met:
  - The RDS instance runs a major engine version of PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, or PostgreSQL 14.
  - The RDS instance runs a minor engine version of 20210228 or later. For more information about how to update the minor engine version of your RDS instance, see Update the minor engine version of an ApsaraDB RDS for PostgreSQL instance.
  - The RDS instance uses standard SSDs or enhanced SSDs (ESSDs).
  - The RDS instance does not use a new general-purpose instance type.

    > **Note** The new general-purpose instance types provide better scalability and performance and reduce the time to create an RDS instance or change the specifications of an RDS instance. The new general-purpose instance types do not support the self-managed AD domain controller connection feature. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types.

- An ECS instance is created. For more information, see Create an ECS instance. Your RDS instance must access the self-managed AD domain by using a private IP address. Therefore, the ECS instance must meet the following conditions:
  - The ECS instance and your RDS instance reside in the same virtual private cloud (VPC).
  - The security group to which the ECS instance belongs is configured to allow access from the private IP address of your RDS instance. For more information, see Add security group rules.

- The firewall of the ECS instance is disabled by default. If the firewall is enabled for the ECS instance, you must configure the firewall to allow access from the private IP address of your RDS instance.
- The image of the ECS instance runs Windows Server 2016 or a later version.

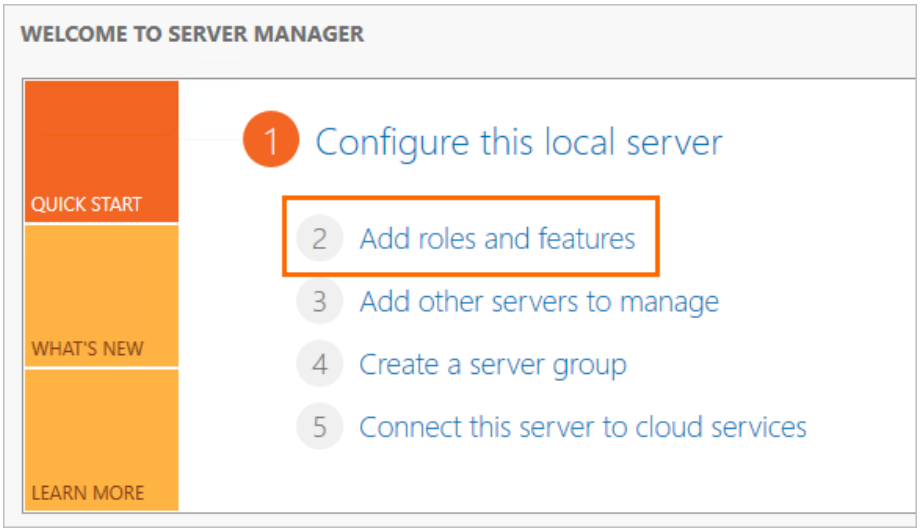
- The domain account belongs to the Domain Admins group.
- Your Alibaba Cloud account is used to log on to the ApsaraDB RDS console.

Procedure

1. Configure an AD domain controller for the ECS instance.
   i. Log on to the ECS instance.

   > **Note** The AD domain controller must run a Windows Server system. We recommend that you use Windows Server 2016 or a later version. In this example, the AD domain controller runs Windows Server 2016.

   ii. Search for and open **Server Manager**.
   iii. In the left-side navigation pane, click Dashboard. On the **Dashboard** page, click **Add roles and features**.

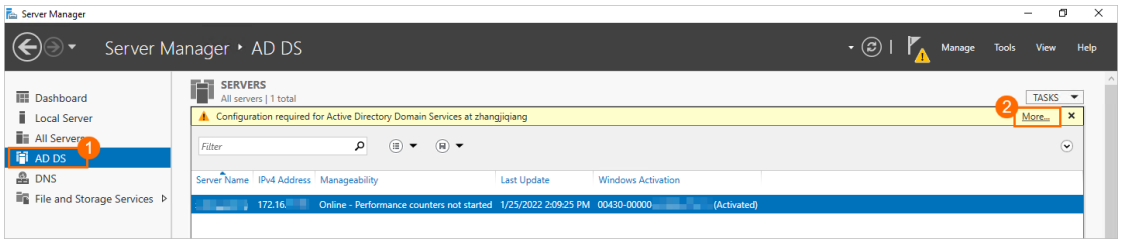


iv. In the **Add Roles and Features Wizard**, configure the following parameters.

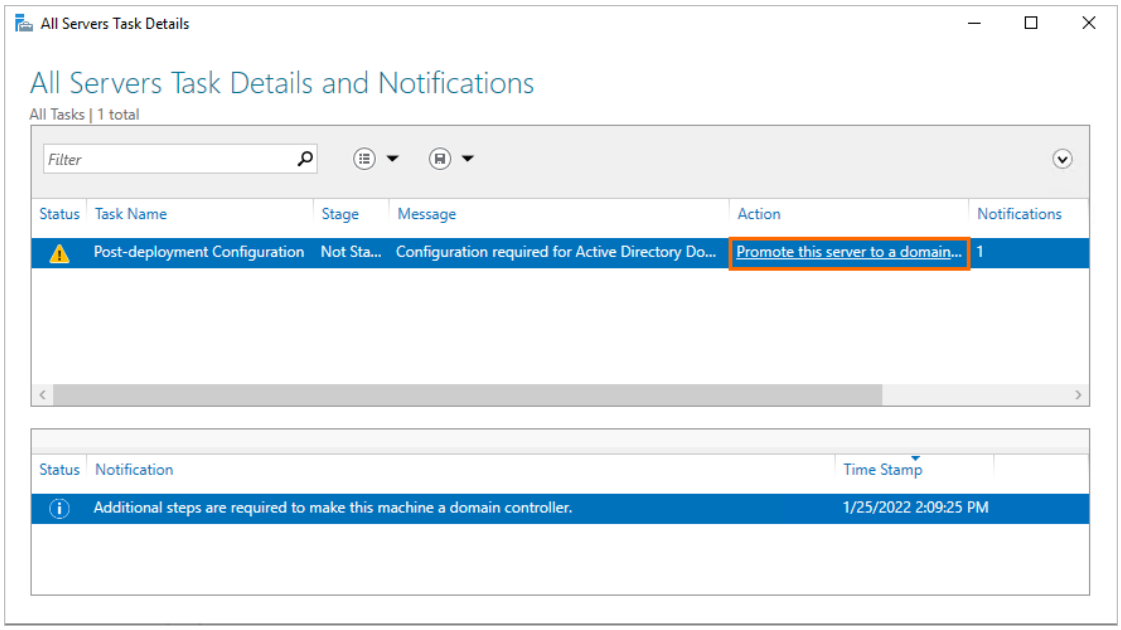
Tab	Description
Before You Begin	Use the default settings.
Installation Type	Use the default settings.
Server Selection	Use the default settings.
Server Roles	■ Select **Active Directory Domain Services**. In the dialog box that appears, click **Add Features**. ■ Select **DNS Server**. In the dialog box that appears, click **Add Features**. **Note** Make sure that your computer uses a fixed IP address. If the IP address dynamically changes, the DNS server becomes unavailable.
Features	Use the default settings.
AD DS	Use the default settings.
DNS Server	Use the default settings.
Install	Click **Install** to add the role that you configured.

v. After the role is added, click **Close** to close the wizard.

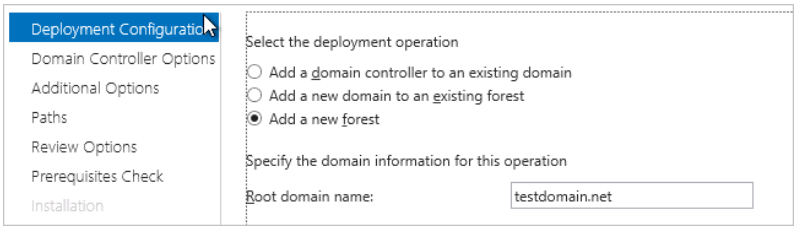
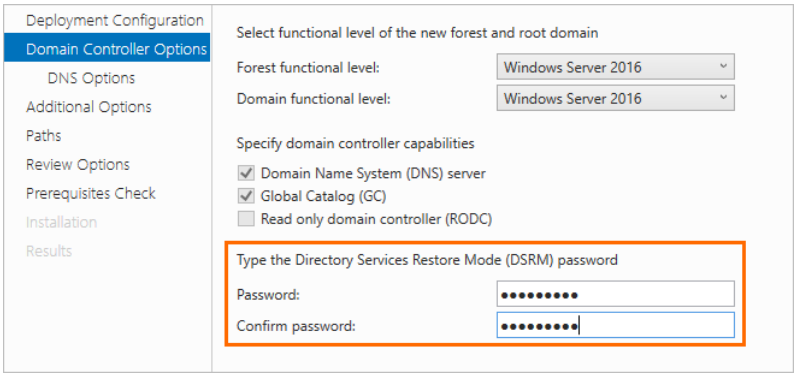
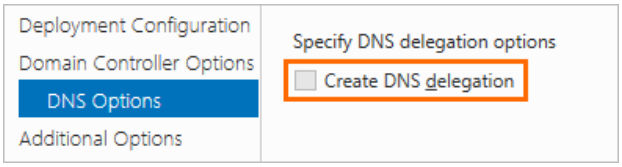
vi. In the left-side navigation pane of **Server Manager**, click **AD DS**. In the upper-right corner of the page that appears, click **More**.



vii. In the **All Servers Task Details and Notifications** dialog box, click **Promote this server to a domain controller.**



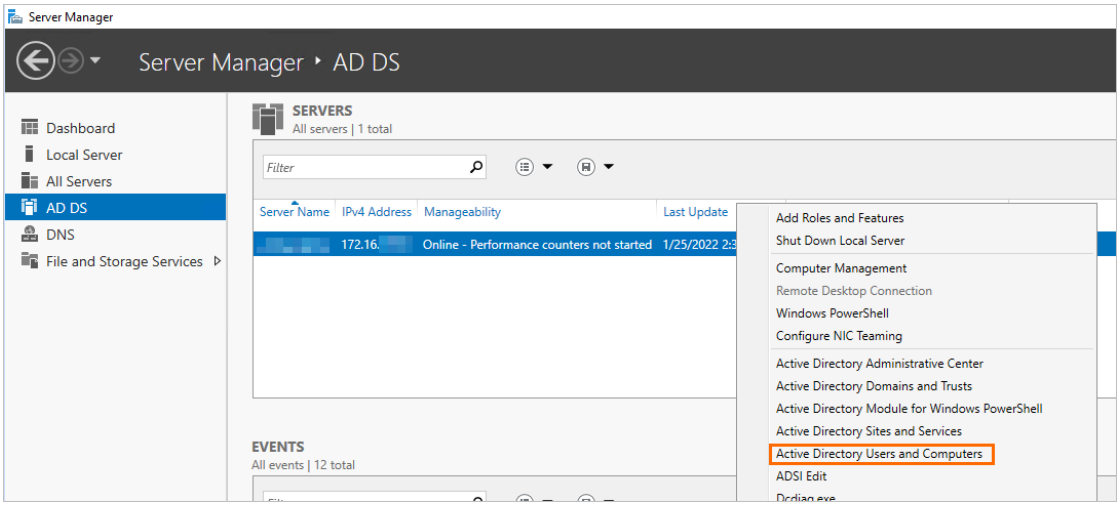
viii. In the **Active Directory Domain Services Configuration Wizard**, configure the following parameters.

Tab	Description
Deployment Configuration	Select **Add a new forest** and configure **Root domain name**. In this example, enter `pgsqldomain.net`. 
Domain Controller Options	Set a **Directory Services Restore Mode (DSRM) password**. 
Server Selection	Clear **Create DNS delegation**. 
Additional Options	Use the default settings.
Paths	Use the default settings.
Review Options	Use the default settings.
Prerequisites Check	Click **Install**.

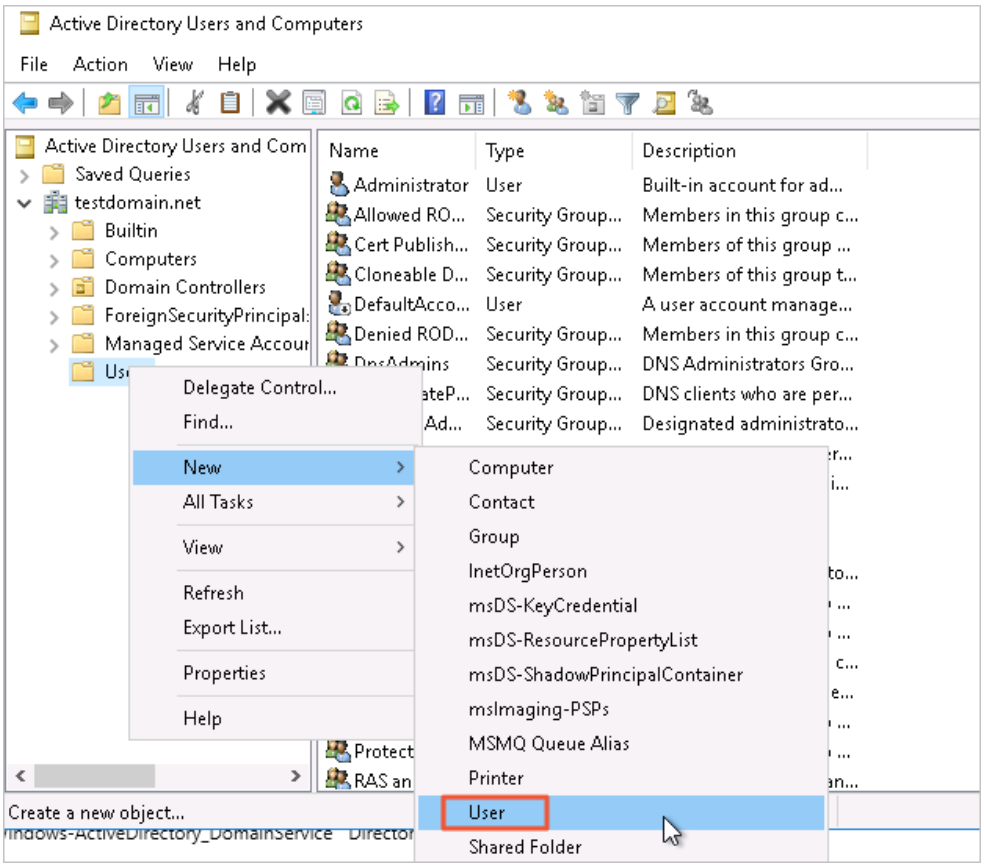
> **Note** After the ECS instance is promoted to an AD domain controller, you must restart the ECS instance. Then, you can perform the subsequent steps.

2. Create an administrator user for the AD domain controller.
   i. Log on to the ECS instance. Then, search for and open **Server Manager**.

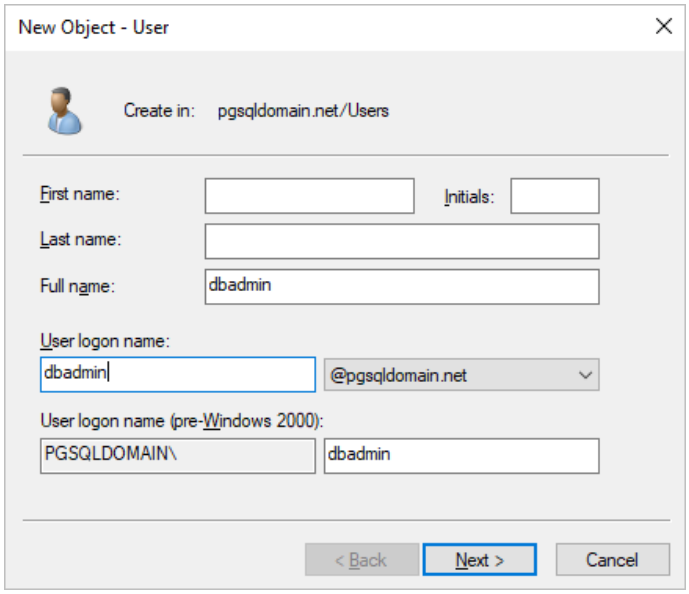
ii. In the left-side navigation pane of **Server Manager** page, click **AD DS**, right-click the AD domain controller that you want to configure, and then select **Active Directory Users and Computers**.



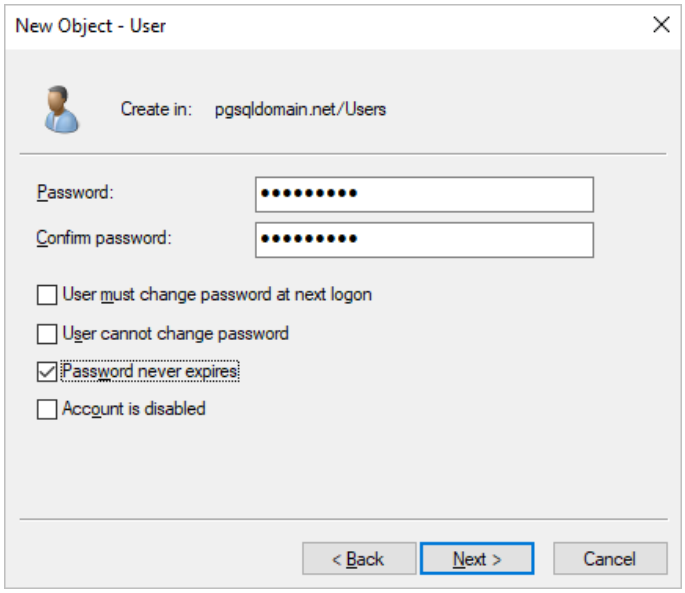
iii. Click **pgsqldomain.net**, right-click **Users**, and then choose **New > User**.



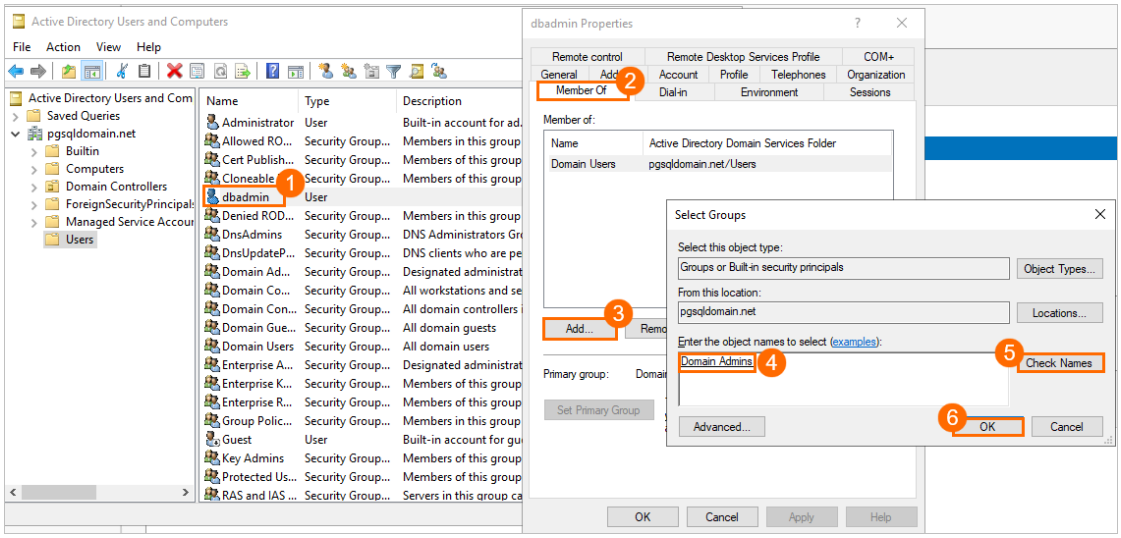
iv. Specify a username and click **Next**.



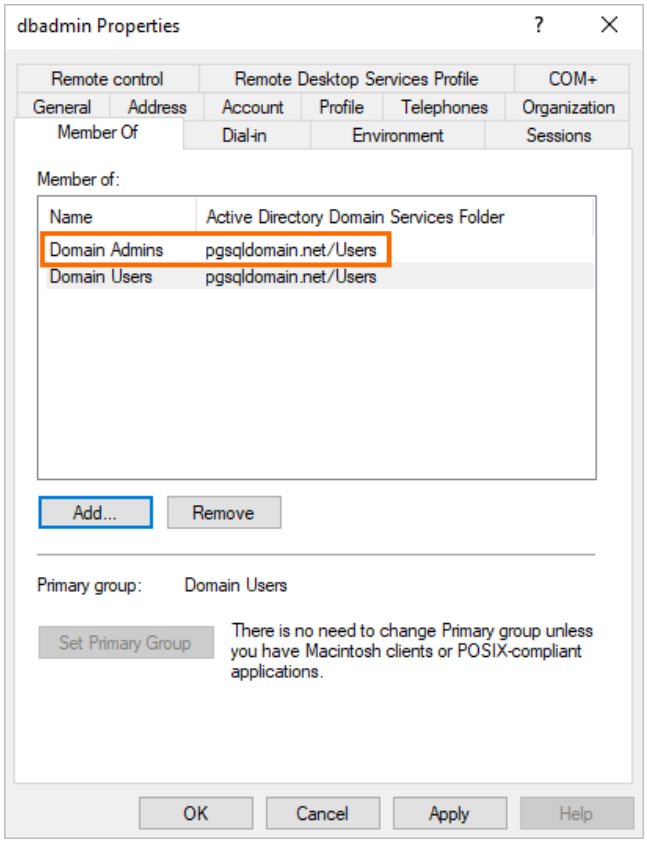
v. Specify a password, select **Password never expires**, and then click **Next**. Then, click **Finish**. A user is created.



vi. Double-click the created user and add the user to the **Domain Admins** administrator group.



After the user is added to the Domain Admins administrator group, the following page appears.



3. Add a standard user to the AD domain controller for logon.

> **Note** You must perform the same operations that are described in Add an administrator user to the AD domain controller. A standard user does not need to be added to the **Domain Admins** administrator group.
>
> In this example, add a standard user named `ldapuser` to the AD domain controller. This user is used to log on to your RDS instance.

4. Configure security group rules for the ECS instance.

i. Log on to the ECS console.

ii. In the left-side navigation pane, choose **Instances & Images > Instances**.

iii. In the top navigation bar, select a region.

iv. On the **Instances** page, find the required ECS instance and click the ID of the ECS instance.

v. In the left-side navigation pane, click **Security Groups**. On the page that appears, click **Add Rules**.

> **Note** A number of ports need to be enabled for the AD domain controller. We recommend that you configure a separate security group for the AD domain controller rather than configuring the AD domain controller in the same security group as other ECS instances.

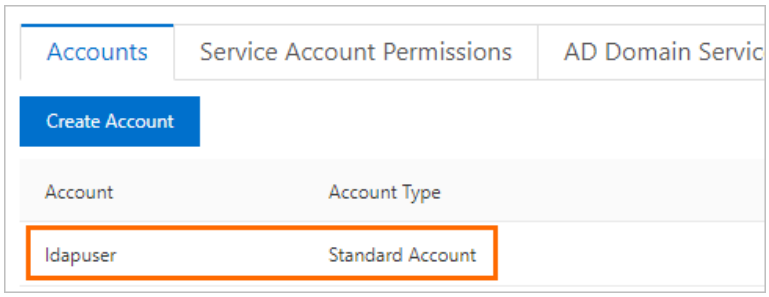
vi. On the **Inbound** tab, click **Add Rule** to allow your RDS instance to access the ECS instance over the following ports.

Protocol type	Port range	Description
TCP	88	The port for the Kerberos authentication protocol.
TCP	135	The port for the Remote Procedure Call (RPC) protocol.
TCP/UDP	389	The port for the Lightweight Directory Access Protocol (LDAP).
TCP	445	The port for the Common Internet File System (CIFS) protocol.
TCP	3268	The port for Global Catalog.
TCP/UDP	53	The port for the DNS service.
TCP	49152~65535	The default dynamic port range for connections. Enter a value in the following format: 49152/65535.

5. Configure your RDS instance.

i.

ii. Create an account named `ldapuser`. For more information, see Create an account on an ApsaraDB RDS for PostgreSQL instance.



> **Note** The username of the account of your RDS instance must be the same as the name of the standard user that is created for the AD domain controller. The passwords of the two accounts can be different. When the AD domain controller is enabled for access control, the AD domain controller verifies the password of the standard user. When the AD domain controller is disabled for access control, ApsaraDB RDS verifies the password of the account of your RDS instance. You can set the password of the account on the **Accounts** page in the ApsaraDB RDS console.

iii. In the left-side navigation pane, click **Accounts**. On the page that appears, click the **AD Domain Services** tab.

If the **AD Domain Service** tab is opened for the first time, ApsaraDB RDS creates the following records by default:

```
host    all             all     0.0.0.0/0    md5
host    replication     all     0.0.0.0/0    md5
```

You can delete or modify those two records.

iv. Click **Edit** of the first record and modify the following parameters.

> **Note** The following table describes only the parameters that are used in the provided example. For more information, see Official documentation of PostgreSQL.

Parameter	Example	Description
priority	0	The priority of the record. If you set this parameter to 0, the record has the highest priority and is automatically generated. Modify the first record and set this parameter to 0. The value 0 specifies the highest priority in the AD domain service.

Parameter	Example	Description
TYPE	host	Valid values: ■ host: The record matches TCP/IP connections, including SSL connections and non-SSL connections. ■ hostssl: The record matches only TCP/IP connections that are established over SSL. Note This parameter takes effect only when SSL encryption is enabled for your RDS instance. For more information, see Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance. ■ hostnossl: The record matches only TCP/IP connections that are not established over SSL.
DATABASE	all	The database that the specified users are allowed to access. If you set this parameter to all, the specified users are allowed to access all databases of your RDS instance. If you specify multiple databases, separate the database names with commas (,).
USER	ldapuser	The user that is allowed to access the databases on your RDS instance. You must specify the usernames in the AD domain for this parameter. If you specify multiple users, separate the usernames with commas (,). Note This parameter can be set only to the usernames of standard users that are created in the AD domain.
ADDRESS	0.0.0.0/0	The IP addresses from which the specified users can access the specified databases. If you set this parameter to 0.0.0.0/0, the specified users are allowed to access the specified databases from all IP addresses.
MASK	None	The mask for the IP address in the record. If the value of the ADDRESS parameter is an IP address, you can use this parameter to specify the mask of the IP address.

Parameter	Example	Description
METHOD	ldap Note LDAP is a protocol that is used to access the directories of databases. In this topic, LDAP is used as an example.	The authentication method. Valid values: ■ trust ■ reject ■ scram-sha-256 ■ md5 ■ password ■ gss ■ sspi ■ ldap ■ radius ■ cert ■ pam Note The value of this parameter must be in lowercase.
OPTION	ldapserver=<*The private IP address of the ECS instance*> ldapbasedn="CN=Users,DC=pgsqldomain,DC=net" ldapbinddn="CN=<*The username of the administrator user of the AD domain controller*>,CN=Users,DC=pgsqldomain,DC=net" ldapbindpasswd="<*The password of the administrator user of the AD domain controller*>" ldapsearchattribute="sAMAccountName"	Optional. The value of this parameter varies based on the value of the Method parameter. In this topic, LDAP is used as an example. You must specify this parameter. For more information, see Authentication Methods.

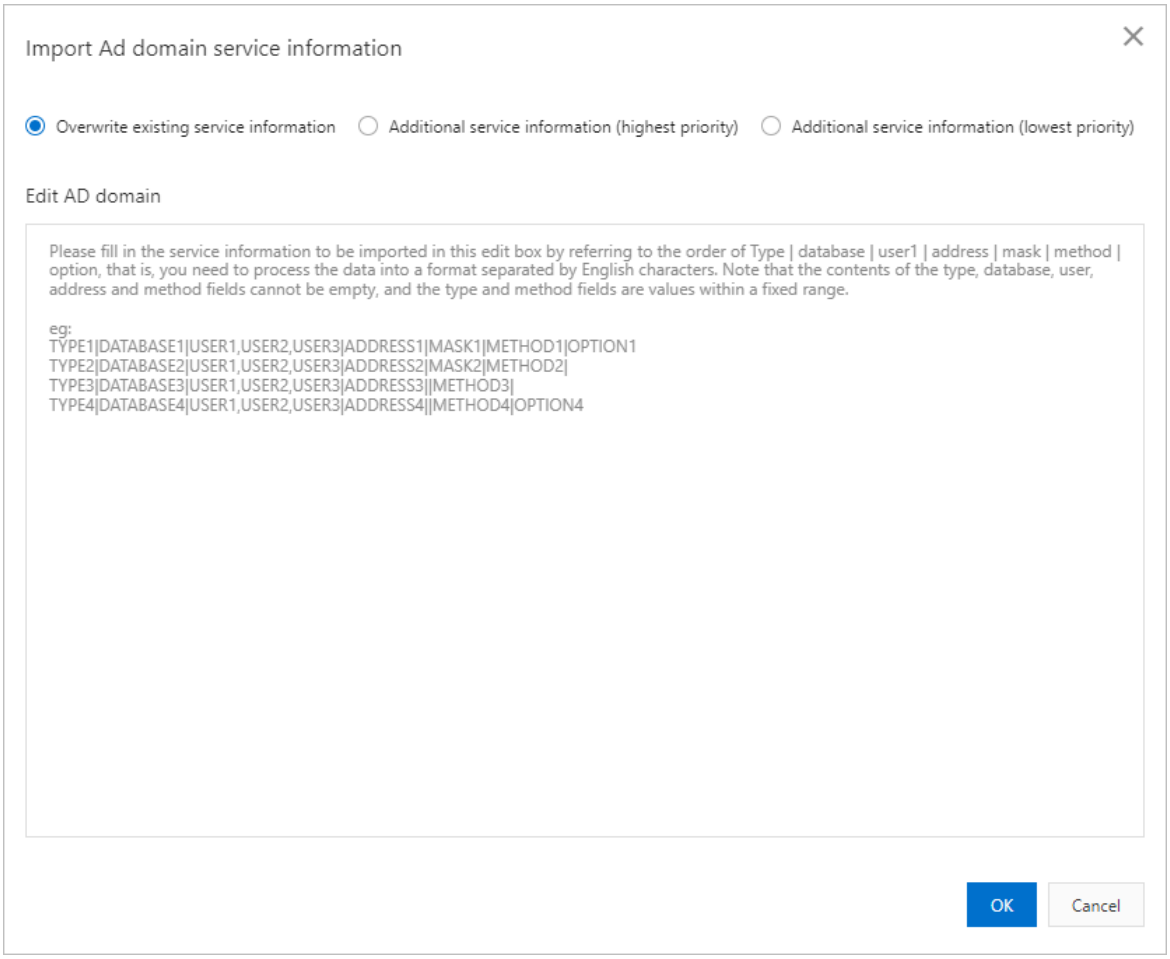
v. Click **add** of the AD domain service record to add a new record. The following information provides the valid values of the new record.

```
host    all            all    0.0.0.0/0    md5
```

vi. Click **OK**. Then, click **Submit**.

Note After you click Submit, the status of your RDS instance changes to **Maintaining Instance** for about 1 minute. The new configurations take effect only for new connections. You must close the existing connections and re-establish these connections for the new configurations to take effect.

6. (Optional) Import the service information of multiple AD domains at a time. You can also manually add the service information of the AD domain.



The following import methods are supported:

- Overwrite existing service information.
- Additional service information (highest priority): If you select this option, the service information of the AD domain is appended to the beginning of the existing service information. The priority of the appended information is higher than the priority of the existing service information.
- Additional service information (lowest priority): If you select this option, the service information of the AD domain is appended to the end of the existing service information. The priority of the appended information is lower than the priority of the existing service information.

Valid format:

```
TYPE|DATABASE|USER1|ADDRESS|MASK|METHOD|OPTION
```

Enter the service information that you want to import in the **Edit AD domain** text box. For more information about the parameters, see Parameters.

Examples:

```
host|all|<The username of the standard user of the AD domain controller>|0.0.0.0/0||ldap|
ldapserver=<The private IP address of the ECS instance> ldapbasedn="CN=Users,DC=pgsqldoma
in,DC=net" ldapbinddn="CN=<The username of the administrator user of the AD domain contro
ller>,CN=Users,DC=pgsqldomain,DC=net" ldapbindpasswd="<The password of the administrator
user of the AD domain controller>" ldapsearchattribute="sAMAccountName"
```

7. Test the connection.

Use a PostgreSQL command-line tool to connect to your RDS instance.

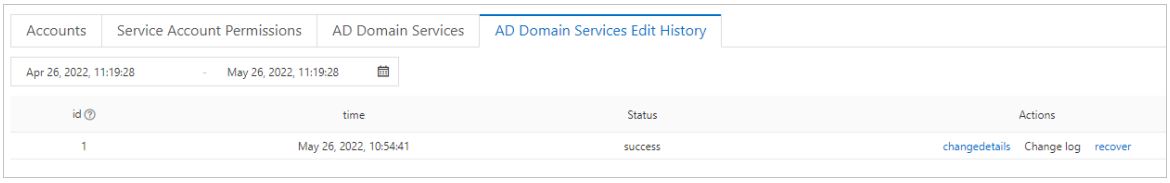
> **Note** You can connect to your RDS instance by using multiple methods. In this topic, a PostgreSQL command-line tool is used. You must install PostgreSQL before you use the PostgreSQL command-line tool. For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance.

Run the following command and use the username and password of the standard user of the AD domain controller to connect to your RDS instance:

```
psql -h <The endpoint of your RDS instance> -U ldapuser -p 5432 -d postgres
```

View the modification history of AD domain service information

1.
2. In the left-side navigation pane, click **Accounts**. On the page that appears, click the **AD Domain Services Edit History** tab.
3. You can view **changedetails** in the **Actions** column. If the modification fails, the status is **Not Taking Effect**. You can click **Change log** to view the error message.



18.7. System accounts of an ApsaraDB RDS for PostgreSQL instance

This topic describes the system accounts that are provided in an ApsaraDB RDS for PostgreSQL instance. In most cases, you do not need to consider the permissions and authorized operations of these system accounts.

Account	Description
• alicloud_rds_admin • pg*********	The O&M account that is used to locally manage the RDS instance. For example, Alibaba Cloud engineers can use this account to initialize the RDS instance by running the **initdb** command, restart the RDS instance, or query the status of the RDS instance.
aurora	The account that is used to remotely manage the RDS instance. The Alibaba Cloud management and control system can use this account to log on to and manage the RDS instance. For example, the system can create databases and accounts on the RDS instance, check the status of the RDS instance, or monitor the performance of the RDS instance.
replicator	The account that is used to replicate data from the primary RDS instance to its secondary RDS instance. This account is available only in RDS High-availability Edition.

Note The IP addresses of the preceding system accounts are internal IP addresses. You can execute the `SELECT usename, client_addr FROM pg_stat_activity;` statement to view the current logon account and its IP address. Example:

```
SELECT usename, client_addr FROM pg_stat_activity;
      usename         client_addr
--------------------+----------------
                    |
 alicloud_rds_admin
 replicator          172.21.XX.XX
 aurora              100.104.XX.XX
 aurora
 aurora              100.104.XX.XX
 aurora              100.104.XX.XX
```

19.Database connections

19.1. Connect to an ApsaraDB RDS for PostgreSQL instance

This topic describes how to connect to an ApsaraDB RDS for PostgreSQL instance. You can connect to an RDS instance by using Data Management (DMS), a command-line tool, pgAdmin, or an application.

Prerequisites

The operations that are described in the following topics are complete:

- Create an ApsaraDB RDS for PostgreSQL instance
- Create an account on an ApsaraDB RDS for PostgreSQL instance
- Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance
- If you connect to your RDS instance over an internal network from an Elastic Compute Service (ECS) instance, the following requirements are met:
  - The ECS instance and the RDS instance belong to the same Alibaba Cloud account.
  - The ECS instance and the RDS instance reside in the same region.
  - The ECS instance and the RDS instance reside in the same virtual private cloud (VPC).
  - The private IP address of the ECS instance is added to an IP address whitelist of the RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

Use DMS to connect to an RDS instance

Log on to the ApsaraDB RDS console, find the RDS instance to which you want to connect, and go to the **Basic Information** page. In the upper-right corner of the page, click **Log On to Database**. Then, enter the required information to log on to the RDS instance.

For more information, see Use DMS to log on to an ApsaraDB RDS for PostgreSQL instance.

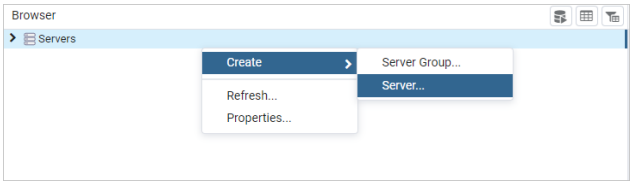
Use pgAdmin to connect to an RDS instance

When you download the PostgreSQL software package from the PostgreSQL official website and install PostgreSQL, **pgAdmin 4** is automatically downloaded and installed. You can also download the pgAdmin software package from the PostgreSQL official website.

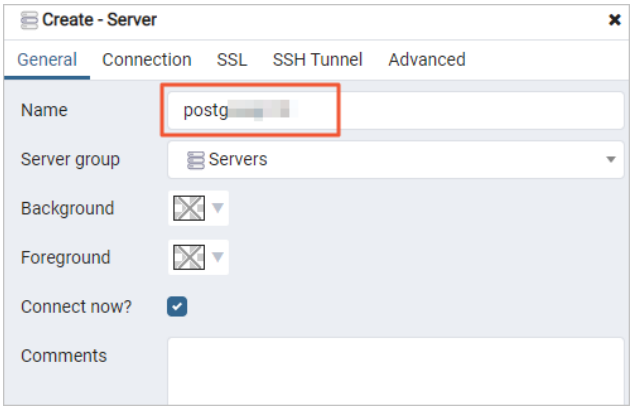
1. Start **pgAdmin 4**.

   > **Note** If you use pgAdmin that is in a later version than version 4 and you use pgAdmin for the first time, you must specify a master password to protect your saved logon credentials such as passwords.

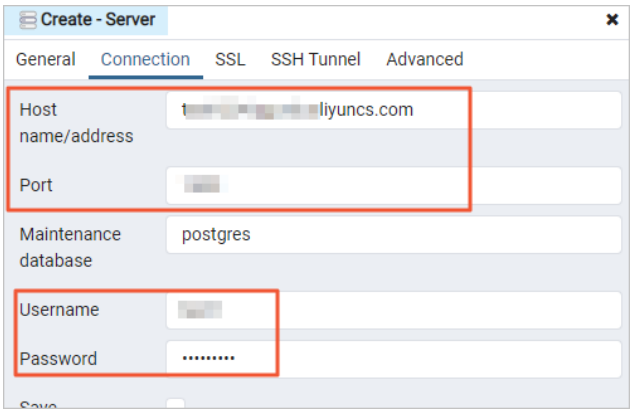
2. Right-click **Servers** and choose **Create > Server...**.



3. On the **General** tab of the Create - Server dialog box, enter the name of the server on which pgAdmin is installed.



4. Click the **Connection** tab and enter the information that is used to connect to the RDS instance.

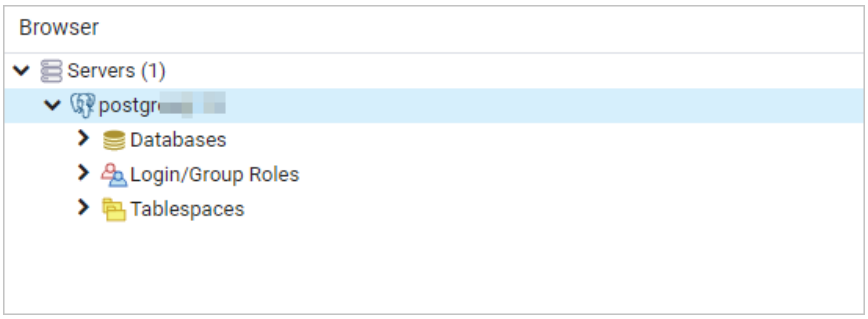


Parameter	Description
Hostname/address	Enter the endpoint of the RDS instance. If you want to connect to the RDS instance over an internal network, enter the internal endpoint of the RDS instance. If you want to connect to the RDS instance over the Internet, enter the public endpoint of the RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Port	Enter the port number that is associated with the specified endpoint.
Username	Enter the username of the account that is used to log on to the RDS instance. For more information about how to create an account for an RDS instance, see Create a database and an account on an ApsaraDB RDS for PostgreSQL instance.
Password	Enter the password of the account that is used to log on to the RDS instance.

5. Click **Save**.

If the information that you enter is correct, the page that is shown in the following figure appears, which indicates that the connection to the RDS instance is successful.

> **Notice** The postgres database is a default system database. Do not perform operations on this database.

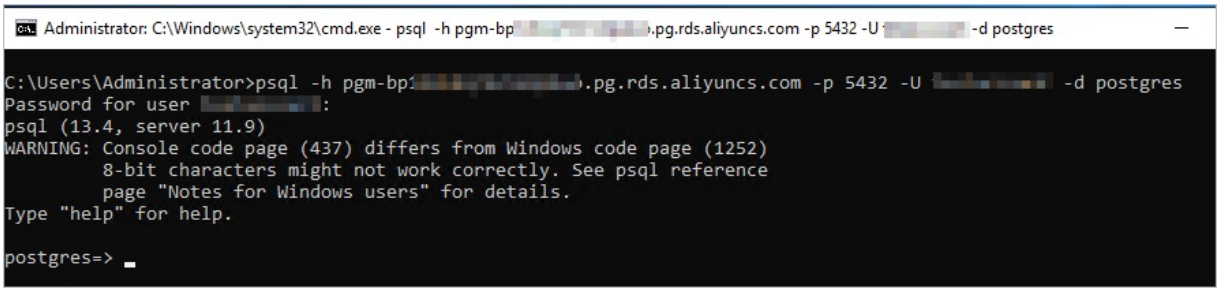


Use a command-line tool to connect to an RDS instance

When you download the PostgreSQL software package from the PostgreSQL official website and install PostgreSQL, **a PostgreSQL command-line tool** is automatically downloaded and installed.

Run the following command in the command-line tool to connect to a database of the RDS instance:

```
psql -h <Endpoint> -U <Username> -p <Port number> -d <Database name>
```



The following table provides details about how to obtain the values of the parameters from the ApsaraDB RDS console.

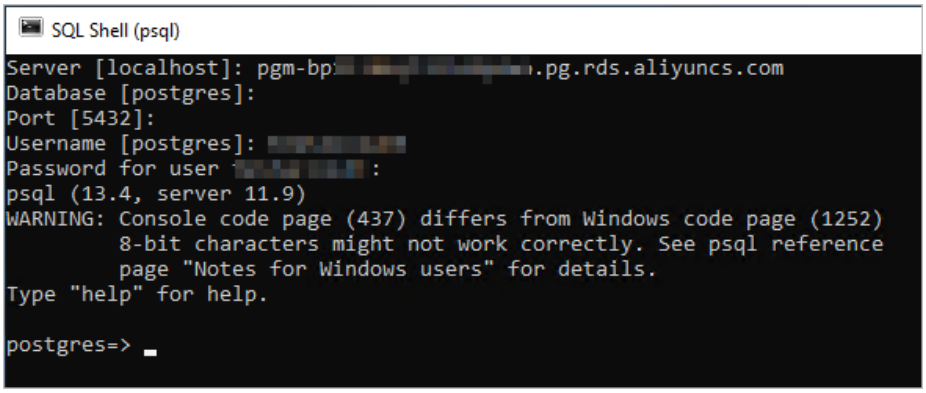
Parameter	How to obtain
Endpoint.	The endpoint that is used to connect to the RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Username	The username of the account that is used to log on to the RDS instance. You can obtain the username from the **Accounts** page. For more information about how to create an account, see Create an account on an ApsaraDB RDS for PostgreSQL instance.
Port number	The port number that is used to connect to the RDS instance. The default port number is 5432. If you have modified the port number, you can obtain the new port number from the **Database Connection** page. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.

Parameter	How to obtain
Database name	The name of the database that you want to connect in the RDS instance. The postgres database is a default system database. Do not perform operations on this database. You can obtain the name of the database that you want to connect from the **Databases Connection** page. For more information about how to create a database, see Create a database on an ApsaraDB RDS for PostgreSQL instance.

Use SQL Shell (psql) to connect to an RDS instance

When you download the PostgreSQL software package from the PostgreSQL official website and install PostgreSQL, **SQL Shell (psql)** is automatically downloaded and installed.

Open the **Start** menu on your computer and click the **SQL Shell (psql)**. Then, enter the required parameters to connect to the RDS instance.



The following table provides details about how to obtain the values of the parameters from the ApsaraDB RDS console.

Parameter	How to obtain
Server	The endpoint that is used to connect to the RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Database	The name of the database that you want to connect in the RDS instance. If you do not specify this parameter, the default value is postgres. The postgres database is a default system database. Do not perform operations on this database. You can obtain the name of the database that you want to connect from the **Databases Connection** page. For more information about how to create a database, see Create a database on an ApsaraDB RDS for PostgreSQL instance.
Port	The port number that is used to connect to the RDS instance. The default port number is 5432. If you have modified the port number, you can obtain the new port number from the **Database Connection** page. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Username	The username of the account that is used to log on to the RDS instance. You can obtain the username from the **Accounts** page. For more information about how to create an account, see Create an account on an ApsaraDB RDS for PostgreSQL instance.

Use an application to connect to an RDS instance

Note In this topic, a Maven project is connected to the RDS instance by using the Java Database Connectivity (JDBC). This connection method is similar to other programming languages.

1. Add dependencies to the pom.xml file:

```
<dependency>
  <groupId>postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <version>8.2-504.jdbc3</version>
</dependency>
```

2. The following code snippet provides an example on how to use the JDBC to connect to the RDS instance:

```
public class DatabaseConnection
{
    public static void main( String[] args ){
        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        //Endpoint
        String hostname = "pgm-bp1i3kkq7321o9****.pg.rds.aliyuncs.com";
        //Port number
        int port = 5432;
        //Database name
        String dbname = "postgres";
        //Username
        String username = "username";
        //Password
        String password = "password";
        String dbUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbname + "?bi
naryTransfer=true";
        Connection dbConnection;
        try {
            dbConnection = DriverManager.getConnection(dbUrl, username, password);
            Statement statement = dbConnection.createStatement();
            //SQL statement that you want to execute
            String selectSql = "SELECT * FROM information_schema.sql_features LIMIT 10";
            ResultSet resultSet = statement.executeQuery(selectSql);
            while (resultSet.next()) {
                System.out.println(resultSet.getString("feature_name"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Configure SSL encryption for an RDS instance

You can configure SSL encryption for an RDS instance. SSL encryption is used to encrypt the connections to the RDS instance and protect the data that is transmitted over the connections. For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance over SSL.

FAQ

How do I use Function Compute to obtain data from my RDS instance?

You can install third-party dependencies on Function Compute. Then, you can use these built-in dependencies to obtain data from ApsaraDB RDS. For more information, see Install third-party dependencies.

19.2. Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance

ApsaraDB RDS for PostgreSQL supports two types of endpoints: internal endpoints and public endpoints. By default, you are provided with an internal endpoint that is used to connect to your RDS instance. If you want to connect to your RDS instance over the Internet, you must apply for a public endpoint.

Internal and public endpoints

Endpoint type	Description
Internal endpoint	• By default, an internal endpoint is provided. You do not need to apply for the internal endpoint. In addition, you cannot release the internal endpoint. However, you can change the network type of your RDS instance. • If an Elastic Compute Service (ECS) instance resides in the same region and has the same network type as your RDS instance, these instances can communicate over an internal network. If your application is deployed on such an ECS instance, you do not need to apply for a public endpoint. For more information, see Change the network type of an ApsaraDB RDS for PostgreSQL instance. • For security and performance purposes, we recommend that you connect to your RDS instance by using the internal endpoint.

Endpoint type	Description

Public endpoint	• You must manually apply for a public endpoint. You can release the public endpoint if it is no longer required.
	• If you cannot connect to your RDS instance by using the internal endpoint, you must apply for a public endpoint. This includes the following scenarios:
	  ○ Connect to your RDS instance from an ECS instance that resides in a different region or has a different network type than your RDS instance. For more information, see Change the network type of an ApsaraDB RDS for PostgreSQL instance.
	  ○ Connect to your RDS instance from a device outside Alibaba Cloud.
	Note
	• You are not charged for the public endpoint or the traffic that is consumed.
	• If you connect to your RDS instance by using the public endpoint, security is compromised. Proceed with caution.
	• We recommend that you migrate your application to an ECS instance that resides in the same region and has the same network type as your RDS instance. This allows you to connect to your RDS instance by using the internal endpoint. The connection expedites transmission and improves security.

Procedure

1.
2. In the left-side navigation pane, click **Database Connection**.
3. Apply for or release a public endpoint for your RDS instance:
   - If you have not applied for a public endpoint, you can click **Apply for Public Endpoint**.
   - If you have applied for a public endpoint, you can click **Release Public Endpoint**.
4. In the message that appears, click **OK**.

Related operations

Operation	Description
Apply for a public endpoint	Applies for a public endpoint for an ApsaraDB RDS instance.
Release a public endpoint	Releases the public endpoint of an ApsaraDB RDS instance.

19.3. Use DMS to log on to an ApsaraDB RDS for PostgreSQL instance

This topic describes how to log on to an ApsaraDB RDS for PostgreSQL instance by using Data Management (DMS).

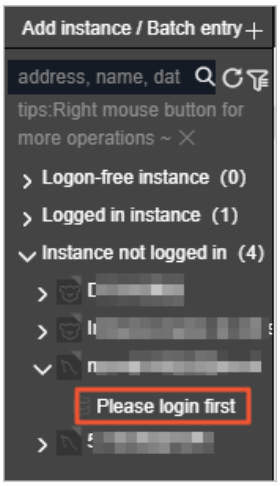
Context

DMS offers an integrated solution that supports data management, schema management, server management, user authorization, security audit, trend analysis, data tracking, business intelligence (BI) charts, and performance analysis and optimization.

Use the new DMS console to log on to an RDS instance

1. Log on to the new DMS console.
2. In the left-side navigation pane, select the RDS instance and click **Please login first**.

   > **Note** If the RDS instance uses the **Security Collaboration** control mode, you need only to click **Logon-free instance** and double-click the instance. You do not need to enter the username and password of the account that is used for the logon.

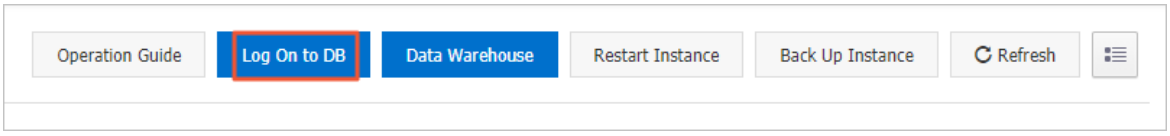
   

3. In the dialog box that appears, enter the username and password of the account that is used for the logon and click **OK**.
4. In the left-side navigation pane, click **Logged in instance** and double-click the RDS instance to switch to the instance.

Use the original DMS console to log on to an RDS instance

> **Note** If you have upgraded DMS to the new version, we recommend that you use the new DMS console to log on to an RDS instance.

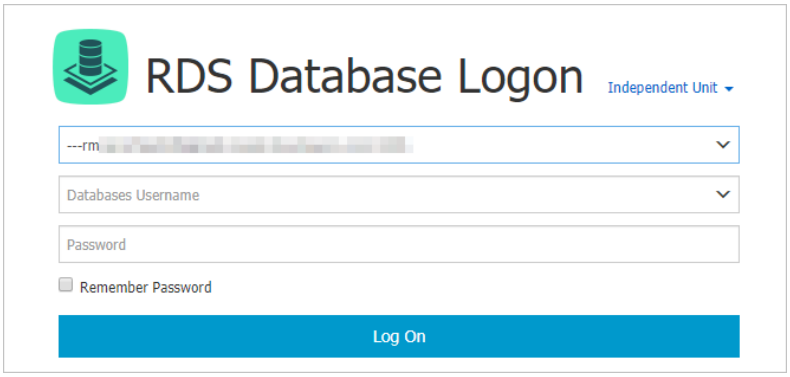
1.
2. In the upper-right corner of the page, click **Log On to Database** to open the RDS Database Logon page.



> **Note** Alibaba Cloud directs you to the original or new DMS console based on your logon history.

3. Configure the following parameters.

Parameter	Description
Endpoint:Port number	The endpoint and port number that are used to log on to the RDS instance. The endpoint and port number are in the `<Endpoint>:<Port number>` format. Example: rm-bpxxxxxxx.rds.aliyuncs.com:3433. For more information about how to view the endpoint and port number of an RDS instance, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Databases Username	The username of the account that is used to log on to the RDS instance.
Password	The password of the preceding account.



4. Click **Log On**.

> **Note** If you want the browser to save the password, select **Remember Password** before you click **Log On**.

5. If the system prompts you to add the CIDR block that contains the IP address of the DMS server to an IP address whitelist of your RDS instance, click **Specify for All Instances** or **Specify for Current Instance**.
6. After the CIDR block is added, click **Log On**.

19.4. View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance

To connect to an ApsaraDB RDS for PostgreSQL instance, you must enter the internal or public endpoint and port number of the RDS instance. This topic describes how to view and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance in the ApsaraDB RDS console.

Change the internal or public endpoint and port number

1.
2. In the left-side navigation pane, click **Database Connection**.
3. Click **Change Endpoint**.
4. In the dialog box that appears, select a connection type, enter the prefix of the endpoint that you want to change, specify the port number, and then click **OK**.

> **Note**
>
> - The prefix of an endpoint must be 8 to 64 characters in length and can contain only letters, digits, and hyphens (-). The prefix must start with a lowercase letter.
> - The port number must be within the range of 1000 to 5999.
> - If your RDS instance uses local SSDs, you cannot change its port numbers.

FAQ

- After I change an endpoint or a port number of my RDS instance, do I need to update the endpoint or port number information in my application?

  Yes, after you change an endpoint or a port number of your RDS instance, you must update the endpoint or port number information on your application. If you do not update the information, your application cannot connect to your RDS instance.

- After I change an endpoint or a port number of my RDS instance, does the change immediately take effect? Do I need to restart my RDS instance?

  After you change an endpoint or a port number of your RDS instance, the change immediately takes effect. You do not need to restart your RDS instance.

- After I change or release an endpoint of my RDS instance, can I use the endpoint for another RDS instance?

  Yes, after you change or release an endpoint of your RDS instance, you can use the endpoint of your RDS instance for another RDS instance.

- Does a primary/secondary switchover trigger changes to the endpoints of my RDS instance?

  No, a primary/secondary switchover does not trigger changes to the endpoints of your RDS instance. However, the IP addresses that are associated with the endpoints change. Your application can still connect to your RDS instance by using the endpoints.

20.Network, VPC, and VSwitch

20.1. Switch an ApsaraDB RDS for PostgreSQL instance to a different vSwitch

This topic describes how to switch an ApsaraDB RDS for PostgreSQL instance that uses standard SSDs or enhanced SSDs (ESSDs) to a different vSwitch.

Prerequisites

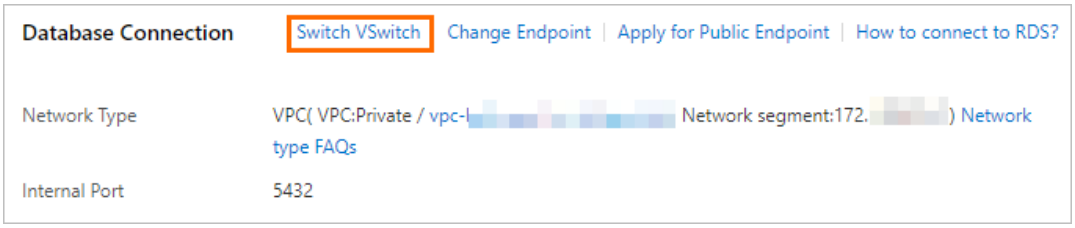
Your RDS instance runs PostgreSQL with standard SSDs or ESSDs.

Impacts

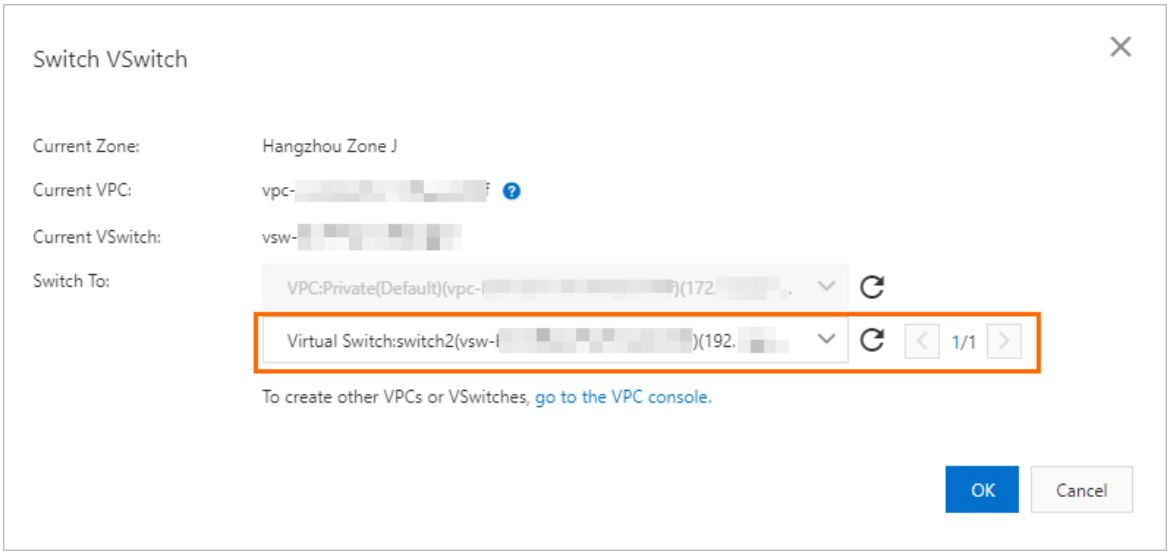
- When you switch your RDS instance to a different vSwitch, a transient connection that lasts approximately 30 seconds occurs. Make sure that your application is configured to automatically reconnect to your RDS instance.
- After the vSwitch of your RDS instance is changed, the virtual IP addresses (VIPs) of your RDS instance change. We recommend that you connect your application to your RDS instance by using an endpoint. For more information, see Configure endpoints for an ApsaraDB RDS for PostgreSQL instance.
- The VIP changes temporarily interrupt the connections to Data Management (DMS) and Data Transmission Service (DTS). After the vSwitch of your RDS instance is changed, these connections are automatically restored to normal.
- If data is cached on your database client, you can read data but cannot write data. After the vSwitch of your RDS instance is changed, we recommend that you immediately clear the cache on the database client.

Procedure

1.
2. In the left-side navigation pane, click **Database Connection**.
3. On the Database Connection page, click **Switch VSwitch**.



4. In the dialog box that appears, select a destination vSwitch and click **OK**.



5. In the message that appears, click **Switch**.

FAQ

How do I change the VPC of an RDS instance?

Purchase a new RDS instance that resides in the required VPC. Then, migrate the data of the original RDS instance to the new RDS instance. For more information, see Use the cloud migration feature to migrate data between ApsaraDB RDS for PostgreSQL instances or Use DTS to migrate data between ApsaraDB RDS for PostgreSQL instances.

20.2. Change the network type of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to change the network type of an ApsaraDB RDS for PostgreSQL instance based on your business requirements.

Network types

- Classic network: RDS instances in the classic network are not isolated. To block unauthorized access to these instances, you must configure IP address whitelists or security groups.
- Virtual private cloud (VPC): Each VPC is an isolated virtual network. VPCs are more secure than the classic network. We recommend that you select the VPC network type.

  You can configure route tables, CIDR blocks, and gateways in a VPC. In addition, you can connect your data center to a VPC by using Express Connect circuits or VPNs. The data center and the VPC comprise a virtual data center. You can use the virtual data center to migrate your workloads to the cloud with no downtime.

> **Note**
>
> - You can select the classic or VPC network type and switch your RDS instance between these network types free of charge.
> - Before you change the network type, you must enable the enhanced whitelist mode for your RDS instance. For more information, see Switch an ApsaraDB RDS for PostgreSQL instance to the enhanced whitelist mode.

View the network type

1.
2. In the left-side navigation pane, click **Database Connection**. On the page that appears, view the network type of the RDS instance.

Change the network type from classic network to VPC

> **Note** Your RDS instance resides in the classic network.

1.
2. In the left-side navigation pane, click **Database Connection**.
3. Click **Switch to other VPC**.
4. In the Switch to VPC dialog box, select a VPC and a vSwitch and specify whether to retain the classic network endpoint.
   - Select a VPC. We recommend that you select the VPC where the Elastic Compute Service (ECS) instance that you want to connect resides. If the ECS instance and the RDS instance reside in different VPCs, these instances cannot communicate over an internal network unless you use Cloud Enterprise Network (CEN) or VPN Gateway to enable network communication between the VPCs of these instances. For more information, see Overview of Alibaba Cloud CEN or Establish IPsec-VPN connections between two VPCs.
   - Select a vSwitch. If no vSwitches are available in the selected VPC, create a vSwitch in the zone in which the RDS instance resides. For more information, see Create a vSwitch.
   - Clear or select **Reserve original classic endpoint**.

Operation	Description
Clear Reserve original classic endpoint	The classic network endpoint is not retained and changes to a VPC endpoint. When you change the network type from classic network to VPC, a transient connection that lasts approximately 30 seconds occurs and ECS instances that reside in the classic network are immediately disconnected from your RDS instance.

Operation	Description
Select Reserve original classic endpoint	The classic network endpoint is retained, and a new VPC endpoint is generated. In this case, the RDS instance runs in hybrid access mode. Classic network-hosted ECS instances and VPC-hosted ECS instances can connect to the RDS instance over an internal network. For more information, see Configure the hybrid access solution for an ApsaraDB RDS for PostgreSQL instance. When you change the network type from classic network to VPC, no transient connection occurs. The connection between each classic network-hosted ECS instance and the RDS instance remains available until the classic network endpoint expires. Before the classic network endpoint expires, add the VPC endpoint to your application that runs on a VPC-hosted ECS instance. This allows ApsaraDB RDS to migrate your workloads to the selected VPC with no downtime. ApsaraDB RDS sends a text message to the mobile number that is bound to your Alibaba Cloud account every day within seven days before the classic network endpoint expires. For more information, see Configure the hybrid access solution for an ApsaraDB RDS for PostgreSQL instance.

5. Add the private IP address of the required VPC-hosted ECS instance to an IP address whitelist of the VPC network type on the RDS instance. This way, the ECS instance can access the RDS instance over an internal network. If no IP address whitelists of the VPC network type are available, create one.
6. Add the VPC endpoint of the RDS instance to the required VPC-hosted ECS instance.
   - If you selected Reserve original classic endpoint, you must add the VPC endpoint to your application that runs on the required VPC-hosted ECS instance before the classic network endpoint expires.
   - If you cleared Reserve original classic endpoint, the connection between each classic network-hosted ECS instance and the RDS instance over an internal network is immediately closed after the network type is changed. You must add the VPC endpoint of the RDS instance to your application that runs on the required VPC-hosted ECS instance.

> **Note**
>
> If the RDS instance resides in a VPC and you want to connect a classic network-hosted ECS instance to the RDS instance over an internal network, you can use ClassicLink to establish a connection. Alternatively, you can migrate the ECS instance to the same VPC as the RDS instance. For more information, see Overview.

Change the VPC of an instance

You cannot directly change the VPC of an RDS instance. You can change the vSwitch of the RDS instance.

- If you want to change the VPC, you must purchase a new RDS instance that resides in the required VPC and then migrate the data of the original RDS instance to the new RDS instance.
- If you want to change the vSwitch, see Switch an ApsaraDB RDS for PostgreSQL instance to a different vSwitch.

Related operations

Operation	Description
ModifyDBInstanceNetworkType	Changes the network type of an ApsaraDB RDS instance.

20.3. Configure the hybrid access solution for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure the hybrid access solution for an ApsaraDB RDS for PostgreSQL instance. This solution allows you to retain both the classic network endpoint and virtual private cloud (VPC) endpoint of your RDS instance. This way, you can migrate your RDS instance from the classic network to a VPC without network interruptions.

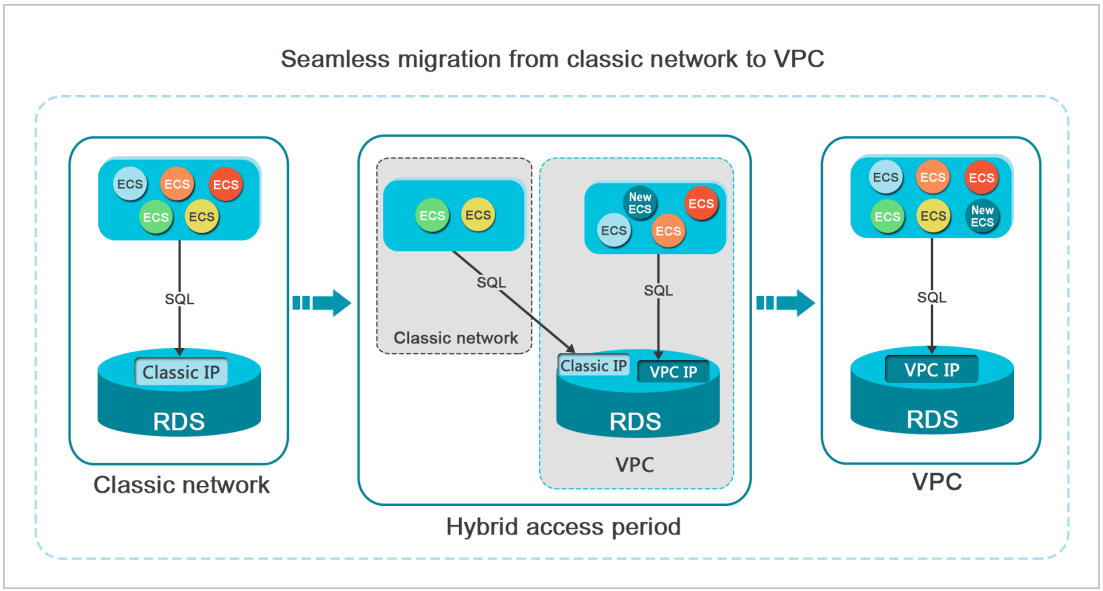
Background information

When you migrate your RDS instance from the classic network to a VPC, the internal classic network endpoint of the instance changes to the internal VPC endpoint. In this case, the endpoint remains unchanged, but the IP address that is bound to the endpoint changes. This change causes a transient connection error of about 30 seconds or less, and classic network-hosted Elastic Compute Service (ECS) instances can no longer connect to your RDS instance over an internal network. To facilitate a smooth migration, ApsaraDB RDS provides the hybrid access solution.

The hybrid access solution allows you to connect your RDS instance from both classic network-hosted ECS instances and VPC-hosted ECS instances. During the validity period of the hybrid access solution, ApsaraDB RDS retains the internal classic network endpoint and generates an internal VPC endpoint. This prevents transient connection errors when you migrate your RDS instance from the classic network to a VPC.

For security and performance purposes, we recommend that you use only the internal VPC endpoint. You must specify a validity period for the hybrid access solution. After the validity period elapses, ApsaraDB RDS releases the internal classic network endpoint. Then, your applications can no longer connect to your RDS instance by using this endpoint. Therefore, you must add the internal VPC endpoint to your applications before the validity period elapses. This allows you to ensure a smooth migration and prevent interruptions to your workloads.

For example, a company uses the hybrid access solution to migrate their RDS instance from the classic network to a VPC. During the validity period of the hybrid access solution, some applications connect to the RDS instance by using the internal VPC endpoint, whereas the others still connect to the RDS instance by using the internal classic network endpoint. When all applications of the company can connect to the RDS instance by using the internal VPC endpoint, the internal classic network endpoint can be released.



Limits

During the validity period of the hybrid access solution, your RDS instance does not support the following operations:

- Change to the classic network type
- Migration to another zone

Prerequisites

- Your RDS instance resides in the classic network.
- The zone where your RDS instance resides provides available VPCs and vSwitches. For more information about how to create VPCs and vSwitches, see Create a VPC.

Migrate your RDS instance from the classic network to a VPC

1.
2. In the left-side navigation pane, click **Database Connection**.
3. Click **Switch to other VPC**.
4. In the dialog box that appears, select a VPC and a vSwitch and specify whether to retain the classic network endpoint.
   - Select a VPC. We recommend that you select the VPC where the required ECS instance resides. If the ECS and RDS instances reside in different VPCs, these instances cannot communicate over an internal network. In this case, if you want these instances to communicate over an internal network, you must create a Cloud Enterprise Network (CEN) instance or an IPsec-VPN connection between the VPCs of these instances. For more information, see Overview and Establish IPsec-VPN connections between two VPCs.
   - Select a vSwitch. If no vSwitches are available in the selected VPC, create a vSwitch in the same zone as your RDS instance. For more information, see Create a vSwitch.
   - Clear or select the **Reserve original classic endpoint** option. For more information, see the following table.

Action	Description
Clear the Reserve original classic endpoint option	The classic network endpoint is not retained and changes to the VPC endpoint. When you change the network type from classic network to VPC, a transient connection error of about 30 seconds occurs. In this case, the connection between each classic network-hosted ECS instance and your RDS instance is closed.
Select the Reserve original classic endpoint option	The classic network endpoint is retained, and a new VPC endpoint is generated. In this case, your RDS instance runs in hybrid access mode. Both classic network-hosted ECS instances and VPC-hosted ECS instances can connect to your RDS instance over an internal network. When you change the network type from classic network to VPC, no transient connection errors occur. The connection between each classic network-hosted ECS instance and your RDS instance remains available until the classic network endpoint expires. Before the classic network endpoint expires, you must add the VPC endpoint to each required VPC-hosted ECS instance. This allows ApsaraDB RDS to migrate your workloads to the selected VPC without network interruptions.

5. Add the private IP address of each required VPC-hosted ECS instance to an IP address whitelist of the VPC network type. This allows the ECS instance to connect to your RDS instance over an internal network. If no IP address whitelists of the VPC network type are available, create one.
6. ○ If you have selected the Reserve original classic endpoint option, add the VPC endpoint of your RDS instance to each required VPC-hosted ECS instance before the classic network endpoint expires.
   ○ If you have cleared the Reserve original classic endpoint option, the connection between each classic network-hosted ECS instance and your RDS instance over an internal network is immediately closed after the network type change is complete. You must add the VPC endpoint of your RDS instance to each required VPC-hosted ECS instance.

> **Note** If you want to connect a classic network-hosted ECS instance to your VPC-hosted RDS instance over an internal network, you can use ClassicLink to establish a connection. Otherwise, you can migrate the ECS instance to the same VPC as your RDS instance. For more information, see Overview.

Change the expiration date of the internal classic network endpoint

During the validity period of the hybrid access solution, you can change the expiration date of the classic network endpoint based on your business requirements. The expiration date is immediately recalculated starting from the day when you make the change. For example, the classic network endpoint is configured to expire on August 18, 2017. On August 15, 2017, you increase the validity period of the classic network endpoint by 14 days. In this case, ApsaraDB RDS releases the classic network endpoint on August 29, 2017.

Perform the following steps:

1.
2. In the left-side navigation pane, click **Database Connection**.
3. In the **Original Classic Network Endpoint** section of the Database Connection page, click **Change Expiration Time**.
4. In the **Change Expiration Time** dialog box, select an expiration date and click **OK**.

21.Database

21.1. Create a database on an ApsaraDB RDS for PostgreSQL instance

Before you start to use your ApsaraDB RDS for PostgreSQL instance, you must create databases on the RDS instance. This topic describes how to create a database on an ApsaraDB RDS for PostgreSQL instance.

Terms

- Instance: a virtualized database server, on which you can create and manage a number of databases.
- Database: a set of organized data that can be shared by a number of users. A database provides the minimal redundancy and is independent of applications. You can consider a database to be a warehouse that is used to store data.
- Character set: a collection of letters, special characters, and encoding rules that are used in a database.

Precautions

- If the RDS instance uses standard or enhanced SSDs, you can create and manage databases in the ApsaraDB RDS console.
- If the RDS instance uses local SSDs, you can create and manage databases by using SQL statements.
- If you want to migrate data from an on-premises database to the RDS instance, you must create a database and an account on the RDS instance. The created database must have the same name as the on-premises database. The created account must have the same name as the account that is authorized to manage the on-premises database.

Create a database for an RDS instance with standard or enhanced SSDs

1.
2. In the left-side navigation pane, click **Accounts**
3. Click **Create Account**.
4. Configure the following parameters.

Parameter	Description
Database Name	○ The name of the database can contain up to 63 characters. ○ The name of the database can contain lowercase letters, digits, hyphens (-), and underscores (_). ○ The name of the database must start with a lowercase letter and end with a lowercase letter or digit.
Supported Character Set	The character set that is supported by the database.
Collate	The rule that is used to sort strings.
Ctype	The type of character that is supported by the database.

Parameter	Description
Authorized Account:	The owner of the database. The owner has all permissions on the database.
Description	The description of the database.

5. Click **Create**.

Create a database for an RDS instance with local SSDs

1. Log on to the ApsaraDB RDS console.
2. In the SQL window, execute the following statement to create a database:

```
CREATE DATABASE name
 [ [ WITH ] [ OWNER [=] user_name ]
        [ TEMPLATE [=] template ]
        [ ENCODING [=] encoding ]
        [ LC_COLLATE [=] lc_collate ]
        [ LC_CTYPE [=] lc_ctype ]
        [ TABLESPACE [=] tablespace_name ]
        [ CONNECTION LIMIT [=] connlimit ] ]
```

For example, if you want to create a database named test, execute the following statement:

```
create database test;
```

What to do next

Connect to the RDS instance. For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance.

21.2. Delete a database from an ApsaraDB RDS for PostgreSQL instance

This topic describes how to delete a database from an ApsaraDB RDS for PostgreSQL instance.

Procedure

> **Note** After you delete a database from an RDS instance, ApsaraDB RDS reclaims the storage that is occupied by the database. You can check the disk usage of an RDS instance on the **Monitoring and Alerts** page of the RDS instance in the ApsaraDB RDS console. For more information, see View the resource and engine metrics of an ApsaraDB RDS for PostgreSQL instance.

1.
2. In the left-side navigation pane, click **Databases**.
3. In the **Actions** column click **Delete**.
4. In the message that appears, click **OK**.

21.3. Change the time zone of an ApsaraDB RDS for PostgreSQL instance

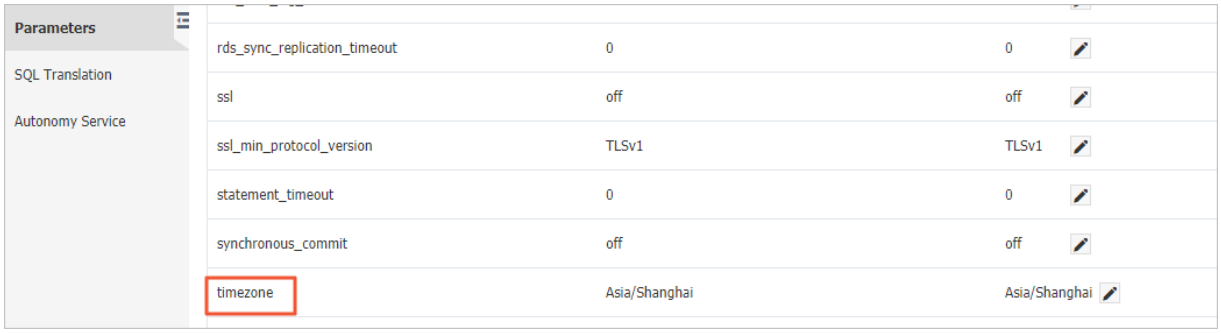
This topic describes how to change the time zone of an ApsaraDB RDS for PostgreSQL instance.

Prerequisites

The RDS instance uses standard or enhanced SSDs.

Change the time zone

You can change the time zone only of an RDS instance that uses standard or enhanced SSDs. To change the time zone, log on to the ApsaraDB for RDS console and modify the **timezone** parameter on the **Parameters** page. For more information, see Manage the parameters of an ApsaraDB RDS for PostgreSQL instance.



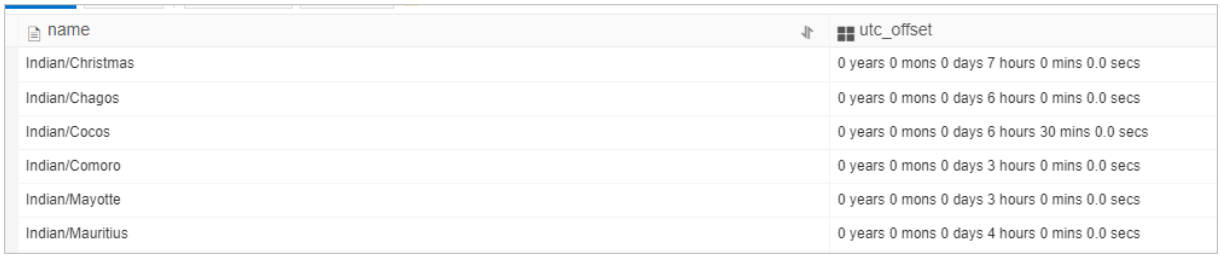
> **Note** If an RDS instance uses local SSDs, the **timezone** parameter is not supported.

Query supported time zones

You can run the following command to query supported time zones:

```
select name,utc_offset from pg_timezone_names;
```

> **Note** For more information about the pg_timezone_names table, see pg_timezone_names.



name	utc_offset
Indian/Christmas	0 years 0 mons 0 days 7 hours 0 mins 0.0 secs
Indian/Chagos	0 years 0 mons 0 days 6 hours 0 mins 0.0 secs
Indian/Cocos	0 years 0 mons 0 days 6 hours 30 mins 0.0 secs
Indian/Comoro	0 years 0 mons 0 days 3 hours 0 mins 0.0 secs
Indian/Mayotte	0 years 0 mons 0 days 3 hours 0 mins 0.0 secs
Indian/Mauritius	0 years 0 mons 0 days 4 hours 0 mins 0.0 secs

22.SQL translation

This topic describes how to use the SQL translation function. You can use this function to translate the SQL statements that are used by other database engines to the SQL statements that can be used by PostgreSQL.

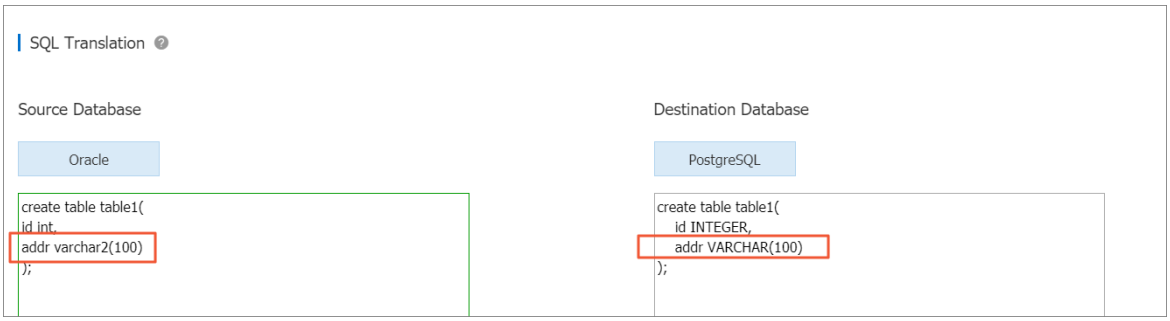
Context

Currently, you can only use the SQL translation function to translate SQL statements from Oracle to PostgreSQL.

Procedure

1.
2. In the left-side navigation pane, click **SQL Translation**.
3. In the **Source Database** section, enter the SQL statements that you want to translate.

> **Note** In the **Destination Database** section, the SQL statements after translation are automatically displayed.



23.Monitoring and alerts

23.1. View the resource and engine metrics of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to view the resource and engine metrics of an ApsaraDB RDS for PostgreSQL instance in the ApsaraDB RDS console.

Procedure

1.
2. In the left-side navigation pane of the page that appears, click **Monitoring and Alerts**.
3. On the **Standard monitoring** tab, specify a time range. Then, you can view the metrics that appear. The following table lists the metrics that are monitored when different PostgreSQL versions and RDS editions are used.

PostgreSQL version and RDS edition	Monitoring type	Metric
PostgreSQL 10 on RDS High-availability Edition with local SSDs, PostgreSQL 10 on RDS Basic Edition, and PostgreSQL 9.4	Resource monitoring	Memory Usage
		CPU Utilization
		Disk Space
		IOPS
		Total Connections
PostgreSQL 10 on RDS High-availability Edition with standard or enhanced SSDs, PostgreSQL 11, PostgreSQL 12, and PostgreSQL 13	Resource monitoring	CPU Utilization
		Memory Usage
		Data Disk IOPS
		Data Disk Usage
		Disk Space Used (MB)
	Engine monitoring	Connections
		Operation Rows
		TPS
		Longest Bloat Duration (s)
		Latency (Bytes)

PostgreSQL version and RDS edition	Monitoring type	Metric
		Slow Queries
		Long-running Transactions
		2PC Transactions

23.2. Set the monitoring frequency of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to set the monitoring frequency of an ApsaraDB RDS for PostgreSQL instance.

Context

ApsaraDB RDS for PostgreSQL provides the following three monitoring frequencies:

- Every 5 Seconds

  > **Note** If the RDS instance runs the RDS Basic Edition or its memory capacity is less than 8 GB, the Every 5 Seconds monitoring frequency is not supported.

- Every 60 Seconds
- Every 300 Seconds

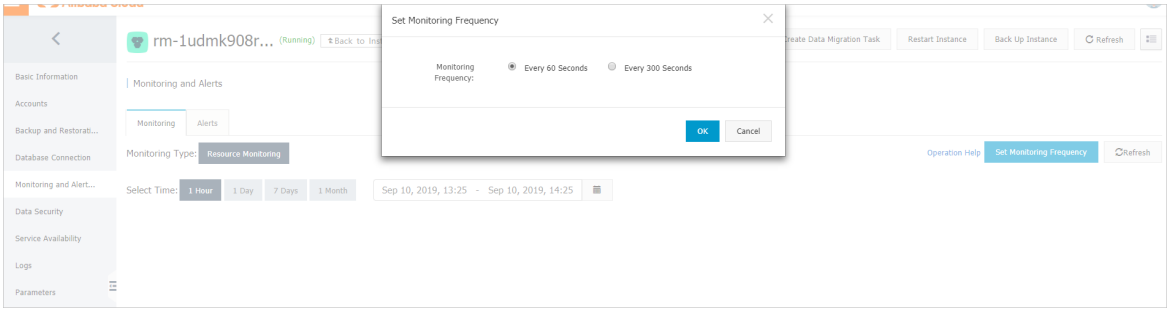
> **Note**

Procedure

1.
2. In the left-side navigation pane, click **Monitoring and Alerts**.

   > **Note** The available metrics vary based on the specified database engine. For more information, see View the resource and engine metrics of an ApsaraDB RDS for PostgreSQL instance.

3. Click the **Monitoring** tab.
4. Click **Set Monitoring Frequency**.
5. In the **Set Monitoring Frequency** dialog box, select a monitoring frequency and click **OK**.

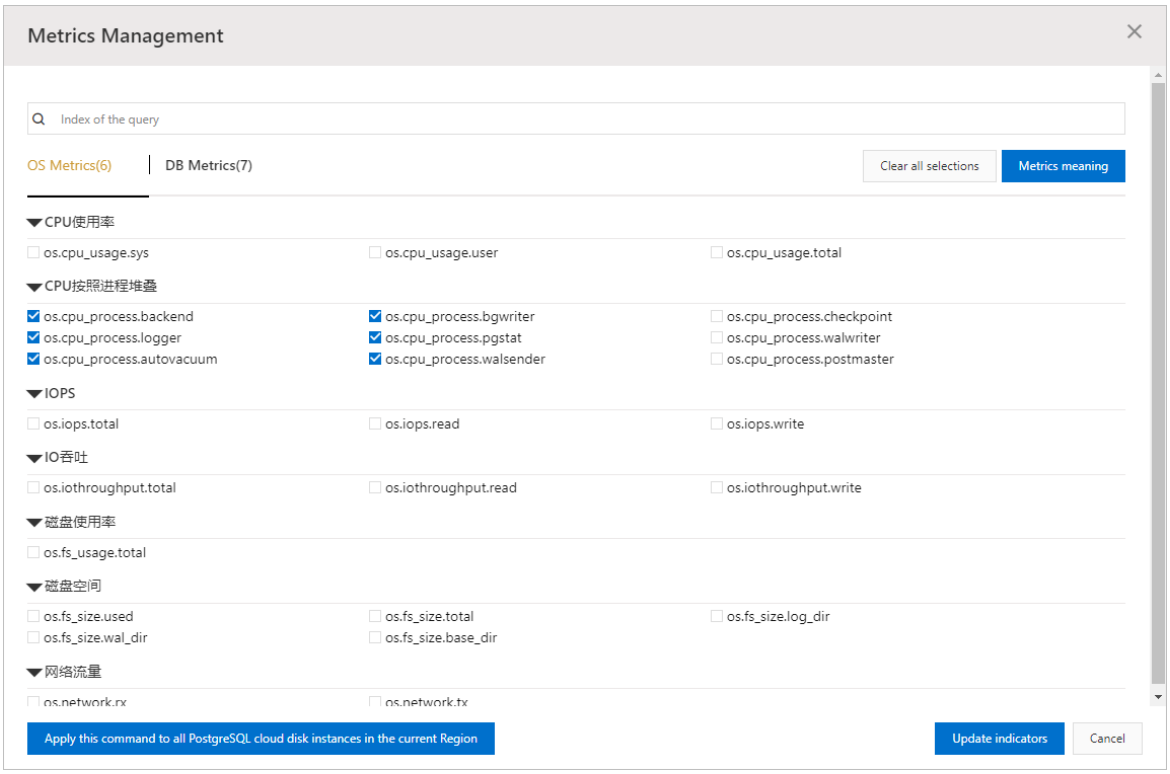


23.3. View the Enhanced Monitoring metrics of an ApsaraDB RDS for PostgreSQL instance

ApsaraDB RDS for PostgreSQL provides Enhanced Monitoring metrics in addition to Standard Monitoring metrics. This topic describes how to view the Enhanced Monitoring metrics of an ApsaraDB RDS for PostgreSQL instance in the ApsaraDB RDS console.

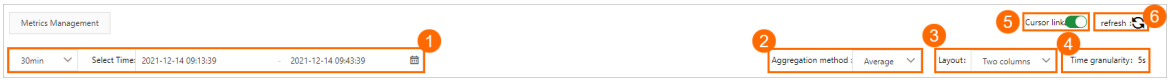
Procedure

1.
2. In the left-side navigation pane, click **Monitoring and Alerts**.
3. On the **Enhanced Monitoring** tab, click **Metrics Management**. On the **OS Metrics** tab and **DB Metrics** tab of the Metrics Management dialog box, select the metrics that you want to view. For more information, see the "References" section of this topic.



> **Note**
>
> - A maximum of 30 metrics can be displayed on the **Enhanced Monitoring** tab.
> - You can apply the selected metrics to all ApsaraDB RDS for PostgreSQL instances in the region of the current RDS instance. This includes the existing ApsaraDB RDS for PostgreSQL instances and all new ApsaraDB RDS for PostgreSQL instances that you create at a later time.
>   - If the current RDS instance is equipped with standard SSDs or enhanced SSDs (ESSDs), you can apply the selected metrics to all ApsaraDB RDS for PostgreSQL instances that are equipped with standard SSDs or ESSDs in the region of the current RDS instance.
>   - If the current RDS instance is equipped with local SSDs, you can apply the selected metrics to all ApsaraDB RDS for PostgreSQL instances that are equipped with local SSDs in the region of the current RDS instance.

4. Click **Update indicators**. On the **Enhanced Monitoring** tab, view the metrics that you selected.
5. On the **Enhanced Monitoring** tab, specify query criteria based on which you want to filter monitoring data.



No.	Functionality	Description
①	Time range	You can query monitoring data over a preset time range or a custom time range. ○ The preset time range can be 30 minutes, 1 hour, 2 hours, 6 hours, 1 day, 7 days, or 30 days. ○ The custom time range is specified by a start time and an end time in the following format: YYYY-MM-DD hh:mm:ss - YYYY-MM-DD hh:mm:ss.
②	Aggregation method	You can specify the method based on which ApsaraDB RDS aggregates monitoring data. The following three aggregation methods are supported: ○ Average ○ Maximum ○ Minimum
③	Layout	You can adjust the layout in which charts displayed. The following layouts are supported: ○ A column ○ Two columns ○ Three columns ○ Four columns

No.	Functionality	Description
④	Time granularity	You can specify the time granularity of the x-axis in each chart that is displayed. The time granularity varies based on the time range that you specify. The following relationships exist between the time granularity and the time range: ◦ If the time range is less than or equal to 1 hour, the time granularity is 5 seconds. ◦ If the time range is greater than 1 hour and less than or equal to 2 hours, the time granularity is 10 seconds. ◦ If the time range is greater than 2 hours and less than or equal to 6 hours, the time granularity is 30 seconds. ◦ If the time range is greater than 6 hours and less than or equal to 12 hours, the time granularity is 1 minute. ◦ If the time range is greater than 12 hours and less than or equal to 1 day, the time granularity is 2 minutes. ◦ If the time range is greater than 1 day and less than or equal to 5 days, the time granularity is 10 minutes. ◦ If the time range is greater than 5 days and less than or equal to 15 days, the time granularity is 30 minutes. ◦ If the time range is greater than 15 days and less than or equal to 30 days, the time granularity is 1 hour.
⑤	Cursor link	You can turn on the Cursor link switch. When you move the pointer over a specific point in time on the x-axis of a chart, all charts on the Enhanced Monitoring tab display the monitoring data that is collected at that specific point in time.
⑥	Refresh	You can manually refresh the Enhanced Monitoring tab to update monitoring data.

References

The following table describes the OS metrics and database metrics that are supported. In the following table, ticks (√) indicate that a metric is supported, and crosses (×) indicate that a metric is not supported.

OS metrics

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
CPU utilization	• Average value: os.cpu_usage.sys.avg • Maximum value: os.cpu_usage.sys.max • Minimum value: os.cpu_usage.sys.min	The CPU utilization for the OS. The value of this metric is calculated based on the following formula: CPU utilization for the OS = Number of CPUs that are consumed by the OS/Total number of CPUs.	%	✔	✔
	• Average value: os.cpu_usage.user.avg • Maximum value: os.cpu_usage.user.max • Minimum value: os.cpu_usage.user.min	The CPU utilization for the user. The value of this metric is calculated based on the following formula: CPU utilization for the user = Number of CPUs consumed by the user/Total number of CPUs.	%	✔	✔
	• Average value: os.cpu_usage.total.avg • Maximum value: os.cpu_usage.total.max • Minimum value: os.cpu_usage.total.min	The CPU utilization for the server. The value of this metric is calculated based on the following formula: CPU utilization for the server = Number of CPUs consumed by the server/Total number of CPUs.	%	✔	✔
	• Average value: os.cpu_process.backend.avg • Maximum value: os.cpu_process.backend.max • Minimum value: os.cpu_process.backend.min	The CPU utilization for the backend process. If one CPU is consumed, the CPU utilization is 100%. If two CPUs are consumed, the CPU utilization is 200%. In this way, you can calculate the CPU utilization for the backend process.	%	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
	• Average value: os.cpu_process.bgwriter.avg • Maximum value: os.cpu_process.bgwriter.max • Minimum value: os.cpu_process.bgwriter.min	The CPU utilization for the bgwriter process. If one CPU is consumed, the CPU utilization is 100%. If two CPUs are consumed, the CPU utilization is 200%. In this way, you can calculate the CPU utilization for the bgwriter process.	%	✔	✔
	• Average value: os.cpu_process.checkpoint.avg • Maximum value: os.cpu_process.checkpoint.max • Minimum value: os.cpu_process.checkpoint.min	The CPU utilization for the checkpoint process. If one CPU is consumed, the CPU utilization is 100%. If two CPUs are consumed, the CPU utilization is 200%. In this way, you can calculate the CPU utilization for the checkpoint process.	%	✔	✔
	• Average value: os.cpu_process.logger.avg • Maximum value: os.cpu_process.logger.max • Minimum value: os.cpu_process.logger.min	The CPU utilization for the logger process. If one CPU is consumed, the CPU utilization is 100%. If two CPUs are consumed, the CPU utilization is 200%. In this way, you can calculate the CPU utilization for the logger process.	%	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
CPU consumption by process	• Average value: os.cpu_process.pgstat.avg • Maximum value: os.cpu_process.pgstat.max • Minimum value: os.cpu_process.pgstat.min	The CPU utilization for the pgstat process. If one CPU is consumed, the CPU utilization is 100%. If two CPUs are consumed, the CPU utilization is 200%. In this way, you can calculate the CPU utilization for the pgstat process.	%	✔	✔
	• Average value: os.cpu_process.walwriter.avg • Maximum value: os.cpu_process.walwriter.max • Minimum value: os.cpu_process.walwriter.min	The CPU utilization for the walwriter process. If one CPU is consumed, the CPU utilization is 100%. If two CPUs are consumed, the CPU utilization is 200%. In this way, you can calculate the CPU utilization for the walwriter process.	%	✔	✔
	• Average value: os.cpu_process.autovacuum.avg • Maximum value: os.cpu_process.autovacuum.max • Minimum value: os.cpu_process.autovacuum.min	The CPU utilization for the autovacuum process. If one CPU is consumed, the CPU utilization is 100%. If two CPUs are consumed, the CPU utilization is 200%. In this way, you can calculate the CPU utilization for the autovacuum process.	%	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
	• Average value: os.cpu_process.walsender.avg • Maximum value: os.cpu_process.walsender.max • Minimum value: os.cpu_process.walsender.min	The CPU utilization for the walsender process. If one CPU is consumed, the CPU utilization is 100%. If two CPUs are consumed, the CPU utilization is 200%. In this way, you can calculate the CPU utilization for the walsender process.	%	✔	✔
	• Average value: os.cpu_process.postmaster.avg • Maximum value: os.cpu_process.postmaster.max • Minimum value: os.cpu_process.postmaster.min	The CPU utilization for the postmaster process. If one CPU is consumed, the CPU utilization is 100%. If two CPUs are consumed, the CPU utilization is 200%. In this way, you can calculate the CPU utilization for the postmaster process.	%	✔	✔
	• Average value: os.iops.total.avg • Maximum value: os.iops.total.max • Minimum value: os.iops.total.min	The disk read and write IOPS of the server.	Counts/s	□	✔
	• Average value: os.iops.read.avg • Maximum value: os.iops.read.max • Minimum value: os.iops.read.min	The disk read IOPS of the server.	Counts/s	□	✔

Category IOPS	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
	• Average value: os.iops.write.avg • Maximum value: os.iops.write.max • Minimum value: os.iops.write.min	The disk write IOPS of the server.	Counts/s	□	✔
	• Average value: os.iops.data.avg • Maximum value: os.iops.data.max • Minimum value: os.iops.data.min	The IOPS of the local data disk.	Counts/s	✔	□
	• Average value: os.iops.wal.avg • Maximum value: os.iops.wal.max • Minimum value: os.iops.wal.min	The IOPS of the local log disk.	Counts/s	✔	□
	• Average value: os.iothroughput.total.avg • Maximum value: os.iothroughput.total.max • Minimum value: os.iothroughput.total.min	The disk read and write throughput of the server.	MB/s	□	✔
	• Average value: os.iothroughput.read.avg • Maximum value: os.iothroughput.read.max • Minimum value: os.iothroughput.read.min	The disk read throughput of the server.	MB/s	□	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
I/O throughout	• Average value: os.iothroughput.write.avg • Maximum value: os.iothroughput.write.max • Minimum value: os.iothroughput.write.min	The disk write throughput of the server.	MB/s	□	✔
	• Average value: os.iothroughput.data.avg • Maximum value: os.iothroughput.data.max • Minimum value: os.iothroughput.data.min	The read and write throughput of the local data disk.	MB/s	✔	□
	• Average value: os.iothroughput.wal.avg • Maximum value: os.iothroughput.wal.max • Minimum value: os.iothroughput.wal.min	The read and write throughput of the local log disk.	MB/s	✔	□
Disk usage	• Average value: os.fs_usage.total.avg • Maximum value: os.fs_usage.total.max • Minimum value: os.fs_usage.total.min	The disk usage of the server.	%	□	✔
	• Average value: os.fs_size.used.avg • Maximum value: os.fs_size.used.max • Minimum value: os.fs_size.used.min	The used disk space of the server.	MB	□	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
Disk space	• Average value: os.fs_size.total.avg • Maximum value: os.fs_size.total.max • Minimum value: os.fs_size.total.min	The total disk space of the server.	MB	□	✔
	• Average value: os.fs_size.log_dir.avg • Maximum value: os.fs_size.log_dir.max • Minimum value: os.fs_size.log_dir.min	The size of log files. This includes audit log files, error log files, and slow SQL log files.	MB	✔	✔
	• Average value: os.fs_size.wal_dir.avg • Maximum value: os.fs_size.wal_dir.max • Minimum value: os.fs_size.wal_dir.min	The size of write-ahead logging (WAL) files.	MB	✔	✔
	• Average value: os.fs_size.base_dir.avg • Maximum value: os.fs_size.base_dir.max • Minimum value: os.fs_size.base_dir.min	The size of data files. This excludes log files and WAL files.	MB	✔	✔
	• Average value: os.network.rx.avg • Maximum value: os.network.rx.max • Minimum value: os.network.rx.min	The throughput of inbound traffic of the server.	MB/s	□	✔

Network traffic	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
	• Average value: os.network.tx.avg • Maximum value: os.network.tx.max • Minimum value: os.network.tx.min	The throughput of outbound traffic of the server.	MB/s	□	✔

Database metrics

> **Note** For more information about database metrics, see the PostgreSQL documentation.

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
Shared buffer hit ratio	• Average value: db.buffers.hit_ratio.avg • Maximum value: db.buffers.hit_ratio.max • Minimum value: db.buffers.hit_ratio.min	The proportion of requests for which the requested content is hit in the shared buffers.	%	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
Shared buffer hits	• Average value: db.buffers.blks_hit.avg • Maximum value: db.buffers.blks_hit.max • Minimum value: db.buffers.blks_hit.min	The number of requests for which the requested content is hit in the shared buffers per second.	Blocks/s	✔	✔
	• Average value: db.io.blks_read.avg • Maximum value: db.io.blks_read.max • Minimum value: db.io.blks_read.min	The number of operations that are performed by the backend process per second to read data from the disks to the buffers.	Counts/s	✔	✔
	• Average value: db.io.buffers_backend.avg • Maximum value: db.io.buffers_backend.max • Minimum value: db.io.buffers_backend.min	The number of operations that are performed by the backend process per second to write data from the buffers to the disks.	Counts/s	✔	✔
	• Average value: db.io.buffers_checkpoint.avg • Maximum value: db.io.buffers_checkpoint.max • Minimum value: db.io.buffers_checkpoint.min	The number of operations that are performed by the checkpoint process per second to write data from the buffers to the disks.	Counts/s	✔	✔

I/O Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
	• Average value: db.io.buffers_clean.avg • Maximum value: db.io.buffers_clean.max • Minimum value: db.io.buffers_clean.min	The number of operations that are performed by the bgwriter process per second to write data from the buffers to the disks.	Counts/s	✔	✔
	• Average value: db.io.buffers_backend_fsync.avg • Maximum value: db.io.buffers_backend_fsync.max • Minimum value: db.io.buffers_backend_fsync.min	The number of times that the backend process calls the fsync() function on the disks per second.	Counts/s	✔	✔
	• Average value: db.checkpoint.checkpoints_sync_time.avg • Maximum value: db.checkpoint.checkpoints_sync_time.max • Minimum value: db.checkpoint.checkpoints_sync_time.min	The amount of time that the checkpoint process spends per second in running the fsync() function on the disks.	ms/s	✔	✔
Che					

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
ckpoint write duration	• Average value: db.checkpoint.checkpoints_write_time.avg • Maximum value: db.checkpoint.checkpoints_write_time.max • Minimum value: db.checkpoint.checkpoints_write_time.min	The amount of time that the checkpoint process spends per second in writing data from the buffers to the disks.	ms/s	✔	✔
Checkpoint quantity	• Average value: db.checkpoint.checkpoints_timed.avg • Maximum value: db.checkpoint.checkpoints_timed.max • Minimum value: db.checkpoint.checkpoints_timed.min	The number of checkpoint processes that are scheduled by the database engine per second.	Counts/s	✔	✔
	• Average value: db.checkpoint.checkpoints_req.avg • Maximum value: db.checkpoint.checkpoints_req.max • Minimum value: db.checkpoint.checkpoints_req.min	The number of checkpoint processes that are requested by the user per second.	Counts/s	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
Connections	• Average value: db.connections.active.avg • Maximum value: db.connections.active.max • Minimum value: db.connections.active.min	The number of connections in the active state.	Counts	✔	✔
	• Average value: db.connections.waiting.avg • Maximum value: db.connections.waiting.max • Minimum value: db.connections.waiting.min	The number of connections in the waiting state.	Counts	✔	✔
	• Average value: db.connections.idle.avg • Maximum value: db.connections.idle.max • Minimum value: db.connections.idle.min	The number of connections in the idle state.	Counts	✔	✔
	• Average value: db.connections.total.avg • Maximum value: db.connections.total.max • Minimum value: db.connections.total.min	The total number of connections.	Counts	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
	• Average value: db.connections.spec.avg • Maximum value: db.connections.spec.max • Minimum value: db.connections.spec.min	The maximum number of connections that are allowed.	Counts	✔	✔
TPS	• Average value: db.transactions.xact_commit.avg • Maximum value: db.transactions.xact_commit.max • Minimum value: db.transactions.xact_commit.min	The number of write transactions that are committed per second.	Counts/s	✔	✔
	• Average value: db.transactions.xact_rollback.avg • Maximum value: db.transactions.xact_rollback.max • Minimum value: db.transactions.xact_rollback.min	The number of write transactions that are rolled back per second.	Counts/s	✔	✔
	• Average value: db.transactions.active.avg • Maximum value: db.transactions.active.max • Minimum value: db.transactions.active.min	The number of transactions in the active state.	Counts	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
Transaction status	• Average value: db.transactions.waiting.avg • Maximum value: db.transactions.waiting.max • Minimum value: db.transactions.waiting.min	The number of transactions in the waiting state.	Counts	✔	✔
	• Average value: db.transactions.idle.avg • Maximum value: db.transactions.idle.max • Minimum value: db.transactions.idle.min	The number of transactions in the idle state. We recommend that you check and process these transactions at the earliest opportunity.	Counts	✔	✔
	• Average value: db.sql.tup_returned.avg • Maximum value: db.sql.tup_returned.max • Minimum value: db.sql.tup_returned.min	The number of rows that are returned per second.	Tuples/s	✔	✔
	• Average value: db.sql.tup_fetched.avg • Maximum value: db.sql.tup_fetched.max • Minimum value: db.sql.tup_fetched.min	The number of rows that are read per second.	Tuples/s	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
SQL	• Average value: db.sql.tup_inserted.avg • Maximum value: db.sql.tup_inserted.max • Minimum value: db.sql.tup_inserted.min	The number of rows that are inserted per second.	Tuples/s	✔	✔
	• Average value: db.sql.tup_deleted.avg • Maximum value: db.sql.tup_deleted.max • Minimum value: db.sql.tup_deleted.min	The number of rows that are deleted per second.	Tuples/s	✔	✔
	• Average value: db.sql.tup_updated.avg • Maximum value: db.sql.tup_updated.max • Minimum value: db.sql.tup_updated.min	The number of rows that are updated per second.	Tuples/s	✔	✔
	• Average value: db.slow_sql.one_second.avg • Maximum value: db.slow_sql.one_second.max • Minimum value: db.slow_sql.one_second.min	The number of SQL statements that have been running for 1 second.	Counts	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
Slow SQL logs	• Average value: db.slow_sql.three_seconds.avg • Maximum value: db.slow_sql.three_seconds.max • Minimum value: db.slow_sql.three_seconds.min	The number of SQL statements that have been running for 3 seconds.	Counts	✔	✔
	• Average value: db.slow_sql.five_seconds.avg • Maximum value: db.slow_sql.five_seconds.max • Minimum value: db.slow_sql.five_seconds.min	The number of SQL statements that have been running for 5 seconds.	Counts	✔	✔
	• Average value: db.long_transactions.active_one_second.avg • Maximum value: db.long_transactions.active_one_second.max • Minimum value: db.long_transactions.active_one_second.min	The number of transactions that have been running for 1 second.	Counts	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
	• Average value: db.long_transactions.active_three_seconds.avg • Maximum value: db.long_transactions.active_three_seconds.max • Minimum value: db.long_transactions.active_three_seconds.min	The number of transactions that have been running for 3 seconds.	Counts	✔	✔
	• Average value: db.long_transactions.idle_one_second.avg • Maximum value: db.long_transactions.idle_one_second.max • Minimum value: db.long_transactions.idle_one_second.min	The number of transactions that have been idle for 1 second.	Counts	✔	✔
	• Average value: db.long_transactions.idle_three_seconds.avg • Maximum value: db.long_transactions.idle_three_seconds.max • Minimum value: db.long_transactions.idle_three_seconds.min	The number of transactions that have been idle for 3 seconds.	Counts	✔	✔

Long transactions Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
	• Average value: db.long_transactions.idle_five_seconds.avg • Maximum value: db.long_transactions.idle_five_seconds.max • Minimum value: db.long_transactions.idle_five_seconds.min	The number of transactions that have been idle for 5 seconds.	Counts	✔	✔
	• Average value: db.long_transactions.two_pc_one_second.avg • Maximum value: db.long_transactions.two_pc_one_second.max • Minimum value: db.long_transactions.two_pc_one_second.min	The number of two-phase transactions that have been running for 1 second.	Counts	✔	✔
	• Average value: db.long_transactions.two_pc_three_seconds.avg • Maximum value: db.long_transactions.two_pc_three_seconds.max • Minimum value: db.long_transactions.two_pc_three_seconds.min	The number of two-phase transactions that have been running for 3 seconds.	Counts	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
	• Average value: db.long_transactions.two_pc_five_seconds.avg • Maximum value: db.long_transactions.two_pc_five_seconds.max • Minimum value: db.long_transactions.two_pc_five_seconds.min	The number of two-phase transactions that have been running for 5 seconds.	Counts	✔	✔
Temporary files	• Average value: db.temp.temp_files.avg • Maximum value: db.temp.temp_files.max • Minimum value: db.temp.temp_files.min	The number of temporary files that are generated per second.	Counts/s	✔	✔
Temporary file size	• Average value: db.temp.temp_bytes.avg • Maximum value: db.temp.temp_bytes.max • Minimum value: db.temp.temp_bytes.min	The size of temporary files that are generated per second.	Bytes/s	✔	✔
Duration of bloat	• Average value: db.swell.swell_time.avg • Maximum value: db.swell.swell_time.max • Minimum value: db.swell.swell_time.min	The execution duration of the longest transaction.	s	✔	✔

Category	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
Maximum transaction ID	• Average value: db.age.max_age.avg • Maximum value: db.age.max_age.max • Minimum value: db.age.max_age.min	The maximum transaction ID on the RDS instance.	xids	✔☺	✔☺
Synchronization latency to read-only	• Average value: db.ro_replica.replay_lag.avg • Maximum value: db.ro_replica.replay_lag.max • Minimum value: db.ro_replica.replay_lag.min	The latency at which the attached read-only RDS instances replay logs.	s	□	✔☺
	• Average value: db.ro_replica.write_lag.avg • Maximum value: db.ro_replica.write_lag.max • Minimum value: db.ro_replica.write_lag.min	The latency at which the attached read-only RDS instances write data.	s	□	✔☺
	• Average value: db.ro_replica.flush_lag.avg • Maximum value: db.ro_replica.flush_lag.max • Minimum value: db.ro_replica.flush_lag.min	The latency at which the attached read-only RDS instances flush data.	s	□	✔☺

inst anc es Cat ego ry	Metric	Description	Unit	ApsaraDB RDS instance that runs PostgreSQL 9.4 or PostgreSQL 10 with local SSDs	ApsaraDB RDS instance that runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, or PostgreSQL 13 with standard SSDs or ESSDs
Logical replication	• Average value: db.slots.max_slot_wal_delay.avg • Maximum value: db.slots.max_slot_wal_delay.max • Minimum value: db.slots.max_slot_wal_delay.min	The maximum latency that is allowed for the replication slot to replicate WAL records. The WAL records that follow the replication start position must be retained. If the replication start position indicates a WAL record that has a relatively high log sequence number (LSN), WAL records may pile up. In this case, we recommend that you make sure these WAL records are processed at the earliest opportunity.	MB	✔	✔

23.4. Manage the alert rules of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to manage the alert rules of an ApsaraDB RDS for PostgreSQL instance. You can turn on the Initiative Alert switch in the ApsaraDB RDS console to enable the default alert rules that are provided by the monitoring and alerting feature. You can also configure custom alert rules in the ApsaraDB RDS console. If the value of a metric meets the conditions that are specified in the alert rule for the metric, an alert is triggered. ApsaraDB RDS sends the alert to all contacts in the alert contact group that is associated with the alert rule.

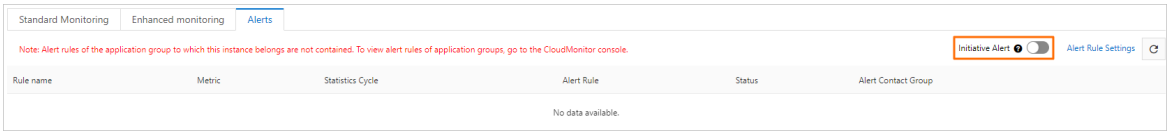
Context

The monitoring and alerting feature of ApsaraDB RDS is implemented by using Cloud Monitor. Cloud Monitor allows you to configure metrics and alert rules. You can also associate alert groups with metrics. If a metric meets the conditions that are specified in an alert rule, alerts are sent as emails to all the contacts in the alert group that is associated with the metric.

Enable default alert rules

The default alert rules help you monitor crucial metrics and identify abnormal metrics at the earliest opportunity.

1.
2. In the left-side navigation pane, click **Monitoring and Alerts**.
3. Click the **Alerts** tab.
4. In the upper-right corner of the tab, turn on the **Initiative Alert** switch.



After you turn on the Initiative Alert switch, ApsaraDB RDS monitors the crucial metrics of the RDS instance based on the default alert rules. The following table describes the default alert rules.

Alert rule	Metric	Statistics cycle	Description
SystemDefault_acs_rds_dashboard_PG_RO_ReadLag	Read-only synchronization delay	60 seconds	If the average value of the Read-only synchronization delay metric is greater than or equal to 7,200 seconds in five consecutive statistics cycles, an alert is triggered. ApsaraDB RDS sends the alert to all contacts in the alert contact group.
SystemDefault_acs_rds_dashboard_conn_usgae	Connections Usage	60 seconds	If the average value of the Connections Usage metric is greater than or equal to 90% in five consecutive statistics cycles, an alert is triggered. ApsaraDB RDS sends the alert to all contacts in the alert contact group.
SystemDefault_acs_rds_dashboard_local_fs_size_usage	Disk Space Usage	60 seconds	If the average value of the Disk Space Usage metric is greater than or equal to 90% in five consecutive statistics cycles, an alert is triggered. ApsaraDB RDS sends the alert to all contacts in the alert contact group.
SystemDefault_acs_rds_dashboard_cpu_usage	CPU utilization	60 seconds	If the average value of the CPU utilization metric is greater than or equal to 90% in five consecutive statistics cycles, an alert is triggered. ApsaraDB RDS sends the alert to all contacts in the alert contact group.

Alert rule	Metric	Statistics cycle	Description
SystemDefault_acs_rds_dashboard_iops_usage	IOPS Utilization	60 seconds	If the average value of the IOPS Utilization metric is greater than or equal to 80% in five consecutive statistics cycles, an alert is triggered. ApsaraDB RDS sends the alert to all contacts in the alert contact group.
SystemDefault_acs_rds_dashboard_PG_MaxSlotWalDelay	Maximum Replication Slot Latency	60 seconds	If the average value of the Maximum Replication Slot Latency metric is greater than or equal to 1,024 MB in five consecutive statistics cycles, an alert is triggered. ApsaraDB RDS sends the alert to all contacts in the alert contact group.
SystemDefault_acs_rds_dashboard_PG_SwellTime	Longest Transaction Duration	60 seconds	If the average value of the Longest Transaction Duration metric is greater than or equal to 36,000 seconds in five consecutive statistics cycles, an alert is triggered. ApsaraDB RDS sends the alert to all contacts in the alert contact group.

5. Optional. Specify an alert contact. If you do not specify an alert contact, ApsaraDB RDS specifies the owner of the Alibaba Cloud account as the alert contact. You can change the alert contact.
   i. Log on to the CloudMonitor console.
   ii. In the left-side navigation pane, choose **Alerts > Alert Contacts**.
   iii. On the Alert Contacts tab, click **Create Alert Contact**. For more information about how to create an alert contact, see Create an alert contact or alert contact group.
   iv. In the search box on the **Alert Contact Group** tab, enter **Default Contact Group** to find the default contact group. Then, click the icon in the **Actions** column for the default contact group.
   v. On the **Create Alert Contact Group** tab, change the alert contact.

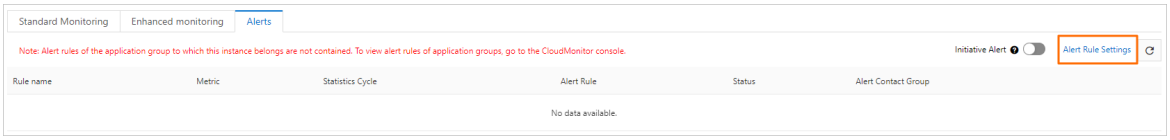
Configure custom alert rules

ApsaraDB RDS allows you to configure custom alert rules for an RDS instance based on your business requirements. The following table describes the metrics that you can configure in custom alert rules.

PG_DBAge	PG_InactiveSlots	PG_MaxExecutingSQLTime
PG_MaxSlotWalDelay	PG_RO_ReadLag	PG_RO_StreamingStatus
PG_ReplayLatency	PG_SwellTime	PG_ActiveConnectionsPerCpu
PG_ConnectionsUtilization	PG_CPUUtilization	PG_IOPSUtilization
PG_INODEUtilization	PG_DISKUtilization	PG_MemoryUtilization

1.
2. In the left-side navigation pane, click **Monitoring and Alerts**.

3. Click the **Alerts** tab.
4. Click **Set Alert Rule** to go to the CloudMonitor console.



5. Create an alert contract group. For more information, see Create an alert contact or alert contact group.
6. Create an alert rule. For more information, see Create a threshold-triggered alert rule.

> **Note**
>
> - When you create an alert rule, set the Product parameter to **RDS-PostgreSQL**.
> - You can also monitor resources based on tags. For more information, see Monitor resources based on tags.

24.Data security

24.1. Switch an ApsaraDB RDS for PostgreSQL instance to the enhanced whitelist mode

This topic describes how to switch an ApsaraDB RDS for PostgreSQL instance from the standard whitelist mode to the enhanced whitelist mode. The enhanced whitelist mode provides higher security than the standard whitelist mode.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions and RDS editions:

- PostgreSQL 10 on RDS High-availability Edition with local SSDs
- PostgreSQL 9.4 on RDS High-availability Edition with local SSDs

Context

RDS instances support the following network isolation modes:

- Standard whitelist mode

  A standard IP address whitelist can contain IP addresses from both the classic network and VPCs.

- Enhanced whitelist mode

  An enhanced IP address whitelist can contain only the IP addresses from the classic network or VPCs. When you create an enhanced IP address whitelist, you must specify the network type of the enhanced IP address whitelist.

Changes incurred

- If your RDS instance resides in a VPC, an IP address whitelist of the VPC network type is automatically created. The new IP address whitelist contains all IP addresses and CIDR blocks that are replicated from the original IP address whitelists.
- If your RDS instance resides in the classic network, an IP address whitelist of the classic network type is automatically created. The new IP address whitelist contains all IP addresses and CIDR blocks that are replicated from the original IP address whitelists.
- If your RDS instance runs in hybrid access mode, the following two IP address whitelists are created: an IP address whitelist of the VPC network type and an IP address whitelist of the classic network type. Both IP address whitelists contain all IP addresses and CIDR blocks that are replicated from the original IP address whitelists. For more information, see Configure the hybrid access solution for an ApsaraDB RDS for MySQL instance.

> **Note**
>
> - After you switch to the enhanced whitelist mode, the Elastic Compute Service (ECS) instance groups that you configured remain unchanged. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
> - ApsaraDB RDS requires approximately 3 minutes to switch your RDS instance to the enhanced whitelist mode. Your application remains connected to your RDS instance during the 3-minute period.

Precautions

- After you switch to the enhanced whitelist mode, you cannot roll the instance back to the standard whitelist mode.
- In enhanced whitelist mode, an IP address whitelist of the classic network type can also be used to allow access over the Internet. If you want to access your RDS instance from a host over the Internet, you must add the public IP address of the host to an IP address whitelist of the classic network type.

Procedure

1.
2. In the left-side navigation pane, click **Data Security**.
3. On the **Whitelist Settings** tab, click **Switch to Enhanced Whitelist (Recommended)**.
4. In the message that appears, click **Confirm**.

FAQ

- My RDS instance runs in enhanced whitelist mode. If I want to access my RDS instance from a host over the Internet, how do I determine the IP address whitelist to which I must add the public IP address of the host?

  If you want to access your RDS instance from a host over the Internet, you must add the public IP address of the host to an IP address whitelist of the classic network type.

- What advantages does the enhanced whitelist mode have over the standard whitelist mode?

  The enhanced whitelist mode allows you to distinguish between the IP addresses from the classic network and the IP addresses from VPCs. If you add an IP address to an IP address whitelist of the VPC network type, the IP address is granted access to your RDS instance only within the specified VPC. However, the IP address is not granted access to your RDS instance over the Internet. This increases the security of your RDS instance.

24.2. Configure an IP address whitelist and security group for an ApsaraDB RDS for PostgreSQL instance

24.2.1. Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance. An IP address whitelist allows only the specified devices to access your RDS instance.

For more information about how to configure an IP address whitelist for an RDS instance that runs a different database engine, see the following topics:

- Configure an IP address whitelist for an ApsaraDB RDS for MySQL instance
- Configure an IP address whitelist for an ApsaraDB RDS for SQL Server instance
- Configure an IP address whitelist for an ApsaraDB RDS for MariaDB TX instance

Scenarios

An IP address whitelist consists of IP addresses and CIDR blocks that are granted access to your RDS instance. You can configure IP address whitelists to provide high-level access control and security protection for your RDS instance. We recommend that you update the configured IP address whitelists on a regular basis.

You must configure IP address whitelists in the following scenarios:

- Scenario 1

  After your RDS instance is created, you must add the IP addresses of specific devices to an IP address whitelist of your RDS instance. This way, these devices can access your RDS instance.

- Scenario 2

  Your RDS instance cannot be connected. In this case, you must check the IP address whitelists of your RDS instance and modify the IP address whitelists that are incorrectly configured.

  The following table provides the IP address whitelist configurations in various connection scenarios.

  > **Note** A virtual private cloud (VPC) is an isolated network on Alibaba Cloud. VPCs provide higher security than the classic network. For more information, see What is a VPC?

Connection scenario	Network type	IP address whitelist configuration
	The ECS instance and your RDS instance reside in the same VPC. This is the recommended connection scenario.	Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.

Connection scenario	Network type	IP address whitelist configuration
Connect an Elastic Compute Service (ECS) instance to your RDS instance	The ECS instance and your RDS instance reside in different VPCs.	Instances in different VPCs cannot communicate with each other over internal networks. Perform the following operations: i. Migrate your RDS instance to the VPC where the ECS instance resides. **Note** This operation is supported only when the ECS instance and your RDS instance reside in the same region. If the ECS instance and your RDS instance reside in different regions, we recommend that you use Data Transmission Service (DTS) to migrate your RDS instance to the region where the ECS instance resides. This way, you can ensure the stability of your database service. . ii. Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.
	The ECS instance and your RDS instance reside in the classic network.	Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.
	The ECS instance resides in the classic network. Your RDS instance resides in a VPC.	Instances of different network types cannot communicate with each other over internal networks. Perform the following operations: i. Migrate the ECS instance from the classic network to the VPC where your RDS instance resides. For more information, see Migrate an ECS instance from the classic network to a VPC. **Note** This operation is supported only when the ECS instance and your RDS instance reside in the same region. If the ECS instance and your RDS instance reside in different regions, we recommend that you use DTS to migrate your RDS instance to the region where the ECS instance resides. This way, you can ensure the stability of your database service. . ii. Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.

Connection scenario	Network type	IP address whitelist configuration
	The ECS instance resides in a VPC. Your RDS instance resides in the classic network.	Instances of different network types cannot communicate with each other over internal networks. Perform the following operations: i. Migrate your RDS instance from the classic network to the VPC where the ECS instance resides. **Note** This operation is supported only when the ECS instance and your RDS instance reside in the same region. If the ECS instance and your RDS instance reside in different regions, we recommend that you use DTS to migrate your RDS instance to the region where the ECS instance resides. This way, you can ensure the stability of your database service. . ii. Add the private IP address of the ECS instance to an IP address whitelist of your RDS instance.
Connect a self-managed host outside the cloud to your RDS instance	None.	Add the public IP address of the self-managed host to an IP address whitelist of your RDS instance. **Note** ◦ The applications that run on the self-managed host connect to the public endpoint of your RDS instance. ◦ For more information about how to obtain the public IP address of the self-managed host, see Why am I unable to connect to my ApsaraDB RDS for MySQL or ApsaraDB RDS for MariaDB instance from a local server over the Internet?

Precautions

- A maximum of 50 IP address whitelists can be configured for each RDS instance.
- When you configure IP address whitelists, the workloads on your RDS instance are not interrupted.
- The IP address whitelist that is labeled default can be cleared but cannot be deleted.
- Do not modify or delete the IP address whitelists that are generated by other Alibaba Cloud services. If you delete the IP address whitelist that is generated by an Alibaba Cloud service, the Alibaba Cloud service cannot connect to your RDS instance. For example, the IP address whitelist labeled ali_dms_group is generated by Data Management (DMS), and the IP address whitelist labeled hdm_security_ips is generated by Database Autonomy Service (DAS).
- The IP address whitelist labeled default contains only the 127.0.0.1 IP address. This indicates that no IP addresses are granted access to your RDS instance.

Configure a standard IP address whitelist

In standard whitelist mode, ApsaraDB RDS does not distinguish between the classic network and VPCs. The IP addresses or CIDR blocks in a standard IP address whitelist are granted access to your RDS instance over both the classic network and VPCs.

1.
2. In the left-side navigation pane, click **Data Security**.
3. Click **Create Whitelist**. In the dialog box that appears, configure parameters such as **Whitelist Name** to create an IP address whitelist. Alternatively, click **Modify** to the right of an existing IP address whitelist to modify the IP address whitelist.
4. Enter the IP addresses or CIDR blocks that require access to your RDS instance. Then, click OK.

   > **Note**
   > - If you enter more than one IP address or CIDR block, you must separate these IP addresses or CIDR blocks with commas (,). Do not add spaces preceding or following the commas. Example: `192.168.0.1,172.16.213.9`.
   > - A maximum of 1,000 IP addresses and CIDR blocks can be configured for each RDS instance. If you want to enter a large number of IP addresses, we recommend that you merge the IP addresses into CIDR blocks, such as 10.10.10.0/24.

5. Optional. If the RDS instance is attached with a read-only RDS instance, you can use the **Synchronize whitelist to read-only instance** parameter to configure the RDS instance to synchronize IP address whitelists to the read-only RDS instance. ApsaraDB RDS supports the synchronization of IP address whitelists to multiple read-only RDS instances.
6. Optional. Click **Loading ECS Inner IP**. In the dialog box that appears, view the IP addresses of all ECS instances that are created within your Alibaba Cloud account. Then, add the IP addresses of the ECS instances that you want to connect to the IP address whitelist.

Configure an enhanced IP address whitelist

In enhanced whitelist mode, ApsaraDB RDS distinguishes between the classic network and VPCs. You must specify the network isolation mode of each enhanced IP address whitelist. For example, if the Network Isolation Mode parameter is set to Classic Network for an IP address whitelist, the IP addresses in the IP address whitelist are granted access to your RDS instance only over the classic network and you cannot connect to your RDS instance over VPCs from these IP addresses.

The enhanced whitelist mode is supported only for RDS instances that are equipped with local SSDs. If your RDS instance runs in enhanced whitelist mode, you can perform the following procedure to configure an enhanced IP address whitelist. For more information about how to switch the network isolation mode of an RDS instance from the standard whitelist mode to the enhanced whitelist mode, see Switch an ApsaraDB RDS for PostgreSQL instance to the enhanced whitelist mode.

1.
2. In the left-side navigation pane, click **Data Security**.
3. On the Whitelist Settings tab, create or modify an IP address whitelist.
   - Create an IP address whitelist
     a. Click **Create Whitelist**.
     b. Set the **Network Isolation Mode** parameter.
     c. Enter a name in the **Whitelist Name** field, add IP addresses or CIDR blocks, and then click OK.
   - Modify an IP address whitelist.

Click **Modify** to the right of the IP address whitelist.

4. In the **Edit Whitelist** dialog box, add IP addresses or CIDR blocks and click **OK**.

   > Note
   > - If you enter more than one IP address or CIDR block, you must separate these IP addresses or CIDR blocks with commas (,). Do not add spaces preceding or following the commas. Example: `192.168.0.1,172.16.213.9`.
   > - A maximum of 1,000 IP addresses and CIDR blocks can be configured for each RDS instance. If you want to enter a large number of IP addresses, we recommend that you merge the IP addresses into CIDR blocks, such as, 10.10.10.0/24.

5. Optional. If the RDS instance is attached with a read-only RDS instance, you can use the **Synchronize whitelist to read-only instance** parameter to configure the RDS instance to synchronize IP address whitelists to the read-only RDS instance. ApsaraDB RDS supports the synchronization of IP address whitelists to multiple read-only RDS instances.
6. Optional. Click **Loading ECS Inner IP**. In the dialog box that appears, view the IP addresses of all ECS instances that are created within your Alibaba Cloud account. Then, add the IP addresses of the ECS instances that you want to connect to the IP address whitelist.

What to do next

Create a database and an account on an ApsaraDB RDS for PostgreSQL instance

Related operations

Operation	Description
DescribeDBInstanceIPArrayList	Queries the IP address whitelists of an ApsaraDB RDS instance.
ModifySecurityIps	Modifies an IP address whitelist of an ApsaraDB RDS instance.

24.2.2. Configure a security group for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure a security group for an ApsaraDB RDS for MySQL instance. A security group is a virtual firewall that is used to control the inbound and outbound traffic of the Elastic Compute Service (ECS) instances in that security group. After you add a security group to your RDS instance, all the ECS instances in that security group can access the instance.

Scenarios

After your RDS instance is created, you must configure IP address whitelists or security groups for the instance. Otherwise, your RDS instance is inaccessible. For more information about how to configure an IP address whitelist, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

For more information about security groups, see Create a security group.

Precautions

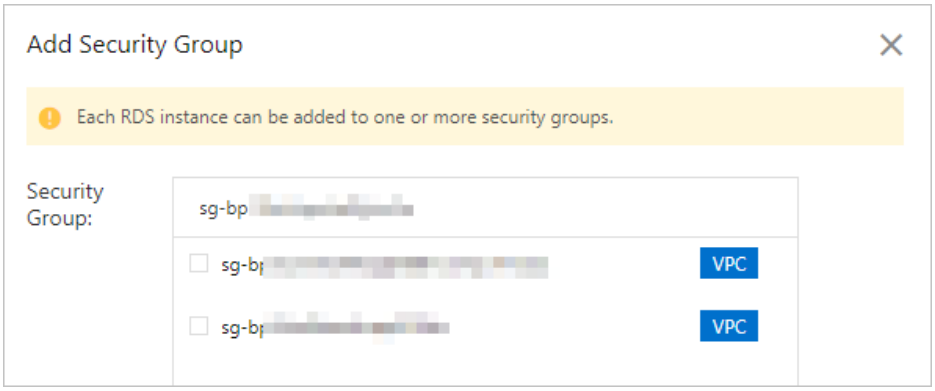
- You can configure both IP address whitelists and security groups. All the IP addresses in the configured IP address whitelists and all the ECS instances in the configured security groups are granted access to your RDS instance.
- A maximum of 10 security groups can be configured for each RDS instance.
- After the ECS instances in a configured security group are updated, the updates are automatically synchronized to the configured security group.
- You can configure only a security group that has the same network type as your RDS instance. In this case, the network types of your RDS instance and the security group that you want to configure must both be VPC or classic network.

> **Note** After you change the network type of your RDS instance, the security groups that you configured become invalid. You must configure the security groups of the specified network type again.

Procedure

1.
2. In the left-side navigation pane, click **Data Security**. On the page that appears, click the **Security Group** tab.
3. Click **Add Security Group**.

   > **Note** Security groups that are followed by a **VPC** tag contain ECS instances that reside in virtual private clouds (VPCs).

4. Select the security group that you want to add, and then click **OK**.

What to do next

Create a database and an account on an ApsaraDB RDS for PostgreSQL instance

Related operations

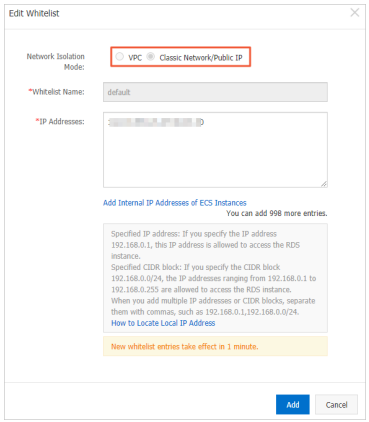
API	Description
DescribeSecurityGroupConfiguration	Queries details about the ECS security groups that are associated with an ApsaraDB RDS instance.
ModifySecurityGroupConfiguration	Modifies details about the ECS security groups that are associated with an ApsaraDB RDS instance.

24.2.3. Errors and FAQ about IP address whitelist settings in ApsaraDB RDS for PostgreSQL

This topic describes the common errors and provides answers to some commonly asked questions about the IP address whitelist settings of an ApsaraDB RDS for PostgreSQL instance.

Common errors

Error	Description	Solution
No IP address whitelists are configured. Your RDS instance has only one default IP address whitelist. The default IP address whitelist contains only the 127.0.0.1 IP address.	The 127.0.0.1 IP address indicates that no devices can access your RDS instance.	Add the IP addresses of the specified devices to an IP address whitelist.
The 0.0.0.0 entry is added to an IP address whitelist during a connectivity test.	The format of the 0.0.0.0 entry is invalid.	Change the 0.0.0.0 IP address to the 0.0.0.0/0 Classless Inter-Domain Routing (CIDR) block. **Notice** The 0.0.0.0/0 CIDR block indicates that all IP addresses are granted access to your RDS instance. We recommend that you add this CIDR block only for a connectivity test. When you run online workloads, do not add this CIDR block to an IP address whitelist.
The public IP addresses in a configured IP address whitelist are inaccessible.	• The public IP addresses dynamically change. • The tool or website that you use to query public IP addresses returns inaccurate results.	For more information, see How do I locate the IP address connected to an RDS for PostgreSQL instance?

Error	Description	Solution
The IP addresses of the specified devices are added to an enhanced IP address whitelist, and the network type of this whitelist differs from the network types of these devices.	In enhanced whitelist mode, ApsaraDB RDS distinguishes between the classic network and virtual private networks (VPCs).	Add the IP addresses to an IP address whitelist whose network type is the same as the network types of the devices. For example, if an IP address is added to an IP address whitelist of the VPC network type, you can connect to your RDS instance from the IP address only over a VPC. 

FAQ

- Can I configure both IP address whitelists and security groups for my RDS instance?

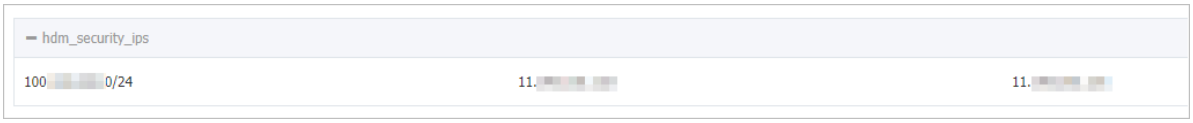
  Yes, you can configure both IP address whitelists and security groups for your RDS instance. All the IP addresses in the configured IP address whitelists and all the Elastic Compute Service (ECS) instances in the configured security groups are granted access to your RDS instance.

- After I configure an IP address whitelist for my RDS instance, does the IP address whitelist immediately take effect?

  After you configure an IP address whitelist for your RDS instance, the IP address whitelist requires about 1 minute to take effect.

- What are the IP address whitelists labeled **ali_dms_group** and **hdm_security_ips**?

  When you connect to your RDS instance from other Alibaba Cloud services, these services generate IP address whitelists upon your authorization. The generated IP address whitelists contain the IP addresses of the servers on which these services run. The IP address whitelist labeled ali_dms_group is generated by Data Management (DMS). The IP address whitelist labeled hdm_security_ips is generated by Database Autonomy Service (DAS). Do not modify or delete the IP address whitelists. If you modify or delete the IP address whitelists, these services cannot access your RDS instance. These services do not perform operations on your business data.

24.3. SSL encryption

24.3.1. Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance. SSL encryption is used to encrypt the connections from a database client to an RDS instance. This helps you protect the data that is transmitted over the connections.

Background information

SSL is a protocol that is developed to ensure secure communication and protect data. From SSL 3.0 onwards, SSL is renamed as TLS. In this topic, cloud certificates are used to describe how to configure SSL encryption.

> **Note** ApsaraDB RDS for PostgreSQL supports TLS 1.0, TLS 1.1, and TLS 1.2.

The following table provides a comparison of the SSL encryption configurations and benefits among various certificates.

Item	Use a cloud certificate to enable SSL encryption	Configure a custom certificate on an ApsaraDB RDS for PostgreSQL instance	Configure a client CA certificate on an ApsaraDB RDS for PostgreSQL instance
How to obtain	Issued by Alibaba Cloud.	Issued by a certification authority (CA) or from a self-signed certificate.	Issued from a self-signed certificate.
Validity period	365 days.	Customized.	Customized.
Number of protected endpoints	1.	1 or more.	Varies based on the cloud or custom certificate that is used. The number of protected endpoints does not vary based on the CA certificate that is used.
Purpose	Used to enable SSL encryption and used by the database client to validate the RDS instance.	Used to enable SSL encryption and used by the database client to validate the RDS instance.	Used by the RDS instance to validate the database client.

> **Note**
> - To enable SSL encryption, you must configure a cloud certificate or a custom certificate.
> - You can choose not to configure a client CA certificate, which is used by the RDS instance to validate the database client.

Prerequisites

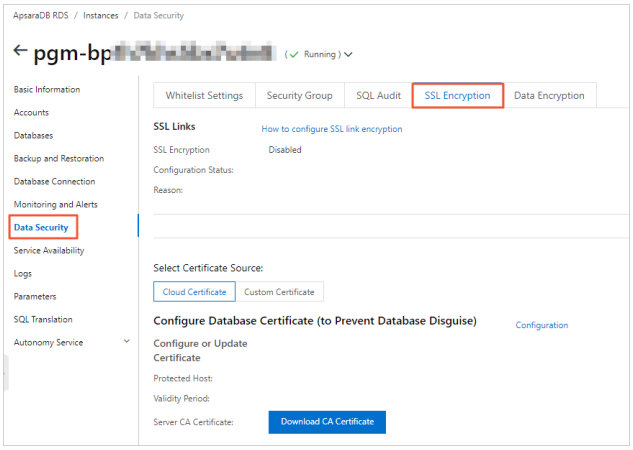
- The RDS instance runs PostgreSQL 10 or later with standard SSDs or enhanced SSDs (ESSDs).
- The pgAdmin 4 client is downloaded. For more information, see pgAdmin 4.

Precautions

- After SSL encryption is enabled, the CPU utilization and the read and write latencies increase.
- After SSL encryption is enabled, you must close the existing connection and establish a new connection to make SSL encryption take effect.
- When you configure a cloud certificate, change the endpoint that is protected by the configured cloud certificate, or disable SSL encryption, the RDS instance restarts. The restart process requires about 3 minutes. We recommend that you perform these operations during off-peak hours.

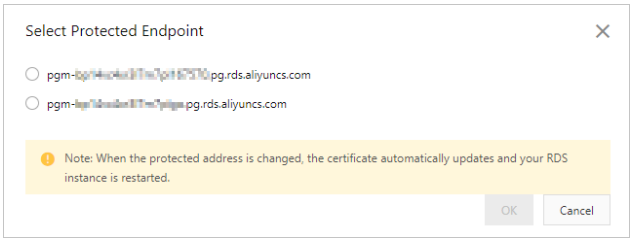
Step 1: Use a cloud certificate to enable SSL encryption

1.
2. Log on to the ApsaraDB RDS console. Find the RDS instance and click the ID of the instance. In the left-side navigation pane, click **Data Security**. On the page that appears, click the **SSL Encryption** tab.



> **Note** If the **SSL Encryption** tab cannot be found, you must check that the RDS instance meets the requirements that are stated in Prerequisites.

3. On the tab that appears, select **Cloud Certificate** for Select Certificate Source and click **Configuration** to the right of **Configure Database Certificate (to Prevent Database Disguise)**. In the dialog box that appears, select the endpoint that you want to protect.

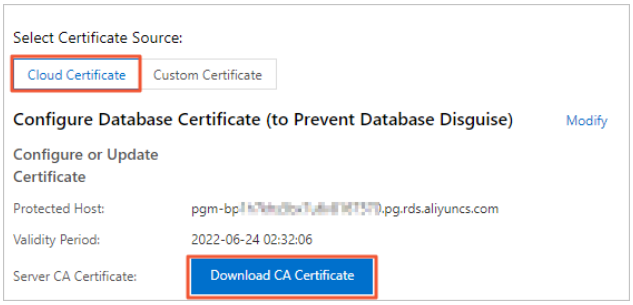


> **Note**
>
> - If you have not applied for a public endpoint, the **Select Protected Endpoint** dialog box displays only the internal endpoint of the RDS instance. If you have applied for a public endpoint, this dialog box displays both the internal endpoint and public endpoint of the RDS instance. However, each cloud certificate can protect only one endpoint. The internal endpoint is more secure than the public endpoint. Therefore, we recommend that you protect the public endpoint. For more information about how to view the internal endpoint and the public endpoint, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
> - For more information about how to protect the internal endpoint and the public endpoint at the same time, see Configure a custom certificate on an ApsaraDB RDS for PostgreSQL instance.
> - After a cloud certificate is configured, the status of the RDS instance changes from **Running** to **Modifying SSL**. After about 3 minutes, the status changes back to **Running**.

Step 2: Download the server CA certificate

After a cloud certificate is configured, the RDS instance provides a server CA certificate. When you connect to the RDS instance from the database client, the database client validates the RDS instance by using the server CA certificate.

1. Click **Cloud Certificate**. Then, click **Download CA Certificate**.



2. Decompress the file that you downloaded.

   The file that you downloaded is a package, which contains the following three files:

   - P7B file: contains the server CA certificate that can be imported into a Windows operating system.
   - PEM file: contains the server CA certificate that can be imported into an operating system rather than Windows or an application that is not Windows-based.
   - JKS file: contains the server CA certificate that is stored in a Java-supported truststore. You can use the file to import the CA certificate chain into a Java-based application. The default password is apsaradb.

Step 3: Connect to the RDS instance from the database client

In this example, pgAdmin is used to describe how to connect to the RDS instance over SSL.

You can connect to the RDS instance from the database client over SSL by using one of the following methods:

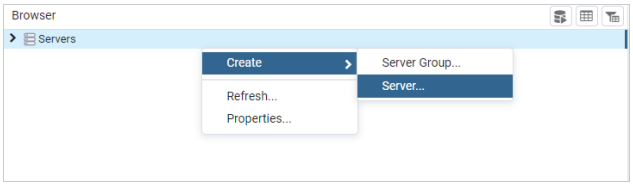
- Use PostgreSQL CLI to connect to an RDS instance over SSL
- Use JDBC to connect to an RDS instance over SSL

> **Note** Before you connect to the RDS instance, you must make sure that you have configured IP address whitelists and created accounts on the instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance and Create a database and an account on an ApsaraDB RDS for PostgreSQL instance.

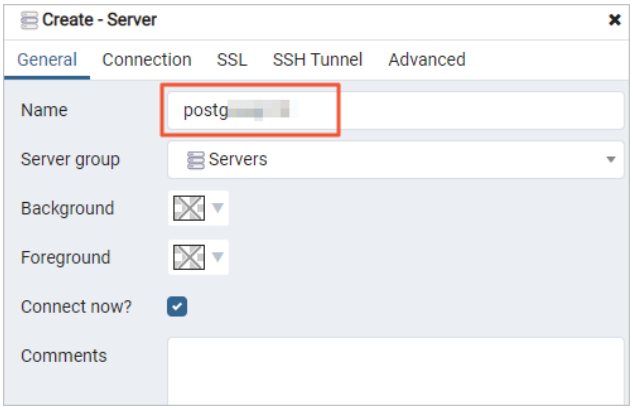
1. Start **pgAdmin 4**.

   > **Note** If you use pgAdmin that is in a later version than version 4 and you use pgAdmin for the first time, you must specify a master password to protect your saved logon credentials such as passwords.

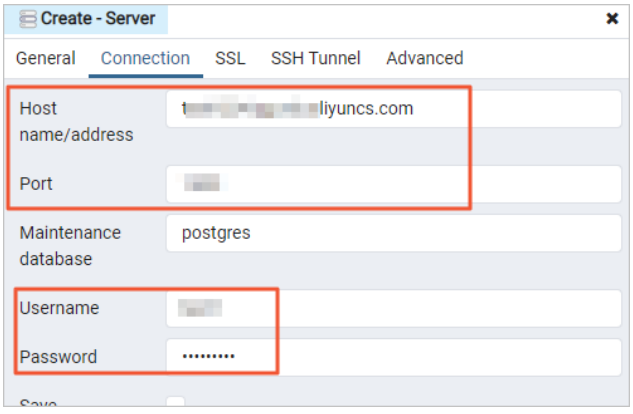
2. Right-click **Servers** and choose **Create > Server...**.



3. On the **General** tab of the Create - Server dialog box, enter the name of the server on which pgAdmin is installed.



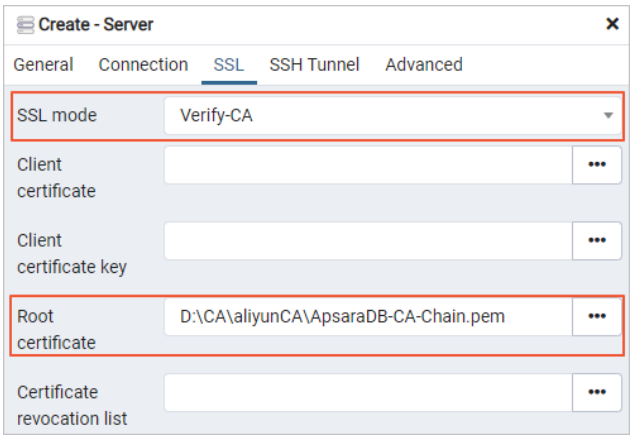
4. Click the **Connection** tab and enter the information that is used to connect to the RDS instance.



Parameter	Description

Parameter	Description
Hostname/address	Enter the endpoint of the RDS instance. If you want to connect to the RDS instance over an internal network, enter the internal endpoint of the RDS instance. If you want to connect to the RDS instance over the Internet, enter the public endpoint of the RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Port	Enter the port number that is associated with the specified endpoint.
Username	Enter the username of the account that is used to log on to the RDS instance. For more information about how to create an account for an RDS instance, see Create a database and an account on an ApsaraDB RDS for PostgreSQL instance.
Password	Enter the password of the account that is used to log on to the RDS instance.

5. Click the SSL tab and configure the required parameters. The following table describes the parameters.



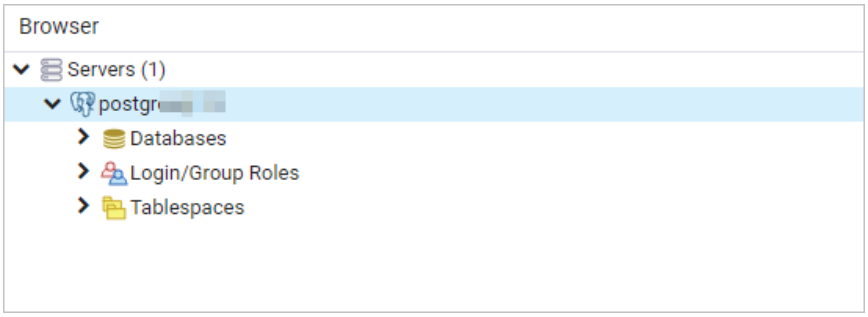
Parameter	Description
SSL mode	For security purposes, we recommend that you set this parameter to Require, Verify-CA, or Verify-Full. The following list provides the meanings of the different values of the SSL mode parameter: ◦ Require: The database client encrypts the SSL connections that are used to transmit data. However, the database client does not validate the RDS instance. ◦ Verify-CA: The database client encrypts the SSL connections that are used to transmit data and validates the RDS instance. ◦ Verify-Full: The database client encrypts the SSL connections that are used to transmit data, validates the RDS instance, and checks whether the CN or Domain Name System (DNS) specified in the server CA certificate is consistent with the value of the **Host name/address** parameter that you set at connection establishments.

Parameter	Description
Root certificate	If you set the **SSL mode** parameter to **Verify-CA** or **Verify-Full**, you must set the Root certificate parameter to the save path of the file that contains the server CA certificate. **Note** ○ In this example, the file that contains the server CA certificate is downloaded from the **SSL** tab and then is decompressed to the *D:\CA\aliyunCA* path on your computer. You can change the path based on your business requirements. ○ In pgAdmin, the file that contains the server CA certificate is in the PEM format.

6. Click **Save**.

   If the information that you enter is correct, the page that is shown in the following figure appears, which indicates that the connection to the RDS instance is successful.

   Notice The postgres database is a default system database. Do not perform operations on this database.



7.

24.3.2. Configure a custom certificate on an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure a custom certificate that is used for SSL encryption on an ApsaraDB RDS for PostgreSQL instance. In ApsaraDB RDS for PostgreSQL, SSL encryption supports cloud certificates and custom certificates.

Prerequisites

- The RDS instance runs PostgreSQL 10 or later with standard SSDs or enhanced SSDs (ESSDs).
- OpenSSL is installed.

  Note Linux operating systems are provided with OpenSSL. If you are using a Linux operating system, you do not need to install OpenSSL. If you are using a Windows operating system, you must download the OpenSSL software package and install OpenSSL. For more information, visit the Win32/Win64 OpenSSL page.

Precautions

- After SSL encryption is enabled, the CPU utilization and the read and write latencies increase.
- After SSL encryption is enabled, you must close the existing connection and establish a new connection to make SSL encryption take effect.
- When you configure a custom certificate, change the content of the configured custom certificate, or disable SSL encryption, the RDS instance restarts. The restart process requires about 3 minutes. We recommend that you perform these operations during off-peak hours.

Step 1: Create a custom certificate

Notice When you create a private key for a server certificate or self-signed certificate, do not enable password encryption. If you enable password encryption, SSL encryption cannot be enabled.

In this example, Community Enterprise Operating System (CentOS) is used. If you are using a Windows operating system, you can configure the `openssl` command by using the same configuration that you use in CentOS. In addition, if you are using a Windows operating system, you must directly copy and edit the files that you need rather than running the `cp` and `vim` commands.

1. Create a self-signed certificate. The self-signed certificate is saved in a file named ca.crt. Also, create a private key for the self-signed certificate. The private key is saved in a file named ca.key.

   ```
   openssl req -new -x509 -days 3650 -nodes -out ca.crt -keyout ca.key -subj "/CN=root-ca"
   ```

2. Create a certificate signing request (CSR). The CSR is used to request a server certificate and is saved in a file named server.csr. Also, create a private key for the server certificate. The private key is saved in a file named server.key.

   Each custom certificate can protect one or more endpoints.

   - If you want to protect a single endpoint, run the following command:

     ```
     openssl req -new -nodes -text -out server.csr -keyout server.key -subj "/CN=pgm-bpxxxxx
     .pg.rds.aliyuncs.com"
     ```

   - If you want to protect multiple endpoints, run the following command:

     a. Copy the openssl.cnf file for temporary use.

     ```
     cp /etc/pki/tls/openssl.cnf  /tmp/openssl.cnf
     ```

     Note If you are using a Windows operating system, the *openssl.cnf* file is stored in the *Installation directory of OpenSSL\bin\cnf* path. You can copy the openssl.cnf file to any other directory on your computer.

     b. Run the following command to open the openssl.cnf file:

     ```
     vim /tmp/openssl.cnf
     ```

c. Enter `i` to enable the edit mode. Then, add the following content to the openssl.cnf file:

```
# Add the following content at the end of the [ req ] element.
req_extensions = v3_req
# Add the [ v3_req ] element.
[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names
# Add the [ alt_names ] element. Then, enter the endpoint that you want to protect following each Domain Name System (DNS) record.
[ alt_names ]
DNS.1 = pgm-bpxxxxx.pg.rds.aliyuncs.com
DNS.2 = pgm-bpxxxxx.pg.rds.aliyuncs.com
```

d. Press `Esc` to exit the edit mode. Then, enter `:wq` to save the openssl.cnf file and exit.

e. Create a CSR. The CSR is used to request a server certificate and is saved in a file named server.csr. Also, create a private key for the server certificate. The private key is saved in a file named server.key.

```
openssl req -new -nodes -text -out server.csr -keyout server.key -config /tmp/openssl.cnf
```

When the server.csr file is being created, you are prompted to configure the following parameters based on your business requirements.

Parameter	Description	Example
Country Name	The code for the country where the RDS instance resides. The country code must be two characters in length. ApsaraDB RDS supports country codes that are created and maintained by the International Organization for Standardization (ISO).	CN
State or Province Name	The province where the RDS instance resides.	Zhejiang
Locality Name	The city where the RDS instance resides.	Hangzhou
Organization Name	The name of the enterprise that purchases the RDS instance.	Alibaba
Organizational Unit Name	The name of the department that uses the RDS instance within the specified enterprise.	Aliyun
Common Name	The domain name from which the request for an SSL certificate is originated. The domain name is specified in the openssl.cnf file. You do not need to specify this parameter.	-
Email Address	You do not need to specify this parameter.	-
A challenge password	You do not need to specify this parameter.	-
An optional company name	You do not need to specify this parameter.	-

3. Create a server certificate. The server certificate is saved in a file named server.crt.
   - If you want to protect a single endpoint, run the following command:

     ```
     openssl x509 -req -in server.csr -text -days 365 -CA ca.crt -CAkey ca.key -CAcreateserial  -out server.crt
     ```

   - If you want to protect multiple endpoints, run the following command:

     ```
     openssl x509 -req -in server.csr -text -days 365 -CA ca.crt -CAkey ca.key -CAcreateserial  -out server.crt -extensions v3_req -extfile /tmp/openssl.cnf
     ```

After you complete the preceding operations, you can obtain the following files:

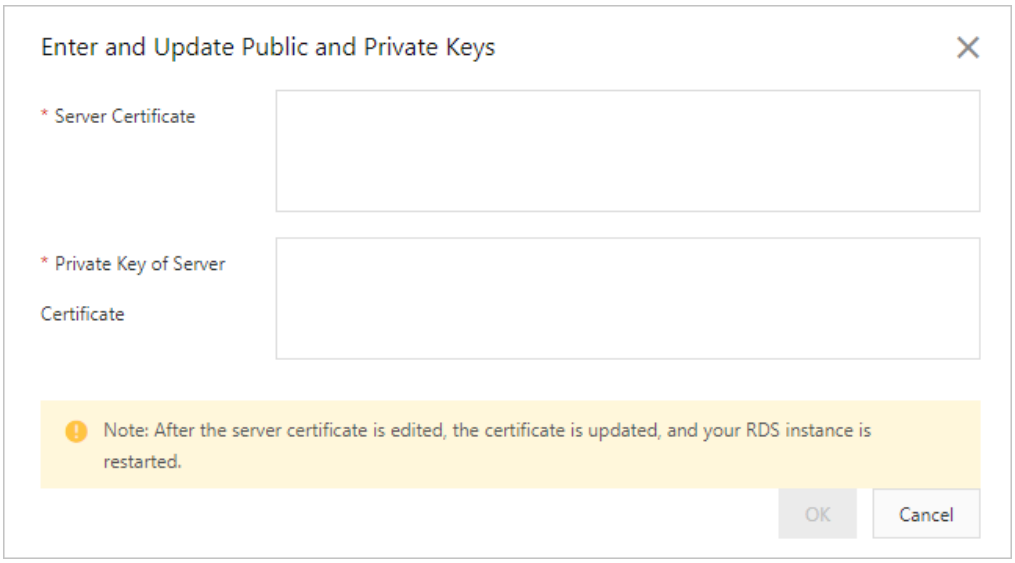
- server.crt: the file that contains the server certificate

- server.key: the file that contains the private key of the server certificate
- ca.crt: the file that contains the self-signed certificate
- ca.key: the file that contains the private key of the self-signed certificate

Step 2: Use the created custom certificate to enable SSL encryption

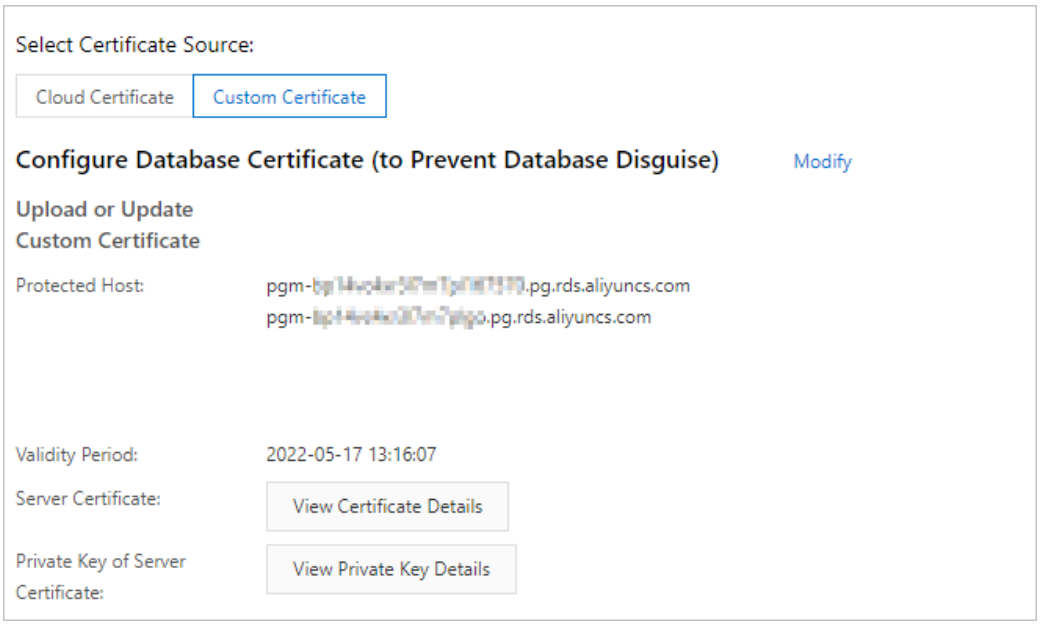
Note After a custom certificate is configured, the status of the RDS instance changes from **Running** to **Modifying SSL**. After about 3 minutes, the status changes back to **Running**.

1. Log on to the ApsaraDB RDS console. Find the RDS instance and click the ID of the instance. In the left-side navigation pane, click **Data Security**. On the page that appears, click the **SSL Encryption** tab.
2. Click **Custom Certificate**. Then, click **Configuration** next to **Configure Database Certificate (to Prevent Database Disguise)**. In the dialog box that appears, specify the server certificate and the private key of the server certificate.



Parameter	Description
Server Certificate	Enter the content of the server.crt file that you created. For more information, see the "Step 1: Create a custom certificate" section of this topic. Make sure that all the content from -----**BEGIN CERTIFICATE**----- to -----**END CERTIFICATE**----- is copied to this field.
Private Key of Server Certificate	Enter the content of the server.key file that you created. For more information, see the "Step 1: Create a custom certificate" section of this topic. Make sure that all the content from -----**BEGIN PRIVATE KEY**----- to -----**END PRIVATE KEY**----- is copied to this field.

When you request a custom certificate, you can select multiple endpoints that you want to protect. If you select multiple endpoints, you can find multiple records in the **Protected Host** parameter.



Step 3: Connect to the RDS instance from a database client

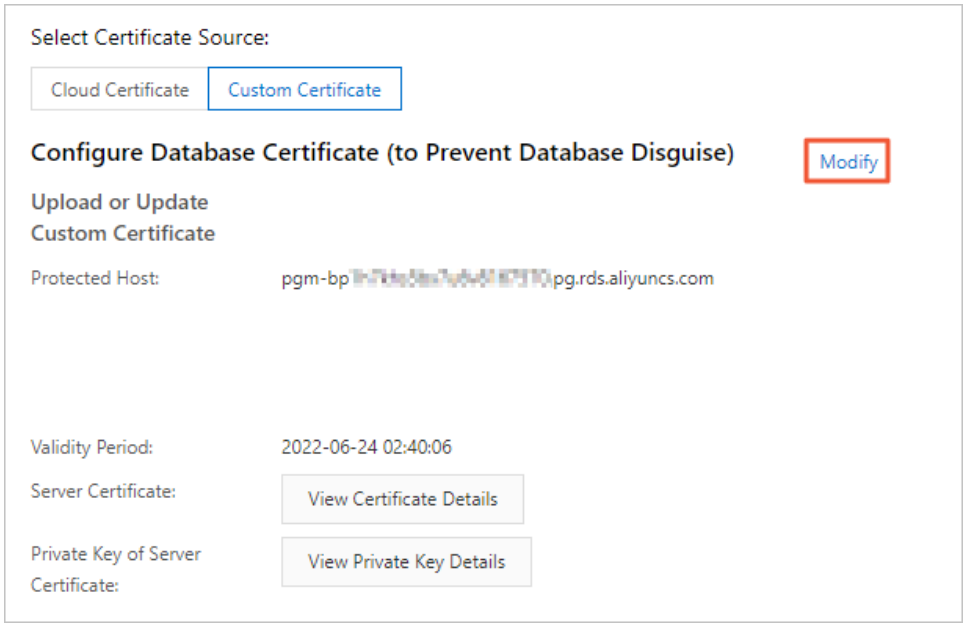
You can connect to the RDS instance from the database client over SSL by using one of the following methods:

- Use pgAdmin to connect to an RDS instance over SSL
- Use PostgreSQL CLI to connect to an RDS instance over SSL
- Use JDBC to connect to an RDS instance over SSL

Step 4: (Optional) Update the created custom certificate

> **Note** This operation triggers a restart of the RDS instance. Proceed with caution.

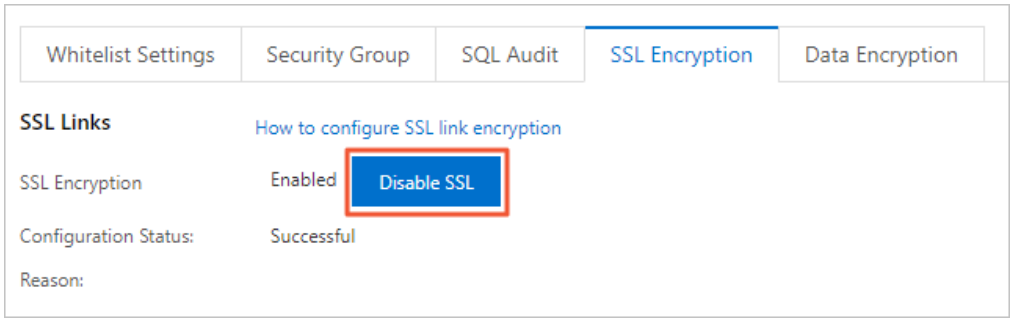
On the SSL tab, click **Modify** next to **Configure Database Certificate (to Prevent Database Disguise)**. In the dialog box that appears, enter the new server certificate that you want to use and the private key of the new server certificate.



Step 5: (Optional) Disable SSL encryption

> **Note** This operation triggers a restart of the RDS instance. Proceed with caution.

On the **SSL** tab, click **Disable SSL**.



24.3.3. Configure a client CA certificate on an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure a client certification authority (CA) certificate on an ApsaraDB RDS for PostgreSQL instance. If you use a cloud certificate or a custom certificate to enable SSL encryption on an RDS instance, a database client validates the RDS instance before the database client connects to the RDS instance. If you want the RDS instance to validate the database client, you must also configure a client CA certificate.

Prerequisites

- SSL encryption is configured, or a custom certificate is configured. For more information, see Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance or Configure a custom certificate on an

ApsaraDB RDS for PostgreSQL instance.

- OpenSSL is installed.

> **Note** Linux operating systems are provided with OpenSSL. If you are using a Linux operating system, you do not need to install OpenSSL. If you are using a Windows operating system, you must download the OpenSSL software package and install OpenSSL. For more information, visit the Win32/Win64 OpenSSL page.

Precautions

- After a client CA certificate is configured, you must close the existing connection and establish a new connection to make SSL encryption take effect.
- When you configure a client CA certificate, change the content of the configured client CA certificate, or modify a certificate revocation list (CRL) on the RDS instance, the RDS instance restarts. The restart process requires about 3 minutes. We recommend that you perform these operations during off-peak hours.

Step 1: Create a client certificate

In this example, Community Enterprise Operating System (CentOS) is used. If you are using a Windows operating system, you can configure the `openssl` command by using the same configuration that you use in CentOS.

1. Create a self-signed certificate. The self-signed certificate is saved in a file named ca1.crt. Also, create a private key for the self-signed certificate. The private key is saved in a file named ca1.key.

```
openssl req -new -x509 -days 3650 -nodes -out ca1.crt -keyout ca1.key -subj "/CN=root-ca1
"
```

2. Create a certificate signing request (CSR). The CSR is used to request a client certificate and is saved in a file named client.csr. Also, create a private key for the client certificate. The private key is saved in a file named client.key.

```
openssl req -new -nodes -text -out client.csr -keyout client.key -subj "/CN=<Username tha
t is used for logons from the database client>"
```

> **Note** In the preceding command, the CN parameter follows the -subj parameter. You must set the CN parameter to the username of the account that is used by the database client to connect to the RDS instance.

3. Create a client certificate. The client certificate is saved in a file named client.crt.

```
openssl x509 -req -in client.csr -text -days 365  -CA ca1.crt -CAkey ca1.key -CAcreateser
ial  -out client.crt
```

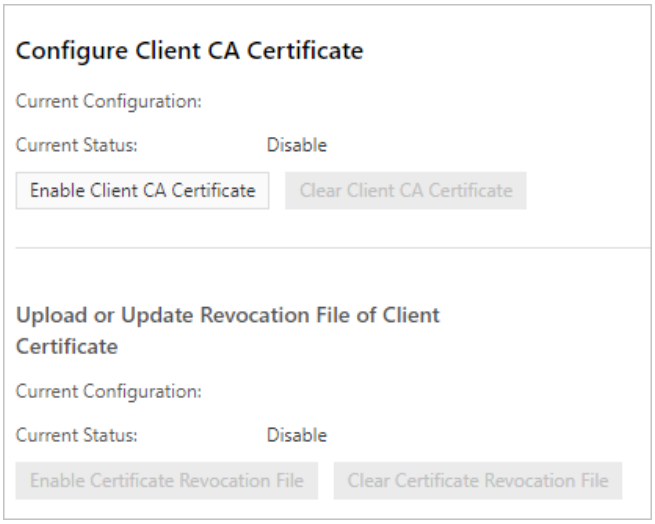
After you complete the preceding operations, you can obtain the following files:

- client.crt: the name of the file that contains the client certificate
- client.key: the file that contains the private key of the client certificate
- ca1.crt: the file that contains the self-signed certificate
- ca1.key: the file that contains the private key of the self-signed certificate

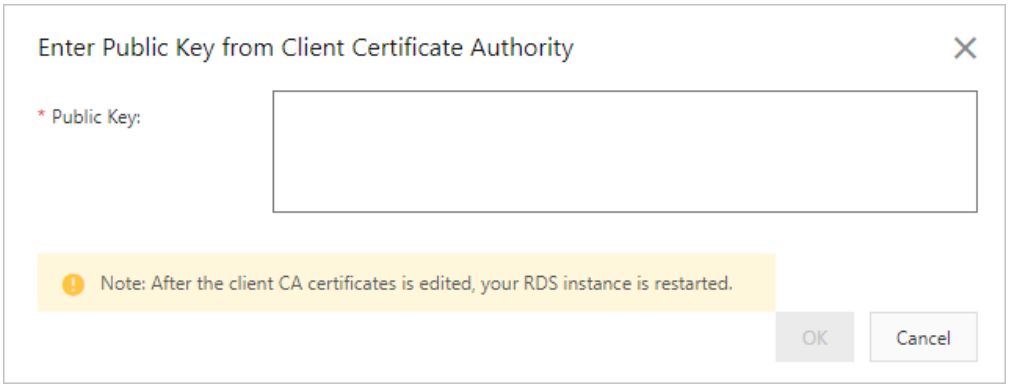
Step 2: Configure a client CA certificate

> **Note** After a client CA certificate is configured, the status of the RDS instance changes from **Running** to **Modifying SSL**. After about 3 minutes, the status changes back to **Running**.

1. Log on to the ApsaraDB RDS console. Find the RDS instance and click the ID of the instance. In the left-side navigation pane, click **Data Security**. On the page that appears, click the **SSL Encryption** tab.
2. Click **Enable Client CA Certificate**.



3. In the dialog box that appears, copy the content of the ca1.crt file to the Public Key field. Then, click **OK**. For more information about how to obtain the ca1.crt file, see the "Step 1: Create a client certificate" section of this topic.



Step 3: Connect to the RDS instance from the database client

You can connect to the RDS instance from the database client over SSL by using one of the following methods:

- Use pgAdmin to connect to an RDS instance over SSL
- Use PostgreSQL CLI to connect to an RDS instance over SSL
- Use JDBC to connect to an RDS instance over SSL

Step 4: (Optional) Configure a CRL file

If you no longer need the client certificate, you can revoke the client certificate. After the client certificate is revoked, the RDS instance denies access requests from the database client.

> **Note** After a CRL file is configured, the status of the RDS instance changes from **Running** to **Modifying SSL**. After about 3 minutes, the status changes back to **Running**.

1. Modify the openssl.cnf file.

```
touch /etc/pki/CA/index.txt
echo 1000 > /etc/pki/CA/crlnumber
```

> **Note**
>
> If you are using a Windows operating system, you must perform the following operations:
>
> i. Create a CA folder in the *Installation directory of OpenSSL\bin* path.
> ii. Create a file named *index.txt* in the CA folder.
> iii. Run the following command by using PostgreSQL CLI:
>
> ```
> echo 1000 > <Installation directory of OpenSSL>\bin\CA\crlnumber
> ```
>
> iv. Modify the *openssl.cnf* file in the *C:\Program Files\Common Files\SSL* path.
>
> ```
> # Find the [ CA_default ] configuration item.
> dir = "<Installation directory of OpenSSL>\\bin\\CA"
> ```

2. Revoke the client certificate, which is contained in the client.crt file.

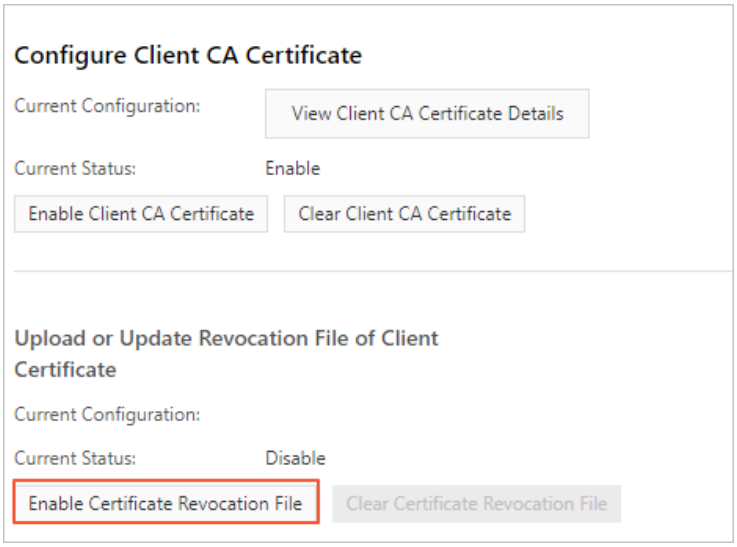
```
openssl ca -revoke client.crt -cert ca1.crt -keyfile ca1.key
```

> **Note** The preceding command requires the self-signed certificate and the private key of the self-signed certificate. The self-signed certificate is contained in the ca1.crt file, and the private key of the self-signed certificate is contained in the ca1.key file. For more information, see the "Step 1: Create a client certificate" section of this topic.

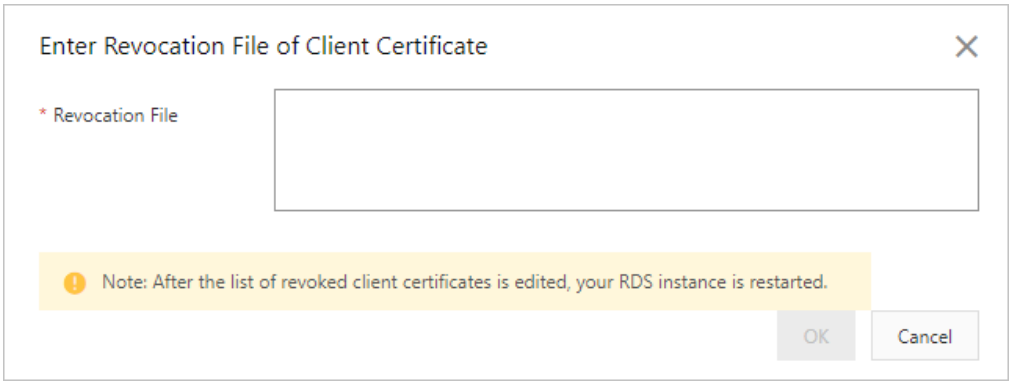
3. Create a CRL. The CRL is saved in a file named client.crl.

```
openssl ca -gencrl -out client.crl -cert ca.crt -keyfile ca.key
```

4. Log on to the ApsaraDB RDS console. Find the RDS instance and click the ID of the instance. In the left-side navigation pane, click **Data Security**. On the page that appears, click the **SSL Encryption** tab.
5. Click **Enable Certificate Revocation File**.



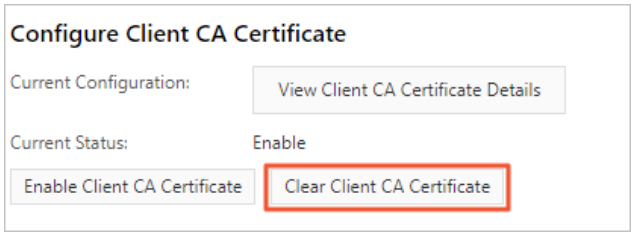
6. In the dialog box that appears, copy the content of the client.crl file to the Revocation File field.



Step 5: (Optional) Update the client certificate

> **Note** This operation triggers a restart of the RDS instance. Proceed with caution.

On the SSL Encryption tab of the Data Security page, click **Clear Client CA Certificate**. Then, click **Enable Client CA Certificate**.

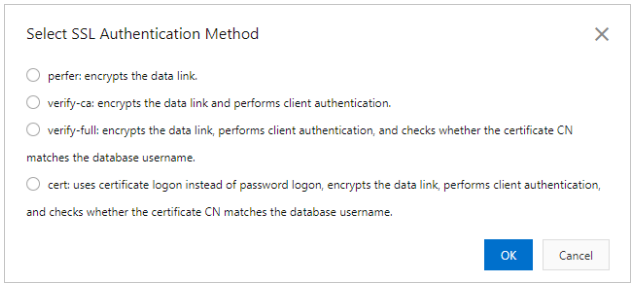


Step 6: (Optional) Configure an ACL

After a client CA certificate is configured, you can configure an access control list (ACL) on the RDS instance. Then, the database client can connect to the RDS instance only after the RDS instance validates the database client based on the SSL mode that you specify. The RDS instance validates the database client by using the client certificate and the private key of the client certificate.

> **Note**
>
> - When you configure an ACL, no operations can be performed on the RDS instance. This configuration process requires about 1 minute.
> - If you have not specified an SSL mode for the database client, the default SSL mode is used. The default SSL mode is prefer. In this case, you can set the `PGSSLMODE` parameter to disable. Then, you can connect to the RDS instance over SSL. If you want to prohibit non-SSL connections, you must specify an SSL mode for the configured ACL after you enable SSL encryption. The SSL mode that you specify cannot be prefer.

Click Modify next to Configure ACL or Configure Replication ACL. In the dialog box that appears, select an SSL mode.



ApsaraDB RDS for PostgreSQL supports the following SSL modes and validation rules:

- cert: A client certificate rather than a password is used to validate the database client. An SSL connection is established. In addition, the system validates the client certificate and checks whether the CN specified in the client certificate is consistent with the username that is used to connect to the RDS instance.
- prefer: An SSL connection is established. If you set the PGSSLMODE parameter on the database client to disable, you can connect the database client to the RDS instance over a non-SSL connection.
- verify-ca: An SSL connection is established, and the system validates the client certificate.
- verify-full: An SSL connection is established, and the system validates the client certificate and checks whether the CN specified in the client certificate is consistent with the username that is used to connect to the RDS instance. This SSL mode is supported only for PostgreSQL 12.

24.3.4. Connect to an ApsaraDB RDS for PostgreSQL instance over SSL

This topic describes how to connect to an ApsaraDB RDS for PostgreSQL instance from a database client over SSL. After you configure SSL encryption for an RDS instance, you can connect to the RDS instance from a database client by using pgAdmin, PostgreSQL CLI, or Java Database Connectivity (JDBC).

Prerequisites

- SSL encryption is enabled for the RDS instance. For more information about how to enable SSL encryption, see Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance or Configure a custom certificate on an ApsaraDB RDS for PostgreSQL instance.
- The following files are obtained:

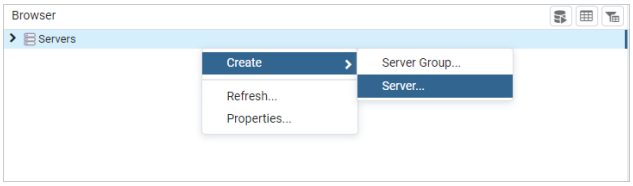
- The client.crt file that contains the client certificate and the client.key file that contains the private key of the client certificate: If you have configured a client certification authority (CA) certificate in the ApsaraDB RDS console, you must obtain these files. Otherwise, these files are optional. For more information, see Configure a client CA certificate on an ApsaraDB RDS for PostgreSQL instance.
- The file that contains the server CA certificate: For more information about how to obtain this file, see Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance or Configure a custom certificate on an ApsaraDB RDS for PostgreSQL instance.

Use pgAdmin to connect to an RDS instance over SSL

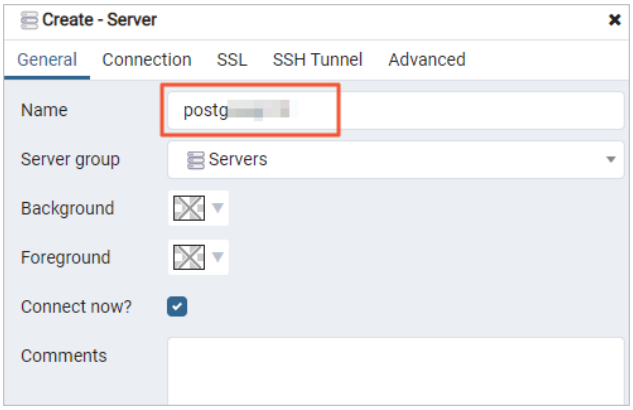
1. Start the pgAdmin 4 client.

   > **Note** If the pgAdmin client runs a later version and you log on the pgAdmin client for the first time, you must specify a master password that is used to protect the saved passwords and other credentials.

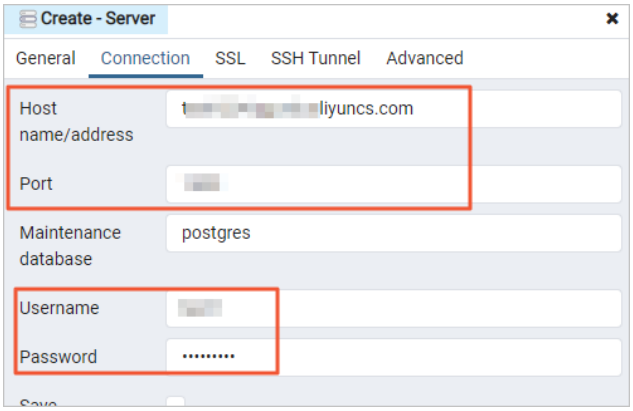
2. Right-click **Servers** and choose **Create > Server**.



3. On the **General** tab of the Create - Server dialog box, enter the name of the server where the pgAdmin client runs.

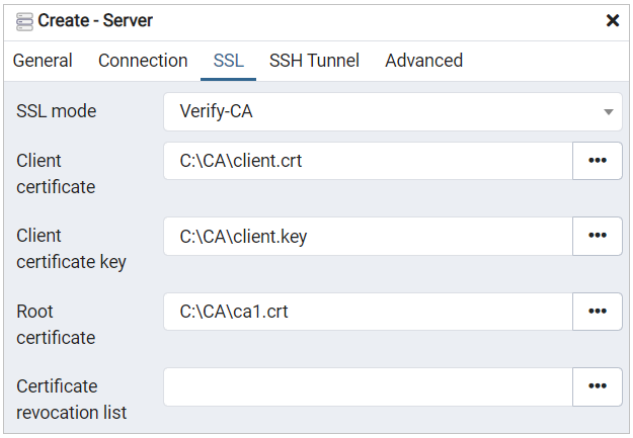


4. Click the **Connection** tab and enter the information that is used to connect to the RDS instance.



Parameter	Description
Hostname/address	Enter the endpoint of the RDS instance. If you want to connect to the RDS instance over an internal network, enter the internal endpoint of the RDS instance. If you want to connect to the RDS instance over the Internet, enter the public endpoint of the RDS instance. For more information, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for PostgreSQL instance.
Port	Enter the port number that is associated with the endpoint.
Username	Enter the username of the account that you use to log on to the RDS instance.
Password	Enter the password of the account that you use to log on to the RDS instance.

5. Click the SSL tab and configure the required parameters. The following table describes the parameters.

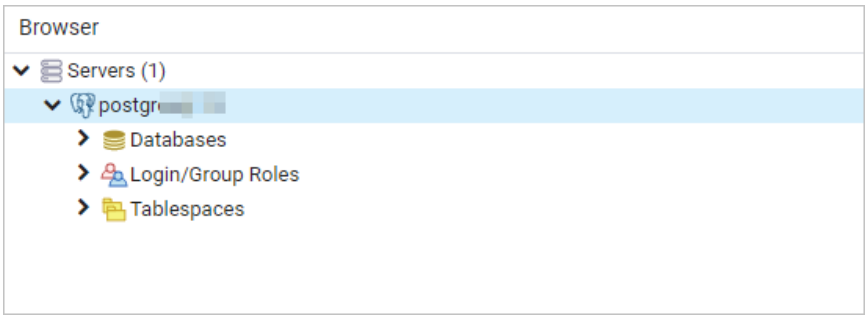


Parameter	Description

Parameter	Description
SSL mode	If SSL encryption is enabled for the RDS instance, the RDS instance allows SSL connections from the database client. You must set the **SSL mode** parameter based on the following scenarios: ○ No access control lists (ACLs) are configured for the RDS instance. For more information, see Configure a client CA certificate on an ApsaraDB RDS for PostgreSQL instance. ■ If you want to connect to the RDS instance from the database client by using SSL connections, set the **SSL mode** parameter to Require, Verify-CA, or Verify-Full. ■ If you do not want to connect to the RDS instance from the database client by using SSL connections, set the **SSL mode** parameter to Disable. ○ ACLs are configured for the RDS instance. In this case, you can connect to the RDS instance from the database client only by using SSL connections. For more information, see Configure a client CA certificate on an ApsaraDB RDS for PostgreSQL instance. Set the **SSL mode** parameter to Require, Verify-CA, or Verify-Full. The following list provides the meanings of the different values of the SSL mode parameter: ○ Require: The database client encrypts the SSL connections that are used to transmit data. However, the database client does not validate the RDS instance. ○ Verify-CA: The database client encrypts the SSL connections that are used to transmit data and validates the RDS instance. ○ Verify-Full: The database client encrypts the SSL connections that are used to transmit data, validates the RDS instance, and verifies that the CN or Domain Name System (DNS) specified in the server certificate is consistent with the endpoint that is configured at connection establishments.
Client certificate	You must set this parameter if you have configured a client certificate. This parameter specifies the save path of the file that contains the client certificate. The client certificate is contained in the client.crt file. For more information, see Configure a client CA certificate on an ApsaraDB RDS for PostgreSQL instance.
Client certificate key	You must set this parameter if you have configured a client certificate. This parameter specifies the save path of the file that contains the private key of the client certificate. The private key is contained in the client.key file. For more information, see Configure a client CA certificate on an ApsaraDB RDS for PostgreSQL instance.
Root certificate	You must set this parameter if you set the **SSL mode** parameter to **Verify-CA** or **Verify-Full**. This parameter specifies the save path of the file that contains the server CA certificate. **Note** In this example, the file that contains the server CA certificate is stored in the *C:\CA* path. You can download and decompress the file to a folder on your computer.

6. Click **Save**.
7. If the information that you entered is correct, the following page appears, which indicates that the connection to RDS instance is successful.

> **Note** The postgres database is the default system database of the RDS instance. Do not perform operations on this database.



Use PostgreSQL CLI to connect to an RDS instance over SSL

> **Note** This method uses the PostgreSQL CLI of PostgreSQL to connect to an RDS instance over SSL. Make sure that PostgreSQL is installed on your computer. For more information, see PostgreSQL documentation.

1. In the *var/lib/pgsql* path, create a folder named *.postgresql*.

```
mkdir /var/lib/pgsql/.postgresql
```

2. Copy the following files to the *.postgresql* folder:
   - The client.crt file that contains the client certificate and the client.key file that contains the private key of the client certificate: If you have configured a client CA certificate in the ApsaraDB RDS console, you must obtain these files. Otherwise, these files are optional. For more information, see Configure a client CA certificate on an ApsaraDB RDS for PostgreSQL instance.
   - The file that contains the server CA certificate: For more information about how to obtain this file, see Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance or Generate a custom certificate for an ApsaraDB RDS for PostgreSQL instance.

```
cp client.crt client.key ca1.crt /var/lib/pgsql/.postgresql/
```

> **Note** In this example, the client.crt file, client.key file, and ca1.crt file are stored in the current path that is opened.

3. Modify the permissions on the .postgresql folder.

```
chown postgres:postgres /var/lib/pgsql/.postgresql/*
chmod 600 /var/lib/pgsql/.postgresql/*
```

4. Run the following command to open the file that contains the environment variables:

```
vim /var/lib/pgsql/.bash_profile
```

5. Enter `i` to enable the edit mode. Then, add the following content to the file:

```
export PGSSLCERT="/var/lib/pgsql/.postgresql/client.crt"
export PGSSLKEY="/var/lib/pgsql/.postgresql/client.key"
export PGSSLROOTCERT="/var/lib/pgsql/.postgresql/ca1.crt"
```

6. Press `Esc` to exit the edit mode. Then, enter `:wq` to save the file and exit.
7. Reload the environment variables.

```
source .bash_profile
```

8. Specify the method that is used by the database client to validate the RDS instance.

```
export PGSSLMODE="verify-full"
```

If SSL encryption is enabled for the RDS instance, the RDS instance allows SSL connections from the database client. You must set the PGSSLMODE parameter based on your business requirements.

ACL configured	SSL connection required	Value of the PGSSLMODE parameter
No	Yes	require, verify-ca, or verify-full
	No	disable
Yes	The database client can connect to the RDS instance only over SSL.	require, verify-ca, or verify-full

> **Note**
>
> The following list provides the meanings of the different values of the PGSSLMODE parameter:
>
> - require: The database client encrypts the SSL connections that are used to transmit data. However, the database client does not validate the RDS instance.
> - verify-ca: The database client encrypts the SSL connections that are used to transmit data and validates the RDS instance.
> - verify-full: The database client encrypts the SSL connections that are used to transmit data, validates the RDS instance, and verifies that the CN or DNS specified in the server certificate is consistent with the endpoint that is configured at connection establishments.

9. Connect to the RDS instance.

```
psql -h <Endpoint> -U <Username> -p <Port number> -d <Database name>
```

The following table provides details about how to obtain the values of the preceding parameters from the ApsaraDB RDS console.

Parameter	Where to obtain
Endpoint	The endpoint that is protected by SSL encryption for the RDS instance. This endpoint is specified by the **Protected Host** parameter on the **SSL Encryption** tab of the Data Security page.
Username	The username that is used to connect to the RDS instance. You can obtain the username from the **Accounts** page.

Parameter	Where to obtain
Port number	The port number that is used to connect to the RDS instance. The default port number is 5432. If you have changed the default port number, you can obtain the new port number from the **Database Connection** page.
Database name	The name of the database that you want to connect on the RDS instance. You can obtain the name of the database from the **Databases** page. The postgres database is a default system database. Do not perform operations on the postgres database.

Use JDBC to connect to an RDS instance over SSL

1. Download the following files to your computer:
   - The client.crt file that contains the certificate file and the client.key file that contains the private key of the client certificate: If you have configured a client CA certificate in the ApsaraDB RDS console, you must obtain these files. Otherwise, these files are optional. For more information, see Configure a client CA certificate on an ApsaraDB RDS for PostgreSQL instance.
   - The file that contains the server CA certificate: For more information about how to obtain this file, see Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance or Generate a custom certificate for an ApsaraDB RDS for PostgreSQL instance.
2. Convert the client.key file to the PK8 format.

```
openssl pkcs8 -topk8 -inform PEM -in client.key -outform der -out client.pk8 -v1 PBE-MD5-
DES
# Enter the password that is used to connect to the RDS instance.
Enter Encryption Password:
Verifying - Enter Encryption Password:
```

3. In this example, the database client runs Maven. In this case, import the Maven dependencies of PostgreSQL into the pom.xml file.

```
<dependency>
   <groupId>org.postgresql</groupId>
   <artifactId>postgresql</artifactId>
   <version>42.2.10</version>
 </dependency>
```

4. Compile a code snippet that is used to establish a JDBC-based SSL connection to the RDS instance:

```
 // Specify the endpoint that is used to connect to the RDS instance.
 String hostname = "pgm-bpxxxxx.pg.rds.aliyuncs.com";
 // Specify the port number that is used to connect to the RDS instance.
 String port = "5432";
 // Specify the name of the database to which you want to connect on the RDS instance.
 String dbname = "postgres";
 String jdbcUrl = "jdbc:postgresql://" + hostname + ":" + port + "/" + dbname+"?binaryTra
nsfer=true";
 Properties properties = new Properties();
 //Specify the username that is used to connect to the specified database on the RDS inst
ance.
 properties.setProperty("user", "username");
 // Specify the password that is used to connect to the specified database on the RDS ins
tance.
 properties.setProperty("password", "*****");
 // Specify the save path of the file that contains the client certificate.
 String path= "D:\\ssl\\";
 // Configure SSL encryption.
 properties.setProperty("ssl", "true");
 // Specify the public key of the CA.
 properties.setProperty("sslrootcert", path + "/" + "root.crt");
 // Specify the private key of the client certificate.
 properties.setProperty("sslkey", path + "/" + "client.pk8");
 // Specify the client certificate.
 properties.setProperty("sslcert", path + "/" + "client.crt");
 // Enter the password that you specified when you converted the client.key file to the P
K8 format.
 properties.setProperty("sslpassword", "*****");
 // Specify the SSL mode, which can be set to require, verify-ca, or verify-full.
 properties.setProperty("sslmode", "verify-ca");
  try {
      Class.forName("org.postgresql.Driver");
      Connection connection = DriverManager.getConnection(jdbcUrl, properties);
      // In this example, the postgres database contains a table named example from which
data is queried.
      PreparedStatement preparedStatement = connection.prepareStatement("select * from "
+
              "example");
      ResultSet resultSet = preparedStatement.executeQuery();
      while (resultSet.next()) {
          ResultSetMetaData rsmd = resultSet.getMetaData();
          int columnCount = rsmd.getColumnCount();
          Map map = new HashMap();
          for (int i = 0; i < columnCount; i++) {
              map.put(rsmd.getColumnName(i + 1).toLowerCase(), resultSet.getObject(i + 1)
);
          }
          System.out.println(map);
      }
  } catch (Exception exception) {
      exception.printStackTrace();
  }
```

24.4. Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure disk encryption for an ApsaraDB RDS for PostgreSQL instance. Disk encryption ensures the security of your data.

Context

Disk encryption protects the data that is stored on standard SSDS or enhanced SSDs (ESSDs) and eliminates the need to modify your business or application. In addition, ApsaraDB RDS automatically applies disk encryption to both the snapshots that are generated from the encrypted standard SSDS or ESSDs and to the standard SSDs or ESSDs that are created from those snapshots.

Disk encryption is free of charge. You are not charged for the read and write operations that you perform on the encrypted standard SSDs or ESSDs.

Prerequisites

- A customer master key (CMK) that is used for disk encryption is created. For more information, see Procedure. You can enable disk encryption for your RDS instance only when you create the RDS instance.
- When you create an RDS instance, the parameters that specify the edition, storage type, and instance family are specified based on the following table.
  - Edition: High-availability Edition.
  - Storage type: Standard SSD or ESSD.
  - Instance family: Dedicated instance family.

Precautions

- You cannot disable disk encryption after you enable the feature.
- If you enable the disk encryption feature for your RDS instance, your RDS instance does not support cross-region backups. For more information, see Use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance.
- Disk encryption does not interrupt your business, and you do not need to modify your applications.
- If you enable disk encryption for your RDS instance, the snapshots that are created for the instance are automatically encrypted. If you use the encrypted snapshots to create an RDS instance that uses standard SSDs or ESSDs, the disk encryption feature is automatically enabled for the new RDS instance.
- If your Key Management Service (KMS) has overdue payments, the standard SSDs or ESSDs of your RDS instance become unavailable. Make sure that your KMS does not have overdue payments. For more information, see What is KMS?
- If you disable or delete the CMK that is used for disk encryption, your RDS instance cannot run as normal. For example, you cannot create snapshots, restore data from snapshots, or rebuild the secondary RDS instance of your RDS instance.

Procedure

1. Log on to the KMS console.
2. In the top navigation bar, select the region where you want to create an RDS instance.
3. Click **Create Key**.
4. Configure the following parameters.

Parameter	Description
KMS Instance	The KMS instance that you use.
Key Spec	The type of the CMK. Valid values: ○ Types of symmetric keys ■ **Aliyun_AES_256** ■ **Aliyun_SM4** ○ Types of asymmetric keys ■ **RSA_2048** ■ **RSA_3072** ■ **EC_P256** ■ **EC_P256K** ■ **EC_SM2** **Note** ○ Aliyun_SM4 and EC_SM2 types are supported only for regions in the Chinese mainland in which managed hardware security modules (HSMs) are used. ○ RSA_3072 is supported only by a dedicated KMS instance.
Purpose	The purpose of the CMK. Valid values: ○ **Encrypt/Decrypt**: encrypts or decrypts data. ○ **Sign/Verify**: generates or verifies a digital signature.
Alias Name	The alias of the CMK, which helps identify the CMK. Aliases are optional to CMKs. For more information, see Overview.
Protection Level	Valid values: ○ **Software**: The CMK is protected by using a software module. ○ **Hsm**: The CMK is managed in an HSM, and the HSM safeguards the CMK.
Description	The description of the CMK.

Parameter	Description
Rotation Period	The interval of automatic rotation of symmetric keys. Valid values: ○ 30 Days. ○ 90 Days. ○ 180 Days. ○ 365 Days. ○ Disable: Automatic rotation is disabled. ○ Customize: You can customize an interval that ranges from 7 days to 730 days. **Note** You can configure this parameter only if you set the Key Spec parameter to Aliyun_AES_256 or Aliyun_SM4.

5. Click **OK**.
6. On the Cloud Resource Access Authorization page, click **Confirm Authorization Policy**. Then, the RDS instance that you created can access your cloud resources. Authorization is required only the first time you enable disk encryption.

   Note You can log on to the RAM console to check whether you have the permissions of the **AliyunRDSInstanceEncryptionDefaultRole** RAM role.

7. Create an RDS instance. Select Disk Encryption when you create the instance. For more information, see Create an ApsaraDB RDS for PostgreSQL instance.

   Note After the RDS instance is created, you can go to the Basic Information page of the instance and view the CMK that is used for disk encryption.

24.5. Fully encrypted database

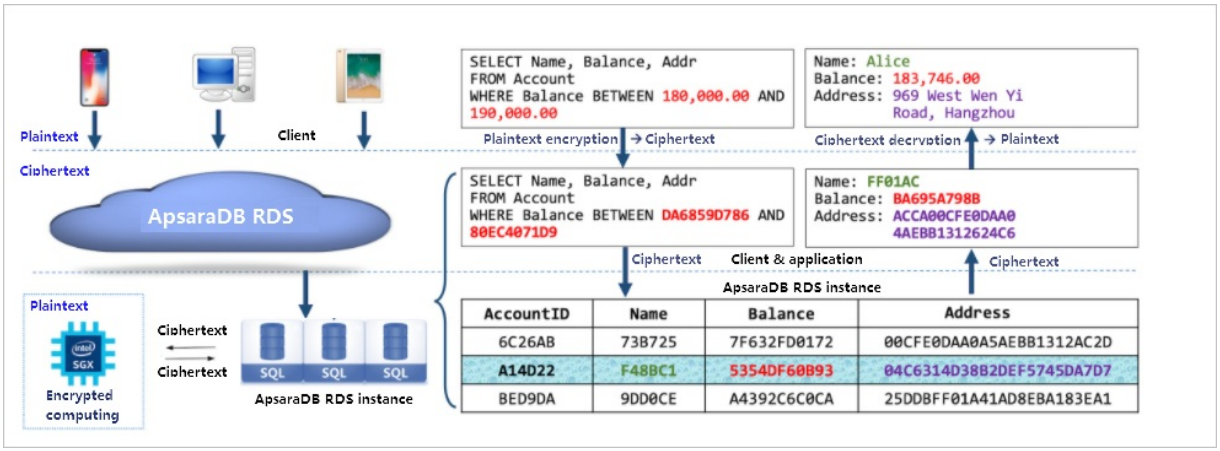
24.5.1. Overview

This topic introduces the fully encrypted database feature in ApsaraDB RDS for PostgreSQL. Before data is uploaded to a fully encrypted database, the data is encrypted on the client side. The fully encrypted database stores all data in ciphertext to ensure that only the owner of the data can access the data in plaintext. This way, data leaks caused by plaintext storage in the cloud are avoided.

Features

Data security depends on secure transmission, storage and usage. RDS databases support SSL encryption to secure the data during the transmission process. RDS databases also support disk encryption to secure the data stored in standard SSD or enhanced SSDs. For more information, see Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance and Configure disk encryption for an ApsaraDB RDS for PostgreSQL instance. The fully encrypted database feature allows data to be encrypted on the client side before it is uploaded to an RDS database. This ensures that internal roles, such as cloud platform software and database administrators, cannot access the data in plaintext. Fully encrypted databases are compatible with all the features of ApsaraDB RDS for PostgreSQL instances.

The following figure shows the architecture of a fully encrypted database.



Description

- You can specify encryption-related attributes for your sensitive data. Data is automatically encrypted on the client side. You do not need to modify the code on the server side.
- A fully encrypted database stores and processes all data in ciphertext. Ciphertext ensures that privilege account users and application developers cannot access the data in plaintext.

24.5.2. Configure the fully encrypted database feature on an ApsaraDB RDS for PostgreSQL instance

This topic describes how to configure the fully encrypted database feature on an ApsaraDB RDS for PostgreSQL instance.

Prerequisites

25.Log and event history management

25.1. Use the SQL Audit feature on an ApsaraDB RDS for PostgreSQL instance

This topic describes how to use the SQL Audit feature on an ApsaraDB RDS for PostgreSQL instance. This feature allows you to audit the executed SQL statements on a regular basis and view the details about the executed SQL statements. After you enable the SQL Audit feature, the performance of the RDS instance is not affected.

Precautions

- You cannot view the SQL statements that are executed before the SQL Audit feature is enabled.
- After you enable the SQL Audit feature, the performance of the RDS instance is not affected.
- The retention period of SQL audit logs is 30 days.
- The exported SQL audit log files can be retained for two days. After the retention period elapses, the system deletes the SQL audit log files.
- The maximum size per SQL statement is 8,000 bytes. If the size of an SQL statement exceeds 8,000 bytes, the excessive bytes cannot be logged.

Billing

- The SQL Audit feature is disabled by default. After you enable the SQL Audit feature, you are charged per hour for using this feature on the RDS instance.

  The fee charged per hour varies in different Alibaba Cloud regions:

  - USD 0.15/GB-hour: China (Hong Kong), US (Silicon Valley), and US (Virginia).
  - USD 0.18/GB-hour: Singapore (Singapore), Japan (Tokyo), Germany (Frankfurt), UAE (Dubai), Australia (Sydney), Malaysia (Kuala Lumpur), India (Mumbai), Indonesia (Jakarta), and UK (London).
  - USD 0.12/GB-hour: all regions except the preceding regions.

- After the SQL Explorer and Audit feature is enabled, ApsaraDB RDS stops billing the Use the SQL Explorer and Audit feature on an ApsaraDB RDS for PostgreSQL instance feature. The pricing of the SQL Explorer and Audit feature is based on the pricing of Database Autonomy Service (DAS) Professional Edition. For more information, see Pricing of DAS Professional Edition.

Enable the SQL Explorer and Audit feature

The SQL Explorer and Audit feature is supported for ApsaraDB RDS for PostgreSQL instances. The **SQL detail search** item is enhanced, and items such as **Full SQL analysis**, **SQL insight-comparison view**, and **SQL insight-source analysis** are added. You can view the item and billing differences between the SQL Audit feature and the SQL Explorer and Audit feature in the dialog box that appears when you click the **SQL Audit** tab. For more information, see Use the SQL Explorer and Audit feature on an ApsaraDB RDS for PostgreSQL instance.

1.
2. In the left-side navigation pane, click **Data Security**.

3. Click the **SQL Audit** tab. In the **SQL insight and audit** dialog box, click **One click upgrade**.

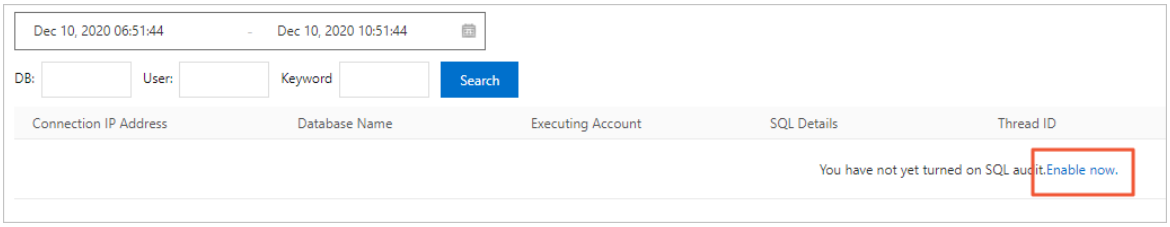
> **Note** When you enable this feature, auto-renewal is enabled for this feature by default to ensure the availability and security of your database service.
>
> - If you want to disable auto-renewal, you can choose **Expenses > Renewal Management** in the top navigation bar. For more information, see the "**Disable auto-renewal**" section in Enable auto-renewal for an ApsaraDB RDS for PostgreSQL instance.
> - If you want to configure billable items, you must enable DAS Professional Edition and connect DAS to your RDS instance. For more information, see Usage notes on DAS Professional Edition and Connect an Alibaba Cloud database instance to DAS.

Enable the SQL Audit feature

If you do not want to enable the **SQL Explorer and Audit** feature and want to use the **SQL Audit** feature, perform the following operations:

1.
2. In the left-side navigation pane, click **Data Security**.
3. Click the **SQL Audit** tab. In the **SQL insight and audit** dialog box, click **Cancel**.
4. Click **Enable now**.

> **Note** If you are using the original ApsaraDB RDS console, click **Enable SQL Audit**.



5. In the message that appears, click **OK**.

After the SQL Audit feature is enabled, you can query SQL statements based on the specified search criteria, such as the time range, databases, users, and keywords.

> **Note** The system collects SQL audit logs from databases and saves the SQL audit logs as CSV files to a log server. You can use the **rds_max_log_files** parameter to specify the maximum number of SQL audit log files that can be retained.

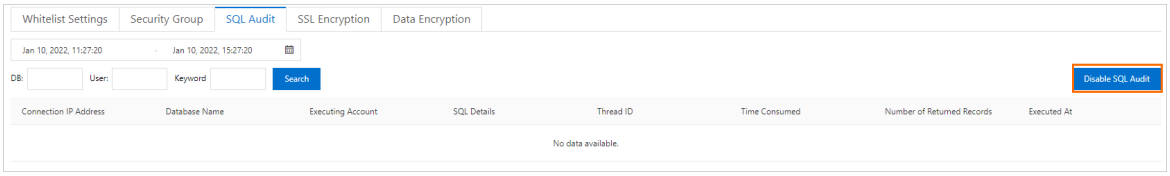
Disable SQL Audit

If you no longer use the SQL Audit feature, you can disable the feature to reduce costs.

> **Note** After the SQL Audit feature is disabled, all SQL audit logs including historical SQL audit logs are deleted. If the RDS instance runs RDS High-availability Edition, we recommend that you export SQL audit logs as files to your computer before you disable the SQL Audit feature. If the RDS instance runs RDS Basic Edition, you cannot export SQL audit logs as files.

1.
2. In the left-side navigation pane, click **Data Security**.

3. Click the **SQL Audit** tab. In the **SQL insight and audit** dialog box, click **Cancel**.
4. Click **Export File** to export and store the SQL audit logs as a file to your computer.
5. After the SQL audit logs are exported as a file to your computer, click **Disable SQL Audit**.



6. In the message that appears, click **OK**.

25.2. View logs

This topic describes how to query error logs, slow query logs, and primary/secondary switching logs of an ApsaraDB RDS for PostgreSQL instance by using the ApsaraDB for RDS console or SQL statements.

> **Note** For information about archive logs, see Back up an ApsaraDB RDS for PostgreSQL instance and Download the data backup files and log backup files of an ApsaraDB RDS for PostgreSQL instance.

Precautions

Primary/secondary switching logs are not available in the Basic Edition.

Procedure

1.
2. In the left-side navigation pane, click **Logs**.
3. On the **Logs** page, click the Error Logs, Slow Query Logs, or Primary/Secondary Switching Logs tab, select a time range, and click **Search**.

Item	Description
Error logs	Record database running errors that occurred in the last month.
Slow query logs	Record SQL statements that take more than one second to execute in the last month. Duplicated SQL statements are removed.
Primary/secondary switching logs	Record switchovers between the primary and secondary instances in the last month. Primary/secondary switching logs are available only for RDS instances that are not in the Basic Edition.

> **Note**
> - If your RDS instance resides in the China (Zhangjiakou-Beijing Winter Olympics) region, the system only retains error logs and slow query logs generated in the last nine days.
> - The system collects logs from databases and saves the CSV log files to a log server. If you want to adjust the maximum number of retained log files, configure the **rds_max_log_files** parameter for the RDS instance. If more log files are retained, you can query logs in a longer time range. However, the log files occupy larger storage space.

Related operations

Operation	Description
DescribeSlowLogs	Queries the list of slow query logs.
DescribeSlowLogRecords	Queries details about slow query logs.
DescribeErrorLogs	Queries error logs.
DescribeBinlogFiles	Queries binary logs.
DescribeSQLLogRecords	Queries details about SQL audit logs.
DescribeSQLLogFiles	Queries the list of SQL audit logs.

25.3. View the event history of an ApsaraDB RDS instance

This topic describes how to view the operation and maintenance (O&M) events that are performed by users and Alibaba Cloud on an ApsaraDB RDS for SQL Server instance. These events include instance creation and parameter reconfiguration.

Billing

The event history feature is free of charge in the public preview phase, but starts to be charged after the public preview phase ends.

Scenarios

- Track instance management operations.
- Audit the security of instance management operations.
- Audit the compliance of the instance management operations that are performed by Alibaba Cloud. This applies to security-demanding sectors, such as finance and government affairs.

View the event history feature

1. Log on to the RDS management console , in the let-side navigation pane, click **Event Center**, and then select a region above.
2. Click the **Historical Events** tab.

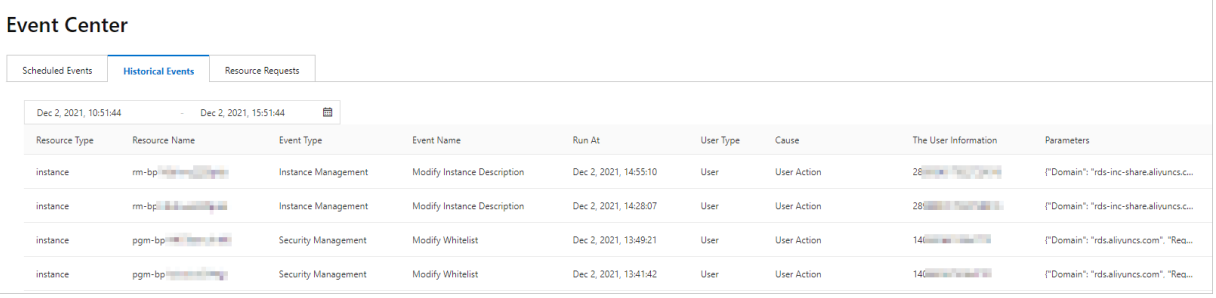
Introduction to the Historical Events page

The Historical Events page shows details about historical events in the selected region. These details include the resource types, resource names, and event types. The following table describes the parameters of a historical event.

Parameter	Description
Resource Type	The type of the RDS resource managed in the event. Only the **Instance** resource type is supported.

Parameter	Description
Resource Name	The name of the RDS resource managed in the event. If the value of the **Resource Type** parameter is **Instance**, the **Resource Name** column displays the ID of the involved RDS instance.
Event Type	The type of the event, for example, **Instance Management**, **Database Management**, **Read-write Splitting**, and **Network Management**. For more information, see Events.
Event Name	The operation executed in the event. For example, if the event type is **Instance Management**, supported operations include **Create Instance**, **Delete Instance**, **Change Specifications**, and **Restart Instance**. For more information, see Events.
Run At	The time when the event was executed.
User Type	The initiator of the event. Valid values: • User: initiates operations by using the ApsaraDB RDS console or the API. • System: initiates automatic O&M operations or periodic tasks. • O&M Administrator: initiates manual O&M operations.
Cause	The cause of the event. Valid values: • User Action: The event was initiated from a user by using the ApsaraDB RDS console or the API. • System Action or O&M Action: The event was initiated from the system or an O&M administrator.
The user information	The ID of the account that is used by a user to perform the event.
Parameters	The request parameters used by a user to initiate the event in the ApsaraDB RDS console.

> **Note**
> - The Historical Events page shows the historical events that were generated about 5 minutes earlier.
> - Historical Events are presented specific to regions. You can select a region in the top navigation bar and then view the historical events in the selected region.



Events

Event type	Operation
Instance Management	Restart Instance (RestartDBInstance)
	Renew (RenewInstance)
	Change Specifications (ModifyDBInstanceSpec)
	Migrate Across Zones (MigrateToOtherZone)
	Shrink Log (PurgeDBInstanceLog)
	Upgrade Kernel Version (UpgradeDBInstanceEngineVersion)
	Modify Instance Description (ModifyDBInstanceDescription)
	Modify Maintenance Window (ModifyDBInstanceMaintainTime)
	Create Read-only Instance (CreateReadOnlyDBInstance)
	Destroy Instance (DestroyDBInstance)
	Modify Upgrade Mode of Kernel Version (ModifyDBInstanceAutoUpgradeMinorVersion)
	Edit Parameters (ModifyParameter)
CloudDBA	Create Diagnostics Report (CreateDiagnosticReport)
Database Management	Create Database (CreateDatabase)
	Delete Database (DeleteDatabase)
	Modify Database Description (ModifyDBDescription)
	Replicate Database Between Instances (CopyDatabaseBetweenInstances)
	Modify System Collation and Time Zone (ModifyCollationTimeZone)
Read-write Splitting	Create Read-write Splitting Endpoint (AllocateReadWriteSplittingConnection)
	Query System-assigned Weight (CalculateDBInstanceWeight)
	Modify Read-write Splitting Policy (ModifyReadWriteSplittingConnection)
	Release Read-write Splitting Endpoint (ReleaseReadWriteSplittingConnection)
Security Management	Enable Enhanced Whitelist (MigrateSecurityIPMode)
	Enable SSL (ModifyDBInstanceSSL)
	Enable TDE (ModifyDBInstanceTDE)

Event type	Operation
	Modify Whitelist (ModifySecurityIps)
Account Management	Create Account (CreateAccount)
	Delete Account (DeleteAccount)
	Authorize Account to Access Database (GrantAccountPrivilege)
	Revoke Database Permissions from Account (RevokeAccountPrivilege)
	Modify Description of Database Account (ModifyAccountDescription)
	Reset Account Password (ResetAccountPassword)
	Reset Permissions of Superuser Account (ResetAccount)
High Availability (HA)	Trigger Switchover Between Primary and Secondary Instances (SwitchDBInstanceHA)
	Modify HA Mode (ModifyDBInstanceHAConfig)
Network Management	Apply for Public Endpoint (AllocateInstancePublicConnection)
	Modify Expiry Time of Endpoint (ModifyDBInstanceNetworkExpireTime)
	Modify Endpoint and Port (ModifyDBInstanceConnectionString)
	Switch Network Type (ModifyDBInstanceNetworkType)
	Release Public Endpoint (ReleaseInstancePublicConnection)
	Switch Between Internal and Public Endpoints (SwitchDBInstanceNetType)
Log Management	Enable/disable Log Audit (ModifySQLCollectorPolicy)
Backup Restoration	Create Data Backup (CreateBackup)
	Clone Instance (CloneDBInstance)
	Create Temporary Instance (CreateTempDBInstance)
	Modify Backup Policy (ModifyBackupPolicy)
	Restore Backup Set to Original Instance (RestoreDBInstance)
	Delete Data Backup (DeleteBackup)
	Restore Database (RecoveryDBInstance)
Cross-region Backup Restoration	Restore Data to New Instance Across Regions (CreateDdrInstance)
	Modify Cross-region Backup Settings (ModifyInstanceCrossBackupPolicy)

Event type	Operation
SQL Server Backup Migration to Cloud	Restore Backup File in OSS to RDS Instance (CreateMigrateTask)
	Make Database Available While Migrating Backup Data to Cloud (CreateOnlineDatabaseTask)
Monitoring	Set Monitoring Frequency (ModifyDBInstanceMonitor)
Data Migration	Create Upload Path for SQL Server (CreateUploadPathForSQLServer)
	Import Data from Other RDS (ImportDatabaseBetweenInstances)
	Cancel Migration Task (CancelImport)
Tag Management	Bind Tags to Instance (AddTagsToResource)
	Remove Tag (RemoveTagsFromResource)

Related operations

Operation	Description
Query historical events	Queries the events of an ApsaraDB RDS instance.
Query status of the event history feature	Queries the status of the historical events feature of an ApsaraDB RDS instance.
Enable or disable the event history feature	Enables or disables the historical events feature of an ApsaraDB RDS instance.

26.Manage pending events

If your ApsaraDB RDS instance has an event pending to be processed, the ApsaraDB RDS console notifies you of the event, so you can handle the event at your earliest opportunity.

You can receive text messages, voice messages, and emails that notify you of pending events such as instance migration and version upgrade events. In addition, after you log on to the ApsaraDB RDS console, you are prompted to manage the pending events. You can view the types, regions, processes, precautions, and affected instances of the pending events. You can also change the value of the Scheduled Disconnection Time parameter.

Prerequisites

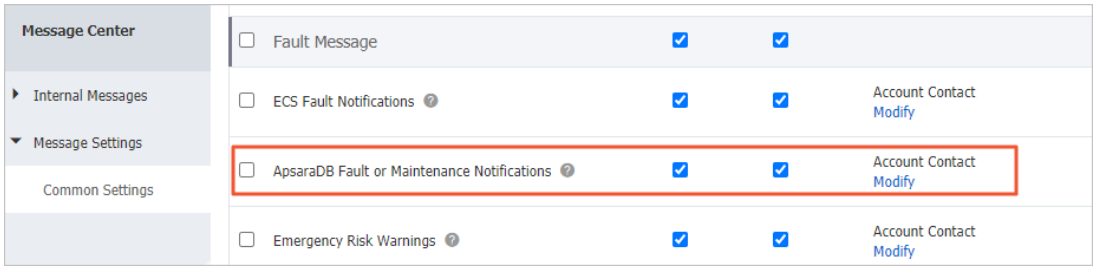
A pending event is found, which is an O&M event.

> **Note** If pending events are found, you can see notification badges on the **Pending Events** button in the upper-right corner of the ApsaraDB RDS homepage.

Precautions

You are notified of ApsaraDB for Redis pending events such as instance migrations or version upgrades at least three days before the events occur. Notifications for high-risk vulnerability fixes are sent three or fewer days before execution due to the urgency of these events. Event notifications are sent by usingphone calls, emails, internal messages, or the ApsaraDB for Redis console. To use this feature, log on to the Message Center console, enable **ApsaraDB Fault or Maintenance Notifications**, and then specify a contact. We recommend that you specify an O&M engineer as the contact.

Message Center settings



Procedure

1. Log on to the ApsaraDB RDS console.
2. Click **Events Center** in the left-side navigation pane or click **Pending Events** upper-right corner of the ApsaraDB RDS homepage,.

   > **Note** If a pending event requires you to schedule the time to handle the event, a message appears, which prompts you to schedule the time at your earliest opportunity.

3. On the **Pending Events** page, select the type and region of the event that you want to handle.

   > **Note** The content of the notification for an event varies based on the value of **Event Type**. The notification provides the process and precautions for the event.

4. View details about the event in the instance list. If you want to change the value of **Scheduled**

Disconnection Time, select an RDS instance and click **Specify Disconnection Time**. In the dialog box that appears, specify the time and click **OK**.

> **Note**
> - The information that is displayed for an event varies based on the type of the event.
> - The value of **Scheduled Disconnection Time** cannot be later than the time that is displayed in the **Set Before** column.

Causes and impacts of events

Cause	Impact type	Impact description
Instance migration Primary/secondary switchover SSL certificate update Backup mode change	Transient connections	From the time specified by the RDS instance is subject to the following impacts: • The RDS instance or its database shards experience transient connections and stay in the read-only state for up to 30 seconds until all data is synchronized. We recommend that you perform the operation during off-peak hours and make sure that your application is configured to automatically reconnect to your database system. • The RDS instance cannot work as expected for Data Management (DMS) or Data Transmission Service (DTS). After the operation is complete, the RDS instance is automatically recovered. Scheduled Disconnection Time
Minor engine version update	Transient connections	From the time specified by the RDS instance is subject to the following impacts: • The RDS instance or its database shards experience transient connections and stay in the read-only state for up to 30 seconds until all data is synchronized. We recommend that you perform the operation during off-peak hours and make sure that your application is configured to automatically reconnect to your database system. • The RDS instance cannot work as expected for Data Management (DMS) or Data Transmission Service (DTS). After the operation is complete, the RDS instance is automatically recovered.
	Differences between minor engine versions	Different minor engine versions provide different features. Before you update the minor engine version of the RDS instance, you must take note of the differences between the previous and new minor engine versions. For more information, see the release notes of minor engine versions. • ApsaraDB RDS: Release notes of minor AliSQL versions, Release notes of minor AliPG versions, and Release notes of minor ApsaraDB RDS for SQL Server versions. • Engine release notes, Release notes and Release notes.

Cause	Impact type	Impact description
Proxy version upgrade	Transient connections	From the time specified by the RDS instance is subject to the following impacts: • The RDS instance or its database shards experience transient connections and stay in the read-only state for up to 30 seconds until all data is synchronized. We recommend that you perform the operation during off-peak hours and make sure that your application is configured to automatically reconnect to your database system. • The RDS instance cannot work as expected for Data Management (DMS) or Data Transmission Service (DTS). After the operation is complete, the RDS instance is automatically recovered.
	Differences between proxy versions	Different proxy versions provide different features. Before you upgrade the proxy version of the RDS instance, you must take note of the differences between the previous and new proxy versions.
Network upgrade	Transient connections	From the time specified by the RDS instance is subject to the following impacts: • The RDS instance or its database shards experience transient connections and stay in the read-only state for up to 30 seconds until all data is synchronized. We recommend that you perform the operation during off-peak hours and make sure that your application is configured to automatically reconnect to your database system. • The RDS instance cannot work as expected for Data Management (DMS) or Data Transmission Service (DTS). After the operation is complete, the RDS instance is automatically recovered.
	VIP connection errors	Network upgrades may involve cross-zone data migration. In this case, the virtual IP address (VIP) of the RDS instance changes. If a database client uses a VIP to connect to the RDS instance, the connection is interrupted. **Note** We recommend that you use a domain name to connect to the RDS instance and disable the DNS cache of your application and the DNS cache of the server on which your application runs.

27.Backup

27.1. Backup fees for an ApsaraDB RDS for PostgreSQL instance

This topic describes the backup fees for an ApsaraDB RDS for PostgreSQL instance.

Overview

A free quota is provided for each RDS instance to store backup files. If the total size of the backup files of your RDS instance does not exceed the free quota, no fees are charged. If the total size exceeds the free quota, you are charged an hourly fee for the excess backup storage that you use. The hourly fee is calculated by using the following formula: **Hourly fee for backup storage = (Total size of backup files - Free quota) × Unit price**.

Backup method	Total size of backup files	Free quota (unit: GB; rounded only up to the next integer)	Unit price (USD/GB)
Snapshot backup	Total size of backup files = Size of data backup files + Size of log backup files You can log on to the ApsaraDB RDS console and go to the Basic Information page of your RDS instance. Then, you can view the total size of backup files in the lower-right corner of the page.	If your RDS instance uses standard SSDs or enhanced SSDs (ESSDs), the free quota is equal to **200%** of the storage capacity of your RDS instance.	0.00004
Physical backup		If your RDS instance uses local SSDs, the free quota is equal to **50%** of the storage capacity of your RDS instance.	0.00020

Usage notes

The backup fees are based on the total size of backup files. Backup files do not consume the storage capacity of your RDS instance. Therefore, the backup fees do not vary based on the storage usage.

When you analyze the backup fees, you must check the total size of backup files. You do not need to check the storage usage.

Methods of reducing backup usage

You can apply to increase the free quota.

You can expand the storage capacity of your RDS instance. For more information, see 变更配置.

The free quota varies based on the storage capacity of your RDS instance. For example, if your RDS instance uses local SSDs and you expand the storage capacity of your RDS instance from 150 GB to 300 GB, the free quota can be increased from 75 GB to 150 GB.

27.2. Back up an ApsaraDB RDS for PostgreSQL instance

This topic describes how to back up an ApsaraDB RDS for PostgreSQL instance. You can configure a backup policy based on which ApsaraDB RDS automatically backs up your RDS instance. If you do not configure a backup policy, the default backup policy is used. You can also manually back up your RDS instance.

Precautions

- Backup files occupy backup storage. Each RDS instance is allocated a free quota for backup storage. If the amount of backup storage that you use exceeds the free quota, you are charged for the excess backup storage that you use. We recommend that you specify a backup cycle based on your business requirements to maximize the usage of the free backup storage. For more information about the free quota for backup storage, see View the free quota for backup storage of an ApsaraDB RDS for PostgreSQL instance.
- We recommend that you familiarize yourself with the billing methods and billable items of backup storage. For more information, see Billable items, billing methods, and pricing.
- We recommend that you familiarize yourself with the billing standards of backup storage. For more information, see Backup fees for an ApsaraDB RDS for PostgreSQL instance.
- Do not execute DDL statements during a backup. These statements trigger locks on tables, and the backup may fail as a result of the locks.
- We recommend that you back up your RDS instance during off-peak hours.
- If your RDS instance has a large amount of data, a backup may require a long period of time.
- Backup files are retained for a specified period of time. Before the specified retention period elapses, we recommend that you download the backup files that you need to your computer.

Overview of data backups and log backups

Database engine	Data backup	Log backup

Database engine	Data backup	Log backup
PostgreSQL	Data backups are copies of the data of your RDS instance. These backups include physical backups and snapshot backups. You can use these backups to restore the data of your RDS instance. For more information, see Restore the data of an ApsaraDB RDS for PostgreSQL instance. Your RDS instance automatically performs physical backups or snapshot backups based on the type of storage media that you use: • If your RDS instance is equipped with local SSDs, full physical backups are supported. • If your RDS instance is equipped with standard SSDs or enhanced SSDs (ESSDs), snapshot backups are supported. You can restore the data of your RDS instance from snapshot backup files to a new RDS instance. You cannot download snapshot backup files. **Note** You can perform only physical backups in the ApsaraDB RDS console. However, you can export logical backup files to your computer. For more information, see Create a logical backup for an ApsaraDB RDS for PostgreSQL instance.	Log backups are copies of the archived binary log files of your RDS instance. **Note** If your RDS instance runs the RDS Basic Edition, log backups are not supported.

Configure a backup policy for an RDS instance

After you configure a backup policy for an RDS instance, ApsaraDB RDS automatically backs up the RDS instance based on the backup policy.

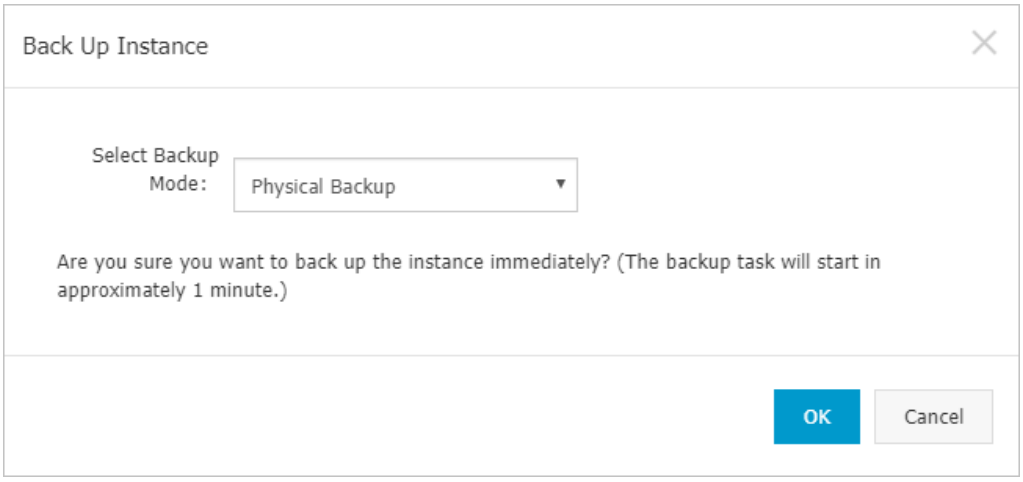
1.
2. In the left-side navigation pane, click **Backup and Restoration**.
3. On the **Backup and Restoration** page, click the **Backup Settings** tab. In the Data Backup Settings section of the tab that appears, click **Edit**.
4. Configure the parameters and click **Save**. The following table describes the parameters.

Parameter	Description
Data Backup Retention (Days)	The number of days for which you want to retain data backup files. Valid values: 7 to 730. Default value: 7. **Note** This parameter takes effect only for data backup files that are generated from regular backups. This parameter does not take effect for data backup files that are generated from **single-digit second backups**.

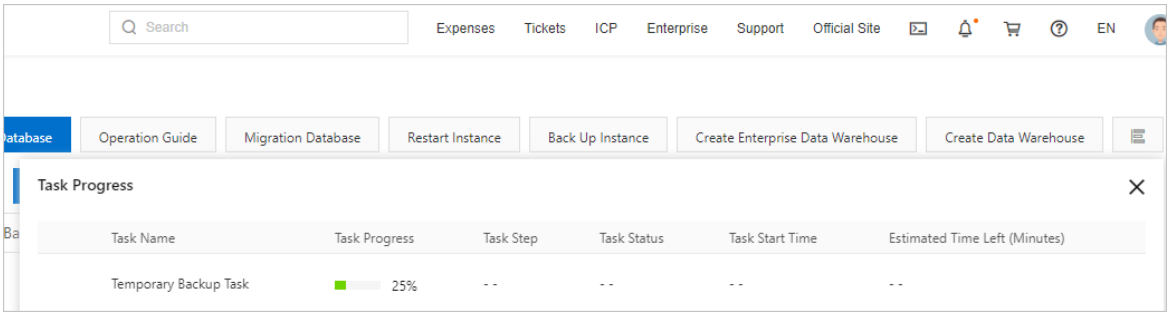
Parameter	Description
Backup Cycle	The cycle based on which you want to perform a backup. You can select one or more days of the week. **Note** For data security purposes, we recommend that you select at least two days of the week.
Single-digit Second Backup	The switch that is used to enable or disable the single-digit second backup feature. If you enable this feature, ApsaraDB RDS can complete each backup in 1 second. **Note** ◦ This feature is supported only for RDS instances that are equipped with ESSDs. ◦ If you enable this feature, ApsaraDB RDS performs a single-digit second backup and a regular backup on the data of the RDS instance based on the values of the **Backup Cycle** and **Backup Time** parameters. ◦ If you enable this feature, each manual backup is performed as a single-digit second backup. For more information, see Manually back up an RDS instance. ◦ The retention period of single-digit second backup files is fixed to seven days.
Backup Time	The hour at which you want to perform a backup.
Log Backup	The switch that is used to enable or disable the log backup feature. If you disable this feature, all log backup files are deleted and you cannot restore the data of the RDS instance to a specified point in time.
Log Retention Period (Days)	◦ The number of days for which you want to retain log backup files. Valid values: 7 to 730. Default value: 7. ◦ The log backup retention period must be shorter than or equal to the data backup retention period.

Manually back up an RDS instance

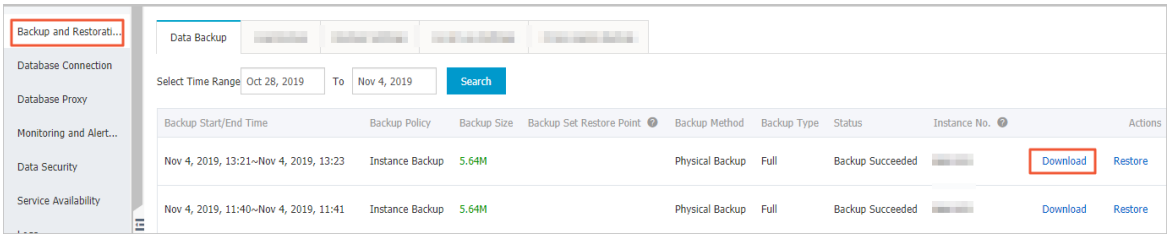
1.
2. In the upper-right corner of the page, click **Back Up Instance**. In the dialog box that appears, configure the parameters and click **OK**.



3. In the upper-right corner of the page, click the **Task Progress** icon to view the progress of the backup task. Wait until the backup task is completed.



> **Note** After the backup task is completed, you can go to the **Backup and Restoration** page to download the backup file that is generated. Some RDS instances do not support the download of backup files. These RDS instances include RDS instances that are equipped with standard SSDs or ESSDs. For more information, see Download the data backup files and log backup files of an ApsaraDB RDS for PostgreSQL instance.



FAQ

1. Can I disable the data backup feature for my RDS instance?

   No, you cannot disable the data backup feature for your RDS instance. However, you can reduce the data backup frequency to as low as twice a week.

2. Can I disable the log backup feature for my RDS instance?

   Yes, you can disable the log backup feature for your RDS instance. To disable the log backup feature, perform the following steps: Log on to the ApsaraDB RDS console, go to the Backup Settings tab, and modify the backup settings to disable the log backup feature.

Related operations

Operation	Description
Create data backup	Creates a data backup for an ApsaraDB RDS instance.
Query the data backup files	Queries the data backup files of an ApsaraDB RDS instance.
查询备份设置	Queries the backup settings of an ApsaraDB RDS instance.
Modify backup settings	Modifies the backup settings of an ApsaraDB for RDS instance.
Delete backup sets	Deletes the data backup files of an ApsaraDB for RDS instance.
Query backup tasks	Queries the backup tasks of an ApsaraDB for RDS instance.
Query log backup files	Queries the log backup files of an ApsaraDB RDS instance.

27.3. Use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance. After you enable this feature, the backup files of the original RDS instance can be replicated from the source region to the destination region. You can use the backup files in the destination region to manage and restore the data of the original RDS instance.

Context

You can use one of the following methods to enable this feature:

- ApsaraDB RDS console
- DBS

If a cross-region backup is complete, you can restore the data of the original RDS instance from the generated cross-region backup file to a new RDS instance that resides in the destination region. For more information, see Restore the data of an ApsaraDB RDS for PostgreSQL instance across regions.

> **Note**
>
> - For more information about the default backup feature, see Back up an ApsaraDB RDS for PostgreSQL instance.
> - For more information about how to enable the cross-region backup feature for an ApsaraDB RDS for MySQL instance, see Enable cross-region backups for an ApsaraDB RDS for MySQL instance.
> - For more information about how to enable the cross-region backup feature for an ApsaraDB RDS for SQL Server instance, see Enable cross-region backups for an ApsaraDB RDS for SQL Server instance.

Differences between cross-region backups and default backups

Item	Cross-region backup	Default backup
Default configuration	By default, cross-region backups are disabled. You must manually enable cross-region backups.	By default, default backups are enabled.
Storage	Cross-region backup files are stored in a region that is different from the region of the original RDS instance.	Default backup files are stored in the region where the original RDS instance resides.
Restoration	Data from cross-region backup files can be restored to the following RDS instances: • Original RDS instance • New RDS instance in the destination region • Existing RDS instance	Data from default backup files can be restored to the following RDS instances: • New RDS instance that resides in the same region as the original RDS instance • Original RDS instance
Retention period	After the original RDS instance is released, its cross-region backup files are still retained based on the cross-region backup retention period that you specify.	By default, after the original RDS instance is released, its default backup files are retained for seven days.

Prerequisites

Where to perform cross-region backups	Prerequisite
ApsaraDB RDS console	The original RDS instance meets the following requirements: • The original RDS instance runs PostgreSQL 9.4 or PostgreSQL 10 on RDS High-availability Edition with local SSDs. • The original RDS instance runs PostgreSQL 10, PostgreSQL 11, PostgreSQL 12, PostgreSQL 13, or PostgreSQL 14 with standard SSDs or enhanced SSDs (ESSDs). The disk encryption feature is disabled for the original RDS instance.

Where to perform cross-region backups	Prerequisite
Database Backup (DBS)	• DBS is activated, and a backup schedule is created. For more information, see Create a backup schedule. • The region that you specify on the DBS buy page is different from the region of the original RDS instance. • The logical backup method is used. • A public endpoint is allocated to the original RDS instance. For more information, see Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.

Notice The RDS instance does not use a new general-purpose instance type. The new general-purpose instance types provide better scalability and performance and reduce the time to create an RDS instance or change the specifications of an RDS instance. The new general-purpose instance types do not support the cross-region backup feature. For more information, see Primary ApsaraDB RDS for PostgreSQL instance types.

Billing

Where to perform cross-region backups	Billing
ApsaraDB RDS console	You are charged for the storage and traffic that are consumed by the cross-region backup files. • Remote storage fee: USD 0.0002 per GB-hour. • Network traffic fee: For more information, see Billing overview.
DBS	If you want to store cross-region backup files in the built-in storage of DBS, you are charged for the storage that you use in DBS. For more information, see Billing overview.

Precautions

Where to perform cross-region backups	Precaution

Where to perform cross-region backups	Precaution
ApsaraDB RDS console	• The cross-region backup feature is not supported by RDS instances for which disk encryption is enabled. • Cross-region backups do not affect default backups. These two types of backups exist at the same time. • After a default backup is complete, a cross-region backup is triggered. During the cross-region backup process, the original RDS instance dumps the generated default backup files to the destination region. • After you enable the cross-region backup feature, the original RDS instance checks whether valid data backup files are generated over the most recent 24 hours. If no valid data backup files are generated over the most recent 24 hours, the original RDS instance triggers a backup on its secondary RDS instance. • Cross-region backups are not supported in some Alibaba Cloud regions due to network reasons. For more information, see Alibaba Cloud regions in which cross-region backups are supported.
DBS	The cross-region backup feature is not supported by RDS instances for which disk encryption is enabled.

Alibaba Cloud regions in which the cross-region backup feature is supported

Source region	Destination region
China (Hangzhou), China (Shanghai), China (Qingdao), China (Beijing), China (Zhangjiakou), China (Hohhot), China (Shenzhen), China (Hong Kong), China (Ulanqab), China (Chengdu), China (Guangzhou), and China (Heyuan)	China (Hong Kong), China (Hangzhou), China (Shanghai), China (Qingdao), China (Shenzhen), China (Zhangjiakou), China (Hohhot), China (Beijing), China (Ulanqab), China (Chengdu), China (Guangzhou), and China (Heyuan) **Note** Cross-region backup files are replicated to a region that is different from the source region. This may vary based on your network environment.
US (Silicon Valley)	US (Virginia)
US (Virginia)	US (Silicon Valley)
China East 1 Finance	China East 2 Finance and China South 1 Finance
China East 2 Finance	China East 1 Finance and China South 1 Finance
China South 1 Finance	China East 1 Finance and China East 2 Finance

ApsaraDB RDS console

- **To enable the cross-region backup feature for a single RDS instance, perform the following operations:**
  i. Log on to the ApsaraDB RDS console. In the left-side navigation pane, click **Backups**. In the top navigation bar, select the region where the RDS instance resides.
  ii. Click the **Cross-region Backup** tab. On the tab that appears, click the **Pending Instances** tab.
  iii. Find the RDS instance for which you want to enable the cross-region backup feature and click **Settings** in the Cross-region Backup Settings column.
  iv. Configure the following parameters.

Parameter	Description
Cross-region Backup Status	Specify whether to enable or disable cross-region backups. Select **Enable**.
Backup Region	Select the destination region to which the backup files of the RDS instance are automatically replicated.
Cross-region Retention Period	Specify the retention period of cross-region backup files. Valid values: 7 to 1825. Unit: days. The longest cross-region backup retention period spans five years. **Note** After the RDS instance expires or is released, its cross-region backup files are still retained based on the cross-region backup retention period that you specify. You can log on to the ApsaraDB RDS console, click Backups in the left-side navigation pane, and then click the Cross-region Backup tab to view the cross-region backup files that are retained.
Cross-region Log Backup Status:	Specify whether to enable or disable cross-region log backups. After you enable cross-region log backups, the log backup files of the RDS instance are automatically replicated to a specified Object Storage Service (OSS) bucket in the destination region.

  v. Click **OK**.

- **To enable cross-region backups for multiple RDS instances at a time, perform the following operations:**
  i. Log on to the ApsaraDB RDS console. In the left-side navigation pane, click **Backups**. In the top navigation bar, select the region where the RDS instance resides.
  ii. Click the **Cross-region Backup** tab. On the tab that appears, click the **Pending Instances** tab.
  iii. Select the RDS instances for which you want to enable cross-region backups. Then, click **Backup Settings**.

  Note You can also click **Settings** in the Cross-region Backup Settings column of a single RDS instance to enable cross-region backups only for that RDS instance.

  iv. Configure the following parameters.

Parameter	Description
Cross-region Backup Status	Specify whether to enable or disable cross-region backups. Select **Enable**.
Backup Region	Select the destination region to which the backup files of the RDS instance are automatically replicated.
Cross-region Retention Period	Specify the retention period of cross-region backup files. Valid values: 7 to 1825. Unit: days. The longest cross-region backup retention period spans five years. **Note** After the RDS instance expires or is released, its cross-region backup files are still retained based on the cross-region backup retention period that you specify. You can log on to the ApsaraDB RDS console, click Backups in the left-side navigation pane, and then click the Cross-region Backup tab to view the cross-region backup files that are retained.
Cross-region Log Backup Status:	Specify whether to enable or disable cross-region log backups. After you enable cross-region log backups, the log backup files of the RDS instance are automatically replicated to a specified Object Storage Service (OSS) bucket in the destination region.

v. Click **OK**.

- **To modify the cross-region backup settings of an RDS instance, perform the following operations:**
  i. Log on to the ApsaraDB RDS console. In the left-side navigation pane, click **Backups**. In the top navigation bar, select the region where the RDS instance resides.
  ii. On the Backups page, click the **Cross-region Backup** tab. Click the **Backup Instances** tab and find the RDS instance whose cross-region backup settings you want to modify. Then, click **Settings** in the Cross-region Backup Settings column to modify the cross-region backup settings of the RDS instance.

  Note If the RDS instance is released, you can modify only the cross-region backup retention period.

- **To disable cross-region backups for an RDS instance, perform the following operations:**

  If you no longer require cross-region backups, you can disable cross-region backups.

  i. Log on to the ApsaraDB RDS console. In the left-side navigation pane, click **Backups**. In the top navigation bar, select the region where the RDS instance resides.
  ii. On the Backups page, click the **Cross-region Backup** tab. Click the **Backup Instances** tab and find the RDS instance for which you want to disable cross-region backups. Then, click **Settings** in the Cross-region Backup Settings column.
  iii. In the dialog box that appears, set the **Cross-region Backup Status** parameter to **Disabled** and set the **Cross-region Retention Period** parameter to **7**.

> **Note** After you disable cross-region backups, no new cross-region backup files are generated. However, the existing cross-region backup files are still retained for at least seven days. You can set the cross-region backup retention period to seven days. After the seven-day retention period that you specify elapses, all existing cross-region backup files are automatically deleted. Then, you are no longer charged for the storage of cross-region backup files.

iv. Click **OK**.

- **To download the cross-region data backup files of an RDS instance, perform the following operations:**

  After a cross-region backup is complete, you can download the generated cross-region backup files in the ApsaraDB RDS console.

  i. Log on to the ApsaraDB RDS console. In the left-side navigation pane, click **Backups**. In the top navigation bar, select the region where the RDS instance resides.

  ii. On the Backups page, click the **Cross-region Backup** tab. Click the **Backup Instances** tab and click the ID of the RDS instance for which you want to download cross-region backup files.

  iii. On the **Data Backup** tab or the **Log Backup** tabs, click **Download** in the Actions column to download the full data backup file or the incremental backup file.

  iv. Click **Download**.

> **Note** If you download data backup files over an internal network, the traffic is free of charge. If you download data backup files the Internet, the traffic is charged. For more information, see Network traffic fees.

DBS

In this example, the source region is China (Hangzhou), and the destination region is China (Beijing).

1.
2.
3. On the Backup Schedules page, find the ID of the backup schedule that you want to configure and click **Configure Backup Schedule** in the **Actions** column.
4. In the **Configure Backup Source and Destination** step of the Configure Backup Schedule wizard, configure the information about the backup source and backup destination and click Next.

   Configure Backup Schedule - Configure Backup Source and Destination

Section	Parameter	Description
N/A	**Schedule Name**	The name of the backup schedule. DBS generates a backup schedule name. We recommend that you set a descriptive name that is easy to identify. Backup schedule names do not have to be unique.
	Backup Mode	By default, the backup method that you selected when you purchased the backup schedule is used. In this example, select Logical Backup.

Section	Parameter	Description
Backup Source Information	**Database Location**	Select **RDS Instance**.
	Instance Region	The region where the original RDS instance resides. The backup schedule is used for cross-region backups. Therefore, select the region where the original RDS instance resides. In this example, select the China (Hangzhou) region.
	RDS Instance ID	The ID of the original RDS instance.
	Database Account	The username of the account that has the read permissions on specific databases of the original RDS instance in the ApsaraDB RDS console. For more information, see Account permissions.
	Password	Enter the password of the database account. After you enter the username and password of the database account, click **Test Connection** next to the password to check whether the information of the database that you want to back up is valid. If the specified parameters are valid, the Test Passed message appears. If the Test Failed message appears, click Check next to Test Failed. Modify the information about the instance that you want to back up based on the check results.
	SSL Encryption	The method that is used to encrypt the backup data. Valid values: ○ **Non-encrypted** ○ **SSL-encrypted**: SSL encrypts network connections at the transport layer to improve the security and integrity of data in transit. However, SSL increases the network connection response time. If you want to select **SSL-encrypted**, you must enable SSL encryption for the original RDS instance before you configure the backup schedule. For more information, see Configure SSL encryption for an ApsaraDB RDS for PostgreSQL instance.

Section	Parameter	Description
Backup Destination Information	Backup Storage Type	The type of storage that is used to store the backup data. Valid values: ○ **DBS Storage (recommended)**: Backup data is stored in DBS without requiring you to create storage space. You are charged based on the volume of your data that is stored in DBS. For more information about the billing method, see Storage fees. To reduce storage costs, we recommend that you use subscription storage plans. For more information, see Use storage plans. ○ **OSS For User**: You must create a bucket in the Object Storage Service (OSS) console in advance. For more information, see Create buckets. **Note** In this example, select **DBS built-in storage (recommended)**. If you select **OSS for user**, you must set the **OSS Bucket Name** parameter.
	Storage Encryption	The method that is used to encrypt the stored data. Valid values: ○ **Encrypted** (recommended): DBS uses AES-256, one of the advanced encryption standard ciphers, to encrypt the stored data. The server-side encryption feature is used in OSS. When you upload an object to OSS, OSS encrypts and stores the object. When you download the encrypted object from OSS, OSS decrypts the object and returns the decrypted object to you. For more information, see Server-side encryption. ○ **Non-encrypted**: The backup data is not encrypted.
	OSS Bucket Name	The name of the OSS bucket that you want to use to store data. This parameter is displayed only when you set the **Backup Storage Type** parameter to **OSS For User**.

5. In the **Edit Backup Objects** step, find the database or table that you want to back up and add it to the **Selected** section. Then, click **Next**.

   Note If you back up an entire database instance, additional database objects such as indexes and stored procedure are backed up. Each database supports different objects. For more information, see Database engines and features.

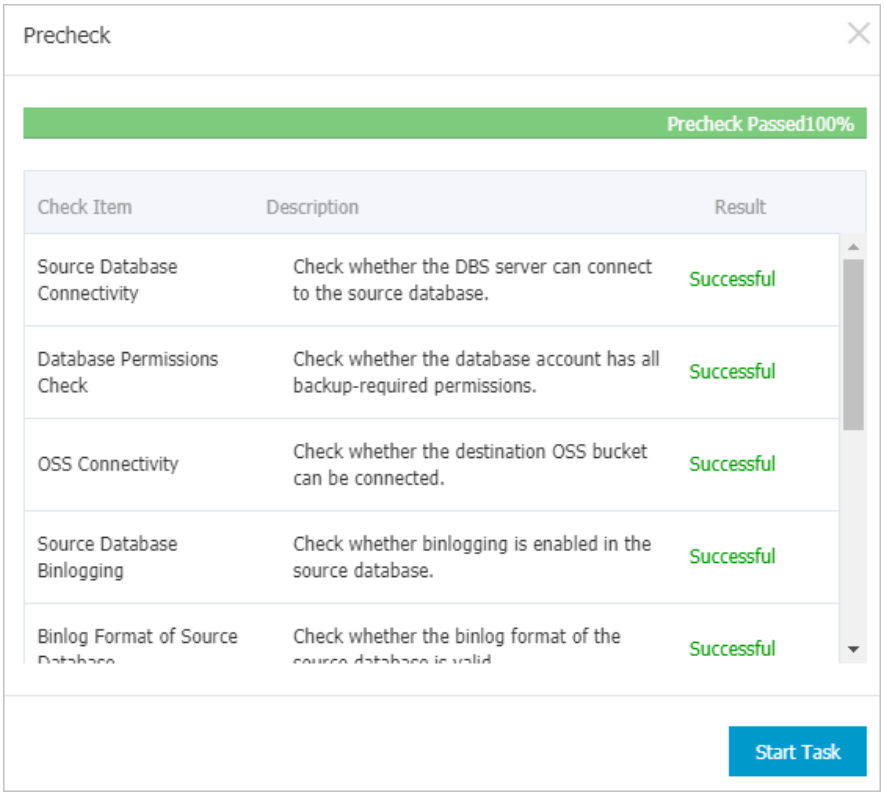
6. In the **Configure Backup Time** step, set the parameters that are described in the following table and click **Next** in the lower-right corner of the page.

Parameter	Description

Parameter	Description
Full-scale Backup Frequency	The frequency of the backup schedule. Select **Periodic Backup** or **Single Backup** as needed. **Note** If you set this parameter to **Periodic Backup**, you must set the **Full Data Backup Recurrence** and **Start At** parameters.
Full Data Backup Recurrence	The days of the week on which DBS runs the backup schedule. You can select one or more days of a week. Select at least one day of the week.
Start At	The start time of the backup, such as **01:00**. We recommend that you set a time during off-peak hours. **Note** If a previous full data backup is not finished at the start time of the next backup, DBS skips the next backup.
Incremental backup	Specifies whether to enable incremental backup. This parameter is displayed only when you set the **Full-scale Backup Frequency** parameter to **Periodic Backup**.
Maximum Concurrent Threads for Full Data Backup	The maximum number of concurrent threads available for a full backup. You can set this parameter to adjust the backup speed. For example, reduce the number of backup threads to minimize the impact on the database. The maximum number of actual concurrent threads varies based on backup schedule specifications. For more information, see How do I change the maximum backup speed?.
Backup network speed limit	The limit on the network bandwidth. You can specify the limit based on your business requirements. Default value: `0`, which indicates that the network bandwidth is not limited.

7. In the **Edit Lifecycle** step, configure the lifecycle for full backup data in the Configure Full Data Backup Lifecycle section.

   If you set the **Incremental real-time backup** parameter to Enable in Step 6, you must configure the lifecycle for incremental backup data. For more information about the lifecycle rules of backup data, see How do I manage the lifecycle rules of backup sets?.
8. After the configurations are complete, click **Precheck** in the lower-right corner of the page.
9. If the Precheck Passed message appears, click **Start Task**.



> **Note** When the **status** of the backup schedule changes to **Running**, the backup schedule takes effect.

After a backup is complete, you can view the backup schedule or restore the backup schedule. For more information, see View backup schedules and Restore the data of an ApsaraDB RDS for PostgreSQL instance across regions.

To download the cross-region data backup files of an RDS instance, perform the following operations:

1. Log on to the DBS console.
2. In the left-side navigation pane, click **Backup Schedules**. In the upper-left corner of the Backup Schedules page, select the region where you purchase the backup schedule. In this example, select the China (Beijing) region.
3. Click the Schedule Name and go to the detail page.
4. Multiple methods are provided to download data backup files on the detail page. For more information, see Overview.

FAQ

After I disable the cross-region backup feature for my RDS instance, why am I still charged for the storage of cross-region backup files?

After you disable the cross-region backup feature for your RDS instance, no new cross-region backup files are generated and you are no longer charged for the traffic that is consumed to transmit cross-region backup files. However, you are still charged for the storage of the existing cross-region backup files within the cross-region backup retention period that you specify. The existing cross-region backup files are retained for at least seven days. You can set the cross-region backup retention period to seven days. After the cross-region backup retention period that you specify elapses, all existing cross-region backup files are automatically deleted and you are no longer charged for the storage of cross-region backup files.

Related API operations

Operation	Description
Check cross-region backup	Checks whether an ApsaraDB RDS instance has a cross-region data backup file that can be used to restore data across regions.
Restore data to a new instance across regions	Restores the data of an ApsaraDB RDS instance to a new RDS instance that resides in a different region than the source region.
Modify cross-region backup settings	Modifies the cross-region backup settings of an ApsaraDB RDS instance.
Query cross-region backup settings	Queries the cross-region backup settings of an ApsaraDB RDS instance.
Query cross-region data backup files	Queries the cross-region data backup files of an ApsaraDB RDS instance.
Query cross-region log backup files	Queries the cross-region log backup files of an ApsaraDB RDS instance.
Query regions that support cross-region backup	Queries the available destination regions to which the cross-region backup files from a specified source region can be stored.
Query the time range to which you can restore data by using a cross-region backup set	Queries the restorable time range that is supported by a specified cross-region backup file.
Query ApsaraDB for RDS instances on which cross-region backup is enabled	Queries the ApsaraDB RDS instances for which cross-region backups are enabled in a specified region and the cross-region backup settings of these instances.

27.4. View the free quota for backup storage of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to view the free quota for backup storage of an ApsaraDB RDS for PostgreSQL instance and how to calculate the amount of excess backup storage that you use. The free quota varies based on the instance configuration.

Context

Backup files occupy backup storage. Each RDS instance is allocated a free quota for backup storage. If the total size of backup files exceeds the free quota, you are charged additional fees.

Formula

RDS instances that are equipped with standard SSDs or enhanced SSDs (ESSDs) support only snapshot backups. The free quota that is provided to store snapshot backup files is calculated by using the following formula: Free quota = 200% × Purchased storage capacity. Unit: GB. The calculation result is rounded only up to the next integer.

RDS instances that are equipped with local SSDs support only physical backups. The free quota that is provided to store physical backup files is calculated by using the following formula: Free quota = 50% × Purchased storage capacity. Unit: GB. The calculation result is rounded only up to the next integer.

The amount of excess backup storage for which you must pay an hourly rate is calculated by using the following formula: Excess backup storage = Size of data backup files + Size of log backup files - Free quota.

For example, your RDS instance is equipped with local SSDs. If the size of data backup files is 30 GB, the size of log backup files is 10 GB, and the purchased storage capacity is 60 GB, the amount of excess backup storage for which you must pay an hourly rate is 10 GB based on the following calculation: `Excess backup storage = 30 + 10 - 50% × 60 = 10 (GB)`. This indicates that you must pay for 10 GB of excess backup storage per hour.

> **Note**
>
> - When you use the RDS Basic Edition, some database engines support a seven-day retention period during which you can retain backup files for free. For more information, visit the ApsaraDB RDS console.

Procedure

1.
2. In the Usage Statistics section of the page that appears, view the free quota that is displayed to the right of the Backup Size parameter.

   > **Note** The free quota for backup storage varies based on the instance configuration.

FAQ

- Do backup files occupy the storage that I purchased when I created my RDS instance?

  No, backup files do not occupy the storage that you purchased when you created your RDS instance. The storage that you purchased when you created your RDS instance is isolated from the storage that is provided to store backup files.

- Can I purchase backup storage based on the subscription billing method?

  No, you cannot purchase backup storage based on the subscription billing method.

27.5. Download the data backup files and log backup files of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to download the unencrypted data backup files and log backup files of an ApsaraDB RDS for PostgreSQL instance. You can archive the backup files that you download. You can also use the backup files that you download to restore the data of the RDS instance to an on-premises database.

Precautions

- If your RDS instance uses standard SSDs or enhanced SSDs (ESSDs), you cannot download the snapshot backup files of the instance.
- A RAM user that has only the read permissions on your RDS instance is not authorized to download the backup files of the instance. You can grant the required permissions to a RAM user in the Resource Access Management (RAM) console.

Procedure

1.
2. In the left-side navigation pane, click **Backup and Restoration**.
3. Click the Data Backup or Archived Logs tab.
   - If you want to download a data backup file, click the **Data Backup** tab.
   - If you want to download a log backup file, click the **Archived Logs** tab.
4.
5. Find the data backup file or log backup file that you want to download. Then, click **Download** in the **Actions** column.

   > Note
   > - If the **Download** button cannot be found, you must check the storage type of the RDS instance. Snapshot backup files cannot be downloaded for RDS instances that use standard SSDs or ESSDs
   > - If you want to download a data backup file and use it to restore data, we recommend that you select the data backup file that is generated at the point in time closest to the point in time to which you want to restore data.
   > - If you want to download an archived log backup file and use it to restore data to an on-premises database, you must take note of the following information:
   >   - The instance No. of the log backup file must be the same as the instance No. of the data backup file that is used together with the log backup file to restore data.
   >   - The start time of the log backup file must be later than the end time of the data backup file that is used together with the log backup file to restore data. The start time of the log backup file must also be earlier than the point in time to which you want to restore data.

6.

27.6. Create a logical backup for an ApsaraDB RDS for PostgreSQL instance

This topic describes how to use pg_dump to create a logical backup for an ApsaraDB RDS for PostgreSQL instance and export the backup file to your computer.

Context

The pg_dump utility provided with PostgreSQL is used to back up individual databases. For more information, visit pg_dump.

This topic uses an ApsaraDB RDS for PostgreSQL instance that runs CentOS 7 and PostgreSQL 10.

Prerequisites

- The IP address of your ECS instance or on-premises host is added to the whitelist of an ApsaraDB RDS for PostgreSQL instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
- Your ECS instance or on-premises host runs the same version of PostgreSQL as the instance.

Precautions

We recommend that you use the privileged account of the ApsaraDB RDS for PostgreSQL instance to ensure that you have all the required permissions.

Back up a database

1. Log on to your ECS instance or on-premises host. Then, run the following command to back up a database from the ApsaraDB RDS for PostgreSQL instance:

```
pg_dump -h '<hostname>' -U <username> -p <port> -Fc <dbname> > <dumpdir>
```

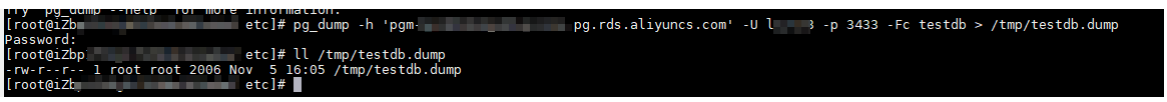
Parameter	Description

Parameter	Description
hostname	The endpoint used to connect to the ApsaraDB RDS for PostgreSQL instance. **Note** ◦ If your ECS instance connects to the ApsaraDB RDS for PostgreSQL instance by using an internal endpoint, you must make sure that the ECS instance and the RDS instance have the same network type. If both instances use the VPC network type, you must also make sure that they reside in the same VPC. For more information, see View the internal and public endpoints of an instance. ◦ If your on-premises host or ECS instance connects to the ApsaraDB RDS for PostgreSQL instance by using a public endpoint, you must make sure that the public endpoint has been allocated to the RDS instance. For more information, see View the internal and public endpoints of an instance and Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.
username	The username of the privileged account of the ApsaraDB RDS for PostgreSQL instance.
port	The port used to connect to the ApsaraDB RDS for PostgreSQL instance.
-Fc	The output file format. `-Fc` specifies to use the custom format, which is ideal when you use pg_restore to import logical backup files and restore databases. For more information, visit pg_dump.
dbname	The name of the database you want to back up.
dumpdir	The directory and name of the logical backup file to export.

Example

```
pg_dump -h 'pgm-bpxxxxxx.pg.rds.aliyuncs.com' -U test123 -p 3433 -Fc testdb > /tmp/testdb
.dump
```

2. When `Password:` appears, enter the password of the privileged account of the ApsaraDB RDS for PostgreSQL instance and press the Enter key.



Back up one or more tables

1. Log on to your ECS instance or on-premises host. Then, run the following command to back up one or more tables from a database in the ApsaraDB RDS for PostgreSQL instance:

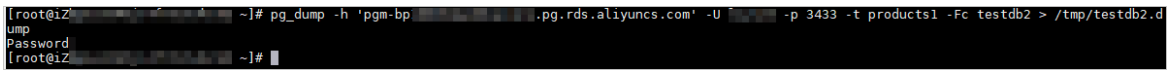
```
pg_dump -h '<hostname>' -U <username> -p <port> -t <table> -Fc <dbname> > <dumpdir>
```

Parameter	Description
hostname	The endpoint used to connect to the ApsaraDB RDS for PostgreSQL instance. **Note** ◦ If your ECS instance connects to the ApsaraDB RDS for PostgreSQL instance by using an internal endpoint, you must make sure that the ECS instance and the RDS instance have the same network type. If both instances use the VPC network type, you must also make sure that they reside in the same VPC. For more information, see View the internal and public endpoints of an instance. ◦ If your on-premises host or ECS instance connects to the ApsaraDB RDS for PostgreSQL instance by using a public endpoint, you must make sure that the public endpoint has been allocated to the RDS instance. For more information, see View the internal and public endpoints of an instance and Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.
username	The username of the privileged account of the ApsaraDB RDS for PostgreSQL instance.
port	The port used to connect to the ApsaraDB RDS for PostgreSQL instance.
table	The name of the table you want to back up. You can use `-t <table>` to specify more than one table.
-Fc	The output file format. `-Fc` specifies to use the custom format, which is ideal when you use pg_restore to import logical backup files and restore databases. For more information, visit pg_dump.
dbname	The name of the database you want to back up.
dumpdir	The directory and name of the logical backup file to export.

Example

```
pg_dump -h 'pgm-bpxxxxxx.pg.rds.aliyuncs.com' -U test123 -p 3433 -t products1 -Fc testdb2
> /tmp/testdb2.dump
```

2. When `Password:` appears, enter the password of the privileged account of the ApsaraDB RDS for PostgreSQL instance and press the Enter key.



Back up a database with one or more tables excluded

1. Log on to your ECS instance or on-premises host. Then, run the following command to back up a database from the ApsaraDB RDS for PostgreSQL instance with one or more tables excluded:

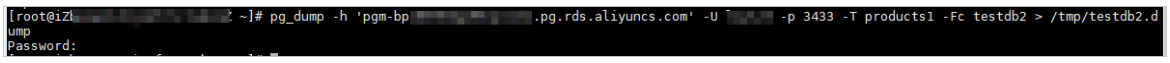
```
pg_dump -h '<hostname>' -U <username> -p <port> -T <table> -Fc <dbname> > <dumpdir>
```

Parameter	Description
hostname	The endpoint used to connect to the ApsaraDB RDS for PostgreSQL instance. **Note** ◦ If your ECS instance connects to the ApsaraDB RDS for PostgreSQL instance by using an internal endpoint, you must make sure that the ECS instance and the RDS instance have the same network type. If both instances use the VPC network type, you must also make sure that they reside in the same VPC. For more information, see View the internal and public endpoints of an instance. ◦ If your on-premises host or ECS instance connects to the ApsaraDB RDS for PostgreSQL instance by using a public endpoint, you must make sure that the public endpoint has been allocated to the RDS instance. For more information, see View the internal and public endpoints of an instance and Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.
username	The username of the privileged account of the ApsaraDB RDS for PostgreSQL instance.
port	The port used to connect to the ApsaraDB RDS for PostgreSQL instance.
table	The name of the table you want to exclude. You can use `-T <table>` to specify more than one table.
-Fc	The output file format. `-Fc` specifies to use the custom format, which is ideal when you use pg_restore to import logical backup files and restore databases. For more information, see pg_dump.
dbname	The name of the database you want to back up.
dumpdir	The directory and name of the logical backup file to export.

Example

```
pg_dump -h 'pgm-bpxxxxx.pg.rds.aliyuncs.com' -U test123 -p 3433 -T products1 -Fc testdb2
> /tmp/testdb2.dump
```

2. When `Password:` appears, enter the password of the privileged account of the ApsaraDB RDS for PostgreSQL instance and press the Enter key.



Back up the schema of a database with data excluded

1. Log on to your ECS instance or on-premises host. Then, run the following command to back up the

schema of a database from the ApsaraDB RDS for PostgreSQL instance.

```
pg_dump -h '<hostname>' -U <username> -p <port> -s -Fc <dbname> > <dumpdir>
```

Parameter	Description
hostname	The endpoint used to connect to the ApsaraDB RDS for PostgreSQL instance. **Note** ◦ If your ECS instance connects to the ApsaraDB RDS for PostgreSQL instance by using an internal endpoint, you must make sure that the ECS instance and the RDS instance have the same network type. If both instances use the VPC network type, you must also make sure that they reside in the same VPC. For more information, see View the internal and public endpoints of an instance. ◦ If your on-premises host or ECS instance connects to the ApsaraDB RDS for PostgreSQL instance by using a public endpoint, you must make sure that the public endpoint has been allocated to the RDS instance. For more information, see View the internal and public endpoints of an instance and Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.
username	The username of the privileged account of the ApsaraDB RDS for PostgreSQL instance.
port	The port used to connect to the ApsaraDB RDS for PostgreSQL instance.
-s	Specifies to only back up the schema of the database. The data of the database is not backed up. For more information, see pg_dump.
-Fc	The output file format. `-Fc` specifies to use the custom format, which is ideal when you use pg_restore to import logical backup files and restore databases. For more information, see pg_dump.
dbname	The name of the database you want to back up.
dumpdir	The directory and name of the logical backup file to export.

Example

```
pg_dump -h 'pgm-bpxxxxx.pg.rds.aliyuncs.com' -U test123 -p 3433 -s -Fc testdb2 > /tmp/tes
tdb2.dump
```

2. When `Password:` appears, enter the password of the privileged account of the ApsaraDB RDS for PostgreSQL instance and press the Enter key.

References

If you need to restore data due to a database exception, see Restore data from a logical backup file.

27.7. Create a full backup of an ApsaraDB RDS for PostgreSQL instance

This topic describes how to use the pg_basebackup utility provided by open source PostgreSQL to create a full backup of your ApsaraDB RDS for PostgreSQL instance and export the backup files to your computer.

Prerequisites

- The IP address of your ECS instance or local host is added to a whitelist of your RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
- Your ECS instance or local host runs the same version of PostgreSQL as your RDS instance.

Context

pg_basebackup backs up all data of a PostgreSQL instance. The backup files can be used for point-in-time recovery. For more information, see pg_basebackup.

In this topic, CentOS 7 and PostgreSQL 12 are used as examples of how to create a full backup.

Precautions

We recommend that you use the privileged account of your RDS instance or an account that has the REPLICATION permission. Otherwise, you may have permission issues during creation.

Procedure

> **Note** pg_basebackup cannot back up a single database or database object. For more information about how to back up a single database or database object, see Create a logical backup for an ApsaraDB RDS for PostgreSQL instance.

1. Log on to your ECS instance or local host. Then, run the following command to back up a database from your RDS instance:

   ```
   pg_basebackup -Ft -Pv -Xs -z -D <backupdir> -Z5 -h '<hostname>' -p <port> -U <username> -W
   ```

   The following table describes parameters in this command. For more information, see pg_basebackup.

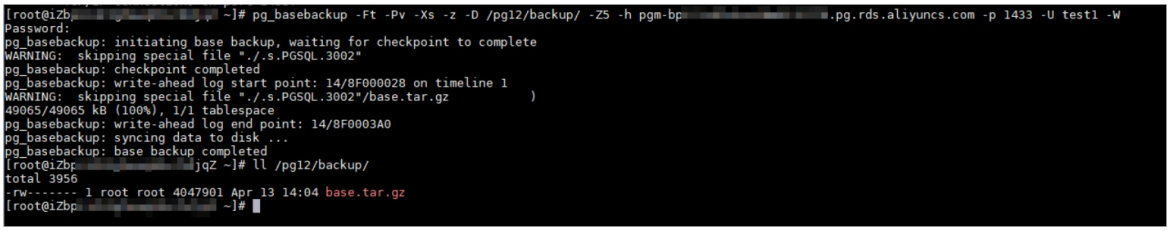
Parameter	Description
backupdir	The directory of backup files that are exported. The system automatically creates this directory. However, if this directory already exists and is not empty, the system reports an error.

Parameter	Description
hostname	The endpoint that you use to connect to your RDS instance. If your ECS and RDS instances are deployed in the same region under the same Alibaba Cloud account and have the same network type, we recommend that you use an internal endpoint. If both network types are VPC, the two instances must be in the same VPC. Use a public endpoint in other scenarios. **Note** If you want to use a public endpoint to access your RDS instance, make sure that you have applied for the public endpoint. For more information, see Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.
port	The port that you use to connect to your RDS instance.
username	A username of your RDS instance.

Example:

```
pg_basebackup -Ft -Pv -Xs -z -D /pg12/backup1/ -Z5 -h pgm-bpxxxxx.pg.rds.aliyuncs.com -p
1433 -U test1 -W
```

2. When `Password:` appears, enter the password of the username of your RDS instance and press Enter.



References

If you want to use the backup files to restore data, see Recovering Using a Continuous Archive Backup.

28.Restoration

28.1. Restore the data of an ApsaraDB RDS for PostgreSQL instance

The topic describes how to restore the data of an ApsaraDB RDS for PostgreSQL instance to a new RDS instance.

Prerequisites

The original RDS instance whose data you want to restore must meet the following requirements:

- The original RDS instance is in the Running state and is not locked.
- The original RDS instance does not have ongoing migration tasks.
- If you want to restore data to a point in time, the log backup feature is enabled for the original RDS instance.

  > **Note** RDS instances that run the Basic Edition do not support the log backup feature. Therefore, you cannot restore data of such an RDS instance to a point in time.

- If you want to restore data from a backup set, the original RDS instance has at least one backup set.

Context

ApsaraDB RDS for PostgreSQL allows you to restore data from a backup set or to a point in time. The following procedure is used to restore data:

1. Restore data to a new RDS instance. This process was formerly known as instance cloning.
2. Log on to the new RDS instance and verify the data.
3. Migrate the data to the original RDS instance.

Precautions

- The new RDS instance and the original RDS instance must have the same IP address whitelist, backup, and parameter settings.
- The data and account information of the new RDS instance must be the same as the data and account information that is indicated by the specified data or log backup file of the original RDS instance.

Billing

The fee required to renew the subscription of an RDS instance is the same as the fee required to purchase an RDS instance.

The price of the RDS instance that you want to purchase varies based on the instance configuration, such as the region, instance type, storage capacity that you select. For more information, go to the ApsaraDB RDS buy page.

Restore data to a new RDS instance

1.
2. In the left-side navigation pane, click **Backup and Restoration**.

3. In the upper-right corner of the page, click **Restore Database (Previously Clone Database)**.
4. Configure the following parameters.

Parameter	Description
Billing Method	○ **Subscription**: A subscription instance is an instance that you can subscribe to for a specified period and pay for up front. Subscription billing is more cost-effective than pay-as-you-go billing. Therefore, we recommend that you select subscription billing with a longer commitment. You can receive larger discounts for longer subscription periods. ○ **Pay-As-You-Go**: A pay-as-you-go instance is charged per hour based on your actual resource usage. We recommend that you select pay-as-you-go billing for short-term use. If you no longer need your pay-as-you-go instance, you can release it to reduce costs.
Restore Mode	○ **By Time**: allows you to restore data to a point in time within the specified log backup retention period. For more information about how to view or change the log backup retention period, see Back up an ApsaraDB RDS for PostgreSQL instance. ○ **By Backup Set**: allows you to restore data from a specified backup set. **Note** The **By Time** option appears only when the log backup feature is enabled.
Edition	○ **Basic**: The database system consists of only one RDS instance. Computing is separated from storage to increase cost-effectiveness. ○ **High-availability**: The database system adopts the classic high-availability architecture that consists of one primary instance and one secondary instance. **Note** The available RDS editions vary based on the selected region and database engine version. For more information, see Overview of ApsaraDB RDS editions
Zone of Primary Node and Zone of Secondary Node	

Parameter	Description
Instance Type	○ **Entry-level**: belongs to the general-purpose instance family. A general-purpose instance exclusively occupies the allocated memory and I/O resources. However, it shares CPU and storage resources with the other general-purpose instances that are deployed on the same server. ○ **Dedicated Instance (Enterprise-level)**: belongs to the dedicated instance family or the dedicated host instance family. A dedicated instance exclusively occupies the allocated CPU, memory, storage, and I/O resources. The top configuration of the dedicated instance family is the dedicated host instance family. A dedicated host instance exclusively occupies all of the CPU, memory, storage, and I/O resources on the server where it is deployed. ○ **Dedicated**: A dedicated cluster exclusively occupies all the resources on a VM or physical host. The permissions to manage hosts in a dedicated cluster can be authorized to you. This allows you to create multiple database instances on a host. For more information, see Add hosts **Note** Each instance type supports a specific number of CPU cores, memory capacity, maximum number of connections, and maximum IOPS. For more information, see Primary instance types
Capacity	The storage capacity that the read-only RDS instance has available to store data files, system files, binary log files, and transaction files. The storage capacity increases in increments of 5 GB. **Note** The dedicated instance family supports exclusive allocations of resources. Therefore, the storage capacity of each instance type with local SSDs in this family is fixed. For more information, see Primary instance types

5. Click **Next: Instance Configuration**.
6. Configure the following parameters.

Parameter	Description
Network Type	○ **Classic Network**: the traditional type of network. ○ **VPC**: the recommended type of network. A virtual private cloud (VPC) is an isolated virtual network that provides higher security and better performance than the classic network. If you select the VPC network type, you must also specify the **VPC** parameter and the **vSwitch of Primary Node** parameter. If you select the **Multi-zone Deployment** method in the **Basic Configurations** step, you must specify both the **VSwitch of Primary Node** and **VSwitch of Secondary Node** parameters. **Note** The RDS instance must have the same network type as the ECS instance that you want to connect. If the RDS and ECS instances both have the VPC network type, these instances must also reside in the same VPC. Otherwise, these instances cannot communicate over an internal network.
Resource Group	The resource group to which the new RDS instance belongs.

Parameter	Description

7. Click **Next: Confirm Order**.
8. Confirm the settings in the **Parameters** section, specify the **Purchase Plan** and **Duration** parameters, read and select Terms of Service, and then click **Pay Now**. You must specify the Duration parameter only when the new RDS instance uses the subscription billing method.

Log on to the new RDS instance and verify the data

For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance.

Migrate data to the original RDS instance

After you verify the data on the new RDS instance, you can migrate the data from the new RDS instance back to the original RDS instance. For more information, see Migrate data between RDS instances.

> **Note** The migration does not interrupt the workloads on the original RDS instance.

28.2. Restore the data of an ApsaraDB RDS for PostgreSQL instance across regions

This topic describes how to restore the data of an ApsaraDB RDS for PostgreSQL instance from a cross-region backup file to the same RDS instance or to a new RDS instance. The new RDS instance must reside in the region where the cross-region backup file is stored.

Prerequisites

The cross-region backup feature is enabled. For more information, see Use the cross-region backup feature for an ApsaraDB RDS for PostgreSQL instance.

> **Note**
> - For more information about how to restore the data of an ApsaraDB RDS for MySQL instance across regions, see Restore the data of an ApsaraDB RDS for MySQL instance across regions.
> - For more information about how to restore the data of an ApsaraDB RDS for SQL Server instance cross regions, see Restore the data of an ApsaraDB RDS for SQL Server instance across regions.

Procedure

1. Log on to the RDS management console , in the left-side navigation pane, click **Backups** , and then select a region above.
2. On the **Backup Instances** tab of the **Cross-region Backup** tab, find your RDS instance and click the ID of the instance. On the page that appears, find the backup file that you want to use, and click **Restore** in the **Actions** column.
3. On the **Restore Database** page, click the **Subscription** or **Pay-As-You-Go** tab and configure the

following parameters.

Parameter	Description
Restore Mode	○ **By Backup Set**: allows you to restore the data of your RDS instance from a data backup file. ○ **By Time**: allows you to restore the data of your RDS instance to a specific point in time. The point in time must be within the specified log backup retention period.
Backup Set	The data backup file from which you want to restore the data of your RDS instance. This parameter appears only when you set the **Restore Mode** parameter to **By Backup Set**.
Restore Point	The point in time to which you want to restore the data of your RDS instance. This parameter appears only when you set the **Restore Mode** parameter to **By Time**. **Note** Both local and cross-region log backup files can be used to restore the data of your RDS instance to a specific point in time.
Region	The region to which the new RDS instance belongs.
Zone	The zone where the new RDS instance resides. Each zone is an independent physical location within a region. Zones in the same region provide the same services. You can create the new RDS instance in the same zone as the Elastic Compute Service (ECS) instance to which you want to connect. You can also create the new RDS instance in a different zone than the ECS instance to which you want to connect.
CPU and Memory	The specifications of the new RDS instance. Each instance type supports a specific number of CPU cores, memory capacity, maximum number of connections, and maximum input/output operations per second (IOPS). For more information, see Primary ApsaraDB RDS instance types.
Capacity	The storage capacity that the new RDS instance has available to store data files, system files, archived log files, and transaction files.
Network Type	○ **Classic Network**: the traditional type of network. ○ **VPC**: the recommended type of network. A virtual private cloud (VPC) is an isolated virtual network that provides higher security and higher performance than the classic network. If you select the VPC network type, you must also select a vSwitch that is associated with the specified VPC.

Note The settings of some parameters cannot be modified. These parameters include Database Engine, Version, and Edition. The same settings of these parameters must be specified for both your RDS instance and the new RDS instance.

4. Specify the **Duration** and **Quantity** parameters. Then, click **Buy Now**. You must specify the Duration parameter when the new RDS instance is billed on a subscription basis.
5. On the **Order Confirmation** page, read and select Terms of Service, Service Level Agreement, and

Terms of Use. Then, click Pay Now and complete the payment.

References

After you create an RDS instance, you must configure IP address whitelists or security groups and create accounts. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance and Create an account on an ApsaraDB RDS for PostgreSQL instance. If you want to connect to the RDS instance over the Internet, you must also apply for a public endpoint. For more information, see Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance. After you complete these operations, you can connect to the RDS instance. For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance.

28.3. Restore data from a logical backup file

This topic describes how to restore data from a logical backup file to an RDS PostgreSQL instance or to an on-premises PostgreSQL database.

Context

A logical backup file is used to restore a small volume of data, the data of a table for example. For a large volume of data, we recommend that you restore it from a full physical backup file to a new RDS instance and then use Alibaba Cloud Data Transmission Service (DTS) to migrate the data to the original RDS instance.

Prerequisites

A logical backup is created for your RDS instance. For more information, see Create a logical backup for an ApsaraDB RDS for PostgreSQL instance.

Precautions

- We recommend that you do not restore data to the default postgres database.
- When you restore the data of a specific table, the system does not try to restore the database objects on which the table depends. Therefore, the restoration to a clean database may fail.

Restore the data of a database

1. Log on to the ECS instance or on-premises host that houses the logical backup file and run the following command to restore the data of a database:

```
pg_restore -h '<hostname>' -U <username> -p <port> -d <dbname> <dumpdir>
```

Parameter	Description

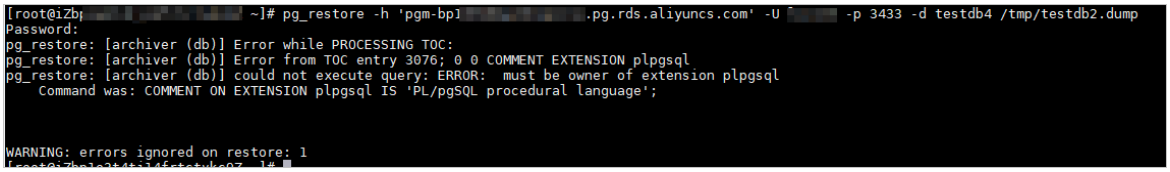
Parameter	Description
hostname	The endpoint used to connect to your RDS instance. **Note** ◦ If the ECS instance connects to your RDS instance by using an internal endpoint, you must make sure that the ECS and RDS instances have the same network type. If both instances use the VPC network type, you must also make sure that they reside in the same VPC. ◦ If the on-premises host or ECS instance connects to your RDS instance by using a public endpoint, you must make sure that a public endpoint has been allocated to your RDS instance. For more information, see Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.
username	The username of the privileged account of your RDS instance.
port	The port used to connect to your RDS instance.
dbname	The name of the database whose data you want to restore.
dumpdir	The directory and name of the logical backup file to use.

Example:

```
pg_restore -h 'pgm-bpxxxxx.pg.rds.aliyuncs.com' -U test123 -p 3433 -d testdb2 /tmp/testdb
.dump
```

2. When `Password:` appears, enter the password of the privileged account of your RDS instance and press Enter.

Note You can ignore the alerts generated by the embedded plpgsql plug-in.



Restore the data of a table

1. Log on to the ECS instance or on-premises host that houses the logical backup file and run the

following command to restore the data of a table:

```
pg_restore -h '<hostname>' -U <username> -p <port> -d <dbname> -t <table> -c <dumpdir>
```

Parameter	Description
hostname	The endpoint used to connect to your RDS instance. **Note** ◦ If the ECS instance connects to your RDS instance by using an internal endpoint, you must make sure that the ECS and RDS instances have the same network type. If both instances use the VPC network type, you must also make sure that they reside in the same VPC. ◦ If the on-premises host or ECS instance connects to your RDS instance by using a public endpoint, you must make sure that a public endpoint has been allocated to your RDS instance. For more information, see Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.
username	The username of the privileged account of your RDS instance.
port	The port used to connect to your RDS instance.
dbname	The name of the database that houses the table whose data you want to restore.
table	The name of the table whose data you want to restore.
-c	`-c`: specifies to delete the database objects on which the table depends before data restoration. For more information, visit pg_restore.
dumpdir	The directory and name of the logical backup file to use.

Example:

```
pg_restore -h 'pgm-bpxxxxx.pg.rds.aliyuncs.com' -U test123 -p 3433 -d testdb2 -t products
-c /tmp/testdb.dump
```

2. When `Password:` appears, enter the password of the privileged account of your RDS instance and press Enter.

Restore the schema of a database with data excluded

1. Log on to the ECS instance or on-premises host that houses the logical backup file and run the following command to only restore the schema of a database:

```
pg_restore -h '<hostname>' -U <username> -p <port> -d <dbname> -s <dumpdir>
```

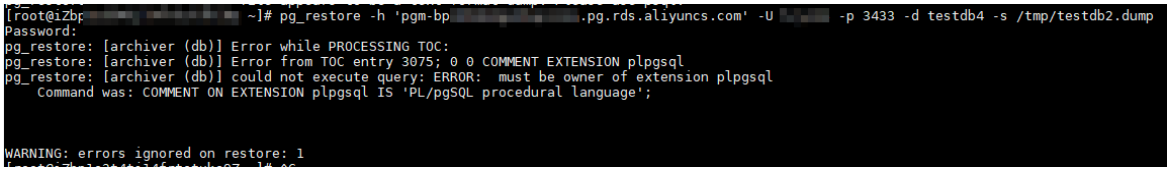
Parameter	Description
hostname	The endpoint used to connect to your RDS instance. **Note** ◦ If the ECS instance connects to your RDS instance by using an internal endpoint, you must make sure that the ECS and RDS instances have the same network type. If both instances use the VPC network type, you must also make sure that they reside in the same VPC. ◦ If the on-premises host or ECS instance connects to your RDS instance by using a public endpoint, you must make sure that a public endpoint has been allocated to your RDS instance. For more information, see Apply for or release a public endpoint on an ApsaraDB RDS for PostgreSQL instance.
username	The username of the privileged account of your RDS instance.
port	The port used to connect to your RDS instance.
dbname	The name of the database whose schema you want to restore.
-s	`-s`: specifies to only restore the schema of the database. The data of the database is not restored. For more information, visit pg_restore.
dumpdir	The directory and name of the logical backup file to use.

Example:

```
pg_restore -h 'pgm-bpxxxxx.pg.rds.aliyuncs.com' -U test123 -p 3433 -d testdb4 -s /tmp/testdb2.dump
```

2. When `Password:` appears, enter the password of the privileged account of your RDS instance and press Enter.

Note You can ignore the alerts generated by the embedded plpgsql plug-in.



29.Performance Optimization and diagnosis

29.1. Overview of DAS

Database Autonomy Service (DAS) is a cloud-based stable, secure, and efficient database service that uses machine learning and the experience of database experts to automate perception, healing, optimization, O&M, and security assurance for databases. DAS prevents service failures that may occur due to manual operations.

Prerequisites

Your ApsaraDB RDS for PostgreSQL instance runs RDS High-availability Edition.

Functionality

DAS provides the following features for your RDS instance:

- Use the session management feature for an ApsaraDB RDS for PostgreSQL instance

  The session management feature is used to view, export, and close the sessions of your RDS instance.
- Use the real-time monitoring feature for an ApsaraDB RDS for PostgreSQL instance

  The real-time monitoring feature is used to view the real-time performance of your RDS instance.
- Storage analysis

  The storage analysis feature is used to view the storage usage, such as the remaining days for which storage is available, the storage usage of individual tables, the tablespace fragments, and the storage exception diagnosis, of your RDS instance. This feature helps you identify the storage exceptions in a timely manner to ensure the stability of your RDS instance.
- Use the performance insight feature for an ApsaraDB RDS for PostgreSQL instance

  The performance insight feature is used to monitor the loads, analyze the associated data, and optimize the performance of your RDS instance. This feature helps you troubleshoot performance issues to increase the stability of your RDS instance.
- Use the performance trends feature for an ApsaraDB RDS for PostgreSQL instance

  The performance trend feature is used to view performance trends over specified time ranges, compare performance trends, and customize charts to view the performance trends of your RDS instance.
- Use the slow query log analysis feature for an ApsaraDB RDS for PostgreSQL instance

  The slow query log analysis feature is used to check the trends, execution, and optimization suggestions for slow query logs of your RDS instance.
- Use the SQL Explorer and Audit feature on an ApsaraDB RDS for PostgreSQL instance

  The SQL Explorer and Audit feature is based on the full request analysis and security audit capabilities of DAS and is integrated with the SQL statement search feature and the SQL Explorer feature. You can use the SQL Explorer and Audit feature to query information about the SQL statements that are executed on your RDS instance. Then, you can use the information to troubleshoot various performance issues.

29.2. Diagnostics

29.2.1. Use the session management feature for an ApsaraDB RDS for PostgreSQL instance

Database Autonomy Service (DAS) provides the session management feature for ApsaraDB RDS for PostgreSQL. You can use this feature to view and export the statistics of the sessions on your ApsaraDB RDS for PostgreSQL instance. You can also terminate the sessions on your RDS instance. This topic describes how to use the session management feature.

Prerequisites

Your ApsaraDB RDS for PostgreSQL instance runs ApsaraDB RDS for PostgreSQL High-availability Edition.

Procedure

1.
2. In the left-side navigation pane, choose **Autonomy Service > Diagnostics**.
3. Click the **Session Management** tab.
4. On the **Session Management** tab, perform the following operations in the **Instance Sessions** and **Session Statistics** sections based on your business requirements:

29.2.2. Use the real-time monitoring feature for an ApsaraDB RDS for PostgreSQL instance

Database Autonomy Service (DAS) provides the real-time monitoring feature for ApsaraDB RDS for PostgreSQL. You can use this feature to view the real-time performance of your ApsaraDB RDS for PostgreSQL instance. This topic describes how to use the real-time monitoring feature.

Prerequisites

Your ApsaraDB RDS for PostgreSQL instance runs ApsaraDB RDS for PostgreSQL High-availability Edition.

Procedure

1.
2. In the left-side navigation pane, choose **Autonomy Service > Diagnostics**.
3. Click the **Real-time Monitoring** tab.
4.

29.2.3. Storage analysis

This topic describes the storage analysis feature of Database Autonomy Service (DAS). You can use this feature to view the storage usage, such as the remaining days for which storage is available, the storage usage of individual tables, the tablespace fragments, and the storage exception diagnosis of your ApsaraDB RDS for PostgreSQL instance. This feature helps you identify the storage exceptions in a timely manner to ensure service stability.

Prerequisites

Your ApsaraDB RDS for PostgreSQL instance runs ApsaraDB RDS for PostgreSQL High-availability Edition.

Procedure

1.
2. In the left-side navigation pane, choose **Autonomy Service > Diagnostics**.
3. Click the **Storage Analysis** tab.

> **Note** For more information, see Storage analysis.

29.2.4. Use the performance insight feature for an ApsaraDB RDS for PostgreSQL instance

Database Autonomy Service (DAS) provides the performance insight feature for ApsaraDB RDS for PostgreSQL. You can use this feature to monitor the loads, analyze the associated data, and optimize the performance of your ApsaraDB RDS for PostgreSQL instance. This feature helps you troubleshoot performance issues to increase the stability of your RDS instance.

Prerequisites

Your ApsaraDB RDS for PostgreSQL instance runs ApsaraDB RDS for PostgreSQL High-availability Edition.

Background information

Procedure

1.
2. In the left-side navigation pane, choose **Autonomy Service > Diagnostics**.
3. On the **Performance Insight** tab, click **Enable Performance Insight**. In the message that appears, click **OK** to enable the performance insight feature.
4. In the **Performance Insight** and **Average Active Session** sections, view the following information about the RDS instance.

> **Note**

29.3. Use the performance trends feature for an ApsaraDB RDS for PostgreSQL instance

Database Autonomy Service (DAS) provides the performance trends feature for ApsaraDB RDS for PostgreSQL. You can use this feature to view performance trends over specific ranges, compare performance trends, and customize charts to view the performance trends of your ApsaraDB RDS for PostgreSQL instance.

Prerequisites

Your ApsaraDB RDS for PostgreSQL instance runs ApsaraDB RDS for PostgreSQL High-availability Edition.

Procedure

1.
2. In the left-side navigation pane, choose **Autonomy Service > Dashboard**.
3. On the page that appears, perform the following operations based on your business requirements:

29.4. Use the slow query log analysis feature for an ApsaraDB RDS for PostgreSQL instance

Database Autonomy Service (DAS) provides the slow query log analysis feature for ApsaraDB RDS for PostgreSQL. You can use this feature to check the trend, execution, and optimization suggestions for slow query logs of your ApsaraDB RDS for PostgreSQL instance.

Prerequisites

Your ApsaraDB RDS for PostgreSQL instance runs ApsaraDB RDS for PostgreSQL High-availability Edition.

Procedure

1.
2. In the left-side navigation pane, choose **Autonomy Service > Slow Query Logs**.
3.

29.5. Use the query governance feature for an ApsaraDB RDS for PostgreSQL instance

Database Autonomy Service (DAS) provides the query governance feature. DAS implements offline data analysis to automatically analyze all slow queries that are caused by specific SQL statements on ApsaraDB RDS for PostgreSQL instances in the previous day and adds tags to the SQL statements based on severity levels at 01:00 every day. This can help you categorize the SQL statements. DAS also provides suggestions to optimize the SQL statements and allows you to export query governance data. This topic describes how to use the query governance feature for an RDS instance.

Prerequisites

Your ApsaraDB RDS for PostgreSQL instance runs ApsaraDB RDS for PostgreSQL High-availability Edition.

Procedure

1.
2. In the left-side navigation pane, choose **Autonomy Service > Slow Query Logs**. On the page that appears, click the **Query Governance** tab.
3. On the **Query Governance** tab, view the query governance results.

- **Overview of query governance results**: This section displays the query governance results after SQL statements are categorized and tags are added to the SQL statements.

  > **Note** Only the value of the **Failed SQL Executions** parameter of an RDS instance for which DAS Professional Edition is enabled is collected.

- **Query governance trend**: This section displays the trend of changes in the query governance results in a specific time range.
- **Top Rankings**: This section displays the **Worst-performing Instances** and **Best-performing Instances** charts.
  - Worst-performing Instances: Instances are displayed in descending order based on the number of slow queries that are executed on each instance. This can help you identify the instance on which the largest number of slow queries are executed.
  - Best-performing Instances: Instances are displayed in ascending order based on changes in the number of slow queries that are executed on each instance. A negative value indicates that a smaller number of slow queries are executed compared with the number of slow queries that are executed on the previous day. A positive value indicates that a larger number of slow queries are executed compared with the number of slow queries that are executed on the previous day. This can help you identify the instance on which SQL statement optimization is most effective.

  We recommend that you handle the instance on which the largest number of slow queries are executed and the instance on which SQL statements require optimization at the earliest opportunity.
- **SQL to Be Optimized**: You can specify filter conditions to filter the SQL statements that you want to manage.

  > **Note** You can specify database names, SQL statement keywords, and tags to filter SQL statements. The database name filter, keyword filter, and tag filter are in the logical AND relation.
  >
  > - You can specify multiple database names. Separate the database names with commas (,). The database names are in the logical OR relation.
  > - You can specify multiple SQL statement keywords. Separate the SQL statement keywords with spaces. The specified keywords are in the logical AND relation.
  > - You can select multiple tags. The selected tags are in the logical OR relation.

  - You can click **Suggestions** in the **Actions** column of the required SQL statement to view the suggestions on query governance.
  - You can click **Add Tag** in the **Actions** column of the required SQL statement to add a tag to the SQL statement. For more information about tags, see the Tag table.

    You can also select multiple SQL statements to add tags to these SQL statements at a time.
  - You can click **Sample** in the **Actions** column of the required SQL statement to view the details of the SQL statement.
  - You can click **Trend** in the **Actions** column of the required SQL statement to view the analysis details of the SQL statement. For more information about how to analyze and manage slow queries, see Use the slow query log analysis feature for an ApsaraDB RDS for PostgreSQL instance.

You can export and share the SQL statements that require optimization based on your business requirements. For more information, see Best practices.

- **Failed SQL**: You can specify filter conditions to filter the SQL statements that you want to view.

  > **Note**
  > - Only the value of the **Failed SQL** parameter of an RDS instance for which DAS Professional Edition is enabled is collected.
  > - You can specify database names and SQL statement keywords to filter SQL statements. The database name filter and the keyword filter are in the logical AND relation.
  >   - You can specify multiple database names. Separate the database names with commas (,). The database names are in the logical OR relation.
  >   - You can specify multiple SQL statement keywords. Separate the SQL statement keywords with spaces. The specified keywords are in the logical AND relation.

Find the sample SQL statement and click **Sample** in the **Actions** column to view the details.

Best practices

- Use tags to identify the SQL statements that require optimization.

  The query governance feature classifies SQL statements that cause slow queries into the following categories: **SQL statements that require optimization** and **SQL statements that do not require optimization**. You can use tags to filter SQL statements and optimize SQL statements based on the severity levels that are specified by the tags. The following table describes the tags supported by DAS.

  We recommend that you handle the SQL statements that require optimization at the earliest opportunity. Before the system analyzes slow queries, you can add one of the following tags to SQL statements that do not require optimization. This can help reduce the number of SQL statements that require optimization.

-
-

29.6. Use the SQL Explorer and Audit feature on an ApsaraDB RDS for PostgreSQL instance

This topic describes how to use the SQL Explorer and Audit feature on an ApsaraDB RDS for PostgreSQL instance. The SQL Explorer and Audit feature is based on the full request analysis and security audit capabilities of Database Autonomy Service (DAS) and is integrated with the SQL statement search feature and the SQL Explorer feature. You can query information about executed SQL statements. Then, you can use the information to troubleshoot various performance issues.

Prerequisites

- DAS Professional Edition is purchased within your Alibaba Cloud account. For more information about how to purchase DAS Professional Edition, see Purchase DAS Professional Edition.
- Your ApsaraDB RDS for PostgreSQL instance runs ApsaraDB RDS for PostgreSQL High-availability Edition.

-

Billing

You are charged for DAS Professional Edition. For more information, see Pricing of DAS Professional Edition.

> **Note** After the SQL Explorer and Audit feature is enabled, ApsaraDB RDS stops billing the Use the SQL Audit feature on an ApsaraDB RDS for PostgreSQL instance feature. The pricing of the SQL Explorer and Audit feature is based on the pricing of DAS Professional Edition.

Overview

- The SQL statement search feature is used to query and export the SQL statements that are executed and the information about the SQL statements. The information includes the database, status, and execution duration of each SQL statement. For more information, see Search.
- The SQL Explorer feature is used to perform health diagnoses on SQL statements, troubleshoot performance issues, and analyze business traffic. For more information, see SQL Explorer.

Enable the SQL Explorer and Audit feature

1.
2. In the left-side navigation pane, choose **Autonomy Service > SQL Explorer and Audit**.
3. If the SQL Explorer and Audit feature is not enabled, click **Enable** on the page that appears. In the dialog box that appears, click **Enable Professional Edition**.

   > **Note** If the page that appears contains the Search and SQL Explorer tabs, the **SQL Explorer and Audit** feature is enabled.

Query information on the Search tab

1.
2. In the left-side navigation pane, choose **Autonomy Service > SQL Explorer and Audit**.
3. On the page that appears, click the **Search** tab. Then, specify the search conditions based on which you want to query the relevant information. For more information about the **Search** tab, see Search.

Query information on the SQL Explorer tab

1.
2. In the left-side navigation pane, choose **Autonomy Service > SQL Explorer and Audit**.
3. On the page that appears, click the **SQL Explorer** tab. Then, view the relevant information of the RDS instance. For more information about the **SQL Explorer** tab, see SQL Explorer.

29.7. Use the monitoring dashboard feature

Database Autonomy Service (DAS) provides the monitoring dashboard feature for ApsaraDB RDS for PostgreSQL. DAS allows you to specify RDS instances and metrics to monitor and compare the metrics of the RDS instances. You can also configure metric linkage. This helps you understand the status of ApsaraDB RDS for PostgreSQL instances.

Prerequisites

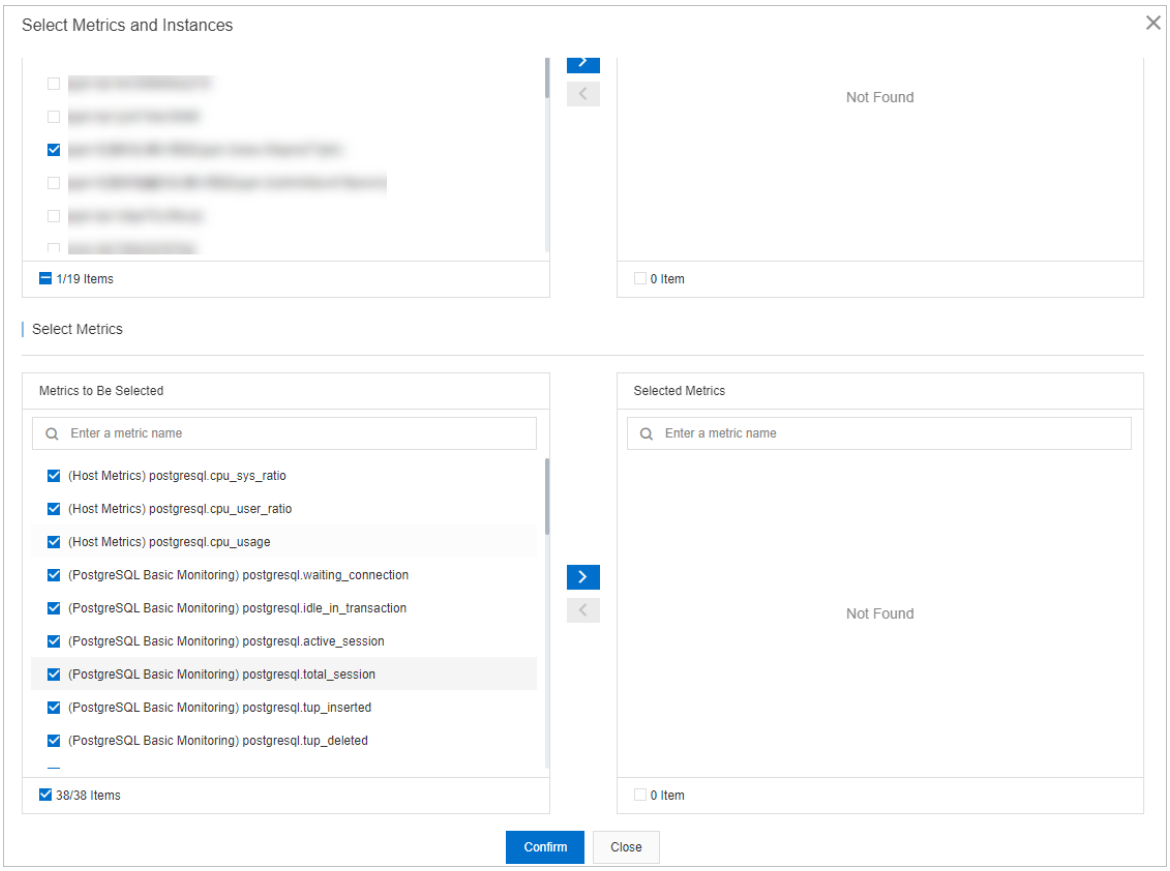
Your ApsaraDB RDS for PostgreSQL instance runs RDS High-availability Edition.

> **Note** DAS provides the monitoring dashboard feature for ApsaraDB RDS for PostgreSQL from May 20, 2022.

Create a monitoring dashboard

1. Log on to the ApsaraDB RDS console.
2. In the left-side navigation pane, click **Performance Center**.
3. On the **Performance Center** page, click the **Monitoring Dashboard** tab.
4. Click the tab for the database engine. Then, click **Add Monitoring Dashboard**.
5. In the dialog box that appears, configure the **Dashboard Name** parameter and click **OK**.
6. Click **Select Instances and Metrics**. In the dialog box that appears, select the RDS instances and the metrics that you want to monitor. Then, click the icon to add the selected RDS instances to the Selected Instances section and the selected metrics to the Selected Metrics section.

   > **Note** For more information about the metrics, see Metrics.



7. Click **Confirm**.

> **Note** To modify the instances or metrics in the monitoring dashboard, click **Add Instances and Metrics**.

View the metric trends of an RDS instance in the monitoring dashboard

1. Log on to the ApsaraDB RDS console.
2. In the left-side navigation pane, click **Performance Center**.
3. On the **Performance Center** page, click the **Monitoring Dashboard** tab.
4. Click the tab for the database engine, select the monitoring dashboard that you want to view, and then specify a time range to view the trend charts of the metrics during the specified time range.

> **Note** When you specify a time range, the end time must be later than the start time, and the interval between the start time and the end time cannot exceed seven days.

   - You can configure the **Instance filtering** parameter to filter for multiple RDS instances and then view and compare the metrics of the RDS instances.
   - You can turn on **Auto Refresh (Every 5 Seconds)** for the system to refresh the trend charts of the metrics every 5 seconds.
   - You can turn on **Linkage Chart** to view the values of different metrics at the same point in time.
   - You can configure the **Chart Layout** parameter to specify the number of trend charts of metrics in each row.
   - You can click **Add Instances and Metrics** to modify the RDS instances or metrics in the dashboard.
   - You can click **Details** in the trend chart of a metric to expand the chart. You can also change the time range to view the changes in the trend of the metric at the specified time range.
   - You can click **Delete** in the trend chart of a metric to delete the chart from the dashboard.

Metrics

Category	Metric	Description
	cpu_sys_ratio	The CPU utilization of the PostgreSQL process in the kernel state.
	cpu_user_ratio	The CPU utilization of the PostgreSQL process in the user state.
	cpu_usage	The CPU utilization of the PostgreSQL process.
	mem_usage	The memory usage of the PostgreSQL process.
	data.r_s	The read IOPS of the disk on which the data directory resides.
	data.iops	The IOPS of the disk on which the data directory resides.

Category	Metric	Description
Host	data.w_s	The write IOPS of the disk on which the data directory resides.
	local_fs_size_total	The total amount of disk space of the server.
	data.fs.used	The amount of used space of the disk on which the data directory resides.
	data.fs.usage	The space usage of the disk on which the data directory resides.
	network_out_io	The outbound network traffic of the PostgreSQL process.
	network_io	The network traffic of the PostgreSQL process.
	network_in_io	The inbound network traffic of the PostgreSQL process.
PostgreSQL basic monitoring	waiting_connection	The number of pending connections.
	idle_in_transaction	The number of idle sessions.
	active_session	The number of active connections.
	total_session	The total number of current connections.
	tup_inserted	The number of records that are inserted per second.
	tup_deleted	The number of records that are deleted per second.
	tup_updated	The number of records that are updated per second.
	tup_returned	The number of records that are returned per second for full table scans.
	tup_fetched	The number of records that are returned per second for index scans.
	tps	The transactions per second (TPS) in PostgreSQL.
	xact_rollback	The number of transactions that are rolled back per second.
	xact_commit	The number of transactions that are committed per second.
	deadlocks	The number of deadlocks per second.
	oldest_snapshot	The point in time before which the generated dead tuples can be recycled by the RDS instance.
	max_sent_delay	The latency of data replication from the secondary RDS instance.

Category	Metric	Description
	max_replay_delay	The latency at which the secondary RDS instance replays transaction commits.
	long_query_5s	The number of SQL statements whose execution duration is greater than or equal to 5 seconds.
	long_query_1s	The number of SQL statements whose execution duration is greater than or equal to 1 second.
	long_query_3s	The number of SQL statements whose execution duration is greater than or equal to 3 seconds.
	long_idle_in_transaction_3s	The number of transactions that are idle for 3 seconds or longer.
	long_idle_in_transaction_5s	The number of transactions that are idle for 5 seconds or longer.
	long_idle_in_transaction_1s	The number of transactions that are idle for 1 second or longer.
	long_2pc_1s	The number of 2PC transactions that last for 1 second or longer.
	long_2pc_5s	The number of 2PC transactions that last for 5 seconds or longer.
	long_2pc_3s	The number of 2PC transactions that last for 3 seconds or longer.

Note You can click the icon on the right of a metric in a dashboard to view the description of the metric.

30.Tag

30.1. Add tags to ApsaraDB RDS instances

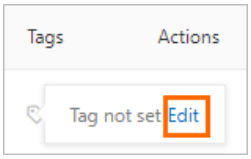
This topic describes how to add tags to one or more ApsaraDB RDS instances. You can use tags to classify a large number of RDS instances. Each tag consists of a key and a value. You can use tag keys and values to further classify RDS instances.

Limits

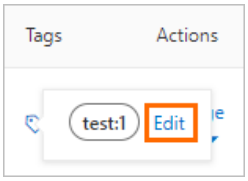
- You can add up to 20 tags to each RDS instance. Each tag must have a unique key. If two tags have the same key, the tag that is created later overwrites the earlier tag.
- You can add tags to up 50 RDS instances at a time.
- RDS instances in different regions do not share the same tag namespace.
- After you remove a tag from an RDS instance, ApsaraDB RDS checks whether the tag is added to other RDS instances. If the tag is not added to other RDS instances, ApsaraDB RDS deletes the tag.

Add tags to an RDS instance

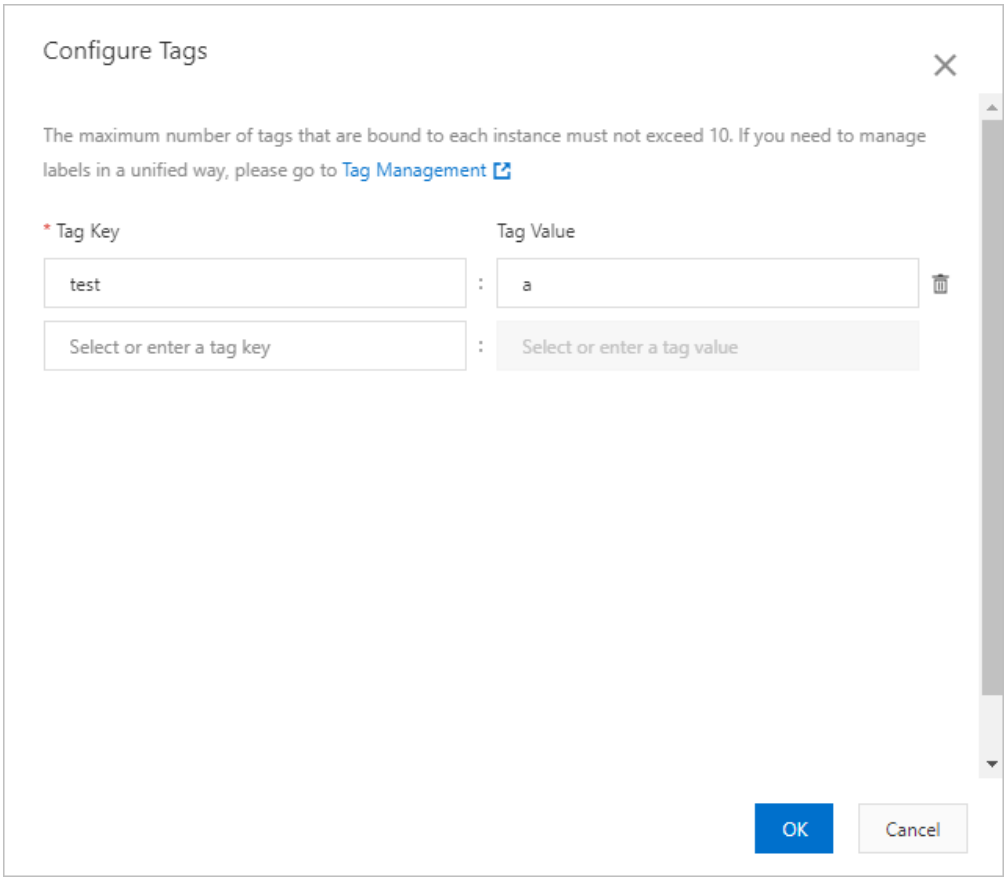
1.
2. Click the icon in the **Tags** column of the required RDS instance and then click **Edit**.



If you have added a tag to the RDS instance, you can click **Edit** to edit the tag.



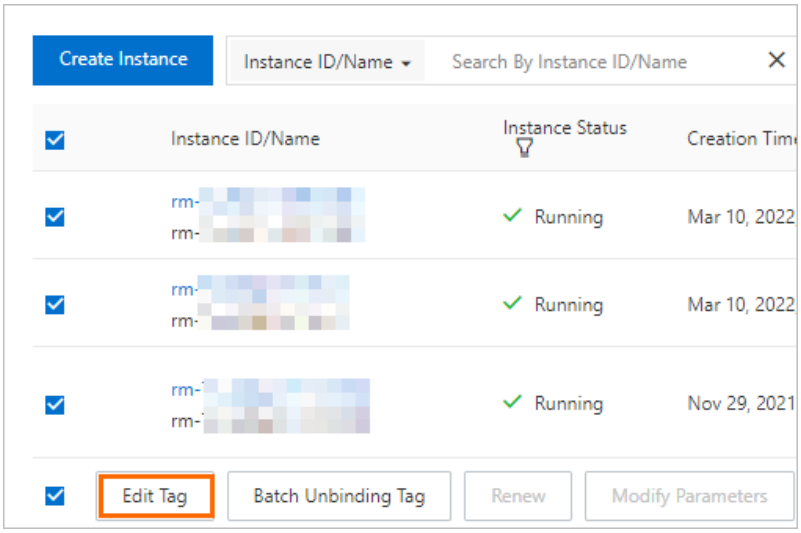
3. In the **Configure Tags** dialog box, configure the **Tag Key** and **Tag Value** parameters and click **OK**.



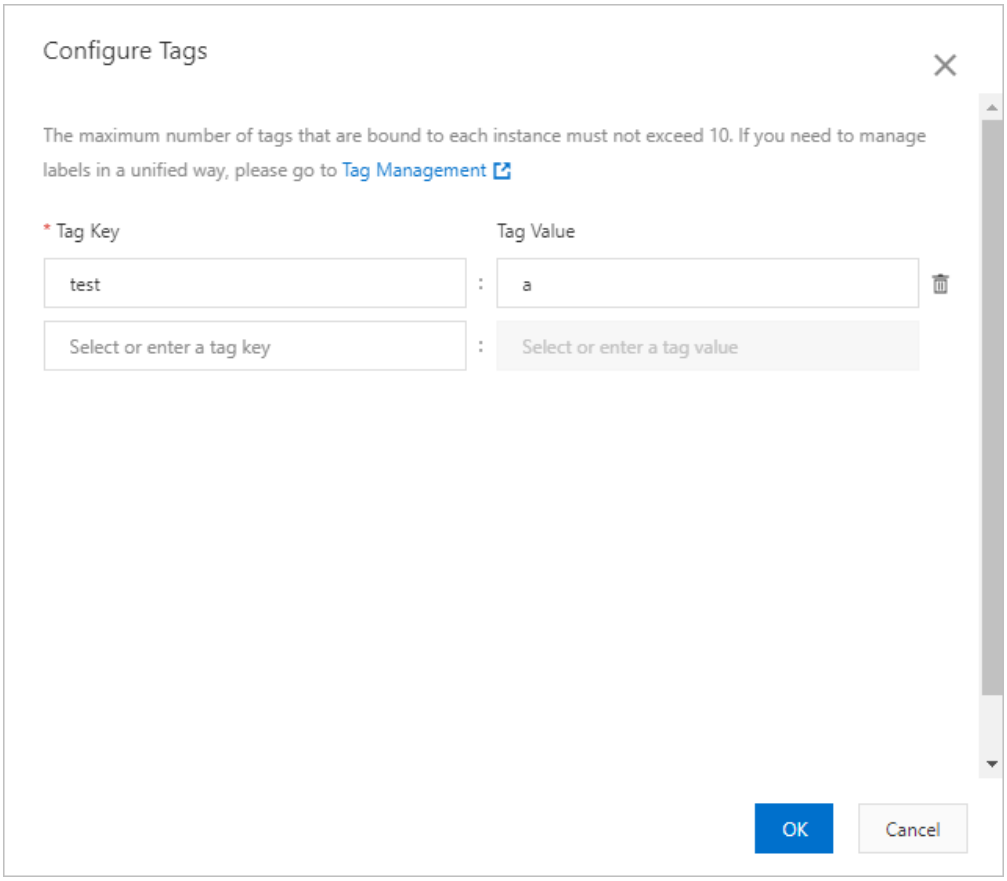
Add tags to multiple RDS instances at a time

1.
2. Select the RDS instances to which you want to add tags and click **Edit Tag** below the instance list.

> **Note** The Edit Tag button is displayed in the lower part of the page.



3. In the **Configure Tags** dialog box, configure the **Tag Key** and **Tag Value** parameters and click **OK**.



Related operations

Operation	Description
Create and bind tags	Adds tags to one or more ApsaraDB RDS instances.

30.2. Remove tags from an ApsaraDB RDS for MySQL instance

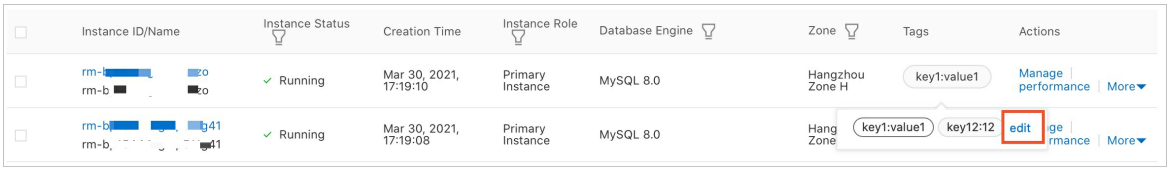
This topic describes how to remove tags from an ApsaraDB RDS for MySQL instance. If you change the configuration of your RDS instance or you no longer require specific tags, you can remove these tags from your RDS instance.

Limits

- You can remove a maximum of 20 tags at a time.
- After you remove a tag from your RDS instance, ApsaraDB RDS checks whether the tag is added to other RDS instances. If the tag is not added to other RDS instances, ApsaraDB RDS deletes the tag.

Procedure

1.
2. Find your RDS instance, move the pointer over the Tags column, and then click **Edit**.



Note After you find your RDS instance, you can also choose **More** > **Edit Tag** in the Actions column.

3. In the dialog box that appears, click the **X** icon next to each tag that you want to remove.

4. Click **OK**.

Related operations

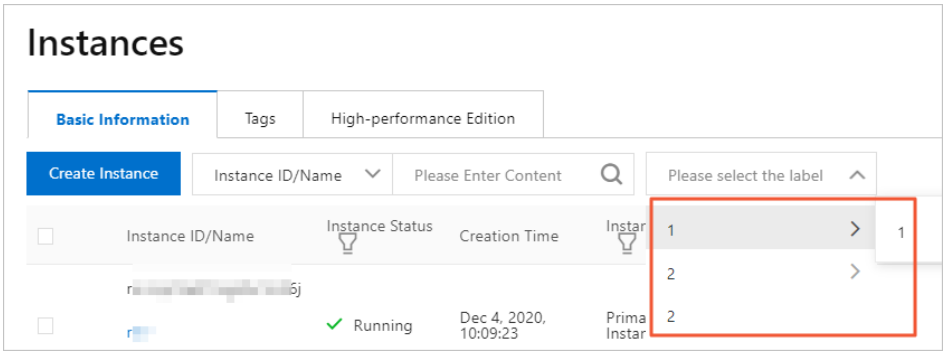
Operation	Description
Unbind tags	Removes tags from one or more RDS instances.

30.3. Use tags to filter ApsaraDB RDS for MySQL instances

This topic describes how to filter ApsaraDB RDS for MySQL instances based on the tags that are added to these instances.

1.
2. Select a **key** and a **value**. Then, ApsaraDB RDS filters your RDS instances based on the specified tag.

Note To cancel the filter condition that is specified by the tag, you can click the X icon to the right of the tag.



Related operations

Operation	Description
Query the tags of ApsaraDB RDS instances	Queries the tags that are added to one or more RDS instances.

31.Best Practices

31.1. Manage permissions in an ApsaraDB RDS for PostgeSQL instance

This topic describes how to manage permissions in an ApsaraDB RDS for PostgreSQL instance.

Principles of permission management

A role is created as a permission set. You can use roles to manage permissions at fine-grained levels. Roles do not have logon permissions. You can create users to which you can grant logon permissions. The management model of ApsaraDB RDS for PostgeSQL specifies that `the permissions of a user consist of the permissions of the role that is associated with the user and the logon permissions`. The permissions of a user vary based on the permissions of the associated role.

Model of permission management

The management model of ApsaraDB RDS for PostgeSQL is easy to use, effective, and suitable for most business scenarios.

- A privileged account can be created for your RDS instance. The privileged account has all permissions on your RDS instance and can be used only by a few database administrators.
- You can create one owner and two roles named {project}_role_readwrite and {project}_role_readonly. You can use the owner and the roles to manage a team or a project.

  > **Note** If you want to manage permissions at more fine-grained levels, you can create roles based on your business requirements.

- You can create users. The permissions of a user `consist of the permissions of the role that is associated with the user and the logon permissions`.
- Multiple schemas can be defined for a team or a project. We recommend that you grant permissions at the schema level or the role level.
- Do not place tables in the schema named `public`. By default, all users have the **CREATE** permission and the **USAGE** permission on the `public` schema.

Example of permission management

This section provides an example on how to manage permissions at the project level. You can also follow the instructions that are provided in the example to manage permissions at the team level.

- A database administrator can use the privileged account named dbsuperuser of your RDS instance.
- Your project is named rdspg, and two schemas named rdspg and rdspg_1 are created.

The following table describes the permissions of the owner and roles that you created in your project.

Owner or Role	Permission on tables	Permission on stored procedures

Owner or Role	Permission on tables	Permission on stored procedures
An owner named rdspg_owner	• DDL: the permissions to perform CREATE, DROP, and ALTER operations. • Data Query Language (DQL): the permissions to perform SELECT operations. • DML: the permissions to perform UPDATE, INSERT, and DELETE operations.	• DDL: the permissions to perform CREATE, DROP, and ALTER operations. • DQL: the permissions to perform SELECT operations and the permissions to call stored procedures.
A role named rdspg_role_readwrite	• DQL: the permissions to perform SELECT operations. • DML: the permissions to perform UPDATE, INSERT, and DELETE operations.	DQL: the permissions to perform SELECT operations and the permissions to call stored procedures. If DDL operations are found in stored procedures, an error message that is related to permissions is displayed.
A role named rdspg_role_readonly	DQL: the permissions to perform SELECT operations.	DQL: the permissions to perform SELECT operations and the permissions to call stored procedures. If DDL operations are found in stored procedures, an error message that is related to permissions is displayed.

You can grant permissions to the users that you create based on your business requirements.

- The permissions of the rdspg_readwrite user consist of the permissions of the rdspg_role_readwrite role and the logon permissions.
- The permissions of the rdspg_readonly user consist of the permissions of the rdspg_role_readonly role and the logon permissions.

Procedure

1. Create an owner named rdspg_owner and two roles named rdspg_role_readwrite and rdspg_role_readonly for your project.

   Use the privileged account named dbsuperuser of your RDS instance to perform the following operations as a database administrator:

```
--- rdspg_owner is the username of the owner. The password in this example is for referen
ce only. Replace the password with the actual password of the owner.
CREATE USER rdspg_owner WITH LOGIN PASSWORD 'asdfy181BASDfadasdbfas';
CREATE ROLE rdspg_role_readwrite;
CREATE ROLE rdspg_role_readonly;
--- Grant the permissions to perform DQL SELECT operations and DML UPDATE, INSERT, and DE
LETE operations on tables that are created by using the credentials of the rdspg_owner ow
ner to the rdspg_role_readwrite role.
ALTER DEFAULT PRIVILEGES FOR ROLE rdspg_owner GRANT ALL ON TABLES TO rdspg_role_readwrite
;
--- Grant the permissions to perform DQL SELECT operations and DML UPDATE, INSERT, and DE
LETE operations on sequences that are created by using the credentials of the rdspg_owner
owner to the rdspg_role_readwrite role.
ALTER DEFAULT PRIVILEGES FOR ROLE rdspg_owner GRANT ALL ON SEQUENCES TO rdspg_role_readwr
ite;
--- Grant the permission to perform DQL SELECT operations on tables that are created by u
sing the credentials of the rdspg_owner owner to the rdspg_role_readonly role.
ALTER DEFAULT PRIVILEGES FOR ROLE rdspg_owner GRANT SELECT ON TABLES TO rdspg_role_readon
ly;
```

2. Create two users named rdspg_readwrite and rdspg_readonly.

   Use the privileged account named dbsuperuser of your RDS instance to perform the following operations as a database administrator:

```
--- Grant the permissions to perform DQL SELECT operations and DML UPDATE, INSERT, and DE
LETE operations to the rdspg_readwrite user.
CREATE USER rdspg_readwrite WITH LOGIN PASSWORD 'dfandfnapSDhf23hbEfabf';
GRANT rdspg_role_readwrite TO rdspg_readwrite;
--- Grant the permission to perform DQL SELECT operations to the rdspg_readonly user.
CREATE USER rdspg_readonly WITH LOGIN PASSWORD 'F89h912badSHfadsd01zlk';
GRANT rdspg_role_readonly TO rdspg_readonly;
```

3. Create a schema named rdspg and grant the permissions on the schema to the rdspg_role_readwrite role and the rdspg_role_readonly role.

   Use the privileged account named dbsuperuser of your RDS instance to perform the following operations as a database administrator:

```
--- Specify the rdspg_owner owner as the owner of the rdspg schema.
CREATE SCHEMA rdspg AUTHORIZATION rdspg_owner;
--- Grant the permissions on the rdspg schema to the rdspg_role_readwrite role and the rd
spg_role_readonly role.
GRANT USAGE ON SCHEMA rdspg TO rdspg_role_readwrite;
GRANT USAGE ON SCHEMA rdspg TO rdspg_role_readonly;
```

> **Note** The rdspg_readwrite user and the rdspg_readonly user inherit the changes to the permissions of their associated roles. You do not need to grant permissions to the rdspg_readwrite user or the rdspg_readonly user.

Scenarios

Scenario 1: Use the rdspg_owner owner to perform DDL CREATE, DROP, and ALTER operations on tables in the rdspg schema

```
CREATE TABLE rdspg.test(id bigserial primary key, name text);
CREATE INDEX idx_test_name on rdspg.test(name);
```

Scenario 2: Use the rdspg_readwrite user or the rdspg_readonly user for business development

The user that you use for business management follows the principle of least privilege (PoLP). We recommend that you use the rdspg_readonly user for business management. Use the rdspg_readwrite user only when you must perform DML operations. PoLP facilitates read/write splitting at the business layer.

> **Note**
>
> - Read/write splitting at the business layer helps reduce the additional cost and performance loss caused by automatic read/write splitting that is performed by the proxy middleware.
> - If no read-only RDS instance is attached to your RDS instance, we recommend that you grant read permissions to one client and grant read and write permissions to the other client. This configuration is for the creation of read-only RDS instances. We also recommend that you follow PoLP and use the rdspg_readonly user for the client to which you granted read permissions.
>   - Use the rdspg_readonly user for the client to which you granted read permissions and set the Java Database Connectivity (JDBC) URL to `the endpoint of read-only RDS instance 1,the endpoint of read-only RDS instance 2,the endpoint of your RDS instance` .
>   - Use the rdspg_readwrite user for the client to which you granted read and write permissions and set the JDBC URL to `the endpoint of your RDS instance` .

- Use the rdspg_readwrite user to perform DQL SELECT operations and DML UPDATE, INSERT, and DELETE operations on the tables in the rdspg schema.

```
INSERT INTO rdspg.test (name) VALUES('name0'),('name1');
SELECT id,name FROM rdspg.test LIMIT 1;
--- The rdspg_readwrite user does not have the permissions to perform DDL CREATE, DROP, and
ALTER operations.
CREATE TABLE rdspg.test2(id int);
ERROR:  permission denied for schema rdspg
LINE 1: create table rdspg.test2(id int);
DROP TABLE rdspg.test;
ERROR:  must be owner of table test
ALTER TABLE rdspg.test ADD id2 int;
ERROR:  must be owner of table test
CREATE INDEX idx_test_name on rdspg.test(name);
ERROR:  must be owner of table test
```

- Use the rdspg_readonly user to perform DQL SELECT operations on the tables in the rdspg schema.

```
INSERT INTO rdspg.test (name) VALUES('name0'),('name1');
ERROR:  permission denied for table test
SELECT id,name FROM rdspg.test LIMIT 1;
 id | name
----+-------
  1 | name0
(1 row)
```

Scenario 3: Grant permissions on a project to another project

You can grant read permissions on the tables of the rdspg project to the employee_readwrite user that belongs to the employee project. Use the privileged account named dbsuperuser of your RDS instance to perform the following operations as a database administrator:

```
--- Grant the permissions of the rdspg_role_readonly role to the employee_readwrite user.
GRANT rdspg_role_readonly TO employee_readwrite;
```

Scenario 4: Create a schema named rdspg_2 and grant the permissions on the rdspg_2 schema to roles

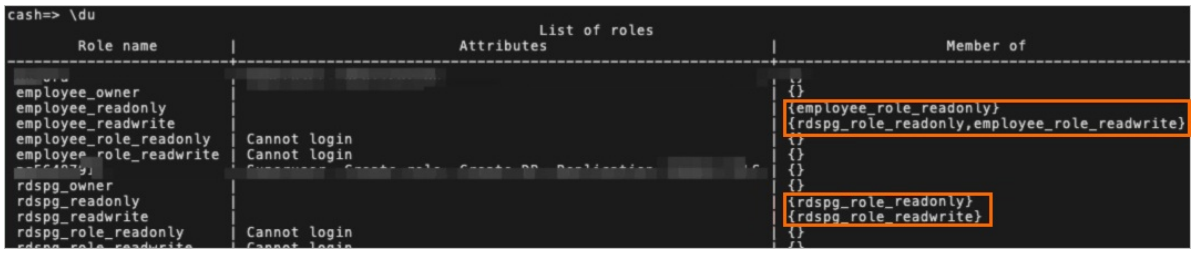
The rdspg_readwrite user, the rdspg_readonly user, and the employee_readwrite user inherit the changes to the permissions of their associated roles. You do not need to grant permissions to the rdspg_readwrite user, the rdspg_readonly user, or the employee_readwrite user. Use the privileged account named dbsuperuser of your RDS instance to perform the following operations as a database administrator:

```
CREATE SCHEMA rdspg_1 AUTHORIZATION rdspg_owner;
--- Grant the access permission on the rdspg_2 schema to roles.
--- Grant the permissions to perform DDL CREATE, DROP, and ALTER operations on tables in the
rdspg_1 schema.
GRANT USAGE ON SCHEMA rdspg_1 TO rdspg_role_readwrite;
GRANT USAGE ON SCHEMA rdspg_1 TO rdspg_role_readonly;
```

Queries on permissions

If you use the management model that is described in this topic, use one of the following methods to query the permissions of the users in your RDS instance:

- Use a command-line tool to connect to your RDS instance. For more information, see Connect to an ApsaraDB RDS for PostgreSQL instance. Then, run the `\du` command.



  The command output in the preceding figure shows that `rdspg_role_readonly,employee_role_readwrite` is displayed in the **Member of** column for the employee_readwrite user. Therefore, the permissions of DQL and DML operations are granted to the employee_readwrite user and the permissions of DQL operations are granted to tables in the rdspg project.

- Use SQL to query the permissions.

```
SELECT r.rolname, r.rolsuper, r.rolinherit,
  r.rolcreaterole, r.rolcreatedb, r.rolcanlogin,
  r.rolconnlimit, r.rolvaliduntil,
  ARRAY(SELECT b.rolname
        FROM pg_catalog.pg_auth_members m
        JOIN pg_catalog.pg_roles b ON (m.roleid = b.oid)
        WHERE m.member = r.oid) as memberof
, r.rolreplication
, r.rolbypassrls
FROM pg_catalog.pg_roles r
WHERE r.rolname !~ '^pg_'
ORDER BY 1;
```

31.2. Migrate data from a user-created PostgreSQL database to an ApsaraDB RDS for PostgreSQL database

This topic describes two methods that are used to migrate data from a user-created PostgreSQL database to an ApsaraDB RDS for PostgreSQL database.

Use Alibaba Cloud Data Transmission Service (DTS)

- Benefits
  - DTS provides a graphical user interface (GUI) that simplifies operations. This GUI can also direct you to technical support in the event of errors.
  - DTS supports the migration of incremental data at the minimal downtime.
  - DTS supports the migration of full data free of charge. However, you must pay for the migration of incremental data.
- Drawbacks

  You cannot specify tables by uploading files.

For more information, see Overview of data migration scenarios.

Use native PostgreSQL

If DTS cannot meet your business requirements, you can use native PostgreSQL to migrate data.

- Benefits
  - Native PostgreSQL ensures compatibility with the user-created PostgreSQL database and the ApsaraDB RDS for PostgreSQL database.
  - Native PostgreSQL migrates data at high speeds.
  - Native PostgreSQL allows you to specify schemas and databases. This way, you can migrate only the tables that you want to migrate.
- Drawbacks
  - Native PostgreSQL does not support the migration of incremental data. It only supports the migration of full data.
  - Native PostgreSQL requires you to understand the technical basics of PostgreSQL and Linux.

For more information, see Manually migrate data from a user-created PostgreSQL database hosted on ECS to an ApsaraDB RDS for PostgreSQL database.

31.3. Configure the collation of a database on an ApsaraDB RDS for PostgreSQL instance

When you initialize an ApsaraDB RDS for PostgreSQL instance, you can configure the collation of each database based on your business requirements. The collation includes the string sort order, character classification method, numeric value format, date and time format, and currency format. In addition, you may also need to configure the LC_COLLATE and LC_CTYPE environment variables.

LC_COLLATE	String sort order
LC_CTYPE	Character classification
LC_MESSAGES	Message language
LC_MONETARY	Currency format
LC_NUMERIC	Numeric value format
LC_TIME	Date and time format

You can configure these environment variables to specify a collation that meets your business requirements in a locale.

Supported character sets

For more information, see Character Set Support.

Name	Description	Language	Server	Bytes/Char	Aliases
BIG5	Big Five	Traditional Chinese	No	January 2	WIN950, Windows950
EUC_CN	Extended UNIX Code-CN	Simplified Chinese	Yes	January 3	-
EUC_JP	Extended UNIX Code-JP	Japanese	Yes	January 3	-
EUC_JIS_2004	Extended UNIX Code-JP, JIS X 0213	Japanese	Yes	January 3	-
EUC_KR	Extended UNIX Code-KR	Korean	Yes	January 3	-
EUC_TW	Extended UNIX Code-TW	Traditional Chinese, Taiwanese	Yes	January 3	-

Name	Description	Language	Server	Bytes/Char	Aliases
GB18030	National Standard	Chinese	No	January 4	-
GBK	Extended National Standard	Simplified Chinese	No	January 2	WIN936, Windows936
ISO_8859_5	ISO 8859-5, ECMA 113	Latin/Cyrillic	Yes	1	-
ISO_8859_6	ISO 8859-6, ECMA 114	Latin/Arabic	Yes	1	-
ISO_8859_7	ISO 8859-7, ECMA 118	Latin/Greek	Yes	1	-
ISO_8859_8	ISO 8859-8, ECMA 121	Latin/Hebrew	Yes	1	-
JOHAB	JOHAB	Korean (Hangul)	No	January 3	-
KOI8R	KOI8-R	Cyrillic (Russian)	Yes	1	KOI8
KOI8U	KOI8-U	Cyrillic (Ukrainian)	Yes	1	-
LATIN1	ISO 8859-1, ECMA 94	Western European	Yes	1	ISO88591
LATIN2	ISO 8859-2, ECMA 94	Central European	Yes	1	ISO88592
LATIN3	ISO 8859-3, ECMA 94	South European	Yes	1	ISO88593
LATIN4	ISO 8859-4, ECMA 94	North European	Yes	1	ISO88594
LATIN5	ISO 8859-9, ECMA 128	Turkish	Yes	1	ISO88599
LATIN6	ISO 8859-10, ECMA 144	Nordic	Yes	1	ISO885910
LATIN7	ISO 8859-13	Baltic	Yes	1	ISO885913
LATIN8	ISO 8859-14	Celtic	Yes	1	ISO885914
LATIN9	ISO 8859-15	LATIN1 with Euro and accents	Yes	1	ISO885915

Name	Description	Language	Server	Bytes/Char	Aliases
LATIN10	ISO 8859-16, ASRO SR 14111	Romanian	Yes	1	ISO885916
MULE_INTERNAL	Mule internal code	Multilingual Emacs	Yes	January 4	-
SJIS	Shift JIS	Japanese	No	January 2	Mskanji, ShiftJIS, WIN932, Windows932
SHIFT_JIS_2004	Shift JIS, JIS X 0213	Japanese	No	January 2	-
SQL_ASCII	unspecified (see text)	any	Yes	1	-
UHC	Unified Hangul Code	Korean	No	January 2	WIN949, Windows949
UTF8	Unicode, 8-bit	all	Yes	January 4	Unicode
WIN866	Windows CP866	Cyrillic	Yes	1	ALT
WIN874	Windows CP874	Thai	Yes	1	-
WIN1250	Windows CP1250	Central European	Yes	1	-
WIN1251	Windows CP1251	Cyrillic	Yes	1	WIN
WIN1252	Windows CP1252	Western European	Yes	1	-
WIN1253	Windows CP1253	Greek	Yes	1	-
WIN1254	Windows CP1254	Turkish	Yes	1	-
WIN1255	Windows CP1255	Hebrew	Yes	1	-
WIN1256	Windows CP1256	Arabic	Yes	1	-
WIN1257	Windows CP1257	Baltic	Yes	1	-

Name	Description	Language	Server	Bytes/Char	Aliases
WIN1258	Windows CP1258	Vietnamese	Yes	1	ABC, TCVN, TCVN5712, VSCII

LC_COLLATE and LC_CTYPE settings supported by a character set

You can execute the following SQL statement to query the LC_COLLATE and LC_CTYPE settings that are supported by a character set from the pg_collation system table:

```
test=> select pg_encoding_to_char(collencoding) as encoding,collname,collcollate,collctype fr
om pg_collation ;
```

If the encoding field of a collation is empty, the collation supports all character sets.

```
  encoding  |       collname        |      collcollate      |       collctype
------------+-----------------------+-----------------------+-----------------------
            | default               |                       |
            | C                     | C                     | C
            | POSIX                 | POSIX                 | POSIX
 UTF8       | aa_DJ                 | aa_DJ.utf8            | aa_DJ.utf8
 LATIN1     | aa_DJ                 | aa_DJ                 | aa_DJ
 LATIN1     | aa_DJ.iso88591        | aa_DJ.iso88591        | aa_DJ.iso88591
 UTF8       | aa_DJ.utf8            | aa_DJ.utf8            | aa_DJ.utf8
 UTF8       | aa_ER                 | aa_ER                 | aa_ER
 UTF8       | aa_ER.utf8            | aa_ER.utf8            | aa_ER.utf8
.......
 EUC_CN     | zh_CN                 | zh_CN                 | zh_CN
 UTF8       | zh_CN                 | zh_CN.utf8            | zh_CN.utf8
 EUC_CN     | zh_CN.gb2312          | zh_CN.gb2312          | zh_CN.gb2312
 UTF8       | zh_CN.utf8            | zh_CN.utf8            | zh_CN.utf8
 UTF8       | zh_HK                 | zh_HK.utf8            | zh_HK.utf8
 UTF8       | zh_HK.utf8            | zh_HK.utf8            | zh_HK.utf8
 EUC_CN     | zh_SG                 | zh_SG                 | zh_SG
 UTF8       | zh_SG                 | zh_SG.utf8            | zh_SG.utf8
 EUC_CN     | zh_SG.gb2312          | zh_SG.gb2312          | zh_SG.gb2312
 UTF8       | zh_SG.utf8            | zh_SG.utf8            | zh_SG.utf8
 EUC_TW     | zh_TW                 | zh_TW.euctw           | zh_TW.euctw
 UTF8       | zh_TW                 | zh_TW.utf8            | zh_TW.utf8
 EUC_TW     | zh_TW.euctw           | zh_TW.euctw           | zh_TW.euctw
 UTF8       | zh_TW.utf8            | zh_TW.utf8            | zh_TW.utf8
 UTF8       | zu_ZA                 | zu_ZA.utf8            | zu_ZA.utf8
 LATIN1     | zu_ZA                 | zu_ZA                 | zu_ZA
 LATIN1     | zu_ZA.iso88591        | zu_ZA.iso88591        | zu_ZA.iso88591
 UTF8       | zu_ZA.utf8            | zu_ZA.utf8            | zu_ZA.utf8
(869 rows)
```

Configure the collation of a database in a locale

- Configure the fields of a database in a locale

  Prerequisites

Familiarize yourself with the collations that are supported by the character set of the database. Then, execute the following SQL statement to query the encoding format of the database:

```
postgres=# select datname,pg_encoding_to_char(encoding) as encoding from pg_database;
```

Information similar to the following output is returned:

```
      datname       | encoding
--------------------+-----------
 template1          | UTF8
 template0          | UTF8
 db                 | SQL_ASCII
 db1                | EUC_CN
 contrib_regression | UTF8
 test01             | UTF8
 test02             | UTF8
 postgres           | UTF8
(8 rows)
```

Procedure

i. When you create a table, execute the following SQL statement to specify a collation that is supported by the character set of the database:

```
CREATE TABLE test1 (
 a text COLLATE "de_DE",
 b text COLLATE "es_ES",
 ...
);
```

ii. Execute the following SQL statement to modify the collation of a column:

> **Note** When you modify the collation of a column in a table, the table is rewritten. Proceed with caution if the table is large.

```
alter table a alter c1 type text COLLATE "zh_CN";
```

- Configure locale settings.

- Execute the following SQL statement to change the sort order that is specified by the ORDER BY clause:

  ```
  test=# select * from a order by c1 collate "C";
  ```

  Information similar to the following output is returned:

  ```
     c1
  --------
   Tom
   Alice
  (2 rows)
  test=# select * from a order by c1 collate "zh_CN";
     c1
  --------
   Alice
   Tom
  (2 rows)
  ```

- Execute the following SQL statement to change the result that is returned from an operator:

  Example 1:

  ```
  select * from a where c1 > 'Tom' collate "C";
  ```

  Information similar to the following output is returned:

  ```
     c1
  --------
   Alice
  (1 row)
  ```

  Example 2:

  ```
  select * from a where c1 > 'Tom' collate "en_US";
  ```

  Information similar to the following output is returned:

  ```
    c1
  ----
  (0 rows)
  ```

- Sort data by using locale indexes.

  You can sort data by using an index only when the collation specified in the ORDER BY clause is the same as the collation of the index. Execute the following SQL statement:

  ```
  create index idxa on a(c1 collate "zh_CN");
  explain select * from a order by c1 collate "zh_CN";
  ```

  Information similar to the following output is returned:

  ```
                                QUERY PLAN
  ------------------------------------------------------------------------
   Index Only Scan using idxa on a  (cost=0.15..31.55 rows=1360 width=64)
  (1 row)
  ```

Configure a rule to sort results in alphabetical order

You can use one of the following four methods to configure a rule that is used to sort results in alphabetical order:

- Use the SQL statements that are supported in your locale. This method does not require you to modify the original data. Execute the following SQL statement:

  ```
  select * from a order by c1 collate "zh_CN";
  ```

  Information similar to the following output is returned:

  ```
     c1
  --------
   Alice
   Tom
  (2 rows)
  ```

- Use the fields that are supported in your locale. If the database contains data, this method requires you to modify the original data. Execute the following SQL statement:

  ```
  alter table a alter c1 type text COLLATE "zh_CN";
  ```

- Use the indexes and SQL statements that are supported in your locale. This method does not require you to modify the original data. Execute the following SQL statements:

  ```
  create index idxa on a(c1 collate "zh_CN");
  explain select * from a order by c1 collate "zh_CN";
  ```

  Information similar to the following output is returned:

  ```
                                QUERY PLAN
  ------------------------------------------------------------------------
   Index Only Scan using idxa on a  (cost=0.15..31.55 rows=1360 width=64)
  (1 row)
  ```

- Set the collation of the database to en_US. By default, the data in this database is sorted in alphabetical order based on the specified collation. Execute the following SQL statement:

  ```
  create database test03 encoding 'UTF8' lc_collate 'zh_CN.utf8' lc_ctype 'zh_CN.utf8'  templ
  ate template0;
  \c test03
  select * from (values ('Alice'),('Tom')) as a(c1) order by c1 ;
  ```

  Information similar to the following output is returned:

  ```
     c1
  --------
   Alice
   Tom
  (2 rows)
  ```

  Note A Chinese character may have more than one pronunciation. For example, the Chongqing city in China may be encoded as the Zhongqing city. Proceed with caution if you want to configure a collation that is used to sort Chinese characters based on pronunciations.

Configure a rule to sort results in alphabetical order by using Greenplum

Greenplum does not allow you to specify collations for individual columns. Therefore, the sorting of results in alphabetical order is different in Greenplum.

You can use Greenplum to convert results among character sets. Then, you can sort the results in binary order. This allows you to obtain similar results that resemble results in alphabetical order. The following code snippet provides an example:

```
select * from (values ('Alice'), ('Tom')) t(id) order by byteain(textout(convert(id,'UTF8','EUC_CN')));
```

Information similar to the following output is returned:

```
   id
--------
 Alice
 Tom
(2 rows)
```

References

PostgreSQL 9.6.2 Documentation - Chapter 23. Localization

31.4. Insert, update, and delete multiple data records at a time

This topic describes how to insert, update, and delete multiple data records of an ApsaraDB RDS for PostgreSQL instance at a time. These operations can reduce the number of interactions between your RDS instance and your application and increase the data processing capability of your RDS instance.

You can insert multiple data records at a time by using one of the following four methods:

- Execute the `INSERT INTO ... SELECT` statement.

```
  postgres=# INSERT INTO tbl1 (id, info ,crt_time) SELECT GENERATE_SERIES(1,10000),'test',NOW();
  INSERT 0 10000
  postgres=# SELECT COUNT(*) FROM tbl1;
   count
  -------
   10001
  (1 row)
```

- Use the `VALUES(),(),...();` function.

```
postgres=# INSERT INTO tbl1 (id,info,crt_time) VALUES (1,'test',NOW()), (2,'test2',NOW()),
(3,'test3',NOW());
INSERT 0 3
```

- Run a `BEGIN; ...Multiple INSERT statements...; END;` transaction. This method allows you to include multiple INSERT statements in one transaction. This reduces the wait time on transaction commit, which in turn improves the performance of your RDS instance.

```
postgres=# BEGIN;
BEGIN
postgres=# INSERT INTO tbl1 (id,info,crt_time) VALUES (1,'test',NOW());
INSERT 0 1
postgres=# INSERT INTO tbl1 (id,info,crt_time) VALUES (2,'test2',NOW());
INSERT 0 1
postgres=# INSERT INTO tbl1 (id,info,crt_time) VALUES (3,'test3',NOW());
INSERT 0 1
postgres=# END;
COMMIT
```

- Use the COPY command. Compared with the INSERT statement, the COPY command is easier-to-use and can insert data at higher efficiency.

```
test03=# \d test
               Table "public.test"
  Column  |            Type             | Modifiers
----------+-----------------------------+-----------
 id       | integer                     | not null
 info     | text                        |
 crt_time | timestamp without time zone |
Indexes:
    "test_pkey" PRIMARY KEY, btree (id)
test03=# COPY test FROM stdin;
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 8    'test'  '2017-01-01'
>> 9    'test9' '2017-02-02'
>> \.
COPY 2
```

> **Note**
>
> The available COPY functions vary based on the language driver that you use. For more information, see the following documentation:
>
> - PostgreSQL JDBC Driver - JDBC 4.2 9.4.1209 API
> - PostgreSQL 9.6.2 Documentation - Functions Associated with the COPY Command

Update multiple data records at a time

```
test03=# UPDATE test SET info=tmp.info from (VALUES (1,'new1'),(2,'new2'),(6,'new6')) AS tmp
(id,info) WHERE test.id=tmp.id;
UPDATE 3
test03=# SELECT * FROM test;
 id |     info     |          crt_time
----+--------------+----------------------------
  3 | hello        | 2017-04-24 15:31:49.14291
  4 | digoal0123   | 2017-04-24 15:42:50.912887
  5 | hello digoal | 2017-04-24 15:57:29.622045
  1 | new1         | 2017-04-24 15:58:55.610072
  2 | new2         | 2017-04-24 15:28:20.37392
  6 | new6         | 2017-04-24 15:59:12.265915
(6 rows)
```

Delete multiple data records at a time

```
test03=# DELETE FROM test USING (VALUES (3),(4),(5)) AS tmp(id) WHERE test.id=tmp.id;
DELETE 3
test03=# SELECT * FROM test;
 id |  info   |          crt_time
----+---------+----------------------------
  1 | new1    | 2017-04-24 15:58:55.610072
  2 | new2    | 2017-04-24 15:28:20.37392
  6 | new6    | 2017-04-24 15:59:12.265915
```

If you want to clear a table, we recommend that you execute the TRUNCATE statement.

```
test03=# SET lock_timeout = '1s';
SET
test03=# TRUNCATE test;
TRUNCATE TABLE
test03=# SELECT * FROM test;
 id | info | crt_time
----+------+----------
(0 rows)
```

31.5. Locate SQL statements with the highest resource consumption

An ApsaraDB RDS for PostgreSQL instance is a large instance application. If your RDS instance processes a large number of requests, it consumes a large number of memory, CPU, I/O, and network resources. SQL optimization is an effective instance optimization method. To achieve the best results of SQL optimization, you must identify the SQL statements that consume the most resources, such as I/O resources.

Instance resources include CPU resources, memory resources, and I/O resources. You can use the pg_stat_statements plug-in to collect statistics on the consumed resources of your RDS instance and analyze the executed SQL statements to identify the SQL statements that consume the most CPU, memory, or I/O resources.

This topic describes how to create the pg_stat_statements plug-in, analyze the SQL statements that consume the most resources, and reset the statistics of resource consumption.

Run the following command to create the pg_stat_statements plug-in in your RDS instance:

```
CREATE EXTENSION pg_stat_statements;
```

Resource consumption statistics generated by the pg_stat_statements plug-in

You can query the resource consumption statistics from the view that is generated by the pg_stat_statements plug-in. Some filter conditions in SQL statements are replaced with variables in the pg_stat_statements plug-in to reduce duplicate statistics.

The view that is generated by the pg_stat_statements plug-in provides the following important information:

- Information about each SQL statement, including the number of times that the SQL statement is executed, the total execution duration, the shortest execution duration, the longest execution duration, the average execution duration, the execution duration variance, the total number of rows that are scanned, the total number of rows that are returned, and the total number of rows that are processed.
- Usage of the shared buffer, including the hit ratio, the miss ratio, the number of dirty data blocks that are generated, and the number of dirty data blocks that are evicted.
- Usage of the local buffer, including the hit ratio, the miss ratio, the number of dirty data blocks that are generated, and the number of dirty data blocks that are evicted.
- Usage of the temp buffer, including the number of dirty data blocks that are read and the number of dirty data blocks that are evicted.
- The duration of read operations and length of write operations on each data block in your RDS instance.

The following table lists the parameters in the resource consumption statistics that are generated by the pg_stat_statements plug-in:

Name	Type	Example	Description
userid	oid	pg_authid.oid	OID of user who executed the statement.
dbid	oid	pg_database.oid	OID of database in which the statement was executed.
queryid	bigint	None	Internal hash code, computed from the statement's parse tree.
query	text	None	Text of a representative statement.
calls	bigint	None	Number of times executed.

Name	Type	Example	Description
total_time	double precision	None	Total time spent in the statement, in milliseconds.
min_time	double precision	None	Minimum time spent in the statement, in milliseconds.
max_time	double precision	None	Maximum time spent in the statement, in milliseconds.
mean_time	double precision	None	Mean time spent in the statement, in milliseconds.
stddev_time	double precision	None	Population standard deviation of time spent in the statement, in milliseconds.
rows	bigint	None	Total number of rows retrieved or affected by the statement.
shared_blks_hit	bigint	None	Total number of shared block cache hits by the statement.
shared_blks_read	bigint	None	Total number of shared blocks read by the statement.
shared_blks_dirtied	bigint	None	Total number of shared blocks dirtied by the statement.
shared_blks_written	bigint	None	Total number of shared blocks written by the statement.
local_blks_hit	bigint	None	Total number of local block cache hits by the statement.
local_blks_read	bigint	None	Total number of local blocks read by the statement.
local_blks_dirtied	bigint	None	Total number of local blocks dirtied by the statement.

Name	Type	Example	Description
local_blks_written	bigint	None	Total number of local blocks written by the statement.
temp_blks_read	bigint	None	Total number of temp blocks read by the statement.
temp_blks_written	bigint	None	Total number of temp blocks written by the statement.
blk_read_time	double precision	None	Total time the statement spent reading blocks, in milliseconds (if track_io_timing is enabled, otherwise zero).
blk_write_time	double precision	None	Total time the statement spent writing blocks, in milliseconds (if track_io_timing is enabled, otherwise zero).

Analyze SQL statements that consume the most resources

- **SQL statements that consume the most I/O resources**
  - Run the following command to view the top five SQL statements that consume the most I/O resources in one call:

    ```
    SELECT userid::regrole, dbid, query FROM pg_stat_statements ORDER BY (blk_read_time+blk_write_time)/calls DESC LIMIT 5;
    ```

  - Run the following command to view the top five SQL statements that consume the most I/O resources in total:

    ```
    SELECT userid::regrole, dbid, query FROM pg_stat_statements ORDER BY (blk_read_time+blk_write_time) DESC LIMIT 5;
    ```

- **SQL statements that consume the most time**
  - Run the following command to view the top five SQL statements that consume the most time in one call:

    ```
    SELECT userid::regrole, dbid, query FROM pg_stat_statements ORDER BY mean_time DESC LIMIT 5;
    ```

  - Run the following command to view the top five SQL statements that consume the most time in total:

    ```
    SELECT userid::regrole, dbid, query FROM pg_stat_statements ORDER BY total_time DESC LIMIT 5;
    ```

- **SQL statements with the most severe response jitter**

  Run the following command to view the top five SQL statements with the most severe response jitter:

  ```
  SELECT userid::regrole, dbid, query FROM pg_stat_statements ORDER BY stddev_time DESC LIMIT 5;
  ```

- **SQL statements that consume the most shared memory resources**

  Run the following command to view the top five SQL statements that consume the most shared memory resources:

  ```
  SELECT userid::regrole, dbid, query FROM pg_stat_statements ORDER BY (shared_blks_hit+shared_blks_dirtied) DESC LIMIT 5;
  ```

- **SQL statements consume the most temporary space**

  Run the following command to view the top five SQL statements that consume the most temporary space:

  ```
  SELECT userid::regrole, dbid, query FROM pg_stat_statements ORDER BY temp_blks_written DESC LIMIT 5;
  ```

Reset resource consumption statistics

The pg_stat_statements plug-in collects accumulative statistics. To view the statistics over a specific period of time, you must query the snapshots of RDS the instance. For more information, see PostgreSQL AWR report (for ApsaraDB PgSQL).

You can run the following command to delete historical statistics on a regular basis:

```
SELECT pg_stat_statements_reset();
```

References

For more information, see PostgreSQL 9.6.2 Documentation - F.29. pg_stat_statements.

31.6. Logical subscription

This topic describes the logical subscription feature of ApsaraDB RDS for PostgreSQL. This feature supports quasi-real-time table-level one-way synchronization between multiple RDS for PostgreSQL instances. This feature is suitable for business scenarios such as data sharing, data aggregation, and data splitting.

If you deploy your business in more than one region, you can use logical subscriptions to share data among these regions. For example, you can share data from the data center that serves your business in a region to other regions. You can also aggregate data from other regions to the data center. This allows you to analyze and query all of your business data in real time.

An example of the logical subscription process is as follows:

Create a publication in the src database of the source RDS instance, publish the public.t1 table, and create a subscription named sub1_from_pub1 on the dst database of the destination RDS instance to subscribe to data from the public.t1 table.

For more information, see Principles and best practices of logical subscription.

Precautions

You can create logical subscriptions between two tables of a single RDS instance or between two RDS instances that reside in the same Virtual Private Cloud (VPC) and can be connected only by using internal endpoints.

Prerequisites

- Your RDS instance runs one of the following PostgreSQL versions:
  - PostgreSQL 14 (with standard or enhanced SSDs)
  - PostgreSQL 13 (with standard or enhanced SSDs)
  - PostgreSQL 12 (with standard or enhanced SSDs)
  - PostgreSQL 11 (with standard or enhanced SSDs)
  - PostgreSQL 10 (with standard or enhanced SSDs)
- The **wal_level** parameter is set to **logical** for your RDS instance. You can reconfigure this parameter on the **Parameters** page in the ApsaraDB for RDS console. After you reconfigure this parameter, you must restart your RDS instance to make the new value take effect. Restarting your RDS instance will terminate all its connections. Make appropriate service arrangements before you restart your RDS instance.
- If you want to create logical subscriptions between two RDS instances that reside in the same VPC, the Classless Inter-Domain Routing (CIDR) block that contains the IP address of the VPC to which one RDS instance belongs is added to an IP address whitelist of the other RDS instance. For example, you can add 172.16.0.0/16 to an IP address whitelist. For more information, see Configure a whitelist for an RDS PostgreSQL instance.
- The account that you use has the permissions of the rds_superuser role. The account can be the privileged account of your RDS instance. The account can also be a standard account that you create by using the privileged account and the `create role xxx with superuesr` command.

Procedure

Follow these steps to create a publication in the source database of the source RDS instance:

1. Connect to the source RDS instance. For more information, see Connect to an RDS PostgreSQL instance.
2. Execute the following statement to create a publication in the source database:

   ```
   CREATE PUBLICATION <The name of the publication> FOR TABLE <The name of the source table>
   ;
   ```

   Example:

   ```
   create publication pub1 for table public.t1;
   ```

   > **Note**
   > - You can publish only persistent base tables. For more information, see CREATE PUBLICATION.
   > - You can run the `select * frompg_publication;` command to view the existing publications of the database that stores the specified source table.

Follow these steps to create a subscription in the destination database of the destination RDS instance:

1. Connect to the destination RDS instance. For more information, see Connect to an RDS PostgreSQL instance.

2. Execute the following statement. In this example, the source and destination databases are on the same RDS instance. If you want to create a subscription between two RDS instances, go to step 3.

```
select * from pg_create_logical_replication_slot('<The name of the subscription>','pgoutput');
```

3. Execute the following statement to create a subscription in the destination database:

```
CREATE SUBSCRIPTION <The name of the subscription>
    CONNECTION '<The information that is required to log on to the source database>'
    PUBLICATION <The name of the publication in the source database>;
```

Example:

```
create subscription sub1_from_pub1
connection 'host=pgm-xxxxx.pgsql.singapore.rds.aliyuncs.com port=3433 user=test password=xxxxx dbname=src'
publication pub1 with (enabled, create_slot, slot_name='sub1_from_pub1');
```

> **Note**
>
> - The information that is required to log on to the source database is in the following format: `host=<The internal endpoint of the source RDS instance> port=<The internal port of the source RDS instance> user=<The username of the account that has permissions on publications on the source RDS instance> password=<The password of the account that has permissions on publications on the source RDS instance> dbname=<The name of the source database>`.
> - If the source and destination databases are on the same RDS instance, you must set the `host` parameter to localhost and the `create_slot` parameter to false. You can query the port number by running the `show port` command. In most cases, the port number is 3002.
> - You can view the subscriptions of your entire database system by running the `select * from pg_subscription;` command.
> - You can append a subscription parameter to the name of the publication by using a WITH clause. For more information, see CREATE SUBSCRIPTION.

31.7. Use event triggers to implement the DDL recycle bin, firewall, and incremental synchronization features

ApsaraDB RDS for PostgreSQL allows you to use event triggers to implement features such as the DDL recycle bin, firewall, incremental synchronization. This helps you reduce maintenance costs and protect data security.

Prerequisites

The RDS instance runs PostgreSQL 10, 11 or 12, and is equipped with standard or enhanced SSDs.

Context

You can create DDL recycle bin and firewall policies based on event triggers to ensure database security.

- Prevent against risky operations, such as DROP TABLE, DROP INDEX, and DROP DATABASE.
- Retrieve data from the recycle bin if a table is deleted by mistake.

You can use two event triggers, pg_get_ddl_command and pg_get_ddl_drop, to collect and save DDL statements in the dts_audit.dts_tb_ddl_command table. The event triggers are implemented by using the pg_func_ddl_command() function. The structure of the dts_audit.dts_tb_ddl_command table is as follows:

```
     Column      |            Type             | Collation | Nullable |      Default       |
Storage  | Stats target | Description
-----------------+-----------------------------+-----------+----------+--------------------+-
---------+--------------+-------------
 event           | text                        |           |          |                    |
extended |              |
 tag             | text                        |           |          |                    |
extended |              | Command tag
 classid         | oid                         |           |          |                    |
plain    |              | OID of catalog the object belonged in
 objid           | oid                         |           |          |                    |
plain    |              | OID the object had within the catalog
 objsubid        | integer                     |           |          |                    |
plain    |              | Object sub-id (e.g. attribute number for columns)
 object_type     | text                        |           |          |                    |
extended |              | Type of the object
 schema_name     | text                        |           |          |                    |
extended |              | Name of the schema the object belonged in, if any; otherwise NULL.
No quoting is applied.
 object_identity | text                        |           |          |                    |
extended |              | Text rendering of the object identity, schema-qualified.
 is_extension    | boolean                     |           |          |                    |
plain    |              | True if the command is part of an extension script
 query           | text                        |           |          |                    |
extended |              | sql text
 username        | text                        |           |          | CURRENT_USER       |
extended |              |
 datname         | text                        |           |          | current_database() |
extended |              |
 client_addr     | inet                        |           |          | inet_client_addr() |
main     |              |
 crt_time        | timestamp without time zone |           |          | now()              |
plain    |              |
```

The required CREATE statements are as follows:

```
CREATE SCHEMA IF NOT EXISTS dts_audit;
CREATE TABLE IF NOT EXISTS dts_audit.dts_tb_ddl_command ( event text,
tag text, classid oid, objid oid, objsubid int,
object_type text, schema_name text, object_identity text, is_extension bool, query text,
username text default current_user, datname text default current_database(),
client_addr inet default inet_client_addr(), crt_time timestamp default now()
);
```

The following section provides examples of how to implement the DDL recycle bin, firewall, incremental synchronization. You can modify the code as needed.

DDL recycle bin

1. Execute the following statements to create the required table, function, and triggers:

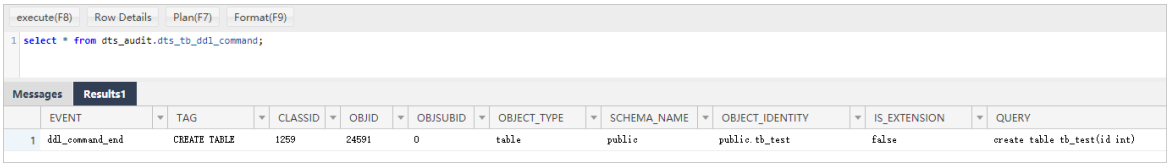
```
/* external/rds_ddl_pulication/rds_ddl_pulication--1.0.sql */
--create schema
CREATE SCHEMA IF NOT EXISTS dts_audit;
--create table for ddl record
CREATE TABLE IF NOT EXISTS dts_audit.dts_tb_ddl_command ( event text,
tag text, classid oid, objid oid, objsubid int,
object_type text, schema_name text, object_identity text, is_extension bool, query text,
username text default current_user, datname text default current_database(), client_addr
inet default inet_client_addr(), crt_time timestamp default now()
);
-- create function for event triggers
create or replace function dts_audit.dts_func_ddl_command() returns event_trigger as $$
declare v1 text;
is_superuser bool = false;
r record;
begin
    -- we don't record  ddl command from superusers
    select u.rolsuper into is_superuser from pg_catalog.pg_roles u where u.rolname = SESS
ION_USER;
    if is_superuser then
        return;
    end if;
    select query into v1 from pg_stat_activity where pid=pg_backend_pid();
    -- RAISE NOTICE 'ddl event:%, command:%', tg_event, tg_tag;
    -- NB:since ddl_command_end cannot collect the details of the drop statement, we use
sql_drop
    if TG_EVENT='ddl_command_end' then
        SELECT * into r FROM pg_event_trigger_ddl_commands();
        if r.classid > 0 then
            insert into dts_audit.dts_tb_ddl_command(event, tag, classid, objid, objsubid
, object_type, schema_name, object_identity, is_extension, query)
            values(TG_EVENT, TG_TAG, r.classid, r.objid, r.objsubid, r.object_type, r.sch
ema_name, r.object_identity, r.in_extension, v1);
         end if;
    end if;
    if TG_EVENT='sql_drop' then
        -- To avoid repeated collection, we filtered 'ALTER TABLE' and 'ALTER FOREIGN TAB
LE'
        if TG_TAG != 'ALTER TABLE' and TG_TAG != 'ALTER FOREIGN TABLE' then
            SELECT * into r FROM pg_event_trigger_dropped_objects();
            insert into dts_audit.dts_tb_ddl_command(event, tag, classid, objid, objsubid
, object_type, schema_name, object_identity, is_extension, query)
            values(TG_EVENT, TG_TAG, r.classid, r.objid, r.objsubid, r.object_type, r.sch
ema_name, r.object_identity, 'f', v1);
        end if;
    end if;
end;
$$ language plpgsql strict;
-- ddl_command_end event trigger
```

```
CREATE EVENT TRIGGER pg_get_ddl_command on ddl_command_end EXECUTE PROCEDURE dts_audit.dts_func_ddl_command();
-- pg_get_ddl_drop event trigger
CREATE EVENT TRIGGER pg_get_ddl_drop on sql_drop EXECUTE PROCEDURE dts_audit.dts_func_ddl_command();
-- grant privileges to all user
GRANT USAGE ON SCHEMA dts_audit TO PUBLIC;
GRANT SELECT, INSERT ON TABLE dts_audit.dts_tb_ddl_command TO PUBLIC;
```

> **Note** After you execute the preceding statements, DDL statements are recorded in the dts_audit.dts_tb_ddl_command table.

2. Execute a DDL statement to test whether it is recorded in the dts_audit.dts_tb_ddl_command table.

```
create table tb_test(id int);
select * from dts_audit.dts_tb_ddl_command;
```



DDL firewall

You can create event triggers as needed and use the ddl_command_start event to block the execution of specific DDL statements.

1. Create a trigger function.

```
CREATE OR REPLACE FUNCTION abort1()
  RETURNS event_trigger
 LANGUAGE plpgsql
  AS $$
BEGIN
  if current_user = 'test1' then
    RAISE EXCEPTION 'event:%, command:%', tg_event, tg_tag;
  end if;
 END;
$$;
```

2. Create a trigger to block DDL statements that create or delete tables.

```
create event trigger b on ddl_command_start when TAG IN ('CREATE TABLE', 'DROP TABLE') execute procedure abort1();
```

3. Log on to the RDS instance as user test1 and try to create a table.

> **Note** The DDL statement is blocked.

Incremental synchronization

On the publication side, executed DDL statements are stored in the dts_audit.dts_tb_ddl_command table. You can synchronize the records from the publication side to the subscription side.

1. Create a publication on the publication side.

```
CREATE PUBLICATION my_ddl_publication FOR TABLE ONLY dts_audit.dts_tb_ddl_command;
```

2. Create the same table on the subscription side.

```
CREATE SCHEMA IF NOT EXISTS dts_audit;
CREATE TABLE IF NOT EXISTS dts_audit.dts_tb_ddl_command ( event text,
tag text, classid oid, objid oid, objsubid int,
object_type text, schema_name text, object_identity text, is_extension bool, query text,
username text, datname text, client_addr inet , crt_time timestamp );
```

3. Create a subscription on the subscription side.

```
CREATE SUBSCRIPTION my_ddl_subscriptin CONNECTION 'host=*** port=*** user=*** password=**
*  dbname=**' PUBLICATION my_ddl_publication;
```

> **Note** Make sure that the **wal_level** parameter is set to **logical** for your RDS instance. You can change the parameter value on the Parameters page in the ApsaraDB for RDS console and restart the instance for the change to take effect. For more information, see Logical subscription.

Example:

```
CREATE SUBSCRIPTION my_ddl_subscriptin CONNECTION 'host=pgm-bpxxxxx.pg.rds.aliyuncs.com p
ort=1433 user=test1 password=xxxxx  dbname=testdb1' PUBLICATION my_ddl_publication;
```

4. Create a trigger for the dts_audit.dts_tb_ddl_command table on the subscription side to perform incremental synchronization of DDL statements.

31.8. Configure automatic failover and read/write splitting

You can use libpq or Java Database Connectivity (JDBC) in PostgreSQL to configure automatic failover and read/write splitting.

Context

In PostgreSQL 10 and later versions, libpq supports failover and JDBC supports failover and load balancing at the driver layer.

- libpq is a C API to PostgreSQL. libpq is a set of library functions that allow client programs to pass queries to the PostgreSQL backend server and to receive the results of these queries.
- JDBC is a Java API to define how client programs access databases. In PostgreSQL, JDBC supports failover and load balancing.

Use libpq to implement automatic failover and read/write splitting

You can use libpq functions to connect to multiple databases. If one database becomes faulty, services are automatically switched to other available databases.

Command:

```
postgresql://[user[:password]@][netloc][:port][,...][/dbname][? param1=value1&...]
```

Example:

In the following example, libpq is connected to a primary ApsaraDB RDS for PostgreSQL database and two of its read-only databases. If at least one database is available, read requests do not fail.

```
postgres://pgm-bpxxx1.pg.rds.aliyuncs.com:3433,pgm-bpxxx2.pg.rds.aliyuncs.com:3433,pgm-bpxxx3
.pg.rds.aliyuncs.com:3433/postgres? target_session_attrs=any
```

Parameters:

target_session_attrs: specifies the type of databases to which libpq connects. Valid values:

- **any**: libpq randomly connects to a database. This is the default value. If the connection is interrupted because the database is faulty, libpq connects to another database to implement failover.
- **read-write**: libpq only connects to a database that supports both read and write. Specifically, libpq connects to the databases in sequence. If a database does not support read and write, libpq disconnects from it and connects to the next database and so on until libpq connects to a database that supports read and write.

For more information about how to use libpq and configure required parameters, see Connection Strings.

You can use the pg_is_in_recovery() function in your application to determine whether the connected database is primary or read-only. This allows you to achieve failover and read/write splitting. Examples:

- Python

```
$ cat pg_conn.py
import psycopg2
conn = psycopg2.connect(database="postgres",host="pgm-bpxxx1.pg.rds.aliyuncs.com,pgm-bpxxx2
.pg.rds.aliyuncs.com,pgm-bpxxx3.pg.rds.aliyuncs.com", user="testxxx", password="xxxxxx", po
rt="3433", target_session_attrs="read-write")
cur = conn.cursor()
cur.execute("select pg_is_in_recovery(), pg_postmaster_start_time()")
row = cur.fetchone()
print "recovery =",row[0]
print "time =",row[1]
$ python pg_conn.py
recovery = False
time = 2020-07-09 15:33:57.79001+08
```

- PHP

```
# cat pg_conn.php
<? php
$conn = pg_connect("host=pgm-bpxxx1.pg.rds.aliyuncs.com,pgm-bpxxx2.pg.rds.aliyuncs.com,pgm-bpxxx3.pg.rds.aliyuncs.com port=3433 dbname=postgres user=testxxx password=xxxxxx target_session_attrs=read-write") or die("Could not connect");
$status = pg_connection_status($conn);
if ($status === PGSQL_CONNECTION_OK) {
print "Connection status ok\n";
} else {
print "Connection status bad\n";
}
$sql = pg_query($conn, "select pg_is_in_recovery()");
while ($row = pg_fetch_row($sql)) {
echo "Recovery-status: $row[0]\n";
}
? >
$ php -f pg_conn.php
Connection status ok
Recovery-status: f
Server: xxx.xxx.xx.xx
```

Use JDBC to implement automatic failover and read/write splitting

You can specify multiple databases separated with commas (,) in the connection URL. The JDBC driver attempts to connect to the databases in sequence until the connection is successful. If all connection attempts fail, an error message is returned.

Command:

```
jdbc:postgresql://node1,node2,node3/accounting? targetServerType=preferSlave&loadBalanceHosts=true
```

Example:

```
jdbc:postgresql://pgm-bpxxx1.pg.rds.aliyuncs.com:3433,pgm-bpxxx2.pg.rds.aliyuncs.com:3433,pgm-bpxxx3.pg.rds.aliyuncs.com:3433/accounting? targetServerType=preferSlave&loadBalanceHosts=true
```

Parameters:

- **targetServerType**: specifies the type of databases to which the JDBC driver connects. Valid values:
  - any: The JDBC driver connects to any database.
  - master: The JDBC driver only connects to the primary database.
  - slave: The JDBC driver only connects to the secondary database.
  - preferSlave: The JDBC driver connects to the secondary database in priority. If no secondary database is available, the JDBC driver connects to the primary database.

  > **Note** A primary database supports write operations, and a secondary database does not.

- **loadBalanceHosts**: specifies whether to randomly connect to the databases. Valid values:
  - False: The databases are connected in the sequence specified in the command. This is the default value.

- True: The databases are randomly connected.

To implement read/write splitting, you can configure two data sources. Set **targetServerType** to **master** for the first data source and set **targetServerType** to **preferSlave** for the second. Then, specify that write operations are performed on the first data source and read operations on the second. If you want to determine whether a connected database is primary or secondary (read-only), use the pg_is_in_recovery() function. This allows you to achieve failover and read/write splitting.

32.Application Solutions

32.1. Real-time precision marketing (user selection)

This topic describes how to use ApsaraDB RDS for PostgreSQL 12 to implement precision marketing on target users in real time.

Prerequisites

- Create an ApsaraDB RDS for PostgreSQL 12 instance.
- Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
- Create an account on an ApsaraDB RDS for PostgreSQL instance.
- Create a database on an ApsaraDB RDS for PostgreSQL instance.

Context

Real-time precision marketing is required in almost all industries, such as the Internet, gaming, and education. Real-time precision marketing allows you to generate and filter user profiles to rapidly locate target users in different industries.

- E-commerce industry: Merchants push advertisements or deliver suitable marketing activities to target users based on user characteristics.
- Gaming industry: Merchants present gifts to players based on player characteristics to improve player activation.
- Education industry: Teachers push targeted exercises to students based on student characteristics to help students learn.
- Search, portal, and video website industries: Merchants push specific contents to users based on user concerns.

These industries have the same pain points:

- The huge volume of data requires a large number of computations.
- The large number of user tags and fields consume large amounts of storage space.
- The number of fields in a database may exceed the limit. Typically, a database can only contain up to 1,000 fields.
- A database that supports inverted indexes can use an array instead of multiple fields to store tags. However, not all databases support inverted indexes.
- The use of an array instead of multiple fields to store tags and the support for inverted indexes require large amounts of storage space.
- A wide range of combinations are used for selection conditions. The use of a single index for each field without a fixed index also requires large amounts of storage space.
- High performance is required for quick responses of real-time marketing.
- Real-time updates of user data and profiles are required to select users with precision. For example, assume that a user browses mobile phones and makes an order at that night. If merchants update user data the next day, their selected target user will be imprecise.

Common services such as ApsaraDB RDS for MySQL have limited resources and cannot handle target user selection in real time.

You can choose one of the following solutions to implement precision marketing in real time based on ApsaraDB RDS for PostgreSQL.

- Solution 1
- Solution 2
- Solution 3

Solution 1

> **Note** Solution 1 is supported for both PostgreSQL and MySQL.

The following table schema is used:

```
KEY: the user ID
Tag 1:
Tag 2:
... Tag N:
```

The following index is used:

```
One index for each tag field
```

The following search method is used:

```
Combination of AND, OR, and NOT where Tag a and Tag b and ...
```

Disadvantages

- The large number of user tags and fields consume large amounts of storage space.
- The number of fields in a database may exceed the limit. Typically, a database can only contain up to 1,000 fields. To solve this limit problem, you can use a many-to-many structure to maintain a single record for each tag.
- A wide range of combinations are used for selection conditions. The use of a single index for each field without a fixed index also requires large amounts of storage space.
- A large amount of data must be updated when new group tags are added.
- The query performance is poor.

Procedure

1. Create a group table, with each record representing a group of people. Example:

   ```
   create table t_tag_dict (
   tag int primary key,   -- Tag (group) ID info text,  -- Group description crt_time timest
   amp  -- Time
   );
   ```

2. Create 100,000 group tags. Example:

```
insert into t_tag_dict values (1, 'Male', now());
insert into t_tag_dict values (2, 'Female', now());
insert into t_tag_dict values (3, 'Older than 24 years old', now());
-- ...
 insert into t_tag_dict
select generate_series(4,100000), md5(random()::text), clock_timestamp();
```

3. Create a user profile table, with each record of a user representing a tag of the user. Example:

```
create table t_user_tag (
uid int8,    -- User ID tag int,            -- Tag (group) of the user mod_time timestamp,
-- Time
primary key (tag,uid)
);
```

4. Set 64 random tags for each of the 10 million male and 10 million female users to generate a total of 1.28 billion records. Example:

```
create or replace function gen_rand_tag(int,int) returns setof int as
$$
  select case when random() > 0.5 then 1::int else 2::int end as tag
  union all
  select ceil(random()*$1)::int as tag from generate_series(1,$2);
$$ language sql strict volatile;
 insert into t_user_tag
select uid, gen_rand_tag(100000,63) as tag, clock_timestamp()
from generate_series(1,20000000) as uid on conflict (uid,tag) do nothing;
 -- You can also use the following method to import tags: create sequence seq;
 vi test.sql
insert into t_user_tag
select uid, gen_rand_tag(100000,63) as tag, clock_timestamp()
from nextval('seq'::regclass) as uid
on conflict(tag,uid) do nothing;
 pgbench -M prepared -n -r -P 1 -f ./test.sql -c 50 -j 50 -t 400000
```

5. Query users who match with tags 1 and 3. Example:

```
1. Number of users select count(*) from
(
select uid from t_user_tag where tag=1
intersect
select uid from t_user_tag where tag=3
) t;
-- Time: 1494.789 ms (00:01.495)
 2. User IDs select uid from t_user_tag where tag=1
intersect
select uid from t_user_tag where tag=3;
-- Time: 3246.184 ms (00:03.246)
```

6. Query users who match with tags 1, 3, 10, or 200. Example:

```
1. Number of users select count(*) from
(
select uid from t_user_tag where tag=1
union
select uid from t_user_tag where tag=3
union
select uid from t_user_tag where tag=10
union
select uid from t_user_tag where tag=200
) t;
-- Time: 3577.714 ms (00:03.578)
 2. User IDs select uid from t_user_tag where tag=1
union
select uid from t_user_tag where tag=3
union
select uid from t_user_tag where tag=10
union
select uid from t_user_tag where tag=200;
-- Time: 5682.458 ms (00:05.682)
```

Solution 2

Note Solution 2 is supported only for PostgreSQL. MySQL does not support arrays or inverted indexes.

The following table schema is used:

```
KEY: the user ID VALUES: the tag array
```

The following index is used:

```
Tag array field: generalized inverted index (GIN)
```

The following search method is used:

```
AND, OR, and NOT
where VALUES @> array[Tags] -- AND
where VALUES && array[Tags] -- OR where not VALUES @> array[Tags] -- NOT
```

Disadvantages

- A database that supports inverted indexes can use an array instead of multiple fields to store tags. However, not all databases support inverted indexes.
- The use of an array instead of multiple fields to store tags and the support for inverted indexes require large amounts of storage space.
- A large amount of data must be updated when new group tags are added.

Procedure

1. Create a group table, with each record representing a group of people. Example:

```
create table t_tag_dict (
tag int primary key,   -- Tag (group) ID info text,  -- Group description crt_time timest
amp  -- Time
);
```

2. Create 100,000 group tags. Example:

```
insert into t_tag_dict values (1, 'Male', now());
insert into t_tag_dict values (2, 'Female', now());
insert into t_tag_dict values (3, 'Older than 24 years old', now());
-- ...
 insert into t_tag_dict
select generate_series(4,100000), md5(random()::text), clock_timestamp();
```

3. Create a user profile table, with the single record of each user representing an array of tags of the user. Example:

```
create table t_user_tags (
uid int8 primary key,   -- User ID tags int[],            -- Array of user tags (groups)
mod_time timestamp     -- Time
);
```

4. Create a function to generate a random array of tags. Example:

```
create or replace function gen_rand_tags(int,int) returns int[] as $$
  select array_agg(ceil(random()*$1)::int) from generate_series(1,$2);
$$ language sql strict;
```

5. Randomly select eight from 100,000 tags. Example:

```
select gen_rand_tags(100000, 8);
                  gen_rand_tags
-----------------------------------------------------
 {43494,46038,74102,25308,99129,40893,33653,29690}
(1 row)
```

6. Set 64 random tags for each of the 10 million male and 10 million female users. Example:

```
insert into t_user_tags
select generate_series(1,10000000),
array_append(gen_rand_tags(100000, 63),1), now();
 insert into t_user_tags
select generate_series(10000001,20000000),
array_append(gen_rand_tags(100000, 63),2), now();
```

7. Create an inverted index for the group tag fields. Example:

```
create index idx_t_user_tags_1 on t_user_tags using gin (tags);
```

8. Query users who match with tags 1 and 3. Example:

```
1. Number of users select count(uid) from t_user_tags where tags @> array[1,3];
 2. User IDs select uid from t_user_tags where tags @> array[1,3];
```

9. Query users who match with tags 1, 3, 10, or 200. Example:

```
1. Number of users select count(uid) from t_user_tags where tags && array[1,3,10,200];
 2. User IDs select uid from t_user_tags where tags && array[1,3,10,200];
```

Solution 3

> **Note** Solution 3 is supported only for PostgreSQL. MySQL does not support bitmaps.

Solution 3 uses the roaringbitmap plug-in to quickly query data. For more information, see Use the roaringbitmap plug-in.

The following table schema is used:

```
KEY: the tag ID
VALUES: the user bitmap
```

The following index is used:

```
Tag ID field: B-tree index
```

The following search method is used:

```
Aggregate bitmap: AND, OR, and NOT and_agg(bitmaps) where KEY in (Tags) -- AND
or_agg(bitmaps) where KEY in (Tags) -- OR
except(bitmap1,bitmap2) -- NOT
```

- Advantages
  - Only a small amount of space is required to store tables.
  - Only a small amount of space is required to store indexes. Only one B-tree index is required to store tags. Typically, the number of tags is less than one million.
  - When a new group tag is added, only a single group bitmap record needs to be added and updates do not consume large amounts of data.
  - The query performance is excellent.
- Disadvantages
  - The maximum length of a bitmap is 1 GB. If the number of users exceeds the maximum length, use an offset as follows:

    ```
    offset0_bitmap, offset1gb_bitmap, ...
    ```

  - User IDs must be numbers and consecutive numbers are recommended. A mapping table is required if user IDs are not numbers.

Procedure

> **Note**
> - When a user ID exceeds 4 billion (INT4), you can use an offset to convert the user ID. For more information about the conversion method, see Troubleshooting for UID overflow.
> - For more information about how to use roaringbitmap, see pg_roaringbitmap.

1. Install the roaringbitmap plug-in. Example:

   ```
   create extension roaringbitmap;
   ```

2. Create a bitmap table that contains user tags. Example:

```
create table t_tag_users (
  tagid int primary key,   -- User tag (group) ID  uid_offset int,          -- Convert th
e user ID from INT8 to INT4.  userbits roaringbitmap,     -- Bitmap of user IDs  mod_time
timestamp       -- Time
);
```

3. Insert data to generate a bitmap table that contains user ID tags. Example:

```
insert into t_tag_users
select tagid, uid_offset, rb_build_agg(uid::int) as userbits from
(
select
  unnest(tags) as tagid,
  (uid / (2^31)::int8) as uid_offset,
  mod(uid, (2^31)::int8) as uid
from t_user_tags
) t
group by tagid, uid_offset;
```

4. Query users who match with tags 1 and 3. Example:

```
1. Number of users select sum(ub) from
(
select uid_offset,rb_and_cardinality_agg(userbits) as ub
from t_tag_users
where tagid in (1,3)
group by uid_offset
) t;
 2. User IDs select uid_offset,rb_and_agg(userbits) as ub
from t_tag_users
where tagid in (1,3)
group by uid_offset;
```

5. Query users who match with tags 1, 3, 10, or 200. Example:

```
1. Number of users select sum(ub) from
(
select uid_offset,rb_or_cardinality_agg(userbits) as ub
from t_tag_users
where tagid in (1,3,10,200)
group by uid_offset
) t;
 2. User IDs select uid_offset,rb_or_agg(userbits) as ub
from t_tag_users
where tagid in (1,3,10,200)
group by uid_offset;
```

Comparison between solutions

Item	Solution 1 (MySQL and PostgreSQL)	Solution 2 (PostgreSQL)	Solution 3 (PostgreSQL)	Improvement of solution 3 than solution 1
Time used for AND queries of user selection	1.5 seconds	0.042 seconds	0.0015 seconds	99900%
Time used for OR queries of user selection	3.6 seconds	3 seconds	0.0017 seconds	211665%
Space usage (table)	63,488 MB	3,126 MB	1,390 MB	4467%
Space usage (index)	62,464 MB	3,139 MB	2 MB	3123100%
Time used for index creation	N/A	20 minutes	Extremely fast (about 0 seconds)	N/A

Note ApsaraDB RDS for MySQL 8.0 and ApsaraDB RDS for PostgreSQL 12 instances used in these solutions have the following specifications: 8-core CPU, 32 GB memory, and Enhanced SSDs of 1,500 GB.

Summary

ApsaraDB RDS for PostgreSQL 12 allows you to use the roaringbitmap plug-in to generate, compress, and parse bitmap data. You can perform bitwise aggregate operations such as AND, OR, NOT, and XOR to implement precision marketing on target users in real time when ten millions of tags are used for hundreds of millions of users.

Compared with the MySQL-based solution, the PostgreSQL-based solution is more cost-effective.

32.2. Image recognition, face recognition, similarity-based retrieval, and similarity-based audience spotting

This topic describes how to use the PASE plug-in of ApsaraDB RDS for PostgreSQL to recognize images and faces, retrieve images, and spot audience members at low costs but high efficiency.

Prerequisites

Your RDS instance runs PostgreSQL 11.

Note The feature described in this topic will be supported for all ApsaraDB for RDS instances that run PostgreSQL 11 and later.

Context

The PASE plug-in supports two popular vector indexing algorithms: IVFFlat and HNSW. This makes it competitive in sectors such as Internet, new retail, and public transportation. ApsaraDB RDS for PostgreSQL will continue to integrate more popular vector indexing algorithms in the industry.

> **Note** For more information about how to use the PASE plug-in, see Use the PASE plug-in for efficient vector search.

Comparison between Solution 1 and Solution 2

Comparison item	Solution 1 (MySQL)	Solution 2 (PostgreSQL)	Competitive advantage of Solution 2 over Solution 1
Single-query response speed	61.45 seconds	0.0025 seconds	2,457,900%
Concurrent queries per second	0.055330	1,056	1,908,449%

> **Note** Both the ApsaraDB RDS for MySQL 8.0 instance tested in Solution 1 and the ApsaraDB RDS for PostgreSQL 11 instance tested in Solution 2 use the following specifications to store and process 1,000,000 images: 4-core CPU, 8 GB of memory, and 1,500 GB of storage space provided by enhanced SSDs.

Challenges

Common relational databases such as MySQL do not support vector-based retrieval. They must traverse all data and return the obtained results to the application layer. The application layer then computes the results to respond to queries. The queries are slow and consume a large number of network bandwidth resources. Even if such databases can be optimized to support operators used for vector-based retrieval, they still need to traverse all data and cannot process highly concurrent queries, because they do not support vector indexing.

If the application layer implements image vector computing, it must load all the data from your database, which is slow. In addition, the application layer cannot load new data from updated images in real time, recognize images, or filter images based on combinations of criteria.

Scenarios

ApsaraDB RDS for PostgreSQL is an optimal solution that is designed to precisely search for similar images at a high speed based on a specific image and a combination of criteria.

- In the smart building sector, ApsaraDB RDS for PostgreSQL scans the faces of employees who enter a building. If it recognizes an employee, it automatically clocks that employee in and sets the elevator to stop at the floor where that employee works.
- In the smart hotel sector, ApsaraDB RDS for PostgreSQL scans the faces of customers who enter a hotel. If it recognizes a customer, it automatically checks that customer in and offers exclusive services to that customer based on their customer membership level.
- In the e-commerce sector, ApsaraDB RDS for PostgreSQL searches for similar items based on the image you specify.
- In the education sector, ApsaraDB RDS for PostgreSQL scans the faces of students to record their behavior in class (for example, whether they are napping, distracted, fidgeting, or raising hands).

- In the public transportation sector, ApsaraDB RDS for MySQL recognizes drivers who violate transportation regulations.
- In the new retail sector, ApsaraDB RDS for PostgreSQL recognizes the faces of customers who enter a retail shop. If it recognizes a customer as a member registered with the retail shop, it sends reminders of new arrivals, provides guidance, and offers exclusive services to that customer.
- In the public transportation sector, ApsaraDB RDS for PostgreSQL enables face scan payments.
- In the gaming sector, ApsaraDB RDS for PostgreSQL facilitates virtual reality-related games.

Before you begin

- Create an RDS instance that runs PostgreSQL 11.
- Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.
- Create an account on an ApsaraDB RDS for PostgreSQL instance.
- Create a database on an ApsaraDB RDS for PostgreSQL instance.

Solution 1

- Characteristics
  - Your database only stores image vectors but does not compute them.
  - The application layer computes image vectors.
- Demerits
  - Your database cannot index or filter vectors.
  - The application layer must load all the data from your database, which is slow. In addition, the application layer cannot load new data from updated images in real time.
  - Your database cannot recognize images or filter images based on combinations of criteria. Images can only be recognized and filtered at the application layer. However, the application layer cannot process highly concurrent queries due to the large number of network transmission records.

Procedure

1. Create a test table. Example:

```
create table if not exists t_vec_80(
    id serial PRIMARY KEY,  -- The primary key of the table.
    c1 int,   -- The other attribute fields of the table.
    c2 int,
    c3 text,
    c4 timestamp,
    vec float4[]    -- The data type of the vectors for image feature values.
);
```

2. Create a function to generate random vectors. This function is used to simulate image feature values. You must specify the feature value of the image you want to query. Example:

```
create or replace function gen_float4_arr(int,int) returns float4[] as $$
  select array_agg(trunc(random()*$1)::float4) from generate_series(1,$2);
$$ language sql strict volatile;
```

3. Write 1,000,000 random vectors. Example:

```
insert into t_vec_80 (c1,c2,c3,c4,vec)
select random()*100, random()*100000, md5(random()::text), clock_timestamp(),
gen_float4_arr(10000, 80)
from generate_series(1,1000000);
```

Note View the write result. Example:

```
select * from t_vec_80 limit 3;
-[ RECORD 1 ]-------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------
id  | 1
c1  | 99
c2  | 7428
c3  | 9b74e40ab38ed4f41b5d50b8eedf8b72
c4  | 2020-02-27 15:36:56.895773
vec | {6469,3787,5852,1642,2798,7728,1527,6990,7399,3460,7802,7682,8102,6499,3428,7687
,567,8894,8144,1685,6139,9549,3613,1714,721,9099,4218,1930,9031,4961,3966,5501,8748,98
18,7143,1546,7547,8671,8536,4946,2132,6338,2629,234,2838,6057,7922,3405,4951,6066,5091
,1091,5615,8704,2805,6336,7804,7024,8266,6836,1985,2233,2337,733,2051,9481,2280,9598,8
152,816,4545,285,7155,7174,519,9993,3232,8441,3399,8183}
-[ RECORD 2 ]-------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------
id  | 2
c1  | 45
c2  | 84908
c3  | a48d421b772486121ef520eb3e285f95
c4  | 2020-02-27 15:36:56.896329
vec | {123,7195,2080,6460,5000,9104,4727,1836,1089,6960,4174,1823,9012,3656,4103,8611,
1808,4920,3157,2094,2076,332,2613,2070,3564,1055,5469,1748,5563,3960,1023,5686,1156,31
03,2147,6156,2208,6874,7993,3298,3834,2167,5121,2847,5823,9225,1458,7632,4145,4615,972
6,6222,4947,2340,8292,8511,3395,3762,259,8958,7722,1282,4644,8878,4386,6792,5035,6594,
3666,3028,9892,7501,5196,5014,348,1019,4239,1806,8652,8384}
-[ RECORD 3 ]-------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
-------------------------------------------------------------
id  | 3
c1  | 64
c2  | 83785
c3  | ea856c452399648fd29b0e0383a169a5
c4  | 2020-02-27 15:36:56.896395
vec | {1369,718,2899,9880,4113,6661,140,3071,4383,1422,7716,3262,5808,4509,8298,2403,8
175,1326,2295,5676,6523,7309,6024,7542,1549,7831,6194,9934,4253,4573,4541,5622,5291,74
40,5503,9405,4101,5643,2477,8485,7066,194,1748,2875,4703,46,5278,2878,1373,7574,8555,7
896,4884,4580,5439,6433,2411,1633,6367,6664,6207,909,2286,1498,8349,7789,903,2451,3433
,3381,936,499,3575,2685,3374,8278,2731,8653,1157,4105}
```

4. Query 1,000,000 records and return them to the client. Example:

```
time psql -h xxx.xxx.xxx.xxx -p 3433 -U digoal postgres -c "select * from t_vec_80" >/dev
/null
```

> **Note** The result is as follows:

```
real    1m1.450s -- The time taken to respond to a single query.
```

5. Test the processing capability of your database for concurrent queries. Example:

```
vi test.sql
select * from t_vec_80;
pgbench -M prepared -n -r -f ./test.sql -c 4 -j 4 -T 600 -h xxx.xxx.xxx.xxx -p 3433 -U di
goal postgres
```

> **Note** The result is as follows:

```
transaction type: ./test.sql
scaling factor: 1
query mode: prepared
number of clients: 4
number of threads: 4
duration: 600 s
number of transactions actually processed: 36
latency average = 72293.794 ms
tps = 0.055330 (including connections establishing)
tps = 0.055330 (excluding connections establishing)
statement latencies in milliseconds:
     72204.857  select * from t_vec_80;
```

Solution 2

- Characteristics
  - ApsaraDB RDS for PostgreSQL stores the values of image feature vectors.
  - The PASE plug-in of ApsaraDB RDS for PostgreSQL creates indexes for image feature vectors.
  - After you specify an image feature vector at the application layer, the application layer retrieves the images similar to the image feature vector from ApsaraDB RDS for PostgreSQL based on the index of the image feature vector. In addition, the application layer returns the distances among vectors and sorts the retrieved images based on these distances.
  - If you specify more than one filter criterion, ApsaraDB RDS for PostgreSQL combines the filter criteria before it filters indexes.
- Merits
  - ApsaraDB RDS for PostgreSQL supports vector indexing, which expedites image searches.
  - ApsaraDB RDS for PostgreSQL filters indexes based on a combination of images and other attribute criteria to converge result sets, increase query speeds, and decrease the volume of data transmitted. A single query can be completed in milliseconds.
  - You can add read-only instances to further increase the overall query throughput.

Procedure

1. Run the following command to create the PASE plug-in:

```
create extension pase;
```

2. Create a test table. Example:

```
create table if not exists t_vec_80(
    id serial PRIMARY KEY,  -- The primary key of the table.
    c1 int,   -- The other attribute fields of the table.
    c2 int,
    c3 text,
    c4 timestamp,
    vec float4[]    -- The data type of the vectors for image feature values.
);
```

3. Create a function to generate random vectors. This function is used to simulate image feature values. You must specify the feature value of the image you want to query. Examples:

```
-- Create a function to generate random vectors.
create or replace function gen_float4_arr1(int,int) returns float4[] as $$
  select array_agg(trunc(random()*$1)::float4) from generate_series(1,$2);
$$ language sql strict volatile;
```

```
-- Create a function based on arrays to generate random neighbor arrays.
create or replace function gen_float4_arr(float4[], int) returns float4[] as $$
  select array_agg( (u + (u*$2/2.0/100) - u*$2/100*random())::float4 ) from unnest($1) u;
$$ language sql strict volatile;
```

4. Write 1,000,000 random vectors. Example:

```
do language plpgsql $$
declare
  v_cent float4[];
begin
  for i in 1..100 loop  -- Specify 100 centroids.
    v_cent := gen_float4_arr1(10000,80);   -- Specify to extract 10,000 values from 80 di
mensions.
    insert into t_vec_80 (vec)
    select
      gen_float4_arr(v_cent, 20)
    from generate_series(1,10000);   -- Specify to extract 10,000 values that center on e
ach centroid, with an increase or decrease of 20% of the total values extracted from each
dimension.
  end loop;
end;
$$;
```

5. Create indexes for image feature vectors by using HNSW. The PASE plug-in supports indexes that are created by using both IVFFlat and HNSW. Example:

> **Note** Before you perform this step, see Use the PASE plug-in for efficient vector search. Make sure that all index parameters, especially the dimensions, are correctly configured.

```
CREATE INDEX idx_t_vec_80_1 ON t_vec_80
USING
  pase_hnsw(vec)
WITH
  (dim = 80, base_nb_num = 16, ef_build = 40, ef_search = 200, base64_encoded = 0);
```

Note The index creation takes 1282997.955 milliseconds, and the indexes created occupy 8,138 MB of storage space. If image feature values are added or updated, you do not need to create indexes, because ApsaraDB RDS for PostgreSQL automatically updates the indexes.

6. Query the top 5 vectors that are similar to a random vector and return them in sequence based on their distances. Example:

```
explain select
  id as v_id,
  vec <? > '7533.44,3740.27,670.119,994.914,3619.27,2018.17,2041.34,5483.19,6370.07,4745.
54,8762.81,1117.59,8254.75,2009.3,6512.47,3876.7,4775.02,384.683,2003.78,7926.78,9101.46,
6801.24,5397.1,7704.49,7546.87,9129.23,9517.36,5723.4,877.649,3117.72,6739.25,8950.36,639
7.09,6687.46,9606.15,557.142,9742.48,1714.25,6682.97,5369.21,6178.99,4983.06,7064.29,5433
.98,7120.7,2980.34,8485.47,1651.98,3656.9,1126.65,10260.1,2139.89,9041.79,4988.89,17.2254
,5482.88,3428.6,10370.7,1749.32,4761.6,2806.65,8040.89,3176.31,9491.93,4355.37,2898.47,28
2.75,3233.86,4248.86,7012.86,9238.95,524.011,2285.75,5363.21,5558.95,10768.8,8689.83,4907
.53,1372.65,1982.05:40:0'::pase as v_dist,
  vec as v_vec
    from t_vec_80
  order by vec <? > '7533.44,3740.27,670.119,994.914,3619.27,2018.17,2041.34,5483.19,6370
.07,4745.54,8762.81,1117.59,8254.75,2009.3,6512.47,3876.7,4775.02,384.683,2003.78,7926.78
,9101.46,6801.24,5397.1,7704.49,7546.87,9129.23,9517.36,5723.4,877.649,3117.72,6739.25,89
50.36,6397.09,6687.46,9606.15,557.142,9742.48,1714.25,6682.97,5369.21,6178.99,4983.06,706
4.29,5433.98,7120.7,2980.34,8485.47,1651.98,3656.9,1126.65,10260.1,2139.89,9041.79,4988.8
9,17.2254,5482.88,3428.6,10370.7,1749.32,4761.6,2806.65,8040.89,3176.31,9491.93,4355.37,2
898.47,282.75,3233.86,4248.86,7012.86,9238.95,524.011,2285.75,5363.21,5558.95,10768.8,868
9.83,4907.53,1372.65,1982.05:40:0'::pase limit 5;
                                                                    QUERY PLAN

------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
----------------
------------------------------------------------------------------------------------------
```

```
------------------------------------------------------------
 Limit  (cost=0.00..7.47 rows=5 width=352)
   ->  Index Scan using idx_t_vec_80_1 on t_vec_80  (cost=0.00..1493015.50 rows=1000000 w
idth=352)
         Order By: (vec <? > '7533.43994140625,3740.27001953125,670.119018554688,994.9140
01464844,3619.27001953125,2018.17004394531,2041.33996582031,5483.18994140625,6370.0698242
1875,4745.5400390
625,8762.8095703125,1117.58996582031,8254.75,2009.30004882812,6512.47021484375,3876.69995
117188,4775.02001953125,384.683013916016,2003.78002929688,7926.77978515625,9101.459960937
5,6801.240234375
,5397.10009765625,7704.490234375,7546.8701171875,9129.23046875,9517.3603515625,5723.39990
234375,877.648986816406,3117.71997070312,6739.25,8950.3603515625,6397.08984375,6687.45996
09375,9606.15039
0625,557.142028808594,9742.48046875,1714.25,6682.97021484375,5369.2099609375,6178.9902343
75,4983.06005859375,7064.2900390625,5433.97998046875,7120.7001953125,2980.34008789062,848
5.4697265625,165
1.97998046875,3656.89990234375,1126.65002441406,10260.099609375,2139.88989257812,9041.790
0390625,4988.89013671875,17.2254009246826,5482.8798828125,3428.60009765625,10370.70019531
25,1749.31994628
906,4761.60009765625,2806.64990234375,8040.89013671875,3176.31005859375,9491.9296875,4355
.3701171875,2898.46997070312,282.75,3233.86010742188,4248.85986328125,7012.85986328125,92
38.9501953125,52
4.010986328125,2285.75,5363.2099609375,5558.9501953125,10768.7998046875,8689.830078125,49
07.52978515625,1372.65002441406,1982.05004882812::'::pase)
(3 rows)
select
  id as v_id,
  vec <? > '7533.44,3740.27,670.119,994.914,3619.27,2018.17,2041.34,5483.19,6370.07,4745.
54,8762.81,1117.59,8254.75,2009.3,6512.47,3876.7,4775.02,384.683,2003.78,7926.78,9101.46,
6801.24,5397.1,7704.49,7546.87,9129.23,9517.36,5723.4,877.649,3117.72,6739.25,8950.36,639
7.09,6687.46,9606.15,557.142,9742.48,1714.25,6682.97,5369.21,6178.99,4983.06,7064.29,5433
.98,7120.7,2980.34,8485.47,1651.98,3656.9,1126.65,10260.1,2139.89,9041.79,4988.89,17.2254
,5482.88,3428.6,10370.7,1749.32,4761.6,2806.65,8040.89,3176.31,9491.93,4355.37,2898.47,28
2.75,3233.86,4248.86,7012.86,9238.95,524.011,2285.75,5363.21,5558.95,10768.8,8689.83,4907
.53,1372.65,1982.05:40:0'::pase as v_dist,
  vec as v_vec
    from t_vec_80
  order by vec <? > '7533.44,3740.27,670.119,994.914,3619.27,2018.17,2041.34,5483.19,6370
.07,4745.54,8762.81,1117.59,8254.75,2009.3,6512.47,3876.7,4775.02,384.683,2003.78,7926.78
,9101.46,6801.24,5397.1,7704.49,7546.87,9129.23,9517.36,5723.4,877.649,3117.72,6739.25,89
50.36,6397.09,6687.46,9606.15,557.142,9742.48,1714.25,6682.97,5369.21,6178.99,4983.06,706
4.29,5433.98,7120.7,2980.34,8485.47,1651.98,3656.9,1126.65,10260.1,2139.89,9041.79,4988.8
9,17.2254,5482.88,3428.6,10370.7,1749.32,4761.6,2806.65,8040.89,3176.31,9491.93,4355.37,2
898.47,282.75,3233.86,4248.86,7012.86,9238.95,524.011,2285.75,5363.21,5558.95,10768.8,868
9.83,4907.53,1372.65,1982.05:40:0'::pase limit 5;
  v_id   |   v_dist    |


v_vec
---------+-------------+-----------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------
-----------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------
----------------
-----------------------------------------------------------------------------------------
```

```
-------------------------------------------------------------------------------------------------
----------------
-------------------------------------------------------------------------------
 1000001 |      613508 | {7508.56,3828.8,670.162,978.82,3654.93,2052.38,2023.41,5518.4,64
78.1,4814.47,8689.2,1130.5,8421.43,2018.39,6534.18,3884.82,4737.2,385.625,2025.83,7917.54
,8892.97,6900.71
,5421.61,7579.82,7649.6,9337.72,9530.01,5818.69,873.353,3105.67,6622.92,9102.99,6360.46,6
737.99,9374.82,545.683,9734.36,1699.74,6753.08,5320.49,6062.47,4870.6,6907.26,5304.41,716
6.67,2997.09,850
8.14,1691.62,3595.89,1113.89,10232.1,2107.41,8846.84,4875.69,17.1081,5574.26,3513.31,1057
6.6,1763.01,4734.1,2780.48,8165.04,3132.32,9586.17,4345.39,2859.25,286.716,3306.16,4260.5
6,7007.33,9126.8
1,528.518,2288.32,5310,5610,10584,8872.31,4843.43,1347.01,1940.52}
 1003628 | 8.55116e+06 | {7532.93,3345.53,694.608,984.268,3507.72,1950.43,2188.66,6043.55
,6832.57,4384.97,8975.91,1290.02,8519.66,2237.75,6985.71,3890.79,4199.22,410.386,2294.93,
7938.11,8989.48,
6374.35,5742.55,7811.5,7492.1,9067.4,9843.13,5885.26,940.365,3435.39,6545.54,8069.38,6126
.34,6906.32,10175.4,505.915,9504.69,1630.76,6832.68,5477.68,6446.75,5109.62,6686.55,5688.
48,6778.92,3100.
2,9182.86,1733.95,3933.06,1116.63,10488.3,2346.63,8257.46,5312.34,16.0066,5078.85,3717.24
,10262.9,1624.57,4406.59,2983.23,7405.85,3159.04,9924.56,4947.86,2573.72,276.545,3673.99,
4487.34,6820.15,
8524.12,486.187,2328.58,4769.64,5541.63,10255.7,8280.42,5141.37,1332.7,1989.67}
 1004945 | 9.21305e+06 | {7348.72,3833.3,706.311,985.864,3632.96,2153.75,2172.06,6427.87,
6502.42,4678.54,8955.37,1207.76,8594.73,1958.02,6839.83,3703.57,4091.18,367.272,1970.81,7
266.62,9198.17,6
869.98,5960.79,7658.46,7180.35,9386.35,10320.3,6593.09,900.23,3330.1,6749.94,9182.85,6839
.25,7254.11,9533.32,580.504,9069.41,1841.88,6840.14,4948.41,6390.41,5102.95,6873.49,5683.
65,7283.23,3124.
15,8727.17,1810.11,3575.12,1111.99,10081.7,2174.01,8797.29,5301.64,17.779,5196.54,3848.29
,9813.85,1514.4,4357.8,2752.47,7138.15,2905.04,10178.2,5025.82,2713.42,267.272,3557.03,43
88.08,6581.85,91
14.22,470.335,2249.53,5274.76,5353.28,10566.6,9153.67,4746.68,1439.05,1996.43}
 1009195 |  9.5047e+06 | {7952.02,3520.88,632.554,1014.25,3682.3,2152.37,2108.55,5609.13,
6663.42,4410.93,7935.51,1272.55,8609.25,2337.6,6845.14,3849.27,3970,422.706,2090.26,8533.
55,9108.23,6752.
42,5636.14,7223.91,7627.38,9467.08,8763.63,6810.94,819.782,3407.48,6512.03,9083.21,6403.4
4,6224.57,9703.19,553.033,9508.63,1823.54,6942.67,5340.35,5954.36,5616.57,6423.06,5320.32
,7837.67,2903.61
,8450.55,1892.85,3821.65,1140.62,10152.7,2306.96,8871.29,5034.8,17.8199,5573.62,3686.87,1
0214.3,1688.62,4667.3,2943.37,7669.45,3079.31,10188.6,4638.13,2907.09,254.251,3438.58,465
7.61,6342.84,948
5.26,465.782,2388.26,5125.77,6048.52,9961.5,8328.46,5174.91,1416.44,1937.93}
 1008523 | 9.65744e+06 | {7255.87,3299.84,671.464,1047.55,3705.29,2031.92,1992.93,5689.99
,6486.58,4153.71,8173.91,1224.91,8320.19,2170.14,6585.28,3911.89,4329.78,401.384,2084.19,
8345.98,9496.74,
7188.78,5137.15,7485.36,6914.55,8471.34,9674.72,6395.1,810.129,3015.94,6551.72,8213.34,65
18.96,6672.72,9064.75,565.507,9560.03,1621.07,7184.9,5224.67,6092.26,4897.21,6021.32,5271
.55,7731.19,3218
.24,8516.33,1660.11,3269.62,1145.53,10584.7,2058.17,7786.21,4795.73,16.5323,5396.69,3830.
83,10393.6,1526.46,4794.47,2644.17,8514.68,3477.77,9360.25,4510.64,2528.64,238.049,3361.8
8,4388.69,7549.8
3,9101.76,545.511,2029.84,5622.08,5770.27,10192.2,8269.93,4979.93,1547.04,2017}
(5 rows)
Time: 2.502 ms
```

7. Test the processing capability of your database for concurrent queries. Example:

```
-- A function is used to simulate a real business scenario, so each image feature value y
ou specify is random.
-- The test method is as follows:
-- Retrieve a random record from the table and increase or decrease the floating-point va
lues extracted from each dimension by 5% to generate a new random vector.
-- Search for the top 5 vectors that are similar to the new random vector and return them
in sequence based on their degrees of similarity.
create or replace function get_vec(
in i_id int,
in i_pect int,
out v_id int,
out v_dist float4,
out v_vec float4[]
) returns setof record as $$
declare
v_vec float4[];
v_pase text;
begin
  select vec into v_vec from t_vec_80 where id=i_id;
  v_vec := gen_float4_arr(v_vec, i_pect);
  v_pase := rtrim(ltrim(v_vec::text, '{'),'}')||':40:0';
  -- raise notice '%', v_pase;
  return query
  select
  id as v_id,
  vec <? > v_pase::pase as v_dist,
  vec as v_vec
    from t_vec_80
  order by vec <? > v_pase::pase limit 5;
end;
$$ language plpgsql strict;
postgres=> select min(id),max(id) from t_vec_80;
   min   |   max
---------+---------
 1000001 | 2000000
(1 row)
vi test.sql
\set id random(1000001,2000000)
select * from get_vec(:id, 5);
pgbench -M prepared -n -r -f ./test.sql -c 12 -j 12 -T 600 -h xxx.xxx.xxx.xxx -p 3433 -U
digoal postgres
```

8. Search for vectors. Example:

```
postgres=> select * from get_vec(1000001,5);
-[ RECORD 1 ]----------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------
```

```
v_id   | 1000001
v_dist | 549580
v_vec  | {7508.56,3828.8,670.162,978.82,3654.93,2052.38,2023.41,5518.4,6478.1,4814.47,868
9.2,1130.5,8421.43,2018.39,6534.18,3884.82,4737.2,385.625,2025.83,7917.54,8892.97,6900.71
,5421.61,7579.82,7649.6,9337.72,9530.01,5818.69,873.353,3105.67,6622.92,9102.99,6360.46,6
737.99,9374.82,545.683,9734.36,1699.74,6753.08,5320.49,6062.47,4870.6,6907.26,5304.41,716
6.67,2997.09,8508.14,1691.62,3595.89,1113.89,10232.1,2107.41,8846.84,4875.69,17.1081,5574
.26,3513.31,10576.6,1763.01,4734.1,2780.48,8165.04,3132.32,9586.17,4345.39,2859.25,286.71
6,3306.16,4260.56,7007.33,9126.81,528.518,2288.32,5310,5610,10584,8872.31,4843.43,1347.01
,1940.52}
-[ RECORD 2 ]----------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------
v_id   | 1004945
v_dist | 8.61114e+06
v_vec  | {7348.72,3833.3,706.311,985.864,3632.96,2153.75,2172.06,6427.87,6502.42,4678.54,
8955.37,1207.76,8594.73,1958.02,6839.83,3703.57,4091.18,367.272,1970.81,7266.62,9198.17,6
869.98,5960.79,7658.46,7180.35,9386.35,10320.3,6593.09,900.23,3330.1,6749.94,9182.85,6839
.25,7254.11,9533.32,580.504,9069.41,1841.88,6840.14,4948.41,6390.41,5102.95,6873.49,5683.
65,7283.23,3124.15,8727.17,1810.11,3575.12,1111.99,10081.7,2174.01,8797.29,5301.64,17.779
,5196.54,3848.29,9813.85,1514.4,4357.8,2752.47,7138.15,2905.04,10178.2,5025.82,2713.42,26
7.272,3557.03,4388.08,6581.85,9114.22,470.335,2249.53,5274.76,5353.28,10566.6,9153.67,474
6.68,1439.05,1996.43}
-[ RECORD 3 ]----------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------
v_id   | 1003628
v_dist | 9.11551e+06
v_vec  | {7532.93,3345.53,694.608,984.268,3507.72,1950.43,2188.66,6043.55,6832.57,4384.97
,8975.91,1290.02,8519.66,2237.75,6985.71,3890.79,4199.22,410.386,2294.93,7938.11,8989.48,
6374.35,5742.55,7811.5,7492.1,9067.4,9843.13,5885.26,940.365,3435.39,6545.54,8069.38,6126
.34,6906.32,10175.4,505.915,9504.69,1630.76,6832.68,5477.68,6446.75,5109.62,6686.55,5688.
48,6778.92,3100.2,9182.86,1733.95,3933.06,1116.63,10488.3,2346.63,8257.46,5312.34,16.0066
,5078.85,3717.24,10262.9,1624.57,4406.59,2983.23,7405.85,3159.04,9924.56,4947.86,2573.72,
276.545,3673.99,4487.34,6820.15,8524.12,486.187,2328.58,4769.64,5541.63,10255.7,8280.42,5
141.37,1332.7,1989.67}
-[ RECORD 4 ]----------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------
v_id   | 1007551
v_dist | 9.44673e+06
```

```
v_vec  | {7781.26,3380.16,599.097,902.545,3547.6,1982.01,2408.72,5823.09,5854.29,4392.29,
9184.52,1268.16,8240.44,2106.41,6257.97,3703.93,4635.53,378.289,1987.59,8185.38,8466.11,7
341.06,5290.8,7422.01,7250.71,8765.47,9341.37,6343.1,865.465,3123.4,5753.41,9331.6,6897.8
,6410.83,8874.91,572.861,9001.73,1567.28,6087.64,5422.22,6226.57,5704.15,6499.31,5340.14,
7157.55,3300.96,8137.33,1648.01,3872.58,1048.15,10322,2171.44,8874.25,4800.68,17.2407,529
7.92,3962.59,10463.2,1482.13,4316.52,2762.2,7293.2,2932.35,10294.3,4539.97,2551.33,266.68
9,3879.39,4287.27,7169.59,8934.47,544.819,2246.28,4860.29,5837.37,10389.2,8959.5,4836.24,
1283.66,2118.71}
-[ RECORD 5 ]----------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------------------------------------------------------------------------
----------------------
v_id   | 1008335
v_dist | 9.48463e+06
v_vec  | {7993.58,3279.23,608.321,947.312,3855.4,2190.95,2013.19,6063.82,6356.44,4670.55,
9118.76,1155.98,8339.1,2082.98,6675.26,3565.42,4172.02,432.199,2115.09,7211.91,8375.44,68
45.13,5692.45,7955.92,7269.1,9351.03,9016.28,5845.67,840.522,2964.57,6185.9,9328.92,6371.
88,6985.29,9314.6,575.449,8884.66,1681.17,6381.56,5767.74,5796.38,4839.26,6309.88,5030.22
,7347.04,3403.45,9072.78,1858.26,3753.29,1008.68,10277.5,2072.03,8010.28,5153.73,17.669,4
755.41,3723.93,10381.5,1512.89,4821.96,3179.53,7987.13,3276.66,8983.62,4408.31,2430.41,28
4.952,3731.14,4382.78,6574.45,9154.04,520.929,2136.69,4835.47,5222.18,10158.4,9192.24,482
0.05,1417.67,2106.94}
```

Note The query result is as follows:

```
transaction type: ./test.sql
scaling factor: 1
query mode: prepared
number of clients: 12
number of threads: 12
duration: 600 s
number of transactions actually processed: 633784
latency average = 11.361 ms
tps = 1056.253960 (including connections establishing)
tps = 1056.298691 (excluding connections establishing)
statement latencies in milliseconds:
         0.001  \set id random(1000001,2000000)
        11.358  select * from get_vec(:id, 5);
```

Summary

ApsaraDB RDS for PostgreSQL uses the PASE plug-in to retrieve high-dimensional vectors based on indexes. This enables ApsaraDB RDS for PostgreSQL to respond to a single query for similar images in milliseconds. PASE can also be used in digital businesses such as searches for profiles and audience members. PASE searches for vectors and filters them based on a combination of criteria simultaneously in your database to expedite queries.

ApsaraDB RDS for PostgreSQL is ideal for businesses with needs for highly concurrent queries. Such businesses include the Internet, new retail, public transportation, smart buildings, education, gaming, medical care, social networking, public security, and airports.

Video tutorials

Multidimensional vector similarity search for image recognition

Promotions

ApsaraDB RDS for PostgreSQL discounts

32.3. Second-level flashback for real-time disaster recovery

This topic describes the flashback technology of Zettabyte File System (ZFS) that is supported by ApsaraDB RDS for PostgreSQL. After you specify your ECS instance as a backup of your RDS instance, ZFS allows your ECS instance to synchronize data from your RDS instance and invoke the cron tool to quickly create snapshots at fixed intervals. If your RDS instance becomes faulty, you can use the snapshots on the ECS instance to restore your RDS instance in seconds.

Context

ZFS is a dynamic file management system that offers features and benefits not found in other file systems available today. ZFS uses the concept of storage pools to manage physical storage devices. This allows ZFS to aggregate physical storage devices into a storage pool. The storage pool describes the characteristics of physical storage devices. These characteristics include device layout and data redundancy. The storage pool also acts as an arbitrary data store from which file systems can be created. File systems are no longer constrained to individual physical devices. In addition, file systems provided with individual physical devices can be shared in the storage pool.

ZFS uses the copy on write technique to manage data. When you write new data, the block that contains the original data is retained. This allows ZFS to store all of the data that is required to create snapshots. You can use the data to create snapshots in seconds. In addition, ZFS allows modifications to a file to be shared among file systems and their snapshots. This allows you to optimize the storage usage of ZFS snapshots.

These features of ZFS allow you to find the latest snapshot. You can use the latest snapshot to restore data in the event of faults or disasters.

Before you begin

- Create a suitable ECS instance. For more information, see ECS instance creation overview. We recommend that your ECS instance run the 64-bit CentOS 7 operating system and have the same specifications, zones, Virtual Private Cloud (VPC), and vSwitch as your RDS instance. This ensures the fastest communication between your ECS and RDS instances over an internal network. If you require cross-region disaster recovery, create a Cloud Enterprise Network instance. For more information, see Tutorial overview.
- Configure a whitelist to grant your ECS instance access to your RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

Create a snapshot

1. Connect to your ECS instance. For more information, see Overview.

2. Install ZFS.

```
wget http://download.zfsonlinux.org/epel/zfs-release.el7_7.noarch.rpm
rpm -ivh zfs-release.el7_7.noarch.rpm
yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
yum install -y "kernel-devel-uname-r == $(uname -r)" zfs
```

3. Modify the rc.local file.

```
vi /etc/rc.local
/sbin/modprobe zfs
chmod +x /etc/rc.local
```

4. Test ZFS. After you check that ZFS is properly running, restart your ECS instance.

```
# modprobe zfs
# zpool list
no pools available
# reboot
```

5. Install PostgreSQL 12 on your ECS instance.

```
yum install https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redha
t-repo-latest.noarch.rpm
yum install postgresql12*
```

6. Configure the following parameters on the disks of your ECS instance:

```
parted -a optimal -s /dev/vdb1 mklabel gpt mkpart primary 1MiB 100%FREE   // Align partit
ions. The /dev/vdb1 path is where a new disk is created.
zpool create zp1 -f -o ashift=13 vdc1    // Configure the ZFS storage pool.
zfs set canmount=off zp1     // Set the canmount parameter.
```

7. Create a directory that is used to store data files.

```
zfs create -o mountpoint=/zpdata01 -o recordsize=8K -o atime=off -o primarycache=metadata
-o logbias=throughput -o secondarycache=none zp1/zpdata01
```

8. Create a directory that is used to store archived write-ahead logging (WAL) files.

```
zfs create -o mountpoint=/zpdata02 -o recordsize=8K -o atime=off -o primarycache=metadata
-o logbias=throughput -o secondarycache=none -o compression=on zp1/zpdata02
zfs set compression=on zp1/zpdata02
# zfs list
NAME           USED  AVAIL     REFER  MOUNTPOINT
zp1           1.29M  1.42T      192K  /zp1
zp1/zpdata01   192K  1.42T      192K  /zpdata01
zp1/zpdata02   192K  1.42T      192K  /zpdata02
```

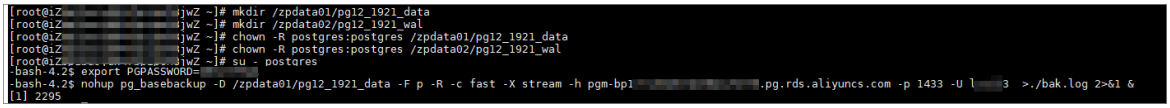
9. Create a directory that is used to store the synchronized data from your RDS instance.

```
mkdir /zpdata01/pg12_1921_data
mkdir /zpdata02/pg12_1921_wal
chown -R postgres:postgres /zpdata01/pg12_1921_data
chown -R postgres:postgres /zpdata02/pg12_1921_wal
```

10. Synchronize data from your RDS instance to your ECS instance.

```
su - postgres
export PGPASSWORD=<The password of the privileged account for your RDS instance>
nohup pg_basebackup -D /zpdata01/pg12_1921_data -F p -R -c fast -X stream -h <The endpoint that is used to connect to your RDS instance> -p <The port that is used to connect to your RDS instance> -U <The username of the privileged account for your RDS instance>  >./bak.log 2>&1 &
```

Note Make sure that you have configured a whitelist to grant your ECS instance access to your RDS instance. For more information, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.



11. Comment out the following parameters in the */zpdata01/pg12_1921_data/postgresql.conf* file:

```
#Fri Mar 13 09:55:03 CST 2020
#ssl_key_file='server.key'
#huge_pages=try
#auto_explain.sample_rate=1
#zhparser.multi_zall=off
#shared_preload_libraries='pg_stat_statements,auth_delay,auto_explain,zhparser,timescaledb,pg_pathman'
#promote_trigger_file='/data/postgresql.trigger'
#ssl=off
#rds_max_log_files=20
#pg_pathman.enable_auto_partition=on
#shared_buffers=32768MB
#zhparser.punctuation_ignore=off
#pg_pathman.override_copy=on
#port=1922
#pg_stat_statements.max=5000
#auth_delay.milliseconds=3s
#auto_explain.log_nested_statements=off
#track_io_timing=on
#zhparser.multi_zmain=off
#auto_explain.log_analyze=off
#archive_mode=on
#ssl_cert_file='server.crt'
#zhparser.multi_short=off
#zhparser.dict_in_memory=off
#auto_explain.log_format=text
#auto_explain.log_min_duration=-1
#rds.rds_max_non_super_conns=12800
#pg_pathman.enable=on
#archive_command='/bin/date'
#auto_explain.log_verbose=off
#log_line_prefix='\1\n\t%p\t%r\t%u\t%d\t%t\t%e\t%T\t%S\t%U\t%E\t\t'
#pg_pathman.enable_runtimemergeappend=on
#zhparser.extra_dicts='dict_extra.xdb'
#auto_explain.log_buffers=off
#pg_stat_statements.track=top
#jit_provider='llvmjit'
#pg_pathman.enable_partitionrouter=off
#pg_stat_statements.track_utility=off
#pg_stat_statements.save=off
#zhparser.dicts_type='EXTRA'
#auto_explain.log_timing=on
#pg_pathman.enable_runtimeappend=on
#zhparser.seg_with_duality=off
#rds.rds_max_super_conns=100
#pg_pathman.enable_partitionfilter=on
#log_destination='stderr,csvlog'
#zhparser.multi_duality=off
#pg_pathman.insert_into_fdw='postgres'
#pg_pathman.enable_bounds_cache=on
#rds.rds_max_non_super_wal_snd=32
#auto_explain.log_triggers=off
#rds_sync_replication_timeout=0
```

12. Modify the postgresql.auto.conf file.

```
vi /zpdata01/pg12_1921_data/postgresql.auto.conf
primary_conninfo = 'user=<The username of the privileged account for your RDS instance> p
assword=''<The password of the privileged account for your RDS instance> '' host=''<The e
ndpoint that is used to connect to your RDS instance>'' port=<The port that is used to co
nnect to your RDS instance> application_name=hello_rds_pg12'
port=1922
shared_buffers=32GB
log_destination='csvlog'
archive_mode=always
archive_command='test ! -f /zpdata02/pg12_1921_wal/%f && cp %p /zpdata02/pg12_1921_wal/%f
'
```

13. Modify the permissions on the created directories.

```
chmod 700 /zpdata02/pg12_1921_wal
chmod 700 /zpdata01/pg12_1921_data
```

14. Start the PostgreSQL 12 database on your ECS instance.

```
su - postgres
/usr/pgsql-12/bin/pg_ctl start -D /zpdata01/pg12_1921_data
```

> **Note** If the following error is reported because the configuration of your ECS instance is low, we recommend that you upgrade your ECS instance:
>
> ```
> HINT:  This error usually means that PostgreSQL's request for a shared memory segment
> exceeded available memory, swap space, or huge pages.
> ```

15. Configure the PostgreSQL 12 database on your ECS instance to automatically start.

```
vi /etc/rc.local
su - postgres -c "/usr/pgsql-12/bin/pg_ctl start -D /zpdata01/pg12_1921_data"
```

16. Configure the PostgreSQL 12 database on your ECS instance to automatically create snapshots of the directory that is used to store data files. You do not need to create snapshots of the directory that is used to store archived WAL files.
    i. Create a script to configure the permission that is used to create snapshots.

    ```
    vi /etc/snap.sh
    STIME=`date +%F%T`
    /usr/sbin/zfs snapshot zp1/zpdata01@$STIME
    chmod 500 /etc/snap.sh
    ```

    ii. Check whether snapshots can be properly created.

    ```
    /etc/snap.sh
    # zfs list -t snapshot
    NAME                              USED  AVAIL     REFER  MOUNTPOINT
    zp1/zpdata01@2020-03-2117:06:47   144K      -      770M  -
    ```

iii. Configure the crond daemon process to automatically start.

```
# systemctl start crond
# systemctl enable crond
# systemctl status crond
● crond.service - Command Scheduler
   Loaded: loaded (/usr/lib/systemd/system/crond.service; enabled; vendor preset: ena
bled)
   Active: active (running) since Sat 2020-03-21 16:16:08 CST; 53min ago
 Main PID: 2526 (crond)
   CGroup: /system.slice/crond.service
           └─2526 /usr/sbin/crond -n
```

iv. Configure the crontab file.

```
# crontab -e
1 1 * * * /etc/snap.sh
# crontab -l -u root
1 1 * * * /etc/snap.sh
```

Note You can configure automatic deletion of snapshots or archives based on your disk usage and business requirements. To manually delete snapshots or archives, follow these steps:

```
Manually delete snapshots:
# zfs list -t snapshot
zNAME                                 USED  AVAIL     REFER  MOUNTPOINT
zp1/zpdata01@2020-03-2117:06:47   144K      -      770M  -
zp1/zpdata01@2020-03-2117:17:01     0B      -      786M  -
# zfs destroy zp1/zpdata01@2020-03-2117:06:47
# zfs list -t snapshot
NAME                                USED  AVAIL     REFER  MOUNTPOINT
zp1/zpdata01@2020-03-2117:17:01     0B      -      786M  -
Manually delete archives:
find /zpdata02/pg12_1921_wal/ -type f -mtime +7 -exec rm -f {} \;
```

17. Check the validity of backup sets. For more information, visit GitHub.
18. Obtain the latency of data synchronization between your RDS and ECS instances.

```
postgres=> select pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_flush_lsn(),sent_lsn)) as
sent_delay,
pg_size_pretty(pg_wal_lsn_diff(pg_current_wal_flush_lsn(),replay_lsn)) as replay_dealy,*

from pg_stat_replication ;
-[ RECORD 1 ]----+------------------------------
sent_delay       | 0 bytes
replay_dealy     | 0 bytes
pid              | 84098
usesysid         | 886185
usename          | rep
application_name | hello_rds_pg12
client_addr      | 192.168.0.173
client_hostname  |
client_port      | 60402
backend_start    | 2020-03-21 16:59:01.890775+08
backend_xmin     |
state            | streaming
sent_lsn         | 11D/97002068
write_lsn        | 11D/97002068
flush_lsn        | 11D/97002068
replay_lsn       | 11D/97002068
write_lag        |
flush_lag        |
replay_lag       |
sync_priority    | 0
sync_state       | async
reply_time       | 2020-03-21 17:01:17.198139+08
```

After you complete the preceding steps, snapshots will be created at fixed intervals on your ECS instance. You can use the created snapshots to restore your RDS instance in seconds for real-time disaster recovery.

Restore your RDS instance in seconds

1. Clone the ZFS file system based on a suitable snapshot.

```
# zfs list -t snapshot
NAME                                USED  AVAIL     REFER  MOUNTPOINT
zp1/zpdata01@2020-03-2117:17:01   312K      -      786M  -
# zfs clone -o mountpoint=/test_recovery zp1/zpdata01@2020-03-2117:17:01 zp1/zpdata_test

[root@iZbp135pwcjjoxqgfpw9k1Z pg12_1921_wal]# cd /test_recovery
[root@iZbp135pwcjjoxqgfpw9k1Z test_recovery]# ll
total 17
drwx------ 20 postgres postgres 35 Mar 21 16:59 pg12_1921_data
```

2. Configure the parameters that are used to restore the data of your RDS instance.

> **Note** If your ECS instance cannot provide a sufficient memory capacity, you can configure a small shared buffer.

```
vi /test_recovery/pg12_1921_data/postgresql.auto.conf
port=1923
shared_buffers=32GB
log_destination='csvlog'
recovery_end_command = 'cp /zpdata02/pg12_1921_wal/%f %p'
recovery_target_time = '2020-03-21 17:28:37.670338+08'
recovery_target_timeline = 'latest'
recovery_target_action = 'pause'
```

3. Delete the socket and pid files from the cloned ZFS file system.

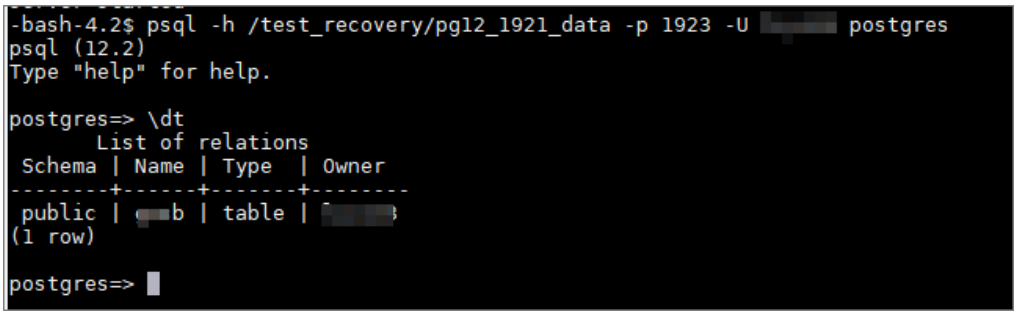
```
rm -f /test_recovery/pg12_1921_data/.s.*
rm /test_recovery/pg12_1921_data/postmaster.pid
```

4. Restore the data from the cloned ZFS file system.

```
su - postgres
/usr/pgsql-12/bin/pg_ctl start -D /test_recovery/pg12_1921_data
```

5. View the restored data.

```
psql -h /test_recovery/pg12_1921_data -p 1923 -U <The username of the privileged account
for your RDS instance> postgres
psql (12.1)
Type "help" for help.
postgres=> \dt
```



> **Note** After you check that the restored data is correct, you can use the pgdump tool to copy the restored data to your RDS instance.

Delete restored data from your ECS instance

After the restoration is complete, run the following commands to delete the restored data from the test_recovery directory on your ECS instance:

```
su - postgres
/usr/pgsql-12/bin/pg_ctl stop -m fast -D /test_recovery/pg12_1921_data
sudo zfs destroy zp1/zpdata_test
```

32.4. Use pgpool for read/write splitting in ApsaraDB RDS for PostgreSQL

This topic describes how to use the pgpool tool of PostgreSQL installed on an ECS instance to implement read/write splitting for your primary and read-only ApsaraDB RDS for PostgreSQL instances.

Context

If you do not use pgpool for high availability, pgpool is stateless. The decrease in performance can be ignored. Additionally, pgpool supports horizontal scaling of your database system. You can use pgpool with the high availability architecture of ApsaraDB RDS for PostgreSQL to implement easy read/write splitting.

Set up a test environment

If you have purchased a primary RDS instance that runs PostgreSQL 10 with local SSDs and have attached read-only instances to the primary instance, you only need to install pgpool. For more information, see Create an RDS PostgreSQL instance and Create an RDS PostgreSQL read-only instance. After you install pgpool, go to Configure pgpool.

> **Note** Read-only instances are under development. These instances apply to primary ApsaraDB RDS for PostgreSQL instances equipped with standard or enhanced SSDs.

The ECS instance you use for testing must meet the following requirements:

- The type of the ECS instance must provide 16-core CPUs, 64 GB of memory, and 1.8 TB of storage capacity that is available on standard SSDs.
- The ECS instance runs the 64-bit CentOS 7.7 operating system.

1. Modify the sysctl.conf file.

```
vi /etc/sysctl.conf
# add by digoal.zhou
fs.aio-max-nr = 1048576
fs.file-max = 76724600
# Optional. Set the kernel.core_pattern parameter to /data01/corefiles/core_%e_%u_%t_%s.%
p.
# The /data01/corefiles directory used to store core dumps is created with the 777 permis
sion before testing. If a symbolic link is used, change the directory to 777.
kernel.sem = 4096 2147483647 2147483646 512000
# Specify the semaphores. You can run the ipcs -l or -u command to obtain the semaphores.
Each group of 16 processes requires 17 semaphores.
kernel.shmall = 107374182
# Specify the total size of shared memory segments. Recommended value: 80% of the memory
capacity. Unit: pages.
kernel.shmmax = 274877906944
# Specify the maximum size of a single shared memory segment. Recommended value: 50% of t
he memory capacity. Unit: bytes. In PostgreSQL versions later than 9.2, the use of shared
memory significantly drops.
kernel.shmmni = 819200
# Specify the total number of shared memory segments that can be generated. At least two
shared memory segments must be generated per PostgreSQL cluster.
```

```
shared memory segments must be generated per PostgreSQL cluster.
net.core.netdev_max_backlog = 10000
net.core.rmem_default = 262144
# The default setting of the socket receive buffer in bytes.
net.core.rmem_max = 4194304
# The maximum receive socket buffer size in bytes
net.core.wmem_default = 262144
# The default setting (in bytes) of the socket send buffer.
net.core.wmem_max = 4194304
# The maximum send socket buffer size in bytes.
net.core.somaxconn = 4096
net.ipv4.tcp_max_syn_backlog = 4096
net.ipv4.tcp_keepalive_intvl = 20
net.ipv4.tcp_keepalive_probes = 3
net.ipv4.tcp_keepalive_time = 60
net.ipv4.tcp_mem = 8388608 12582912 16777216
net.ipv4.tcp_fin_timeout = 5
net.ipv4.tcp_synack_retries = 2
net.ipv4.tcp_syncookies = 1
# Enable SYN cookies. If an SYN waiting queue overflows, you can enable SYN cookies to de
fend against a small number of SYN attacks.
net.ipv4.tcp_timestamps = 1
# Reduce the time after which a network socket enters the TIME-WAIT state.
net.ipv4.tcp_tw_recycle = 0
# If you set this parameter to 1 to enable the recycle function, the network sockets in t
he TIME-WAIT state over TCP connections are recycled. However, if network address transla
tion (NAT) is used, TCP connections may fail. Therefore, we recommend that you set this p
arameter to 0 on the database server.
net.ipv4.tcp_tw_reuse = 1
# Enable the reuse function. This function enables network sockets in the TIME-WAIT state
to be reused over new TCP connections.
net.ipv4.tcp_max_tw_buckets = 262144
net.ipv4.tcp_rmem = 8192 87380 16777216
net.ipv4.tcp_wmem = 8192 65536 16777216
net.nf_conntrack_max = 1200000
net.netfilter.nf_conntrack_max = 1200000
vm.dirty_background_bytes = 409600000
# If the size of dirty pages reaches the specified limit, a background scheduling process
(for example, pdflush) is invoked to flush the dirty pages to disks. These are the pages
that are generated n seconds earlier. The value of n is calculated by using the following
formula: n = The value of the dirty_expire_centisecs parameter/100.
# The default limit is 10% of the memory capacity. If the memory capacity is large, we re
commend that you specify the limit in bytes.
vm.dirty_expire_centisecs = 3000
# Specify the maximum period to retain dirty pages. Dirty pages are flushed to disks afte
r the time period specified by this parameter elapses. The value 3000 indicates 30 second
s.
vm.dirty_ratio = 95
# The processes that users call to write data onto disks must actively flush dirty pages
to disks. This applies when the background scheduling process to flush dirty pages is slo
w and the size of dirty pages exceeds 95% of the memory capacity. These processes include
fsync and fdatasync.
# Set this parameter properly to prevent user-called processes from flushing dirty pages
to disks. This allows you to create multiple RDS instances on a single server and use con
trol groups to limit the input/output operations per second (IOPS) per instance.
```

```
vm.dirty_writeback_centisecs = 100
# Specify the time interval at which the background scheduling process (for example, pdfl
ush) flushes dirty pages to disks. The value 100 indicates 1 second.
vm.swappiness = 0
# Disable the swapping function.
vm.mmap_min_addr = 65536
vm.overcommit_memory = 0
# Specify whether you can allocate more memory space than the physical host has available
. If you set this parameter to 1, the system always considers the available memory space
sufficient. Therefore, if the memory capacity provided in the test environment is low, we
recommend that you set this parameter to 1.
vm.overcommit_ratio = 90
# Specify the memory capacity that can be allocated when the overcommit_memory parameter
is set to 2.
vm.swappiness = 0
# Disable the swapping function.
vm.zone_reclaim_mode = 0
# Disable non-uniform memory access (NUMA). You can also disable NUMA in the vmlinux file
.
net.ipv4.ip_local_port_range = 40000 65535
# Specify the range of TCP or UDP port numbers that are allocated by the physical host.

fs.nr_open=20480000
# Specify the maximum number of file handles that a single process can open.

# Note the following parameters:
#vm.extra_free_kbytes = 4096000   # If the physical host provides a low memory capacity,
do not specify a large value such as 4096000. If you specify a large value, the physical
host may not start.
#vm.min_free_kbytes = 6291456    # We recommend that you increase the value of the vm.min
_free_kbytes parameter by 1 GB for every 32 GB of memory.
# If the physical host provides a low memory capacity, we recommend that you do not confi
gure the preceding two parameters.
# vm.nr_hugepages = 66536
# If the size of the shared buffer exceeds 64 GB, we recommend that you use huge pages. Y
ou can specify the page size by setting the Hugepagesize parameter in the /proc/meminfo f
ile.
#vm.lowmem_reserve_ratio = 1 1 1
# If the memory capacity exceeds 64 GB, we recommend that you set this parameter. Otherwi
se, we recommend that you retain the default value 256 256 32.
```

2. Modify the limits.conf file.

```
vi /etc/security/limits.conf
* soft    nofile  1024000
* hard    nofile  1024000
* soft    nproc   unlimited
* hard    nproc   unlimited
* soft    core    unlimited
* hard    core    unlimited
* soft    memlock unlimited
* hard    memlock unlimited
# Comment out the other parameters in the limits.conf file.
# Comment out the /etc/security/limits.d/20-nproc.conf file.
```

3. Disable transparent huge pages, configure huge pages, and start PostgreSQL.

```
chmod +x /etc/rc.d/rc.local
vi /etc/rc.local
# Disable transparent huge pages.
if test -f /sys/kernel/mm/transparent_hugepage/enabled; then
   echo never > /sys/kernel/mm/transparent_hugepage/enabled
fi
# Configure huge pages for two instances. Each instance has a 16-GB shared buffer.
#sysctl -w vm.nr_hugepages=17000
# Start the two instances.
su - postgres -c "pg_ctl start -D /data01/pg12_3389/pg_root"
su - postgres -c "pg_ctl start -D /data01/pg12_8002/pg_root"
```

4. Create a file system.

> **Warning** If you use a new disk, you must verify that the new disk belongs to the vdb partition instead of the vda partition. If the new disk belongs to the vda partition, data may be deleted from the new disk.

```
parted -a optimal -s /dev/vdb mklabel gpt mkpart primary 1MiB 100%FREE
mkfs.ext4 /dev/vdb1 -m 0 -O extent,uninit_bg -E lazy_itable_init=1 -b 4096 -T largefile -
L vdb1
vi /etc/fstab
LABEL=vdb1 /data01 ext4 defaults,noatime,nodiratime,nodelalloc,barrier=0,data=writeback 0
0
mkdir /data01
mount -a
```

5. Start the irqbalance command line tool.

```
systemctl status irqbalance
systemctl enable irqbalance
systemctl start irqbalance
systemctl status irqbalance
```

6. Install PostgreSQL 10 and pgpool.

```
yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
yum install -y https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

yum install -y https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-re
dhat-repo-latest.noarch.rpm
yum search all postgresql
yum search all pgpool
yum install -y postgresql12*
yum install -y pgpool-II-12-extensions
```

7. Initialize the data directory of your database system.

```
mkdir /data01/pg12_3389
chown postgres:postgres /data01/pg12_3389
```

8. Configure environment variables for the postgres user.

```
su - postgres
vi .bash_profile
# Append the following parameters:
export PS1="$USER@`/bin/hostname -s`-> "
export PGPORT=3389
export PGDATA=/data01/pg12_$PGPORT/pg_root
export LANG=en_US.utf8
export PGHOME=/usr/pgsql-12
export LD_LIBRARY_PATH=$PGHOME/lib:/lib64:/usr/lib64:/usr/local/lib64:/lib:/usr/lib:/usr/
local/lib:$LD_LIBRARY_PATH
export DATE=`date +"%Y%m%d%H%M"`
export PATH=$PGHOME/bin:$PATH:.
export MANPATH=$PGHOME/share/man:$MANPATH
export PGHOST=$PGDATA
export PGUSER=postgres
export PGDATABASE=db1
alias rm='rm -i'
alias ll='ls -lh'
unalias vi
```

9. Initialize your primary RDS instance.

```
initdb -D $PGDATA -U postgres -E UTF8 --lc-collate=C --lc-ctype=en_US.utf8
```

10. Modify the postgresql.conf file.

```
listen_addresses = '0.0.0.0'
port = 3389
max_connections = 1500
superuser_reserved_connections = 13
unix_socket_directories = '., /var/run/postgresql, /tmp'
tcp_keepalives_idle = 60
tcp_keepalives_interval = 10
tcp_keepalives_count = 10
shared_buffers = 16GB
huge_pages = on
work_mem = 8MB
maintenance_work_mem = 1GB
dynamic_shared_memory_type = posix
vacuum_cost_delay = 0
bgwriter_delay = 10ms
bgwriter_lru_maxpages = 1000
bgwriter_lru_multiplier = 10.0
bgwriter_flush_after = 512kB
effective_io_concurrency = 0
max_worker_processes = 128
max_parallel_maintenance_workers = 3
max_parallel_workers_per_gather = 4
parallel_leader_participation = off
max_parallel_workers = 8
backend_flush_after = 256
wal_level = replica
synchronous_commit = off
full_page_writes = on
wal_compression = on
wal_buffers = 16MB
```

```
wal_writer_delay = 10ms
wal_writer_flush_after = 1MB
checkpoint_timeout = 15min
max_wal_size = 64GB
min_wal_size = 8GB
checkpoint_completion_target = 0.2
checkpoint_flush_after = 256kB
random_page_cost = 1.1
effective_cache_size = 48GB
log_destination = 'csvlog'
logging_collector = on
log_directory = 'log'
log_filename = 'postgresql-%a.log'
log_truncate_on_rotation = on
log_rotation_age = 1d
log_rotation_size = 0
log_min_duration_statement = 1s
log_checkpoints = on
log_connections = on
log_disconnections = on
log_line_prefix = '%m [%p] '
log_statement = 'ddl'
log_timezone = 'Asia/Shanghai'
autovacuum = on
log_autovacuum_min_duration = 0
autovacuum_vacuum_scale_factor = 0.1
autovacuum_analyze_scale_factor = 0.05
autovacuum_freeze_max_age = 800000000
autovacuum_multixact_freeze_max_age = 900000000
autovacuum_vacuum_cost_delay = 0
vacuum_freeze_table_age = 750000000
vacuum_multixact_freeze_table_age = 750000000
datestyle = 'iso, mdy'
timezone = 'Asia/Shanghai'
lc_messages = 'en_US.utf8'
lc_monetary = 'en_US.utf8'
lc_numeric = 'en_US.utf8'
lc_time = 'en_US.utf8'
default_text_search_config = 'pg_catalog.english'
```

11. Modify the pg_hba.conf file.

> **Note** Pgpool is installed on the same ECS instance as the database server where PostgreSQL resides. If you specify the 127.0.0.1 IP address in the pg_hba.conf file, you must enter the correct password to ensure a successful logon.

```
# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 trust
# Allow replication connections from localhost, by a user with the
# replication privilege.
local   replication     all                                     trust
host    replication     all             127.0.0.1/32            trust
host    replication     all             ::1/128                 trust
host db123 digoal 0.0.0.0/0 md5
```

12. Create a user authorized with streaming replication permissions. Example:

```
db1=# create role rep123 login replication encrypted password 'xxxxxxx';
CREATE ROLE
```

13. Create a user who is authorized to manage your RDS instances. Example:

```
db1=# create role digoal login encrypted password 'xxxxxxx';
CREATE ROLE
db1=# create database db123 owner digoal;
CREATE DATABASE
```

14. Create a user who is authorized to check the health heartbeats between pgpool and your read-only RDS instances. With the parameters of pgpool properly configured, this user can check the write-ahead logging (WAL) replay latency on each read-only RDS instance. Example:

```
create role nobody login encrypted password 'xxxxxxx';
```

Create a secondary RDS instance

To simplify the test procedure, follow these steps to create a secondary RDS instance on the same ECS instance as your primary RDS instance.

1. Use the pg_basebackup tool to create an online secondary RDS instance. Example:

```
pg_basebackup -D /data01/pg12_8002/pg_root -F p --checkpoint=fast -P -h 127.0.0.1 -p 3389
-U rep123
```

2. Modify the postgresql.conf file of the secondary RDS instance.

```
cd /data01/pg12_8002/pg_root
vi postgresql.conf
# The secondary RDS instance has the following configuration different from the primary R
DS instance:
port = 8002
primary_conninfo = 'hostaddr=127.0.0.1 port=3389 user=rep123' # You do not need to set th
e password. This is because trust relationships are configured on the primary RDS instanc
e.
hot_standby = on
wal_receiver_status_interval = 1s
wal_receiver_timeout = 10s
recovery_target_timeline = 'latest'
```

3. Configure the standby.signal file of the secondary RDS instance.

```
cd /data01/pg12_8002/pg_root
touch standby.signal
```

4. Check whether the data synchronization between the primary and secondary RDS instances is normal.

```
db1=# select * from pg_stat_replication ;
-[ RECORD 1 ]----+------------------------------
pid              | 21065
usesysid         | 10
usename          | postgres
application_name | walreceiver
client_addr      | 127.0.0.1
client_hostname  |
client_port      | 47064
backend_start    | 2020-02-29 00:26:28.485427+08
backend_xmin     |
state            | streaming
sent_lsn         | 0/52000060
write_lsn        | 0/52000060
flush_lsn        | 0/52000060
replay_lsn       | 0/52000060
write_lag        |
flush_lag        |
replay_lag       |
sync_priority    | 0
sync_state       | async
reply_time       | 2020-02-29 01:32:40.635183+08
```

Configure pgpool

1. Query the location where pgpool is installed.

```
# rpm -qa|grep pgpool
pgpool-II-12-extensions-4.1.1-1.rhel7.x86_64
pgpool-II-12-4.1.1-1.rhel7.x86_64
# rpm -ql pgpool-II-12-4.1.1
```

2. Modify the pgpool.conf file.

```
# cd /etc/pgpool-II-12/
cp pgpool.conf.sample-stream pgpool.conf
vi pgpool.conf
# ------------------------------
# pgPool-II configuration file
# ------------------------------
#
# This file consists of lines of the form:
#
#   name = value
#
# Whitespace may be used.  Comments are introduced with "#" anywhere on a line.
# The complete list of parameter names and allowed values can be found in the
# pgPool-II documentation.
#
# This file is read on server startup and when the server receives a SIGHUP
# signal.  If you edit the file on a running system, you have to SIGHUP the
```

```
# server for the changes to take effect, or use "pgpool reload".  Some
# parameters, which are marked below, require a server shutdown and restart to
# take effect.
#
#------------------------------------------------------------------------------
# CONNECTIONS
#------------------------------------------------------------------------------
# - pgpool Connection Settings -
listen_addresses = '0.0.0.0'
                                    # Host name or IP address to listen on:
                                    # '*' for all, '' for no TCP/IP connections
                                    # (change requires restart)
port = 8001
                                    # Port number
                                    # (change requires restart)
socket_dir = '/tmp'
                                    # Unix domain socket path
                                    # The Debian package defaults to
                                    # /var/run/postgresql
                                    # (change requires restart)
reserved_connections = 0
                                    # Number of reserved connections.
                                    # Pgpool-II does not accept connections if over
                                    # num_init_chidlren - reserved_connections.
# - pgpool Communication Manager Connection Settings -
pcp_listen_addresses = ''
                                    # Host name or IP address for pcp process to listen on
:
                                    # '*' for all, '' for no TCP/IP connections
                                    # (change requires restart)
pcp_port = 9898
                                    # Port number for pcp
                                    # (change requires restart)
pcp_socket_dir = '/tmp'
                                    # Unix domain socket path for pcp
                                    # The Debian package defaults to
                                    # /var/run/postgresql
                                    # (change requires restart)
listen_backlog_multiplier = 2
                                    # Set the backlog parameter of listen(2) to
                                    # num_init_children * listen_backlog_multiplier.
                                    # (change requires restart)
serialize_accept = off
                                    # whether to serialize accept() call to avoid thunderi
ng herd problem
                                    # (change requires restart)
# - Backend Connection Settings -
backend_hostname0 = '127.0.0.1'
                                    # Host name or IP address to connect to for backend 0

backend_port0 = 3389
                                    # Port number for backend 0
backend_weight0 = 1
                                    # Weight for backend 0 (only in load balancing mode)
backend_data_directory0 = '/data01/pg12_3389/pg_root'
                                    # Data directory for backend 0
```

```
                                    # Data directory for backend 0
backend_flag0 = 'ALWAYS_MASTER'
                                    # Controls various backend behavior
                                    # ALLOW_TO_FAILOVER, DISALLOW_TO_FAILOVER
                                    # or ALWAYS_MASTER
backend_application_name0 = 'server0'
                                    # walsender's application_name, used for "show pool_no
des" command
backend_hostname1 = '127.0.0.1'
backend_port1 = 8002
backend_weight1 = 1
backend_data_directory1 = '/data01/pg12_8002/pg_root'
backend_flag1 = 'DISALLOW_TO_FAILOVER'
backend_application_name1 = 'server1'
# - Authentication -
enable_pool_hba = on
                                    # Use pool_hba.conf for client authentication
pool_passwd = 'pool_passwd'
                                    # File name of pool_passwd for md5 authentication.
                                    # "" disables pool_passwd.
                                    # (change requires restart)
authentication_timeout = 60
                                    # Delay in seconds to complete client authentication
                                    # 0 means no timeout.
allow_clear_text_frontend_auth = off
                                    # Allow Pgpool-II to use clear text password authentic
ation
                                    # with clients, when pool_passwd does not
                                    # contain the user password
# - SSL Connections -
ssl = off
                                    # Enable SSL support
                                    # (change requires restart)
#ssl_key = './server.key'
                                    # Path to the SSL private key file
                                    # (change requires restart)
#ssl_cert = './server.cert'
                                    # Path to the SSL public certificate file
                                    # (change requires restart)
#ssl_ca_cert = ''
                                    # Path to a single PEM format file
                                    # containing CA root certificate(s)
                                    # (change requires restart)
#ssl_ca_cert_dir = ''
                                    # Directory containing CA root certificate(s)
                                    # (change requires restart)
ssl_ciphers = 'HIGH:MEDIUM:+3DES:! aNULL'
                                    # Allowed SSL ciphers
                                    # (change requires restart)
ssl_prefer_server_ciphers = off
                                    # Use server's SSL cipher preferences,
                                    # rather than the client's
                                    # (change requires restart)
ssl_ecdh_curve = 'prime256v1'
                                    # Name of the curve to use in ECDH key exchange
ssl_dh_params_file = ''
```

```
                                    # Name of the file containing Diffie-Hellman parameter
s used
                                    # for so-called ephemeral DH family of SSL cipher.
#---------------------------------------------------------------------------
# POOLS
#---------------------------------------------------------------------------
# - Concurrent session and pool size -
num_init_children = 128
                                    # Number of concurrent sessions allowed
                                    # (change requires restart)
max_pool = 4
                                    # Number of connection pool caches per connection
                                    # (change requires restart)
# - Life time -
child_life_time = 300
                                    # Pool exits after being idle for this many seconds
child_max_connections = 0
                                    # Pool exits after receiving that many connections
                                    # 0 means no exit
connection_life_time = 0
                                    # Connection to backend closes after being idle for th
is many seconds
                                    # 0 means no close
client_idle_limit = 0
                                    # Client is disconnected after being idle for that man
y seconds
                                    # (even inside an explicit transactions!)
                                    # 0 means no disconnection
#---------------------------------------------------------------------------
# LOGS
#---------------------------------------------------------------------------
# - Where to log -
log_destination = 'syslog'
                                    # Where to log
                                    # Valid values are combinations of stderr,
                                    # and syslog. Default to stderr.
# - What to log -
log_line_prefix = '%t: pid %p: '    # printf-style string to output at beginning of each l
og line.
log_connections = on
                                    # Log connections
log_hostname = off
                                    # Hostname will be shown in ps status
                                    # and in logs if connections are logged
log_statement = off
                                    # Log all statements
log_per_node_statement = off
                                    # Log all statements
                                    # with node and backend informations
log_client_messages = off
                                    # Log any client messages
log_standby_delay = 'if_over_threshold'
                                    # Log standby delay
                                    # Valid values are combinations of always,
                                    # if_over_threshold, none
# - Syslog specific
```

```
# - Syslog specific -
syslog_facility = 'LOCAL0'
                                    # Syslog local facility. Default to LOCAL0
syslog_ident = 'pgpool'
                                    # Syslog program identification string
                                    # Default to 'pgpool'
# - Debug -
#log_error_verbosity = default          # terse, default, or verbose messages
#client_min_messages = notice           # values in order of decreasing detail:
                                        #   debug5
                                        #   debug4
                                        #   debug3
                                        #   debug2
                                        #   debug1
                                        #   log
                                        #   notice
                                        #   warning
                                        #   error
#log_min_messages = warning             # values in order of decreasing detail:
                                        #   debug5
                                        #   debug4
                                        #   debug3
                                        #   debug2
                                        #   debug1
                                        #   info
                                        #   notice
                                        #   warning
                                        #   error
                                        #   log
                                        #   fatal
                                        #   panic
#------------------------------------------------------------------------------
# FILE LOCATIONS
#------------------------------------------------------------------------------
pid_file_name = '/var/run/pgpool-II-12/pgpool.pid'
                                    # PID file name
                                    # Can be specified as relative to the"
                                    # location of pgpool.conf file or
                                    # as an absolute path
                                    # (change requires restart)
logdir = '/tmp'
                                    # Directory of pgPool status file
                                    # (change requires restart)
#------------------------------------------------------------------------------
# CONNECTION POOLING
#------------------------------------------------------------------------------
connection_cache = on
                                    # Activate connection pools
                                    # (change requires restart)
                                    # Semicolon separated list of queries
                                    # to be issued at the end of a session
                                    # The default is for 8.3 and later
reset_query_list = 'ABORT; DISCARD ALL'
                                    # The following one is for 8.2 and before
#reset_query_list = 'ABORT; RESET ALL; SET SESSION AUTHORIZATION DEFAULT'
#------------------------------------------------------------------------------
```

```
# REPLICATION MODE
#------------------------------------------------------------------------------
replication_mode = off
                                    # Activate replication mode
                                    # (change requires restart)
replicate_select = off
                                    # Replicate SELECT statements
                                    # when in replication mode
                                    # replicate_select is higher priority than
                                    # load_balance_mode.
insert_lock = off
                                    # Automatically locks a dummy row or a table
                                    # with INSERT statements to keep SERIAL data
                                    # consistency
                                    # Without SERIAL, no lock will be issued
lobj_lock_table = ''
                                    # When rewriting lo_create command in
                                    # replication mode, specify table name to
                                    # lock
# - Degenerate handling -
replication_stop_on_mismatch = off
                                    # On disagreement with the packet kind
                                    # sent from backend, degenerate the node
                                    # which is most likely "minority"
                                    # If off, just force to exit this session
failover_if_affected_tuples_mismatch = off
                                    # On disagreement with the number of affected
                                    # tuples in UPDATE/DELETE queries, then
                                    # degenerate the node which is most likely
                                    # "minority".
                                    # If off, just abort the transaction to
                                    # keep the consistency
#------------------------------------------------------------------------------
# LOAD BALANCING MODE
#------------------------------------------------------------------------------
load_balance_mode = on
                                    # Activate load balancing mode
                                    # (change requires restart)
ignore_leading_white_space = on
                                    # Ignore leading white spaces of each query
white_function_list = ''
                                    # Comma separated list of function names
                                    # that don't write to database
                                    # Regexp are accepted
black_function_list = 'currval,lastval,nextval,setval'
                                    # Comma separated list of function names
                                    # that write to database
                                    # Regexp are accepted
black_query_pattern_list = ''
                                    # Semicolon separated list of query patterns
                                    # that should be sent to primary node
                                    # Regexp are accepted
                                    # valid for streaming replication mode only.
database_redirect_preference_list = ''
                                    # comma separated list of pairs of database and node i
```

```
d.
                                    # example: postgres:primary,mydb[0-4]:1,mydb[5-9]:2'
                                    # valid for streaming replication mode only.
app_name_redirect_preference_list = ''
                                    # comma separated list of pairs of app name and node i
d.
                                    # example: 'psql:primary,myapp[0-4]:1,myapp[5-9]:stand
by'
                                    # valid for streaming replication mode only.
allow_sql_comments = off
                                    # if on, ignore SQL comments when judging if load bala
nce or
                                    # query cache is possible.
                                    # If off, SQL comments effectively prevent the judgmen
t
                                    # (pre 3.4 behavior).
disable_load_balance_on_write = 'transaction'
                                    # Load balance behavior when write query is issued
                                    # in an explicit transaction.
                                    # Note that any query not in an explicit transaction
                                    # is not affected by the parameter.
                                    # 'transaction' (the default): if a write query is iss
ued,
                                    # subsequent read queries will not be load balanced
                                    # until the transaction ends.
                                    # 'trans_transaction': if a write query is issued,
                                    # subsequent read queries in an explicit transaction
                                    # will not be load balanced until the session ends.
                                    # 'always': if a write query is issued, read queries w
ill
                                    # not be load balanced until the session ends.
statement_level_load_balance = off
                                    # Enables statement level load balancing
#------------------------------------------------------------------------------
# MASTER/SLAVE MODE
#------------------------------------------------------------------------------
master_slave_mode = on
                                    # Activate master/slave mode
                                    # (change requires restart)
master_slave_sub_mode = 'stream'
                                    # Master/slave sub mode
                                    # Valid values are combinations stream, slony
                                    # or logical. Default is stream.
                                    # (change requires restart)
# - Streaming -
sr_check_period = 3
                                    # Streaming replication check period
                                    # Disabled (0) by default
sr_check_user = 'nobody'
                                    # Streaming replication check user
                                    # This is necessary even if you disable streaming
                                    # replication delay check by sr_check_period = 0
sr_check_password = ''
                                    # Password for streaming replication check user
                                    # Leaving it empty will make Pgpool-II to first look f
or the
```

```
                                    # Password in pool_passwd file before using the empty
password
sr_check_database = 'postgres'
                                    # Database name for streaming replication check
delay_threshold = 512000
                                    # Threshold before not dispatching query to standby no
de
                                    # Unit is in bytes
                                    # Disabled (0) by default
# - Special commands -
follow_master_command = ''
                                    # Executes this command after master failover
                                    # Special values:
                                    #   %d = failed node id
                                    #   %h = failed node host name
                                    #   %p = failed node port number
                                    #   %D = failed node database cluster path
                                    #   %m = new master node id
                                    #   %H = new master node hostname
                                    #   %M = old master node id
                                    #   %P = old primary node id
                                    #   %r = new master port number
                                    #   %R = new master database cluster path
                                    #   %N = old primary node hostname
                                    #   %S = old primary node port number
                                    #   %% = '%' character
#------------------------------------------------------------------------------
# HEALTH CHECK GLOBAL PARAMETERS
#------------------------------------------------------------------------------
health_check_period = 5
                                    # Health check period
                                    # Disabled (0) by default
health_check_timeout = 10
                                    # Health check timeout
                                    # 0 means no timeout
health_check_user = 'nobody'
                                    # Health check user
health_check_password = ''
                                    # Password for health check user
                                    # Leaving it empty will make Pgpool-II to first look f
or the
                                    # Password in pool_passwd file before using the empty
password
health_check_database = ''
                                    # Database name for health check. If '', tries 'postgr
es' first,
health_check_max_retries = 60
                                    # Maximum number of times to retry a failed health che
ck before giving up.
health_check_retry_delay = 1
                                    # Amount of time to wait (in seconds) between retries.

connect_timeout = 10000
                                    # Timeout value in milliseconds before giving up to co
nnect to backend.
```

```
                                    # Default is 10000 ms (10 second). Flaky network user
may want to increase
                                    # the value. 0 means no timeout.
                                    # Note that this value is not only used for health che
ck,
                                    # but also for ordinary connection to backend.
#------------------------------------------------------------------------------
# HEALTH CHECK PER NODE PARAMETERS (OPTIONAL)
#------------------------------------------------------------------------------
#health_check_period0 = 0
#health_check_timeout0 = 20
#health_check_user0 = 'nobody'
#health_check_password0 = ''
#health_check_database0 = ''
#health_check_max_retries0 = 0
#health_check_retry_delay0 = 1
#connect_timeout0 = 10000
#------------------------------------------------------------------------------
# FAILOVER AND FAILBACK
#------------------------------------------------------------------------------
failover_command = ''
                                    # Executes this command at failover
                                    # Special values:
                                    #   %d = failed node id
                                    #   %h = failed node host name
                                    #   %p = failed node port number
                                    #   %D = failed node database cluster path
                                    #   %m = new master node id
                                    #   %H = new master node hostname
                                    #   %M = old master node id
                                    #   %P = old primary node id
                                    #   %r = new master port number
                                    #   %R = new master database cluster path
                                    #   %N = old primary node hostname
                                    #   %S = old primary node port number
                                    #   %% = '%' character
failback_command = ''
                                    # Executes this command at failback.
                                    # Special values:
                                    #   %d = failed node id
                                    #   %h = failed node host name
                                    #   %p = failed node port number
                                    #   %D = failed node database cluster path
                                    #   %m = new master node id
                                    #   %H = new master node hostname
                                    #   %M = old master node id
                                    #   %P = old primary node id
                                    #   %r = new master port number
                                    #   %R = new master database cluster path
                                    #   %N = old primary node hostname
                                    #   %S = old primary node port number
                                    #   %% = '%' character
failover_on_backend_error = off
                                    # Initiates failover when reading/writing to the
                                    # backend communication socket fails
                                    # If set to off, pgpool will report an
```

```
                                    # error and disconnect the session.
detach_false_primary = off
                                    # Detach false primary if on. Only
                                    # valid in streaming replication
                                    # mode and with PostgreSQL 9.6 or
                                    # after.
search_primary_node_timeout = 300
                                    # Timeout in seconds to search for the
                                    # primary node when a failover occurs.
                                    # 0 means no timeout, keep searching
                                    # for a primary node forever.
#------------------------------------------------------------------------------
# ONLINE RECOVERY
#------------------------------------------------------------------------------
recovery_user = 'nobody'
                                    # Online recovery user
recovery_password = ''
                                    # Online recovery password
                                    # Leaving it empty will make Pgpool-II to first look f
or the
                                    # Password in pool_passwd file before using the empty
password
recovery_1st_stage_command = ''
                                    # Executes a command in first stage
recovery_2nd_stage_command = ''
                                    # Executes a command in second stage
recovery_timeout = 90
                                    # Timeout in seconds to wait for the
                                    # recovering node's postmaster to start up
                                    # 0 means no wait
client_idle_limit_in_recovery = 0
                                    # Client is disconnected after being idle
                                    # for that many seconds in the second stage
                                    # of online recovery
                                    # 0 means no disconnection
                                    # -1 means immediate disconnection
auto_failback = off
                                    # Dettached backend node reattach automatically
                                    # if replication_state is 'streaming'.
auto_failback_interval = 60
                                    # Min interval of executing auto_failback in
                                    # seconds.
#------------------------------------------------------------------------------
# WATCHDOG
#------------------------------------------------------------------------------
# - Enabling -
use_watchdog = off
                                     # Activates watchdog
                                     # (change requires restart)
# -Connection to up stream servers -
trusted_servers = ''
                                     # trusted server list which are used
                                     # to confirm network connection
                                     # (hostA,hostB,hostC,...)
                                     # (change requires restart)
```

```
ping_path = '/bin'
                                    # ping command path
                                    # (change requires restart)
# - Watchdog communication Settings -
wd_hostname = ''
                                    # Host name or IP address of this watchdog
                                    # (change requires restart)
wd_port = 9000
                                    # port number for watchdog service
                                    # (change requires restart)
wd_priority = 1
                                    # priority of this watchdog in leader election
                                    # (change requires restart)
wd_authkey = ''
                                    # Authentication key for watchdog communication
                                    # (change requires restart)
wd_ipc_socket_dir = '/tmp'
                                    # Unix domain socket path for watchdog IPC socket
                                    # The Debian package defaults to
                                    # /var/run/postgresql
                                    # (change requires restart)
# - Virtual IP control Setting -
delegate_IP = ''
                                    # delegate IP address
                                    # If this is empty, virtual IP never bring up.
                                    # (change requires restart)
if_cmd_path = '/sbin'
                                    # path to the directory where if_up/down_cmd exists
                                    # If if_up/down_cmd starts with "/", if_cmd_path will
be ignored.
                                    # (change requires restart)
if_up_cmd = '/usr/bin/sudo /sbin/ip addr add $_IP_$/24 dev eth0 label eth0:0'
                                    # startup delegate IP command
                                    # (change requires restart)
if_down_cmd = '/usr/bin/sudo /sbin/ip addr del $_IP_$/24 dev eth0'
                                    # shutdown delegate IP command
                                    # (change requires restart)
arping_path = '/usr/sbin'
                                    # arping command path
                                    # If arping_cmd starts with "/", if_cmd_path will be
ignored.
                                    # (change requires restart)
arping_cmd = '/usr/bin/sudo /usr/sbin/arping -U $_IP_$ -w 1 -I eth0'
                                    # arping command
                                    # (change requires restart)
# - Behaivour on escalation Setting -
clear_memqcache_on_escalation = on
                                    # Clear all the query cache on shared memory
                                    # when standby pgpool escalate to active pgpool
                                    # (= virtual IP holder).
                                    # This should be off if client connects to pgpool
                                    # not using virtual IP.
                                    # (change requires restart)
wd_escalation_command = ''
                                    # Executes this command at escalation on new active p
gpool
```

```
gpool.
                                    # (change requires restart)
wd_de_escalation_command = ''
                                    # Executes this command when master pgpool resigns fr
om being master.
                                    # (change requires restart)
# - Watchdog consensus settings for failover -
failover_when_quorum_exists = on
                                    # Only perform backend node failover
                                    # when the watchdog cluster holds the quorum
                                    # (change requires restart)
failover_require_consensus = on
                                    # Perform failover when majority of Pgpool-II nodes
                                    # aggrees on the backend node status change
                                    # (change requires restart)
allow_multiple_failover_requests_from_node = off
                                    # A Pgpool-II node can cast multiple votes
                                    # for building the consensus on failover
                                    # (change requires restart)
enable_consensus_with_half_votes = off
                                    # apply majority rule for consensus and quorum comput
ation
                                    # at 50% of votes in a cluster with even number of no
des.
                                    # when enabled the existence of quorum and consensus

                                    # on failover is resolved after receiving half of the

                                    # total votes in the cluster, otherwise both these
                                    # decisions require at least one more vote than
                                    # half of the total votes.
                                    # (change requires restart)
# - Lifecheck Setting -
# -- common --
wd_monitoring_interfaces_list = ''  # Comma separated list of interfaces names to monitor
.
                                    # if any interface from the list is active the watchd
og will
                                    # consider the network is fine
                                    # 'any' to enable monitoring on all interfaces except
loopback
                                    # '' to disable monitoring
                                    # (change requires restart)
wd_lifecheck_method = 'heartbeat'
                                    # Method of watchdog lifecheck ('heartbeat' or 'query
' or 'external')
                                    # (change requires restart)
wd_interval = 10
                                    # lifecheck interval (sec) > 0
                                    # (change requires restart)
# -- heartbeat mode --
wd_heartbeat_port = 9694
                                    # Port number for receiving heartbeat signal
                                    # (change requires restart)
wd_heartbeat_keepalive = 2
                                    # Interval time of sending heartbeat signal (sec)
```

```
                                    # (change requires restart)
wd_heartbeat_deadtime = 30
                                    # Deadtime interval for heartbeat signal (sec)
                                    # (change requires restart)
heartbeat_destination0 = 'host0_ip1'
                                    # Host name or IP address of destination 0
                                    # for sending heartbeat signal.
                                    # (change requires restart)
heartbeat_destination_port0 = 9694
                                    # Port number of destination 0 for sending
                                    # heartbeat signal. Usually this is the
                                    # same as wd_heartbeat_port.
                                    # (change requires restart)
heartbeat_device0 = ''
                                    # Name of NIC device (such like 'eth0')
                                    # used for sending/receiving heartbeat
                                    # signal to/from destination 0.
                                    # This works only when this is not empty
                                    # and pgpool has root privilege.
                                    # (change requires restart)
#heartbeat_destination1 = 'host0_ip2'
#heartbeat_destination_port1 = 9694
#heartbeat_device1 = ''
# -- query mode --
wd_life_point = 3
                                    # lifecheck retry times
                                    # (change requires restart)
wd_lifecheck_query = 'SELECT 1'
                                    # lifecheck query to pgpool from watchdog
                                    # (change requires restart)
wd_lifecheck_dbname = 'template1'
                                    # Database name connected for lifecheck
                                    # (change requires restart)
wd_lifecheck_user = 'nobody'
                                    # watchdog user monitoring pgpools in lifecheck
                                    # (change requires restart)
wd_lifecheck_password = ''
                                    # Password for watchdog user in lifecheck
                                    # Leaving it empty will make Pgpool-II to first look
for the
                                    # Password in pool_passwd file before using the empty
password
                                    # (change requires restart)
# - Other pgpool Connection Settings -
#other_pgpool_hostname0 = 'host0'
                                    # Host name or IP address to connect to for other pgp
ool 0
                                    # (change requires restart)
#other_pgpool_port0 = 5432
                                    # Port number for other pgpool 0
                                    # (change requires restart)
#other_wd_port0 = 9000
                                    # Port number for other watchdog 0
                                    # (change requires restart)
#other_pgpool_hostname1 = 'host1'
```

```
#other_pgpool_port1 = 5432
#other_wd_port1 = 9000
#------------------------------------------------------------------------------
# OTHERS
#------------------------------------------------------------------------------
relcache_expire = 0
                                # Life time of relation cache in seconds.
                                # 0 means no cache expiration(the default).
                                # The relation cache is used for cache the
                                # query result against PostgreSQL system
                                # catalog to obtain various information
                                # including table structures or if it's a
                                # temporary table or not. The cache is
                                # maintained in a pgpool child local memory
                                # and being kept as long as it survives.
                                # If someone modify the table by using
                                # ALTER TABLE or some such, the relcache is
                                # not consistent anymore.
                                # For this purpose, cache_expiration
                                # controls the life time of the cache.
relcache_size = 8192
                                # Number of relation cache
                                # entry. If you see frequently:
                                # "pool_search_relcache: cache replacement happend"
                                # in the pgpool log, you might want to increate this n
umber.
check_temp_table = catalog
                                # Temporary table check method. catalog, trace or none
.
                                # Default is catalog.
check_unlogged_table = on
                                # If on, enable unlogged table check in SELECT stateme
nts.
                                # This initiates queries against system catalog of pri
mary/master
                                # thus increases load of master.
                                # If you are absolutely sure that your system never us
es unlogged tables
                                # and you want to save access to primary/master, you c
ould turn this off.
                                # Default is on.
enable_shared_relcache = on
                                # If on, relation cache stored in memory cache,
                                # the cache is shared among child process.
                                # Default is on.
                                # (change requires restart)
relcache_query_target = master      # Target node to send relcache queries. Default is mas
ter (primary) node.
                                # If load_balance_node is specified, queries will be s
ent to load balance node.
#------------------------------------------------------------------------------
# IN MEMORY QUERY MEMORY CACHE
#------------------------------------------------------------------------------
memory_cache_enabled = off
                                # If on, use the memory cache functionality, off by de
fault
```

```
                                    # (change requires restart)
memqcache_method = 'shmem'
                                    # Cache storage method. either 'shmem'(shared memory)
or
                                    # 'memcached'. 'shmem' by default
                                    # (change requires restart)
memqcache_memcached_host = 'localhost'
                                    # Memcached host name or IP address. Mandatory if
                                    # memqcache_method = 'memcached'.
                                    # Defaults to localhost.
                                    # (change requires restart)
memqcache_memcached_port = 11211
                                    # Memcached port number. Mondatory if memqcache_method
= 'memcached'.
                                    # Defaults to 11211.
                                    # (change requires restart)
memqcache_total_size = 67108864
                                    # Total memory size in bytes for storing memory cache.

                                    # Mandatory if memqcache_method = 'shmem'.
                                    # Defaults to 64MB.
                                    # (change requires restart)
memqcache_max_num_cache = 1000000
                                    # Total number of cache entries. Mandatory
                                    # if memqcache_method = 'shmem'.
                                    # Each cache entry consumes 48 bytes on shared memory.

                                    # Defaults to 1,000,000(45.8MB).
                                    # (change requires restart)
memqcache_expire = 0
                                    # Memory cache entry life time specified in seconds.
                                    # 0 means infinite life time. 0 by default.
                                    # (change requires restart)
memqcache_auto_cache_invalidation = on
                                    # If on, invalidation of query cache is triggered by c
orresponding
                                    # DDL/DML/DCL(and memqcache_expire).  If off, it is on
ly triggered
                                    # by memqcache_expire.  on by default.
                                    # (change requires restart)
memqcache_maxcache = 409600
                                    # Maximum SELECT result size in bytes.
                                    # Must be smaller than memqcache_cache_block_size. Def
aults to 400KB.
                                    # (change requires restart)
memqcache_cache_block_size = 1048576
                                    # Cache block size in bytes. Mandatory if memqcache_me
thod = 'shmem'.
                                    # Defaults to 1MB.
                                    # (change requires restart)
memqcache_oiddir = '/var/log/pgpool/oiddir'
                                    # Temporary work directory to record table oids
                                    # (change requires restart)
white_memqcache_table_list = ''
                                    # Comma separated list of table names to memcache
```

```
                                    # that don't write to database
                                    # Regexp are accepted
black_memqcache_table_list = ''
                                    # Comma separated list of table names not to memcache

                                    # that don't write to database
                                    # Regexp are accepted
```

Note You must reconfigure the following parameters:

```
listen_addresses = '0.0.0.0'
port = 8001
socket_dir = '/tmp'
reserved_connections = 0
pcp_listen_addresses = ''
pcp_port = 9898
pcp_socket_dir = '/tmp'
# - Backend Connection Settings -
backend_hostname0 = '127.0.0.1'
                                    # Host name or IP address to connect to for backend
0
backend_port0 = 3389
                                    # Port number for backend 0
backend_weight0 = 1
                                    # Weight for backend 0 (only in load balancing mode
)
backend_data_directory0 = '/data01/pg12_3389/pg_root'
                                    # Data directory for backend 0
backend_flag0 = 'ALWAYS_MASTER'
                                    # Controls various backend behavior
                                    # ALLOW_TO_FAILOVER, DISALLOW_TO_FAILOVER
                                    # or ALWAYS_MASTER
backend_application_name0 = 'server0'
                                    # walsender's application_name, used for "show pool
_nodes" command
backend_hostname1 = '127.0.0.1'
backend_port1 = 8002
backend_weight1 = 1
backend_data_directory1 = '/data01/pg12_8002/pg_root'
backend_flag1 = 'DISALLOW_TO_FAILOVER'
backend_application_name1 = 'server1'
# - Authentication -
enable_pool_hba = on
                                    # Use pool_hba.conf for client authentication
pool_passwd = 'pool_passwd'
                                    # File name of pool_passwd for md5 authentication.

                                    # "" disables pool_passwd.
                                    # (change requires restart)
allow_clear_text_frontend_auth = off
                                    # Allow Pgpool-II to use clear text password authen
tication
                                    # with clients, when pool_passwd does not
                                    # contain the user password
# - Concurrent session and pool size -
```

```
num_init_children = 128
                                    # Number of concurrent sessions allowed
                                    # (change requires restart)
max_pool = 4
                                    # Number of connection pool caches per connection
                                    # (change requires restart)
# - Life time -
child_life_time = 300
                                    # Pool exits after being idle for this many seconds

child_max_connections = 0
                                    # Pool exits after receiving that many connections

                                    # 0 means no exit
connection_life_time = 0
                                    # Connection to backend closes after being idle for
this many seconds
                                    # 0 means no close
client_idle_limit = 0
                                    # Client is disconnected after being idle for that
many seconds
                                    # (even inside an explicit transactions!)
                                    # 0 means no disconnection
#------------------------------------------------------------------------------
# LOGS
#------------------------------------------------------------------------------
# - Where to log -
log_destination = 'syslog'
                                    # Where to log
                                    # Valid values are combinations of stderr,
                                    # and syslog. Default to stderr.
log_connections = on
                                    # Log connections
log_standby_delay = 'if_over_threshold'
                                    # Log standby delay
                                    # Valid values are combinations of always,
                                    # if_over_threshold, none
#------------------------------------------------------------------------------
# FILE LOCATIONS
#------------------------------------------------------------------------------
pid_file_name = '/var/run/pgpool-II-12/pgpool.pid'
                                    # PID file name
                                    # Can be specified as relative to the"
                                    # location of pgpool.conf file or
                                    # as an absolute path
                                    # (change requires restart)
logdir = '/tmp'
                                    # Directory of pgPool status file
                                    # (change requires restart)
#------------------------------------------------------------------------------
# CONNECTION POOLING
#------------------------------------------------------------------------------
connection_cache = on
                                    # Activate connection pools
                                    # (change requires restart)
                                    # Semicolon separated list of queries
```

```
                                    # Semicolon separated list of queries
                                    # to be issued at the end of a session
                                    # The default is for 8.3 and later
reset_query_list = 'ABORT; DISCARD ALL'
#------------------------------------------------------------------------------
# LOAD BALANCING MODE
#------------------------------------------------------------------------------
load_balance_mode = on
                                    # Activate load balancing mode
                                    # (change requires restart)
ignore_leading_white_space = on
                                    # Ignore leading white spaces of each query
white_function_list = ''
                                    # Comma separated list of function names
                                    # that don't write to database
                                    # Regexp are accepted
black_function_list = 'currval,lastval,nextval,setval'
                                    # Comma separated list of function names
                                    # that write to database
                                    # Regexp are accepted
black_query_pattern_list = ''
                                    # Semicolon separated list of query patterns
                                    # that should be sent to primary node
                                    # Regexp are accepted
                                    # valid for streaming replication mode only.
database_redirect_preference_list = ''
                                    # comma separated list of pairs of database and nod
e id.
                                    # example: postgres:primary,mydb[0-4]:1,mydb[5-9]:2
'
                                    # valid for streaming replication mode only.
app_name_redirect_preference_list = ''
                                    # comma separated list of pairs of app name and nod
e id.
                                    # example: 'psql:primary,myapp[0-4]:1,myapp[5-9]:st
andby'
                                    # valid for streaming replication mode only.
allow_sql_comments = off
                                    # if on, ignore SQL comments when judging if load b
alance or
                                    # query cache is possible.
                                    # If off, SQL comments effectively prevent the judg
ment
                                    # (pre 3.4 behavior).
disable_load_balance_on_write = 'transaction'
                                    # Load balance behavior when write query is issued

                                    # in an explicit transaction.
                                    # Note that any query not in an explicit transactio
n
                                    # is not affected by the parameter.
                                    # 'transaction' (the default): if a write query is
issued,
                                    # subsequent read queries will not be load balanced

                                    # until the transaction ends.
```

```
                                    # 'trans_transaction': if a write query is issued,

                                    # subsequent read queries in an explicit transactio
n
                                    # will not be load balanced until the session ends.

                                    # 'always': if a write query is issued, read querie
s will
                                    # not be load balanced until the session ends.
statement_level_load_balance = off
                                    # Enables statement level load balancing
#------------------------------------------------------------------------------
# MASTER/SLAVE MODE
#------------------------------------------------------------------------------
master_slave_mode = on
                                    # Activate master/slave mode
                                    # (change requires restart)
master_slave_sub_mode = 'stream'
                                    # Master/slave sub mode
                                    # Valid values are combinations stream, slony
                                    # or logical. Default is stream.
                                    # (change requires restart)
# - Streaming -
sr_check_period = 3
                                    # Streaming replication check period
                                    # Disabled (0) by default
sr_check_user = 'nobody'
                                    # Streaming replication check user
                                    # This is necessary even if you disable streaming
                                    # replication delay check by sr_check_period = 0
sr_check_password = ''
                                    # Password for streaming replication check user
                                    # Leaving it empty will make Pgpool-II to first loo
k for the
                                    # Password in pool_passwd file before using the emp
ty password
sr_check_database = 'postgres'
                                    # Database name for streaming replication check
delay_threshold = 512000
                                    # Threshold before not dispatching query to standby
node
                                    # Unit is in bytes
                                    # Disabled (0) by default
#------------------------------------------------------------------------------
# HEALTH CHECK GLOBAL PARAMETERS
#------------------------------------------------------------------------------
health_check_period = 5
                                    # Health check period
                                    # Disabled (0) by default
health_check_timeout = 10
                                    # Health check timeout
                                    # 0 means no timeout
health_check_user = 'nobody'
                                    # Health check user
health_check_password = ''
                                    # Password for health check user
```

```
                                    # Password for health check user
                                    # Leaving it empty will make Pgpool-II to first loo
k for the
                                    # Password in pool_passwd file before using the emp
ty password
health_check_database = ''
                                    # Database name for health check. If '', tries 'pos
tgres' first,
health_check_max_retries = 60
                                    # Maximum number of times to retry a failed health
check before giving up.
health_check_retry_delay = 1
                                    # Amount of time to wait (in seconds) between retri
es.
connect_timeout = 10000
                                    # Timeout value in milliseconds before giving up to
connect to backend.
                                    # Default is 10000 ms (10 second). Flaky network us
er may want to increase
                                    # the value. 0 means no timeout.
                                    # Note that this value is not only used for health
check,
                                    # but also for ordinary connection to backend.
#------------------------------------------------------------------------------
# FAILOVER AND FAILBACK
#------------------------------------------------------------------------------
failover_on_backend_error = off
                                    # Initiates failover when reading/writing to the
                                    # backend communication socket fails
                                    # If set to off, pgpool will report an
                                    # error and disconnect the session.
relcache_expire = 0  # After the configuration file is restructured, we recommend that
you set this parameter to 1, reload the configuration file, and then set this paramete
r to 0 again. You also have the option to set this parameter to a specific point in ti
me.
                                    # Life time of relation cache in seconds.
                                    # 0 means no cache expiration(the default).
                                    # The relation cache is used for cache the
                                    # query result against PostgreSQL system
                                    # catalog to obtain various information
                                    # including table structures or if it's a
                                    # temporary table or not. The cache is
                                    # maintained in a pgpool child local memory
                                    # and being kept as long as it survives.
                                    # If someone modify the table by using
                                    # ALTER TABLE or some such, the relcache is
                                    # not consistent anymore.
                                    # For this purpose, cache_expiration
                                    # controls the life time of the cache.
relcache_size = 8192
                                    # Number of relation cache
                                    # entry. If you see frequently:
                                    # "pool_search_relcache: cache replacement happend"

                                    # in the pgpool log, you might want to increate thi
s number.
```

3. Configure the pool_passwd file.

> **Note** If you connect to your RDS instances by using pgpool, you must configure the pool_passwd file. This is because pgpool supports the authentication protocol of PostgreSQL.

```
cd /etc/pgpool-II-12
# Run the followong command:
#pg_md5 --md5auth --username=username password
# Generate the passwords of the digoal and nobody users. The passwords are automatically
written into the pool_passwd file.
pg_md5 --md5auth --username=digoal "xxxxxxx"
pg_md5 --md5auth --username=nobody "xxxxxxx"
```

4. Use the system to automatically generate the pool_passwd file.

```
cd /etc/pgpool-II-12
# cat pool_passwd
digoal:md54dd55116da69d3d03bf2e3a1470564f9
nobody:md54240e76623e2511d607f431043a5d1c1
```

5. Configure the pgpool_hba file.

```
cd /etc/pgpool-II-12
cp pool_hba.conf.sample pool_hba.conf
vi pool_hba.conf
host all all 0.0.0.0/0 md5
```

6. Configure the pcp.conf file.

> **Note** The pcp.conf file is used to manage the users and passwords of pgpool. This is not related to the users and passwords of your RDS instances.

```
cd /etc/pgpool-II-12
# pg_md5 abc  # In this command, you set the password to abc and encrypt it by using the
MD5 encryption algorithm.
900150983cd24fb0d6963f7d28e17f72
cp pcp.conf.sample pcp.conf
vi pcp.conf
# USERID:MD5PASSWD
manage:900150983cd24fb0d6963f7d28e17f72  # In this command, the manage user is used to ma
nage pgpool.
```

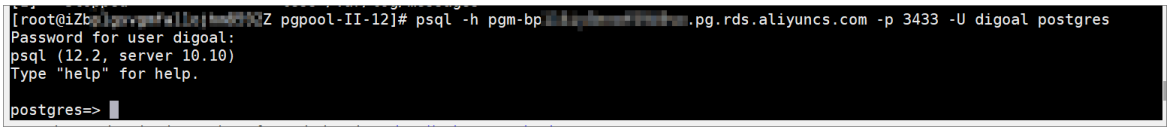
7. Start pgpool.

```
cd /etc/pgpool-II-12
pgpool -f ./pgpool.conf -a ./pool_hba.conf -F ./pcp.conf
```

> **Note** If you want to view the logs of pgpool, run the following command:
>
> ```
> less /var/log/messages
> ```

8. Use pgpool to connect to your RDS instances.

```
psql -h 127.0.0.1 -p 8001 -U digoal postgres
```



FAQ

- How do I test whether read/write splitting is successful?

  You can connect to your RDS instances by using pgpool and call the pg_is_in_recovery() function. Then, close the connection, re-establish a connection, and call the pg_is_in_recovery() function again. If you receive the false value and then the true value, pgpool routes requests to your primary RDS instance and then to your read-only RDS instances. This indicates that read/write splitting is successful.

- Does pgpool increase the latency?

  Pgpool increases the latency slightly. In the test environment you set up in this topic, the latency increases by about 0.12 milliseconds.

- How does pgpool check the latency and health on my read-only RDS instances?
  - If the WAL replay latency on a read-only RDS instance exceeds the specified limit, pgpool stops routing SQL requests to the read-only instance. Pgpool resumes routing SQL requests to the read-only instance only after it detects that the WAL replay latency on the read-only instance falls below the specified limit.

    > **Note** Connect to your primary RDS instance and query the location where the current WAL data record is written. This location is referred to as log sequence number (LSN) 1. Then, connect to a read-only RDS instance and query the location where the current WAL data record is replayed. This location is referred to as LSN 2. You can obtain the number of bytes between LSN 1 and LSN 2. This number indicates the latency.

  - Pgpool monitors the health of your read-only RDS instances. If a read-only instance is unhealthy, pgpool stops routing requests to the read-only instance.

- How do I stop pgpool and reload the configuration of pgpool?

  Run the `pgpool --help` command to obtain more information about the commands used in pgpool. Example:

```
cd /etc/pgpool-II-12
pgpool -f ./pgpool.conf -m fast stop
```

- How do I configure pgpool if more than one read-only RDS instance is attached to my primary RDS instance?

  Add the configurations of all the attached read-only RDS instances to the pgpool.conf file. Example:

```
backend_hostname1 = 'xx.xx.xxx.xx'
backend_port1 = 8002
backend_weight1 = 1
backend_data_directory1 = '/data01/pg12_8002/pg_root'
backend_flag1 = 'DISALLOW_TO_FAILOVER'
backend_application_name1 = 'server1'
backend_hostname2 = 'xx.xx.xx.xx'
backend_port1 = 8002
backend_weight1 = 1
backend_data_directory1 = '/data01/pg12_8002/pg_root'
backend_flag1 = 'DISALLOW_TO_FAILOVER'
backend_application_name1 = 'server1'
```

- How do I use pcp to obtain the status of my read-only RDS instances?

  To obtain the status of your read-only RDS instances by using pcp, run the following command:

```
# pcp_node_info -U manage -h /tmp -p 9898 -n 1 -v
Password: Enter the password.
Hostname               : 127.0.0.1
Port                   : 8002
Status                 : 2
Weight                 : 0.500000
Status Name            : up
Role                   : standby
Replication Delay      : 0
Replication State      :
Replication Sync State :
Last Status Change     : 2020-02-29 00:20:29
```

- Which listening ports are used by pgpool for read/write splitting?

  The following listening ports are used by pgpool for read/write splitting:

  - Primary RDS instance: Port 3389
  - Secondary RDS instance: Port 8002
  - pgpool: Port 8001
  - pcp: Port 9898

32.5. User preference recommendation system

This topic describes how to use the HyperLogLog (HLL) plug-in to design a recommendation system, which recommends content to a user based on the preferences of the user. The recommendation system is implemented based on similarity computing of PostgreSQL.

Context

A recommendation system can be used to increase user stickiness and conversion rate for the following websites:

- E-commerce websites
- Music websites
- News websites

- App websites

In this topic, a music website is used as an example to describe how to design a recommendation system and elaborate on the differences between the conventional design and the design that is based on HHL and similarity computing of PostgreSQL.

Design background

1. After a user (uid) finishes listening to a song (vid), the recommendation system binds a tag (tagid) to the song. More than one tag is allowed per song.

   ```
   uid ->> tags ->> musics
   ```

2. The recommendation system obtains the popularity of each tag based on the number of songs to which the tag is bound.

   ```
   tag(count distinct music)
   ...
   ```

3. The recommendation system obtains the top 5 tags and their recommendation weights.

   ```
   tag1:40%
   tag2:20%
   tag3:15%
   tag4:15%
   tag5:10%
   ```

4. The recommendation system excludes the songs to which the user finishes listening. The recommendation system obtains a library of songs to which the user has not listened. Then, the recommendation system recommends new songs to the user based on the recommendation weights of the songs in the library. For example, the recommendation weights can be the rankings of the songs in descending order.

Conventional design

The conventional design is suitable for all types of databases. However, if your database system has a large amount of data, the conventional design may use aggregate queries that run slowly. Example:

```
create table t_like(
uid int,  -- The ID of a user.
tagid int,  -- The ID of the tag that is bound to a song.
vid int,   -- The ID of a song.
mod_time timestamp,  -- The time of the last update. An update is triggered only when one day
has passed since the last update.
primary key (uid,tagid,vid)
);
insert into t_like values (:uid, :tagid, :vid, :mod_time)
 on conflict (uid,tagid,vid) do update
set mod_time=excluded.mod_time
where
excluded.mod_time - t_like.mod_time > interval '1 day'
;
-- Obtain the top 10 tags over the last day.
select tagid, count(*) from t_like
where uid=:uid
and now()-mod_time < interval '1 day'
group by tagid
order by count(*) desc limit 10;
```

The following code snippet is an example of stress testing:

```
vi test.sql
\set uid random(1,50000)
\set tagid random(1,5000)
\set vid random(1,10000000)
insert into t_like values (:uid, :tagid, :vid, now())
 on conflict (uid,tagid,vid) do update
set mod_time=excluded.mod_time
where
excluded.mod_time - t_like.mod_time > interval '1 day';
pgbench -M prepared -n -r -P 1 -f ./test.sql -c 32 -j 32 -T 240
transaction type: ./test.sql
scaling factor: 1
query mode: prepared
number of clients: 32
number of threads: 32
duration: 240 s
number of transactions actually processed: 80975327
latency average = 0.095 ms
latency stddev = 0.340 ms
tps = 337396.279382 (including connections establishing)
tps = 337406.018908 (excluding connections establishing)
statement latencies in milliseconds:
         0.000  \set uid random(1,50000)
         0.000  \set tagid random(1,5000)
         0.000  \set vid random(1,10000000)
         0.094  insert into t_like values (:uid, :tagid, :vid, now())
db1=# select tagid, count(*) from t_like
where uid=1
and now()-mod_time < interval '1 day'
group by tagid
order by count(*) desc limit 10;
 tagid | count
-------+-------
  2519 |     4
  3049 |     4
  3648 |     4
  1777 |     3
  1352 |     3
  1491 |     3
  1064 |     3
   572 |     3
   692 |     3
   301 |     3
(10 rows)
Time: 3.947 ms
```

Design based on HLL and similarity computing of PostgreSQL

The design based on HLL and similarity computing of PostgreSQL stores the IDs of the songs to which each user finishes listening. This design has the following benefits over the conventional design:

- Stores a small amount of data by using approximate clustered hash values in place of actual values.
- Supports index-based queries without the need for computing. The index-based queries allow your database system to respond to queries within milliseconds.

- Supports operations that can be used for sliding window computing to meet more diversified business requirements. These operations include HASH UNION and HASH ADD.

> **Note** For more information about how to use the HLL plug-in, see Use the hll plug-in.

1. The recommendation system maintains one HLL per tag. The HLL contains a hash value that consists of the IDs of all the songs to which each user finishes listening.

```
create table t_like (
uid int,
tagid int, -- The ID of the tag.
w1 hll, w1_mod_time timestamp, -- The hash value that consists of the IDs of the songs to
which the user finishes listening on a Monday within the tag.
w2 hll, w2_mod_time timestamp, -- The hash value that consists of the IDs of the songs to
which the user finishes listening on a Tuesday within the tag.
w3 hll, w3_mod_time timestamp, -- The hash value that consists of the IDs of the songs to
which the user finishes listening on a Wednesday within the tag.
w4 hll, w4_mod_time timestamp, -- The hash value that consists of the IDs of the songs to
which the user finishes listening on a Thursday within the tag.
w5 hll, w5_mod_time timestamp, -- The hash value that consists of the IDs of the songs to
which the user finishes listening on a Friday within the tag.
w6 hll, w6_mod_time timestamp, -- The hash value that consists of the IDs of the songs to
which the user finishes listening on a Saturday within the tag.
w7 hll, w7_mod_time timestamp, -- The hash value that consists of the IDs of the songs to
which the user finishes listening on a Sunday within the tag.
whole hll,                  -- The hash value that consists of the IDs of the songs to w
hich the user finishes listening from a Monday to Sunday period within the tag.
primary key (uid,tagid)
);
```

> **Note** You can specify the w1 to w7 day fields based on your business requirements. For example, if you want to view only the data of a specific day, you can specify only one of the w1 to w7 day fields.

2. After a user finishes listening to a song, the recommendation system writes the information of the song to the current day field. If the field has a value but the value is not last modified on the current day, the recommendation system overwrites the existing value by using the new value. Otherwise, the recommendation system appends a hash value to the current day field. The hash value consists of both the existing value and the new value. The preceding logic is implemented by using the `INSERT INTO ON CONFLICT` syntax.

```
-- Configure the hash value that consists of the IDs of all the songs to which each user
finishes listening within a tag.
insert into t_like (
uid,
tagid,
w5,
w5_mod_time,
whole
)
values (
1,  -- uid
200,  -- The ID of the tag.
hll_hash_integer(12346)||hll_empty(),  -- The ID of the song to which the user finishes l
istening. If the user finishes listening to more than one song, the subsequent coding con
tinues.
now(),
hll_hash_integer(12346)||hll_empty()   -- The ID of the song to which the user finishes l
istening.
)
on conflict (uid,tagid)
do update
set w5=
case
when date(t_like.w5_mod_time) <> current_date
then excluded.w5
else hll_union(coalesce(t_like.w5,hll_empty()), excluded.w5)
end,
w5_mod_time = excluded.w5_mod_time,
whole = hll_union(coalesce(t_like.whole,hll_empty()), excluded.whole)
where
hll_union(coalesce(t_like.w5,hll_empty()), excluded.w5) <> coalesce(t_like.w5,hll_empty()
)
or
hll_union(coalesce(t_like.whole,hll_empty()), excluded.whole) <> coalesce(t_like.whole,hl
l_empty())
;
```

Note You can perform HLL UNION operations to merge all the data updates to the data records of a user within a tag at a time.

3. Query the top 10 tags of the user with the uid value being 1 over the last two days. Example:

```
select tagid,
hll_cardinality( hll_union(coalesce(w4,hll_empty()), coalesce(w5,hll_empty())) ) as vids

from t_like
where uid = 1
order by 2 desc limit 10;
 tagid | vids
-------+------
   200 |    2
(1 row)
```

4. Create an index. Example:

```
create index idx_t_like_1 on t_like (uid, hll_cardinality( hll_union(coalesce(w4,hll_empt
y()), coalesce(w5,hll_empty())) ));
```

5. View the query plan. Example:

```
postgres=# explain select tagid,
hll_cardinality( hll_union(coalesce(w4,hll_empty()), coalesce(w5,hll_empty())) ) as vids

from t_like
where uid = 1
order by 2 desc limit 10;
                                        QUERY PLAN

-----------------------------------------------------------------------------------------
--
 Limit  (cost=0.11..0.15 rows=1 width=12)
   ->  Index Scan Backward using idx_t_like_1 on t_like  (cost=0.11..0.15 rows=1 width=12
)
         Index Cond: (uid = 1)
(3 rows)
```

6. Write tens of millions of data records and perform a stress test. Example:

```
vi test.sql
\set uid random(1,50000)
\set tagid random(1,5000)
\set vid random(1,10000000)
insert into t_like (
uid,
tagid,
w5,
w5_mod_time,
whole
)
values (
:uid,
:tagid,
hll_hash_integer(:vid)||hll_empty(),
now(),
hll_hash_integer(:vid)||hll_empty()
)
on conflict (uid,tagid)
do update
set w5=
case
when date(t_like.w5_mod_time) <> current_date
then excluded.w5
else hll_union(coalesce(t_like.w5,hll_empty()), excluded.w5)
end,
w5_mod_time = excluded.w5_mod_time,
whole = hll_union(coalesce(t_like.whole,hll_empty()), excluded.whole)
where
hll_union(coalesce(t_like.w5,hll_empty()), excluded.w5) <> coalesce(t_like.w5,hll_empty()
)
or
hll_union(coalesce(t_like.whole,hll_empty()), excluded.whole) <> coalesce(t_like.whole,hl
l_empty())
;
pgbench -M prepared -n -r -P 1 -c 32 -j 32 -T 120 -f ./test.sql
```

The following test result is obtained:

```
transaction type: ./test.sql
scaling factor: 1
query mode: prepared
number of clients: 32
number of threads: 32
duration: 120 s
number of transactions actually processed: 24636321
latency average = 0.156 ms
latency stddev = 0.339 ms
tps = 205301.110313 (including connections establishing)
tps = 205354.851711 (excluding connections establishing)
statement latencies in milliseconds:
         0.001  \set uid random(1,5000000)
         0.001  \set tagid random(1,5000)
         0.000  \set vid random(1,10000000)
         0.154  insert into t_like (
```

7. Query the tags of a user based on the tag popularity. Example:

```
postgres=# select tagid,
hll_cardinality( hll_union(coalesce(w4,hll_empty()), coalesce(w5,hll_empty())) ) as vids

from t_like
where uid = 1
order by 2 desc limit 10;
 tagid | vids
-------+------
   200 |    2
  1413 |    1
  1996 |    1
  2642 |    1
  3664 |    1
  4340 |    1
(6 rows)
Time: 0.688 ms
```

> **Note** This query is responded in 0.688 milliseconds.

You can use the preceding design method to implement other business logic in the recommendation system. For example, you can use the following design to filter songs to which a user finishes listening:

- Determine whether the ID of a song is included in the specified hash value by using fuzzy match. Example:

```
postgres=# select whole || hll_hash_integer(1) = whole
from
t_like
where uid=1 and tagid=200;  -- If the value false is returned, the song with the vid value
being 1 is not included.
 ? column?
----------
 f
(1 row)
postgres=# select whole || hll_hash_integer(12345) = whole
from
t_like
where uid=1 and tagid=200;   -- If the value true is returned, the song with the vid value
being 12345 is included.
 ? column?
----------
 t
(1 row)
```

- Determine whether the ID of a song is included in the specified hash value by using exact match. The following example shows how to create a table:

```
create table t_like_lossless (
uid int,
vid int,
primary key (uid,vid)
);
```

> **Note** If you run the query based on a primary key, the query runs fast.

32.6. Use an ApsaraDB RDS for PostgreSQL instance as the real-time data analytics instance of an ApsaraDB RDS for MySQL instance

This topic describes how to migrate the data of an ApsaraDB RDS for MySQL instance to an ApsaraDB RDS for PostgreSQL instance and use the ApsaraDB RDS for PostgreSQL instance as the real-time data analytics instance of the ApsaraDB RDS for MySQL instance.

Context

If you want to analyze the data of an ApsaraDB RDS for MySQL instance by using the features of ApsaraDB RDS for PostgreSQL or you want to process spatio-temporal data or analyze user profile data by using GIS, you can use Data Transmission Service (DTS) to migrate the data of the ApsaraDB RDS for MySQL instance to an ApsaraDB RDS for PostgreSQL instance. Then, you can use the ApsaraDB RDS for PostgreSQL instance as the real-time data analytics instance of the ApsaraDB RDS for MySQL instance.

Prerequisites

- An ApsaraDB RDS for MySQL instance is created. This instance is known as the source instance. For more

information, see Create an ApsaraDB RDS for MySQL instance.

- An ApsaraDB RDS for PostgreSQL instance is created. This instance is known as the destination instance. For more information, see Create an ApsaraDB RDS for PostgreSQL instance.
- The available storage space of the destination instance must be greater than the storage space that is occupied on the source instance.

Procedure

1. Prepare test data on the source instance.
2. Create a migration task.
3. Migrate the data of the source instance to the destination instance..

Prepare test data on the source instance

1. Connect to the source instance.

```
mysql -h <Endpoint> -u <Username> -P <Port number> -p
```

2. Create a test database named db1.

```
CREATE DATABASE db1;
```

3. Log on to the db1 database.

```
USE db1;
```

4. Create a test table named test_mm and a test table named test_innodb.

```
CREATE TABLE `test_mm` (
`id` INT (11) NOT NULL AUTO_INCREMENT,
`user_id` VARCHAR (20) NOT NULL,
`group_id` INT (11) NOT NULL,
`create_time` datetime NOT NULL,
PRIMARY KEY (`id`),KEY `index_user_id` (`user_id`) USING HASH
) ENGINE = innodb AUTO_INCREMENT = 1
DEFAULT CHARSET = utf8;
```

```
CREATE TABLE `test_innodb` (
`id` INT (11) NOT NULL AUTO_INCREMENT,
`user_id` VARCHAR (20) NOT NULL,
`group_id` INT (11) NOT NULL,
`create_time` datetime NOT NULL,
PRIMARY KEY (`id`),
KEY `index_user_id` (`user_id`) USING HASH
) ENGINE = innodb AUTO_INCREMENT = 1
DEFAULT CHARSET = utf8;
```

5. Create a rand_ string function that is used to generate random strings.

```
delimiter $$
CREATE FUNCTION rand_string(n int) RETURNS varchar(255)
begin
declare chars_str varchar(100)
default "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
declare return_str varchar(255) default "";
declare i int default 0;
while i < n do
set return_str=concat(return_str,substring(chars_str,floor(1+rand()*62),1));
set i= i+1;
end while;
return return_str;
end $$
delimiter ;
```

6. Create a stored procedure that is used to insert test data.

```
delimiter $$
CREATE PROCEDURE `insert_data`(IN n int)
BEGIN
DECLARE i INT DEFAULT 1;
WHILE (i <= n ) DO
INSERT into test_mm (user_id,group_id,create_time ) VALUEs
(rand_string(20),FLOOR(RAND() * 100) ,now() );
set i=i+1;
END WHILE;
END $$
delimiter ;
```

7. Call the stored procedure that you created.

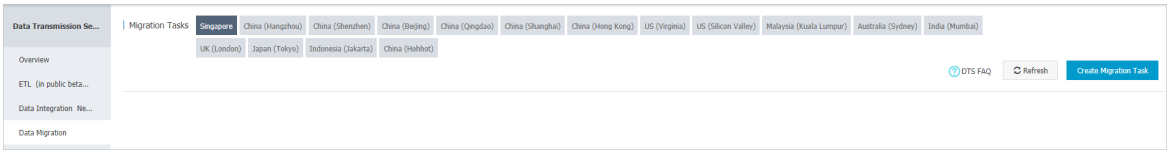
```
CALL insert_data(1000000);
INSERT INTO test_innodb SELECT * FROM test_mm;
```

Create a migration task

1. Before you create a migration task, you must log on to the ApsaraDB RDS console and create a database in the destination instance. The database that you create is the destination database to which you can migrate the data of the source instance. For more information, see Create a database on an ApsaraDB RDS for PostgreSQL instance.

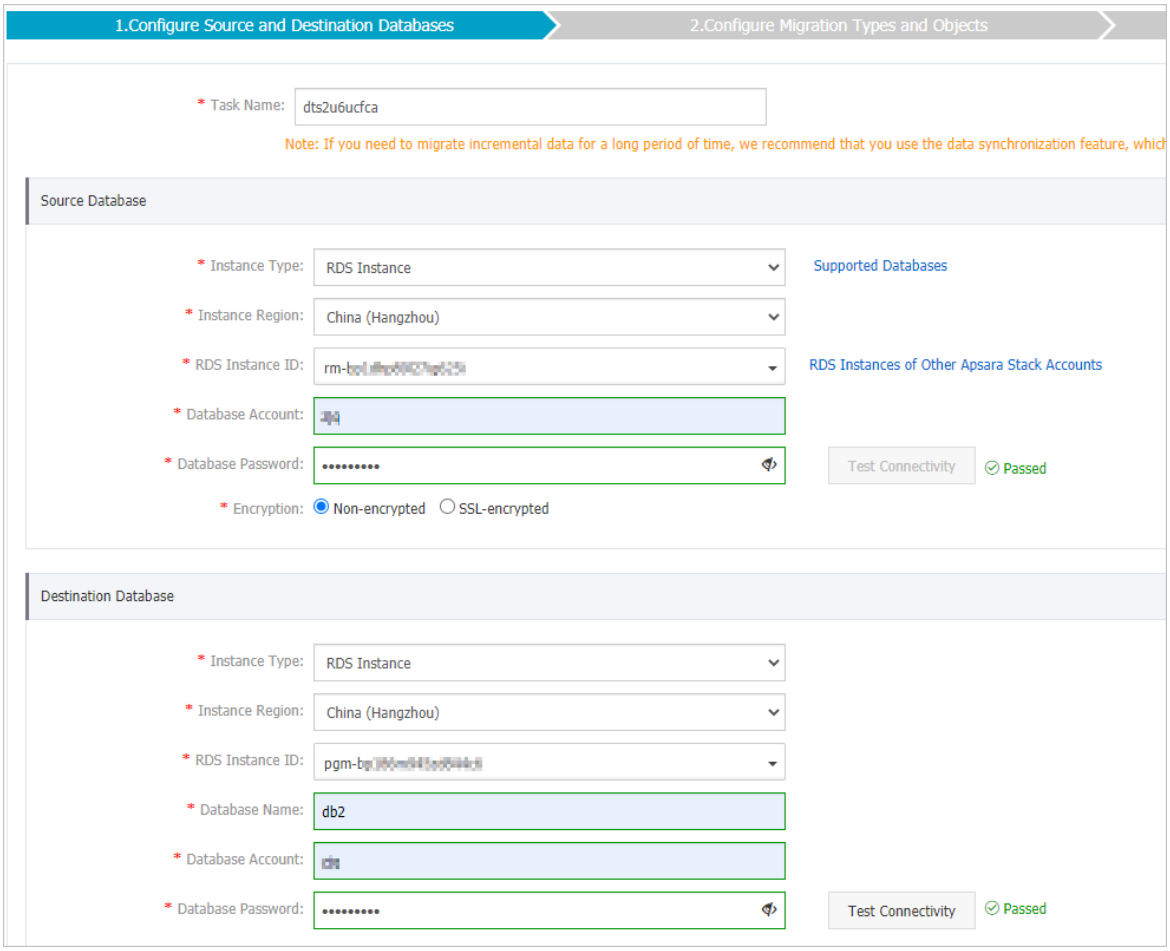
   > **Note** In the example provided in this section, the database that you create is named db2.

2. Log on to the DTS console
3. In the left-side navigation pane, click **Data Migration**.
4. In the upper **Migration Tasks** section of the page, select the region where the destination instance resides.



5. In the upper-right corner of the page, click **Create Migration Task**.

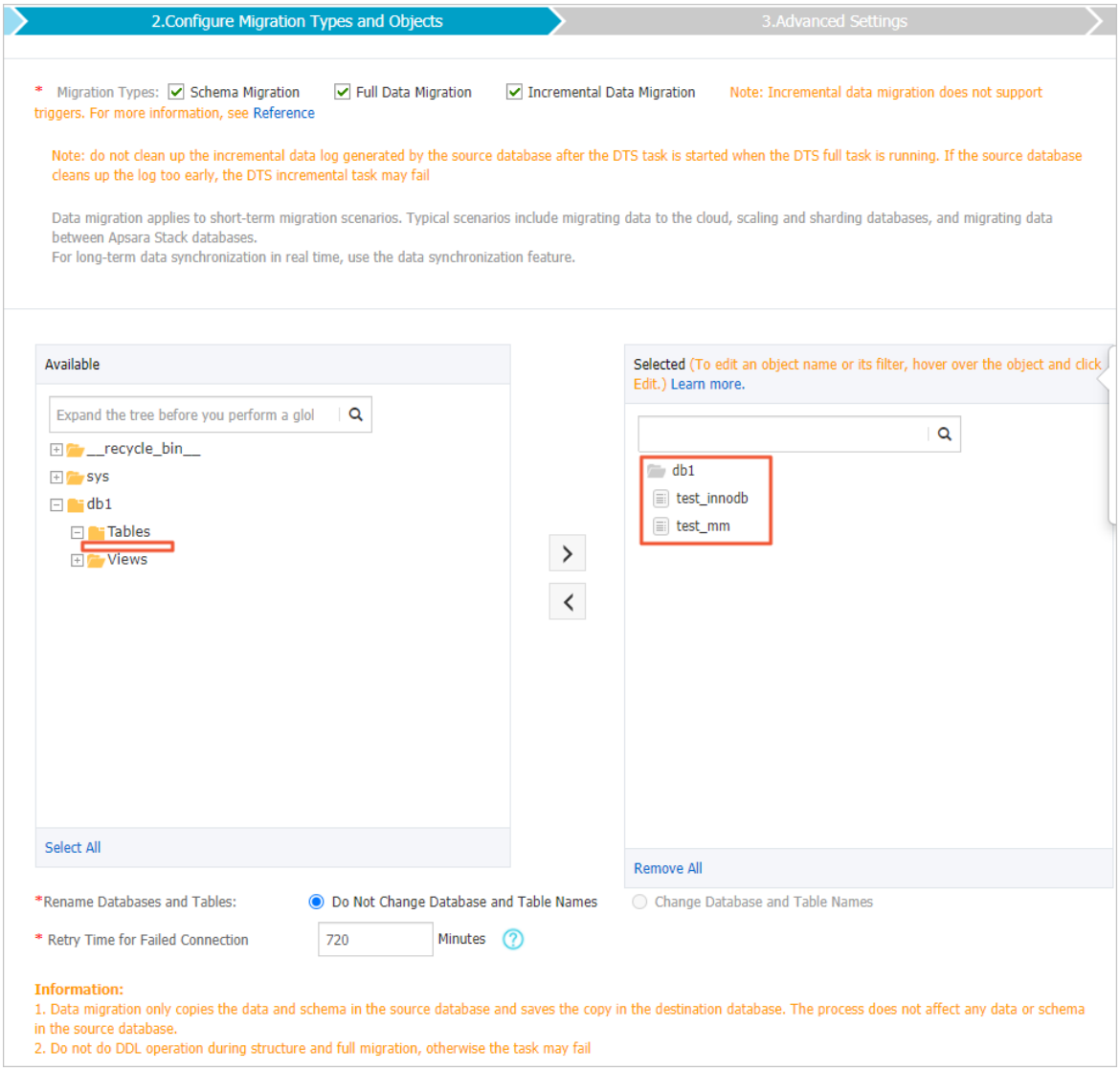
6. Configure the source and destination databases.



Section	Parameter	Description
N/A	**Task Name**	DTS automatically generates a task name. We recommend that you specify an informative name for easy identification. You do not need to use a unique task name.
	Instance Type	Select **RDS Instance**.
	Instance Region	Select the region where the source instance resides.
	RDS Instance ID	Select the ID of the source instance.

Section	Parameter	Description
Source Database	Database Account	Enter the username of the account has permissions on the source database in the source instance. The account must have the following permissions: ○ The SELECT permission that is required for the schema migration. ○ The SELECT permission that is required for full data migration. ○ The REPLICATION CLIENT permission, REPLICATION SLAVE permission, SHOW VIEW permission, and SELECT permission that are required for incremental data migration.
	Database Password	Enter the password of the database account.
	Connection Method	Select **Non-encrypted** or **SSL-encrypted**. If you select **SSL-encrypted**, you must enable SSL encryption for the RDS instance before you configure the data migration task. For more information, see Configure SSL encryption for an ApsaraDB RDS for MySQL instance.
Destination Database	Instance Type	Select **RDS Instance**.
	Instance Region	Select the region where the destination instance resides.
	RDS Instance ID	Select the ID of the destination instance.
	Database Account	Enter the username of the account that has permissions on the destination database in the destination instance. The account must have the following permissions: ○ LOGIN permission. ○ CONNECT permission and CREATE permission on the destination database. ○ CREATE permission on the schema of the destination database.
	Database Password	Enter the password of the database account.

7. Click **Test Connectivity** next to the Database Password in the Source Database section and the Source Database section. You can proceed with the next step only after the source instance and the destination instance passes the connectivity test.
8. In the lower-right corner of the page, click **Set Whitelist and Next**.
9. Specify the migration types and the tables that you want to migrate.

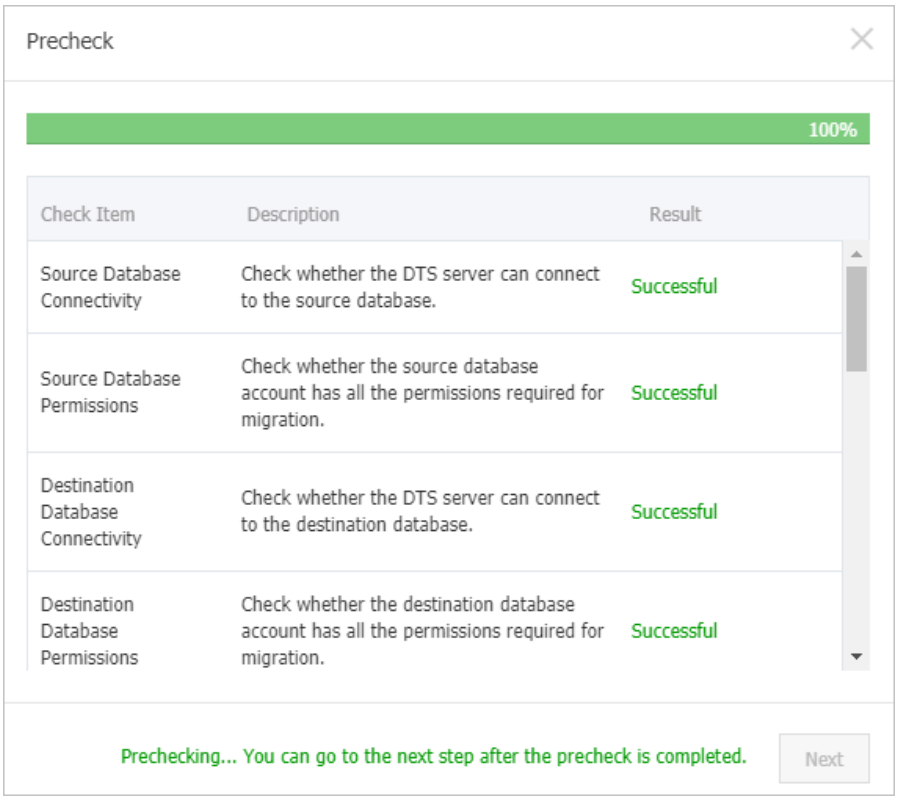


Parameter	Description
Migration Types	Select Schema Migration, Full Data Migration, and Incremental Data Migration.
Available	Select the tables that you want to migrate. For this example, select the test_innodb table and the test_mm table.
Selected	View the selected tables.
Rename Databases and Tables	The default value is **Do Not Change Database and Table Names**. If you want to modify the names of multiple databases and tables at a time, select **Change Database and Table Names**. In the lower-right corner of the page, click **Advanced Settings** to modify the names of multiple databases and tables at a time.
Retry Time for Failed Connection	The default value is 720 minutes. You do not need to change the default value.

10. In the lower part of the page, click **Precheck**.

> **Note**
>
> - A precheck is required before the migration task starts. The migration task starts only after it passes the precheck.
> - If the migration task fails the precheck, you can click the icon next to each failed check item to view the details about the failure.
>   - You can troubleshoot the failed check items as instructed. Then, you must run the precheck again.
>   - If you do not want to troubleshoot the failed check items, you can ignore them and perform a precheck again.

11. When **Precheck Passed 100%** is displayed on the Precheck page, click **Next**.



12. In the **Confirm Settings** dialog box, specify the Channel Specification and select Data Transmission Service.

> **Note** DTS provides various migration specifications. The migration speed varies based on the migration specifications that you select based on your business requirements. For more information, see Specifications of data migration instances.

13. Select **Data Transmission Service (Pay-As-You-Go) Service Terms**.
14. Click **Buy and Start** to start the migration task. You can view the progress of the migration task in the migration task list.

Migrate the data of the source instance to the destination instance.

1. View the progress of the full data migration operation.
   i. Connect to the destination instance.

```
psql -h <Endpoint> -U <Username> -p <Port number> -d db2
```

> **Note** The db2 database is created in the "Create a migration task" section to run as the destination database. Therefore, when you connect to the destination instance, you can specify the database name as db2.

   ii. Run the `\dn` command to check whether the db1 database is mapped as a schema named db1 in the db2 database.

```
db2=> \dn
   List of schemas
  Name  |   Owner
--------+------------
 db1    | test1
 public | pg*******
(2 rows)
```

> **Note** After the data of the db1 database in the source instance is migrated to the db2 database in the destination instance, a schema named db1 is generated in the db2 database to accommodate the data of the db1 database.

   iii. Run the `\dt+ db1.*` command to view the status of the tables in the db1 database.

```
db2=> \dt+ db1.*
                            List of relations
 Schema |    Name     | Type  | Owner | Persistence | Size  | Description
--------+-------------+-------+-------+-------------+-------+-------------
 db1    | test_innodb | table | test1 | permanent   | 65 MB |
 db1    | test_mm     | table | test1 | permanent   | 65 MB |
(2 rows)
```

iv. Run the following commands to query the number of data records in the test_innodb table and the test_mm table:

```
# Query the number of data records in the test_innodb table.
SELECT COUNT(*) FROM db1.test_innodb;
# Query the number of data records in the test_mm table.
SELECT COUNT(*) FROM db1.test_mm;
```

Result:

```
db2=> SELECT COUNT(*) FROM db1.test_innodb;
  count
---------
 1000000
(1 row)
db2=> SELECT COUNT(*) FROM db1.test_mm;
  count
---------
 1000000
(1 row)
```

> **Note**
>
> After the data of the db1 database in the source instance is migrated to the db2 database in the destination instance, a schema named db1 is generated in the db2 database to accommodate the data of the db1 database. Therefore, when you run a query in the destination instance to query the data of the db1 database, you must specify the schema name as db1.
>
> If you do not want to specify the schema every time when you query the data of the db1 database, you can set the search_path parameter.
>
> ```
> db2=> show search_path;
>    search_path
> -----------------
>  "$user", public
> (1 row)
> db2=> set search_path = db1, "$user", public;
> SET
> db2=> show search_path;
>      search_path
> ----------------------
>  db1, "$user", public
> (1 row)
> ```

2. Test whether the data that is inserted into the source database is continuously migrated to the destination database.

i. Insert data into the source database.

```
INSERT INTO test_innodb (user_id, group_id, `create_time`) VALUES ('testuser', 1, '20
21-07-29 12:00:00');
```

Result

```
mysql> INSERT INTO test_innodb (user_id, group_id, `create_time`) VALUES ('testuser',
1, '2021-07-29 12:00:00');
Query OK, 1 row affected (0.04 sec)
mysql> SELECT * FROM test_Innodb WHERE user_id = 'testuser';
+---------+----------+----------+---------------------+
| id      | user_id  | group_id | create_time         |
+---------+----------+----------+---------------------+
| 1000001 | testuser |        1 | 2021-07-29 12:00:00 |
+---------+----------+----------+---------------------+
1 row in set (0.03 sec)
```

ii. Check whether the data that is inserted into the source database is migrated to the destination database.

```
db2=> SELECT * FROM test_Innodb WHERE user_id = 'testuser';
   id    | user_id  | group_id |     create_time
---------+----------+----------+---------------------
 1000001 | testuser |        1 | 2021-07-29 12:00:00
(1 row)
```

3. Test whether the data that is updated in the source database is continuously migrated to the destination database.

i. Update data in the source database.

```
UPDATE test_innodb set group_id = 2 WHERE user_id = 'testuser';
```

Result:

```
mysql> UPDATE test_innodb set group_id = 2 WHERE user_id = 'testuser';
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT * FROM test_innodb WHERE user_id = 'testuser';
+---------+----------+----------+---------------------+
| id      | user_id  | group_id | create_time         |
+---------+----------+----------+---------------------+
| 1000001 | testuser |        2 | 2021-07-29 12:00:00 |
+---------+----------+----------+---------------------+
1 row in set (0.03 sec)
```

ii. Check whether the data that is updated in the source database is migrated to the destination database.

```
db2=> SELECT * FROM test_innodb WHERE user_id = 'testuser';
   id    | user_id  | group_id |     create_time
---------+----------+----------+---------------------
 1000001 | testuser |        2 | 2021-07-29 12:00:00
(1 row)
```

4. Test whether the data that is deleted from the source instance is continuously migrated to the

destination database.

i. Delete data from the source database.

```
DELETE FROM test_innodb WHERE user_id = 'testuser';
```

Result:

```
mysql> DELETE FROM test_innodb WHERE user_id = 'testuser';
Query OK, 1 row affected (0.03 sec)
mysql> SELECT * FROM test_innodb WHERE user_id = 'testuser';
Empty set (0.03 sec)
mysql> SELECT MAX(id) FROM test_innodb;
+---------+
| MAX(id) |
+---------+
| 1000000 |
+---------+
1 row in set (0.03 sec)
```

> **Note** When no data is inserted, the maximum value of the id field in the return result is 1000000. After data is inserted, the value of the id field increases to 1000001. After data is deleted, the value of the id field decreases to 1000000.

ii. Check whether the data that is deleted from the source database is also deleted from the destination database.

```
db2=> SELECT * FROM test_innodb WHERE user_id = 'testuser';
 id | user_id | group_id | create_time
----+---------+----------+-------------
(0 rows)
db2=> SELECT MAX(id) FROM test_innodb;
   max
---------
 1000000
(1 row)
```

32.7. Use ShardingSphere to develop ApsaraDB RDS for PostgreSQL

ShardingSphere is an open source ecosystem that consists of a set of distributed database middleware solutions.

Prerequisites

All PostgreSQL versions that are used with ApsaraDB RDS support ShardingSphere.

Context

ShardingSphere is suitable for services that run in databases with thorough, well-organized logical sharding. ShardingSphere provides the following features:

- Data sharding
  - Database sharding and table sharding

  - Read/write splitting
  - Sharding strategy customization
  - Centerless distributed primary key
- Distributed transaction
  - Unified transaction API
  - XA transaction
  - BASE transaction
- Database orchestration
  - Dynamic configuration
  - Orchestration and governance
  - Data encryption
  - Tracing and observability
  - Elastic scaling out (planning)

For more information, see the ShardingSphere documentation.

ShardingSphere products

ShardingSphere includes three independent products. You can choose the product that best meets your business requirements. The following table describes these products.

Category	Sharding-JDBC	Sharding-Proxy	Sharding-Sidecar
Supported database engine	All JDBC-compatible database engines such as MySQL, PostgreSQL, Oracle, and SQL Server	MySQL and PostgreSQL	MySQL and PostgreSQL
Number of connections consumed	High	Low	High
Supported heterogeneous language	Java	Not limited	Not limited
Impact on performance	Low	Moderate	Low
Centerless	Supported	Not supported	Supported
Stateless API	Not supported	Supported	Not supported

Modify configuration files

1. On your Elastic Compute Service (ECS) instance, run the following commands to go to the directory in which the configuration files are stored.

   ```
   cd apache-shardingsphere-incubating-4.0.0-sharding-proxy-bin
   cd conf
   ```

2. Run the **ll** command to view all files that are stored in the directory.

   The information similar to the following command output is displayed:

```
total 24
-rw-r--r-- 1 501 games 3019 Jul 30  2019 config-encrypt.yaml
-rw-r--r-- 1 501 games 3582 Apr 22  2019 config-master_slave.yaml
-rw-r--r-- 1 501 games 4278 Apr 22  2019 config-sharding.yaml
-rw-r--r-- 1 501 games 1918 Jul 30  2019 server.yaml
```

Note

- config-encrypt.yaml: the data encryption configuration file.
- config-master_slave.yaml: the read/write splitting configuration file.
- config-sharding.yaml: the data sharding configuration file.
- server.yaml: the common configuration file.

3. Modify the configuration files.

Note For more information about the configuration files, see the ShardingSphere documentation. This topic only describes how to modify the data sharding configuration file and the common configuration file.

- Example on modifying the data sharding configuration file:

```
schemaName: #The name of the logical data source.
dataSources: #The configuration of the data source. You can configure more than
one data source by using the data_source_name element.
  <data_source_name>: #You do not need to configure a database connection pool.
This is different in Sharding-JDBC.
    url: #The URL that is used to connect to your RDS instance.
    username: #The username that is used to log on to your RDS instance.
    password: #The password that is used to log on to your RDS instance.
    connectionTimeoutMilliseconds: 30000 #The connection timeout period, which
is measured in milliseconds.
    idleTimeoutMilliseconds: 60000 #The idle-connection reclaiming timeout peri
od, which is measured in milliseconds.
    maxLifetimeMilliseconds: 1800000 #The maximum connection time to live (TTL)
, which is measured in milliseconds.
    maxPoolSize: 65 #The maximum number of connections that are allowed.
shardingRule: #You do not need to configure a sharding rule. The sharding rule
is the same as the sharding rule in Sharding-JDBC.
```

- Example on modifying the common configuration file:

- Proxy properties

```
#You do not need to configure the proxy properties that you can find in Sharding-JDBC.
props:
  acceptor.size: #The number of worker threads that receive requests from the client. The default number is equal to the number of cores multiplied by 2.
  proxy.transaction.type: #The type of transaction processed by the proxy. Valid values: LOCAL, XA, and BASE. Default value: LOCAL. The value XA specifies to use Atomikos as the transaction manager. The value BASE specifies to copy the .jar package that implements the ShardingTransactionManager operation to the lib directory.
  proxy.opentracing.enabled: #Specifies whether to enable the link tracing feature. By default, this feature is disabled.
  check.table.metadata.enabled: #Specifies whether to check the consistency of metadata among the shaded tables during startup. Default value: false.
  proxy.frontend.flush.threshold: #The number of packets that are returned in a batch during a complex query.
```

- Permission verification

```
This part of the configuration is used to verify your permissions when you attempt to log on to Sharding-Proxy. After you configure the username, password, and authorized databases, you must use the correct username and password to log on to Sharding-Proxy from the authorized databases.
authentication:
  users:
    root: # The username of the root user.
      password: root# The password of the root user.
    sharding: # The username of the root user.
      password: sharding# The password of the sharding user.
      authorizedSchemas: sharding_db, masterslave_db # The databases on which the specified user has permissions. If you want to specify more than one database, separate them with commas (,). You are granted the permissions of the root user by default. This means that you can access all databases.
```

Set up a test environment

- On your ECS instance, install a database client that runs Java.

```
yum install -y java
```

- Configure your RDS instance, which runs PostgreSQL 12.
  - Create an account whose username is r1.
  - Set the password of the account to "PW123321!".
  - Create the following databases whose owners are user r1: db0, db1, db2, and db3.
  - Add the IP address of your ECS instance to an IP address whitelist of the RDS instance.

> **Note**
>
> - For more information about how to create an ApsaraDB RDS for PostgreSQL instance, database, and account, see Create an ApsaraDB RDS for PostgreSQL instance and Create a database and an account on an ApsaraDB RDS for PostgreSQL instance.
> - For more information about how to configure an IP address whitelist, see Configure an IP address whitelist for an ApsaraDB RDS for PostgreSQL instance.

- Run the **vi** command on the common configuration file.

```
vi /root/apache-shardingsphere-incubating-4.0.0-sharding-proxy-bin/conf/server.yaml
```

- Configure the common configuration file based on the following example:

```
authentication:
  users:
    r1:
      password: PW123321!
      authorizedSchemas: db0,db1,db2,db3
props:
  executor.size: 16
  sql.show: false
```

Test horizontal sharding

1. Modify the data sharding configuration file.
   i. Run the **vi** command to open the common configuration file.

```
vi /root/apache-shardingsphere-incubating-4.0.0-sharding-proxy-bin/conf/config-shardi
ng.yaml
```

   ii. Configure the common configuration file based on the following example:

```
schemaName: sdb
dataSources:
  db0:
    url: jdbc:postgresql://pgm-bpxxxxx.pg.rds.aliyuncs.com:1433/db0
    username: r1
    password: PW123321!
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 65
  db1:
    url: jdbc:postgresql://pgm-bpxxxxx.pg.rds.aliyuncs.com:1433/db1
    username: r1
    password: PW123321!
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 65
  db2:
    url: jdbc:postgresql://pgm-bpxxxxx.pg.rds.aliyuncs.com:1433/db2
    username: r1
    password: PW123321!
```

```
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 65
  db3:
    url: jdbc:postgresql://pgm-bpxxxxx.pg.rds.aliyuncs.com:1433/db3
    username: r1
    password: PW123321!
    connectionTimeoutMilliseconds: 30000
    idleTimeoutMilliseconds: 60000
    maxLifetimeMilliseconds: 1800000
    maxPoolSize: 65
shardingRule:
  tables:
    t_order:
      actualDataNodes: db${0..3}.t_order${0..7}
      databaseStrategy:
        inline:
          shardingColumn: user_id
          algorithmExpression: db${user_id % 4}
      tableStrategy:
        inline:
          shardingColumn: order_id
          algorithmExpression: t_order${order_id % 8}
      keyGenerator:
        type: SNOWFLAKE
        column: order_id
    t_order_item:
      actualDataNodes: db${0..3}.t_order_item${0..7}
      databaseStrategy:
        inline:
          shardingColumn: user_id
          algorithmExpression: db${user_id % 4}
      tableStrategy:
        inline:
          shardingColumn: order_id
          algorithmExpression: t_order_item${order_id % 8}
      keyGenerator:
        type: SNOWFLAKE
        column: order_item_id
  bindingTables:
    - t_order,t_order_item
  defaultTableStrategy:
    none:
```

2. Start ShardingSphere and listen to port 8001.

```
cd ~/apache-shardingsphere-incubating-4.0.0-sharding-proxy-bin/bin/
./start.sh 8001
```

3. Connect to a database whose owner is user r1.

```
psql -h 127.0.0.1 -p 8001 -U r1 sdb
```

4. Create a table.

```
create table t_order(order_id int8 primary key, user_id int8, info text, c1 int, crt_time timestamp);
create table t_order_item(order_item_id int8 primary key, order_id int8, user_id int8, info text, c1 int, c2 int, c3 int, c4 int, c5 int, crt_time timestamp);
```

> **Note** When you create a table, the system automatically creates horizontal shards in the specific database based on the sharding strategy that you specify.

More

- If you want to know the SQL parsing and routing statements that are used in ShardingSphere, perform the following operations:
  - Run the **vi** command to open the common configuration file.

    ```
    vi /root/apache-shardingsphere-incubating-4.0.0-sharding-proxy-bin/conf/server.yaml
    ```

  - Configure the common configuration file based on the following example:

    ```
    authentication:
      users:
        r1:
          password: PW123321!
          authorizedSchemas: db0,db1,db2,db3
    props:
      executor.size: 16
      sql.show: true # Specifies to log the SQL statements that are parsed.
    ```

- If you want to test writes and queries, run the following commands:
  - Write examples

    ```
    insert into t_order (user_id, info, c1, crt_time) values (0,'a',1,now());
    insert into t_order (user_id, info, c1, crt_time) values (1,'b',2,now());
    insert into t_order (user_id, info, c1, crt_time) values (2,'c',3,now());
    insert into t_order (user_id, info, c1, crt_time) values (3,'c',4,now());
    ```

  - Query example 1

    ```
    select * from t_order;
    ```

    Sample result

    ```
          order_id      | user_id | info | c1 |          crt_time
    --------------------+---------+------+----+----------------------------
     433352561047633921 |       0 | a    |  1 | 2020-02-09 19:48:21.856555
     433352585668198400 |       1 | b    |  2 | 2020-02-09 19:48:27.726815
     433352610813050881 |       2 | c    |  3 | 2020-02-09 19:48:33.721754
     433352628370407424 |       3 | c    |  4 | 2020-02-09 19:48:37.907683
    (4 rows)
    ```

- Query example 2

```
sdb=> select * from t_order where user_id=1;
```

Sample result

```
     order_id      | user_id | info | c1 |          crt_time
-------------------+---------+------+----+----------------------------
 433352585668198400 |       1 | b    |  2 | 2020-02-09 19:48:27.726815
(1 row)
```

- If you want to view ShardingSphere logs, go to the following path:

```
/root/apache-shardingsphere-incubating-4.0.0-sharding-proxy-bin/logs/stdout.log
```

- If you want to use the pgbench stress testing, perform the following operations:
  i. Create a file named test sql and open the file.

```
vi test.sql
```

  ii. Configure the commands for the pgbench stress testing based on the following example:

```
\set user_id random(1,100000000)
\set order_id random(1,2000000000)
\set order_item_id random(1,2000000000)
insert into t_order (user_id, order_id, info, c1 , crt_time) values (:user_id, :order_i
d,random()::text, random()*1000, now()) on conflict (order_id) do update set info=exclu
ded.info,c1=excluded.c1,crt_time=excluded.crt_time;
insert into t_order_item (order_item_id, user_id, order_id, info, c1,c2,c3,c4,c5,crt_ti
me) values (:order_item_id, :user_id,:order_id,random()::text, random()*1000,random()*1
000,random()*1000,random()*1000,random()*1000, now()) on conflict(order_item_id) do not
hing;
pgbench -M simple -n -r -P 1 -f ./test.sql -c 24 -j 24  -h 127.0.0.1 -p 8001 -U r1 sdb
-T 120
progress: 1.0 s, 1100.9 tps, lat 21.266 ms stddev 6.349
progress: 2.0 s, 1253.0 tps, lat 18.779 ms stddev 7.913
progress: 3.0 s, 1219.0 tps, lat 20.083 ms stddev 13.212
```