ALIBABA CLOUD

Alibaba Cloud

Realtime Compute
Flink SQL Development Guide

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

The Realtime Compute for Apache Flink development platform provides multiple features for Realtime Compute Flink SQL jobs, including data storage management, job development, job debugging, job administration, monitoring and alerting, and job optimization.

Flink SQL Developer Guide consists of the following topics:

- Data storage

  You can manage upstream and downstream data storage systems, such as ApsaraDB RDS, DataHub, and Tablestore, for jobs on the Realtime Compute for Apache Flink development platform. After you register resources from these systems with Realtime Compute for Apache Flink, you can preview or sample their related data, or obtain the data definition language (DDL) statements that are automatically generated to reference these resources. For more information about data storage, see Overview.

  > **Note** For more information about how to add the IP addresses of Realtime Compute for Apache Flink to a whitelist of an upstream or downstream storage system, see Configure a whitelist for accessing storage resources.

- Job development

  This topic describes how to develop, publish, and start a Flink SQL job. For more information, see Develop a job, Publish a job, and Start a job.

- Job debugging

  This topic describes how to debug Flink SQL jobs. Online debugging and local debugging are supported. For more information, see Online debugging.

- Job administration

  This topic describes how to view the administration information of a Realtime Compute for Apache Flink job, such as the running information, curve charts, and failover. For more information, see Overview, Metrics, and Failover.

- Monitoring and alerting

  This topic describes how to create and activate alert rules. For more information, see Monitoring and alerting.

- Job optimization

  This topic describes how to optimize Flink SQL jobs, such as skills for optimizing Flink SQL code, automatic configuration optimization, performance optimization by auto scaling, and performance optimization by manual configuration. For more information, see Recommended Flink SQL practices,Performance optimization by using auto scaling, and Optimize performance by manual configuration.

- Flink SQL

  This topic describes the syntax of Flink SQL. For more information, see Overview.

2.Data storage

2.1. Overview

Alibaba Cloud Realtime Compute for Apache Flink provides a page to manage various storage systems, such as ApsaraDB RDS and Tablestore. Realtime Compute for Apache Flink provides you an end-to-end cloud-based management solution.

Limits

A Realtime Compute for Apache Flink cluster in exclusive mode can access only storage resources in the same virtual private cloud (VPC), region, and security group as the cluster.

Data storage in Realtime Compute for Apache Flink

In Realtime Compute for Apache Flink, data storage has the following meanings:

- It refers to the storage systems or database tables (hereinafter referred to as storage resources) at the upstream and downstream nodes of Realtime Compute for Apache Flink.
- It indicates how to use the data storage feature of Realtime Compute for Apache Flink. This feature is used to manage the upstream and downstream storage resources.

> **Note** Before you register storage resources with Realtime Compute for Apache Flink, you must authorize Realtime Compute for Apache Flink to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

Realtime Compute for Apache Flink allows you to reference both upstream and downstream storage resources by using plaintext AccessKey pairs or registering storage resources.

Use a plaintext AccessKey pair

To reference upstream and downstream storage resources by using a plaintext AccessKey pair, you must configure the `accessId` and `accessKey` parameters in the WITH clause of the related DDL statement. For more information, see Overview. This way, you can authorize an Alibaba Cloud account and its RAM users to access the resources of the current or another Alibaba Cloud account. If User A or a RAM user created within the Alibaba Cloud account of User A wants to use the storage resources of User B, User A can set the AccessKey pair of User B in the following DDL statement in plaintext mode:

```
CREATE TABLE in_stream(
  a varchar,
  b varchar,
  c timestamp
) with (
  type='datahub',
  endPoint='http://dh-cn-hangzhou.aliyuncs.com',
  project='<dataHubProjectName>',
  topic='<dataHubTopicName>',
  accessId='<accessIdOfUserB>',
  accessKey='<accessKeyOfUserB>'
);
```

Register a storage resource

Realtime Compute for Apache Flink allows you to manage and reference both upstream and downstream storage resources that have been registered with Realtime Compute for Apache Flink. After storage resources are registered, you can preview or sample the relevant data, or obtain the DDL statements that are automatically generated to reference the resources. This helps you manage your cloud storage resources in end-to-end mode.

> **Note** You can register only storage resources of the current Alibaba Cloud account. Therefore, User A or a RAM user created within the Alibaba Cloud account of User A can register only storage resources purchased by User A. If you want to use storage resources of another Alibaba Cloud account, you must use the plaintext AccessKey pair of the specified Alibaba Cloud account in the relevant DDL statement.

- Register storage resources

  To register upstream and downstream storage resources with Realtime Compute for Apache Flink before you reference them, perform the following steps:

  i. Log on to the .
  ii. In the top navigation bar, click **Development**.
  iii. In the left-side navigation pane of the **Development** page, click **Storage**.
  iv. In the upper-right corner of the **Storage** tab, click **+Registration and Connection**.
  v. In the **Register Data Store and Test Connection** dialog box, configure the parameters for storage resources.

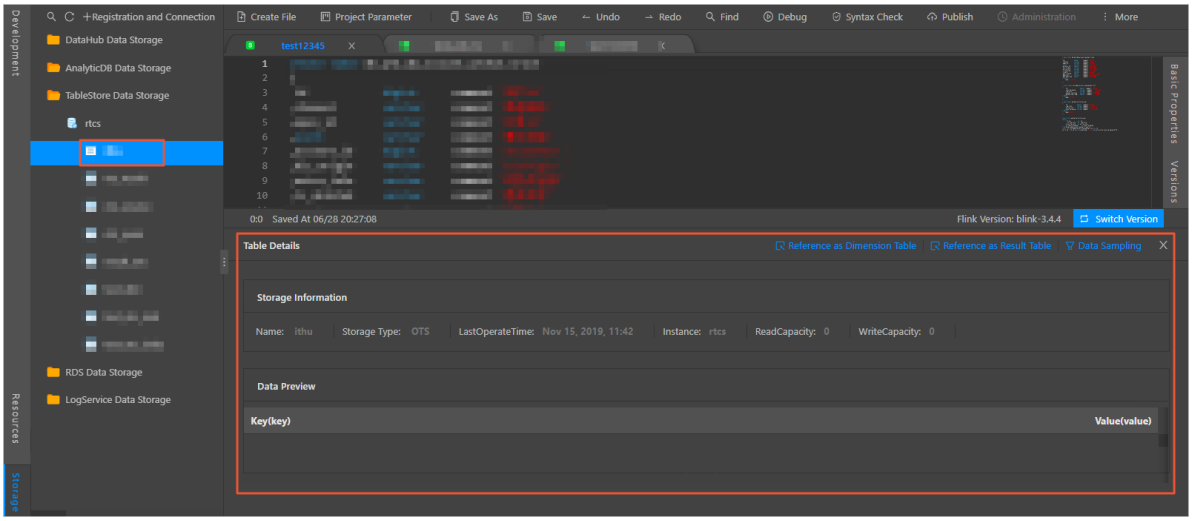
     Realtime Compute for Apache Flink allows you to register the following types of storage resources. For more information about how to register storage resources of a specific type, click the following links:

     - Register a Tablestore instance
     - Register an ApsaraDB for RDS instance
     - Register a Log Service project

- Preview data from a registered storage resource

  To preview data from a registered storage resource, perform the following steps:

  i. In the left-side navigation pane of the **Development** page, click **Storage**.
  ii. On the **Storage** tab, double-click the folder of a registered storage resource and its subfolder to find the table that you want to view, and double-click the name of the table.
  iii. In the **Table Details** pane, view data of the storage resource in the **Data Preview** section.



- Obtain the DDL statements that are automatically generated to reference a storage resource

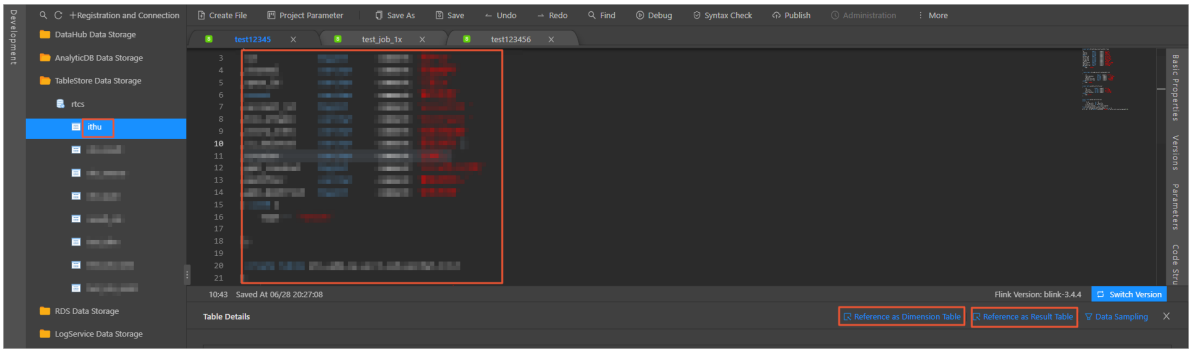
  To obtain the DDL statements that are automatically generated to reference a storage resource, perform the following steps:

  i. In the left-side navigation pane of the **Development** page, click **Storage**.

  ii. On the **Storage** tab, double-click the folder of a registered storage resource and its subfolder to find the table that you want to view, and double-click the name of the table.

  iii. In the **Table Details** pane, click **Reference as Source Table**, **Reference as Result Table**, or **Reference as Dimension Table**. Then, you can obtain the DDL statements that are automatically generated to reference the table.

  > **Note** The automatically generated DDL statements contain only the basic parameters in the WITH clause to ensure connectivity between Realtime Compute for Apache Flink and storage resources. You can add other parameters to the WITH clause in addition to the basic parameters.



- Test network connectivity by using the network detection feature

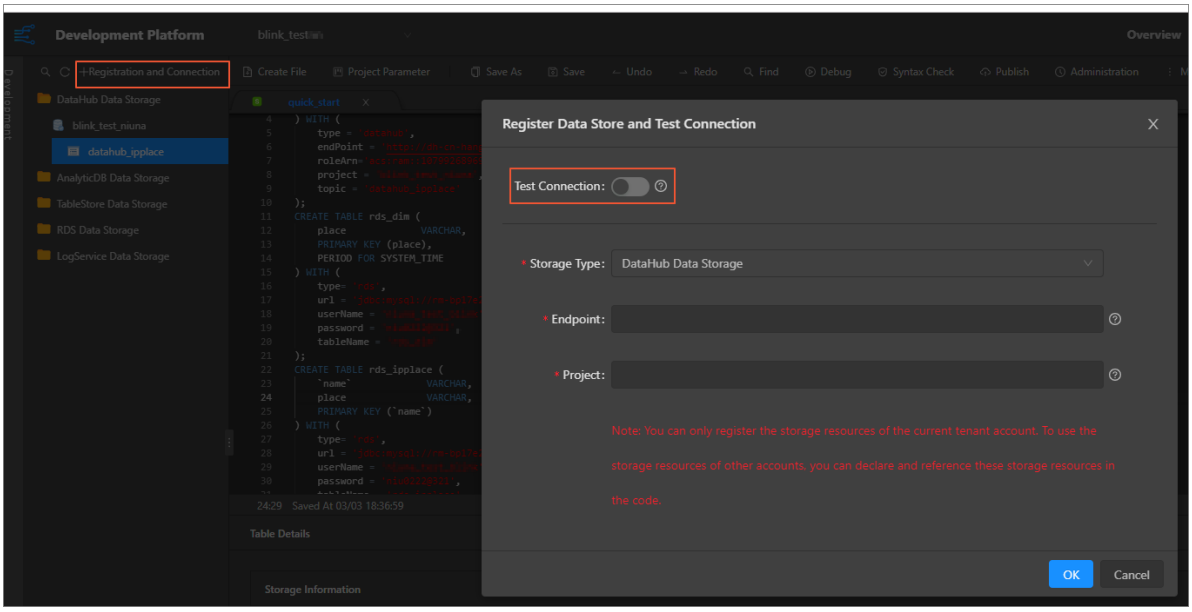
  > **Note** The network detection feature is not supported in the China (Hangzhou) region of Finance Cloud because Cloud Assistant is not installed in the region.

  Realtime Compute for Apache Flink provides the network detection feature for data storage. This feature allows you to test network connectivity between Realtime Compute for Apache Flink and storage resources. To enable the network detection feature, perform the following steps:

  i. In the left-side navigation pane of the **Development** page, click **Storage**.

ii. In the upper-right corner of the **Storage** tab, click **+Registration and Connection**.

iii. In the **Register Data Store and Test Connection** dialog box, turn on **Test Connection**.



2.2. Data storage resource registration

2.2.1. Register an AnalyticDB for MySQL instance

This topic describes the parameters that are required to register an AnalyticDB for MySQL instance.

> **Notice** This topic applies only to AnalyticDB for MySQL V2.0. Realtime Compute for Apache Flink does not allow you to register AnalyticDB for MySQL V3.0 to store result tables. To use the result tables of AnalyticDB for MySQL V3.0, you must create and reference the result tables in plaintext mode. For more information, see Create an AnalyticDB for MySQL V3.0 result table.

Register storage resources

> **Note** Before you use Realtime Compute for Apache Flink to register storage resources, you must grant Realtime Compute for Apache Flink the permission to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the Development page, click **Storage**.
4. In the upper-left corner of the Storage page, click **+Registration and Connection**.
5. In the **Register Data Store and Test Connection** dialog box, configure the storage parameters.
6. Click **OK**.

Storage parameters

Parameter	Description
URL	The URL of the AnalyticDB for MySQL database. You can use the URL to log on to the AnalyticDB for MySQL console. Enter the URL of the AnalyticDB for MySQL database based on the deployment mode of your Realtime Compute for Apache Flink: • If Realtime Compute for Apache Flink is deployed in shared mode, enter the classic network URL of the AnalyticDB for MySQL database. • If Realtime Compute for Apache Flink is deployed in exclusive mode, enter the VPC URL of the AnalyticDB for MySQL database. To view the URL, perform the following steps: 1. Log on to the AnalyticDB for MySQL console. 2. In the left-side navigation pane, click Clusters. On the Clusters page, click the ID of the instance to go to the **Cluster Information** page. 3. In the **Network Information** section, view the URL.
Database	The name of the AnalyticDB for MySQL database or the name of the AnalyticDB for MySQL instance.
AccessKey ID	The AccessKey ID of your Alibaba Cloud account.
AccessKey Secret	The AccessKey secret of your Alibaba Cloud account.

2.2.2. Register a Tablestore instance

This topic describes how to use Realtime Compute for Apache Flink to register a Tablestore instance.

Introduction to Tablestore

Tablestore is a NoSQL database service that is built based on Alibaba Cloud Apsara system. Tablestore allows you to store and access large amounts of structured data in real time. Tablestore provides the following benefits: low latency and simple computing. Therefore, Tablestore is suitable for storing dimension tables and result tables of Realtime Compute for Apache Flink.

Register storage resources

> **Note** Before you use Realtime Compute for Apache Flink to register storage resources, you must grant Realtime Compute for Apache Flink the permission to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the Development page, click **Storage**.
4. In the upper-left corner of the Storage page, click **+Registration and Connection**.

5. In the **Register Data Store and Test Connection** dialog box, configure the storage parameters.
6. Click **OK**.

Parameters

- Endpoint
  - The virtual private cloud (VPC) endpoint of the Tablestore instance that you want to register. You can view this endpoint in the Tablestore console. For more information about the VPC endpoint, see VPC endpoint.
  - To set Accessed By to **Any Network**, perform the following steps:
    a. Log on to the Tablestore console.
    b. In the left-side navigation pane, click All Instances. In the **Instance Name** column of the All Instances page, click the name of the Tablestore instance that you want to register.
    c. On the Instant Management page, click the **Network Management** tab. On this tab, click **Change** next to Accessed By.
    d. In the dialog box that appears, select **Any Network** from the Accessed By drop-down list.
    e. Click **OK**.
- Instance Name

  Enter the name of the Tablestore instance.

2.2.3. Register an ApsaraDB for RDS instance

This topic describes how to use Realtime Compute for Apache Flink to register an ApsaraDB for RDS instance.

Introduction to ApsaraDB for RDS

ApsaraDB for RDS is a stable, reliable, and scalable online database service. Based on Apsara Distributed File System and high-performance storage services, ApsaraDB for RDS supports a wide range of database engines, such as MySQL, SQL Server, PostgreSQL, and Postgres Plus Advanced Server (PPAS). ApsaraDB for RDS provides comprehensive solutions for database operations and maintenance (O&M), such as disaster recovery, data backup, data recovery and restoration, monitoring, and data migration.

> **Note**
> - If Realtime Compute for Apache Flink frequently writes data to a table or a resource file, a deadlock may occur. In scenarios that require highly frequent or highly concurrent writes, we recommend that you use Tablestore to store result tables. For more information, see Create a Tablestore result table.
> - Realtime Compute for Apache Flink does not allow you to register the ApsaraDB for RDS V8.0 data store in the console to store result tables. To use result tables of ApsaraDB for RDS V8.0, you must use the plaintext mode to create and reference the result tables. For more information, see Use a plaintext AccessKey pair.

Register storage resources

> **Note** Before you use Realtime Compute for Apache Flink to register storage resources, you must grant Realtime Compute for Apache Flink the permission to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the Development page, click **Storage**.
4. In the upper-left corner of the Storage page, click **+Registration and Connection**.
5. In the **Register Data Store and Test Connection** dialog box, configure the storage parameters.
6. Click **OK**.

Parameters

> **Note** When you register storage resources in an ApsaraDB for RDS instance, the IP addresses of Realtime Compute for Apache Flink are automatically added to the whitelist for accessing the ApsaraDB for RDS instance.

Parameter	Description
Storage Type	The type of storage resources that you want to register. From the Storage Type drop-down list, select **RDS Data Storage**.
Region	The region where the ApsaraDB for RDS instance resides.
Instance	The ID of the ApsaraDB for RDS instance that you want to register. > **Note** Enter the instance ID instead of the instance name.
DBName	The name of the ApsaraDB for RDS database. > **Note** Enter the database name instead of the instance name.
User Name	The username that is used to access the ApsaraDB for RDS database.
Password	The password that is used to access the ApsaraDB for RDS database.

2.2.4. Register a Log Service project

This topic describes how to use Realtime Compute for Apache Flink to register a Log Service project. This topic also provides answers to commonly asked questions about the registration process.

Introduction to Log Service

Log Service is an end-to-end logging service. Log Service allows you to collect, consume, ship, query, and analyze log data in a quick way. It improves the operations and maintenance (O&M) efficiency, and provides the capability to process large amounts of data. Log Service is used to store streaming data. Therefore, Realtime Compute for Apache Flink can use the streaming data that is stored in Log Service as input data.

Register storage resources

> **Note** Before you use Realtime Compute for Apache Flink to register storage resources, you must grant Realtime Compute for Apache Flink the permission to access these resources. For more information, see Assign a RAM role to an account that uses Realtime Compute for Apache Flink in exclusive mode.

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the Development page, click **Storage**.
4. In the upper-left corner of the Storage page, click **+Registration and Connection**.
5. In the **Register Data Store and Test Connection** dialog box, configure the storage parameters.
6. Click **OK**.

Parameters

- Endpoint

  The endpoint of the Log Service project that you want to register. The endpoint of a Log Service project varies based on the region of this project. For more information, see Endpoints.

  > **Note**
  > - The endpoint of a Log Service project must start with `http://` and cannot end with a forward slash (`/`). For example, the endpoint can be `http://cn-hangzhou-intranet.log.aliyuncs.com` .
  > - Realtime Compute for Apache Flink and Log Service are deployed in the internal network of Alibaba Cloud. We recommend that you enter the endpoint of the classic network or the virtual private cloud (VPC) for the project. We recommend that you do not enter the public endpoint. If Realtime Compute for Apache Flink accesses a Log Service project over the Internet by using the public endpoint, the system may consume the resources of Internet bandwidth. In this case, the system performance may be compromised.

- Project

  The name of the Log Service project that you want to register.

> **Note** In Realtime Compute for Apache Flink, you can register only the Log Service projects that are owned by the current Alibaba Cloud account. Assume that User A owns Project A of Log Service. If User B needs to use the storage resources of Project A in Realtime Compute for Apache Flink, this system does not allow User B to register Project A. If you need to use the Log Service project that is owned by another Alibaba Cloud account, you can use the plaintext mode to use the project. For more information, see Use a plaintext AccessKey pair.

FAQ

What do I do if I fail to register a storage resource in Realtime Compute for Apache Flink?

Realtime Compute for Apache Flink uses a storage software development kit (SDK) to access different data storage systems. The Storage tab on the Realtime Compute for Apache Flink development platform helps you manage data from different data storage systems. If you fail to register a storage resource in Realtime Compute for Apache Flink, troubleshoot the issue by performing the following steps:

- Check whether the Log Service project is created in your Alibaba Cloud account. Log on to the Log Service console and check whether you can access the project.
- Check whether the Log Service project is owned by your Alibaba Cloud account. You cannot register a project that is owned by another Alibaba Cloud account.
- Check whether the endpoint and the name of the Log Service project are valid. The endpoint of the Log Service project must start with `http://` and cannot end with a forward slash (`/`).
- Check whether the endpoint of the Log Service project is the classic network endpoint. Realtime Compute for Apache Flink does not support VPC endpoints.
- Check whether you have already registered the Log Service project. Realtime Compute for Apache Flink provides a registration check mechanism to prevent duplicate registrations.

Why does Realtime Compute for Apache Flink support only time-based data sampling?

Log Service stores streaming data and supports only time-based data sampling. You can specify only time parameters in the Log Service API. Therefore, Realtime Compute for Apache Flink supports only time-based data sampling.

2.3. Authorize Realtime Compute for Apache Flink to access a VPC

To allow Realtime Compute for Apache Flink in shared mode to access storage resources in a virtual private cloud (VPC), you must grant Realtime Compute for Apache Flink this permission. This topic describes how to grant Realtime Compute for Apache Flink the permission to access a VPC.

Background information

Realtime Compute for Apache Flink projects in shared mode are deployed in the classic network. To allow Realtime Compute for Apache Flink to access storage resources in a VPC, you must grant the system the permission to access the VPC. Realtime Compute for Apache Flink can access only ApsaraDB RDS instances in a VPC.

> **Note**
>
> - Realtime Compute for Apache Flink clusters in exclusive mode can access VPCs without authorization because these clusters reside in VPCs.
> - After you grant Realtime Compute for Apache Flink in shared mode the permission to access a VPC, performance issues may occur when the system accesses storage resources in the VPC. For example, the bandwidth may be restricted. To avoid such issues, we recommend that you do not access storage resources in a VPC from Realtime Compute for Apache Flink in shared mode.
> - As of December 24, 2019, Realtime Compute for Apache Flink in shared mode is no longer available. You cannot purchase projects in this mode. You can only scale out, scale in, or renew existing shared-mode projects. We recommend that you purchase the exclusive mode or the Flink cloud-native mode of Realtime Compute for Apache Flink based on your business requirements.

Procedure

1. Log on to the .
2. Move the pointer over the username in the upper-right corner.
3. In the list that appears, click **Project Management**.
4. In the left-side navigation pane, click **VPC Access Authorization**.
5. In the upper-right corner of the **VPC Access Authorization** page, click **Add Authorization**.
6. In the **Authorize StreamCompute VPC Access** dialog box, configure the parameters. The following table describes the parameters.

Parameter	Description
Name	The name of the VPC.
Region	The region in which the storage resource resides.
VPC ID	The ID of the VPC. To view the VPC ID of an ApsaraDB RDS instance, perform the following steps: i. Log on to the ApsaraDB RDS console. ii. In the left-side navigation pane, click **Instances**. iii. In the top navigation bar, select the region in which the ApsaraDB RDS instance resides. iv. On the Instances page, find the ApsaraDB RDS instance and click the instance ID in the Instance ID/Name column. v. In the left-side navigation pane, click **Database Connection**. vi. On the **Database Connection** page, view the VPC ID in **Network Type**. For example, the VPC ID of the ApsaraDB RDS instance is `vpc-bp1lysht98wrvl9n3****`.

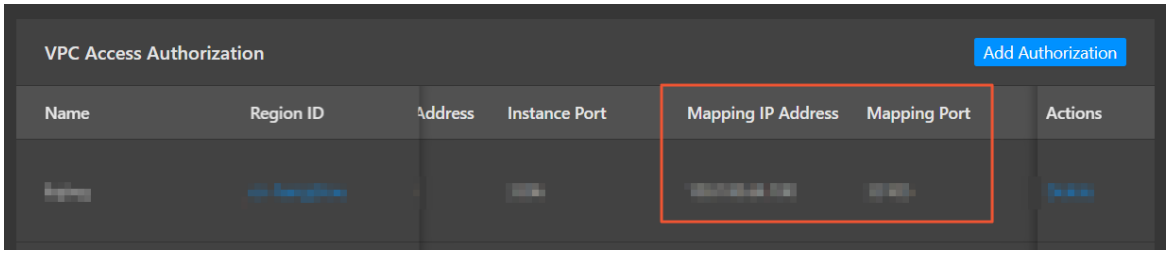
Parameter	Description
Instance ID	The ID of the ApsaraDB RDS instance in the VPC. To view the ID of an ApsaraDB for RDS instance, perform the following steps: i. Log on to the ApsaraDB RDS console. ii. In the left-side navigation pane, click **Instances**. iii. In the top navigation bar, select the region where the ApsaraDB RDS instance resides. iv. On the Instances page, find the ApsaraDB RDS instance and click the instance ID in the **Instance ID/Name** column. v. On the **Basic Information** page, view the ID of the ApsaraDB RDS instance.
Instance Port	The port number of the ApsaraDB RDS instance in the VPC. For more information about how to view the port number of an ApsaraDB RDS instance, see View and change the internal and public endpoints and port numbers of an ApsaraDB RDS for MySQL instance.

FAQ

Q: How do I configure the url parameter when I use data definition language (DDL) statements to reference a storage resource in a VPC?

A: When you use DDL statements to reference storage resources in a VPC, you can configure the url parameter in the WITH clause based on the **Mapping IP Address** and **Mapping Port** parameters on the **VPC Access Authorization** page. For example, you can configure url='jdbc:mysql://<mappingIP>:<mappingPort>/<databaseName>'. To obtain the values of the **Mapping IP Address** and **Mapping Port** parameters, perform the following steps:

1. Log on to the .
2. Move the pointer over the username in the upper-right corner.
3. In the list that appears, click **Project Management**.
4. In the left-side navigation pane, click **VPC Access Authorization**.
5. On the **VPC Access Authorization** page, view the values of the **Mapping IP Address** and **Mapping Port** parameters.



2.4. Configure a whitelist for accessing storage resources

By default, a newly created database instance does not allow access from IP addresses that are not included in its whitelists. To allow Realtime Compute for Apache Flink to access the database instance, you must add the IP addresses of Realtime Compute for Apache Flink to a whitelist of the database instance. This topic describes how to add the IP addresses of Realtime Compute for Apache Flink to a whitelist of an ApsaraDB for RDS instance.

IP addresses to be added to the whitelist

To access storage resources from a Realtime Compute for Apache Flink cluster in exclusive mode, you only need to add the IP addresses of the ENI to the whitelist. To view the IP addresses of the ENI, perform the following steps:

1. Log on to the .
2. Move the pointer over the username in the upper-right corner.
3. In the drop-down list, click **Project Management**.
4. In the left-side navigation pane, click **Clusters**.
5. On the **Clusters** page, click the name of the target cluster.
6. In the cluster information dialog box, view the **ENI** of the cluster.

Configure a whitelist for an ApsaraDB for RDS instance

When you reference an ApsaraDB for RDS database in Realtime Compute for Apache Flink, Realtime Compute for Apache Flink needs to frequently read and write data in the ApsaraDB for RDS database. In this case, you must add the IP addresses of Realtime Compute for Apache Flink to a whitelist of the ApsaraDB for RDS instance. For more information, see Use a database client or the CLI to connect to an ApsaraDB RDS for MySQL instance.

3.Job development

3.1. Develop a job

This topic describes how to create a Realtime Compute for Apache Flink job. This topic also describes the features provided on the Development page, such as syntax check, Flink SQL code assistance, and management of Flink SQL code versions.

Context

Note

Write the Flink SQL code of a job

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. At the top of the **Development** page, click **Create File**
4. In the **Create File** dialog box, specify the parameters. The following table describes these parameters.

Parameter	Description
File Name	The name of the file that contains the Fink SQL code. Note The file name must be unique in the existing project.
File Type	◦ For Realtime Compute for Apache Flink in shared mode, the valid value is only **FLINK_STREAM/SQL**. ◦ For Realtime Compute for Apache Flink in exclusive mode, valid values are **FLINK_STREAM/DATASTREAM** and **FLINK_STREAM/SQL**.
Storage Path	The folder of the file that contains the Fink SQL code. On the right side of the existing folder, you can also click the  icon to create a subfolder.

5. Click **OK**.
6. On the page that appears, write the Flink SQL code for the job.

> **Note**
>
> - On the right side of the **Development** page, you can click **Code Structure** to check the Flink SQL code structure.
> - On the left side of the **Development** page, we recommend you click **Storage** to manage upstream and downstream storage resources. For more information, see Overview.

Specify job parameters

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. In the left-side navigation pane of the **Development** page, click the job name.
4. On the right side of the **Development** page for the job, click **Parameters**.
5. Specify the parameters required for the job.

   For more information about job parameters, see Job parameters.

Specify project parameters

The job parameters are valid for a single job. The project parameters are valid for all the jobs in the project. After the project parameters are configured, the following two operations are performed:

- Replacing variables: After you click Start, Debug, or Syntax Check, the system replaces the variables in SQL jobs or in the code of the jobs that are created by using the Flink DataStream API.
- Distributing parameters: The project-level system parameters are merged with job parameters and startup parameters. The startup parameters can be configured for only batch jobs. The following parameters are sorted in descending order based on their priorities: startup parameters > job parameters > project-level system parameters. The merged parameters are used as final parameters and are distributed to the Blink job. For example, if job parameters and project parameters conflict, the job parameters prevail.

1. Log on to the .
2. In the top navigation bar, move the pointer over your profile picture. In the list that appears, click **Project Management**.
3. In the **Project Name** column of the **Projects** page, click the name of the project for which you want to configure parameters.
4. In the top navigation bar, click **Development**.
5. On the left side of the **Development** page, click the job for which you want to configure the parameters.
6. Enable project parameter configuration.

   By default, `disable.project.config=false` is specified. This indicates that you cannot configure project parameters. You can enable project parameter configuration by using the following methods:

   - SQL jobs: On the right side of the Development page, click **Parameters** and specify the following setting: `enable.project.config=true`.
   - Flink DataStream job: In the job code, specify `enable.project.config=true parameter`.

7. At the top of the Development page, click **Project Parameter**.
8. Configure the project parameters

   Project parameters support only two job types: SQL jobs and Flink DataStream jobs. When you configure project system parameters, you must add the job type to the beginning of the project parameters. For example, you can use `sql.name=LiLei` or `datastream.name=HanMeimei`.

Enable syntax check

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. On the left side of the **Development** page, click the job for which you want to check syntax.
4. At the top of the **Development** page, click **Syntax Check**.

> Note
> - When you save a Flink SQL job, the system automatically checks the syntax of this job.
> - **Syntax Check** takes effect for only Flink SQL statements that have complete logic. Otherwise, **Syntax Check** does not take effect.

Flink SQL code assistance

- Syntax check

  After you modify the Flink SQL code, the system automatically saves the code and checks the syntax. If a syntax error is detected, the system displays the cause of the error, the row and column where the error occurred on the **Development** page.

- Intelligent code completion

  When you enter Flink SQL statements on the **Development** page, the system automatically performs intelligent code completion, including keywords, built-in functions, tables, and fields.

- Syntax highlighting

  The system highlights keywords in Flink SQL statements and displays different structures in different colors.

Management of Flink SQL code versions

Realtime Compute for Apache Flink allows you to manage Flink SQL code versions. A new code version is generated each time you publish a job. You can use code versions to track versions, modify the code, and roll the code back to an earlier version.

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. On the left side of the **Development** page, click the job for which you want to manage the code version.
4. On the right side of the **Development** page, click **Versions**.
5. In the Versions pane, find the code version of the job and choose **Actions > More**.
6. Select one of the following options from the drop-down list:
   - **Compare**: checks the difference between the current version and an earlier version.

- **Rollback**: rolls the code back to an earlier version.
- **Delete**: deletes a code version. By default, you can reserve a maximum of 20 Flink SQL code versions in Realtime Compute for Apache Flink. If the number of code versions is less than 20, you can publish a job. If the number of code versions is 20, the system does not allow you to publish a job and prompts you to delete earlier versions.

  > **Note** You can publish a job only if the number of versions is less than 20.

- **Locked**: locks the current version.

  > **Note** You can submit a new version only after you unlock the current version.

3.2. Publish a job

After you develop and debug a job, and pass the syntax check, you can publish the job to the production environment.

Procedure

1. Configure resources

   Specify the resource configuration mode based on your requirements. We recommend that you use the default configuration if you publish the job for the first time.

   > **Note** Realtime Compute for Apache Flink supports manual resource configuration. For more information, see Optimize performance by manual configuration.

2. Check data

   Check parameter settings and click **Next**.

3. Publish the job

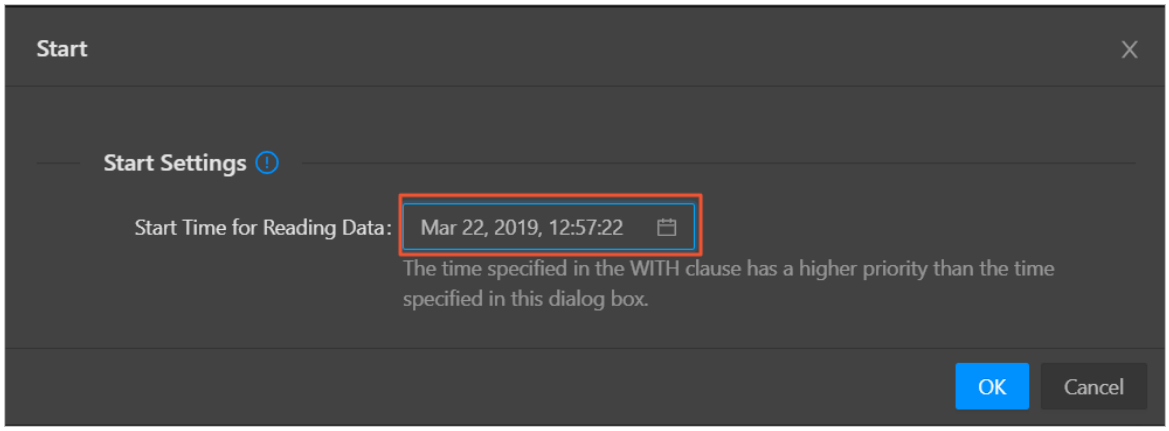
   Click **Publish**.

3.3. Start a job

After you develop and publish a job, you can start the job on the Administration page.

Procedure

1. Log on to the .
2. In the top navigation bar, click **Administration**.
3. In the **Actions** column of the **Administration** page, find the job that you want to start and click **Start**.
4. In the **Start** dialog box, configure the **Start Time for Reading Data** parameter.



5. Click **OK**. The job is started.

   Start Time for Reading Data specifies the time when Realtime for Apache Flink starts to read data from the source table. The time indicates the time when data is generated.

   - If you specify the current time, Realtime Compute for Apache Flink reads data that is generated from the current time.
   - If you select a previous time point, Realtime Compute for Apache Flink reads data that is generated from this time point. This is used to trace historical data.

   > **Note** After the job is started, you can check the running information on the Overview tab. For more information, see Overview.

3.4. Suspend a job

After you modify the resource configuration of a job, you must suspend and resume the job to make the changes take effect. This topic describes how to suspend a job.

Context

> **Notice**
>
> - You can only **suspend** a job that is in the **Running** state.
> - **Suspending** a job does not clear its task status. For example, if the job you **suspend** is running a COUNT operation, the COUNT operation continues from the last successful checkpoint after you **resume** the job.
> - The Suspend (checkpoint) operation is supported in Realtime Compute V3.5.0 and later. If your Realtime Compute is earlier than V3.5.0, the following error message is displayed when you try to perform this operation: **An error occurred. System error: The BLINK version is abnormal. Error reason: blink version >= blink-3.5 is required, instance blink-3.4.4.**

Procedure

1. Log on to the Realtime Compute development platform.
2. In the top navigation bar, click **Administration**.
3. On the **Administration** page, find the target job, and click **Suspend** in the **Actions** column.

> **Note** The **Suspend (checkpoint)** operation in **More** suspends the job and triggers a checkpoint event. Therefore, the **Suspend (checkpoint)** operation takes longer time to suspend a job than the **Suspend** operation.

3.5. Terminate a job

After you modify the SQL logic, change the job version, add parameters to the WITH clause, or add job parameters for a job, you must terminate and then start the job to make the changes take effect. This topic describes how to terminate a job.

> **Notice**
>
> - You can only **terminate** a job that is in the **Running** or **Starting** state.
> - If you **terminate** a job, its task status is cleared. For example, if the job you **terminate** is running a COUNT operation, the COUNT operation starts from 0 after you **start** the job.
> - The **Terminate (checkpoint)** operation is supported in Realtime Compute for Apache Flink V3.5.0 and later. If your Realtime Compute for Apache Flink is earlier than V3.5.0, the following error message is displayed when you try to perform this operation: **An error occurred. System error: The BLINK version is abnormal. Error reason: blink version >= blink-3.5 is required, instance blink-3.4.4**.

To terminate a job, perform the following steps:

1. Log on to the .
2. In the top navigation bar, click **Administration**.
3. On the **Administration** page, find the job that you want to terminate, and click **Terminate** in the **Actions** column.

> **Note** The **Terminate (checkpoint)** operation under **More** is different from the **Terminate** operation. The system triggers a checkpoint when you perform the **Terminate (checkpoint)** operation to terminate a job. Therefore, the time consumed to terminate a job by performing the **Terminate (checkpoint)** operation is longer than that by performing the **Terminate** operation. The job status is cleared after the job is terminated. The Terminate (checkpoint) operation has other functions in some scenarios. For example, if the upstream storage system is Message Queue for Apache Kafka, the system submits an offset each time it triggers a checkpoint. This ensures that the number of offsets submitted to the Kafka server is consistent with the amount of data consumed.

4.Job debugging

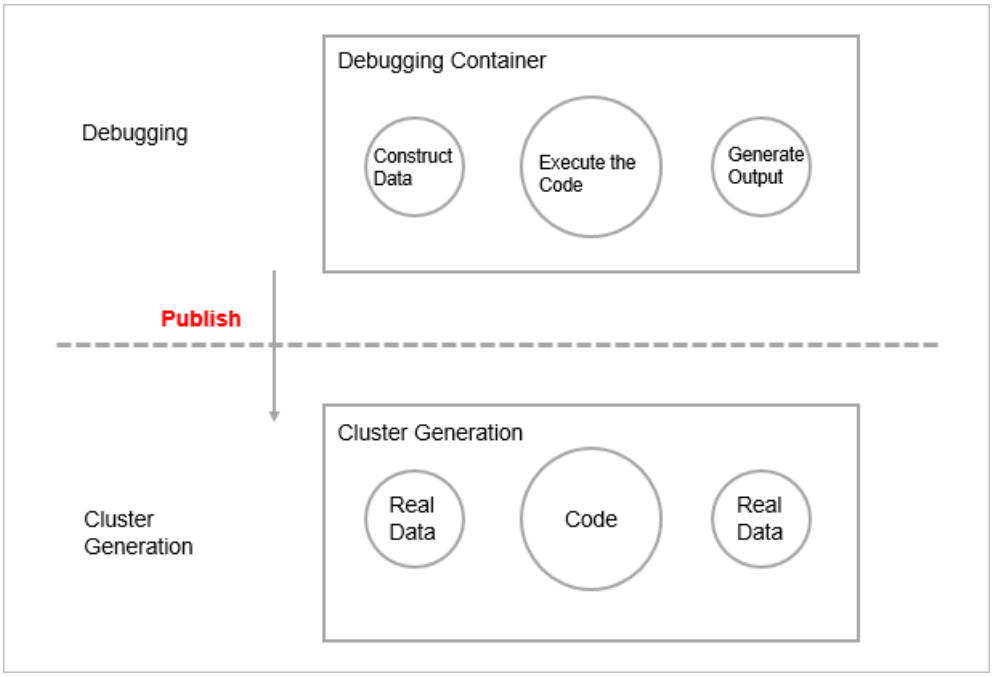
4.1. Perform local debugging

The Realtime Compute for Apache Flink development platform provides you with a local debugging environment. You can upload custom data, simulate job running, and check output in the local debugging environment to make sure that your business logic is valid.

Characteristics

The local debugging environment is completely isolated from the production environment. In the local debugging environment, all Flink SQL jobs run in an independent debugging container, and the debugging results are displayed on pages in the debugging environment. Local debugging does not affect online production streams, Realtime Compute for Apache Flink jobs, or data storage systems.

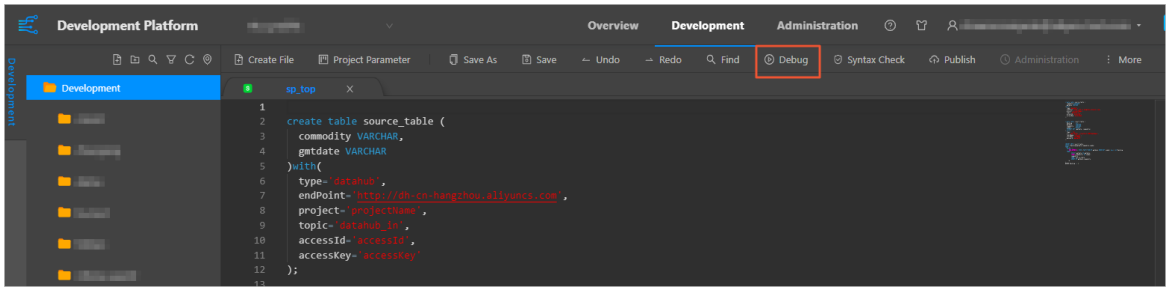
> **Note** In local debugging mode, Realtime Compute for Apache Flink cannot detect running failures caused by incompatible data formats in data storage resources. For example, Realtime Compute for Apache Flink cannot detect whether the length of output data exceeds the maximum length specified when you create an ApsaraDB RDS table.



Procedure

> **Note** Before you debug a job, make sure that you have developed the job. For more information, see Develop a job.

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. On the top of the job edit section, click **Debug**.



4. In the **Debug File** dialog box, enter the test data. You can obtain test data by using one of the following methods:
   - Upload local data
     a. In the data preview section, click **Download Template**.
     b. Edit the debugging data based on the template.

     > **Note** The default delimiter for debugging data is a comma (,). For more information about how to define a delimiter, see Delimiter of the debugging data.

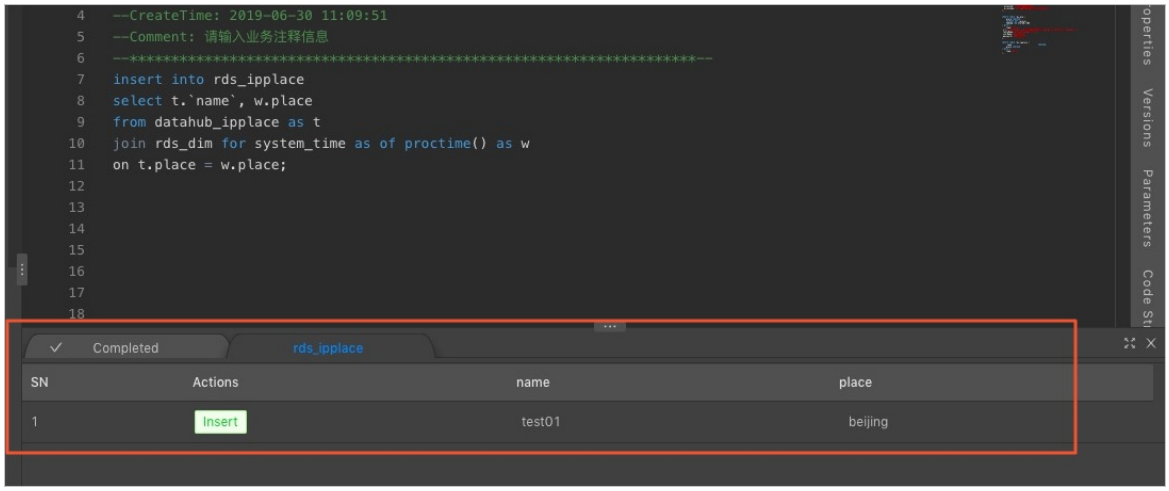
     c. In the **Upload File** section, click **Upload** to upload the debugging data.
   - Sample online data

     > **Note** Before you use the **Sequential Online Data Sampling** feature, make sure that the data source contains data during the sampling.

     a. In the data preview section, click **Random Online Data Sampling** or **Sequential Online Data Sampling**.
     b. Enter the sample configuration information.
     c. Click **OK**.

   > **Note** After the data is uploaded, you can view it in the data preview section.

5. Click **OK**.
6. In the lower part of the job edit section, view the debugging results.



Delimiter of the debugging data

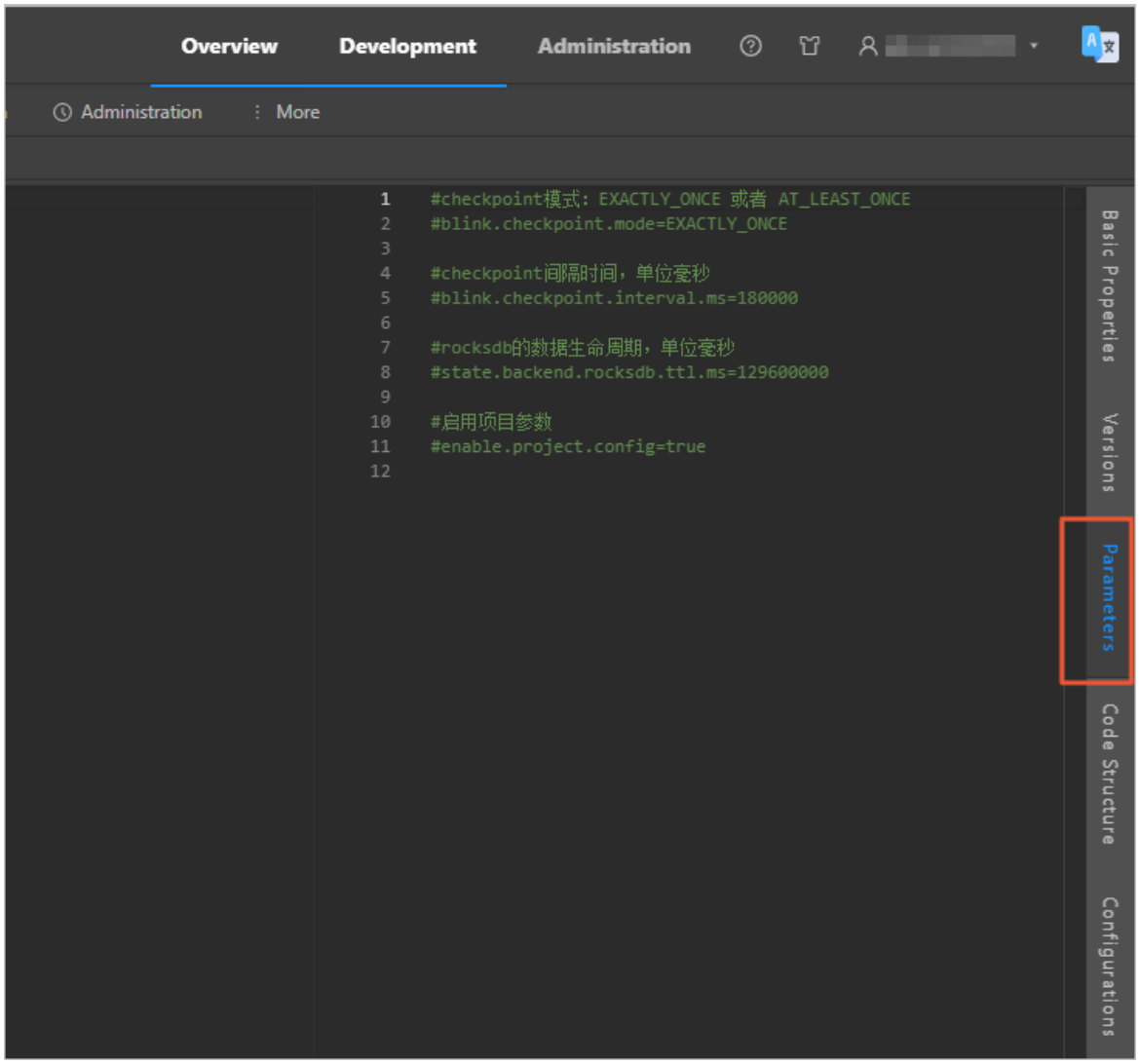
By default, the debugging data uses a comma (,) as the delimiter. If the input data, such as a JSON file, already contains commas (,), you must define another delimiter for the debugging data, such as a vertical bar (|).

> **Note** Realtime Compute for Apache Flink supports only a single character as a delimiter, such as a vertical bar (|). You cannot use a string as a delimiter, such as `aaa`.

To define a delimiter for debugging data, perform the following steps:

> **Note** Before you configure a delimiter, make sure that you have developed a job. For more information, see Develop a job.

1. On the right side of the job edit section, click **Parameters**.



2. In the parameter edit section of the job, enter the configuration of the delimiter. The following code shows how to configure a vertical bar (|) as the delimiter:

```
debug.input.delimiter = |
```

UDX logs

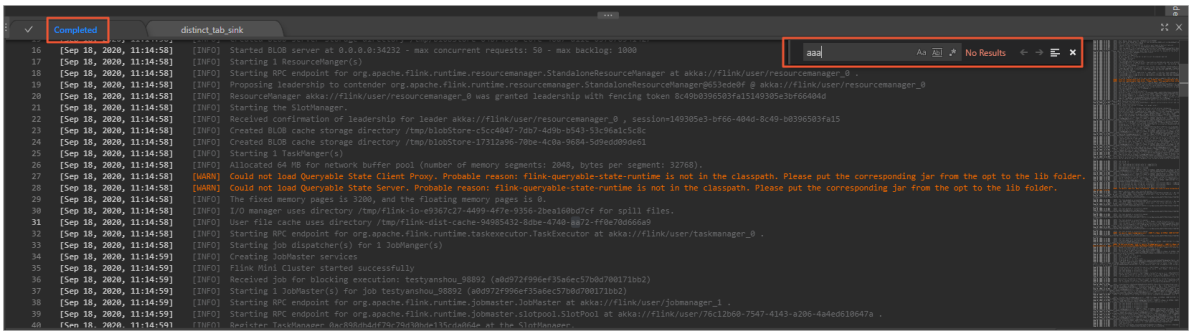
- Display user-defined extension (UDX) logs generated during local debugging.

  In Java, use the following method to convert the log format so the logs can be parsed by Realtime Compute for Apache Flink and display the UDX logs generated during local debugging:

```
public static void debugMsgOutput(String msg) {
    System.out.println(
      String.format("{\"type\":\"log\",\"level\": \"INFO\", \"time\": \"%s\", \"message\"
: \"%s\", \"throwable\": \"null\"}\n",new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").format(
new Date()), msg));
}
```

- View UDX logs.

  After the debugging is completed, you can view the UDX logs on the **Completed** tab at the lower part of the job edit section.



> **Note** You can press Ctrl+F to search for the logs.

4.2. Online debugging

The Realtime Compute for Apache Flink development platform provides an online debugging environment for you to debug your Realtime Compute for Apache Flink jobs. Compared with local debugging, online debugging consumes more compute units but validates business logic more accurately.

Online debugging uses real data storage resources to reduce the output differences between debugging and production. This helps you identify issues in the debugging phase.

Procedure

1. Develop a job. For more information, see Develop a job.
2. Modify the `type` parameter in the data definition language (DDL) statements of data storage resources.
   - Source table: `type = 'random'`
   - Result table: `type = 'print'`
3. Publish the job. For more information, see Publish a job.
4. Start the job. For more information, see Start a job.

Connectors

The Realtime Compute for Apache Flink development platform provides the following two types of connectors for online debugging:

- Source table `random`: periodically generates random data of a specific type.
- Result table `print`: generates computing results.

Parameters in a connector table

- random table

Parameter	Description
type	Required. The type of the connector. The value can only be random.
interval	Optional. The time interval at which data is generated. Unit: milliseconds. Default value: 500.

- print table

Parameter	Description
type	Required. The type of the connector. The value can only be print.
ignoreWrite	Optional. Specifies whether to ignore write operations. Default value: false. Valid values: ◦ false: Data is written to the result table, and logs are generated. ◦ true: Data is not written to the result table. The result table is empty, and no logs are generated.

Examples

- Test code

```
CREATE TABLE random_source (
  instr               VARCHAR
) WITH (
  type = 'random'
);
CREATE TABLE print_sink(
  instr VARCHAR,
  substr VARCHAR,
  num INT
)with(
  type = 'print'
);
INSERT INTO print_sink
SELECT
  instr,
  SUBSTRING(instr,0,5) AS substr,
  CHAR_LENGTH(instr) AS num
FROM random_source
```

- Test results

Query online debugging results

> **Note** Before you query the results of online debugging, make sure that you have published and started the job. For more information, see Publish a job and Start a job.

To query the results of online debugging, follow these steps:

1. Log on to the .
2. In the top navigation bar, click **Administration** to go to the **Administration** page.
3. In the **Job Name** column, click the name of the job to go to the **Job Administration** page.
4. In the **Vertex Topology** section, click the required result table node.
5. On the top of the Execution Vertex page, navigate to **Subtask List > View Logs** to go to the **View Logs** page.
6. View the logs.
   - Output of the print result table

     Click **View Logs** in the Actions column of taskmanager.out.
   - Output of UDX logs

     If you use user defined extensions (UDXs), you can view the logs by using the following methods. For more information about how to use UDXs, see Overview.
     - system.out or system.err method

       Click **View Logs** in the Actions column of `taskmanager.out` or `taskmanager.err`.

- SLF4J Logger method

  Click **View Logs** in the Actions column of `taskmanager.log`.

5.Job administration

5.1. Go to the Job Administration page

On the **Job Administration** page, you can view information about a job, such as the **running information**, **curve charts**, **failover information**, and **properties and parameters**. This topic describes how to go to the Job Administration page.

1. Log on to the Realtime Compute Console.
2. In the top navigation bar, click **Administration**.
3. In the **Jobs** section, click the target job name under the **Job Name** field.

5.2. Overview

The Overview page displays the real-time running information about a job. You can analyze and determine whether a job is healthy and meets your expectations based on the job status.

Go to the Overview page

1. Go to the **Administration** page in Realtime Compute for Apache Flink.
2. At the top of the **Job Administration** page, click **Overview**.

Task status

The **Task Status** section displays the number of tasks in each state. A task can be in one of the following states:

- **Created**
- **Running**
- **Failed**
- **Completed**
- **Scheduling**
- **Canceling**
- **Canceled**

Job instantaneous values

Below the **Task Status** section, you can view the instantaneous values of a job.

Parameter	Description
Input TPS	The number of blocks that are read from the source table per second. For Log Service, you can package multiple data records into one log group for reading. In this case, the value indicates the number of log groups that are read from the source table per second.
Input RPS	The number of data records that are read from the source table per second.
Output RPS	The number of data records that are written to the result table per second.

Parameter	Description
Input BPS	The number of blocks that are read from the source table per second. Unit: bytes/s.
Consumed CUs	The current compute units (CUs) used by the job.
Start Time	The start time of the job.
Runtime	The duration for which the job has been running.

Vertex topology

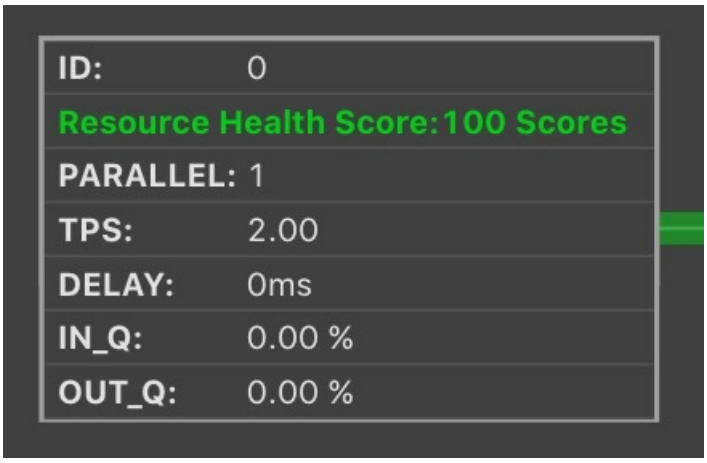
The **Vertex Topology** section displays the execution graph of the underlying computing logic of Realtime Compute for Apache Flink. Each component represents a task. Each data stream flows from one or more data source tables to one or more data result tables. The data flow resembles an arbitrary directed acyclic graph (DAG). To execute a job more efficiently, the underlying logic of Realtime Compute for Apache Flink chains subtasks to form an operator and chains operators to form a task. Each task is executed in a thread. This feature reduces thread switching, message serialization or deserialization, data swapping in the buffer, and data latency. However, this feature increases the overall throughput. An operator indicates the computational logic. A task is a collection of multiple operators.

- Display mode

  By default, the **Vertex Topology** section displays the topology. In the upper-right corner of the Vertex Topology section, you can click **List Mode** to change the display mode.

- Task status information

- Task parameters in view mode



In view mode, the information about the task is displayed in the related box. The following table describes the parameters.

Parameter	Description
Resource Health Score	The **resource health score** is obtained based on the specified check mechanism. The value indicates the job performance. If the **resource health score** is less than 60, data is stacked up for the task. This results in poor data processing performance. **Note** If the data processing performance is poor, we recommend that you optimize the performance by manual configuration. For more information, see Optimize performance by manual configuration.
PARALLEL	The parallelism for the task.
TPS	The number of blocks that are read from the source table per second.
DELAY	The processing delay of the task.
IN_Q	The percentage of the input queues for the task.
OUT_Q	The percentage of the output queues for the task.

- Task parameters in list mode

  At the bottom of the Vertex Topology section, you can view the information about the task in list mode. The following table describes the parameters of the task.

Parameter	Description
Name	The name of the task.
Status	The status of the task.
In Queue	The percentage of the input queues for the task.
Out Queue	The percentage of the output queues for the task.
Delay(ms)	The processing delay of the task.
TPS	The amount of data that is read from input nodes per second.
Bytes Received	The amount of data that is received by the task.
Records Received	The number of data records that are received by the task.
Bytes Sent	The amount of data that is sent from the task.
Records Sent	The number of data records that are sent from the task.
Task	The status of each parallelism for the task.

- Task thread information

  Click the task node. On the **SubTasks** tab, view the thread list of the task.

Vertex information

> **Note** The following features are supported only in Realtime Compute for Apache Flink V3.0.0 and later.

- Display Vertex operator information

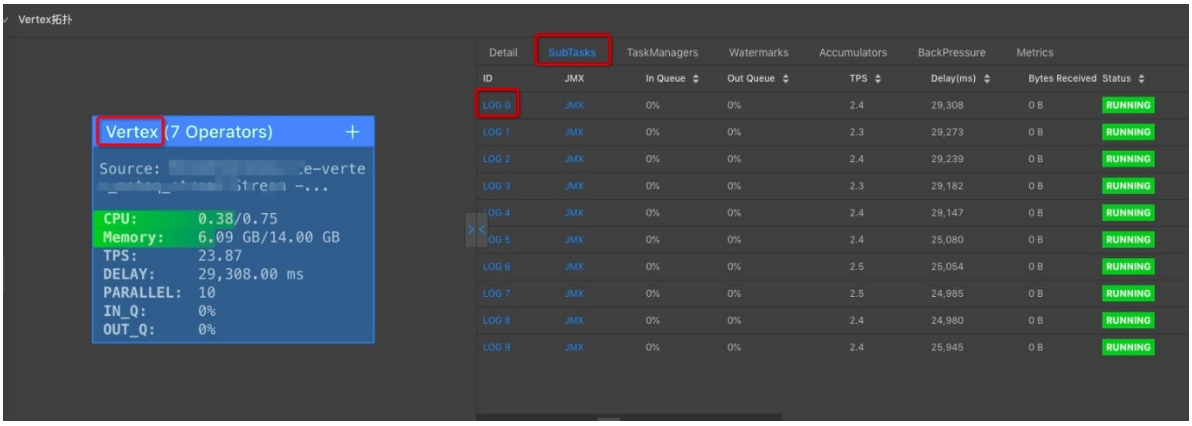
  In the Vertex Topology section, click the plus sign (+) in the upper-right corner of a Vertex box to display the Vertex operator information.

- Display Vertex details

  In the upper-right corner of the Vertex Topology section, click **Expand All** to display Vertex details.

- Display the Vertex details page

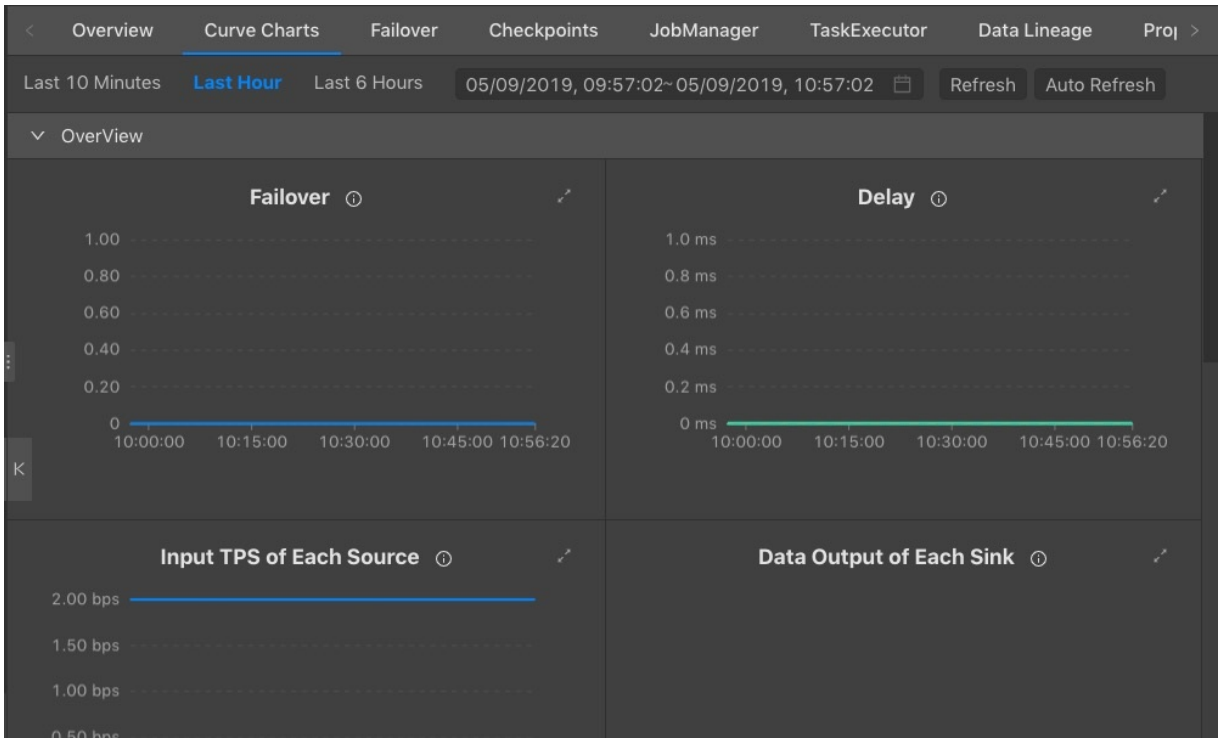
In the Vertex Topology section, click the Vertex border or the name in the Vertex list to display the Vertex details page on the right. In the Vertex details page, click the **SubTasks** tab. On the page that appears, click ID to go to the **TaskManager** page of the related log. For example, below the ID column, click **LOG 0**.



5.3. Metrics

Realtime Compute for Apache Flink shows core metrics of your job on the Curve Charts tab to help you diagnose the status of the job.

The following figure shows the curve chart of a metric.



> **Note**
>
> - The metrics are displayed only when a Realtime Compute for Apache Flink job is in the **running** state. If the job is in the **suspended** or **terminated** state, the metrics of this job are not displayed.
> - The metrics are collected and displayed on the Curve Charts tab after the job is running for more than one minute. This causes the latency in the data that is displayed in the curve charts.

Go to the Curve Charts tab

1. Go to the **Job Administration** page in the Realtime Compute for Apache Flink console.
2. In the upper part of the **Job Administration** page, click the **Curve Charts** tab.

Overview

- **Failover**

  The failover curve chart displays the frequency of failovers that are caused by errors or exceptions for the current job. To calculate the failover rate, divide the total number of failovers that occurred within the minute that precedes the current failover time by 60. For example, if a failover occurred once within the last minute, the failover rate is 0.01667. The failover rate is calculated by using the following formula: 1/60 = 0.01667.

- **Delay**

  To help you obtain the full-link timeliness and job performance, Realtime Compute for Apache Flink provides the following latency metrics:

  - **Processing Delay**: Processing delay = Current system time - Event time at which the system processes the last data record. If no more data enters upstream storage systems, the processing delay gradually increases as the system time continues to move forward.
  - **Data Pending Time**: Data pending time = Time when data enters Realtime Compute for Apache Flink - Event time. Even if no more data enters upstream storage systems, the queued time does not increase. The queued time is used to assess whether the Realtime Compute for Apache Flink job has backpressure.
  - **Data Arrival Interval**: Data arrival interval = Processing delay - Data pending time. If the Realtime Compute for Apache Flink job has no backpressure, the queued time is short and stable. In this case, this metric reflects the degree of data sparsity between the data sources. If the Realtime Compute for Apache Flink job has backpressure, the queued time is long or unstable. In this case, this metric cannot be used for reference.

> **Note**
>
> - Realtime Compute for Apache Flink uses a distributed computing framework. The preceding three latency metrics obtain values of each shard or partition of the data source. Then, the metrics report the maximum values among all the shards or all the partitions to the development platform of Realtime Compute for Apache Flink. Therefore, the aggregated data arrival interval that is displayed on the development platform is different from the interval that is obtained by using the following formula: Data arrival interval = Processing delay - Data pending time.
> - If no more data enters a shard or a partition of the data source, the processing delay gradually increases.

- **Input TPS of Each Source**

  This chart displays statistics on all streaming data input of a Realtime Compute for Apache Flink job. The chart records the number of blocks that are read from the source table per second. This helps you obtain the transactions per second (TPS) of a data storage system. Different from TPS, the records per second (RPS) metric indicates the number of records read from the source table per second. These records are resolved from the blocks. For example, if Log Service reads five log groups per second, the value of TPS is 5. If eight log records are resolved from each log group, a total of 40 log records are resolved. In this case, the value of RPS is 40.

- **Data Output of Each Sink**

  This chart displays statistics on all output data of a Realtime Compute for Apache Flink job. This helps you obtain the RPS of a data storage system. In most cases, if no data output is detected during system operations and maintenance (O&M), you must check the input of the upstream storage system and the output of the downstream storage system.

- **Input RPS of Each Source**

  This chart displays statistics on all input streaming data of a Realtime Compute for Apache Flink job. This helps you obtain the RPS of a data storage system. If no data output is detected during system O&M, you must check the RPS to determine whether the input data from the upstream storage system is normal.

- **Input BPS of Each Source**

  This chart displays statistics on all input streaming data of a Realtime Compute for Apache Flink job. This chart records the traffic that is used to read the input source table per second. This helps you obtain the bytes per second (BPS) of the traffic.

- **Dirty Data from Each Source**

  This chart displays the number of dirty data records in the data source of a Realtime Compute for Apache Flink job in different time periods.

- **Auto Scaling Successes and Failures**

  This chart displays the number of auto scaling successes and the number of auto scaling failures.

  > **Notice** This curve chart is suitable only for Realtime Compute for Apache Flink whose version is later than V3.0.0.

- **CPUs Consumed By Auto Scaling**

  This chart displays the number of CPUs consumed when auto scaling is performed.

> **Notice** This curve chart is suitable only for Realtime Compute for Apache Flink whose version is later than V3.0.0.

- **Memory Consumed by Auto Scaling**

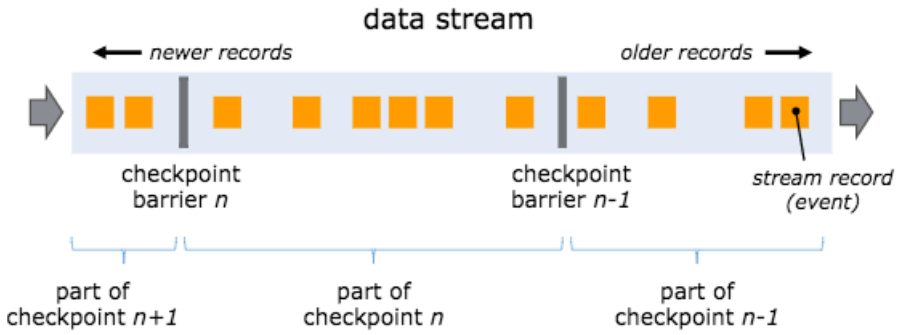
  This chart displays the memory space consumed when auto scaling is performed.

> **Notice** This curve chart is suitable only for Realtime Compute for Apache Flink whose version is later than V3.0.0.

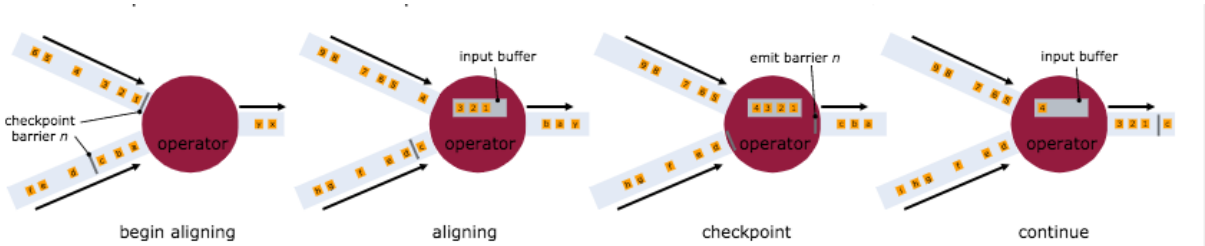
Advanced View

Alibaba Cloud Realtime Compute for Apache Flink provides a fault tolerance mechanism that allows you to restore data streams and ensures that the data streams are consistent with the application. The fault tolerance mechanism is used to create consistent snapshots of distributed data streams and the related states. These snapshots work as consistency checkpoints to which the system can fall back if a failure occurs.

One of the core concepts for distributed snapshots is barriers. Barriers are inserted into data streams and flow together with the data streams to the downstream. Barriers do not overtake data records. The records flow strictly in line. A barrier divides a data stream into two parts. One part enters the current snapshot and the other part enters the next snapshot. Each barrier has a snapshot ID. If the data flows before a barrier is inserted in the data stream, the data is included in the specified snapshot. Barriers are lightweight. Barriers do not interfere with the processing of data streams. Multiple barriers from different snapshots can co-exist in the same data stream. This allows multiple snapshots to be concurrently created.



Barriers are inserted into data streams at the data source. If a barrier from Snapshot n is inserted, the system automatically records the checkpoint of Snapshot n in the data stream. This checkpoint is indicated by Sn. Then, the barrier continues to flow to the downstream. When an intermediate operator receives barriers for Snapshot n from all the input streams, the operator emits only one barrier for Snapshot n to all the output streams. When the sink operator that is the destination of the directed acyclic graph (DAG) stream receives Barrier n from each of its input streams, the operator acknowledges to the checkpoint coordinator that Snapshot n is created. After all the sink operators acknowledge that Snapshot n is created, this snapshot is considered completed.



The following table describes the curve charts of checkpoint metrics.

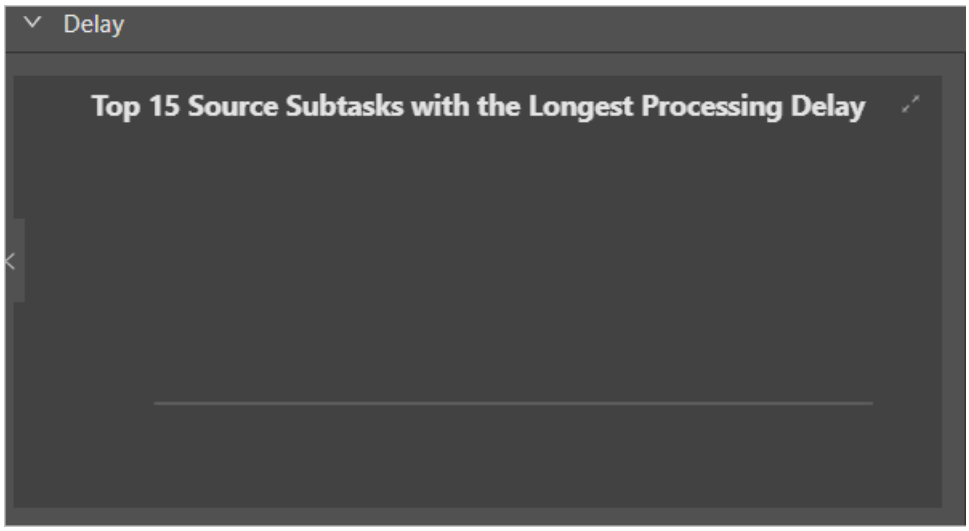
Curve chart	Description
Checkpoint Duration	Displays the time that is consumed to create a checkpoint. Unit: milliseconds.
Checkpoint Size	Displays the memory size that is required to create a checkpoint.
Checkpoint Alignment Time	Displays the duration consumed by all the data streams to flow from the upstream nodes to the node on which you create a checkpoint. When the sink operator receives Barrier n from all the input streams, the operator acknowledges to the checkpoint coordinator that Snapshot n is created. The sink operator represents the destination of the DAG stream. After all the sink operators acknowledge that snapshot n is created, this snapshot is considered completed. This duration is known as the checkpoint alignment time.
Checkpoint Count	Displays the number of checkpoints within a specific period of time.
Get	Displays the longest duration for which a subtask performs a GET operation on the RocksDB within a specific period of time.
Put	Displays the longest duration for which a subtask performs a PUT operation on the RocksDB within a specific period of time.
Seek	Displays the longest duration for which a subtask performs a SEEK operation on the RocksDB within a specific period of time.
State Size	Displays the state size of the job within a specific period of time. If the size increases at a high rate, we recommend that you check for potential issues in the job.
GMS GC Time	Displays the duration for which the underlying container of the job performs garbage collection.
GMS GC Rate	Displays the frequency at which the underlying container of the job performs garbage collection.

Watermark

Curve chart	Description
Watermark Delay	Displays the difference between the watermark time and the system time.
Dropped Records per Second	Displays the number of data records that are dropped per second. If a data record arrives at the window after the watermark time, the data record is dropped.
Dropped Records	Displays the total number of dropped data records. If a data record arrives at the window after the watermark time, the data record is dropped.

Delay

Top 15 Source Subtasks with the Longest Processing Delay



This chart displays the processing delay of each source subtask.

Throughput

Curve chart	Description
Task Input TPS	Displays the data input status of all the tasks in a job.
Task Output TPS	Displays the data output status of all the tasks in a job.

Queue

Curve chart	Description
Input Queue Usage	Displays the data input queue of all the tasks in a job.
Output Queue Usage	Displays the data output queue of all the tasks in a job.

Tracing

Curve chart	Description
Time Used In Processing Per Second	Displays the duration for which a task processes data per second.
Time Used In Waiting Output Per Second	Displays the duration for which a task waits for output data per second.
Task Latency Histogram Mean	Displays the latency of each task.
Wait Output Histogram Mean	Displays the duration for which each task waits for output.
Wait Input Histogram Mean	Displays the duration for which each task waits for input.
Partition Latency Mean	Displays the latency of concurrent tasks in each partition.

Process

Curve chart	Description
Process Memory RSS	Displays the memory usage of each process.
CPU Usage	Displays the CPU utilization of each process.

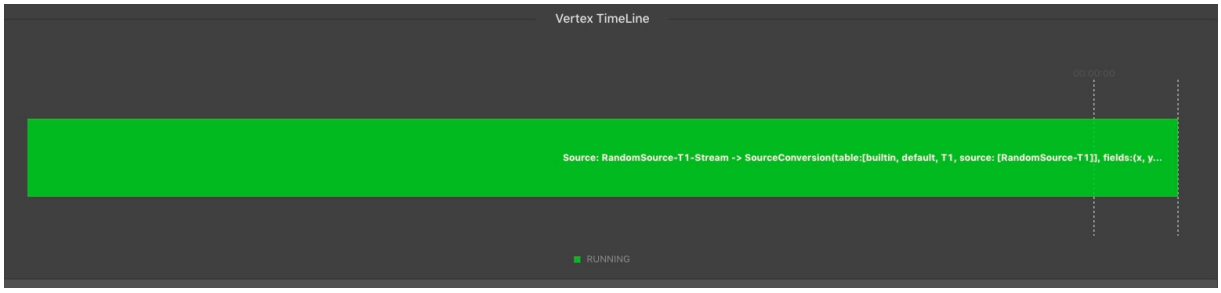
JVM

Curve chart	Description
Memory Heap Used	Displays the Java Virtual Machine (JVM) heap memory usage of a job.
Memory Non-Heap Used	Displays the JVM non-heap memory usage of a job.
Thread Count	Displays the number of threads in a job.
GC(CMS)	Displays the number of times that a job completes garbage collection (GC).

5.4. Timeline

The Timeline page displays the running status of each Vertex from the start offset to the current time.

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.



Go to the Timeline page

1. Go to the **Job Administration** page.
   i. Log on to the Realtime Compute Console.
   ii. In the top navigation bar, click **Administration**.
   iii. In the **Jobs** section, click the target job name under the **Job Name** field.
2. At the top of the **Job Administration** page, click **Timeline**.

5.5. Failover

Alibaba Cloud Realtime Compute provides the Failover page for the current job. On the Failover page, you can view the running status and error messages of the current job.

Go to the Failover page

1. Go to the **Job Administration** page.
   i. Log on to the Realtime Compute Console.
   ii. In the top navigation bar, click **Administration**.
   iii. In the **Jobs** section, click the target job name under the **Job Name** field.
2. At the top of the **Job Administration** page, click **Failover**.

Latest FailOver

The **Latest FailOver** tab displays the current errors of the job.

> **Note** This feature is only applicable to Realtime Compute V3.0 or earlier.

FailOver History

The **FailOver History** tab displays the historical errors of the job.

> **Note** This feature is only applicable to Realtime Compute V3.0 or earlier.

Root Exception

The **Root Exception** tab displays the current exceptions of the job.

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

Exception History

The **Exception History** tab displays the historical exceptions of the job.

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

5.6. Checkpoints

Alibaba Cloud Realtime Compute provides a fault tolerance that allows you to restore data streams and make sure that the data streams are consistent with the application. The central part of the fault tolerance is to create consistent snapshots of distributed data streams and their states. These snapshots act as consistency checkpoints to which the system can fall back when a failure occurs.

Go to the Checkpoints page

1. Go to the **Job Administration** page.
   i. Log on to the Realtime Compute Console.
   ii. In the top navigation bar, click **Administration**.
   iii. In the **Jobs** section, click the target job name under the **Job Name** field.
2. At the top of the **Job Administration** page, click **Checkpoints**.

Overview

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

The **Overview** tab displays the latest checkpoint information, such as the process, duration, and state size of the checkpoint at each node.

History

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

The **History** tab displays the recent checkpoint information. Click the plus sign (+) at the beginning of the row to display the checkpoint information, such as the process, duration, and state size of the checkpoint at each node.

Summary

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

The **Summary** tab displays the average, maximum, and minimum values of completed checkpoints.

Configuration

> **Note** This feature is only applicable to Realtime Compute V3.0 or later.

The **Configuration** tab displays the configuration information of the checkpoints.

Completed Checkpoints

> **Note** This feature is only applicable to Realtime Compute V3.0 or earlier.

The **Completed Checkpoints** tab displays the information about the completed checkpoints.

Parameter	Description
ID	The ID of the checkpoint.
Start Time	The start time of the checkpoint.
Durations (ms)	The time spent on creating the checkpoint.

Task Latest Completed Checkpoint

> **Note** This feature is only applicable to Realtime Compute V3.0 or earlier.

The **Task Latest Completed Checkpoint** tab displays the details about the latest checkpoint.

Parameter	Description
SubTask ID	The ID of the subtask.
State Size (Bytes)	The state size of the checkpoint.
Durations (ms)	The time spent on creating the checkpoint.

5.7. JobManager

JobManager plays an important part in the startup process of a Realtime Compute for Apache Flink cluster. You can view the JobManager parameter information on the JobManager tab.

Go to the JobManager tab

1. Go to the **Job Administration** page.
2. On the **Job Administration** page, click the **JobManager** tab.

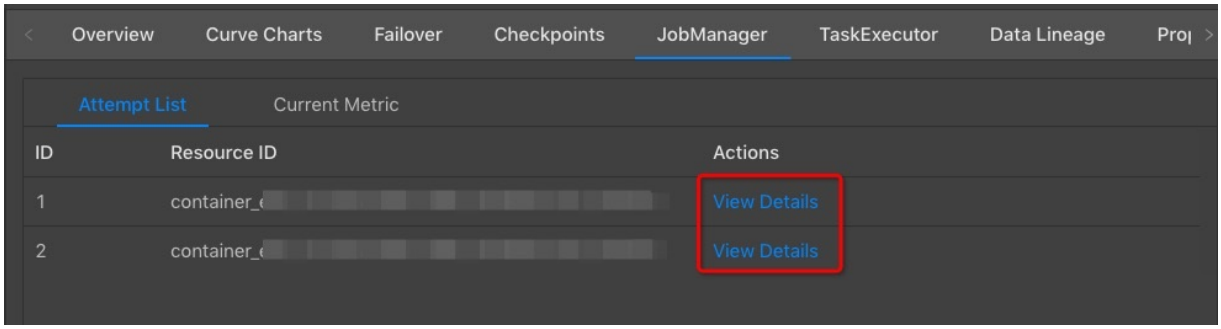
Role of JobManager in the cluster startup process

JobManager plays an important part in the startup process of a Realtime Compute for Apache Flink cluster. The following items describe the startup process of a Realtime Compute for Apache Flink cluster:

1. When a Realtime Compute for Apache Flink cluster is started, one JobManager and several TaskExecutors are started at the same time.
2. The client submits tasks to the JobManager.
3. The JobManager assigns tasks to TaskExecutors.
4. The TaskExecutors report the heartbeat and statistical information to the JobManager.

JobManager parameters

On the Job Administration page, click the **JobManager** tab. On the **Attempt List** tab, click **View Details** in the **Actions** column to view detailed information about the JobManager.



5.8. TaskExecutor

This topic describes the role of TaskExecutors in the startup process of a Realtime Compute for Apache Flink cluster and explains the TaskExecutor tab.

> **Notice** This topic applies to only Realtime Compute for Apache Flink whose version is earlier than V3.0.

Background information

TaskExecutors are an indispensable part in starting a Realtime Compute for Apache Flink cluster. TaskExecutors receive tasks from and return execution results to the JobManager. The number of slots is specified when a TaskExecutor is started. Only one task thread can be executed in each slot. A TaskExecutor receives tasks from the JobManager, and then builds a Netty connection with its upstream to receive and process data.

Go to the TaskExecutor tab

1. Log on to the .
2. In the top navigation bar, click **Administration**.
3. In the **Jobs** section, click the name of the required job under the **Job Name** field.
4. On the **Job Administration** page, click the **TaskExecutor** tab.

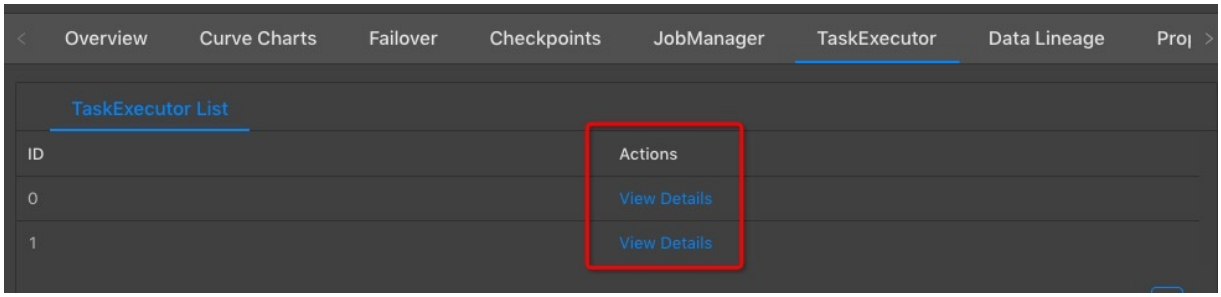
Role of TaskExecutors in cluster startup

TaskExecutors are an indispensable part in starting a Realtime Compute for Apache Flink cluster. The following items describe the startup process of a Realtime Compute for Apache Flink cluster:

1. When a Realtime Compute for Apache Flink cluster is started, one JobManager and several TaskExecutors are started at the same time.
2. The client submits tasks to the JobManager.
3. The JobManager assigns tasks to TaskExecutors.
4. The TaskExecutors report the heartbeat and statistical information to the JobManager.

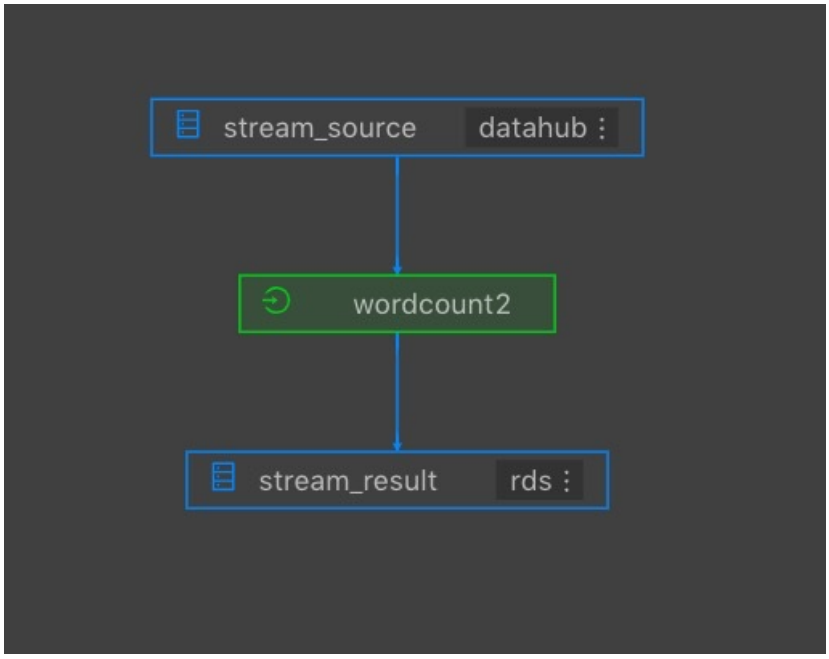
TaskExecutor tab

The TaskExecutor tab provides a list of tasks and the entries to their details.



5.9. Data lineage

The data lineage of a Realtime Compute job reflects the dependency between upstream and downstream data of the job. In scenarios where the business dependency between the upstream and downstream data of a job is complex, Realtime Compute provides a data topology on the Data Lineage page to clearly show the dependency.



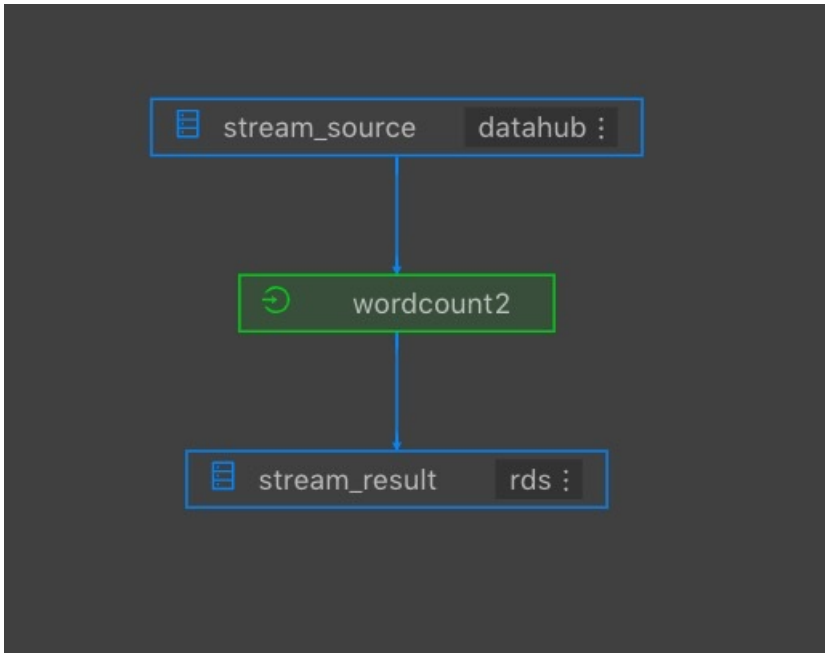
Go to the Data Lineage page

1. Go to the **Job Administration** page.
   i. Log on to the Realtime Compute Console.
   ii. In the top navigation bar, click **Administration**.
   iii. In the **Jobs** section, click the target job name under the **Job Name** field.
2. At the top of the **Job Administration** page, click **Data Lineage**.

Data sampling

The Data Lineage page provides the data sampling feature for source tables and result tables of jobs. The data to be sampled is the same as the data displayed on the data development page. The data sampling feature allows you to check data at any time on the data administration page, thus facilitating fault locating. To enable the data sampling feature, follow these steps:

1. Click the table name in the upstream and downstream of the job.



2. At the bottom of the **Data Sampling** page, click **OK**.

5.10. Properties and parameters

The Properties and Parameters tab provides detailed information about the current job, such as the current running information and running history.

Go to the Properties and Parameters tab

1. Go to the **Job Administration** page in the Realtime Compute for Apache Flink console.
2. On the **Job Administration** page, click the **Properties and Parameters** tab.

Code

On the **Code** tab, you can preview the SQL job code. In the upper-right corner of the Code tab, click **Edit Job** to go to the **Development** page.

Resource Configuration

- On the **Resource Configuration** tab, you can view the configuration of the resources that are used in a job, such as CPUs, memory, and parallelism.
- After auto scaling is enabled, you can query the auto scaling iteration history on the Resource Configuration tab.

> **Note** Only Realtime Compute for Apache Flink V3.0.0 and later allow you to query the auto scaling iteration history.

Properties

On the **Properties** tab, you can view basic information about a job.

Runtime Parameters

On the **Runtime Parameters** tab, you can view the job running parameters, such as the underlying checkpoint and start time.

History

On the **History** tab, you can view the operation information about a job, such as **Operated By**, **Start Offset**, and **End Time**.

Parameters

On the **Parameters** tab, you can specify the job parameters supported by Realtime Compute for Apache Flink. For example, you can customize the delimiter for debugging.

5.11. Job diagnosis

Realtime Compute offers job diagnosis to help you troubleshoot job issues.

> **Note** Only jobs that are in the running state can be diagnosed.

Procedure

1. Log on to the .
2. In the top navigation bar, click **Administration**.
3. On the Administration page that appears, find the target job, move the pointer over the More icon in the **Actions** column, and click **Check**.

Check metrics

- **Failover**
  - **Job failover**: Check whether the job encountered a failover within the last 30 minutes.
  - **Application Master (AM) failover**: Check whether a failover is detected in AM.
- **Blink Metric Job latency**: Check the job latency. If a latency occurs, the nodes with backpressure are displayed.
  - **High latency**: The latency is longer than 100 seconds and shorter than 200 seconds.
  - **Excessively high latency**: The latency is 200 seconds or longer.
- This tab displays the Yarn check result.
- This tab displays the **OS** check result.

6.Job optimization

6.1. Overview

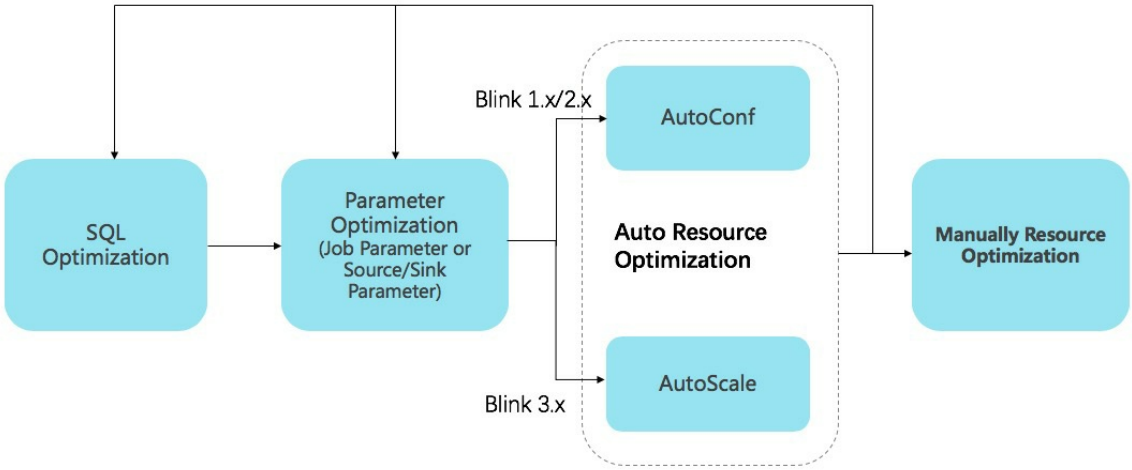
After you implement the business logic, and then publish and start a Realtime Compute job, you need to optimize the job to meet performance requirements.

Purposes

- Jobs can start and run properly.
- The job latency and throughput meet performance requirements.
- Resources can be used efficiently to reduce the cost.

Procedure

The following figure shows the procedure of job optimization.



1. Optimize the SQL code.

   SQL optimization allows you to select an appropriate SQL implementation method based on business requirements. For example, you can optimize aggregation functions, resolve data hotspot issues, optimize the TopN algorithm, use built-in functions, deduplicate data records, and avoid use of regular expressions. For more information, see Recommended Flink SQL practices.

2. Optimize performance by adjusting parameter settings.
   - Adjust job parameter settings.

     Select an underlying optimization policy. For example, you can enable miniBatch to reduce state data access. For more information, see Job parameters.

   - Adjust parameter settings of upstream and downstream data storage.

     Optimize the read and write operations performed on the upstream and downstream storage systems. For example, you can read or write data in batches to improve the throughput. You can also configure the cache policy to improve the efficiency of joining dimension tables. For more information, see Upstream and downstream storage parameters.

3. Optimize resource configuration automatically.

   To simplify job optimization, Realtime Compute provides the automatic configuration optimization feature. We recommend that you use this feature for job optimization. For more information, see Performance optimization by using auto scaling.

4. Optimize resource configuration manually or repeat the optimization process.
   - Optimize resource configuration manually.

     If automatic configuration optimization cannot meet your requirements, you can manually optimize the resource configuration. For more information, see Optimize performance by manual configuration.
   - Repeat the optimization process.

     If the optimization result cannot meet your requirements, repeat the previous steps.

6.2. Recommended Flink SQL practices

This topic describes the recommended syntax, configurations, and functions used to optimize Flink SQL performance.

Optimize the Group By functions

- Enable microBatch or miniBatch to improve the throughput

  The microBatch and miniBatch policies are both used for micro-batch processing. If either of the policies is enabled, Realtime Compute for Apache Flink processes data when the data cache meets the trigger condition. This reduces the frequency at which Realtime Compute for Apache Flink accesses the state data, and therefore improves the throughput and reduces data output.

  The microBatch and miniBatch policies are different from each other in terms of the trigger mechanism. The miniBatch policy triggers micro-batch processing by using the timer threads that are registered with each task. This consumes some thread scheduling overheads. The microBatch policy is an enhancement of the miniBatch policy. The microBatch policy triggers micro-batch processing based on event messages, which are inserted into the data sources at a specific interval. The microBatch policy outperforms the miniBatch policy because it provides higher data serialization efficiency, reduces backpressure, and achieves higher throughput at a lower latency.

  - Use scenarios

    Micro-batch processing achieves higher throughput at the expense of higher latency. We recommend that you do not enable micro-batch processing in scenarios that require extremely low latency. However, in data aggregation scenarios, we recommend that you enable micro-batch processing to improve job performance.

    > **Note** You can also enable microBatch to resolve data jitter when data is aggregated in two phases.

- Enabling method

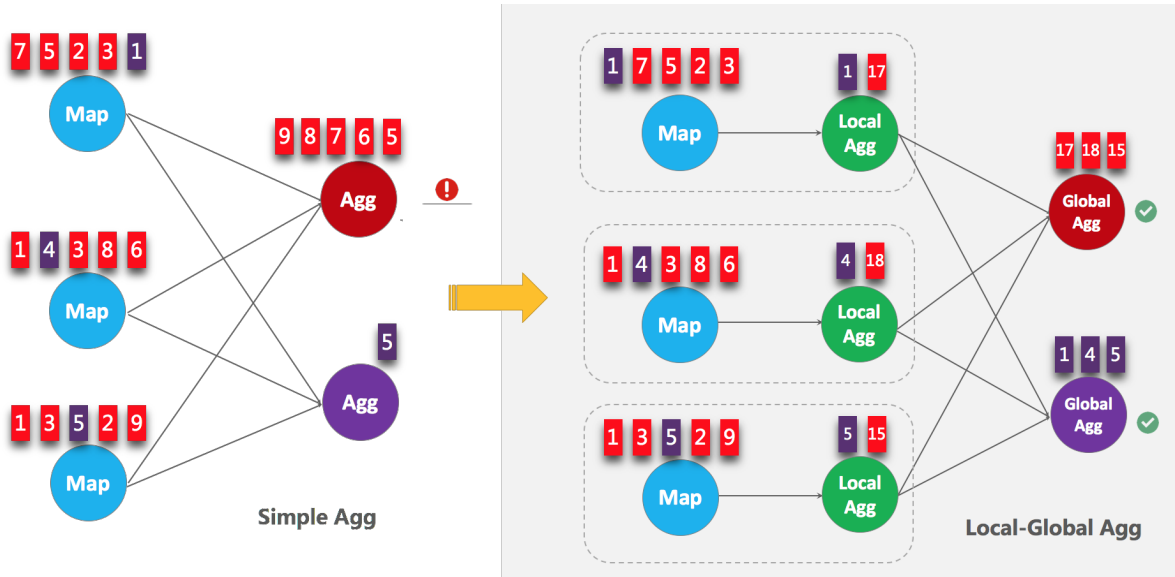
  microBatch and miniBatch are disabled by default. To enable them, configure the following parameters:

```
# Enable window miniBatch in Realtime Compute for Apache Flink V3.2 or later. By defaul
t, window miniBatch is disabled for Realtime Compute for Apache Flink V3.2 or later.
sql.exec.mini-batch.window.enabled=true
# The interval at which a large amount of data is generated. You must specify this para
meter when you enable microBatch. We recommend that you set this parameter to the same
value as that of blink.miniBatch.allowLatencyMs.
blink.microBatch.allowLatencyMs=5000
# When you enable microBatch, you must reserve the settings of the following two miniBa
tch parameters:
blink.miniBatch.allowLatencyMs=5000
# The maximum number of data records that can be cached for each batch. You must set th
is parameter to avoid the out of memory (OOM) error.
blink.miniBatch.size=20000
```

- Enable LocalGlobal to resolve common data hotspot issues

  The LocalGlobal policy divides the aggregation process into two phases: local aggregation and global aggregation. They are similar to the combine and reduce phases in MapReduce. In the local aggregation phase, Realtime Compute for Apache Flink locally aggregates a micro batch of data at each input node (LocalAgg), and generates an accumulator value for each batch (accumulator). In the global aggregation phase, Realtime Compute for Apache Flink merges the accumulator values (merge) to obtain the final result (GlobalAgg).

  The LocalGlobal policy can eliminate data skew by using local aggregation and resolve data hotspot issues in global aggregation. Therefore, job performance is enhanced. The following figure shows how the LocalGlobal policy resolves data skew issues.



- Use scenarios

  You can enable LocalGlobal to improve the performance of general aggregate functions, such as SUM, COUNT, MAX, MIN, and AVG, and resolve data hotspot issues when you execute these functions.

  > **Note** To enable LocalGlobal, you must define a user-defined aggregate function (UDAF) to implement the `merge` method.

- Enabling method

  In Realtime Compute for Apache Flink V2.0 or later, LocalGlobal is enabled by default. When the blink.localAgg.enabled parameter is set to true, LocalGlobal is enabled. This parameter takes effect only when microBatch or miniBatch is enabled.

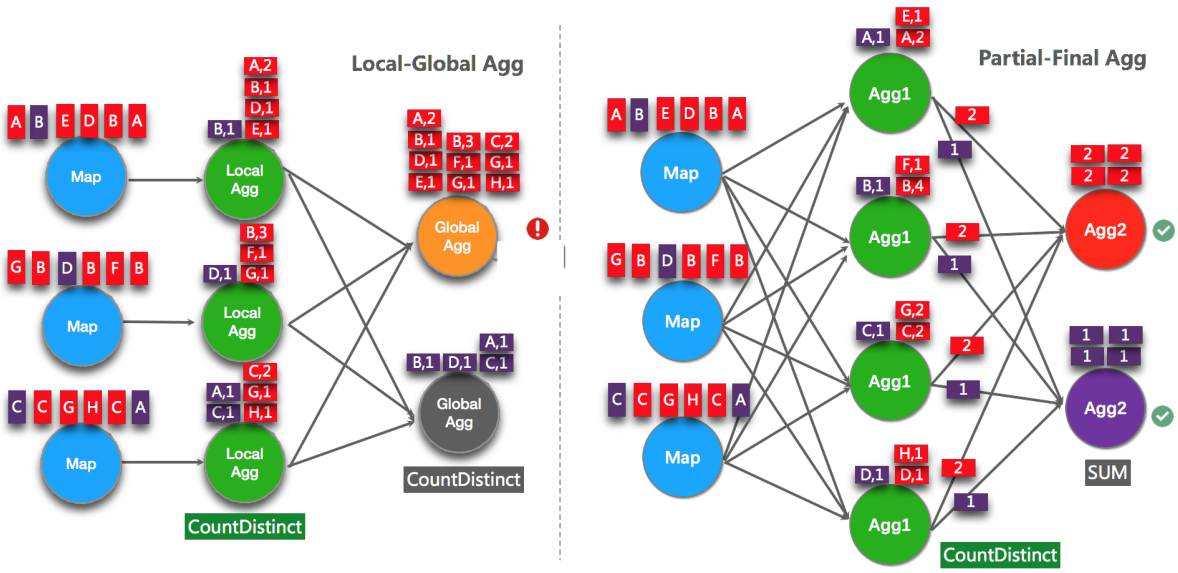
- Verification

  To determine whether LocalGlobal is enabled, check whether the **GlobalGroupAggregate** or **LocalGroupAggregate** node exists in the generated topology.

- Enable PartialFinal to resolve data hotspot issues when you execute the COUNT DISTINCT function

  The LocalGlobal policy effectively improves the performance of general aggregate functions, such as SUM, COUNT, MAX, MIN, and AVG. However, it is not effective for improving the performance of the COUNT DISTINCT function. This is because local aggregation cannot effectively remove duplicate distinct keys. As a result, a large amount of data remains stacked up in the global aggregation phase.

  If you execute the COUNT DISTINCT function in Realtime Compute for Apache Flink versions earlier than V2.2.0, you must add a layer that scatters data by a distinct key so that you can divide the aggregation process into two phases to resolve data hotspot issues. Realtime Compute for Apache Flink `V2.2.0` and later versions provide the PartialFinal policy to automatically scatter data and divide the aggregation process. The following figure shows the differences between LocalGlobal and PartialFinal.



- Use scenarios

  The PartialFinal policy applies to scenarios in which the aggregation performance cannot meet your requirements when you use the COUNT DISTINCT function.

  > **Note**
  > - You cannot enable PartialFinal in the Flink SQL code that contains UDAFs.
  > - We recommend that you enable PartialFinal only when the amount of data is large. This is because the PartialFinal policy automatically scatters data to two aggregation layers and introduces additional network shuffling. If the amount of data is not large, resources are wasted.

- Enabling method

  PartialFinal is disabled by default. To enable PartialFinal, set the `blink.partialAgg.enabled` parameter to true.

- Verification

  To determine whether PartialFinal is enabled, check whether **expandable** nodes exist in the generated topology, or whether the number of aggregation layers changes from one to two.

- Use the AGG WITH FILTER syntax to improve job performance when you use the COUNT DISTINCT function

  > **Note** Only Realtime Compute for Apache Flink V2.2.2 and later versions support this syntax.

  Statistical jobs record unique visitors (UVs) in different dimensions, such as UVs of the entire network, UVs of mobile clients, and UVs of PCs. We recommend that you use the standard AGG WITH FILTER syntax instead of AGG WITH CASE WHEN to implement multi-dimensional statistical analysis. The SQL optimizer of Realtime Compute for Apache Flink can analyze the filter parameter. This way, Realtime Compute for Apache Flink can execute the COUNT DISTINCT function on the same field with different filter conditions by sharing the state data. This reduces the read and write operations on the state data. The performance test shows that the use of AGG WITH FILTER improves job performance by one time higher than the use of AGG WITH CASE WHEN.

  - Use scenarios

    We recommend that you replace the AGG WITH CASE WHEN syntax with the AGG WITH FILTER syntax. This particularly improves job performance when you execute the COUNT DISTINCT function on the same field with different filter conditions.

  - Original statement

    ```
    COUNT(distinct visitor_id) as UV1 , COUNT(distinct case when is_wireless='y' then visitor_id else null end) as UV2
    ```

  - Optimized statement

    ```
    COUNT(distinct visitor_id) as UV1 , COUNT(distinct visitor_id) filter (where is_wireless='y') as UV2
    ```

Optimize the TopN algorithm

- TopN algorithm

If the input streams of TopN are static streams (such as source), TopN supports only one algorithm: AppendRank. If the input streams of TopN are dynamic streams (such as streams that are processed by using the AGG or JOIN function), TopN supports the following three algorithms in descending order of performance: UpdateFastRank, UnaryUpdateRank, and RetractRank. The name of the algorithm used is contained in the node name in the topology.

- UpdateFastRank is the optimal algorithm.

  The following two conditions must be met if you want to use this algorithm:

  - The input streams must contain the primary key information, such as ORDER BY AVG.
  - The values of the fields or functions in the ORDER BY clause are updated monotonically in the opposite order of sorting. For example, you can define the ORDER BY clause as ORDER BY COUNT, ORDER BY COUNT_DISTINCT, or ORDER BY SUM (positive) DESC. This optimization is supported only in Realtime Compute for Apache Flink V2.2.2 or later.

    If you want to obtain an optimization plan, you must add a filter condition in which the SUM parameter is positive when you use ORDER BY SUM DESC, and make sure that the value of total_fee is positive.

```
insert
  into print_test
SELECT
  cate_id,
  seller_id,
  stat_date,
  pay_ord_amt -- The rownum field is not included in the output data. This reduces th
e amount of output data to be written to the result table.
FROM (
    SELECT
      *,
      ROW_NUMBER () OVER (
        PARTITION BY cate_id,
        stat_date -- Ensure that the stat_date field is included. Otherwise, the data
may be disordered when the state data expires.
        ORDER
          BY pay_ord_amt DESC
      ) as rownum -- Sort data by the sum of the input data.
    FROM (
        SELECT
          cate_id,
          seller_id,
          stat_date,
          -- Note: The result of the SUM function is monotonically increasing because
the values returned by the SUM function are positive. Therefore. TopN can use optimiz
ed algorithms to obtain top 100 data records.
          sum (total_fee) filter (
            where
              total_fee >= 0
          ) as pay_ord_amt
        FROM
          random_test
        WHERE
          total_fee >= 0
        GROUP
          BY cate_name,
          seller_id,
          stat_date
      ) a
  )
WHERE rownum <= 100;
```

  - UnaryUpdateRank is second only to UpdateFastRank in terms of performance. To use this algorithm, make sure that the input streams contain the primary key information.
  - RetractRank ranks last in terms of performance. We recommend that you do not use this algorithm in the production environment. Check input streams. If input streams contain the primary key information, use UnaryUpdateRank or UpdateFastRank to optimize job performance.

- Optimization method

- Exclude the rownum field

  Do not include rownum in the output of TopN. We recommend that you sort the results immediately after they are displayed in the frontend. This can significantly reduce the amount of data that needs to be written to the result table. For more information, see TopN.

- Increase the cache size of TopN

  TopN provides a state cache to improve the access efficiency of state data. This improves the performance. The following formula is used to calculate the hit rate of TopN cache:

  ```
  cache_hit = cache_size*parallelism/top_n/partition_key_num
  ```

  Take Top100 as an example. Assume that the cache contains 10,000 records and the parallelism is 50. If the number of keys for the PARTITION BY function is 100,000, the cache hit rate equals 5% (10000 × 50/100/100000 = 5%). The hit rate is low, which indicates that large amounts of requests will access the disk state data. As a result, job performance significantly deteriorates. Therefore, if the number of keys for the PARTITION BY function is large, you may increase the cache size and heap memory of TopN. For more information, see Optimize performance by manual configuration.

  ```
  ## In this example, if you increase the cache size of TopN from the default value 10000
  to 200000, the cache hit rate may reach 100% (200000 × 50/100/100000 = 100%).
  blink.topn.cache.size=200000
  ```

- Include a time field in the PARTITION BY function

  For example, you want to include the day field in your statement for a daily ranking. Otherwise, the TopN result may become disordered when the state data expires.

Optimize the deduplication performance

> **Note** Only Blink 3.2.1 supports efficient deduplication solutions.

Input streams of Realtime Compute for Apache Flink may contain duplicate data. Therefore, deduplication is highly required. Realtime Compute for Apache Flink offers two policies to efficiently remove duplicate data: Deduplicate Keep FirstRow and Deduplicate Keep LastRow.

- Syntax

  Flink SQL does not support deduplication statements. To reserve the first or last duplicate record under the specified primary key and discard the rest of the duplicate records as required, Realtime Compute for Apache Flink uses the ROW_NUMBER OVER WINDOW statement of Flink SQL. Deduplication is a special TopN statement.

  ```
  SELECT *
  FROM (
     SELECT *,
      ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]
        ORDER BY timeAttributeCol [asc|desc]) AS rownum
     FROM table_name)
  WHERE rownum = 1
  ```

Element	Description

Element	Description
ROW_NUMBER()	Specifies an over window to compute the row number. The row number starts from 1.
PARTITION BY col1[, col2..]	Optional. Specifies the partition key columns that store primary keys of duplicate records.
ORDER BY timeAttributeCol [asc|desc])	Specifies the column that sorts records based on a time attribute (proctime or rowtime). You can sort records in ascending order (Deduplicate Keep FirstRow) or descending order (Deduplicate Keep LastRow) based on the time attribute.
rownum	Specifies the number of rows. You can set this element in either of the following ways: `rownum = 1` and `rownum <= 1`.

Based on the preceding syntax, deduplication includes two steps:

i. Use the `ROW_NUMBER()` window function to sort data by the specified time attribute and mark the data with rankings.

> **Note**
> - If the time attribute is proctime, Realtime Compute for Apache Flink removes duplicate records based on the time at which the records are processed by Realtime Compute for Apache Flink. In this case, the ranking may vary each time.
> - If the time attribute is rowtime, Realtime Compute for Apache Flink removes duplicate records based on the time at which the records are written to Realtime Compute for Apache Flink. In this case, the ranking always remains the same.

ii. Reserve the first record under the specified primary key and remove the rest of the duplicate records.

> **Note** You can sort records in ascending or descending order of the time attribute.
> - Deduplicate Keep FirstRow: Realtime Compute for Apache Flink sorts records in ascending order of the time attribute and reserves the first record under the specified primary key.
> - Deduplicate Keep LastRow: Realtime Compute for Apache Flink sorts records in descending order of the time attribute and reserves the first record under the specified primary key.

- Deduplicate Keep FirstRow

  If you select the Deduplicate Keep FirstRow policy, Realtime Compute for Apache Flink reserves the first record under the specified primary key but discards the rest of the duplicate records. In this case, the state data stores only the primary key information, and the access efficiency of the state data is significantly improved. The following sample code is an example:

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b ORDER BY proctime) as rowNum
  FROM T
)
WHERE rowNum = 1
```

> **Note** The preceding code removes duplicate records from table T based on the b field, and reserves the first record under the specified primary key based on the system time. The proctime attribute indicates the processing time attribute. Realtime Compute for Apache Flink sorts data records in table T based on this attribute. To remove duplicate records based on the system time, you can also call the `PROCTIME` function to avoid the need to declare the proctime attribute.

- Deduplicate Keep LastRow

  If you select the Deduplicate Keep LastRow policy, Realtime Compute for Apache Flink reserves the last record under the specified primary key and discards the rest of the duplicate records. This policy slightly outperforms the LAST_VALUE function in terms of performance. The following sample code of Deduplicate Keep LastRow is an example:

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY rowtime DESC) as rowNum
  FROM T
)
WHERE rowNum = 1
```

> **Note** The preceding code removes duplicate records in table T based on the b and d fields, and reserves the last record under the specified primary key based on the time at which the records are written to Realtime Compute for Apache Flink. The rowtime attribute indicates the event time at which the records are written to Realtime Compute for Apache Flink. Realtime Compute for Apache Flink sorts records in table T based on this attribute.

Use efficient built-in functions

- Use built-in functions to replace user-defined extensions (UDXs)

  Built-in functions of Realtime Compute for Apache Flink are under continual optimization. We recommend that you use built-in functions to replace UDXs whenever possible. Realtime Compute for Apache Flink V2.0 optimizes built-in functions in the following aspects:

  - Improves serialization and deserialization efficiency.
  - Allows operations at the byte level.

- Use single-character delimiters in the KEYVALUE function

The signature of the KEYVALUE function is `KEYVALUE(content, keyValueSplit, keySplit, keyName)`. When keyValueSplit and KeySplit are single-character delimiters, such as a colon (:) or a comma (,), Realtime Compute for Apache Flink uses an optimization algorithm. Realtime Compute for Apache Flink directly searches for the required keyName values among the binary data without the need to segment the entire content. This improves job performance by approximately 30%.

- Use the MULTI_KEYVALUE function when multiple key-value pairs exist

  > **Note** The MULTI_KEYVALUE function is supported only in Realtime Compute for Apache Flink V2.2.2 or later.

  Job performance is significantly affected if a query involves multiple KEYVALUE functions on the same content. Assume that the content contains 10 key-value pairs. To extract all the 10 values and use them as fields, you must write 10 KEYVALUE functions to parse the content 10 times. As a result, job performance deteriorates.

  In this case, we recommend that you use the table-valued function MULTI_KEYVALUE, which requires only one SPLIT parsing on the content. This improves job performance by 50% to 100%.

- Use the LIKE operator with caution
  - To match records that start with the specified content, use `LIKE 'xxx%'`.
  - To match records that end with the specified content, use `LIKE '%xxx'`.
  - To match records that contain the specified content, use `LIKE '%xxx%'`.
  - To match records that are the same as the specified content, use `LIKE 'xxx'`, which is equivalent to `str = 'xxx'`.
  - To match an underscore (`_`), use `LIKE '%seller/id%' ESCAPE '/'`. The underscore (`_`) is escaped because it is a single-character wildcard in SQL and can match any characters. If you use `LIKE '%seller_id%'`, a lot of results are returned, such as `seller_id`, `seller#id`, `sellerxid`, and `seller1id`. These results may be unsatisfactory.

- Avoid the use of regular expressions

  Running regular expressions can be time-consuming and may require a hundred more times of computing resources in comparison with other operations such as plus, minus, multiplication, and division. If you run regular expressions under some particular circumstances, your job may be stuck in an infinite loop. Therefore, use the LIKE operator whenever possible. For information about common regular expressions, click the following related link:

  - REGEXP
  - REGEXP_EXTRACT
  - REGEXP_REPLACE

Optimize network transmission

Common partitioner policies include:

1. KeyGroup/Hash: distributes data based on specified keys.
2. Rebalance: distributes data to each channel by using round-robin scheduling.
3. Dynamic-Rebalance: dynamically distributes data to channels with lower load based on the load status of output channels.
4. Forward: similar to Rebalance if keys and channels are unchained. If keys and channels are chained,

Realtime Compute for Apache Flink distributes data under specified keys to the related channels.

5. Rescale: distributes data in one-to-many or many-to-one mode between input and output channels.

- Use Dynamic-Rebalance to replace Rebalance

  When you use Dynamic-Rebalance, Realtime Compute for Apache Flink writes data to subpartitions with lower load based on the amount of buffered data in each subpartition so that it can achieve dynamic load balancing. Compared with the static Rebalance policy, Dynamic-Rebalance can balance the load and improve the overall job performance when the computing capacity of output computing nodes is unbalanced. If you find the load of output nodes is unbalanced when you use Rebalance, you may prefer to use Dynamic-Rebalance. To use Dynamic-Rebalance, set the `task.dynamic.rebalance.enabled` parameter to true. The default value is false.

- Use Rescale to replace Rebalance

  > **Note** Rescale is supported only in Realtime Compute for Apache Flink V2.2.2 or later.

  Assume that you have 5 parallel input nodes and 10 parallel output nodes. If you use Rebalance, each input node distributes data to all 10 output nodes by using round-robin scheduling. If you use Rescale, each input node only needs to distribute data to two output nodes by using round-robin scheduling. This reduces the number of channels, increases the buffering speed of each subpartition, and therefore improves the network efficiency. When input data is even and the numbers of parallel input and output nodes are the same, you can use Rescale to replace Rebalance. To use Rescale, set the `enable.rescale.shuffling` parameter to true. The default value is false.

Recommended configuration

In summary, we recommend that you use the following job configuration:

```
# Exactly-once semantics.
blink.checkpoint.mode=EXACTLY_ONCE
# The checkpoint interval in milliseconds.
blink.checkpoint.interval.ms=180000
blink.checkpoint.timeout.ms=600000
# Realtime Compute for Apache Flink V2.X uses Niagara as the state backend and uses it to s
et the lifecycle (in milliseconds) of the state data.
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000
# Realtime Compute for Apache Flink V2.X enables micro-batch processing with an interval of
five seconds.
blink.microBatch.allowLatencyMs=5000
# The allowed latency for a job.
blink.miniBatch.allowLatencyMs=5000
# The size of a batch.
blink.miniBatch.size=20000
# Enable local aggregation. This feature is enabled by default in Realtime Compute for Apac
he Flink V2.X, but you must manually enable it if you use Realtime Compute for Apache Flink
V1.6.4.
blink.localAgg.enabled=true
# Enable PartialFinal to resolve data hotspot issues when you execute the COUNT DISTINCT fu
nction in Realtime Compute for Apache Flink V2.X.
blink.partialAgg.enabled=true
# Enable UNION ALL for optimization.
blink.forbid.unionall.as.breakpoint.in.subsection.optimization=true
# Enable OBJECT REUSE for optimization.
#blink.object.reuse=true
# Configure garbage collection for optimization. (You cannot set this parameter if you use
a Log Service source table.)
blink.job.option=-yD heartbeat.timeout=180000 -yD env.java.opts='-verbose:gc -XX:NewRatio=3
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:ParallelGCThreads=4'
# Specify the time zone.
blink.job.timeZone=Asia/Shanghai
```

6.3. Performance optimization by using automatic configuration

To improve user experience, Realtime Compute allows you to use automatic configuration to optimize job performance.

> **Note** Automatic configuration applies to Blink 1.0 and Blink 2.0.

Background and scope

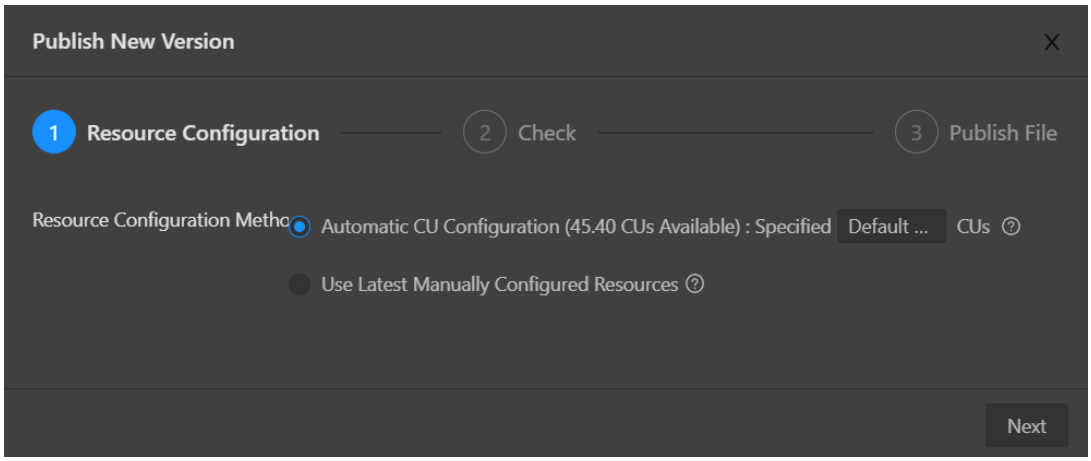
If all the operators and both the upstream and downstream storage systems of your Realtime Compute job meet the performance requirements and remain stable, automatic configuration can help you properly adjust job configurations, such as operator resources and parallelism. It also helps optimize your job throughout the entire process to resolve performance issues such as low throughput or upstream and downstream backpressure.

In the following scenarios, you can use this feature to optimize job performance but cannot eliminate job performance bottlenecks. To eliminate the performance bottlenecks, manually configure the resources or contact the Realtime Compute support team.

- Performance issues exist in the upstream or downstream storage systems of a Realtime Compute job.
  - Performance issues in the data source, such as insufficient DataHub partitions or Message Queue (MQ) throughput. In this case, you must increase the partitions of the relevant source table.
  - Performance issues in a data sink, such as a deadlock in ApsaraDB RDS.
- Performance issues of user-defined extensions (UDXs) such as user-defined functions (UDFs), user-defined aggregate functions (UDAFs), and user-defined table-valued functions (UDTFs) in your Realtime Compute job.

Description

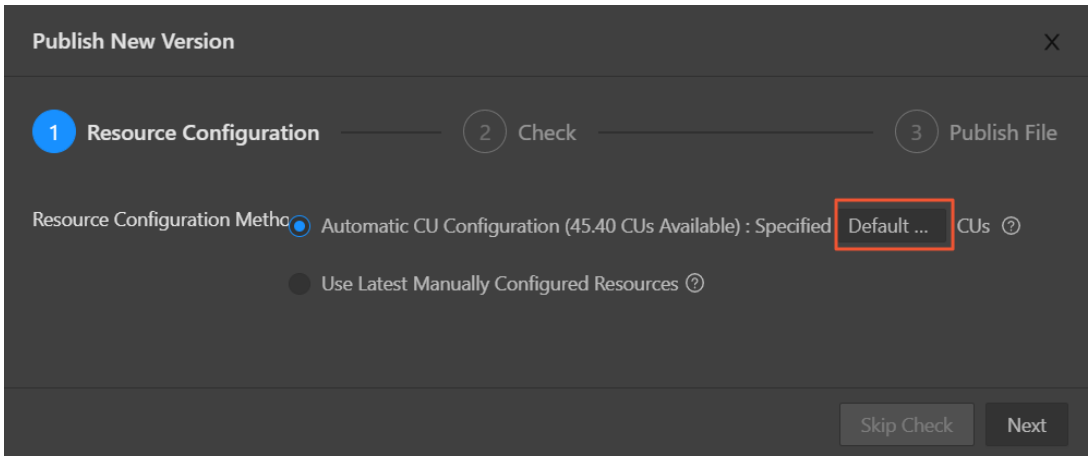
- New jobs
  i. Publish a job.
    a. After you complete SQL development and syntax check on the **Development** page, click **Publish**. The **Publish New Version** dialog box appears.



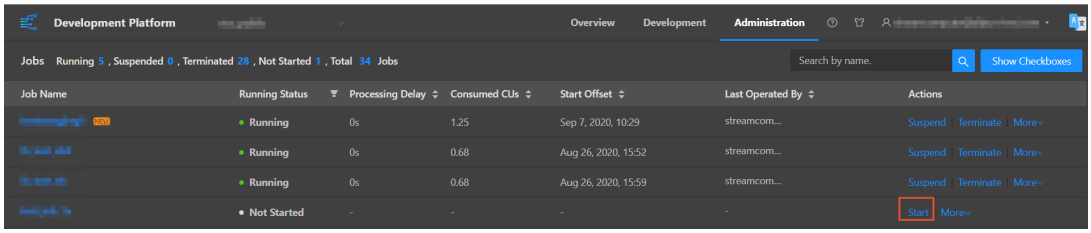
    b. Specify **Resource Configuration Method**.
      - **Automatic CU Configuration**: If you select this option, you can specify the number of compute units (CUs). The automatic configuration algorithm generates an optimized resource configuration and assigns a value for the number of CUs based on the default configuration. If you use automatic CU configuration for the first time, the default number of CUs is used. This algorithm generates an initial configuration based on empirical data when you use automatic CU configuration for the first time. We recommend that you select Automatic CU Configuration if your job has been running for 5 to 10 minutes and its metrics, such as source RPS, remain stable for 2 to 3 minutes. You can obtain the optimal configuration after you repeat the optimization process for three to five times.
      - **Use Latest Manually Configured Resources**: The latest saved resource configuration is used. If the latest resource configuration is generated based on automatic CU configuration, the latest resource configuration is used. If the latest resource configuration is obtained based on the manual configuration, the manual configuration is used.

  ii. Use the default configuration to start the job.

a. Use the default configuration to start the job, as shown in the following figure.



b. On the Administration page, find the job and click **Start** in the Actions column to start the job.



Assume that the default number of CUs generated the first time is 71.

> **Note** Make sure that your job runs longer than 10 minutes and its metrics such as source RPS remain stable for 2 to 3 minutes before you select Automatic CU Configuration for Resource Configuration Method.

iii. Use the automatic CU configuration to start a job.

a. Resource performance optimization

If you select Automatic CU Configuration for Resource Configuration Method and specify 40 CUs to start your job, you can change the number of CUs based on your job to optimize resource performance.

- Determine the minimum number of CUs.

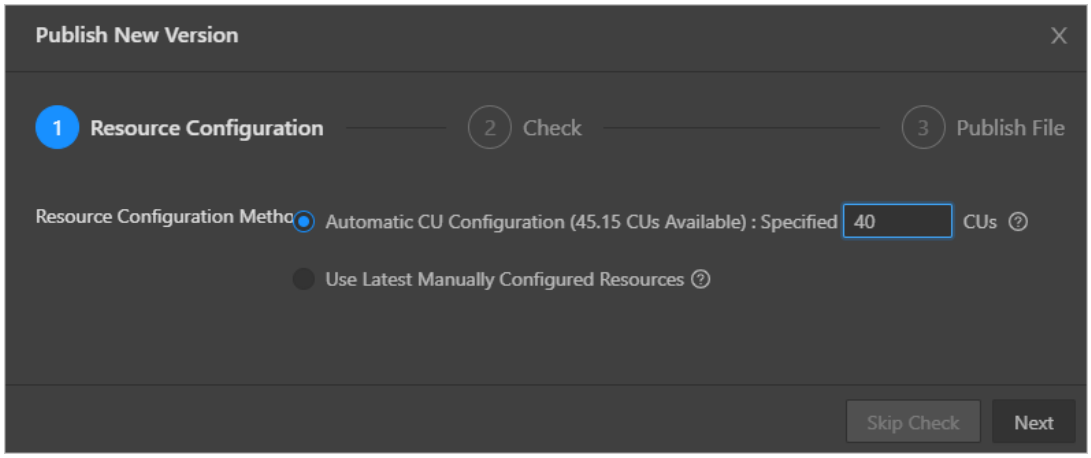
  We recommend that you set the number of CUs to a value that is greater than or equal to 50% of the default value. The number of CUs cannot be less than 1. Assume that the default number of CUs for automatic CU configuration is 71. The recommended minimum number of CUs is 36, which is calculated by using the following formula: 71 CUs × 50% = 35.5 CUs.

- Increase the number of CUs.

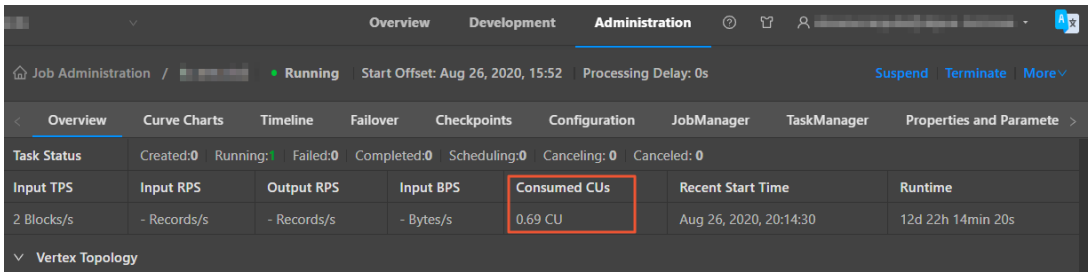
  If the throughput of your Realtime Compute job does not meet your requirements, increase the number of CUs. We recommend that you increase the number of CUs by more than 30% of the current value. For example, if the number of CUs that you specified last time is 10 CUs, you can increase the number to 13.

- Repeat the optimization process.

  If the first optimization attempt does not meet your requirements, repeat the process until you obtain the desired results. You can change the number of CUs based on your job status after each optimization attempt.



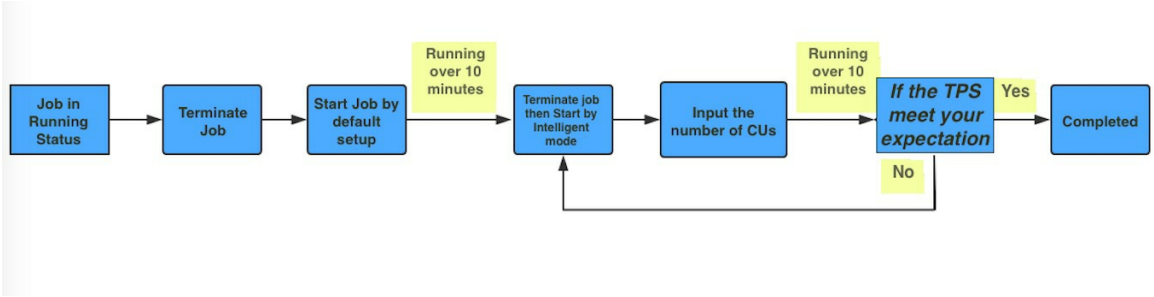
b. View the result of optimization. The following figure shows an example.



> **Note** Do not select **Use Latest Manually Configured Resources** for a new job. Otherwise, an error is returned.

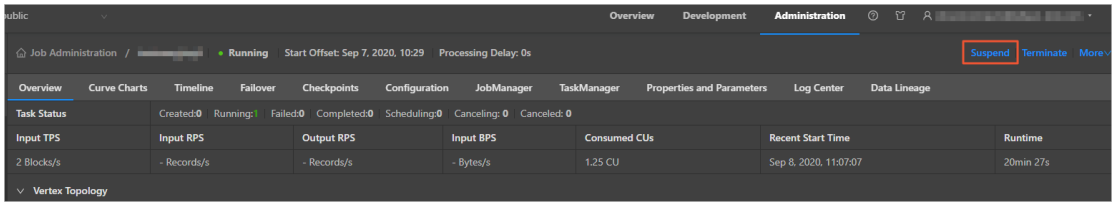
- Existing jobs

- The following figure shows the optimization process of automatic configuration.

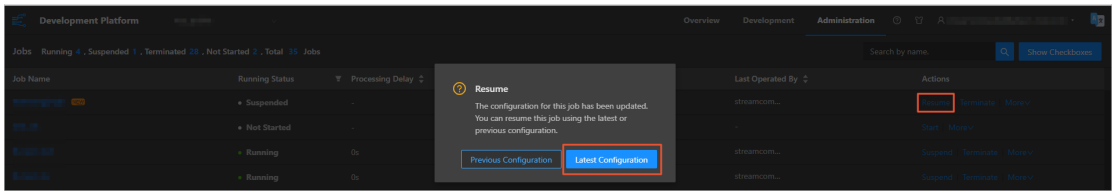


> **Note**
>
> - Before you use automatic configuration for a job that is in the running state, check whether stateful operations are involved. This is because the saved state data of a job may be cleared during the optimization process of automatic configuration.
> - If you make changes to a job, for example, modifying SQL statements or changing the Realtime Compute version, automatic configuration may fail. These changes may lead to topology changes, which results in some issues. For example, curve charts may not be able to display the latest data, or the state data may not able to be used for fault tolerance. In this case, resource configurations cannot be optimized based on the job running history and therefore an error is returned when you perform automatic configuration. To rectify the fault, you must treat the changed job as a new job and repeat the previous operations.

- Procedure
  a. Suspend the job.



  b. Repeat the steps performed for new jobs and resume the job with the latest configuration.



FAQ

The optimization result of automatic configuration may not be accurate in the following scenarios:

- If the job runs only for a short period of time, the data collected during data sampling is insufficient. We recommend that you increase the running duration of the job and make sure that the curves of job metrics such as source RPS remain stable for at least 2 to 3 minutes.
- A job fails. We recommend that you check and fix the failure.
- Only a small amount of data is available for a job. We recommend that you retrieve more historical data.

- The effect of automatic configuration is affected by multiple factors. Therefore, the latest configuration obtained by using automatic configuration may not be optimal. If the effect of automatic configuration does not meet your requirements, you can manually configure the resources. For more information, see Optimize performance by manual configuration.

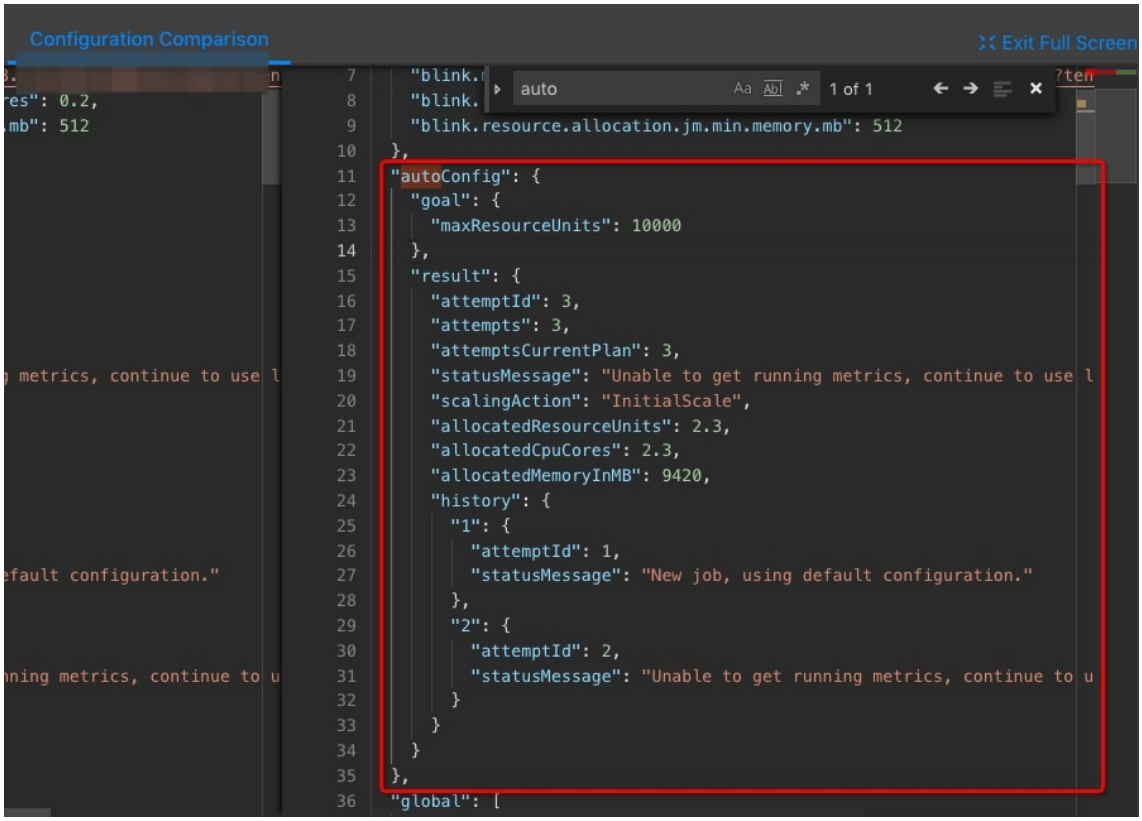
Recommendations

- To help automatic configuration accurately collect the runtime metric information of a job, make sure that the job runs stably for more than 10 minutes before you apply automatic configuration to the job.
- Job performance can be improved after you use automatic configuration for three to five times.
- When you use automatic configuration, you can specify the start offset to retrieve historical data or even accumulate large amounts of data for a job to create backpressure to accelerate the optimization effect.

Method used to determine the effectiveness of automatic configuration

Automatic configuration of Realtime Compute is enabled based on a JSON configuration file. After you use automatic configuration to optimize a job, you can view the JSON configuration file to check whether the feature is running as expected.

- You can view the JSON configuration file by using one of the following methods:
  i. View the file on the job edit page, as shown in the following figure.



  ii. View the file on the Job Administration page, as shown in the following figure.

- JSON configuration description

```
"autoconfig" : {
    "goal": {  // The goal of automatic configuration.
        "maxResourceUnits": 10000.0,  // The maximum number of CUs for a Blink job. This value cannot be changed. Therefore, you can ignore this item when you check whether the feature is running as expected.
        "targetResoureUnits": 20.0  // The number of CUs that you specified. The specified number of CUs is 20.
    },
    "result" : {  // The result of automatic configuration. We recommend that you pay attention to this item.
      "scalingAction" : "ScaleToTargetResource",  // The action of automatic configuration. *
      "allocatedResourceUnits" : 18.5, // The total resources allocated by automatic configuration.
      "allocatedCpuCores" : 18.5,      // The total CPU cores allocated by automatic configuration.
      "allocatedMemoryInMB" : 40960    // The total memory size allocated by automatic configuration.
      "messages" : "xxxx"  // We recommend that you pay attention to these messages. *
    }
}
```

  - scalingAction: If the value of this parameter is `InitialScale`, this is the first time that you use automatic configuration. If the value of this parameter is `ScaleToTargetResource`, this is not the first time that you use automatic configuration.

- If no message appears, automatic configuration runs properly. If some messages appear, you must analyze these messages. Messages are categorized into the following two types:
  - Warning: This type of message indicates that automatic configuration runs properly but you must pay attention to potential issues, such as insufficient partitions in a source table.
  - Error or exception: This type of message indicates that automatic configuration failed. The following error message is usually displayed: `Previous job statistics and configuration will be used`. The automatic configuration for a job fails in the following two scenarios:
    - The job or Blink version is modified before you use automatic configuration. In this case, the previous running information cannot be used for automatic configuration.
    - An error message that contains "exception" is reported when you use automatic configuration. In this case, you must analyze the error based on the job running information and logs. If you do not have enough information, submit a ticket.

Error messages

IllegalStateException

If the following error messages are displayed, the state data cannot be used for fault tolerance. To resolve this issue, terminate the job, clear its state, and then specify the start offset to re-read the data.

If you cannot migrate the job to a backup node, perform the following steps to mitigate the negative impact of service interruption: Roll back the job to an earlier version and specify the start offset to re-read the data during off-peak hours. To roll back the job, click **Versions** on the right side of the **Development** page. On the page that appears, move the pointer over More in the Actions column, click Compare, and then click Roll Back to Version.

```
java.lang.IllegalStateException: Could not initialize keyed state backend.
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator.initKeyedState(Abstr
actStreamOperator.java:687)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator.initializeState(Abst
ractStreamOperator.java:275)
    at org.apache.flink.streaming.runtime.tasks.StreamTask.initializeOperators(StreamTask.j
ava:870)
    at org.apache.flink.streaming.runtime.tasks.StreamTask.initializeState(StreamTask.java:
856)
    at org.apache.flink.streaming.runtime.tasks.StreamTask.invoke(StreamTask.java:292)
    at org.apache.flink.runtime.taskmanager.Task.run(Task.java:762)
    at java.lang.Thread.run(Thread.java:834)
Caused by: org.apache.flink.api.common.typeutils.SerializationException: Cannot serialize/d
eserialize the object.
    at com.alibaba.blink.contrib.streaming.state.AbstractRocksDBRawSecondaryState.deseriali
zeStateEntry(AbstractRocksDBRawSecondaryState.java:167)
    at com.alibaba.blink.contrib.streaming.state.RocksDBIncrementalRestoreOperation.restore
RawStateData(RocksDBIncrementalRestoreOperation.java:425)
    at com.alibaba.blink.contrib.streaming.state.RocksDBIncrementalRestoreOperation.restore
(RocksDBIncrementalRestoreOperation.java:119)
    at com.alibaba.blink.contrib.streaming.state.RocksDBKeyedStateBackend.restore(RocksDBKe
yedStateBackend.java:216)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator.createKeyedStateBack
end(AbstractStreamOperator.java:986)
    at org.apache.flink.streaming.api.operators.AbstractStreamOperator.initKeyedState(Abstr
actStreamOperator.java:675)
    ... 6 more
Caused by: java.io.EOFException
    at java.io.DataInputStream.readUnsignedByte(DataInputStream.java:290)
    at org.apache.flink.types.StringValue.readString(StringValue.java:770)
    at org.apache.flink.api.common.typeutils.base.StringSerializer.deserialize(StringSerial
izer.java:69)
    at org.apache.flink.api.common.typeutils.base.StringSerializer.deserialize(StringSerial
izer.java:28)
    at org.apache.flink.api.java.typeutils.runtime.RowSerializer.deserialize(RowSerializer.
java:169)
    at org.apache.flink.api.java.typeutils.runtime.RowSerializer.deserialize(RowSerializer.
java:38)
    at com.alibaba.blink.contrib.streaming.state.AbstractRocksDBRawSecondaryState.deseriali
zeStateEntry(AbstractRocksDBRawSecondaryState.java:162)
    ... 11 more
```

6.4. Performance optimization by using auto scaling

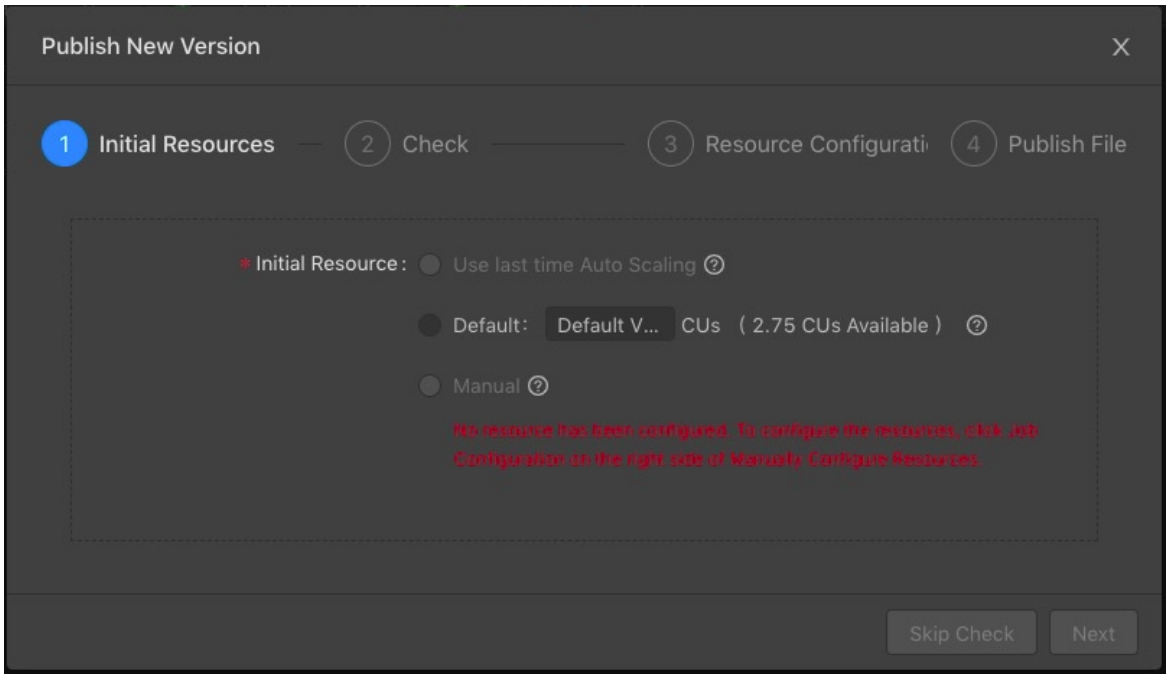
Realtime Compute for Apache Flink earlier than V3.0.0 provides AutoConf to improve job performance. However, AutoConf requires you to frequently restart the job. Realtime Compute for Apache Flink V3.0.0 and later versions support auto scaling to resolve this issue. After you start a job, Realtime Compute for Apache Flink adjusts the job configuration to reach the preset performance goal based on resource configuration rules. This process does not require any manual operations.

> **Note**
>
> - Auto scaling is supported only in Realtime Compute for Apache Flink V3.0.0 and later.
> - Before you upgrade Realtime Compute for Apache Flink to V3.0.0, delete all PlanJSON files generated in Realtime Compute for Apache Flink earlier than V3.0.0 and reacquire configuration files.

Enable auto scaling

You can enable auto scaling when you publish a job.

1. Go to the job editing page on the Realtime Compute for Apache Flink development platform.
2. In the upper part of the job editing section, click **Publish** to go to the **Publish New Version** page.
3. In the **Initial Resources** step, select a resource type and click **Next**.



   - **Use last time Auto Scaling**: uses the PlanJSON file for the latest auto scaling to start the job. You can select **Use last time Auto Scaling** when the following conditions are met:
     - The job is published with auto scaling enabled and uses the latest configuration.
     - The job is in the **Suspended** state.

- The auto scaling configuration is obtained. To obtain the auto scaling configuration, click **Configurations** on the right, move the pointer to Configurations in the upper-right corner, and select **Acquire Auto Scaling Configuration**.

  - **Default**: uses the default resource configuration to start the job. You can select this option to publish a new job or an existing job whose logic is not modified and compatible with Realtime Compute for Apache Flink.
  - **Manual**: uses manually configured resource configuration to start the job. Select this option to manually configure resources or modify the **auto scaling configuration**.

4. In the **Check** step, check the job and click **Next**.
5. In the **Resource Configuration** step, configure **auto scaling** parameters and click **Next**.

Parameter	Description
Automatic Scaling	Specifies whether to enable auto scaling. Select **ON**.
Maximum number of PlanJson CUs	The maximum available CUs for the job. One CU consists of 1 CPU core and 4 GB of memory. The value of this parameter must be less than the number of available CUs in the project.

Parameter	Description
Optimization Policy	The policy for optimizing the job configuration. Valid value: **Data Pending Time**. Realtime Compute for Apache Flink optimizes the job configuration based on **Optimization Policy** and **Expected Value**.
Expected Value	The threshold of the data pending time, in seconds. If data from the data source is pending for a period of time that exceeds the threshold, Realtime Compute for Apache Flink triggers auto scaling to adjust the parallelism for the job.

> **Note** For example, set Expected Value to 5. If data from the data source is pending for more than 5 seconds, Realtime Compute for Apache Flink keeps reducing the parallelism for the job until the pending time is shortened to less than 5 seconds.

6. In the **Publish File** step, click **Publish**.
7. Start the job. For more information about this issue, see Start a job.

Disable auto scaling

> **Note** You can disable auto scaling for a job only if auto scaling is enabled for the job when the job is published.

You can disable auto scaling for a job in the Running state.

1. Go to the **Job Administration** page on the Realtime Compute for Apache Flink development platform.
2. Click **Disable Auto Scaling** in the upper-right corner.
3. Click **Confirm**.

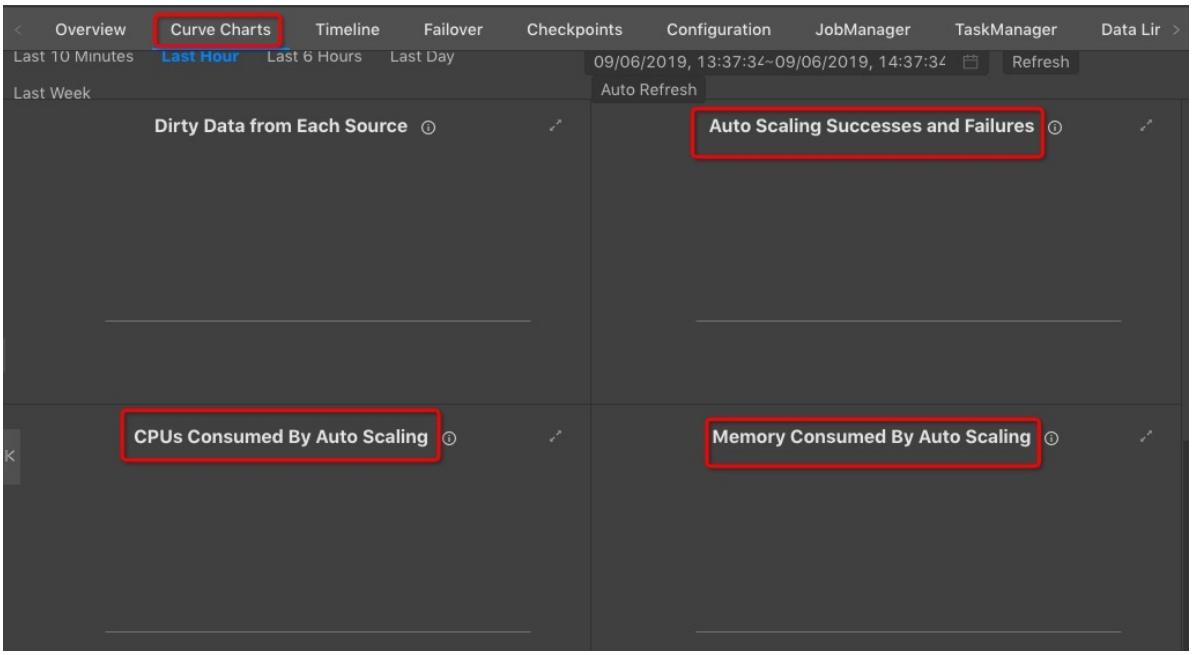
> **Note** The actions in the **Auto Configuration** column on the **Job Administration** page are available only if auto scaling is enabled when you publish the job.

View information about auto scaling

> **Note** Go to the **Job Administration** page. To go to the **Job Administration** page, perform the following steps:

- AutoScale Metric

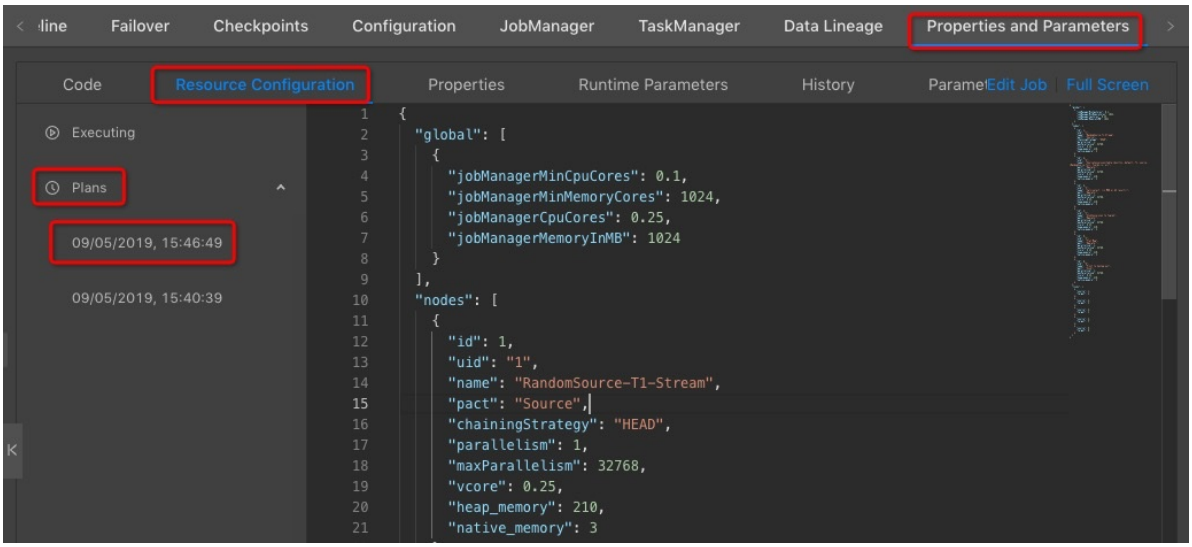
Navigate to **Curve Charts > Overview** to view information about auto scaling.



Metric	Description
Auto Scaling Successes and Failures	The number of successful and failed auto scaling operations
CPUs Consumed By Auto Scaling	The CPU resources used for auto scaling.
Memory Consumed By Auto Scaling	The memory resources used for auto scaling.

- Information about the PlanJSON file generated for auto scaling

  On the Job Administration page, navigate to **Properties and Parameters > Resource Configuration > Plans**. Select the required version to view information about the PlanJSON file that is generated for auto scaling.



FAQ

- Q: What do I do if auto scaling cannot be triggered?

  A: To troubleshoot this issue, perform the following steps:

  - Check whether the job frequently fails. To trigger auto scaling, make sure that the job is in correct logic and runs in a stable manner.
  - View **JobManager** logs to check whether system exceptions occur.

- Q: What do I do if auto scaling does not take effect?

  A: To troubleshoot this issue, perform the following steps:

  i. Check whether the resources consumed by the job reach the upper limit.
  ii. Check the source node logic to determine whether excessive operators are connected to the source node. If excessive operators are connected to the source node, edit the PlanJSON file to remove some operators. For more information, see Optimize performance by manual configuration.
  iii. View **JobManager** logs to check whether system exceptions occur.

- Q: What issues might I run into if I enable auto scaling?

  A:

  - The job is automatically restarted.

    If you enable auto scaling, Realtime Compute for Apache Flink adjusts the parallelism and resources based on the number of data streams. The job may automatically restart the job to adjust resources when the data streams increase or decrease.

  - Data transmission is delayed within a short period of time.

    When the data streams enter the off-peak period, Realtime Compute for Apache Flink triggers auto scaling to reduce the parallelism and resources. When the data streams increase, the resources may become insufficient for data transmission, which causes delay within a short period of time.

  - The job cannot be resumed.

    In some scenarios, auto scaling does not work, and the job may be delayed. Realtime Compute for Apache Flink has to frequently adjust the job configuration. As a result, the job cannot be resumed.

  - The parallelism is reduced and then increased.

    For a job that uses window functions or aggregate functions, when the state data increases, the performance for accessing the state data degrades, and the parallelism is reduced when the job is started. When the job is running, the parallelism increases as the state data accumulates until the amount of the state data becomes steady.

6.5. Optimize performance by manual configuration

You can optimize the performance of a Realtime Compute for Apache Flink job by adjusting the settings of job, resource, and upstream and downstream storage parameters.

Overview

You can configure the following types of parameters to optimize job performance:

- Upstream and downstream storage parameters
- Job parameters, such as miniBatch
- Resource parameters, such as parallelism, core, and heap_memory

This topic describes how to configure the preceding three types of parameters. After you reconfigure parameters for a job, you must terminate and then start the job, or suspend and then resume the job to apply new settings. For more information, see Apply new configurations.

Upstream and downstream storage parameters

In Realtime Compute for Apache Flink, each data record can trigger read and write operations on the source and result tables. This affects the performance of upstream and downstream storage resources. To address this performance issue, you can configure batch size parameters to specify the number of data records that can be read from a source table or written to a result table at a time. The following table describes the source and result tables that support batch size parameters.

Table	Parameter	Description	Value
DataHub source table	batchReadSize	The maximum number of data records that can be read at a time.	Optional. Default value: 10.
DataHub result table	batchSize	The maximum number of data records that can be written at a time.	Optional. Default value: 300.
Log Service source table	batchGetSize	The maximum number of log items that can be read from a log group at a time.	Optional. Default value: 100.
AnalyticDB for MySQL V2.0 result table	batchSize	The maximum number of data records that can be written at a time.	Optional. Default value: 1000.
ApsaraDB RDS result table	batchSize	The maximum number of data records that can be written at a time.	Optional. Default value: 4096.

Note To configure the batch data read and write feature, you can add the preceding parameters to the WITH clause in a DDL statement for a storage system. For example, add `batchReadSize='<number>'` to the WITH clause.

Job parameters

The miniBatch parameter can be used only to optimize the GROUP BY operator. If you use Flink SQL to process streaming data, Realtime Compute for Apache Flink reads state data each time a data record arrives. This consumes a large number of I/O resources. If you configure the miniBatch parameter, Realtime Compute for Apache Flink uses the same key to read the state data only once for the data records and generates only the latest data record. This reduces the frequency to read the state data and minimizes downstream data updates. You can configure the miniBatch parameter based on the following rules:

- After you add parameters for a job, terminate and then start the job to apply the new settings.

- After you change parameter settings for a job, suspend and then resume the job to apply the new settings.

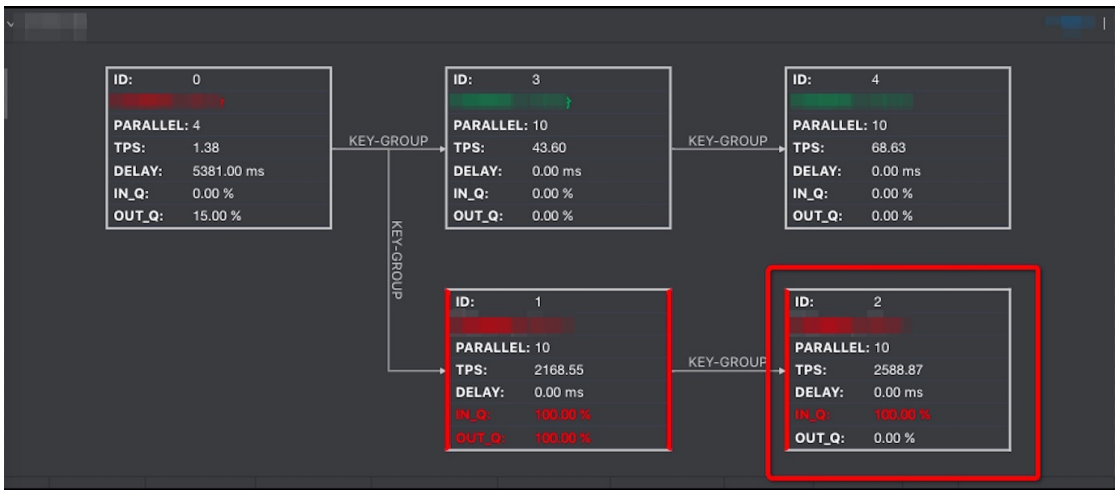
```
# Enable window miniBatch in Realtime Compute for Apache Flink V3.2 and later. By default,
window miniBatch is disabled in Realtime Compute for Apache Flink V3.2 and later.
sql.exec.mini-batch.window.enabled=true
# Use the exactly-once semantics.
blink.checkpoint.mode=EXACTLY_ONCE
# Specify the checkpoint interval. Unit: milliseconds.
blink.checkpoint.interval.ms=180000
blink.checkpoint.timeout.ms=600000
# Use Niagara as the state backend to configure time-to-live (TTL) for the state backend in
Realtime Compute for Apache Flink V2.0 and later. The unit of TTL is milliseconds.
state.backend.type=niagara
state.backend.niagara.ttl.ms=129600000
# In Realtime Compute for Apache Flink V2.0 and later, enable micro-batch processing that i
s performed at an interval of 5 seconds. You cannot configure this parameter when you use a
window function.
blink.microBatch.allowLatencyMs=5000
# Specify the latency that is allowed for a job.
blink.miniBatch.allowLatencyMs=5000
# Enable miniBatch for the node that joins two streams.
blink.miniBatch.join.enabled=true
# Specify the size of a batch of data.
blink.miniBatch.size=20000
# Enable local aggregation. By default, this feature is enabled in Realtime Compute for Apa
che Flink V2.0 and later. If you use Realtime Compute for Apache Flink V1.6.4, you must man
ually enable this feature.
blink.localAgg.enabled=true
# Enable partial aggregation to improve efficiency when you run the CountDistinct function
in Realtime Compute for Apache Flink V2.0 and later.
blink.partialAgg.enabled=true
# Enable UNION ALL for optimization.
blink.forbid.unionall.as.breakpoint.in.subsection.optimization=true
# Configure garbage collection for optimization. You cannot configure this parameter when y
ou use a Log Service source table.
blink.job.option=-yD heartbeat.timeout=180000 -yD env.java.opts='-verbose:gc -XX:NewRatio=3
-XX:+PrintGCDetails -XX:+PrintGCDateStamps -XX:ParallelGCThreads=4'
# Specify the time zone.
blink.job.timeZone=Asia/Shanghai
```

Resource parameters

To optimize resource configurations, perform the following steps:

1. Issue analysis

i. In the following topology, the percentage of the input queues at Task Node 2 reaches 100%. The data at Task Node 2 is stacked up and causes backpressure on Task Node 1. At Task Node 1, the percentage of the output queues has reached 100%.



ii. Click Task Node 2.

iii. In the Vertex Topology section of the Overview tab, click the **SubTask List** tab. Then, find the subtask in which the value of **In Queue** is `100%`.

iv. Click **LOG 0** in the ID column in the row of the subtask.

v. On the **Metrics Graph** tab, view the CPU and memory usage.

2. Performance

i. On the right side of the job editing page, click the **Configurations** tab to view the details about resource configurations.

ii. On the page that appears, change the parameter values of one or more operators in a group.

- To change the parameter values of one operator, perform the following steps:
  a. In the **GROUP** box, click the plus sign (+) in the upper-right corner.
  b. Move the pointer over the Operator box.
  c. Click the  icon next to the operator name.
  d. In the **Modify Operator Data** dialog box, change the parameter values and click OK.
- To change the parameter values of multiple operators in a group at a time, perform the following steps:
  a. Move the pointer over the **GROUP** box.
  b. Click the  icon next to **GROUP**.
  c. In the **Modify Operator Data** dialog box, change the parameter values based on your business requirements and click OK.

iii. In the upper-right corner of the **Configurations** page, choose **Configurations > Apply**.

- If the job performance is not significantly improved after you change the values of the resource parameters for the group, perform the following steps to troubleshoot the issue:
  a. Check whether data skew exists on the operator.
  b. Check whether subtasks of complex operators, such as GROUP BY, WINDOW, and JOIN, are running as expected.
- To remove an operator from a chain, perform the following steps:
  a. Click the operator that you want to remove.
  b. In the Modify Operator Data dialog box, set chainingStrategy to `HEAD`.

  If the chainingStrategy parameter of this operator is set to `HEAD`, you must also set the chainingStrategy parameter to `HEAD` for the next operator. The following table describes the valid values of the chainingStrategy parameter.

Parameter	Description
ALWAYS	Operators are combined to increase the parallelism and optimize job performance.
NEVER	Operators are not combined with the upstream and downstream operators.
HEAD	Operators are combined with only the downstream operators.

3. Rules and suggestions
   - We recommend that you set `core:heap_memory` to 1:4. This indicates that each CPU core is assigned 4 GB of memory. Examples:
     - If the core parameter of operators is set to 1 and the heap_memory parameter of the operator is set to 3, the system assigns 1 compute unit (CU) and 4 GB of memory to the chain.
     - If the core parameter of operators is set to 1 and the heap_memory parameter of operators is set to 5, the system assigns 1.25 CUs and 5 GB of memory to the chain.

     > **Note**
     > - The total number of cores for an operator is calculated by using the following formula: Value of the parallelism parameter × Value of the core parameter.
     > - The total heap_memory size for an operator is calculated by using the following formula: Value of the parallelism parameter × Value of the heap_memory parameter.
     > - The core value for a chain is the maximum core value among the operators in the group. The heap_memory size for a chain is the total heap_memory size of all the operators in the chain.

   - **parallelism**

- Source node

  The number of source nodes is a multiple of the number of upstream partitions. For example, if the number of source nodes is 16, you must set the parallelism parameter to a divisor of 16, such as 16, 8, or 4. The divisor must exclude 16.

  > **Note** The value of the parallelism parameter for the source nodes cannot exceed the number of shards for the source nodes.

- Operator node

  Specify the parallelism parameter of the operator nodes based on the estimated queries per second (QPS).

  - If the QPS is low, you can set the number of operator nodes to the value that is the same as the parallelism of the source nodes.
  - If the QPS is high, make sure that the number of operator nodes is greater than the parallelism of the source nodes. For example, if the parallelism is 16, set the number of operator nodes to a value that is greater than 16, such as 64, 128, or 256.

- Sink node

  Set the parallelism parameter of the sink nodes to a value that is two to three times the number of downstream partitions.

  > **Note** Do not set the parallelism parameter of the sink nodes to a value that is greater than three times the number of downstream partitions. Otherwise, write timeout or failures may occur. For example, if the number of sink nodes is 16, do not set the parallelism parameter of these sink nodes to a value that is greater than 48.

- **core**

  This parameter specifies the number of CPU cores. You can specify this parameter based on the actual CPU utilization. The recommended value of this parameter is 0.25. The default value is 0.1.

- **heap_memory**

  The heap memory size. Unit: MB. You can configure this parameter based on the actual memory usage. The default value is 256.

- **state_size**

  You must set the **state_size** parameter to `1` for task nodes where the GROUP BY, undefinedJOIN, OVER, or WINDOW operators are used. This way, the system assigns extra memory for the operator to access state data. The default value of the **state_size** parameter is 0.

  > **Note** If you do not set **state_size** to `1`, the job may fail.

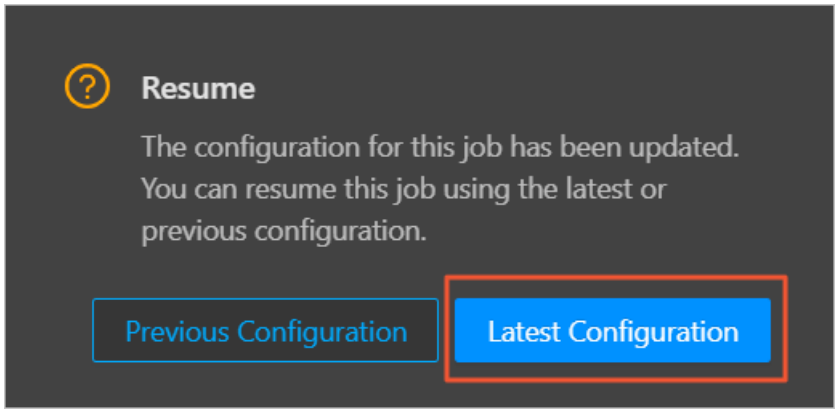
Apply new configurations

After you configure the parameters, we recommended that you suspend and then resume the job, but not terminate and then start the job. This ensures that the configurations take effect. The job status is cleared when the job is terminated. This may change execution results.

> **Note**
> - You can suspend and then resume a job after you change the values of the resource parameters, parameters in the WITH clause, or job parameters.
> - You can terminate and then start a job after you modify the SQL logic, change the job version, add parameters to the WITH clause, or add job parameters.

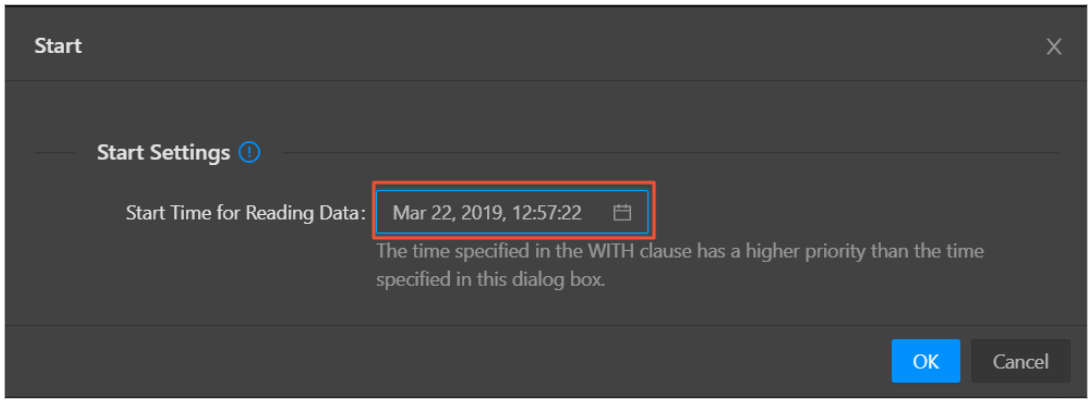
After you restart or resume the job, you can click the **Overview** tab on the **Administration** page and click the **Vertex Topology** tab to check whether the new configurations take effect.

- To suspend and resume a job, perform the following steps:
  i. Publish a job. For more information, see Publish a job. Set **Resource Configuration Method** to **Use Latest Manually Configured Resources**.
  ii. On the **Administration** page, find the job that you want to suspend and click **Suspend** in the Actions column.
  iii. On the **Administration** page, find the job that you want to resume and click **Resume** in the Actions column.
  iv. In the **Resume** dialog box, click **Latest Configuration**.



- To terminate and then start a job, perform the following steps:
  i. Terminate a job.
     a. Log on to the .
     b. In the top navigation bar, click **Administration**.
     c. On the **Administration** page, find the job that you want to terminate, and click **Terminate** in the **Actions** column.
  ii. Start the job.
     a. Log on to the .
     b. In the top navigation bar, click **Administration**.
     c. On the **Administration** page, find the job that you want to start, and click **Start** in the **Actions** column.

d. In the **Start** dialog box, specify **Start Time for Reading Data**.



e. Click **OK**. The job is started.

> **Note** Start Time for Reading Data specifies the time when Realtime Compute for Apache Flink starts to read data from the source table.
>
> - If you select the current time, Realtime Compute for Apache Flink reads data that is generated after the current time.
> - If you select a previous time, Realtime Compute for Apache Flink reads data that is generated from the selected time. This is used to trace historical data.

Parameters

- Global

  isChainingEnabled specifies whether chaining is enabled. The default value is true. Use the default value for this parameter.

- Nodes

Parameter	Description	Allow modification
id	The unique ID of the node. The node ID is generated by the system.	No
uid	The unique user identifier (UID) of the node. The UID is used to generate the operator ID. If you do not specify this parameter, the UID is the same as the node ID.	No
pact	The node type, such as data source, operator, or data sink.	No
name	The name of the node. You can customize this parameter.	Yes
slotSharingGroup	Specifies whether subtasks can share the same slot. Use the default value for this parameter.	No

Parameter	Description	Allow modification
chainingStrategy	Defines the operator chaining strategy. If an operator is combined with an upstream operator, they run in the same thread. They are combined into an operator chain that has multiple running steps. Valid values: ○ ALWAYS: Operators are combined to increase the parallelism and optimize job performance. ○ NEVER: Operators are not combined with the related upstream or downstream operators. ○ HEAD: Operators are combined with only the downstream operators.	Yes
parallelism	The number of parallel jobs on the node. You can increase the value based on your business requirements. Default value: 1.	Yes
core	The number of CPU cores. You can specify this parameter based on the actual CPU utilization. Default value: 0.1. Recommended value: 0.25.	Yes
heap_memory	The heap memory size. You can specify this parameter based on the memory size that needs to be used. Default value: 256 MB.	Yes
direct_memory	The non-heap memory of a Java Virtual Machine (JVM). Unit: MB. Default value: 0.	You can change the value of this parameter, but we recommend that you use the default value.
native_memory	The JVM non-heap memory that is used for the Java Native Interface (JNI). Default value: 0. You can set this parameter to 10 based on your business requirements. This memory is mainly used for state backends.	You can change the value of this parameter, but we recommend that you use the default value.

- Chain

  A Flink SQL task is a directed acyclic graph (DAG) that contains multiple nodes or operators. Some upstream and downstream operators can be combined into a new operator when the operators run in the same thread. This process is known as a chain. As a result, the total number of CPU cores for the new operator is the maximum number of CPU cores among all the operators in the chain. The memory size for the chain equals the total memory size of all the operators in the chain. An operator chain can significantly reduce data transmission costs.

  > **Note**
  > - Only operators that have the same parallelism value can be combined to form a chain.
  > - You cannot add a GROUP BY operator to a chain.

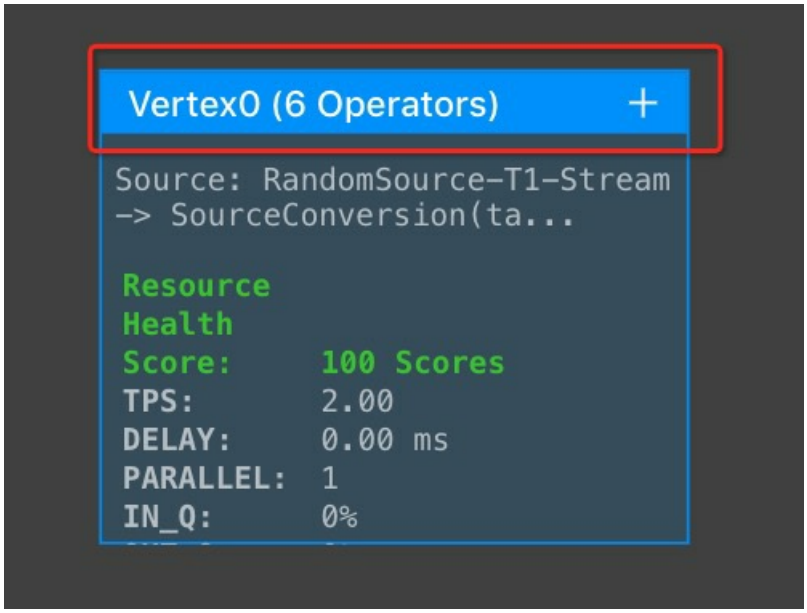
6.6. Typical backpressure scenarios and optimization ideas

Backpressure is an important concept in streaming shuffle. If the processing capability of downstream storage systems is insufficient, Realtime Compute notifies upstream storage systems to stop sending data to avoid data loss. In this scenario, backpressure occurs. This topic describes typical backpressure scenarios and optimization ideas.

Backpressure detection mechanism

A job backpressure detection mechanism is provided in Realtime Compute versions later than V3.0.0. With this mechanism, Realtime Compute detects congestion in the output network buffer of a vertex to determine whether backpressure exists in the vertex. A vertex is a group of operators associated as a chain. To check the backpressure for a job, follow these steps:

1. In the top navigation bar, click **Administration**.
2. In the left-side navigation pane, click the running job for which you want to check the backpressure. In the **Vertex Topology** section of the **Overview** tab that appears, click the blue border of the vertex that you want to check for the job.



3. In the right-side pane, click the **BackPressure** tab and view the backpressure status in the **Status** column.
   - If **high** with a red indicator is displayed, the vertex has backpressure.
   - If **ok** with a green indicator is displayed, the vertex does not have backpressure.

Backpressure scenarios and optimization ideas

> **Note** In the following vertex topology diagrams, the vertex in green indicates that no backpressure is detected, whereas the vertex in red indicates that backpressure is detected.

- **Scenario 1: Only one vertex exists and no backpressure is detected.**

Due to Flink features, no network buffer is configured on the output of the last vertex. In this case, data is directly written into downstream storage systems. If a job has only one or the last vertex, the backpressure detection fails. Therefore, this vertex topology diagram does not indicate that no backpressure is detected in the job. To further determine if and where backpressure exists, you must split the operators in Vertex 0. For more information about how to split the operators, see Resource parameters.

- **Scenario 2: Multiple vertices exist and backpressure is detected on the second to last vertex.**

This vertex topology diagram shows that Vertex 1 has backpressure and Vertex 2 has a performance bottleneck. You can check the operator names in Vertex 2 to determine the actions that you can take.

  - If only write operations into downstream storage systems are involved, the backpressure may be caused by the slow writing speed. We recommend that you increase the parallelism for Vertex 2 or set the `batchsize` parameter for the result table. For more information, see Upstream and downstream storage parameters.
  - If operations in addition to the write operation into downstream storage systems are involved, you must split the operators that correspond to those operations for further check. For more information about how to split the operators, see Resource parameters.

- **Scenario 3: Multiple vertices exist and backpressure is detected on a vertex other than the second to last vertex.**

This vertex topology diagram shows that Vertex 0 has backpressure and Vertex 1 has a performance bottleneck. You can check the operator names in Vertex 1 to determine the actions that you can take. The common operations and related optimization methods used in this scenario are as follows:

- GROUP BY operation: You can increase the parallelism or set the `miniBatch` parameter to optimize the state operation. For more information, see Job parameters.
- JOIN operation between dimension tables: You can increase the parallelism or set a cache policy for dimension tables. For more information, see relevant dimension table documents.
- User-defined extension (UDX) operation: You can increase the parallelism or optimize the related UDX code.

- **Scenario 4: Multiple vertices exist and no backpressure is detected on all the vertices.**

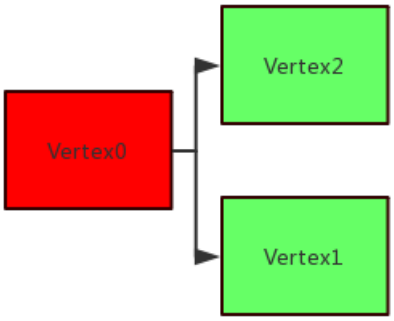
This vertex topology diagram shows that Vertex 0 has a potential performance bottleneck. You can check the operator names in Vertex 0 to determine the actions that you can take.

- If only read operations from the source table are involved, the slow reading speed causes high latency. However, Realtime Compute does not have performance bottlenecks. In this case, you can increase the parallelism of the source operator or set the `batchsize` parameter for reading the source data. For more information, see Upstream and downstream storage parameters.

  > **Note** The parallelism of the source operator cannot be greater than the number of shards of the upstream storage systems.

- If operations in addition to the read operation from the source table are involved, we recommend that you split the operators involved in other operations first. For more information about how to split operators, see Resource parameters.

- **Scenario 5: Backpressure is detected on a vertex but no backpressure is detected on its subsequent parallel vertices.**



This vertex topology diagram shows that Vertex 0 has backpressure but whether Vertex 1 or Vertex 2 has a performance bottleneck cannot be determined. You can preliminarily determine the vertex where a performance bottleneck exists based on the **IN_Q** metric of Vertex 1 and Vertex 2. The vertex whose **IN_Q** remains 100% for a long period of time may have a performance bottleneck. To further determine where the performance bottleneck exists, you must split the operators of the vertex. For more information about how to split operators, see Resource parameters.

6.7. SQL Tuning Advisor

6.7.1. Partitioned All Cache

If you use the Cache All policy to join a super-large dimension table with another table, processes may fail to load full data of the dimension table into the cache. In this case, you can use the Partitioned All Cache policy to optimize the loading performance.

Background information

When you join a dimension table with another table, you can set the cache parameter to ALL to use the Cache All policy. This policy requires that all processes load full data of the dimension table to the cache. The memory size configured for the join node must be at least twice that of the dimension table.

If the dimension table is large, the join node consumes a large amount of memory. This also increases the garbage collection overhead. If the size of a dimension table exceeds 1 TB, only partial data of the table can be loaded to the memory. Optimization by using Partitioned All Cache is introduced to resolve this issue. If Partitioned All Cache is enabled, input data is shuffled based on join keys, and each process needs to load only the required data of the dimension table to the cache.

You can set partitionedJoin to true to enable Partitioned All Cache. This reduces memory consumption. However, input data shuffling based on join keys causes additional network and CPU overheads. We recommend that you do not enable Partitioned All Cache in the following scenarios:

- Input data is severely skewed on the join keys. If you use Partitioned All Cache in this scenario, data skew slows down the running of some nodes.
- The size of the dimension table is small. For example, the size of the dimension table is less than 2 GB.

If you enable Partitioned All Cache in this scenario, the memory consumption is slightly reduced, whereas high network and CPU overheads are generated.

Optimization method

Add partitionedJoin = 'true' to the WITH clause of the DDL statement of the dimension table.

Sample code

```
 CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id),
  PERIOD FOR SYSTEM_TIME -- The identifier of a dimension table.
) with (
  type = 'odps',
  endPoint = 'your_end_point_name',
  project = 'your_project_name',
  tableName = 'your_table_name',
  accessId = 'your_access_id',
  accessKey = 'your_access_key',
  `partition` = 'ds=20180905',
  cache = 'ALL',
  partitionedJoin = 'true' -- Enable Partitioned All Cache.
);
```

6.7.2. miniBatch and microBatch

You can increase the throughput by enabling miniBatch or microBatch.

Background information

Both miniBatch and microBatch are used for micro-batch processing. If you enable miniBatch or microBatch, data processing is triggered when the data in the cache reaches a specified threshold. This reduces the frequency at which Realtime Compute for Apache Flink accesses the state data. This way, the throughput is increased and data output is reduced. However, the two methods have different triggering mechanisms:

- miniBatch triggers micro-batch processing by using the timer threads that are registered with each task. This requires some thread scheduling overheads.
- microBatch triggers micro-batch processing by using event messages, which are inserted into the data sources at a specified interval. microBatch is an enhancement of miniBatch. microBatch has higher data serialization efficiency, higher throughput, lower backpressure, and lower latency than miniBatch.

We recommend that you enable micro-batch processing in most scenarios, such as data aggregation, to improve system performance. We recommend that you do not enable micro-batch processing in the following scenarios:

- Low latency is required. Micro-batch processing achieves high throughput at the expense of latency.
- The aggregation degree of GroupAggregate is low (O/I > 0.8). In such scenarios, almost no data is aggregated in a batch.

Optimization method

On the **Parameters** tab of the **Development** page, set blink.microBatch.allowLatencyMs or blink.miniBatch.allowLatencyMs. The two parameters have the same effect. The unit of them is millisecond.

> **Note** If you set both parameters for a job, we recommend that you set them to the same value. If they are set to different values, the second one in the code takes effect.

Sample code

- Enable microBatch.

```
blink.microBatch.allowLatencyMs=5000
```

- Enable miniBatch.

```
blink.miniBatch.size=20000
```

- Enable microBatch and miniBatch at the same time.

```
# The maximum number of data records that can be cached in each batch. You must set this parameter to avoid out of memory (OOM).
blink.miniBatch.size=20000
blink.microBatch.allowLatencyMs=5000
blink.miniBatch.allowLatencyMs=6000--- This configuration takes effect.
```

6.7.3. Cache policy

When you join two dimension tables, you can configure a cache policy to improve the job throughput.

Background information

In Realtime Compute for Apache Flink, you can set the cache parameter to specify a cache policy:

- If the cache parameter is set to LRU, some data in the dimension table is cached. The system creates a local LRU cache for each join node. Realtime Compute for Apache Flink searches for data in the cache each time it reads a data record in the source table. The data that meets the requirement is returned. This reduces I/O requests. If no data in the cache meets the requirement, Realtime Compute for Apache Flink searches the dimension table. The data that meets the requirement is stored in the cache for subsequent queries.

  To limit the volume of data that can be stored in the cache, you can set cacheSize. To regularly update the data of a dimension table, you can set cacheTTLMs to adjust the cache expiration time. cacheTTLMs takes effect for all cached data records. If a cached data record is not accessed within the specified period of time, it is removed from the cache.

- If the cache parameter is set to All, all data in the dimension table is cached. The system creates an asynchronous thread for the join node to synchronize data between the cache and dimension table. The input data is blocked from the moment a job is started to the moment loading data to the cache is completed. This ensures that the data in the dimension table is loaded to the cache before input data processing starts.

Realtime Compute for Apache Flink searches the cache in subsequent dimension table queries. If the data that meet the requirement cannot be found in the cache, the join key does not exist. If data in the cache expires, Realtime Compute for Apache Flink reloads the data in the dimension table to the cache. The reloading process does not affect the join operation of dimension tables. The reloaded data is stored in the temporary memory. The atomic substitution operation is performed after all data in the dimension table is reloaded.

If cache is set to ALL, the join operation of dimension tables can achieve excellent performance because few I/O requests are initiated. However, the memory must be large enough to store the data of two dimension tables.

> **Note** Do not specify the cache policy if data in the cache is not allowed for each dimension table query.

Optimization method

Add cache='LRU' or cache='ALL' to the WITH clause in the DDL statement of the dimension table. The following table describes the parameters related to the cache policy.

Parameter	Description	Required	Remarks
cache	The cache policy.	No	• *None*: indicates that data is not cached. This is the default value. • *LRU*: If you set this parameter to LRU, you must configure cacheSize, cacheTTLMs, and partitionedJoin. • *ALL*: If you set this parameter to ALL, you must configure cacheTTLMs, cacheReloadTimeBlackList, and partitionedJoin.
cacheSize	The cache size. Unit: rows.	No	You can set this parameter only after you set the cache parameter to *LRU*. Default value: *10000*.
cacheTTLMs	The cache expiration period or the cache reloading interval. Unit: milliseconds.	No	• If you set the cache parameter to *LRU*, this parameter specifies the cache expiration period. The cache does not expire by default. • If you set the cache parameter to *ALL*, this parameter specifies the cache reloading interval. The cache is not reloaded by default.

Parameter	Description	Required	Remarks
cacheReloadTimeBlackList	The periods during which cache reloading is not allowed. This parameter takes effect if the cache parameter is set to ALL. During the periods specified by this parameter, the cache is not reloaded (for example, in Double 11).	No	Optional. This parameter is empty by default. Example: '2017-10-24 14:00 -> 2017-10-24 15:00, 2017-11-10 23:30 -> 2017-11-11 08:00'. • Separate multiple periods with commas (,). • Separate the start time and end time of each period with a hyphen and a greater-than sign (->).
partitionedJoin	Specifies whether to enable Partitioned All Cache.	No	By default, this parameter is set to false, which indicates that Partitioned All Cache is disabled. If Partitioned All Cache is enabled, data is shuffled before the source table is associated with the dimension table based on join keys. • If the cache parameter is set to *LRU*, the cache hit rate increases. • If the cache parameter is set to *ALL*, memory consumption is reduced because only the required data is cached for each concurrent job.

Sample code

```
 CREATE TABLE white_list (
  id varchar,
  name varchar,
  age int,
  PRIMARY KEY (id)
) with (
  type = 'odps',
  endPoint = 'your_end_point_name',
  project = 'your_project_name',
  tableName = 'your_table_name',
  accessId = 'your_access_id',
  accessKey = 'your_access_key',
  `partition` = 'ds=20180905',
  cache = 'ALL'
);
```

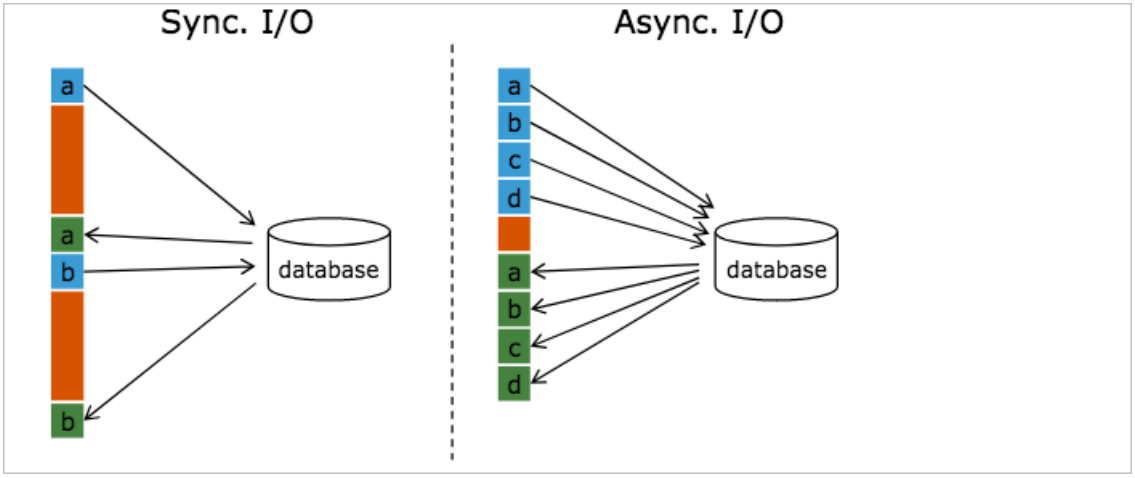
6.7.4. Asynchronous mode

You can enable the asynchronous mode and configure related parameters to improve throughput when you join dimension tables.

Background information

By default, the synchronous mode is used when you join dimension tables. The system queries the physical table and returns the association result each time a data record is added to the physical table. This results in low throughput and high latency. The asynchronous mode is introduced to process query requests in parallel, so that consecutive requests do not need to wait for processing.

Flink SQL implements asynchronous JOIN operations for dimension tables based on Flink Async I/O and asynchronous clients. This significantly improves throughput. The following figure shows the comparison between the synchronous and asynchronous modes.



Optimization method

Add async='true' to the WITH clause in the data definition language (DDL) statement of a dimension table. The following table describes the parameters related to the asynchronous mode.

Parameter	Description	Required	Remarks
async	Specifies whether to enable the asynchronous mode.	No	Default value: false.
asyncResultOrder	Specifies whether to sort asynchronous results.	No	Valid values: • unordered (default value) • ordered
asyncTimeoutMs	The timeout period of an asynchronous request, in milliseconds.	No	Default value: 180000.
asyncCapacity	The maximum number of asynchronous requests in an asynchronous request queue.	No	Default value: 100.

Parameter	Description	Required	Remarks
asyncCallbackThreads	The number of callback threads.	No	An asynchronous request is implemented by a thread. onComplete is called if an asynchronous request succeeds and onError is called if it fails. The number of times onComplete and onError is called determines the size of the thread pool. Default value: 50.
asyncConnectionQueueMaxsize	The maximum number of requests that can be sent.	No	If the number of requests waiting for a server to process reaches the value specified by this parameter, asynchronous request calling is blocked to prevent out of memory of the client. Default value: 100.
asyncCallbackQueueMaxsize	The maximum number of requests in a callback processing queue.	No	If the number of requests waiting for callback processing reaches the value specified by this parameter, asynchronous request calling is blocked to prevent out of memory of the client. Default value: 500.

Sample code

```
CREATE TABLE dim_cn_item(
    rowkey VARCHAR,
    item_id VARCHAR,
    title VARCHAR,
    cate_id VARCHAR,
    cate_name VARCHAR,
    cate_level1_id VARCHAR,
    cate_level2_id VARCHAR,
    cate_level3_id VARCHAR,
    cate_level1_name VARCHAR,
    cate_level2_name VARCHAR,
    cate_level3_name VARCHAR,
    pinlei_id VARCHAR,
    pinlei_name VARCHAR,
    bu_id VARCHAR,
    bu_name VARCHAR,
    PRIMARY KEY(rowkey)
) WITH(
    type='alihbase',
    diamondKey = 'xxxx',
    diamondGroup ='yyyy',
    cacheTTLMs='3600000',
    async='true',
    cache='LRU',
    columnFamily='cf',
    cacheSize='1000',
    tableName='yourTableName'
);
```

FAQ

- Error information

```
Caused by: org.apache.flink.table.api.TableException: Output mode can not be UNORDERED if
the input is an update stream.
at org.apache.flink.table.plan.util.TemporalJoinUtil$.validate(TemporalJoinUtil.scala:340
)
at org.apache.flink.table.plan.nodes.common.CommonTemporalTableJoin.translateToPlanIntern
al(CommonTemporalTableJoin.scala:144)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translat
eToPlanInternal(StreamExecTemporalTableJoin.scala:98)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translat
eToPlanInternal(StreamExecTemporalTableJoin.scala:39)
at org.apache.flink.table.plan.nodes.exec.ExecNode$class.translateToPlan(ExecNode.scala:5
8)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.org$apac
he$flink$table$plan$nodes$exec$StreamExecNode$$super$translateToPlan(StreamExecTemporalTa
bleJoin.scala:39)
at org.apache.flink.table.plan.nodes.exec.StreamExecNode$class.translateToPlan(StreamExec
Node.scala:38)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translat
eToPlan(StreamExecTemporalTableJoin.scala:39)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecTemporalTableJoin.translat
eToPlan(StreamExecTemporalTableJoin.scala:39)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.translateToPlanIntern
al(StreamExecCalc.scala:89)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.translateToPlanIntern
al(StreamExecCalc.scala:43)
at org.apache.flink.table.plan.nodes.exec.ExecNode$class.translateToPlan(ExecNode.scala:5
8)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.org$apache$flink$tabl
e$plan$nodes$exec$StreamExecNode$$super$translateToPlan(StreamExecCalc.scala:43)
at org.apache.flink.table.plan.nodes.exec.StreamExecNode$class.translateToPlan(StreamExec
Node.scala:38)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecCalc.translateToPlan(Strea
mExecCalc.scala:43)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecSink.translate(StreamExecS
ink.scala:158)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecSink.translateToPlanIntern
al(StreamExecSink.scala:103)
at org.apache.flink.table.plan.nodes.physical.stream.StreamExecSink.translateToPlanIntern
al(StreamExecSink.scala:53)
at org.apache.flink.table.plan.nodes.exec.ExecNode$class.translateToPlan(ExecNode.scala:5
8)
at
```

- Cause

  If the upstream data is Update Stream, the asyncResultOrder parameter is set to unordered when you join dimension tables.

  > **Note** Update Stream can be the TopN operation or the JOIN operation on two data streams.

- Solution

  Set the asyncResultOrder parameter to ordered in the WITH clause for the dimension table.

6.7.5. APPROX_COUNT_DISTINCT

You can optimize your job performance by using the APPROX_COUNT_DISTINCT function. Compared with COUNT(DISTINCT), this function returns an approximate count.

Background information

When the COUNT(DISTINCT) function is used, distinct key information is saved in state data of the aggregate node. If a large number of distinct keys exist, the read/write overhead of state data is large. This causes a bottleneck in job performance optimization. In many cases, accurate computation is not necessary. If you are willing to achieve high job performance at the expense of accuracy, you can use the APPROX_COUNT_DISTINCT function. APPROX_COUNT_DISTINCT supports miniBatch and local-global optimization on the aggregate node. When you use this function, make sure that the following requirements are met:

- The input data does not contain retracted messages.
- A large number of distinct keys, such as unique visits (UVs), exist. The APPROX_COUNT_DISTINCT function cannot bring obvious benefits if only a small number of distinct keys exist.

Optimization method

Use APPROX_COUNT_DISTINCT(user) to replace COUNT(DISTINCT user) in the SQL. The syntax of APPROX_COUNT_DISTINCT(user) is:

```
APPROX_COUNT_DISTINCT(col [, accuracy])
```

where:

- col indicates the name of a field, which can be of any type.
- accuracy specifies the calculation accuracy. A larger value indicates higher accuracy, higher state overhead, and lower performance. This field is optional. Valid values: (0.0, 1.0). Default value: 0.99.

Sample code

- Test data

a (VARCHAR)	c (BIGINT)
Hi	1
Hi	2
Hi	3
Hi	4
Hi	5
Hi	6

- Test statement

```
SELECT
  a,
  APPROX_COUNT_DISTINCT(b) as b,
  APPROX_COUNT_DISTINCT(b, 0.9) as c
FROM MyTable
GROUP BY a;
```

- Test results

a (VARCHAR)	b (BIGINT)	c (BIGINT)
Hi	5	5

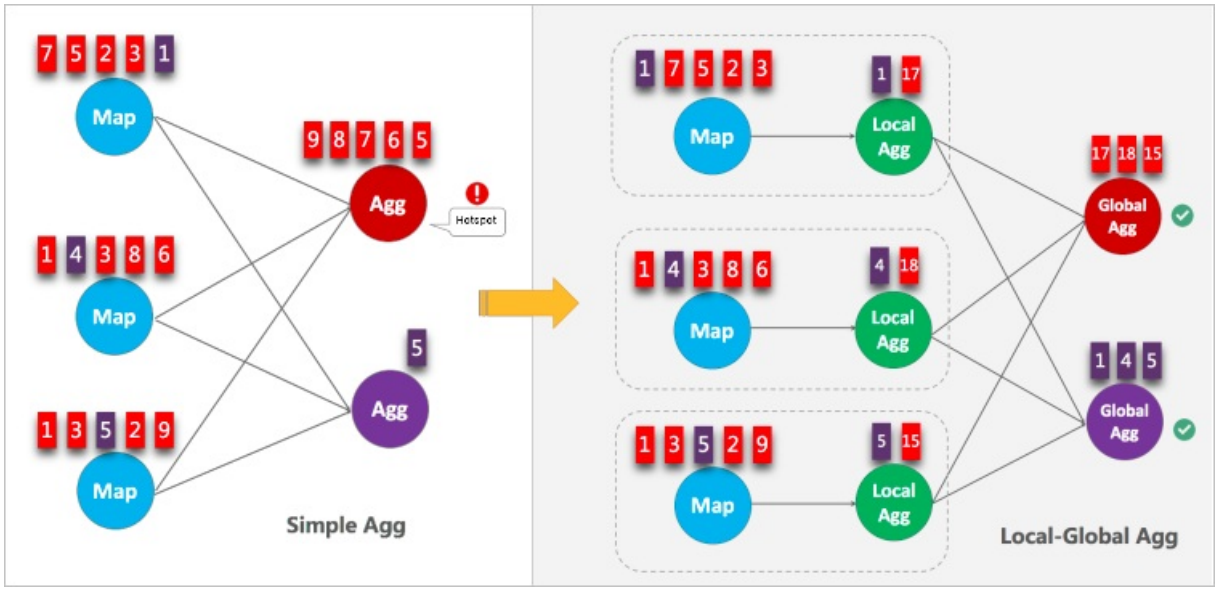
6.7.6. Local-global optimization

You can use local-global optimization to resolve data skew issues of an aggregate node.

Background information

When you use local-global optimization, the aggregation process is divided into two phases: local aggregation and global aggregation. They are similar to the combine and reduce phases in MapReduce. In the local aggregation phase, the system aggregates data that is locally buffered at the input node into batches and generates an accumulator for each batch of data. In the global aggregation phase, the system merges the accumulators to obtain the global aggregation result.

Local-global optimization can eliminate data skews by using local aggregation and resolve the data hotspot issues in global aggregation. This improves job performance. You can use local-global optimization to improve the performance of common aggregate functions, such as SUM, COUNT, MAX, MIN, and AVG. You can also use it to resolve data hotspot issues that occur when you use these functions.



Optimization method

A user-defined aggregate function (UDAF) must execute the merge method to trigger local-global optimization. For more information about how to execute the merge method, see Sample code.

> **Note** In Blink 2.0 and later versions, local-global optimization is enabled by default. If you want to disable local-global optimization, set blink.localAgg.enabled to false in **job parameters**.

Sample code

```
import org.apache.flink.table.functions.AggregateFunction;
public class CountUdaf extends AggregateFunction<Long, CountUdaf.CountAccum> {
    // Define the data structure of the accumulator that stores state data of the COUNT UDA
F.
    public static class CountAccum {
        public long total;
    }
    // Initialize the accumulator of the COUNT UDAF.
    public CountAccum createAccumulator() {
        CountAccum acc = new CountAccum();
        acc.total = 0;
        return acc;
    }
    // The getValue method is used to compute the result of the COUNT UDAF based on the acc
umulator that stores state data.
    public Long getValue(CountAccum accumulator) {
        return accumulator.total;
    }
    // The accumulate method is used to update the accumulator that is used by the COUNT UD
AF to store state data based on input data.
    public void accumulate(CountAccum accumulator, Object iValue) {
        accumulator.total++;
    }
    public void merge(CountAccum accumulator, Iterable<CountAccum> its) {
         for (CountAccum other : its) {
            accumulator.total += other.total;
         }
    }
}
```

6.7.7. ROW_NUMBER OVER WINDOW

You can use the ROW_NUMBER OVER WINDOW function to efficiently deduplicate source data.

Background information

Deduplication aims to obtain top N records. Realtime Compute for Apache Flink supports two deduplication policies:

- Deduplicate Keep FirstRow: retains only the first record under a key. The state data contains only the key information, so the node performance is high after you enable deduplication by using ROW_NUMBER OVER WINDOW.
- Deduplicate Keep LastRow: retains only the last record under a key. This policy slightly outperforms the LAST_VALUE function.

Optimization method

SQL does not have deduplication syntax. Realtime Compute for Apache Flink uses the ROW_NUMBER OVER WINDOW function to deduplicate data.

```
SELECT *
FROM (
   SELECT *,
    ROW_NUMBER() OVER ([PARTITION BY col1[, col2..]
    ORDER BY timeAttributeCol [asc|desc]) AS rownum
   FROM table_name)
WHERE rownum = 1;
```

Parameter	Description
ROW_NUMBER()	Calculates the row number. It is a window function that contains an OVER clause. The value starts from 1.
PARTITION BY col1[, col2..]	Optional. Specifies partition columns for storing primary keys of duplicate records.
ORDER BY timeAttributeCol [asc|desc])	Specifies the column by which you want to sort data. You must specify a time attribute, which can be proctime or rowtime. You must also specify the sort order, which can be asc (Deduplicate Keep FirstRow) or desc (Deduplicate Keep LastRow). **Note** • If you do not specify the time attribute, proctime is used by default. • If you do not specify the sort order, asc is used by default.
rownum	The current row number. Only rownum=1 and rownum<=1 are supported.

When the ROW_NUMBER OVER WINDOW function is executed, two levels of queries are performed:

1. The ROW_NUMBER() function is used to sort data records under a key by the time attribute and mark the records with their rankings.

   Note
   - If the time attribute is proctime, Realtime Compute for Apache Flink removes duplicate records based on the time when the records are processed by Realtime Compute for Apache Flink. In this case, the rankings may vary each time.
   - If the time attribute is rowtime, Realtime Compute for Apache Flink removes duplicate records based on the time when the records are written to Realtime Compute for Apache Flink. In this case, the rankings remain unchanged.

2. The data records are sorted by their rankings and only the first or last one is retained.

Sample code

- Deduplicate Keep FirstRow

  In this example, Realtime Compute for Apache Flink removes duplicate data records in table T based on field b, and retains the first data record that is processed by Realtime Compute for Apache Flink.

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b ORDER BY proctime) as rowNum
  FROM T
)
WHERE rowNum = 1;
```

- Deduplicate Keep LastRow

  In this example, Realtime Compute for Apache Flink removes duplicate data records in table T based on the b and d fields, and retains the last record that is written to Realtime Compute for Apache Flink.

```
SELECT *
FROM (
  SELECT *,
    ROW_NUMBER() OVER (PARTITION BY b, d ORDER BY rowtime DESC) as rowNum
  FROM T
)
WHERE rowNum = 1;
```

6.7.8. Partial-final optimization

You can resolve COUNT(DISTINCT) hotspot issues by using partial-final optimization.

Background information

To resolve the COUNT(DISTINCT) hotspot issues, you can change the aggregation process to two-layer aggregation by adding a scattered layer at which data is scattered based on the distinct keys. This is referred to as partial-final optimization. Blink 2.2.0 and later versions support partial-final optimization. Partial-final optimization is suitable for the following scenarios:

- The COUNT(DISTINCT) function has been used but the performance requirement of an aggregate node is not met.
- The aggregate node where the COUNT(DISTINCT) function is executed does not have user-defined aggregate functions (UDAFs).

> **Note** Partial-final optimization divides the aggregation process into two layers, which causes additional network shuffling. Therefore, resources are wasted if the data amount is small.

Optimization method

On the **Parameters** tab of the **Development** page, set blink.partialAgg.enabled to true.

After partial-final optimization is enabled, check whether an expand node is included in the topology or whether the aggregation process has two layers.

Sample code

```
blink.partialAgg.enabled=true
```

7.Monitoring and alerting

This topic describes the monitoring and alerting process in Realtime Compute for Apache Flink and how to create alert rules in Realtime Compute for Apache Flink.

Introduction to CloudMonitor

CloudMonitor helps you collect the monitoring metrics of cloud resources or other custom monitoring metrics, check service availability, and configure alerts based on these monitoring metrics. CloudMonitor helps you view the cloud resource usage, business information, and service health status. In addition, you can receive alerts and respond to these alerts at the earliest opportunity to keep your applications running properly.

Create alert rules

For more information about how to create an alert rule, see Configure an alert rule.

Monitoring items of Realtime Compute for Apache Flink

Monitoring item	Unit	Metric	Dimensions	Statistics
Service delay	s	inputDelay	userId, regionId, projectName, and jobName	Average
Read records per second (RPS)	RPS	ParserTpsRate	userId, regionId, projectName, and jobName	Average
Write RPS	RPS	SinkOutTpsRate	userId, regionId, projectName, and jobName	Average
Failover rate **Note** The failover rate is the average number of failovers per second in the last minute. For example, if one failover occurred in the last minute, the failover rate is 0.01667 (1/60 = 0.01667).	%	TaskFailoverRate	userId, regionId, projectName, and jobName	Average
Processing delay	s	FetchedDelay	userId, regionId, projectName, and jobName	Average

View monitoring metrics

1. Log on to the .
2. In the top navigation bar, click **Administration**.
3. On the **Administration** page, click the name of the job for which you want to view monitoring metrics.
4. In the upper-right corner of the page, choose **More > Monitor**.
5. On the page that appears, view the monitoring metrics of the job.

8.Customize log levels and download paths

You can configure the parameters of a Realtime Compute for Apache Flink job to customize the log download levels and download paths.

> **Notice** Only Realtime Compute for Apache Flink V3.2 and later allows you to customize the log levels and paths.

Procedure

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. In the **Development** section of the left-side navigation pane, double-click the folder that stores the required job and find the job.
4. Double-click the job to go to the job editing page.
5. On the right side of the job editing page, click the **Parameters** tab. In the pane that appears, enter the configuration data of Log4j.

Log4j configuration parameter	Description
Root logger	Processes some operations for logging. The syntax of the root logger is `log4j.rootLogger = [ level ] , appenderName, appenderName, …` . The syntax indicates that the log records whose level is at least the specified level are delivered to one or more destinations. The following items describe the parameters in the syntax: ○ level: specifies the level of logs. The levels of logs include ERROR, WARN, INFO, and DEBUG that are sorted in descending order. ERROR indicates serious errors for the job. WARN indicates potentially harmful situations for the job. INFO indicates informational messages for the job. DEBUG indicates the debugging information about the job. ○ appenderName: specifies the name of the destination to which log records are delivered. You can specify multiple appenders for the root logger.

Log4j configuration parameter	Description

Appender	Specifies the destination to which log records are delivered. To define an appender, use the following syntax:
	`log4j.appender.appenderName = fully.qualified.name.of.appender.class`
	`log4j.appender.appenderName.option1 = value1`
	`...`
	`log4j.appender.appenderName.optionN = valueN`
	Appenders have the following two types:
	◦ Appenders provided by Log4j
	■ `org.apache.log4j.ConsoleAppender`: delivers log records to a console.
	■ `org.apache.log4j.FileAppender`: delivers log records to a file.
	■ `org.apache.log4j.DailyRollingFileAppender`: generates a log file every day.
	■ `org.apache.log4j.RollingFileAppender`: generates a new file when a file reaches its maximum size.
	■ `org.apache.log4j.WriterAppender`: delivers log records in the stream format to a specified destination.
	◦ Custom appenders
	Custom appenders allow you to deliver log records to only the Log Service and Object Storage Service (OSS) services. The classes of the two services are fixed. The following example lists the fixed classes.
	■ Log Service: `log4j.appender.loghub=com.alibaba.blink.log.loghub.BlinkLogHubAppender`
	■ OSS: `log4j.appender.oss=com.alibaba.blink.log.oss.BlinkOssAppender`
	You can also specify other configuration items that are supported by the Log4j syntax.
	For more information, see Change the log level to DEBUG and deliver log records to an OSS bucket..

Note After you specify the job parameters, restart the job. You can view the new logs in the OSS bucket.

6. **Terminate** the job. For more information, see Terminate a job.
7. **Start** the job. For more information, see Start a job.

Considerations

- The OSS path must be the same as that for the Realtime Compute for Apache Flink cluster that is

deployed in exclusive mode.

- **The logs that you can download** are provided through Log4j. *system.out* logs are not included in these logs.
- The specified Log Service or OSS can interact with the cluster where the job resides.
- The job can be started in most cases even if the custom log output configuration is invalid. However, the logs of the job cannot be displayed based on the configuration.

Change the log level to DEBUG and deliver log records to an OSS bucket.

Notice If you specify the log4j.rootLogger parameter, you may fail to view log information or troubleshoot related issues on the Realtime Compute for Apache Flink development platform. Use this parameter with caution.

```
#Change the log level to DEBUG and export logs to a specific file in an OSS bucket.
log4j.rootLogger=DEBUG, file, oss
#This parameter setting is fixed. You do not need to change the setting. Configure the appe
nder class for OSS.
log4j.appender.oss=com.alibaba.blink.log.oss.BlinkOssAppender
#The endpoint.
log4j.appender.oss.endpoint=oss-cn-hangzhou****.aliyuncs.com
#The AccessKey ID.
log4j.appender.oss.accessId=U****4ZF
#The AccessKey secret.
log4j.appender.oss.accessKey=hsf****DeLw
#The OSS bucket name.
log4j.appender.oss.bucket=et****
#The subdirectory that is used to store logs.
log4j.appender.oss.subdir=/luk****/test/
```

Disable the output of the logs for a specified package, and deliver logs to a specified Logstore of Log Service.

```
#Disable the output of the logs for the log4j.logger.org.apache.hadoop package.
log4j.logger.org.apache.hadoop = OFF
#This parameter setting is fixed. You do not need to change the setting. Configure the LogH
ub appender.
log4j.appender.loghub = com.alibaba.blink.log.loghub.BlinkLogHubAppender
#Deliver only logs of the ERROR level to Log Service.
log4j.appender.loghub.Threshold = ERROR
#The name of a project in Log Service.
log4j.appender.loghub.projectName = blink-errdumpsls-test
#The name of a Logstore in Log Service.
log4j.appender.loghub.logstore = logstore-3
#The endpoint of Log Service.
log4j.appender.loghub.endpoint = http://cn-shanghai****.sls.aliyuncs.com
#The AccessKey ID.
log4j.appender.loghub.accessKeyId = Tq****WR
#The AccessKey secret.
log4j.appender.loghub.accessKey = MJ****nfVx
```

Change the log level to WARN, and disable the output of the logs for a specified package.

```
#Change the log level to WARN.
log4j.rootLogger=WARN,file
#Disable the output of the logs for the log4j.logger.org.apache.hadoop package.
log4j.logger.org.apache.hadoop = OFF
```

9.Manage Blink versions of a Realtime Compute for Apache Flink cluster deployed in exclusive mode

Realtime Compute for Apache Flink provides the version management feature for you to manage Blink versions of a Realtime Compute for Apache Flink cluster in exclusive mode.

Install a version

1. Go to the Version Management page
   i. Log on to the .
   ii. Move the pointer over the username in the top navigation bar and click **Project Management**.
   iii. In the left-side navigation pane, choose **Cluster Management > Clusters**.
   iv. On the **Clusters** page, find the cluster for which you want to manage Blink versions. In the **Actions** column, choose **More > Version Management**.
2. On the **Installable Version** tab, find the version you want to install, and click **Install** in the **Actions** column.

   On the **Installable Version** tab, choose a proper version based on your business requirements and the version features.

Tag	Description
stable	The recommended stable Blink version.
beta	The beta version programmed for testing. **Note** We recommend that you install a beta version in only specified scenarios. The performance and the stability of the beta version cannot be ensured in other scenarios.
No tag	The historical stable version.

Note

- You can install one version at a time. During installation, the cluster status is **Version Installing**.
- You cannot install a version that has been installed.
- Each version is configured with a service period. The default service period is one year. After the service period expires, you can continue to use the version, but Alibaba Cloud no longer maintains the version.

Switch a version

Blink features vary based on Blink versions. You can configure a Blink version based on your business requirements. You can **switch the Blink version** for a job.

1. Log on to the .
2. In the top navigation bar, click **Development**.
3. On the **Development** page, double-click the job in a folder that stores the job to go to the **job editing** page.
4. In the right side of the **job editing** page, click Version Information.
5. From the version list, select the Blink version you want to switch to.

Uninstall a Blink version

Realtime Compute for Apache Flink allows you to install only three Blink versions. If you already install three Blink versions and need to install a new version, you must uninstall an existing version.

1. Go to the Version Management page
   i. Log on to the .
   ii. Move the pointer over the username in the top navigation bar and click **Project Management**.
   iii. In the left-side navigation pane, choose **Cluster Management > Clusters**.
   iv. On the **Clusters** page, find the cluster for which you want to uninstall a version. In the **Actions** column, choose **More > Version Management**.
2. On the **Installed Versions** tab, find the version, and click **Uninstall** in the **Actions** column.

> Note
> - You cannot uninstall a Blink version that is set to **current**.
> - You cannot uninstall a Blink version that is referenced by a Realtime Compute for Apache Flink job.

Specify the current version

You can select an installed Blink version and set the version to the current version of a job in the current cluster. Blink features vary based on Blink versions. You can set a Blink version to the current version of a Realtime Compute for Apache Flink cluster based on your business requirements.

1. Go to the Version Management page
   i. Log on to the .
   ii. Move the pointer over the username in the top navigation bar and click **Project Management**.
   iii. In the left-side navigation pane, choose **Cluster Management > Clusters**.
   iv. On the **Clusters** page, find the cluster for which you want to set a version to the current version. In the **Actions** column, and choose **More > Version Management**.
2. On the **Installed Version** page, find the version that you want to set to the current version, and click **Set to current** in the **Actions** column.

FAQ

- What can I do if the `blink-<version> already installed` error message appears during installation?

  The error message is returned because the Blink version has been installed. You do not need to install the version again.

- What can I do if the `Flink versions exceeded max limitation:3` error message appears during installation?

  The error message is returned because the number of versions exceeds the upper limit. To install a new version, you must delete existing versions until the number of installed versions is less than three.

- What can I do if the `Node:<nodeName> in project:<projectName> still ref the version:<blinkVersion>` error message appears during uninstallation?

  The error message is returned because the Blink version is referenced by an online job. You must bring the job offline based on the job name and project name provided in the error message.

- What can I do if the following message appears when I click **Syntax Check** or **Publish**?

  ```
  code:[30006], brief info:[blink script not exist, please check blink version], context info:[blink script:[/home/admin/blink/blink-2.2.6-hotfix0/bin/flink], blink version:[/home/admin/blink/blink-2.2.6-hotfix0/bin/flink]]
  ```

  The error message is returned because the Blink version used by the job does not exist. In the Realtime Compute for Apache Flink console, choose **Version Management > Installed Versions** to check whether the version exists.

  - If the version does not exist, switch to another version or install the version.
  - If the version exists, submit a ticket.