Stream Compute Service

Operation Guide

Product Documentation

Contents

Operation Guide
Managing Jobs
Job Overview

Last updated：2023-11-08 10:14:24

You can view job overview, cluster overview, and other information on the **Jobs** page.
The job overview displays the summary of jobs in the current workspace, including the jobs by type, running and stopped jobs, and jobs with events. The cluster overview displays clusters associated with the workspace.

Note

Jobs with metric alarms counts the number of jobs which have uncleared alarms within the last day.
Jobs with events counts the number of jobs which have events within the last 6 hours.
Cluster overview displays details of monthly subscribed clusters associated with the workspace.

Job Types

Last updated：2023-11-08 10:15:00

You can log in to the Stream Compute Service console to create a job. At present, four job types are available for selection on the **Create job** page: SQL, JAR, ETL, and Python. You can select a job type based on your business needs and scenarios.

SQL job

Compared with other programming languages, SQL costs less in terms of studying it. SQL-based job development lowers the requirements for data developers to use Flink. A SQL job allows quick view of dynamic and static data in a stream. It is suitable for building powerful data conversion or analysis pipelines. In addition, SQL jobs apply identical semantics for both streaming and batch inputs, so that the same computing results can be obtained.

JAR job

A JAR job is developed based on the code of `Flink DataStream API` or `Flink Table API` . Those who develop a JAR job need to have knowledge of Java or Scala DataStream API. This job type is suitable for users who focus on the underlying part of stream computing and require high complexity. To develop a JAR job, you need to develop and compile the JAR package in the local system first.

ETL job

An extract, transform, and load (ETL) job enables you to collect data from various sources. It transforms the data, provides some additional information, and stores the results. An ETL job is easy to operate. It takes about 1 minute to create a lightweight ETL job. To start an ETL job, you even do not need to have the knowledge of programming languages, and you just need to select a data source table and a destination table and configure field mapping based on the business logic. The data in your business systems will be extracted, cleansed/transformed, and loaded into a data warehouse.

Python job

A Python job is developed based on the Python code. It requires the developers to have knowledge of Python and the libraries/packages supported by it. Compared with other programming languages, Python is easy to learn. Python

specifies fewer conventions and special cases in its syntax. It focuses on what you want to accomplish with your code, but not rich language representations. It is relatively easy to learn and use. To develop a Python job, you need to first write your Python files and package them into a .zip file in the local system, and then upload the Python package before configuring the job in the console.

Job Information

Last updated：2023-11-08 10:11:04

You can log in to the Stream Compute Service console, and click **Jobs** on the left sidebar to view your jobs. Then, you can click a job name in the **Job list** to view the details of this job on the **Job overview** page. The meanings of the fields on the page are described below.

Field	Description
Job name	The name of the job, which is set when the job is created and can be modified.
Cluster	The name of the cluster where the job resides.
Cluster ID	The ID of the cluster where the job resides.
Job ID	The serial ID of the job, usually starting with cql- (assigned at random, immutable).
Job type	The type of the job. Four types are available: JAR, SQL, Python, and ETL.
Status	The current status of the job, such as uninitialized, unpublished, operating, running, stopped, or error.
Region	The geographical region of the cluster where the job resides, such as Guangzhou, Shanghai, or Beijing.
AZ	The AZ of the cluster where the job resides, such as Shanghai Zone 3.
Online version	The running version.
Creation time	The time when the job is created.
Cumulative run time	The total run time of the job.
Start time	The time when this job run starts.
Run time	The duration of this job run.
Compute resources	The number of CUs used in this job run. It is the sum of the number of JobManager CUs and that of TaskManager CUs, where the number of JobManager CUs = 1 (1 for each job by default) and that of TaskManager CUs = Maximum parallelism x CUs per TaskManager.

Job Operations

Last updated：2023-11-08 10:12:24

You can perform the following four operations on a job in the Stream Compute Service console: Publish, Run, Stop, and Create copy. Some of these operations can be performed in batches. This document describes these operations in detail.

Publish

Description

After a job draft is developed, click **Save** and **Publish draft** to publish it as an online version. After the job is **published** successfully, depending on the job status, you can **run** a published version or **stop** an online running version of it.

Directions

After a job draft is developed, click **Save** and **Publish draft** to publish it as an online version. The version number of a new version is automatically generated by the system, but you need to enter a version description.
If no job version is **running** online, directly **run** the published version.
If a job version is **running**, **stop** it first and select **Create a checkpoint when stopping the job**, if necessary, in the pop-up window. After the job is completely stopped, **run** the new version.

Run

Description

Run corresponds to **Stop**. It means starting a new job running instance. To run a running job, **stop** the online running version first, and then perform **Run**.

Directions

You can run a **Stopped** job using one of the following methods.
1. In the Stream Compute Service console, click **Jobs > Operations > More**, and select **Run** in the drop-down list. The job status will become **Operating**. After a moment, the job status will become **Running**, indicating that the job is successfully started.
2. Publish and run a new version of the job. The job status will become **Operating**. After a moment, the job status will become **Running**, indicating that the job is successfully started.

Resuming a job from a checkpoint

When running a job, you can select **Use a checkpoint**.

This option is unavailable to a job without checkpoints.

Note

A cluster of an old version does not support selecting a checkpoint. To use this feature, submit a ticket to upgrade your cluster first.

Notes

If any exception occurs when you are operating a job, the job will go back to its actual status. For example, if the operation of stopping a job is aborted, the job status will be **Stopped**; if the operation fails and the job is still running, the job status will be **Running**. If this is the case, a triangle with an exclamation point will be displayed on the right of the status field, which will show the error message when you move the pointer over it.

Operation suggestions will be provided in common error messages.

If an error message is not easy for you to understand, please contact us. We will continuously improve error messages.

Note

When a job is running, modify the configurations of upstream and downstream products with caution, including but not limited to deleting or adding topics of CKafka data sources and sinks, and locking tables, modifying table structure, adding constraints, and stopping operations on MySQL data sources and sinks. Otherwise, the running job may be affected, resulting in incomplete data or job exception.

Stop

Description

Stop means stopping the execution of a job. When performing this operation, you need to decide whether to save all runtime status data.

Directions

You can stop a **Running** job using one of the following methods.

1. In the Stream Compute Service console, click **Jobs** > **Operations** > **More**, and select **Stop** in the drop-down list. The job status will become **Operating**. After a moment, the job status will become **Stopped**, indicating that the job is completely stopped.

2. Publish a new version, and stop the current running version. The job status will become **Operating**. After a moment, the job status will become **Stopped**, indicating that the job is completely stopped.

Note

If you want to retain the current state data of the job and make the job resume when it is started next time, select **Create a checkpoint when stopping the job**.

Create copy

Description

Job copies can be created quickly to facilitate job migration or development of similar jobs.

Directions

Creating a job copy

1. Go to the job details page, and select **Create copy** in the drop-down list of **Job operations** in the top right corner.
2. Click **Create copy**, and then select a target directory and target cluster and enter a copy name in the pop-up window.
3. Click **Confirm**. You will see the result shown in the pop-up window.
4. Click **Job name** to go to the details page of the created job.

Creating job copies in batches

1. Click **Batch** on the top of the job list, and select **Create copy**.
2. Select the jobs for which you want to create a copy.
3. Click **Create copy**, and then select a target cluster in the pop-up window. **In this operation, you cannot select a target directory. The directory of a copy will be the same as that of the source job.**
4. Click *Confirm**. You will see the result (success or failure) in the pop-up window.

Note

If the target cluster does not support the Flink version of the source job, the Flink version of the job copy will be changed to the default Flink version of the target cluster.

If the source job uses fine-grained resources, but the target cluster does not support such resources, both the JobManager and TaskManager specs of the job will be set to 1 CU.

Jobs can be copied only within the workspace where they reside, and a target cluster must be one associated with the workspace.

Checkpoints of source jobs will be synced (by sharing the same checkpoint path) to new jobs created by copying, but only checkpoints manually triggered will be synced.

Developing Jobs

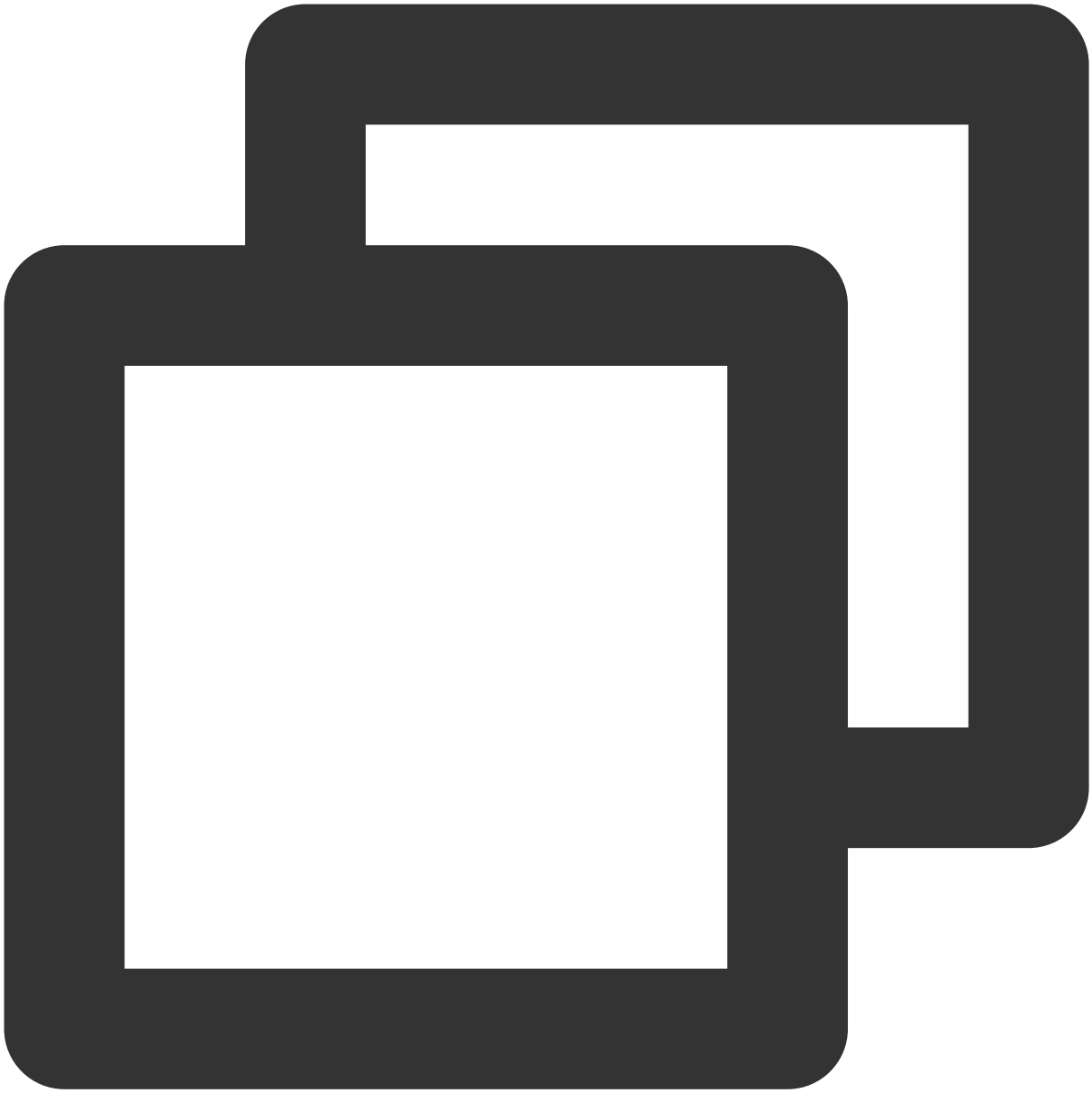
Developing Jobs in Batches

Last updated：2023-11-08 10:09:02

Stream Compute Service supports development of batch jobs of the following types: JAR and Python.

JAR job

The following example shows how to configure the batch job execution mode in the JAR job code.

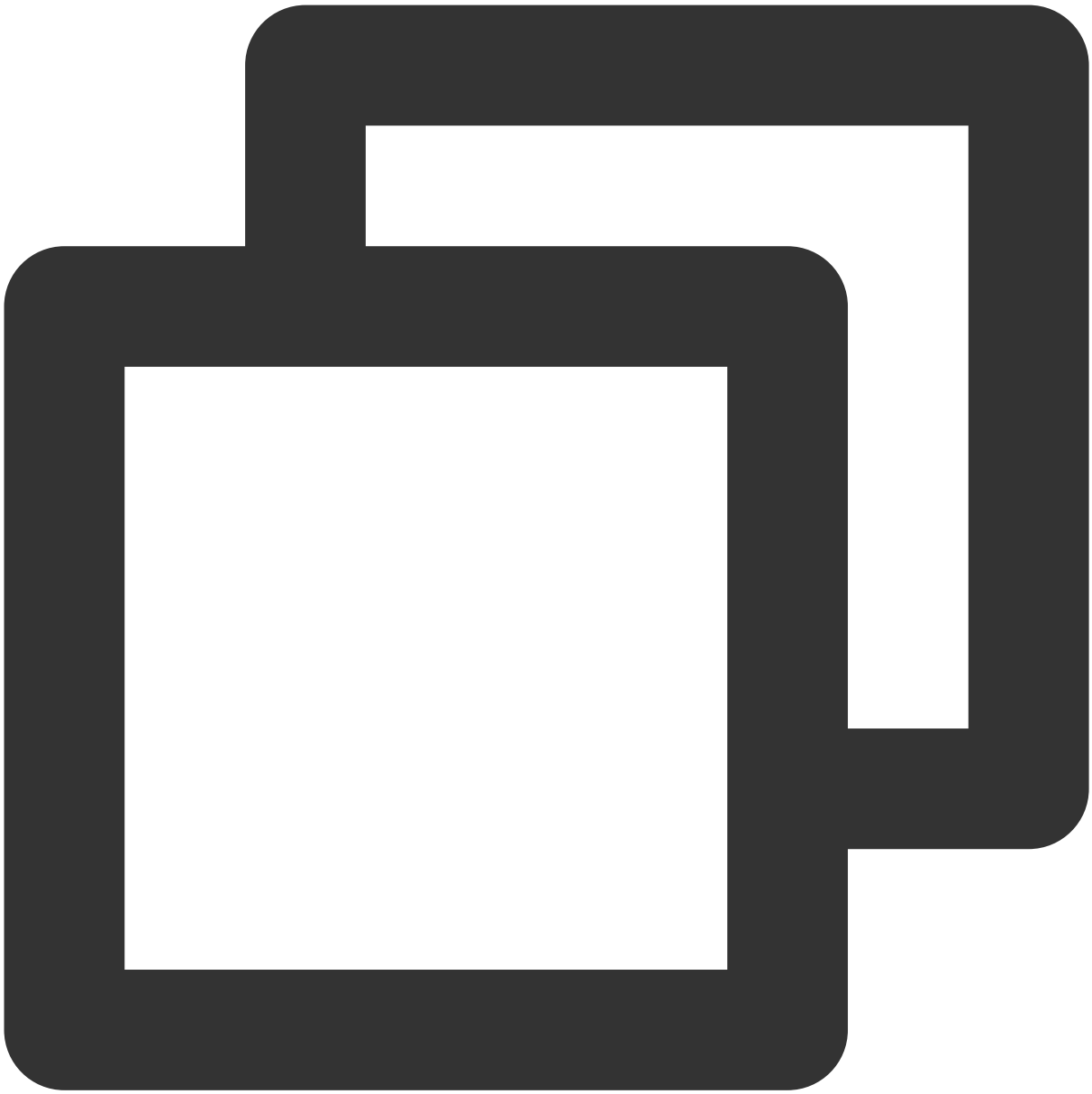


```
StreamExecutionEnvironment env = StreamExecutionEnvironment.getExecutionEnvironment
env.setRuntimeMode(RuntimeExecutionMode.BATCH);
```

1. Change `RuntimeExecutionMode.STREAMING` to `RuntimeExecutionMode.BATCH` in a streaming job.
2. Package the modified job into a .jar file and upload it to **Dependencies** in the platform.
3. Use the uploaded .jar file to complete the development of the JAR batch job in the platform.

Python job

The following example shows how to configure the batch job execution mode in the Python job code.



```
env_settings = EnvironmentSettings.new_instance().in_batch_mode().use_blink_planner
table_env = TableEnvironment.create(env_settings)
```

1. Change `EnvironmentSettings.new_instance().in_streaming_mode()` to `EnvironmentSettings.new_instance().in_batch_mode()` in a streaming job. For details, see Intro to the Python Table API.
2. Package the modified job into a .py file and upload it to **Dependencies** in the platform.
3. Use the uploaded .py file to complete the development of the Python batch job in the platform.

Advanced Job Parameters

Last updated：2023-12-26 17:49:27

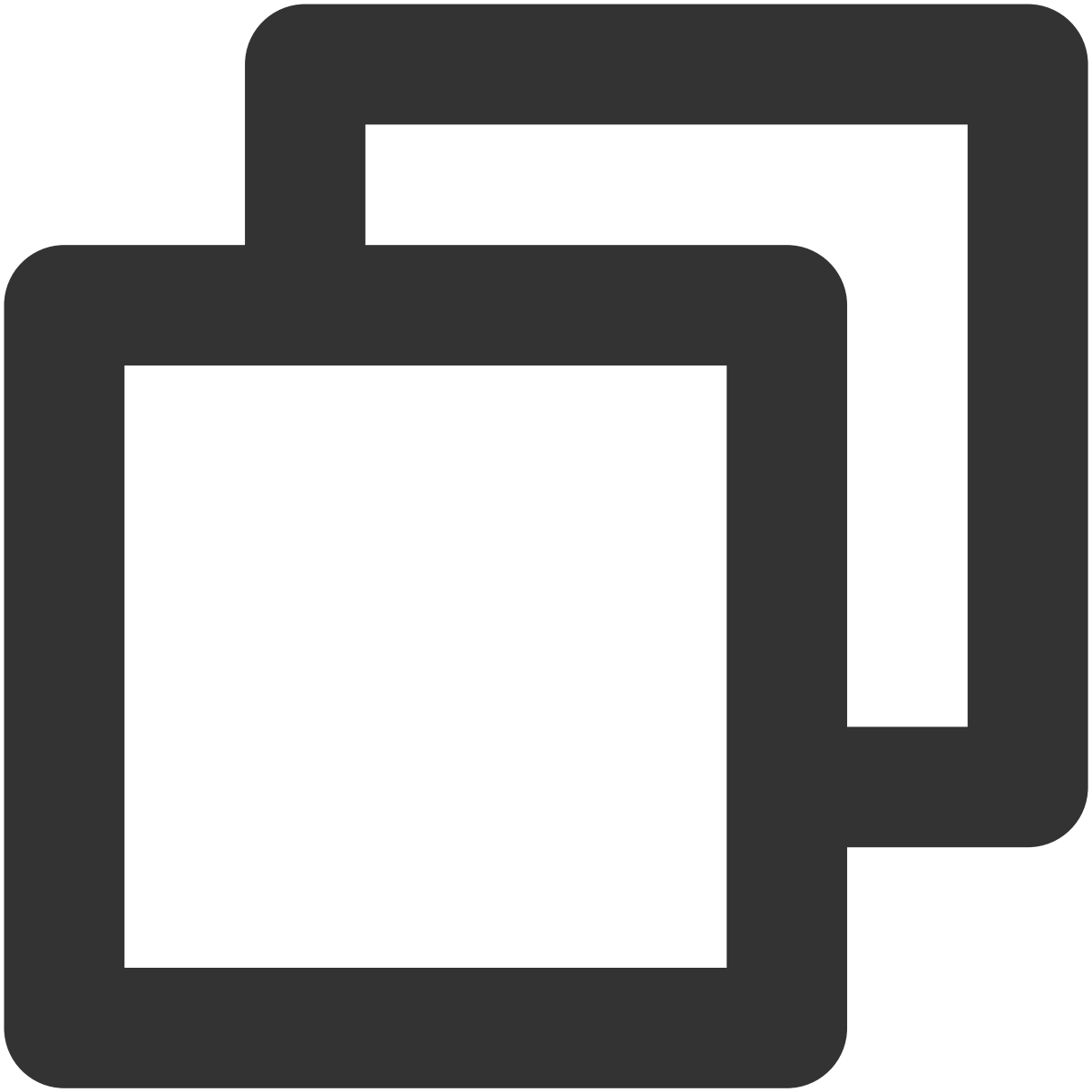
Overview

You can set more custom Flink parameters in **Job parameters > Advanced parameters** to tune a job. For example, you can set the job restart policy, adjust the mini-match settings of SQL, disable async checkpointing, set minimum checkpoint interval, and adjust the cache size of `RocksDB StateBackend` .
Custom advanced parameters must comply with the YAML syntax, configured in the "key: value" format (note there is a space following the colon). After job parameters are modified, you need to re-publish and start the job to apply new parameters. For details of parameters in Flink v1.11, see Configuration.

Example

Setting the state backend of a job

RocksDB state backend is used by default in Stream Compute Service. It allows access of a larger state, but its throughput and performance are inferior to those of the memory-based FileSystem state backend.
If your job state is small, and you require low latency and high throughput, change the state backend to the FileSystem state backend using the following statement:

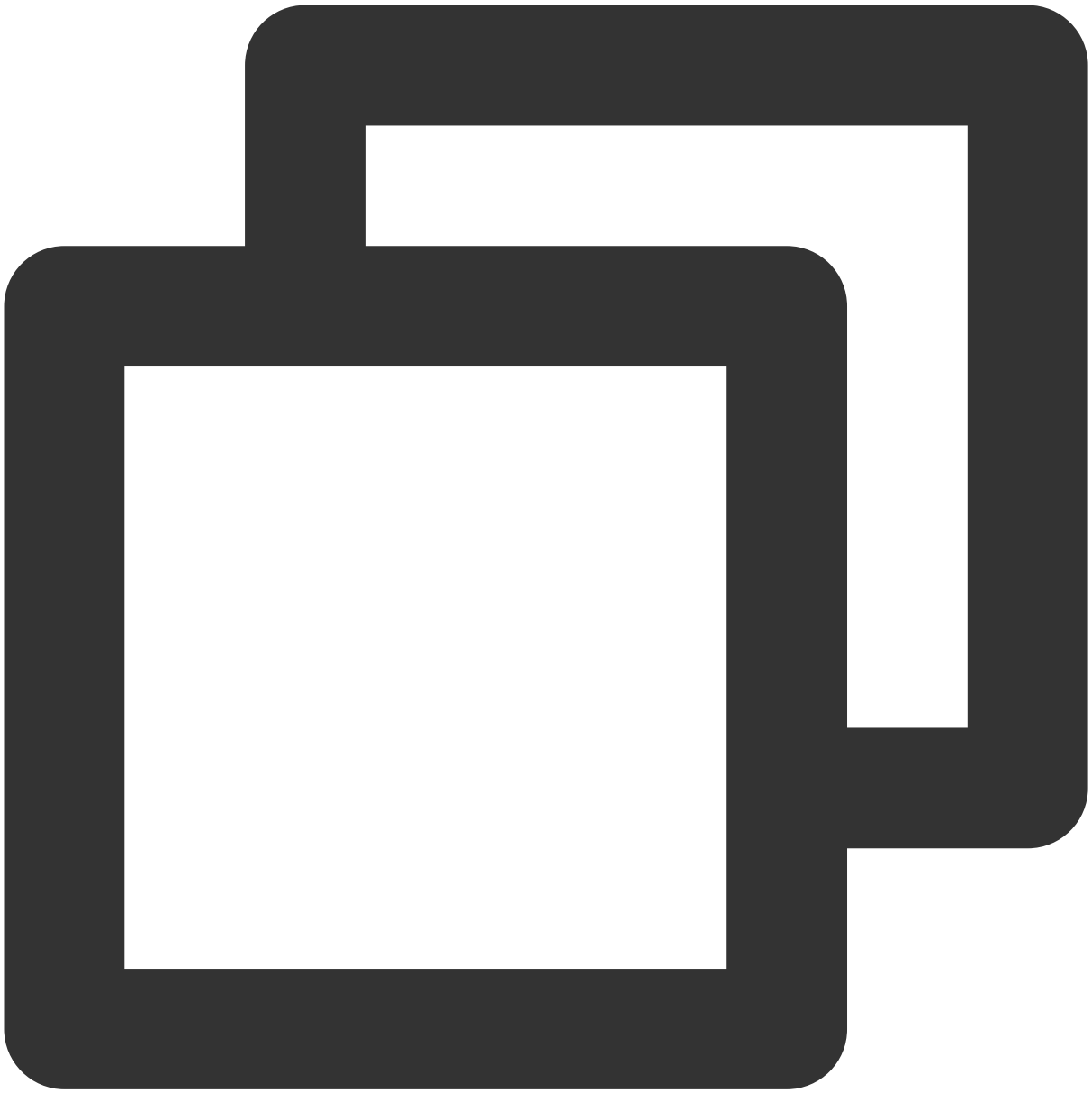


```
state.backend: filesystem
```

Setting job restart policy and threshold

By default, a Flink job can be internally restarted (hot restart when the JobManager is still active, which takes about 15s) a maximum of five times after crash. If a crash occurs again after the number of restarts reaches the threshold, the JobManager will exit, resulting in a longer cold recovery period of the job (about 3-5 minutes). If checkpointing hasn't been enabled for the job, a lot of its state and data may be lost.

To adjust the number of internal restarts allowed of the job, configure the following parameter (in this example, a maximum of 100 internal restarts are allowed. Set the parameter with caution):



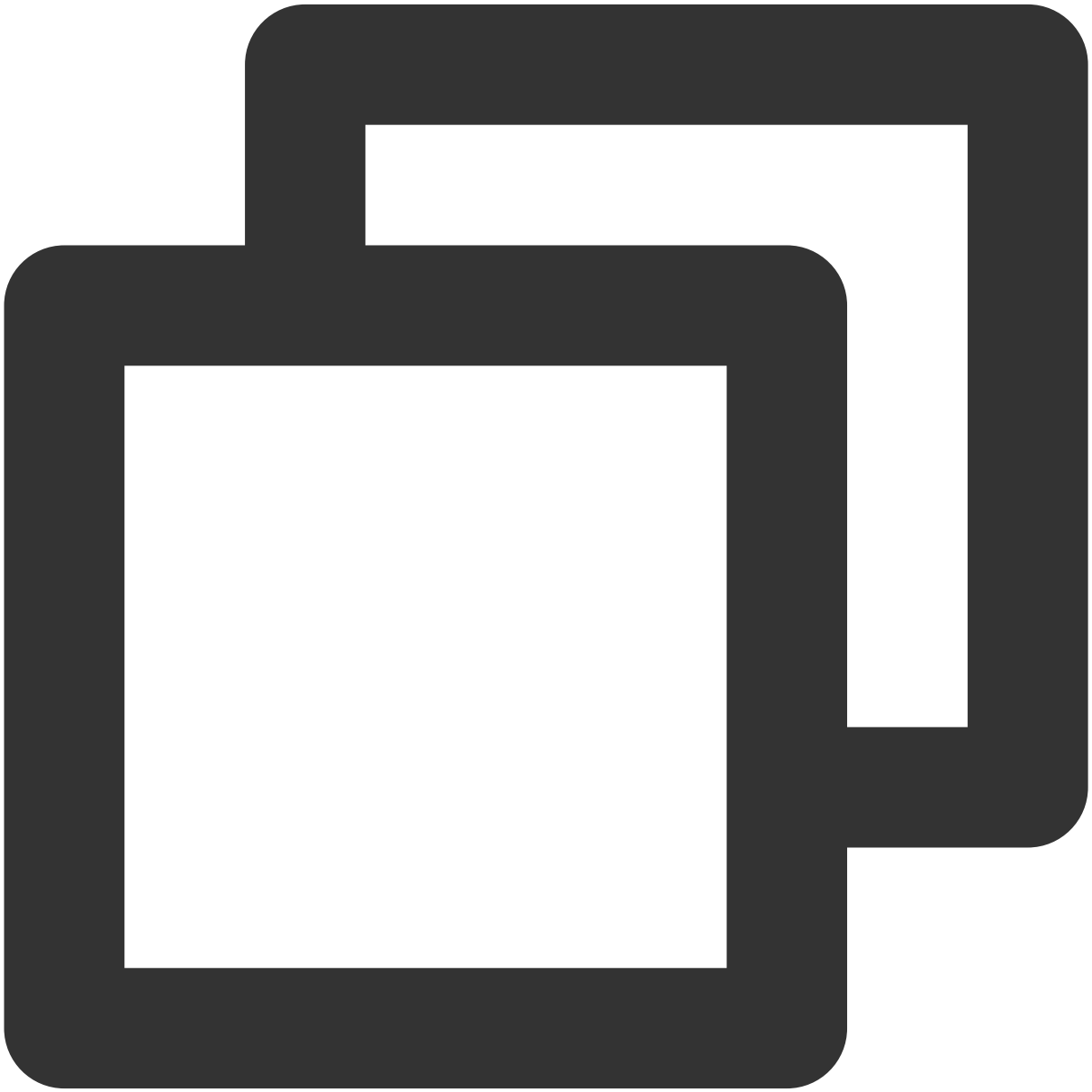
```
restart-strategy: fixed-delay
restart-strategy.fixed-delay.attempts: 100
restart-strategy.fixed-delay.delay: 5 s
```

Setting the JVM overhead percentage

The percentage of JVM overhead defaults to 10% in Flink. When the RocksDB state backend is used, it requires a larger memory in this area, which may cause overuse and JVM to be killed by the container management system. To minimize this situation and keep the job using the RocksDB state backend more stable, we recommend you increase the value of this parameter as appropriate.

Note

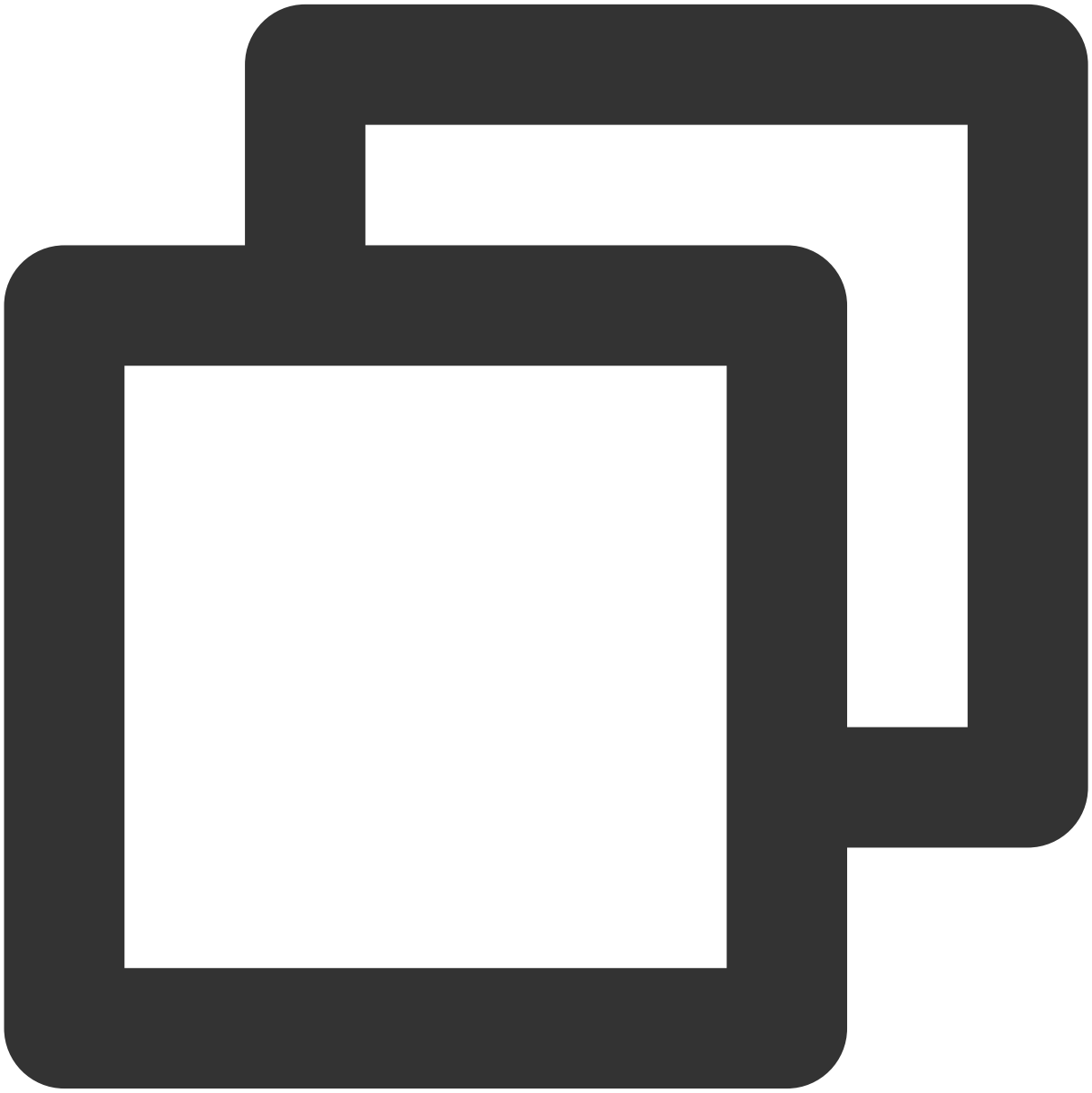
Increasing the value of this parameter will lower the percentage of available heap memory in JVM, making the job more prone to the out-of-memory (OOM) error in the heap. Please proceed with caution.



```
taskmanager.memory.jvm-overhead.fraction: 0.3
```

Setting the checkpoint policy to at-least-once

The default checkpoint policy is exactly-once in Stream Compute Service. This policy can ensure exact state consistency after a crashed job is recovered, but it may sometimes cause a high latency.
If a part of duplicate data is allowed to be used in computing (resulting in inaccurate results for a short period of time) when the crashed job is recovered, you can change the Flink checkpoint policy to at-least-once for better checkpoint performance, especially when the state is huge and multiple streams have different rates.



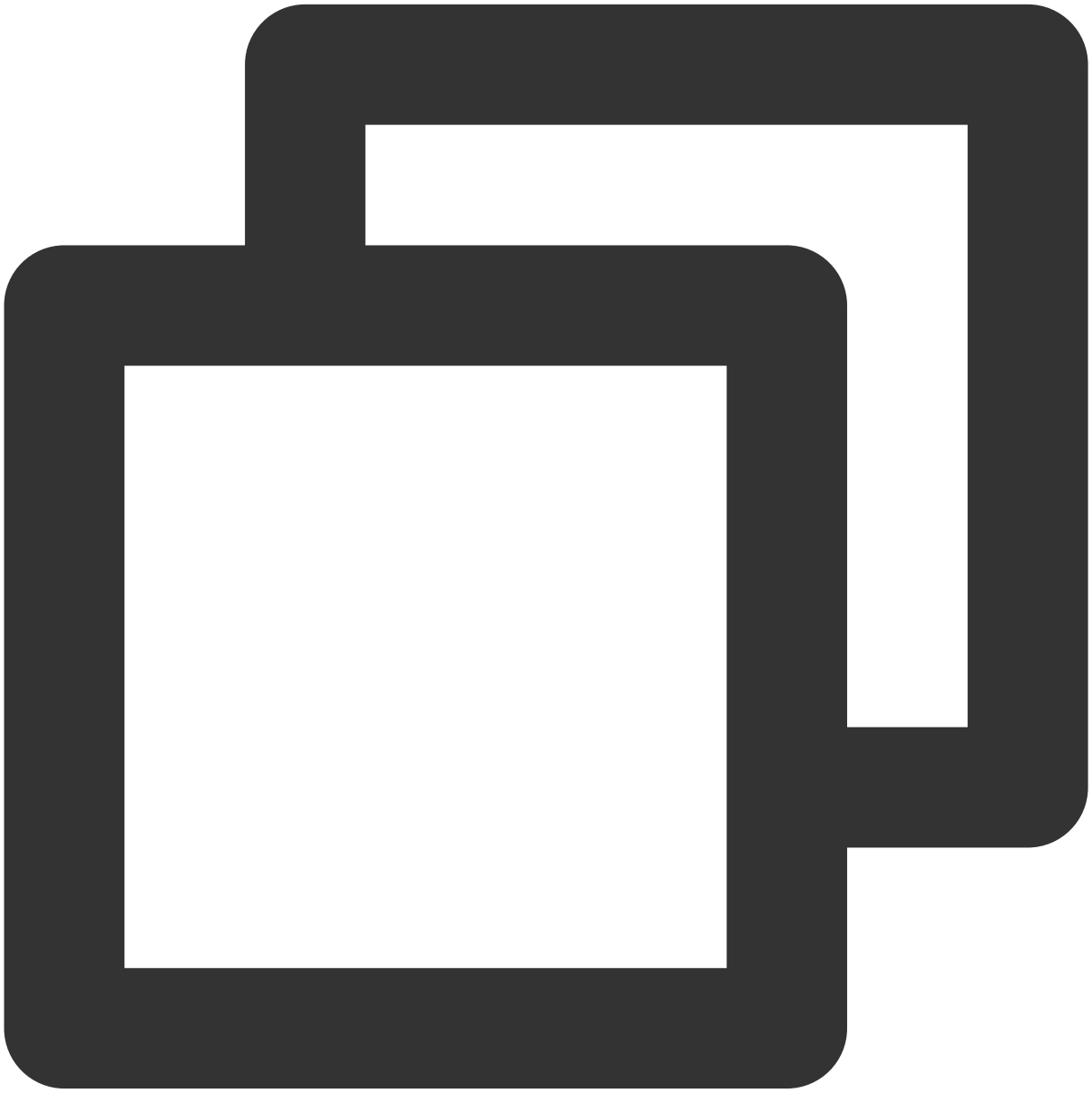
```
execution.checkpointing.mode: AT_LEAST_ONCE
```

Disabling operator chaining

By default, operators with the same parallelism are chained together if possible in the execution graph in Flink to avoid additional serialization or deserialization of data transferred between upstream and downstream operators. If you want to view the data inflow and outflow of each operator to facilitate troubleshooting, disable this operator chaining feature.

Note

Disabling this feature may cause the running efficiency of the job to decline greatly. Please proceed with caution.

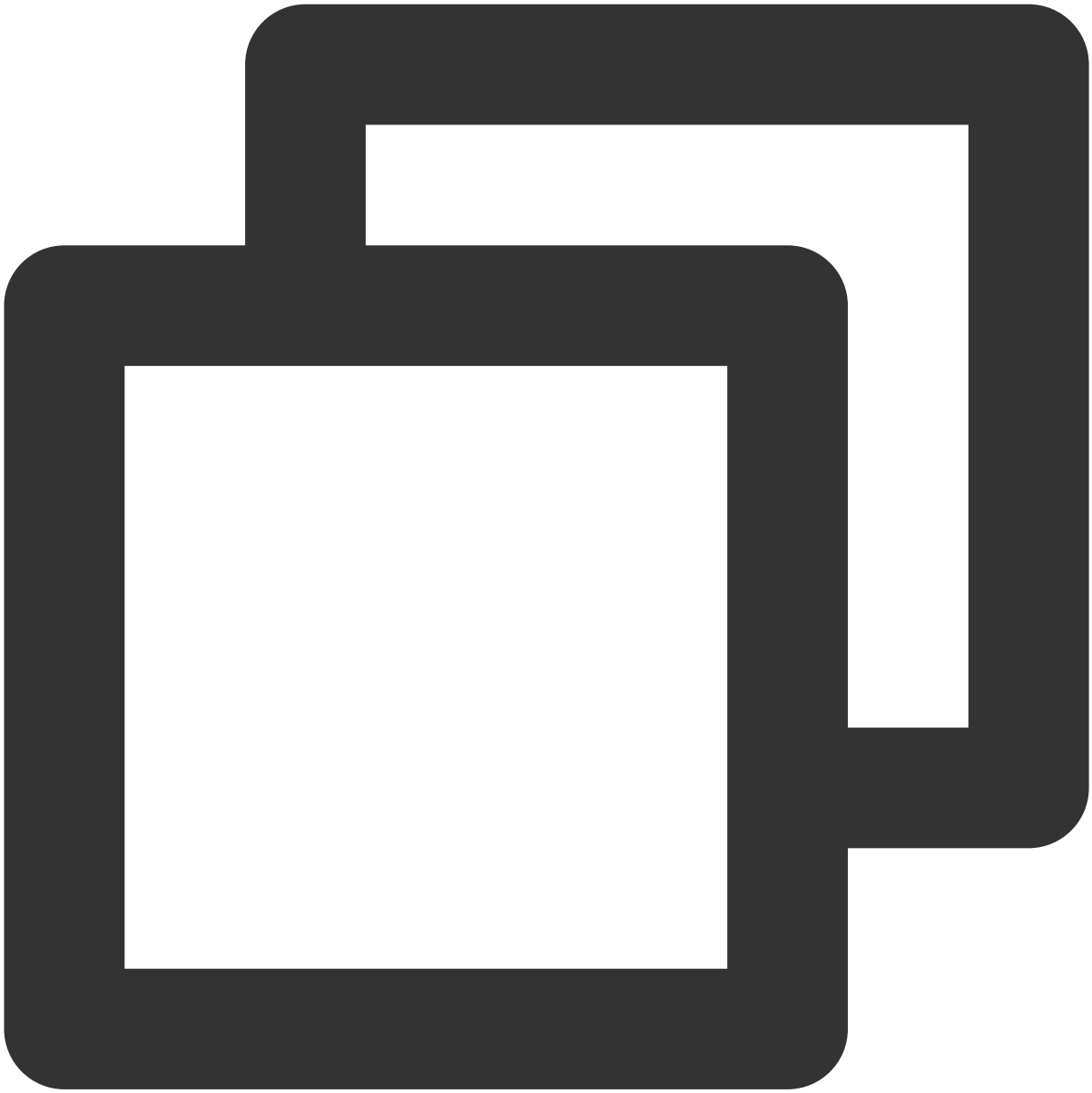


```
pipeline.operator-chaining: false
```

Setting the checkpoint timeout of a job

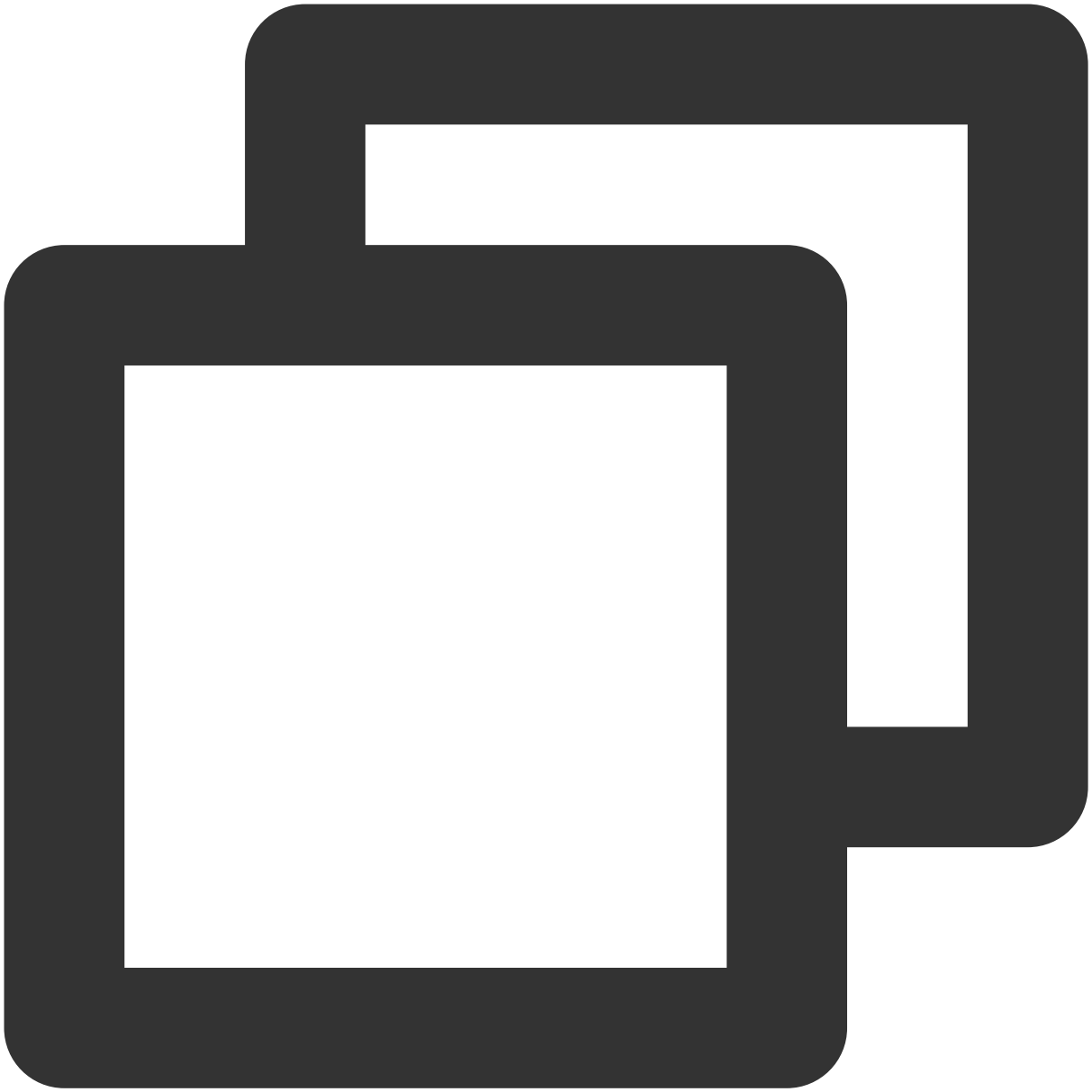
The checkpoint timeout defaults to 20 minutes (1,200s) in Stream Compute Service.

If your job state is large, you can set a longer timeout with the following parameter:



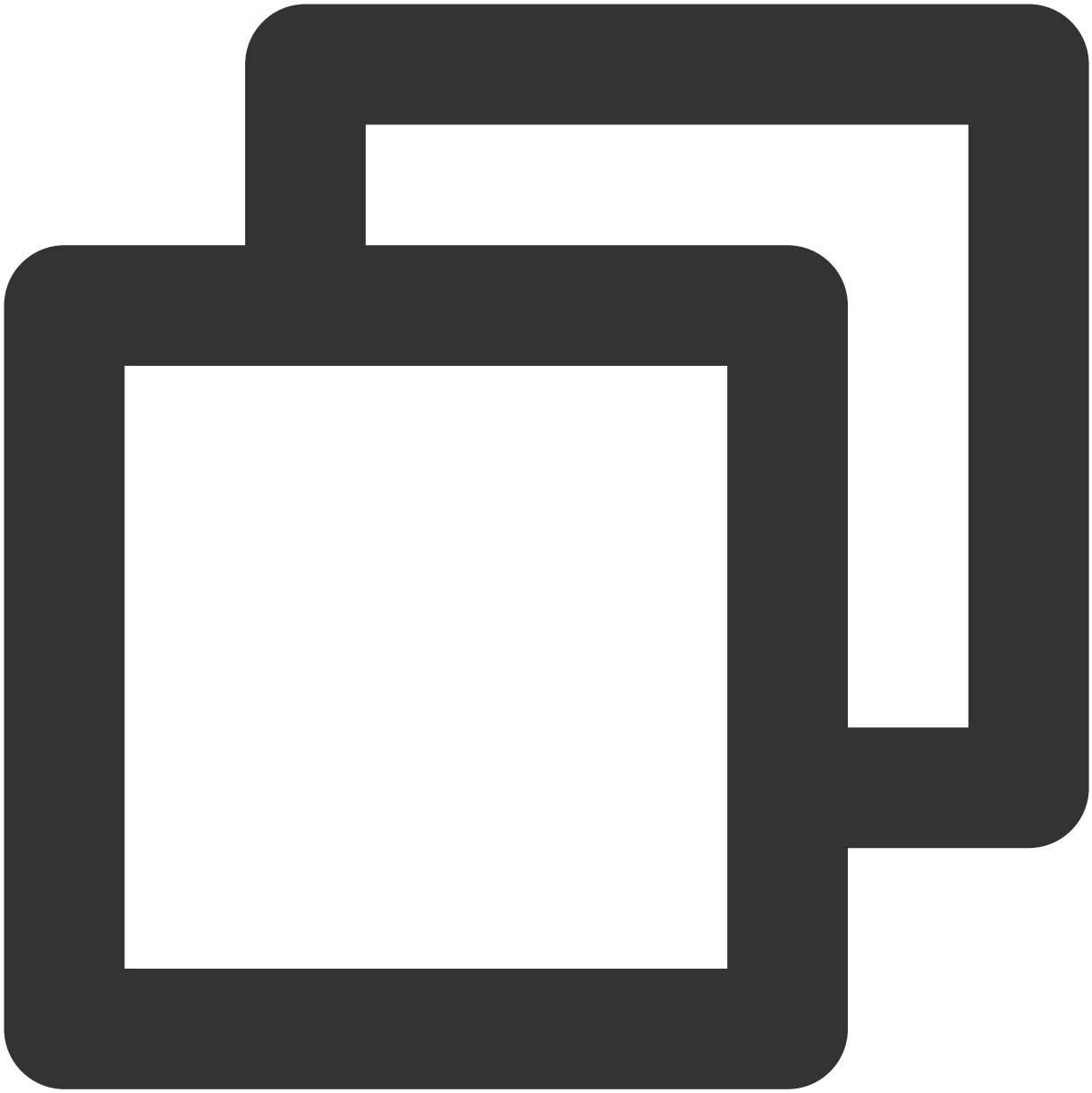
```
execution.checkpointing.timeout: 3000s
```

You can also reduce the timeout:



```
execution.checkpointing.timeout: 1000s
```

You also need to add the following statement on the editing page of a SQL job, with the value set to the configured timeout. For details, see Flink Configuration Options.



```
set CHECKPOINT_TIMEOUT= '1000 s';
```

Setting the checkpoint storage policy

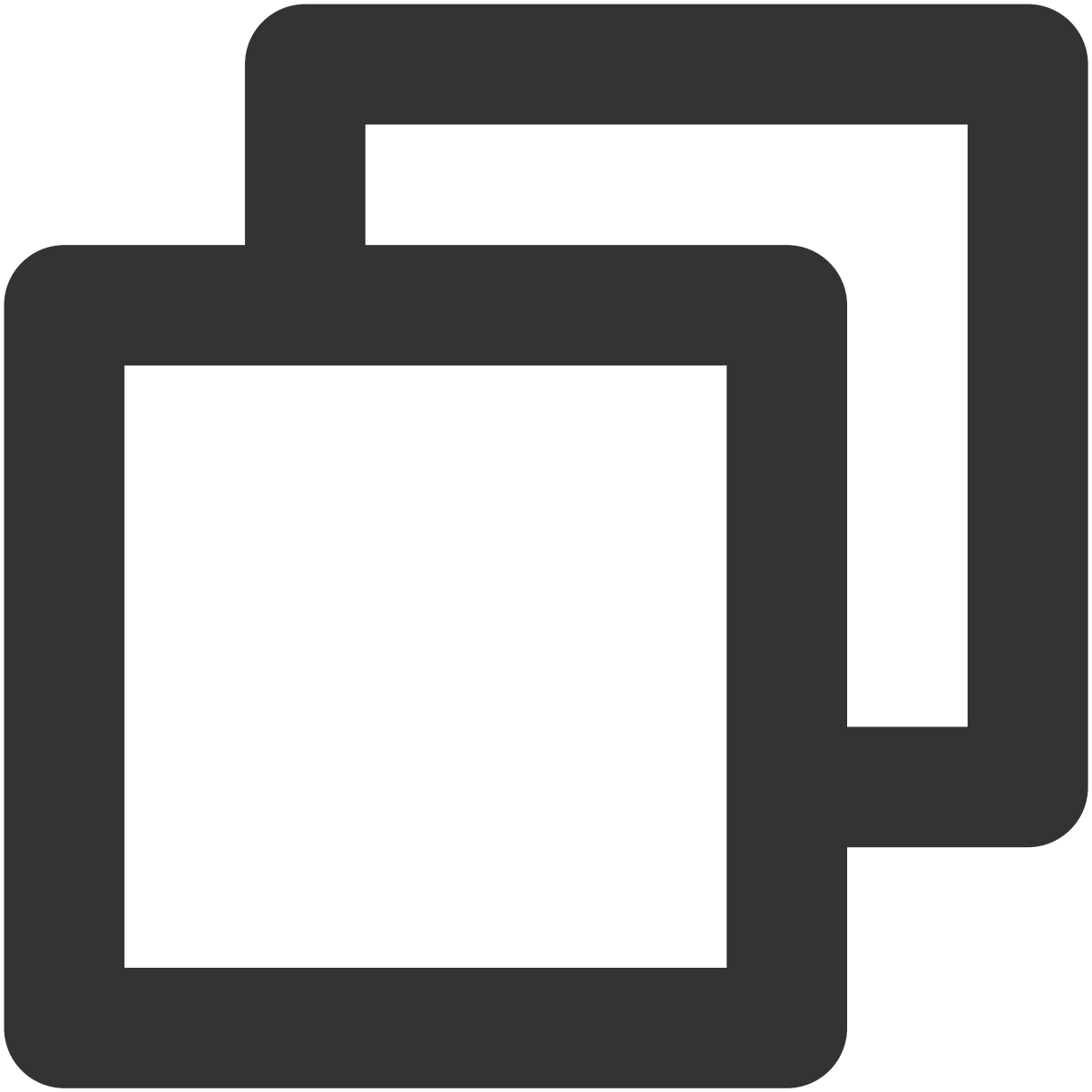
In Stream Compute Service, three checkpoint storage policies are available to Flink jobs: `DELETE_ON_CANCELLATION`, `RETAIN_ON_CANCELLATION`, and `RETAIN_ON_SUCCESS`. The default policy is `DELETE_ON_CANCELLATION`. If this parameter is not set, the default policy will be used.

The following table compares these policies.

Checkpoint Storage Policy	Checkpoint Clearing

DELETE_ON_CANCELLATION (default)	1. Create a checkpoint when the job is stopped, and delete the old checkpoint (so cannot recover the job from the old one) 2. Create no checkpoint when the job is stopped, and delete the old checkpoint (so cannot recover the job from checkpoint)
RETAIN_ON_CANCELLATION	1. Create a checkpoint when the job is stopped, and delete the old checkpoint (so cannot recover the job from the old one) 2. Create no checkpoint when the job is stopped, and do not delete the old checkpoint (so can recover the job from checkpoint)
RETAIN_ON_SUCCESS	1. Create a checkpoint when the job is stopped, and do not delete the old checkpoint (so can recover the job from checkpoint) 2. Create no checkpoint when the job is stopped, and do not delete the old checkpoint (so can recover the job from checkpoint)

You can set the checkpoint storage policy of a job in **Job parameters** > **Advanced parameters**. After the setting, you need to restart the job to apply the policy.



```
execution.checkpointing.externalized-checkpoint-retention: RETAIN_ON_SUCCESS
```

Note

For a JAR or Python Flink job, you are not advised to explicitly set the checkpoint storage policy in the JAR package, because the settings there will overwrite those in advanced parameters.

Setting more options

Flink provides many other options. For a complete list, see the Flink documentation.

Note

Not all options are supported in Stream Compute Service. Before making any adjustment, please carefully read the following use limits and fully understand the relevant issues and risks to avoid unstable job running, failure to start a job, or other events due to inappropriate parameter adjustments.

Use limits

The following parameters are set by the Stream Compute Service system and cannot be modified. Please do not pass them in through advanced parameters.

Non-customizable Parameters
kubernetes.container.image
kubernetes.jobmanager.cpu
taskmanager.cpu.cores
kubernetes.taskmanager.cpu
jobmanager.heap.size
jobmanager.heap.mb
jobmanager.memory.process.size
taskmanager.heap.size
taskmanager.heap.mb
taskmanager.memory.process.size
taskmanager.numberOfTaskSlots
env.java.opts (you can customize another two separate parameters: `env.java.opts.taskmanager` and `env.java.opts.jobmanager`)

Setting the Maximum Parallelism of a Job

Last updated：2023-11-07 18:18:45

Each Flink job has an attribute called **maximum parallelism** (MaxParallelism). It determines the maximum degree of parallelism and specifies the upper limit for dynamic scaling.
This parameter involves the most underlying state allocation logic in Flink, so it cannot be changed once set. If you must change this parameter (you want to scale out the cluster to CUs exceeding `MaxParallelism`, for example), Flink has to discard the current runtime state of the job and restart it.
In other words, when the job is recovered from checkpoint (a checkpoint taken when Checkpoint or Savepoint is triggered), the new maximum parallelism of operators specified cannot exceed this value. Otherwise, Flink will raise an exception and stop starting the job. If the Flink job is not started from checkpoint, it can still be properly started.

Parameter description

By default, this configuration option has been set in **Job parameters > Advanced parameters** of a new job draft, and you do not need to change it. If you delete this configuration option, the maximum parallelism defaults to `2048`.
Note
For jobs created before April 14, 2021 (the date when the Flink version was updated), the maximum parallelism defaults to `128`. To increase their maximum parallelism, you need to manually change the value of this parameter, and the system will discard the existing runtime state and restart the jobs.
In most cases, you can ignore the setting of the maximum parallelism. You need to set this parameter only when the maximum parallelism of any operator in the job exceeds `MaxParallelism`, or when you want to explicitly restrict the maximum scaling capabilities of the job.

Notes

The minimum value of `pipeline.max-parallelism` is the maximum parallelism of all operators in the job. For example, if a job has 5 operators, whose parallelism is `[1, 5, 100, 2, 2]`, respectively, the minimum value of `pipeline.max-parallelism` allowed is `100`.
In Stream Compute Service, the maximum value of `pipeline.max-parallelism` is fixed to `16384`.
However, we recommend you maintain `pipeline.max-parallelism` at 2048 or lower to avoid unnecessary runtime overhead or reduced processing capabilities of the job.

Configuring Job Resources

Last updated：2023-11-08 10:05:51

Overview

You can configure the JobManager spec, TaskManager spec, and default operator parallelism in **Job parameters** > **Resources**. In the **Resources** section, you can configure compute resources suitable for your job.
Stream Compute Service provides three CU specs: 0.5 CU, 1 CU, and 2 CUs. The JobManager spec and TaskManager spec can be set to different values. 0.5 CU represents fine-grained resources, which are available only to some new clusters by default. The JobManager and TaskManager spec options are not displayed for existing clusters. If you need to configure these two options for an existing cluster, please contact us.

CUs used by a job

CUs used by job = JobManager spec + TaskManager spec x maximum parallelism of all operators in job.
You can adjust the specs and the default operator parallelism based on the maximum number of CUs available to the job given in the **Resources** section. For example, the figure below shows that a maximum of 8 CUs are available to the current job, so if you set the TaskManager spec to 2 CUs, the default operator parallelism can be set to 3 at most.

Resources for scenarios with low resource consumption

Some jobs may have no data sync most of the time, and using 1-CU resources causes a waste of resources. For such jobs, you can adjust the JobManager and TaskManager specs to 0.5 CU (1 CU in total) and the default operator parallelism to 1, reducing waste.

FAQs

1. What are fine-grained resources?
Fine-grained resources refer to resources whose compute unit specs can be smaller than 1 CU (1 core and 4 GB memory). Stream Compute Service currently provides three CU specs: 0.5 CU, 1 CU, and 2 CUs. The JobManager and TaskManager specs can be set to different values.
2. Why does the actual job parallelism fail to reach the maximum parallelism of the job?
If fine-grained resources are used, there is a remote possibility that resource fragmentation affects the job running,

leading to an actual parallelism of the job smaller than the maximum one. In this case, you can select appropriate resource specs to avoid resource fragmentation as much as possible. If you need any help, please contact us.

Managing Versions

Last updated：2023-11-08 10:07:24

The version management feature allows you to roll a job back to a historical version.
On the **Development & Testing** page, click **Manage version** to view all published versions of the current job in the side pop-up window. Here, you can click a version to view job details in this version, but the version is not editable. In the **Manage version** pop-up window, you can delete unnecessary versions or compare versions to identify their differences in code and configurations.

Changing the job version

To change a job to a published version, you need to stop its online running version first, and then click **Run version** to run the target version. You need to select a running policy based on the job status, without entering the version description, and then click **Confirm**.

Comparing versions

You can compare versions to identify their differences and select a version to run. The version comparison feature enables you to compare the SQL code and configurations in different versions of a SQL job.
In the **Manage version** pop-up window, select a version as the initial version, and click **Compare**.
In the **Compare versions** pop-up window, view the code of the initial version. Select a version on the left to compare it with the current (initial) version.
The code differences are highlighted for easy view.
Click **Configuration** to view configuration differences of the versions.

Deleting a version

In the **Manage version** pop-up window, click **Delete** for the target version to delete it (unable to delete a running version).

Monitoring Jobs
Viewing Monitoring Information

Last updated：2023-11-07 18:05:17

Overview

For a running (or once run) streaming job, you can view its monitoring information with the following two methods.

Viewing in the console

Log in to the Stream Compute Service console, click the name of the target job, and click the **Monitoring** tab to view key job metrics, such as incoming data records, outgoing data records, operator computing time, CPU utilization, and heap memory utilization.
Beta features: For major regions such as Beijing, Guangzhou, and Shanghai, fine-grained metrics are available on the **Monitoring** page, such as by Job, TaskManager, or Task.

Viewing on Tencent Cloud Observability Platform (TCOP)

On the job list page in the console, click **Tencent Cloud Observability Platform** on the right to go to the TCOP console and view monitoring metrics in detail there. You can also configure job-specific alarm policies there.
Note
Stream Compute Service also supports monitoring Flink metrics with Prometheus, which allows you to save, analyze, and display various job metrics.

Illustrations of the Stream Compute Service console

On the **Jobs** page in the console, you can view the running of jobs.
Take **high_cpu** in the above figure as an example. Click **Job name** to go to the details page.
On the **Overview** page of the **Monitoring** tab, select a time range.
Default available ranges include last hour, last day, and last 7 days. You can set a custom time range.
Two sampling granularity options are available: **1-minute granularity** and **5-minute granularity**, with smoother curves formed at the latter.

Metrics available on the overview page

The most critical runtime metrics of the job are provided on the overview page, such as incoming data records, outgoing data records, operator computing time, sink watermark delay (based on the current timestamp), job restarts,

TaskManager CPU utilization, TaskManager heap memory utilization, and TaskManager old GC time and count, helping you quickly identify common job exceptions.

Checkpoint metrics (in beta)

Note

Checkpoint metrics are in beta testing and available only for Guangzhou, Beijing, and Shanghai. For other regions, please stay tuned.

After checkpointing is enabled for a Flink job, its runtime information will be saved in checkpoints for recovering the job when necessary. The following metrics are displayed on this page:

Last checkpoint size (Bytes): The size of the last checkpoint.

Checkpoint time (ms): The time taken to make the last checkpoint.

Total checkpoint failures: The total number of checkpoint failures.

JobManager metrics (in beta)

Note

JobManager metrics are in beta testing and available only for Guangzhou, Beijing, and Shanghai. For other regions, please stay tuned.

When a Flink job is started, only one JobManager (JM) will be used. Therefore, the metrics displayed here are those of this JobManager.

JM CPU Load (%): `Status.JVM.CPU.Load` of the JobManager, representing the CPU utilization of the JVM.

JM Heap Memory (Bytes): The heap memory usage of the JobManager.

JM GC Count: `Status.JVM.GarbageCollector.<GarbageCollector>.Count` of the JobManager, representing the GC count of the JobManager.

JM GC Time (ms): `Status.JVM.GarbageCollector.<GarbageCollector>.Time` of the JobManager, representing the GC time of the JobManager.

TaskManager metrics (in beta)

Note

TaskManager metrics are in beta testing and available only for Guangzhou, Beijing, and Shanghai. For other regions, please stay tuned.

When a Flink job is started, one or more TaskManagers will be used, depending on the specified parallelism. All TaskManagers will be displayed in the list. You can select a TaskManager to view its metrics. Available TaskManager metrics include the following:

CPU Load (%): `Status.JVM.CPU.Load` of the TaskManager, representing the CPU utilization of the JVM.

Heap Memory (Bytes): The heap memory usage of the TaskManager.

GC Count: `Status.JVM.GarbageCollector.<GarbageCollector>.Count` of the TaskManager, representing the GC count of the TaskManager.

GC Time (ms): `Status.JVM.GarbageCollector.<GarbageCollector>.Time` of the TaskManager, representing the GC time of the TaskManager.

Pod Memory (Bytes): The memory usage of the Pod where the TaskManager resides. This metric represents the memory usage of the whole Pod, including the JVM heap memory, non-heap direct memory, overhead, and memory of other auxiliary services in the Pod. If the value of this metric is too large, the Pod faces the risk of being killed due to OOM.

Pod CPU (%): The CPU utilization of the Pod where the TaskManager resides. This metric represents the CPU utilization of the whole Pod, including the CPU usage of the JVM and other auxiliary services in the Pod.

Task metrics (in beta)

Note

Task metrics are in beta testing and available only for Guangzhou, Beijing, and Shanghai. For other regions, please stay tuned.

The execution graph of a Flink job contains one or more tasks. You can view the metrics of a task in the graph.

OutPoolUsage: The percentage of output queues. When this metric reaches 100%, the task is backpressured, which needs to be resolved with some methods, such as setting a larger operator parallelism.

OutputQueueLength: The number of output queues.

InPoolUsage: The percentage of input queues. When this metric reaches 100%, the task is backpressured, which needs to be resolved with some methods, such as setting a larger operator parallelism.

InputQueueLength: The number of input queues.

CurrentInputWatermark: The last (minimum) watermark the task has received.

Configuring Monitoring Alarms (Numerical Metrics)

Last updated：2023-11-08 10:19:09

The alarm policies for numerical metric monitoring in Stream Compute Service are implemented through **Tencent Cloud Observability Platform (TCOP)**. This document describes some common scenarios. For more details, see Overview of TCOP.

Note

TCOP no longer allows users to configure event alarms. This feature has been transferred to EventBridge. Please configure alarms of various events as instructed in Configuring alarms of job events.

Viewing job alarm policies

In the TCOP console, select **Alarm Management > Policy Management** to view alarm policies configured for all of your products. Alternatively, enter Stream Compute Service in the search box in the top right corner of the page to view all alarms configured for Stream Compute Service.

Adding a job alarm policy

1. In the TCOP console, select **Alarm Management > Policy Management**, click **Create Policy**, and enter a policy name and remarks (optional).
2. In the **Policy Type** drop-down list, select Stream Compute Service, and select **Alarm Object** as prompted. Here, you can configure the policy for a specific job or for all jobs, and press Shift to select multiple items.
3. Then, select **Trigger Condition**. In Select Templete, select an existing template, or add one. If you don't want to use a template, select **Configure manually**, where you can set thresholds and alarms for monitoring metrics mentioned above.
4. Select **Create Template** in **Notification Template**, set the recipient object, notification cycle, and receiving channel.
5. Click **Complete** to apply this new alarm policy.

Note

Since "Job restarts" and "Total checkpoint failures" are cumulative, use monthly alarm policies for these two metrics.

Configuring job alarms by tag

1. In the Tag console, select **Tag List** > **Create Tag**.
Select a tag key, enter a tag value, and click **OK**.
2. Select the tag just created when creating a job or on the job overview page.
Editing a tag on the job overview page:
After the tag is complete, click **Conform**.

3. Log in to the TCOP console, select **Alarm Management** > **Policy Management** > **Create Policy**, and then select the tag.
4. Configure alarm notifications.

Configuring Event Alarms (Events)

Last updated：2023-11-08 10:19:40

Stream Compute Service supports the detection and display of various events of a job, as detailed here. When an event occurs, it will be pushed to your EventBridge. On the Event Rule page in EventBridge, you can configure event matching rules to receive events from the backend of Stream Compute Service.
At present, EventBridge supports the following receiving channels free of charge: SMS, call, Email, Message Center, WeCom bot, and webhook.
Note
Job events in all regions will be reported to the event bus named "default" in EventBridge in Guangzhou. If a different region or event bus is set when an event rule is configured, event push will fail.
For an account, please do not send more than 1,000 messages within 24 hours to avoid failure to receive alarms after the limit is exceeded.

Viewing job alarm policies

1. On the Event Rule page in EventBridge, select Guangzhou as the region to view all configured rules.
2. Click **Edit** on the right of a rule to view its details, such as the related Tencent Cloud service and delivery method.

Adding a job alarm policy

1. On the Event Rule page in EventBridge, select Guangzhou as the region, and click **Create**.
2. On the **Create event rule** page, enter a rule name and rule description in the **Basic information** section. We recommend you use standard naming rules to identify rules.
3. In the **Event matching** section, set the mode to **Template**, the Tencent Cloud service to **Stream Compute Service**, and the event type to **All Events** or desired events (event names are case-sensitive). In addition, EventBridge supports flexible event patterns as described in Event Pattern. You can match event name and other fields by prefix, suffix, exclusion, or inclusion as needed.
Click **Custom events** or **Edit match rule** to set filter fields in the JSON script of the event matching rule.
Note: In the JSON script of the event matching rule, you can enter specific values following "data" to configure event alarms for jobs by job ID, job directory, job creator, cluster, workspace, or other dimensions, and the event alarms will be triggered accordingly. For more information, see Event Pattern.
The table below describes some common fields.

Field	Description	Example
instanceId	The job ID	cql-xxxxxx

folderId	The ID of the job directory	folder-xxxxxxxx
creatorUin	The UIN of the job creator	123456
clusterId	The ID of the cluster where the job resides	cluster-xxxxxxxx
workSpaceId	The ID of the workspace where the job resides	space-xxxxxxxx

For other fields, see **Event sample** of **Event Type**.

4. In the **Delivery target** section, set the trigger method to **Notification message**, and set recipients, notification period, delivery method, and webhook.

5. Select **Enable event rules now**, and click **Complete** to finish the rule creation.

Monitoring Metric List

Last updated：2023-11-07 18:09:05

The monitoring metric list provides the meanings of all metrics, helping you use the monitoring feature in Stream Compute Service.

Monitoring metric list

Note

You can view the following metrics in TCOP console > Stream Compute Service, and configure alarms here.

Metric	Description	Examp
job_records_in_per_second	The total number of records the job receives from all sources per second.	22478
job_records_out_per_second	The total number of records the job emits to all sinks per second.	12017
job_bytes_in_per_second	The total number of bytes the job receives from all sources (Kafka sources only) per second.	78657
job_bytes_out_per_second	The total number of bytes the job emits to all sinks (Kafka sinks only) per second.	15687
job latency	The total latency it takes the data to flow through all operators. Sample errors may exist, so the value is for reference only.	275 m
job_service_delay	The difference between the current timestamp and the watermark at the sink (if there are multiple sinks, the maximum difference is used).	5432 r
job_cpu_load	The average CPU utilization of all TaskManagers of the job.	23.85%
taskmanager_status_jvm_memory_heap_used_percentage	The average heap memory utilization of all TaskManagers of the job.	57.12%
taskmanager_status_jvm_memory_heap_used	The total heap memory used of all	83089

	TaskManagers of the job.	Bytes
taskmanager_memory_heap_committed	The total heap memory committed of all TaskManagers of the job.	49372 Bytes
taskmanager_memory_heap_max	The total max heap memory of all TaskManagers of the job.	49372 Bytes
taskmanager_status_jvm_memory_nonheap_used	The total non-heap memory (JVM metaspace and code cache) used of all TaskManagers of the job.	29665 Bytes
taskmanager_memory_nonheap_committed	The total non-heap memory (JVM metaspace and code cache) committed of all TaskManagers of the job.	10321 Bytes
taskmanager_status_jvm_memory_nonheap_max	The total max non-heap memory (JVM metaspace and code cache) of all TaskManagers of the job.	78014 Bytes
taskmanager_status_jvm_memory_process_memoryused	The max JVM memory (RSS) of all TaskManagers of the job, including heap, non-heap, native, and other areas. This metric is used to give an early warning for OOM Killed events in a Pod.	35970 Bytes
taskmanager_memory_direct_count	The sum of buffers in the direct buffer pools of all TaskManagers of the job.	10993
taskmanager_memory_direct_used	The total direct buffer pools used of all TaskManagers of the job.	36032 Bytes
taskmanager_memory_direct_max	The total max direct buffer pools of all TaskManagers of the job.	36032 Bytes
taskmanager_memory_mapped_count	The sum of buffers in the mapped buffer pools of all TaskManagers of the job.	4 Item
taskmanager_memory_mapped_used	The total mapped buffer pools used of all TaskManagers of the job.	33554
taskmanager_memory_mapped_max	The total max mapped buffer pools of all TaskManagers of the job.	33554
jobmanager_jvm_old_gc_count	The old GC count of the JobManager of the job.	3.00 T

jobmanager_jvm_old_gc_time	The old GC time of the JobManager of the job.	701.0(
jobmanager_jvm_young_gc_count	The young GC count of the JobManager of the job.	53.00
jobmanager_jvm_young_gc_time	The young GC time of the JobManager of the job.	4094.(
job_lastcheckpointduration	The time taken to make the last checkpoint of the job.	723.0(
job_lastcheckpointsize	The size of the last checkpoint of the job.	75132
taskmanager_jvm_old_gc_count	The **sum** of old GC counts of all TaskManagers of the job.	9.00 T
taskmanager_jvm_old_gc_time	The sum of old GC time of all TaskManagers of the job.	2014.(
taskmanager_jvm_young_gc_count	The **sum** of young GC counts of all TaskManagers of the job.	889.0(
taskmanager_jvm_young_gc_time	The sum of young GC time of all TaskManagers of the job.	15051
job_numberofcompletedcheckpoints	The number of successful checkpoints of the job.	11.00
job_numberoffailedcheckpoints	The number of failed checkpoints of the job.	1.00 T
job_numberofinprogresscheckpoints	The number of checkpoints in progress (not completed) of the job.	1.00 T
job_totalnumberofcheckpoints	The total number of checkpoints (in progress, completed, and failed) of the job.	13.00
job_numrecordsinbutfailed	The number of failed records (such as raising various exceptions) in the operator. If its value is greater than 1, the semantics of Exactly-Once will be affected. It is a testing parameter for reference only.	0.00 T
jobmanager_job_numrestarts	The recorded number of job restarts	10.00

	due to crash (excluding restart of the job after the JobManager exits) of the JobManager of the job.	
jobmanager_status_jvm_memory_heap_used_percentage	The heap memory utilization of the JobManager of the job.	31.34%
jobmanager_memory_heap_used	The heap memory used of the JobManager of the job.	10400 Bytes
jobmanager_memory_heap_committed	The heap memory committed of the JobManager of the job.	33182 Bytes
jobmanager_memory_heap_max	The max heap memory of the JobManager of the job.	33182 Bytes
jobmanager_status_jvm_memory_nonheap_used	The non-heap memory (JVM metaspace and code cache) used of the JobManager of the job.	11736 Bytes
jobmanager_memory_nonheap_committed	The non-heap memory (JVM metaspace and code cache) committed of the JobManager of the job.	12218 Bytes
jobmanager_status_jvm_memory_nonheap_max	The max non-heap memory (JVM metaspace and code cache) of the JobManager of the job.	78014 Bytes
jobmanager_status_jvm_memory_used	The JVM memory used (RSS) of the JobManager of the job, including heap, non-heap, native and other areas. This metric is used to give an early warning for OOM Killed events in a Pod.	35970 Bytes
jobmanager_cpu_load	The CPU utilization of the JobManager of the job.	7.12%
jobmanager_cpu_time	The CPU service time (ms) of the JobManager of the job.	83449
jobmanager_downtime	For a non-running (failed or recovering) job, the duration of this downtime; for a running job, the value of this metric is 0.	10884
job_uptime	For a running job, the duration of continuous running of this job without interruption.	20230

job_restartingtime	The time taken for the last restart of the job.	19718
jobmanager_lastcheckpointrestoretimestamp	The Unix timestamp of the last job recovery from checkpoint (in ms), whose value will be `-1` if no recovery is performed.	16219 ms
jobmanager_memory_mapped_count	The number of buffers in the mapped buffer pool of the JobManager of the job.	4.00 It
jobmanager_memory_mapped_memoryused	The mapped buffer pool used of the JobManager of the job.	33554
jobmanager_memory_mapped_totalcapacity	The max mapped buffer pool of the JobManager of the job.	33554
jobmanager_memory_direct_count	The number of buffers in the direct buffer pool of the JobManager of the job.	22.00
jobmanager_memory_direct_memoryused	The direct buffer pool used of the JobManager of the job.	57576
jobmanager_memory_direct_totalcapacity	The max direct buffer pool of the JobManager of the job.	57781
jobmanager_numregisteredtaskmanagers	The number of registered TaskManagers of the job, which is generally equal to the max operator parallelism. The decline in the number of TaskManagers indicates that some TaskManagers are disconnected, and the job may crash and try to recover.	3.00 TaskM
jobmanager_numrunningjobs	The number of running jobs, with `1` for proper job running and `0` for job crash.	1.00 J
jobmanager_taskslotsavailable	The number of task slots available, with `0` for proper job running and a value other than `0` for possible non-running of the job for a short period of time.	0.00 S
jobmanager_taskslotstotal	In Stream Compute Service, a TaskManager has only one task slot, so	3.00 S

	the total number of task slots is equal to the number of registered TaskManagers.	
jobmanager_threads_count	The number of active threads in the JobManager of the job, including daemon and non-daemon threads.	77.00
taskmanager_cpu_time	The CPU service time (ms) of all TaskManagers of the job.	20292
taskmanager_network_availablememorysegments	The sum of memory segments available in all TaskManagers of the job.	32890
taskmanager_network_totalmemorysegments	The sum of total memory segments assigned to all TaskManagers of the job.	32931
taskmanager_threads_count	The total number of active threads in all TaskManagers of the job, including daemon and non-daemon threads.	207.0(
job_lastcheckpointsize	The size of the last checkpoint.	1,024
job_lastcheckpointduration	The time taken to make the last checkpoint.	100ms
job_numberoffailedcheckpoints	The number of failed checkpoints.	50 Byt
JM CPU Load	The JVM CPU utilization of the JobManager.	12%
JM Heap Memory	The heap memory usage of the JobManager.	50 Byt
JM GC Count	`Status.JVM.GarbageCollector.<GarbageCollector>.Count` of the JobManager, representing the GC count of the JobManager.	5 time
JM GC Time	`Status.JVM.GarbageCollector.<GarbageCollector>.Time` of the JobManager, representing the GC time of the JobManager.	64ms
TaskManager CPU Load	The JVM CPU utilization of the selected TaskManager.	70%

TaskManager Heap Memory	The heap memory usage of the selected TaskManager.	50 byt
TaskManager GC Count	`Status.JVM.GarbageCollector.<GarbageCollector>.Count` of the selected TaskManager, representing the GC count of the TaskManager.	5 time
TaskManager GC Time	`Status.JVM.GarbageCollector.<GarbageCollector>.Time` of the selected TaskManager, representing the GC time of the TaskManager.	5ms
Task OutPoolUsage	The percentage of output queues. When this metric reaches 100%, the task is backpressured.	64%
Task OutputQueueLength	The number of output queues.	6
Task InPoolUsage	The percentage of input queues. When this metric reaches 100%, the task is backpressured.	64%
Task InputQueueLength	The number of input queues.	6
Task CurrentInputWatermark	The current watermark of the task.	16238
Data import time (ETL)	The delay of a source taking the data in the job.	10 ms
job_records_in_per_second (ETL)	The total rate of all sources in the job.	342 R
SourceIdleTime (ETL)	The interval between data batches processed by a source in the job, which indirectly reflects the idle time of the source.	24532
SynDelay (ETL)	The delay of a source taking the data and processing it in the job.	1345 r
BinLogPos (ETL)	The MySQL binary log coordinates or PostgreSQL log sequence number (LSN) of the job.	26069
job latency (ETL)	The average delay between the sink and source operators of the job.	49 ms

DbFlushDelay (ETL)	The sum of the database flush delay and async callback time of the job.	30 ms
job_records_out_per_second (ETL)	The total rate of all sinks in the job.	234 R
Source - full sync (ETL)	The full data sync progress of the job.	30%
Source - incremental sync (ETL)	For MySQL, sync delay refers to the gap between the binlog coordinates of the current source and the latest binlog coordinates of the MySQL instance source collected in the last sampling; for PostgreSQL, sync delay refers to the gap between the LSN of the current source and the latest LSN of the PostgreSQL instance source collected in the last sampling.	205
Kafka - records_lag max	The maximum of kafka-lag-max (the difference of Kafka producer and consumer offsets) reported by the TaskManager.	100
Kafka - records_lag min	The minimum of kafka-lag-max (the difference of Kafka producer and consumer offsets) reported by the TaskManager.	50
Kafka - records_lag mean	The mean of kafka-lag-max (the difference of Kafka producer and consumer offsets) reported by the TaskManager.	80
Kafka - records_lag sum	The sum of kafka-lag-max (the difference of Kafka producer and consumer offsets) reported by the TaskManager.	500
CurrentFetchEventtimeLag (ms)	Formula: FetchTime (the time the source fetches the data) – EventTime (data event time). This metric reflects the retention of data in the external system.	10
CurrentEmitEventtimeLag (ms)	Formula: EmitTime (the time the data leaves the source) – EventTime (data event time). This metric reflects the	20

	retention of data between the external system and the Source.	
taskmanager_job_task_backpressuredtimemspersecond (%)	The maximum of all subtask backpressure percentages in the job.	30%
taskmanager_job_task_dataskewcoefficient	This metric is the coefficient of variation (= standard deviation/mean) of subtask inputs of each job. A value less than 10% represents a weak skew.	10%

Connecting to Prometheus

Last updated：2023-11-07 17:56:22

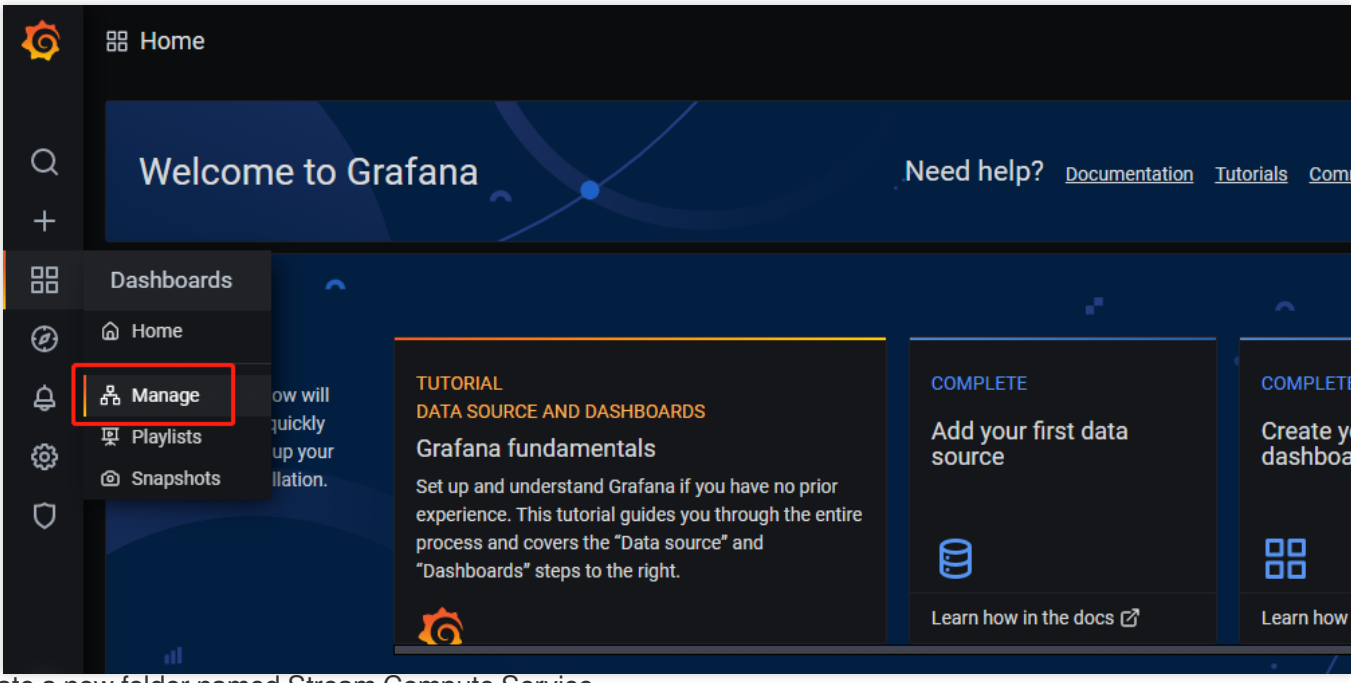
About Prometheus

Prometheus is a flexible time series database generally used for monitoring data storage, computing, and alerting. With the Pushgateway service provided by Prometheus, you can push Built-in Flink metrics and custom business metrics to your self-built or TencentCloud Managed Service for Prometheus server, so that you can group, aggregate, and display these metrics in Grafana.
We recommend you use TencentCloud Managed Service for Prometheus in Tencent Cloud Observability Platform (TCOP) to save you on deployment and Ops costs. Meanwhile, Notification Template is available, allowing alarm notifications to be sent to recipients via SMS, call, Email, WeCom bot, and other methods.

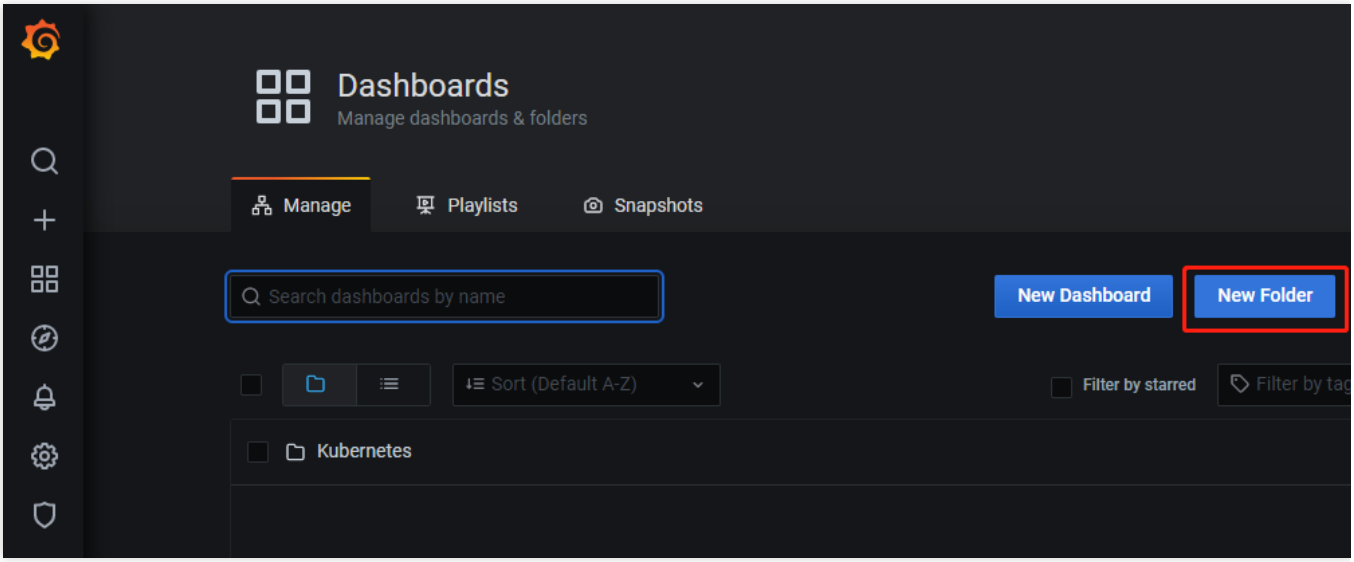
Importing Grafana dashboard for Stream Compute Service

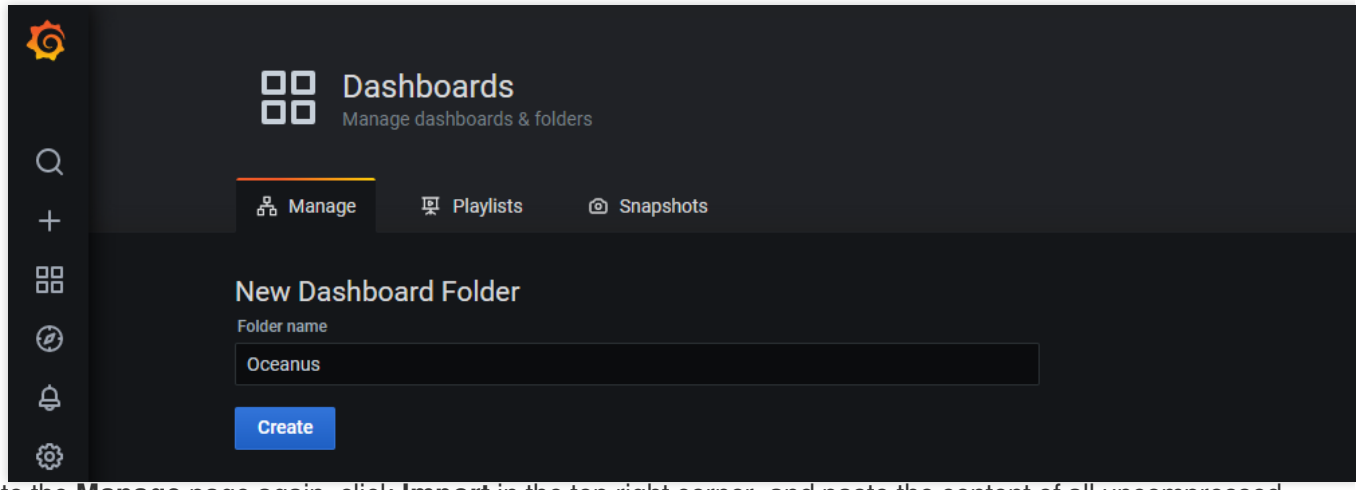
1. Download the Grafana dashboard template here and unzip it to the local system.
2. In the Grafana dashboard of Prometheus, click

>  **Manage** on the left sidebar.



3. Create a new folder named Stream Compute Service.



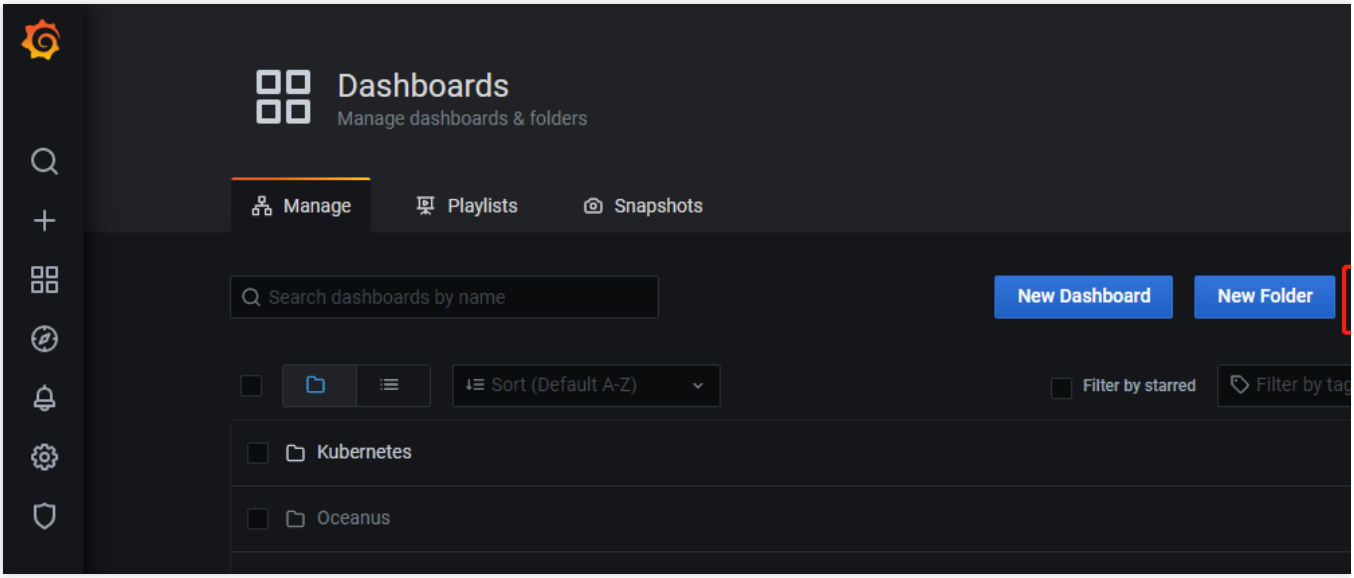


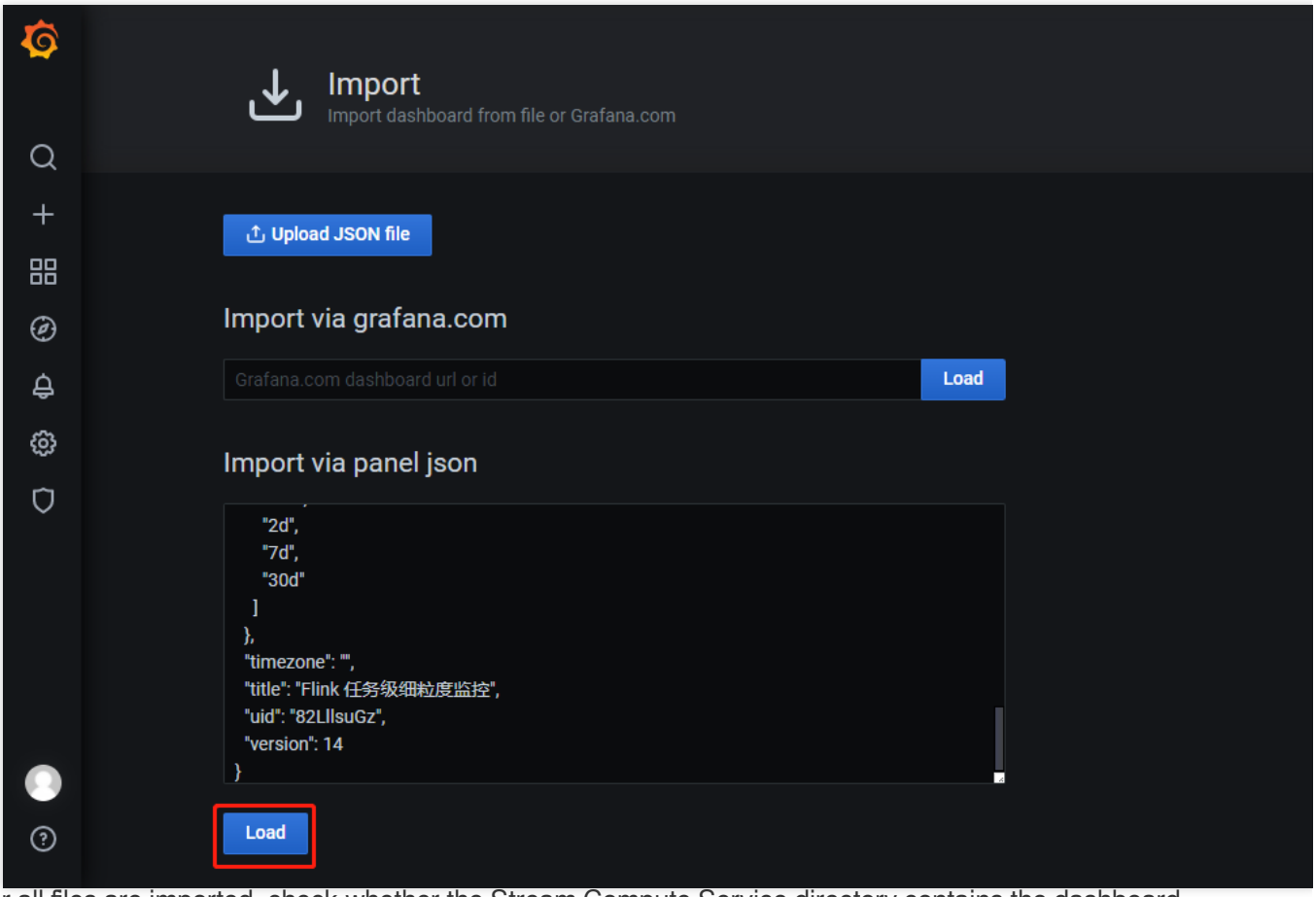
4. Go to the **Manage** page again, click **Import** in the top right corner, and paste the content of all uncompressed JSON files **one by one**.

Note

Import each JSON file as instructed below.

Don't change the dashboard UID (don't click **Change uid**) to avoid failures of links for redirecting between dashboards.





5. After all files are imported, check whether the Stream Compute Service directory contains the dashboard.

Enabling Prometheus metric reporting for a job

Note

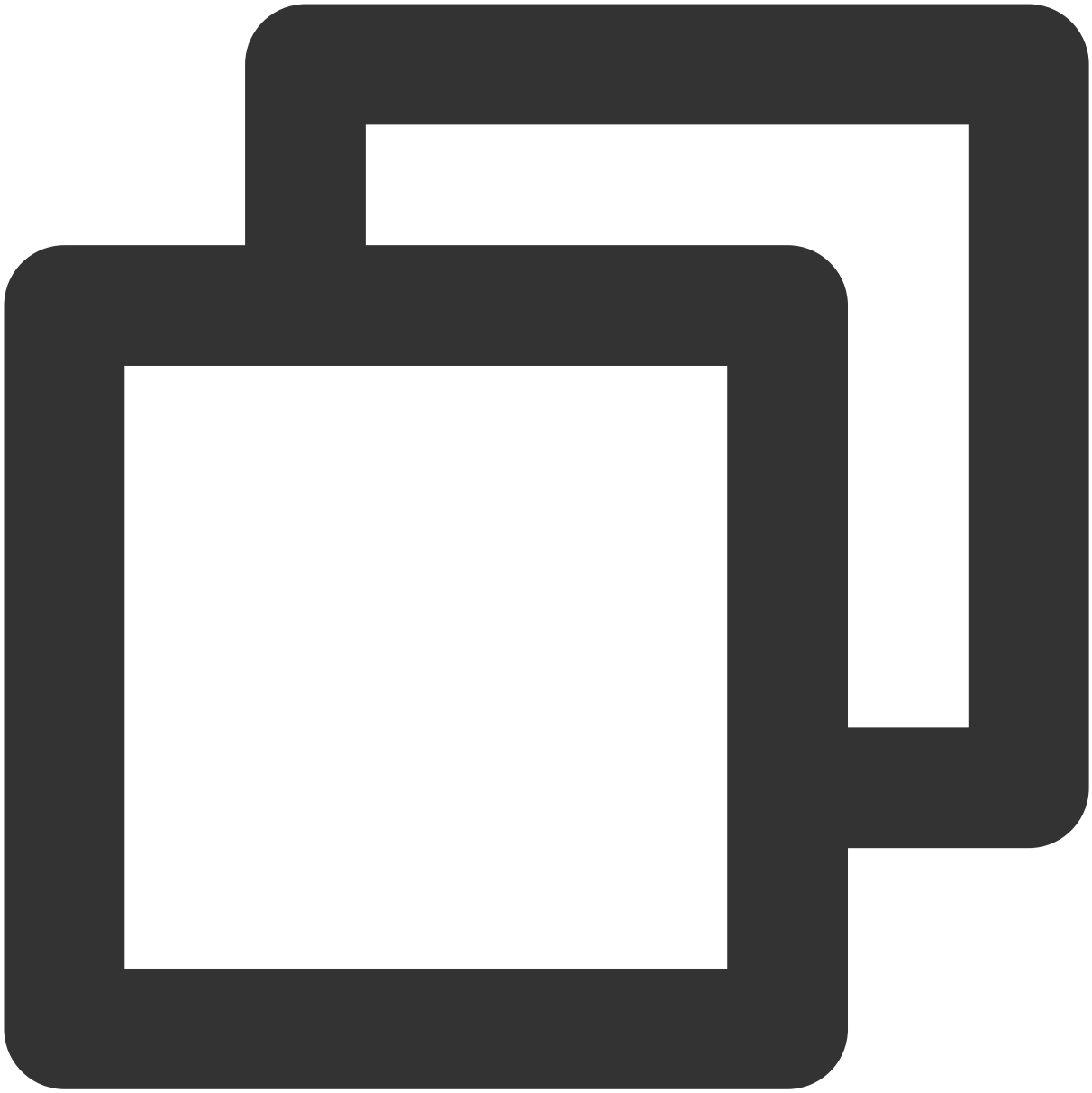
You need to separately configure Prometheus metric reporting for each job.

After the configurations of a job are modified, you must click **Publish draft** and **Run version** to restart the job before monitoring data is reported.

1. In the Stream Compute Service console, click a target job to go to its **Development & Testing** page.
2. Click **Job parameters**, and add the following in the advanced parameters.

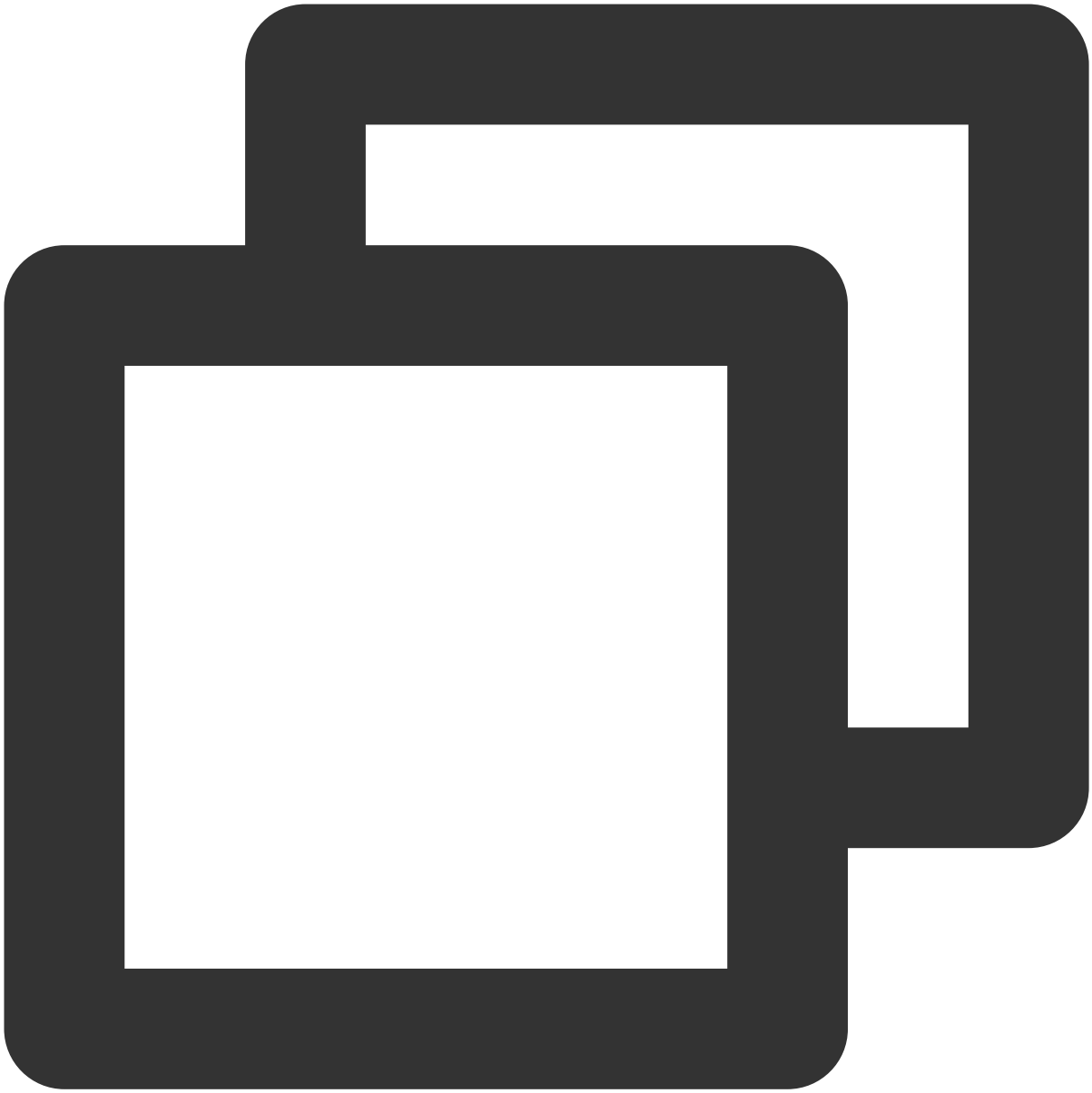
Note

The variables in ${ } need to be replaced with your actual values.



```
metrics.reporters: promgateway
metrics.reporter.promgateway.host: ${Prometheus PushGateway IP}
metrics.reporter.promgateway.port: ${Prometheus PushGateway port}
```

If TencentCloud Managed Service for Prometheus provided by TCOP is used, the following authentication information (password is the token shown in the console) is also required:



```
metrics.reporter.promgateway.needBasicAuth: true
metrics.reporter.promgateway.password: ${Prometheus password}
```

3. Publish and start the job with new configurations, wait about 1 minute (reporting period), and view data in the dashboard.

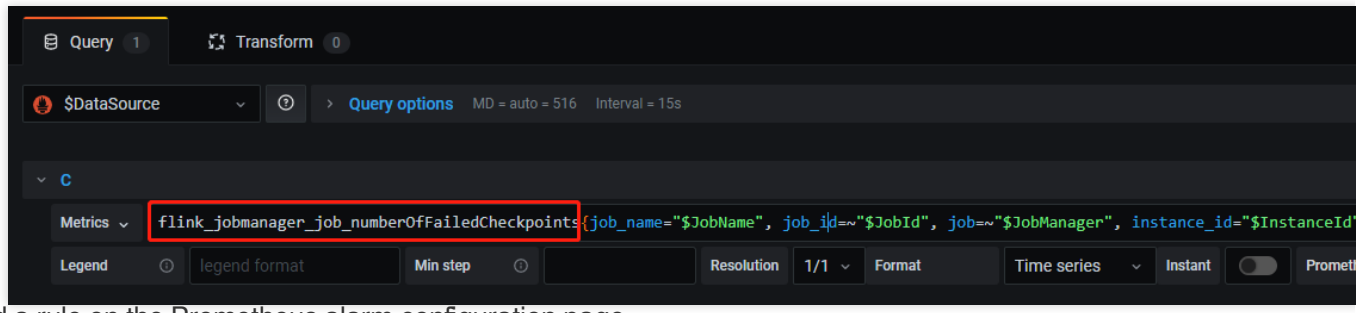
4. Edit the Prometheus dashboard to meet your specific monitoring requirements.

Configuring alarms

To configure an alarm based on a metric, `job_numberoffailedcheckpoints`, for example, and display the configured alarm policy in TencentCloud Managed Service for Prometheus, follow these steps:

1. Select a target metric (here is `job_numberoffailedcheckpoints`) on the dashboard.

2. View the query condition of the metric on its editing page.



3. Add a rule on the Prometheus alarm configuration page.

Note

In the rule PromQL, do not enter any Grafana variable shown above in { }, such as `instance_id="$InstanceId"`. If filters are required, enter specific values in { }, such as `instance_id="cql-abcd0012"`.

A tag in the source (such as job_id) can be referenced in the alarm object and alarm message, such as `{{ $labels.job_id }}`, and the value of the query statement can be expressed as `{{ $value }}`.

4. When an alarm is triggered or cleared, a notification will be sent via the specified receiving channels. In addition, you can configure other receiving channels such as SMS, call, and Email in Notification Template.

Note

The following shows notifications sent via the WeCom bot

Viewing the Flink UI of a Job

Last updated：2023-11-07 17:45:54

You can view the Flink UI (the native Flink dashboard) of a running job via the following entry:

1. Log in to the Stream Compute Service console, and select **Jobs**, click the name of the target job, and click **Flink UI > Go to Flink UI**.
2. Log in to the Stream Compute Service console, select a target job, click its **Name** to go to its details page, and click **Flink UI** in the top right corner of this page.

Clicking **Go to Flink UI** will open a new tab, where you need to enter your username and password to visit the Flink UI page. The username is admin, and the password is the Flink UI password set when you create the cluster. If you have forgotten the password, reset it on the login page or in **Compute resources**.

Job Logs

Configuring Running Log Collection for a Job

Last updated：2023-11-07 17:13:06

You can set one or more CLS log topics for a Stream Compute Service cluster, so that you can view and search for logs on the log page of a job and select them for use by running logs of different jobs. You can also store job running logs of a Stream Compute Service cluster in COS, which, compared with CLS, costs less, but does not allow you to view and search for logs in real time.

Associating CLS with a cluster

On the details page of a cluster, you can configure CLS logsets and topics (3 at most) for the cluster.

Creating logsets and log topics

In the Stream Compute Service console, you can create your dedicated logsets and log topics (all prefixed with "Oceanus_"). When you need to select a logset, you can select an existing logset or create one.
Log topics created in the Stream Compute Service console contain special system settings and are not recommended for other products.

Selecting an existing log topic

When associating a log topic with a Stream Compute Service cluster, you can select an existing log topic, but it must be one created in the Stream Compute Service console.

Setting a default log collection method for a cluster

You can set a default log collection method for a cluster to simplify the setting of job parameters. A new job created after the default method is set will use the logset and log topic in the default configurations.

Setting a running log collection method for a job

For log collection with CLS, since multiple CLS log topics can be associated with a cluster, the parameter configurations of a new job need to specify the log topic to which the running logs of this job will be reported, or specify a default log collection method.

For log collection with COS, job logs will be stored in the COS bucket you associate with the cluster during its creation, and the log storage path is `job-running-log/` .

Setting a running log level for a job

You can set the log level of a job on its parameters page. Four log level options are available: `DEBUG` , `INFO` , `WARN` , and `ERROR` . After a log level is set, the logs of the job will be output at this level. If no log level is available for selection, submit a ticket to upgrade your cluster first.

Batch changing log levels of jobs

To batch changing the log levels of multiple jobs, select **Jobs > Batch > Change log level** in the console, select a log level (options: `DEBUG` , `INFO` , `WARN` , and `ERROR`), and click **Confirm**. After this change, the logs of these jobs will be output at the new level. If no log level is available for selection, submit a ticket to upgrade your cluster first.

Events and Diagnosis
Diagnosis with Logs

Last updated：2023-11-07 17:52:44

Overview

In the Stream Compute Service console, two categories of logs are available: **start logs** and **running logs**.
Start logs: When SQL, JAR, or other types of jobs are submitted in a cluster, the startup process of generating a Flink execution graph starts first, and logs generated in this process are referred to as start logs. When a job fails to start, a yellow triangle with an exclamation mark point (⚠) will appear next to its name in the console, over which you can move the pointer to view details. You can also read the log context of the errors on the logs page.
Running logs: After the execution graph of a job is generated, its JobManager and TaskManagers will be started, and the execution graph will be submitted to the cluster for execution. From this point, the job status becomes "running", and logs printed by the JobManager and TaskManagers are called running logs.

Keywords of common exceptions

Job failure causes

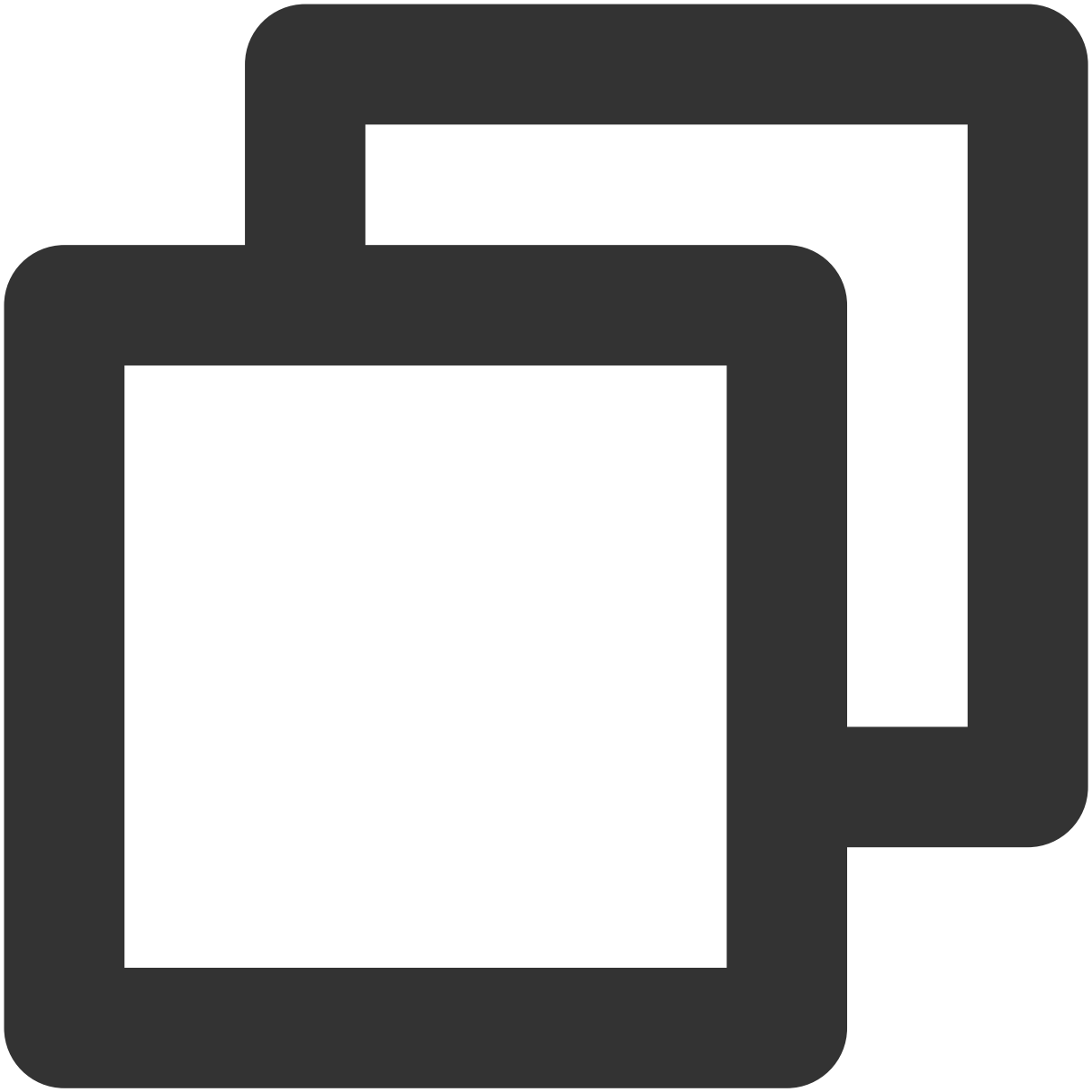
You can search by `from RUNNING to FAILED` to identify the direct cause of a job crash, and the information following `Caused by` in the stack trace represents the failure details.

OOM

If `java.lang.OutOfMemoryError` appears, it is probably that OOM has occurred in the heap memory. In this case, you need to increase the operator parallelism (CUs) of the job and optimize the memory usage to avoid OOM.

JVM exit and other fatal errors

The following keywords are generally followed by a process exit code and can help identify fatal JVM or Akka errors that cause a JVM to be forcibly shut down.

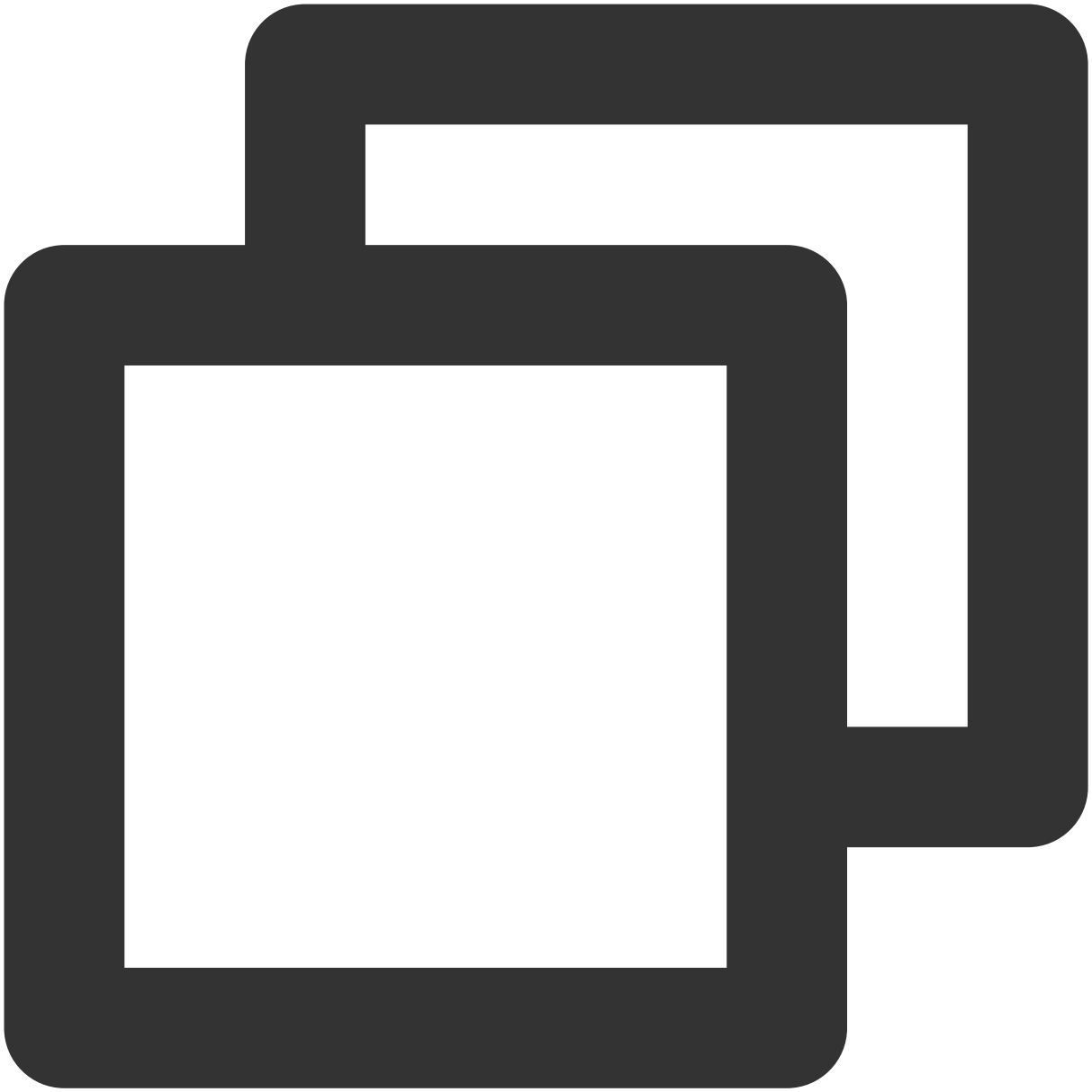


```
exit code OR shutting down JVM OR fatal OR kill OR killing
```

For example, the fatal error of ZooKeeper connection loss shown in the figure below hits the keyword `fatal` .

Checkpoint failure (timeout)

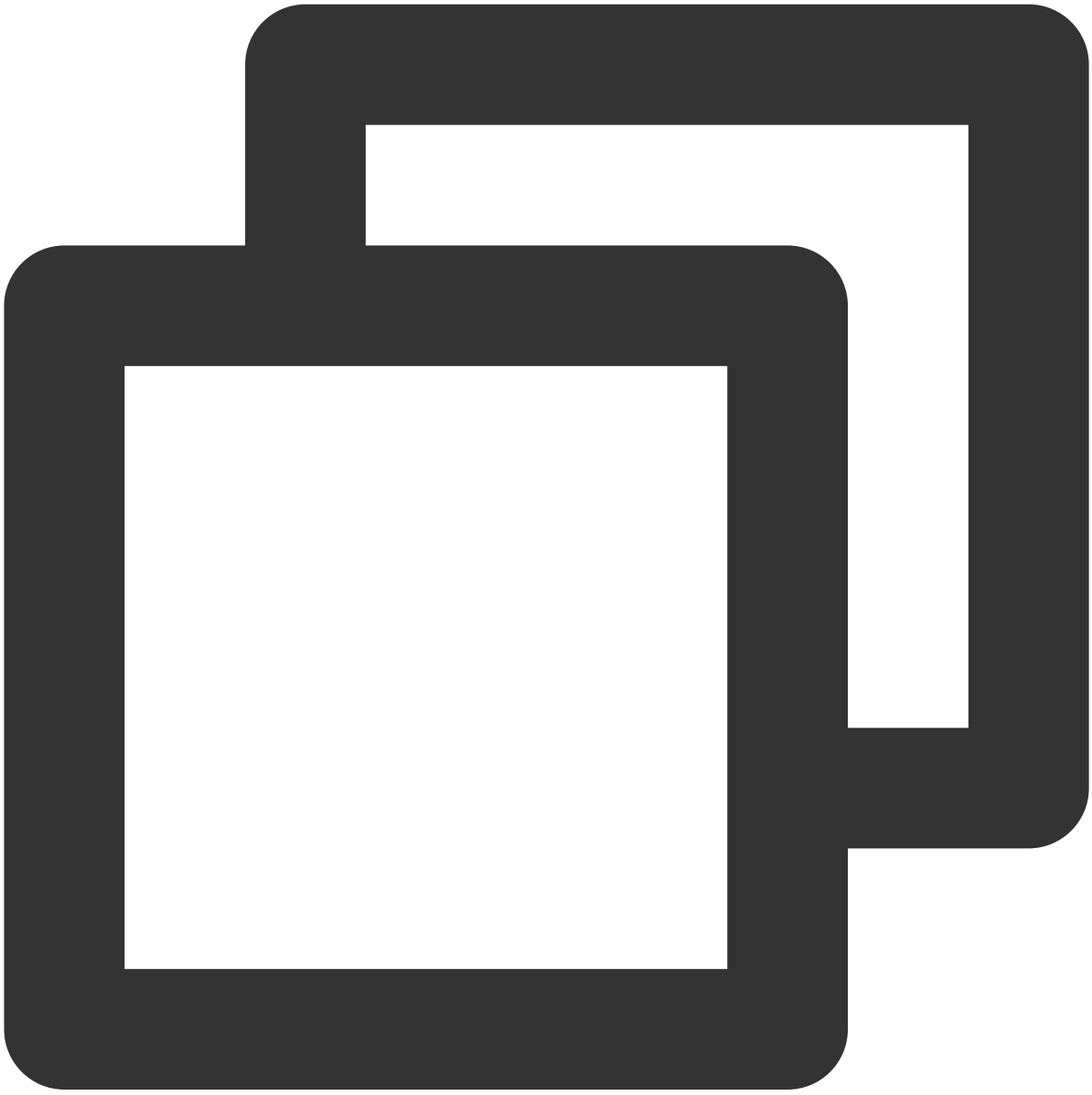
The following keywords indicate that a checkpoint fails. In this case, please analyze the issue based on the specific causes. For example, `declined` represents a checkpoint failure due to resource unavailability (the job is not running), the existence of `FINISHED` operators, checkpoint timeout, incomplete checkpoint files, or other reasons.



```
Checkpoint was declined
Checkpoint was canceled
Checkpoint expired
job has failed
Task has failed
Failure to finalize
```

Timeout/Failure

The following keywords indicate that an access timeout may occur to an external system (such as MySQL or Kafka) due to network failure or other reasons. **The results provided may contain much configuration content. Please check whether this represents an error.** For example, `Timeout expired while fetching topic metadata` for Kafka represents an initialization timeout, and `Communications link failure` for MySQL represents disconnection (which may be a client timeout due to no data inflow for a long period).



```
java.util.concurrent.TimeoutException
timeout
failure
timed out
```

```
failed
```

Exception

`Exception` indicates that an exception may have occurred. For example, the **start logs** of a Flink job in the following figure indicates that the job fails to be submitted due to an exception. Search by `Exception` will display specific exceptions following `Caused by` in the stack traces at all levels.

Note

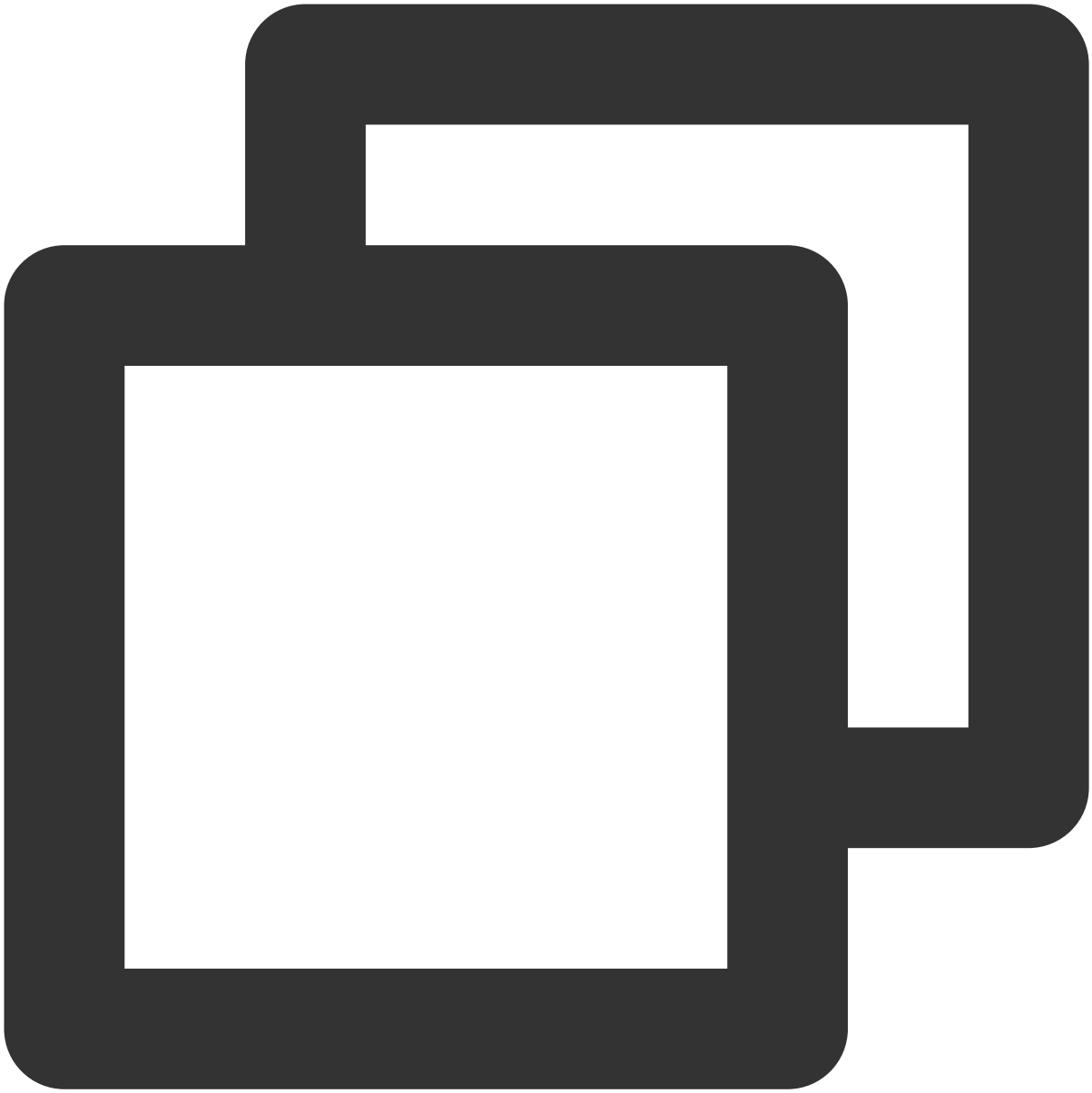
Not all logs containing `Exception` can be found by search due to keyword segmentation rules.

WARN and ERROR logs

In general, you can search for all logs containing `WARN` or `ERROR`, where many results may be found. Please filter the information as needed. For example, some logs may contain `WARN` and `ERROR` themselves and do not represent errors.

Ignorable errors

The following common errors in the Stream Compute Service logs do not affect the running of jobs and can be skipped during troubleshooting:



```
WARN  org.apache.flink.core.plugin.PluginConfig - The plugins directory [plugins] d
WARN  org.apache.flink.shaded.zookeeper3.org.apache.zookeeper.ClientCnxn - SASL con
ERROR org.apache.flink.shaded.curator4.org.apache.curator.ConnectionState - Authent
WARN  org.apache.hadoop.util.NativeCodeLoader - Unable to load native-hadoop librar
WARN  org.apache.flink.kubernetes.utils.KubernetesInitializerUtils  - Ship director
WARN  org.apache.flink.configuration.GlobalConfiguration - Error while trying to sp
```

```
WARN  org.apache.flink.shaded.curator4.org.apache.curator.utils.ZKPaths - The versi

WARNING: Unable to load JDK7 types (annotations, java.nio.file.Path): no Java7 supp
```

Viewing Critical Events

Last updated：2023-11-07 16:38:39

Overview

Various events may occur during the running of a job, such as start, job running failure, checkpointing failure, and other exceptions. A comprehensive events page is provided in the Stream Compute Service console, allowing you to view and subscribe to critical events.

On the events page, you can select a target event type, and further filter events by running instance ID and time range. You can click **Reset filter** to clear filters and reset to defaults and pull the latest events.

Note

To avoid the generation of too many events, the max time range for filtering is limited to 7 days within the past 90-day period.

Event types

Job start and stop

When you click **Publish draft** on the **Development & Testing** page of a job, or when the job exits due to crash and the event is detected, the system will try to start the job, and automatically create a new instance ID for this run. Later, you will see a new start event on the events page. When you stop or restart the job, or it crashes and exits, a stop event with the above-mentioned instance ID will be generated. **The job start time and end time refer to the time points when the internal process of the job is completed, but not the time points when you operate on the UI.**

For example, the information in the figure below indicates that the instance is started on 2021-11-10 16:49:30 and stopped on 2021-10-10 16:55:52 by you or the system.

Job running failure and recovery

When a job is restarted during its running (its status changes from `RUNNING` to `RESTARTING` or `FAILED`), a job running failure event will be generated. If the job is `RUNNING` again, a failed job recovery event will be generated.

You can select **Operation** > **Solution** to view causes of and solutions to the event. You can also configure alarms for job running failure events.

Checkpointing failure and recovery

If checkpointing is enabled for a job, and a checkpoint fails to be taken, a checkpoint failure event will be generated. If the checkpoint succeeds later, a failed checkpoint recovery event will be generated.
You can select **Operation > Solution** to view causes of and solutions to the event. You can also configure alarms for checkpoint failure events.

Job exception events (in beta)

The Stream Compute Service backend continuously monitors and analyzes the running of jobs. When a job encounters severe exceptions (such as TaskManager full GC too long, CPU load too high, and abnormal Pod exit), the corresponding events will be pushed to you for reference, so that you can determine whether the job is properly running.

Note

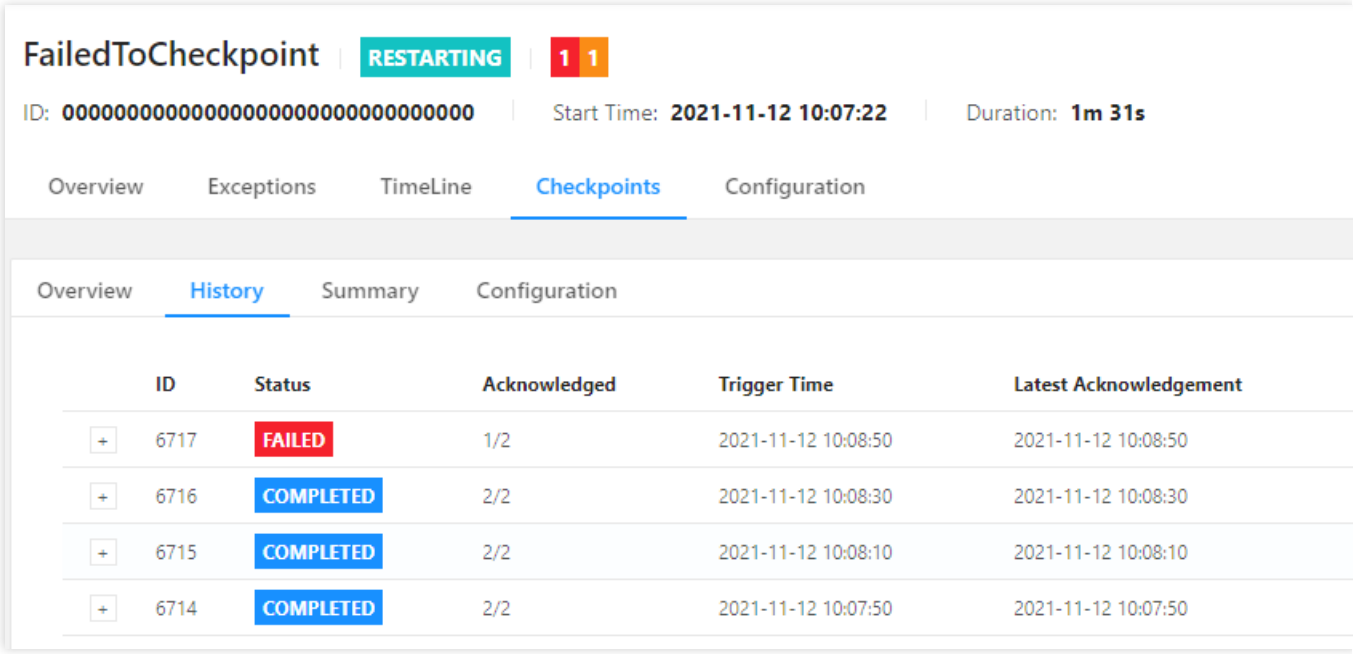
To avoid bothering you unnecessarily, at most one job exception event (other than the abnormal Pod exit) will be pushed per hour.
This feature is in beta testing. It supports detecting severe problems only, and thresholds cannot be adjusted. It will be further improved and upgraded. Please stay tuned.

Events
Checkpointing Failure

Last updated：2023-11-07 16:40:32

Overview

A checkpoint failure event in Stream Compute Service indicates that, for a job for which checkpointing is enabled, a checkpoint fails due to timeout or any other reason.
For a long-running job, an occasional checkpoint failure may not represent severe exceptions in the job, and you just need to handle the issue only when checkpoints fail frequently. For example, a checkpoint (ID: 6717) of a job fails, as shown on the **Checkpoints** page of the Flink UI.



Conditions

Trigger

A checkpoint of a job fails, with `FAILED` as its final status.

Clearing

A subsequent checkpoint of the job succeeds, with `COMPLETED` as its final status.

Alarms

You can configure an alarm policy for this event to receive trigger and clearing notifications in real time.

Suggestions

The causes of a checkpoint failure event are available on the events page. Depending on the Flink execution links used, the direct causes of checkpoint failure or some common errors may be displayed, so further analysis is required based on the specific issue.
You can also, based on the time of checkpoint failure, view the error logs of the JobManager and TaskManagers near this time point on the logs and the Flink UI pages of the job as instructed in Viewing the Logs of a Job and Viewing the Flink UI of a Job, respectively.
If you fail to identify errors as stated above due to too many TaskManagers or logs, you can search for exception logs under the instance ID of the checkpoint failure event as instructed in Diagnosis with Logs.
If the problem is still not found with the above diagnosis, please check as instructed in Viewing Monitoring Information whether resource overuse exists. In particular, you can focus on TaskManager CPU usage, heap memory usage, full GC count, full GC time, and other critical metrics to check whether exceptions exist.

Job Failure

Last updated：2023-11-07 16:48:29

Overview

A **job failure** event in Stream Compute Service indicates that the status of a Flink job changes from running to failed or restarting, which may cause interrupted data processing, output delay in the downstream, and other issues.

Conditions

Trigger

1. The status of a Flink job changes from `RUNNING` to `FAILED` or `RESTARTING`. Later, the Flink JobManager will recover the job in about 10s, with the running instance ID after recovery remaining unchanged.
2. A Flink job is restarted too many times or too frequently, exceeding the limit (the threshold is generally controlled by `restart-strategy.fixed-delay.attempts` and defaults to 5, and we recommend you increase it in a production environment) given in the Restart Policies. This will result in the exit of both the JobManager and the TaskManagers, and the system will try to recover the job from the last successful checkpoint within about 2 minutes, with the running instance ID after recovery increased by 1.

Clearing

After the Flink or Stream Compute Service system recovers the job back to `RUNNING`, a failed job recovery event will be generated, indicating the end of this event.

Alarms

You can configure an alarm policy for this event to receive trigger and clearing notifications in real time.

Suggestions

You can search for exception logs under the instance ID of the job for which the event is generated, as instructed in Diagnosis with Logs. Generally speaking, error messages before and after the keywords `from RUNNING to FAILED` contain the direct causes of the job failure. We recommend you analyze the issue based on these error messages together with the logs of the JobManager and the TaskManagers.

If the problem is still not found with the above diagnosis, please check as instructed in Viewing Monitoring Information whether resource overuse exists. You can focus on TaskManager CPU usage, heap memory usage, full GC count, full GC time, and other critical metrics to check whether exceptions exist.

Abnormal TaskManager Pod Exit

Last updated：2023-11-07 16:57:54

Overview

TaskManagers of a Flink Job run in a Kubernetes Pod. When the Pod stops, the Pod exit event is generated, and you can determine whether the Pod is in exception based on the return code, status, and other information.

Note

The same Pod may be re-built several times by Kubernetes due to exceptions, so you may receive an identical event several times.

Trigger conditions

The system monitors the exit of the TaskManager Pod in real time, and determines whether an exit is caused by SIGTERM based on the exit code (the normal exit code is 143). An exit code other than 143 indicates that the exit is not initiated by the JobManager, but is caused by TaskManager errors, and this case is determined as an abnormal Pod exit event.

Alarm configuration

You can configure an alarm policy as instructed in Configuring Event Alarms (Events) for this event to receive trigger and clearing notifications in real time.

Suggestions

Status Code	Possible Cause	Solution
137	The memory occupied by the job exceeded the memory quota of the Pod, and the Pod was killed due to OOM.	Increase the operator parallelism and the Task Manager spec (CUs) as instructed in Configuring Job Resources.
-1	This is the code of the basic policy, indicating that the Pod has exited, but no exit code is	Submit a ticket to contact the technicians for help.

	returned due to system errors or other reasons.	
0	During the startup process, the Pod cannot assign IPs in the associated subnet (no IPs available, for example), resulting in startup failure and exit.	Check whether available IPs are sufficient in the VPC subnet associated with the cluster. If yes, submit a ticket to contact the technicians for help.
1	An exception occurred during Flink initialization, resulting in startup failure.	This is generally caused by basic conflicts or overwriting of critical configuration files. You can search logs by `Could not start cluster entrypoint` and view relevant exceptions. If no cause can be identified, submit a ticket to contact the technicians for help.
2	A fatal error occurred during the startup of the Flink JobManager.	Search logs by `Fatal error occurred in the cluster entrypoint` and view relevant exceptions. If no cause can be identified, submit a ticket to contact the technicians for help.
239	An uncaptured fatal error occurred in Flink execution threads.	Search logs by `produced an uncaught exception. Stopping the process` and other keywords and view relevant exceptions. If no cause can be identified, submit a ticket to contact the technicians for help.

Abnormal JobManager Pod Exit

Last updated：2023-11-07 16:34:40

Overview

The JobManager of a Flink job manages and schedules the whole job, and its failure may cause severe consequences such as job crash and state loss. Therefore, the system will continuously detect and push abnormal JobManager exit events. In addition, to guarantee JobManager availability, HA configurations are enabled for each job, so that a new JobManager can be automatically selected and the job recovered when the existing JobManager exits unexpectedly.
In case of an abnormal JobManager Pod exit, the job generally can be automatically recovered, but the job completeness after recovery depends on whether checkpointing is enabled for the job and on the specific implementation logic of operators. Therefore, we recommend you check the job outputs (such as error data and duplicated data) after the job is recovered.
Note
The same Pod may be re-built several times by Kubernetes due to exceptions, so you may receive an identical event several times.

Trigger conditions

The system monitors the exit of the TaskManager Pod in real time, and determines whether an exit is caused by SIGTERM based on the exit code (the normal exit code is 143). An exit code other than 143 indicates that the exit is not initiated by the JobManager, but is caused by TaskManager errors, and this case is determined as an abnormal Pod exit event.

Alarm configuration

You can configure an alarm policy as instructed in [Configuring Event Alarms (Events)] for this event to receive trigger and clearing notifications in real time.

Suggestions

Status Code	Possible Cause	Solution

137	The memory occupied by the job exceeded the memory quota of the Pod, and the Pod was killed due to OOM.	This may be caused by inappropriate implementation of the source connector, with high memory pressure on the JobManager. If no cause can be identified, submit a ticket to contact the technicians for help.
-1	This is the code of the basic policy, indicating that the Pod has exited, but no exit code is returned due to system errors or other reasons.	Submit a ticket to contact the technicians for help.
0	During the startup process, the Pod cannot assign IPs in the associated subnet (no IPs available, for example), resulting in startup failure and exit.	Check whether available IPs are sufficient in the VPC subnet associated with the cluster. If yes, submit a ticket to contact the technicians for help.
1	An exception occurred during Flink initialization, resulting in startup failure.	This is generally caused by basic conflicts or overwriting of critical configuration files. You can search logs by `Could not start cluster entrypoint` and view relevant exceptions. If no cause can be identified, submit a ticket to contact the technicians for help.
2	A fatal error occurred during the startup of the Flink JobManager.	Search logs by `Fatal error occurred in the cluster entrypoint` and view relevant exceptions. If no cause can be identified, submit a ticket to contact the technicians for help.
239	An uncaptured fatal error occurred in Flink execution threads.	Search logs by `produced an uncaught exception. Stopping the process` and other keywords and view relevant exceptions. If no cause can be identified, submit a ticket to contact the technicians for help.

TaskManager Full GC Too Long

Last updated：2023-11-07 16:32:30

Overview

A TaskManager of a Flink job is a JVM process with its own heap memory. Both storing the runtime state of Flink operators and other operations can cause the use of too much heap memory.
When the JVM heap memory is about to be used up, full GC (a memory recovery mechanism) is triggered to release the space. If only a small size of memory is recovered each time and it is difficult to release the heap memory in time, full GC will be triggered frequently and continuously in the JVM. This operation will occupy a large amount of CPU time, making the execution threads of the job fail, and this event is triggered.
Note
This feature is in beta testing, so custom rules are not supported. This capability will be available in the future.

Trigger conditions

The system detects the full GC time of all TaskManagers of a Flink job every 5 minutes.
If the increased full GC time of a TaskManager accounts for more than 30% of a detection period (the full GC time exceeds 1.5 minutes within 5 minutes), a severe full GC problem exists in the job, and this event is triggered.
Note
To avoid frequent alarms, at most one push of this event can be triggered per hour for each running instance ID of each job.

Alarms

You can configure an alarm policy as instructed in Configuring Event Alarms (Events) for this event to receive trigger and recovery notifications in real time.

Suggestions

If you receive a push notification of this event, we recommend you configure more resources for the job as instructed in Configuring Job Resources. For example, you can increase the TaskManager spec (increased max available space of the TaskManager heap memory to contain more state data), or set a larger operator parallelism (reduced amount of data processed by a TaskManager to reduce the memory used) for more efficient data processing.

You can also adjust advanced Flink parameters as instructed in Advanced Job Parameters. For example, you can set `taskmanager.memory.managed.size` to a smaller value to increase the available heap memory. However, you must make adjustments under the guidance of an expert who fully understands the memory allocation mechanisms in Flink. Otherwise, this operation probably poses other issues.

If `OutOfMemoryError: Java heap space` or similar keywords are found in the job crash logs, you can enable the feature of Collecting Pod Crash Events, and set `-XX:+HeapDumpOnOutOfMemoryError` as described in the document, so that the local heap dump can be captured in time for analysis in case of an OOM crash of the job.

If `OutOfMemoryError: Java heap space` is not found in the logs, and the job is properly running, we recommend you configure alarms for the job, and add **job failure event** in the alarm rules of Stream Compute Service to timely receive job failure event pushes.

If the problem persists after all above methods are used, submit a ticket to contact the technicians for help.

Too-High TaskManager CPU Load

Last updated：2023-11-08 10:20:59

Overview

A TaskManager of a Flink job executes the various types of operator logic you define. If its CPU load is too high, reduced throughput, increased delay, and other issues may occur. This event is triggered when the majority of TaskManagers of a job are almost fully loaded for a long period.

Note

This feature is in beta testing, so custom rules are not supported. This capability will be available in the future.

Trigger conditions

The system detects the CPU utilization of all TaskManagers of a Flink job every 5 minutes.
If the CPU utilization of a TaskManager exceeds 90% in 5 consecutive data points, its CPU load is considered to be too high.
If more than 80% of the TaskManagers of the job are under an extremely high CPU load, this event is triggered and pushed.

Note

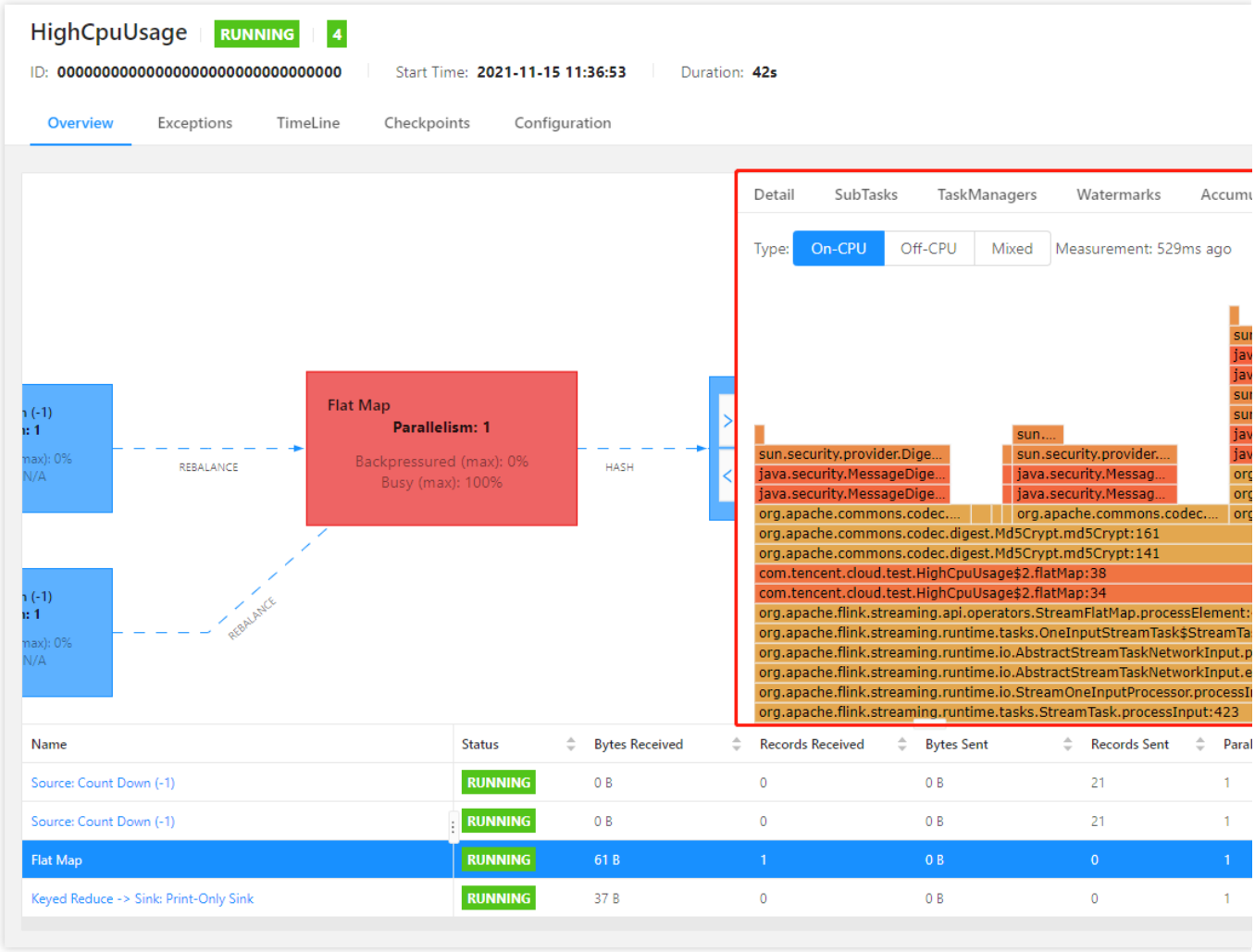
To avoid frequent alarms, at most one push of this event can be triggered per hour for each running instance ID of each job.

Alarms

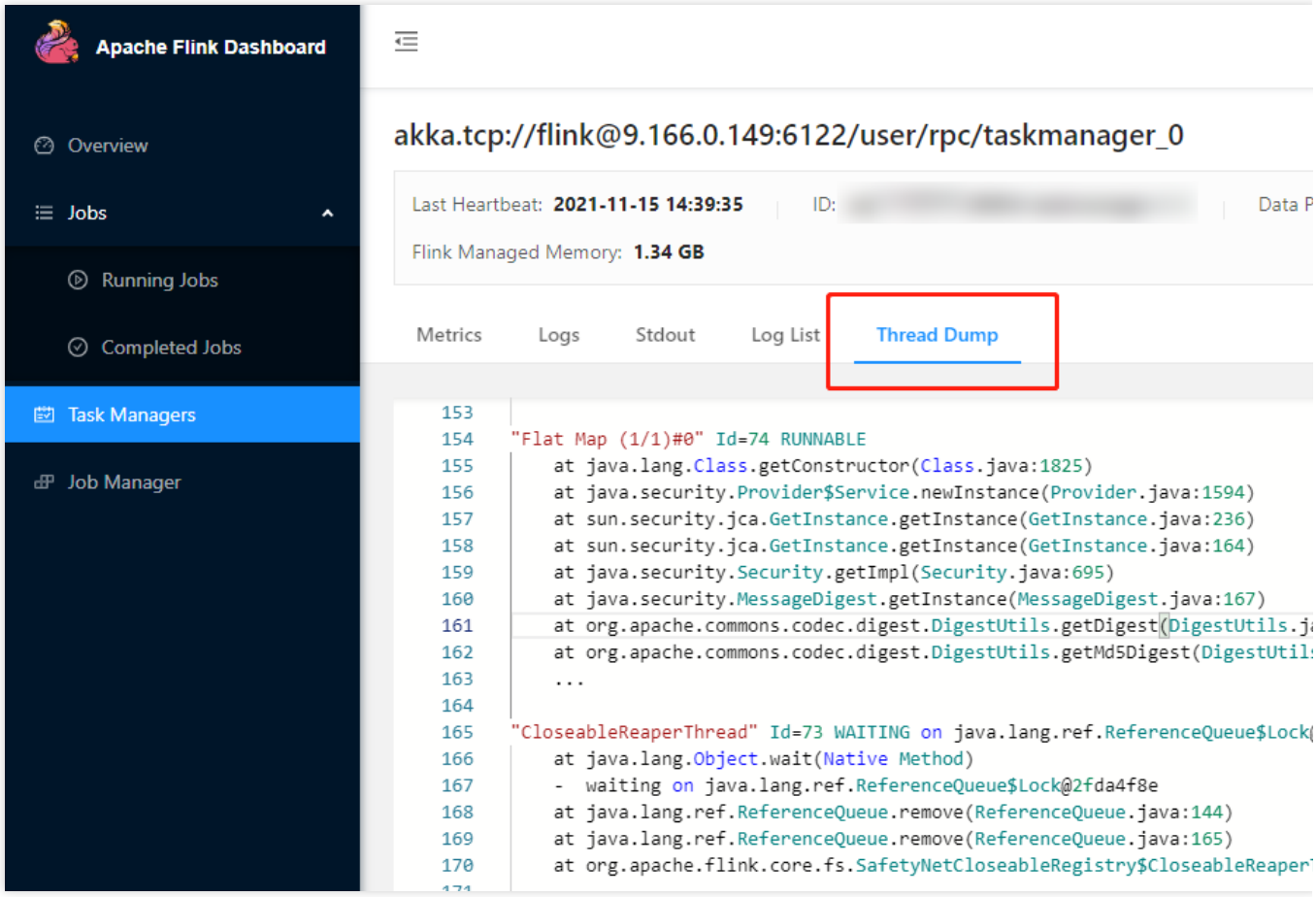
You can configure an alarm policy as instructed in Configuring Event Alarms (Events) for this event to receive trigger and clearing notifications in real time.

Suggestions

If Flink v1.13 or later is used, you can use the built-in Flame Graphs in Flink UI to analyze method call hotspots, i.e., those methods that occupy a lot of CPU time. Specifically, you need to add `rest.flamegraph.enabled: true` to the advanced parameters of the job as instructed in Advanced Job Parameters, and publish the new job version to use flame graphs.



If flame graphs are not used, or Flink v1.11 or earlier is used, you need to check the thread dump information of the TaskManagers in the Flink UI to identify methods that are frequently called in busy operators.



If the problem persists after all above methods are used, submit a ticket to contact the technicians for help.

High/Severe TaskManager Backpressure

Last updated：2023-11-07 16:23:00

Overview

Back pressure as described in Monitoring Back Pressure is an exception on a job: An operator is producing data faster than the downstream operators can consume due to slow processing by the downstream operators, congested transmission links, or other reasons, resulting in data piling. Then, piling will gradually occur in the upstream operators and finally in the data source, so less data is consumed than expected. If back pressure is not relieved for a long time, the total throughput of the job will decline greatly, even to 0.
If moderate back pressure occurs in an operator (the operator's back pressure displayed in the Flink Web UI is less than 50%, for example), you may observe the operator for more time to check whether the back pressure is occasional. A back pressure exceeding 50% (as shown below) probably affects the job performance greatly, and you need to handle it as soon as possible.

Note

This feature is in beta testing, so custom rules are not supported. This capability will be available in the future.

Trigger conditions

The system detects the operator back pressure in a Flink job every 5 minutes. If the back pressure of an operator (if there are multiple operator parallelism values, the max is used) is higher than 50%, the detection will continue until an operator is found to have a back pressure (`Backpressued` in the figure) lower than the threshold but a busyness (`Busy` in the figure) higher than 50%. This operator is generally the root cause of the back pressure, whose data processing rate is relatively slow. If you view the Flink Web UI of the job as instructed here at this moment, you will see a series of gray operators followed by a red operator.
If the back pressure of an operator in the link is higher than 50% but lower than 80%, an **OceanusBackpressureHigh** event is triggered. If the back pressure exceeds 80%, an **OceanusBackpressureTooHigh** event is triggered.

Note

To avoid frequent alarms, at most one push of this event can be triggered per hour for each running instance ID of each job.
Back pressure detection is available only to Flink v1.13 or later.

Alarm configuration

You can configure an alarm policy as instructed in Configuring Event Alarms (Events) for this event to receive trigger and clearing notifications in real time.

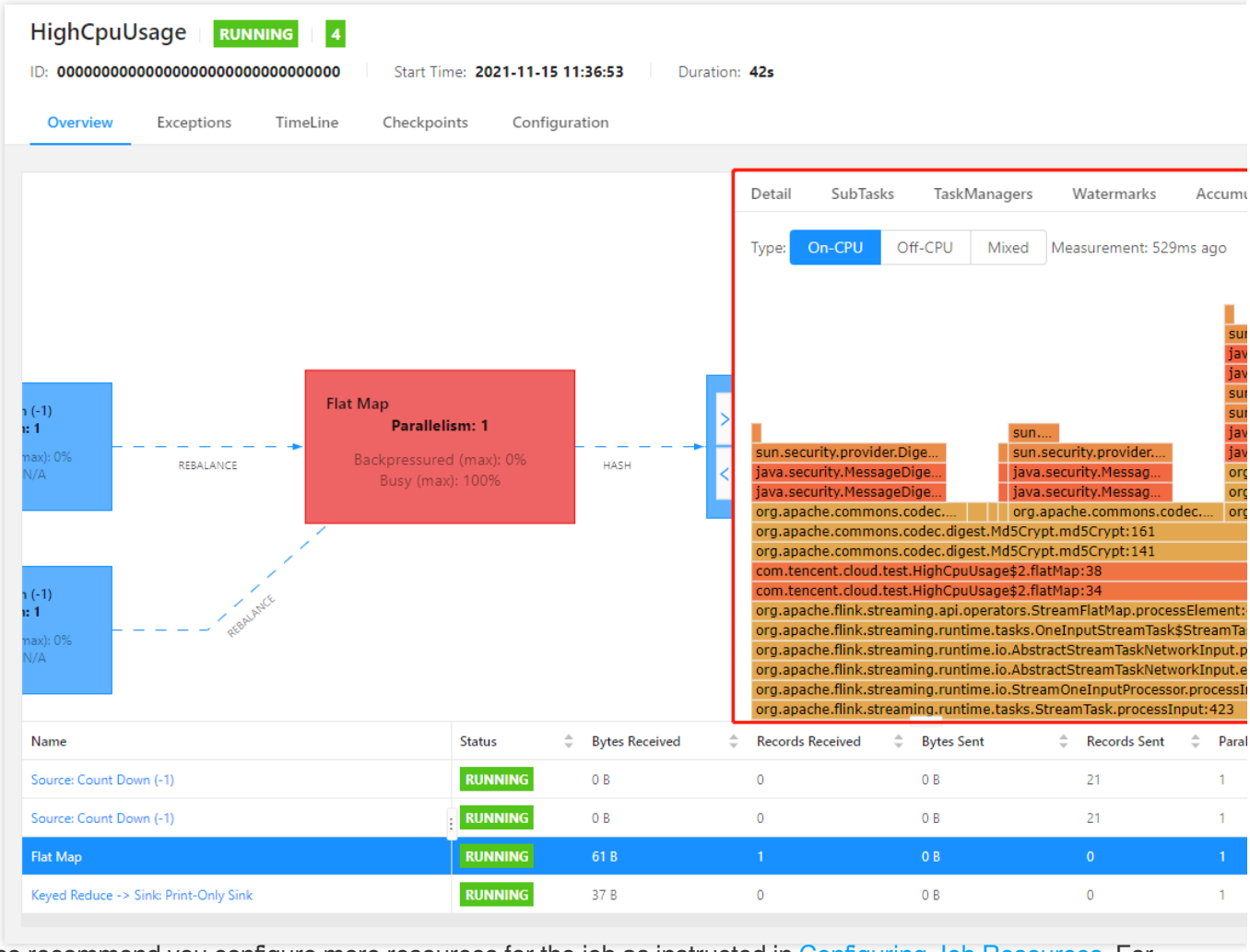
Note

`OceanusBackpressureHigh` and `OceanusBackpressureTooHigh` are two **different** alarm events. If you only care about a severe back pressure event that affects the job running, you can set an alarm for `OceanusBackpressureTooHigh` only.

Suggestions

If you receive the push notification of this event, we recommend you immediately view the Flink Web UI as instructed here and analyze the current execution graph. If the source operator causing the back pressure is spotted, we recommend you use the built-in Flame Graphs in Flink UI to analyze method call hotspots, i.e., those methods that occupy a lot of CPU time. Specifically, you need to add `rest.flamegraph.enabled: true` to the advanced parameters of the job as instructed in Advanced Job Parameters, and publish the new job version to use flame graphs.

For example, in the CPU flame graph of a busy operator shown below, the method for MD5 calculation has used too much CPU time and restricted the job performance. In this case, you can modify the calculation logic in this operator to avoid frequent calls of this method, use a more efficient algorithm, or take other optimization actions.



We also recommend you configure more resources for the job as instructed in Configuring Job Resources. For example, you can increase the TaskManager spec (an increased CPU quota of a TaskManager to manipulate more state) or set a larger operator parallelism (reduced amount of data processed by a TaskManager to reduce the computing pressure of its CPU) for more efficient data processing.

If the source of back pressure is not found, and the problem persists after all above methods are used, submit a ticket to contact the technicians for help.

JobManager CPU Load Too High

Last updated：2023-11-07 15:38:00

Overview

The JobManager of a Flink job manages and schedules the whole job. If its CPU load is too high, various exceptions may occur in the job. This event is triggered when the JobManager of the job is almost fully loaded for a long period of time.

Note

This feature is in beta testing, so custom rules are not supported. This capability will be available in the future.

Trigger conditions

The system detects the CPU utilization of the JobManager of a Flink job every 5 minutes.

If the CPU utilization of the JobManager exceeds 80% in 5 consecutive data points, its CPU load is considered to be too high.

Note

To avoid frequent alarms, at most one push of this event can be triggered per hour for each running instance ID of each job.

Alarm configuration

You can configure an alarm policy as instructed in Configuring Event Alarms (Events) for this event to receive trigger and clearing notifications in real time.

Suggestions

The reasons for a too-high JobManager CPU load are complicated. We recommend you configure more resources for the job as instructed in Configuring Job Resources, setting a larger JobManager spec, for example.

You can also submit a ticket to contact the technicians for help.

JobManager Full GC Too Long

Last updated：2023-11-07 15:43:23

Overview

The JobManager of a Flink job manages and schedules the whole job. It is a JVM process with its own heap memory. For a source connector using the FLIP-27 interface, its enumerator will record the shard information in the heap memory. Too many shards may result in the use of too much heap memory, affecting the stability of the job as a whole. When the JVM heap memory is about to be used up, full GC (a memory recovery mechanism) is triggered to release the space. If only a small size of memory is recovered each time and it is difficult to release the heap memory in time, full GC will be triggered frequently and continuously in the JVM. This operation will occupy a large amount of CPU time, making the execution threads of the job fail, and this event is triggered.

Note

This feature is in beta testing, so custom rules are not supported. This capability will be available in the future.

Trigger conditions

The system detects the full GC time of the JobManager of a Flink job every 5 minutes.

If the increased full GC time of the JobManager accounts for more than 30% of a detection period (the full GC time exceeds 1.5 minutes within 5 minutes), a severe full GC problem exists in the job, and this event is triggered.

Note

To avoid frequent alarms, at most one push of this event can be triggered per hour for each running instance ID of each job.

Alarm configuration

You can configure an alarm policy as instructed in Configuring Event Alarms (Events) for this event to receive trigger and clearing notifications in real time.

Suggestions

If you receive a push notification of this event, we recommend you configure more resources for the job as instructed in Configuring Job Resources. For example, you can increase the JobManager spec (increase max available space of the JobManager heap memory to contain more state data).

If you use MySQL CDC, we recommend you increase the size per shard in the WITH parameter (set `scan.incremental.snapshot.chunk.size` to a larger value) to avoid the JobManager heap memory from being used up due to too many shards.

If `OutOfMemoryError: Java heap space` is not found in the logs, and the job is properly running, we recommend you configure alarms for the job, and add **job failure event** in the alarm rules of Stream Compute Service to timely receive job failure event pushes.

If the problem persists after all above methods are used, submit a ticket to contact the technicians for help.

Managing Metadata Databases and Tables

Last updated：2023-11-07 16:29:28

Metadata refers to the databases and tables you reference in a streaming job. You can manage existing databases and tables in metadata, and quickly reference metadata in SQL job development.

Creating a database

Log in to the Stream Compute Service console, create a SQL job in **Jobs** as instructed in Creating a SQL Job, go to the **Development & Testing* tab of the job, and operate the metadata via** Database/Table references **on the left. In the initial state, only the default directory** `[_dc]` and the default database `[_db]` are there.
On the Database/Table references **page, select** Create > Create database **in the top right corner, select a catalog and enter a database name in the pop-up window, and click** Confirm**.

Creating a table

Step 1. On the **Database/Table references** page, select **Create > Create table** in the top right corner, select a catalog and database in the pop-up window, click **Next**, and select a method (three options: **Template**, **Custom**, and **Cloud resource**) to create the table. If you select **Custom** or **Cloud resource**, enter the required connection information.
For the **Custom** method:
Step 2. Click **Next** to go to the DDL editor page, and select a custom connector as the **Referenced package**. For how to upload a custom connector, see Managing Dependencies. After editing DDL statements, click **Syntax check** to check whether there are syntax errors, and click **Complete** to save the created metadata table.
Note
When creating a metadata table, you can set variables in the WITH parameters, and the variable naming rule is `${variable name}:default value}` , such as `${job_name}:job_test` .
Note
You cannot set custom variables for the `connector` and `version` attributes.

Referencing metadata table in a SQL job

To directly reference a metadata table in a SQL job, hover the pointer over a target metadata table, and click **Operation** and **Reference**. A three-segment reference format is used, such as `dc` . `db` . `test_table` . If parameters are used in the metadata table creation statements, click **Replace table variables** to replace parameters with actual values.

Table lineage

You can view the dependency of a metadata table using the table lineage. Specifically, hover the pointer over a target metadata table, click **Operation**, and select **View lineage**.

Editing DDL

You can edit the DDL of a metadata table as follows: Click **Operation**, and select **Edit DDL**. The WITH parameters can be edited, but the connector type cannot. After modifications are completed in the WITH parameter editor on the right, click **Update DDL** (updated DDL is displayed on the left), and click **Save** to complete the editing.

Variables

Last updated：2023-11-07 16:25:49

Variables can be used in **Variables** of a SQL job. **Replace table variables** allows replacing variables in both temporary tables and metadata tables of a SQL job with actual values.
Variable syntax: `${variable name}:default value`

Note

`_` is used as the separator in a variable name.

Creating a variable

Log in to the Stream Compute Service console, create a SQL job in **Jobs** as instructed in Creating a SQL Job, go to the **Development & Testing** tab of the job, and click **Variables** on the left.
On the **Variables** page, click **Create > Create variable** in the top right corner, enter the required information in the pop-up window, and click **Confirm**.

Referencing global variables in a table

On the **Database/Table references** page, click **Create > Create table** in the top right corner, select a catalog and database in the pop-up window, click **Next**, and select a method (three options: **Template**, **Custom**, and **Cloud resource**) to create the table. If you select **Custom** or **Cloud resource**, enter the required connection information. Set table variables in the WITH parameters and click **Complete**. For table variable naming rules, see Managing Tables.

Referencing global variables in a SQL job

You can directly reference global variables in a SQL job as follows: Click **Replace table variables**, select **Referenced global variable**, and click **Submit**.

Hive Catalogs

Last updated：2023-11-07 16:26:48

Overview

You can configure and use Hive catalogs, and view Hive metadata of a SQL job in the Stream Compute Service console. After metadata is stored in a Hive metastore, you have no need to explicitly declare the DDL statements in the job and can directly reference the metadata in the three-segment format.

Hive support

Flink Version	Description
1.11	Not supported
1.13	Hive v2.2.0, v2.3.2, v2.3.5, and v3.1.1 supported
1.14	Not supported

Prerequisites

You have activated the Hive metastore service on the Hive metastore.
The related commands are as follows:
`hive --service metastore` : Activate the Hive metastore service.
`ps -ef|grep metastore` : Check whether the service is successfully activated.

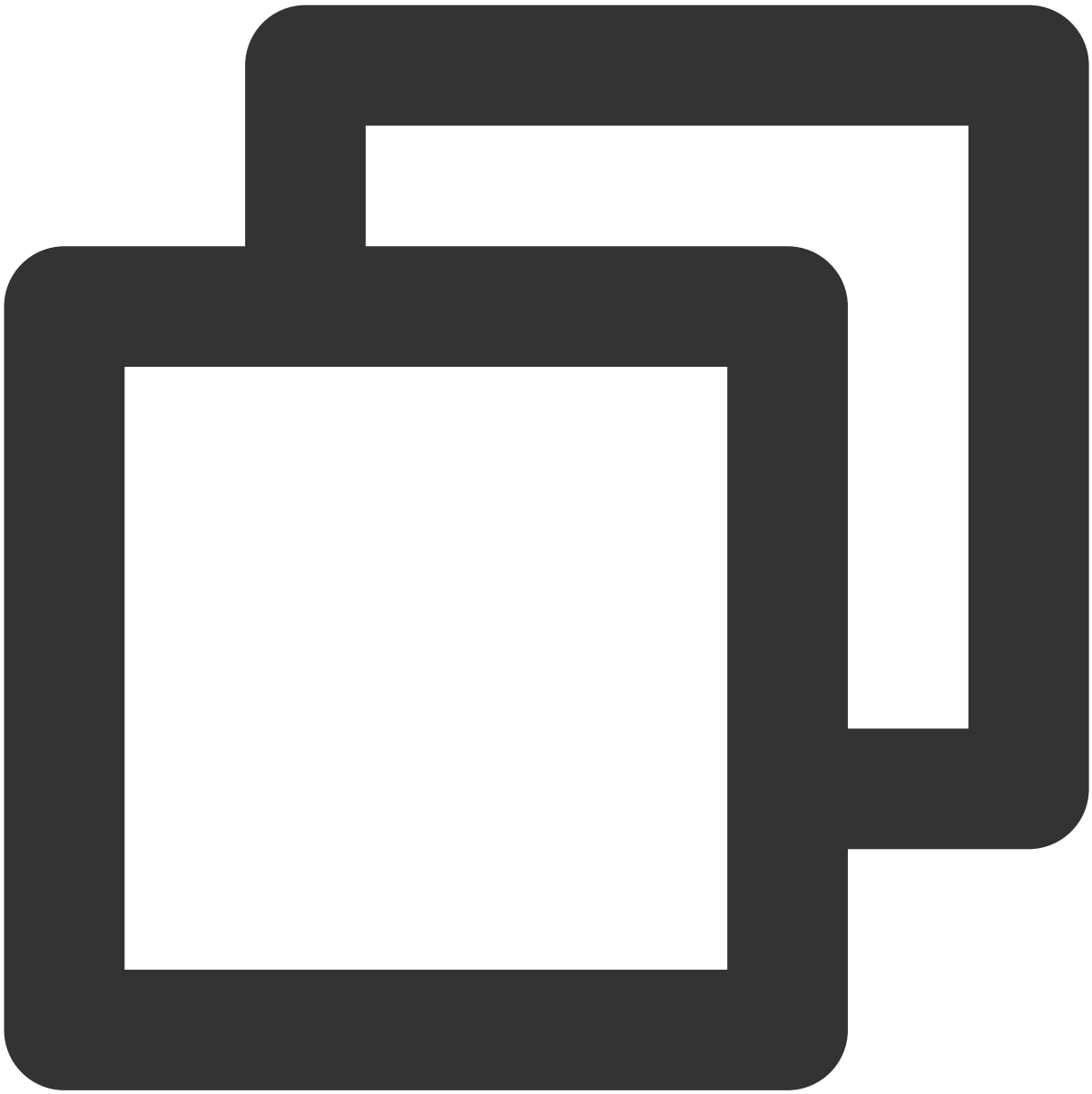
Directions

Creating a Hive catalog

Switch to the `_dc` directory, and click **Create Hive catalog**.
Upload the configuration files `hive-site.xml` (adding `urls` to it), `hdfs-site.xml` , `hivemetastore-site.xml` , and `hiveserver2-site.xml` (download them here).

Creating a database

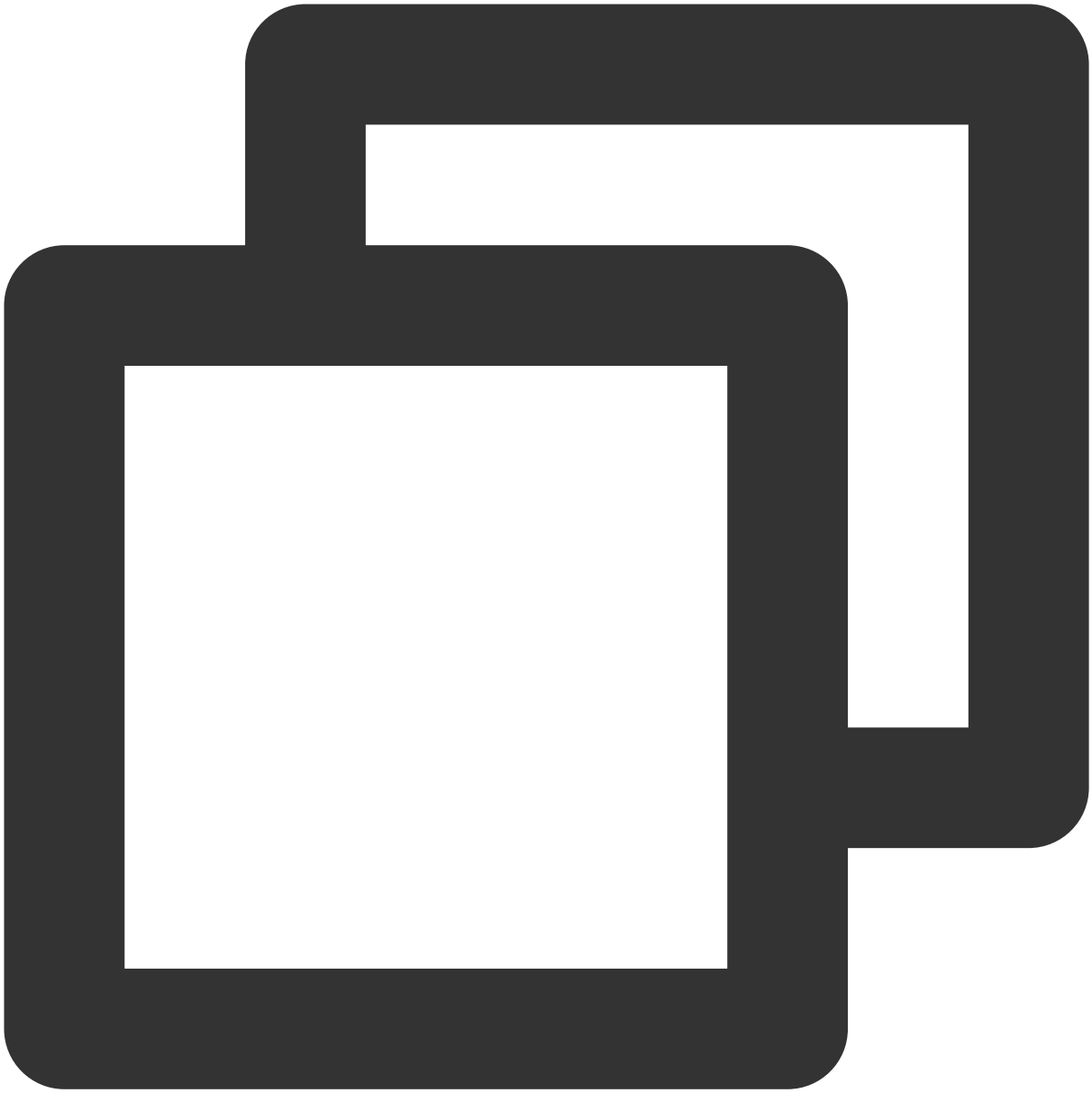
You can create databases in a SQL job. The database reference uses a two-segment format of `catalog_name.database_name`.



```
CREATE DATABASE IF NOT EXISTS `hiveCatalogName`.`databaseName`;
```

Creating a table

You can create tables in a SQL job. The table reference uses a three-segment format of `catalog_name.database_name.table_name`.

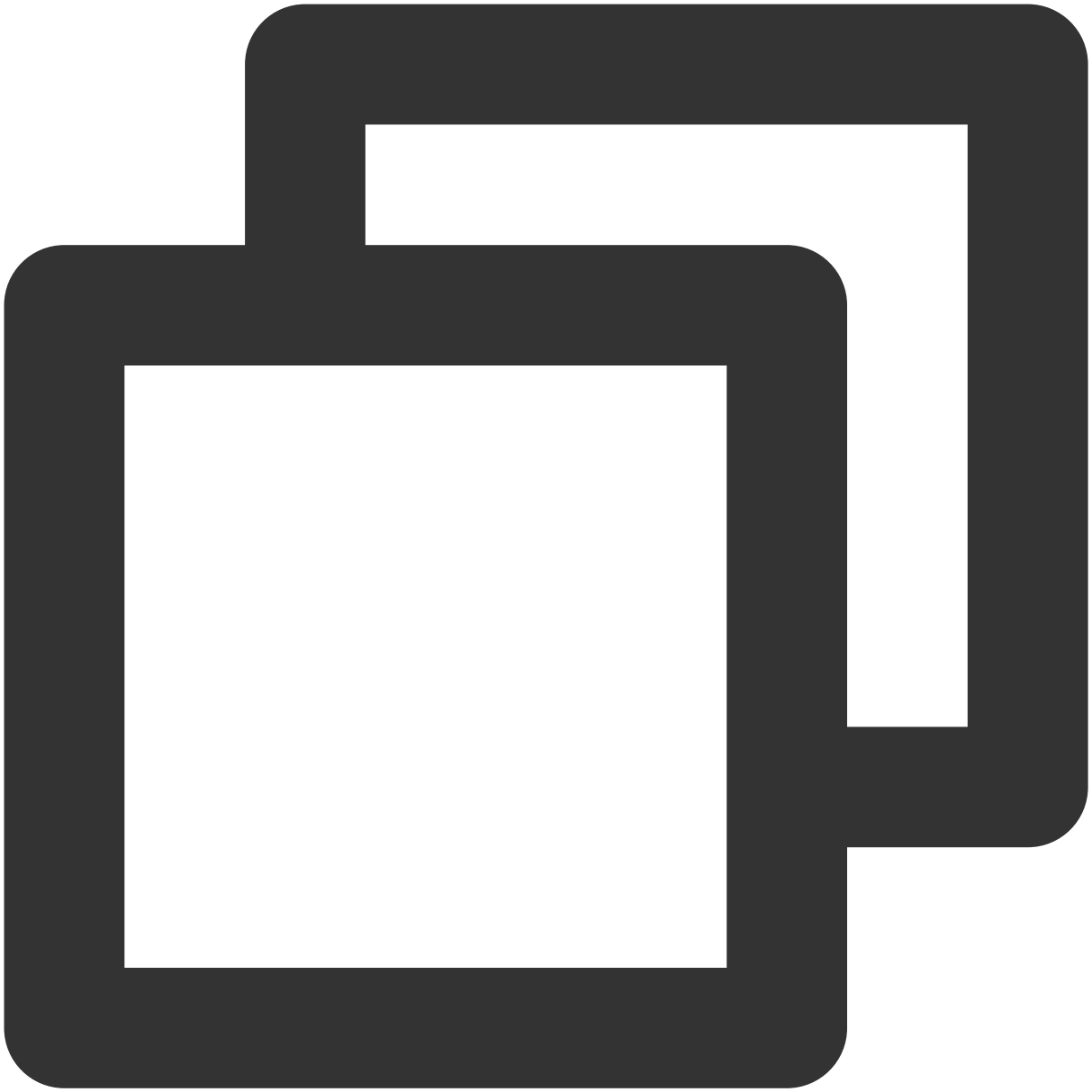


```
CREATE TABLE IF NOT EXISTS `hiveCatalogName`.`databaseName`.`tableName` (
  user_id INT,
  item_id INT,
  category_id INT,
  -- ts AS localtimestamp,
  -- WATERMARK FOR ts AS ts,
  behavior VARCHAR
) WITH (
  'connector' = 'datagen',
  'rows-per-second' = '1', -- The number of records generated per second
  'fields.user_id.kind' = 'sequence', -- Whether a bounded sequence (if yes, the ou
```

```
    'fields.user_id.start' = '1', -- The start value of the sequence
    'fields.user_id.end' = '10000', -- The end value of the sequence
    'fields.item_id.kind' = 'random', -- A random number without range
    'fields.item_id.min' = '1', -- The minimum random number
    'fields.item_id.max' = '1000', -- The maximum random number
    'fields.category_id.kind' = 'random', -- A random number without range
    'fields.category_id.min' = '1', -- The minimum random number
    'fields.category_id.max' = '1000', -- The maximum random number
    'fields.behavior.length' = '5' -- Random string length
  );
```

Referencing a Hive catalog table in a SQL job

Place the cursor to the position where a metadata table is to be inserted in a SQL job, find the target table on the left sidebar, and click **Operation > Reference**.



```
INSERT INTO
  `hiveCatalogName`.`databaseName`.`sink_tableName`
SELECT
  *
FROM
  `hiveCatalogName`.`databaseName`.`source_tableName`;
```

Note

You can reference only one Hive catalog in a job.

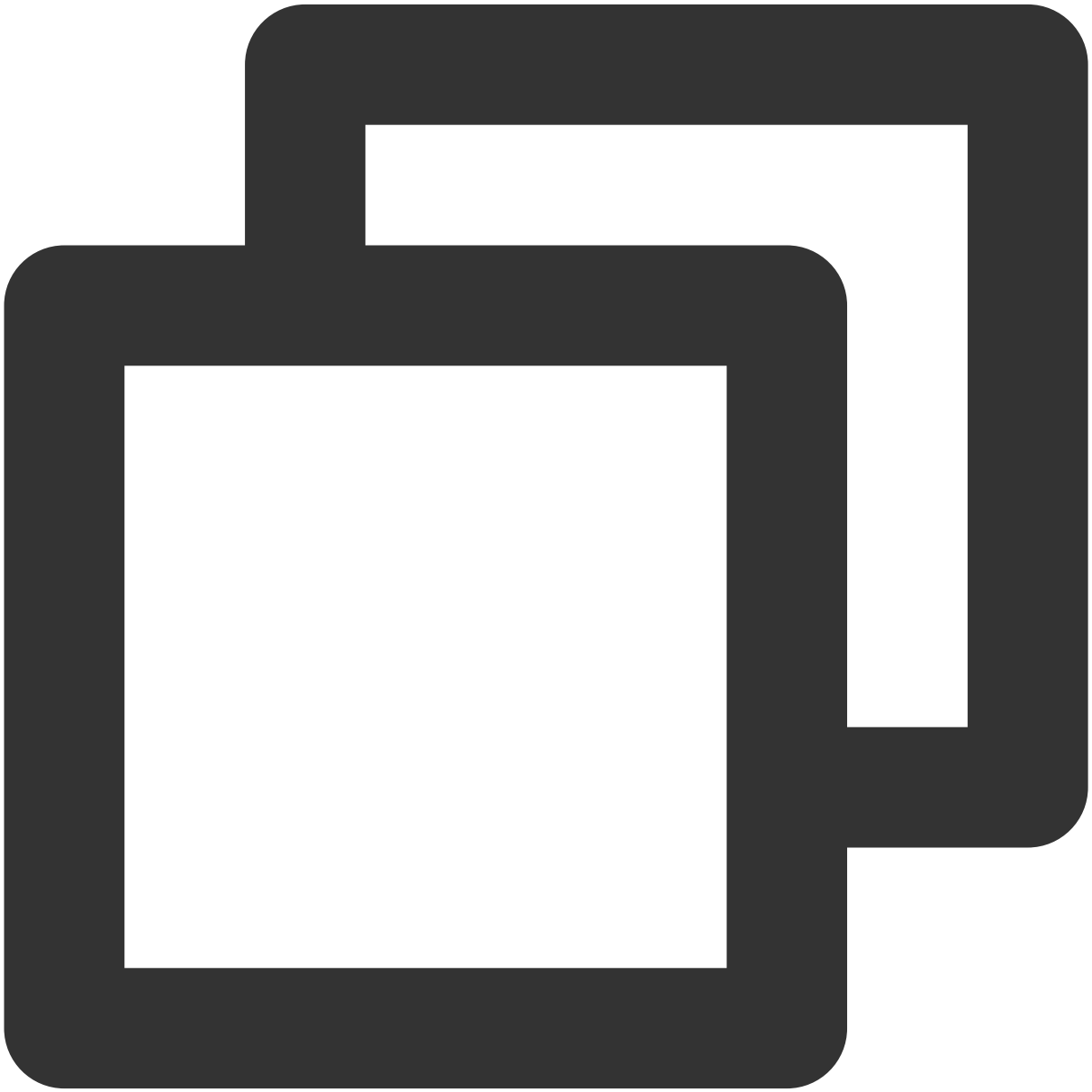
The DROP operation is unavailable on a Hive catalog.

Deleting a Hive metastore

On the left sidebar, click **Delete** of the Hive catalog to be deleted.

Granting permissions

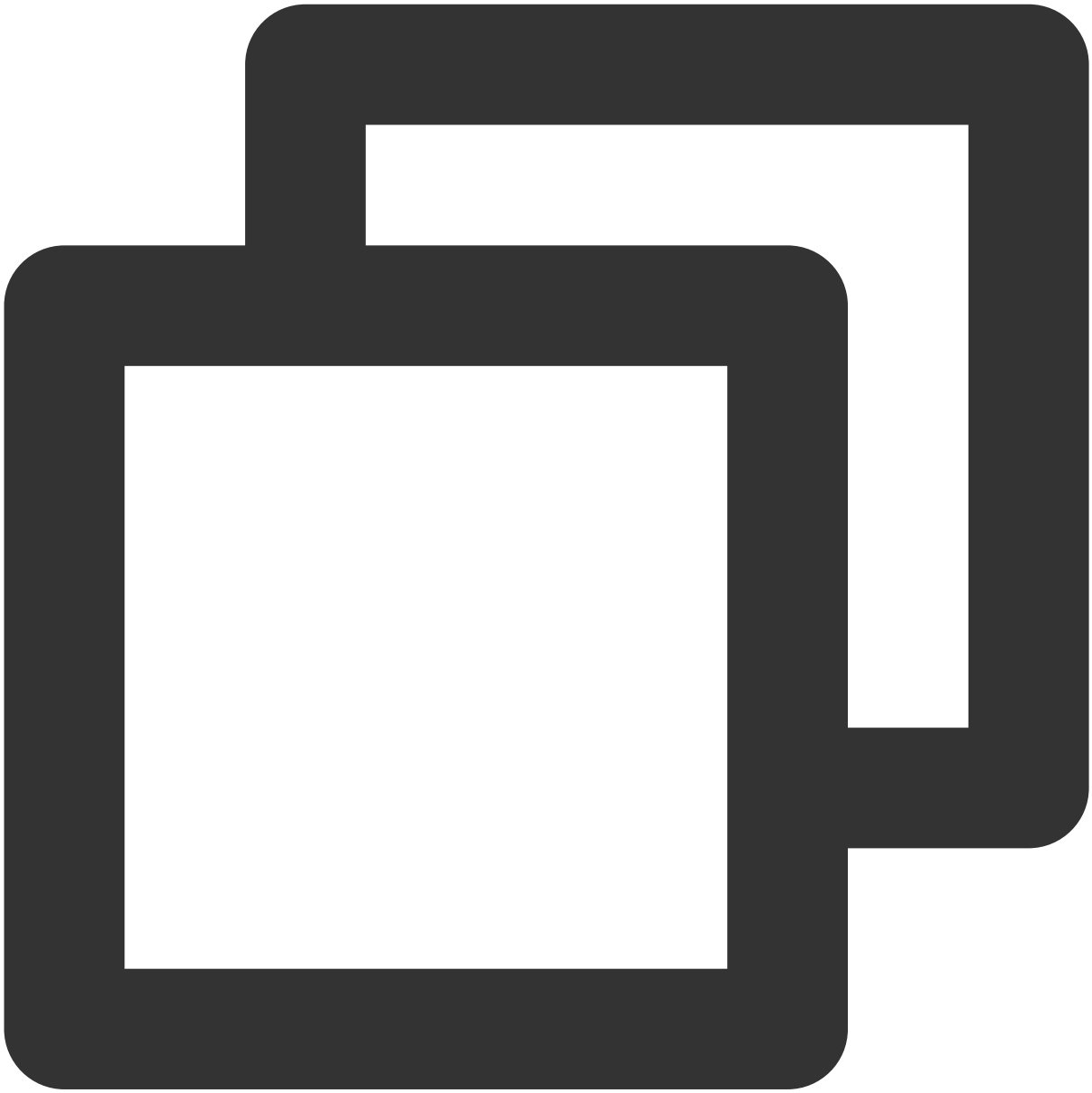
To use Hive catalogs in Stream Compute Service, access to HDFS files is required during the job execution, and the related Flink user must be granted the access permissions. The details are as follows:
Execute the following commands on all Hive master nodes.



```
useradd flink
groupadd supergroup
```

```
usermod -a -G supergroup flink
hdfs dfsadmin -refreshUserToGroupsMappings
```

We recommend you grant the permissions in Hive, and add the following configuration options to `hive-site.xml` :



```
<property>
<name>hive.metastore.authorization.storage.checks</name>
<value>true</value>
<description>Should the metastore do authorization checks against
the underlying storage for operations like drop-partition (disallow
```

```
the drop-partition if the user in question doesn't have permissions
to delete the corresponding directory on the storage).</description><property>
```

Managing Checkpoints

Last updated：2023-11-08 10:16:47

Viewing checkpoint information

Log in to the Stream Compute Service console, select **Jobs** on the left sidebar, and click the **Checkpoints** tab of a job to view its checkpoints. The checkpoint list of the job is displayed there.
The checkpoint list provides the following information:
Checkpoint ID/description: The ID uniquely identifies the current checkpoint, and the description is the checkpoint information specified by you or automatically generated by the system.
Trigger time: The time when the checkpointing is triggered.
Completion time: The time when the checkpointing is completed.
Time: The time taken to perform checkpointing.
Status: The checkpoint status. Valid values: Creating, Present, Cleared, Timeout, Failed, and so on.
Source: The checkpoint source. **Created during running** means the checkpoint is manually taken by a user, while **Created when the job is stopped** means the **Create a checkpoint when stopping the job** option is selected and the checkpoint is taken.
Job version: The job configuration version to which the checkpoint corresponds.
Location: The storage address of the checkpoint, currently a COS path.
Note
Cleared means the checkpoint has been manually or automatically cleared from its COS path and is unavailable for job start.

Manually creating a checkpoint

You can manually create a checkpoint of a running job, which contains all the current state data of the job and can be used for job upgrade and testing. Steps are as follows: On the **Checkpoints** page of a job, click **Trigger checkpoint**, enter a description in the pop-up window, and click **Confirm**.
Then, a checkpoint whose source is **Created during running** will appear in the checkpoint list. Please wait until its status changes from **Running** to **Completed**. A **Completed** checkpoint can be used to recover the job state during job start.
Note
If the **Checkpoints** tab shows that the current cluster does not support checkpoints, submit a ticket to upgrade the cluster.

Recovering a job from checkpoint

When running a job, you can select **Use a checkpoint** to recover the state of the job. Specifically, you select a desired checkpoint and click **Confirm**.

Setting a checkpoint storage policy

By default, the latest checkpoints of a job are saved in Flink. For how to recover a job from checkpoint, see Setting a checkpoint storage policy.
By default, the latest 5 checkpoints of a job are saved. You can adjust the number of checkpoints saved using `state.checkpoints.num-retained` in the advanced parameters.

Tuning Jobs
Automated Tuning

Last updated：2023-11-07 15:50:37

Overview

In general, job tuning may cost you a lot of time. For example, when publishing a new job, you need to consider the parallelism, the number of TaskManagers, the TaskManager spec (in CUs), and other configurations of the job. During the running of a job, you may also need to adjust the resources allocated to it to maximize the resource utilization, and allocate more resources to it in case of a back pressure or an increased delay.
To this end, Stream Compute Service provides the auto tuning feature, enabling you to adjust the parallelism and resources of a job more properly. It helps optimize your jobs from a global perspective, addressing various performance tuning issues such as insufficient throughput, busy jobs, and resource waste.

Use limits

Auto tuning cannot resolve the performance bottlenecks of a streaming job itself,
because a tuning policy handles the job based on certain assumptions, including smooth changes in flows, no data skew, and linear improvement of the throughput of each operator along with the increase of the parallelism. If the actual logic of the job is substantially deviated from these assumptions, the job may encounter exceptions. Manual tuning is required if the job itself has some problems. Common job exceptions are as follows:
Unable to change the job parallelism.
The job cannot run properly or is continuously restarted.
Issues related to user-defined features (UDF) of user-defined functions.
Severe data skew.
Auto tuning cannot resolve issues arising from external systems.
A failure or slow access of an external system may cause a larger job parallelism, resulting in increased pressure and a crash of the external system. You need to resolve external system issues by yourself. Common issues are as follows:
Insufficient partitions or throughput of the source message queue.
Performance issues of the sink in the downstream.
Deadlock in a database in the downstream.

Precautions

Auto tuning is in beta testing, so we recommend you not enable auto-scaling for critical production jobs.
The job will be restarted after auto tuning is triggered, so it will stop processing data for a short period of time. A job with a large state takes a longer period of time to start, and this may result in flow stop for a longer period of time, so we recommend you not enable auto-scaling for such a job.
The interval between two auto tuning triggers defaults to 10 minutes.
If you have enabled auto tuning for a JAR job, make sure no job parallelism is configured in the job code. Otherwise, job resources cannot be adjusted through auto-scaling, which means the auto tuning will be invalidated.
The auto tuning process is serial, so do not enable auto tuning for all jobs on a cluster with limited resources to avoid interference.

Default tuning rules

For a job for which auto tuning is enabled, the Stream Compute Service system automatically tunes its parallelism and TaskManager spec to optimize it.
1. Specifically, parallelism is tuned to achieve the throughput for the changing job flows, and the CPU utilization of TaskManagers and the data processing time of each operator are monitored to tune the parallelism of the job as a whole accordingly. Details are as follows:
When the CPU utilization of all TaskManagers is greater than 80% for 10 minutes, the default parallelism of the job is increased to twice the existing value, but the number of job CUs will not exceed the specified max available CUs (default: 64 CUs).
When the data processing time percentage of any Vertex node in the job is greater than 80% for 10 minutes, the default parallelism of the job is increased to twice the existing value, but the number of job CUs will not exceed the specified max available CUs (default: 64 CUs).
When the CPU utilization of all TaskManagers is less than for 4 hours, and the data processing time percentage of all Vertex nodes in the job is less than 20% for 4 hours, the default parallelism of the job is reduced to half the existing value (to 1 at most).
2. When auto tuning is enabled, the memory usage of TaskManagers is also monitored to adjust the memory settings of the job. Details are as follows:
When the heap memory utilization of all TaskManagers is greater than 80% for 1 hour, the TaskManager spec (in CUs) is increased to twice the existing value.
When the heap memory utilization of all TaskManagers is less than 30% for 4 hours, the TaskManager spec (in CUs) is reduced to half the existing value.
Note
The job parallelism can be reduced to 1 at most. The TaskManager spec may be different, depending on whether fine-grained resources are used. It can be 0.25, 0.5, 1, or 2 if fine-grained resources are used, and can only be 1 otherwise.

Managing Dependencies

Last updated：2023-11-07 15:52:47

Dependencies are external resources you use in jobs in the Stream Compute Service console. Two types of dependencies are available:
JAR package: Used as the main package of a JAR job, or as a custom function or connector of a SQL or JAR job. You need to upload JAR packages in **Dependencies**, and reference them in JAR and SQL jobs.
Configuration file: The resource files to be accessed by a JAR job, such as text or configuration files. After configuration files are uploaded in **Dependencies**, you can use them (accessing configurations, for example) in JAR jobs via a fixed path.

Creating a dependency

Log in to the Stream Compute Service console, select **Dependencies** on the left sidebar, and click **Create dependency**. On the dependency creation page, set a region, dependency type, directory, upload method, and dependency description, and click **Confirm**. The dependency created will appear in the dependency list.
Region: **The region selected must be the same as that where the job for which the dependency is created resides, or that of the private cluster associated**.
Dependency type: JAR package or configuration file.
Upload method: The local and COS methods are available.
For the local method, click **Select**, and select and upload a local dependency file (up to 50 MB for a JAR package or 2 MB for a configuration file). To upload a larger package or file, submit a ticket.
For the COS method, click **Select**, and select a dependency from the COS bucket list. To use dependencies this way, you must upload them to the COS console first.
The dependency name must be identical to the uploaded file and cannot be changed. We recommend you specify a distinguishable and readable name for a file to upload.

Uploading a new dependency version

Upload a new dependency version as follows: Click **Upload new version** in the **Operation** column of the dependency for which a new version is to add, enter a version description in the pop-up window, and click **Confirm** to generate a new dependency version (with a version number automatically generated).
Up to 20 versions can be retained for a dependency. If 20 versions already exist, and you still need to upload a new version, delete unnecessary versions first.

Viewing dependency information

Log in to the Stream Compute Service console, and select **Dependencies** to view all dependencies in the current region. Click a target dependency to view the version of its package and associated jobs (those that have referenced this dependency). A job in Stream Compute Service references to a specific version of a dependency, which means a job version references to a dependency version.

Deleting a dependency

You can delete a dependency as a whole via the dependency list. This operation will delete all versions of the dependency. You can also delete a specific version of the dependency via the **Dependency version** section.
A dependency version referenced by a job (regardless of the job status) cannot be deleted directly. Instead, you need to first delete the appropriate job version or remove the reference to the dependency version from the job version, and then delete the dependency version. When a dependency version cannot be deleted, the dependency as a whole cannot be deleted directly, either.

Viewing a file

If the dependency type is configuration file, **View** will be available in the **Operation** column of the dependency version list, allowing you to view the content of the configuration file.
For some existing dependencies, their names may be different from those of the corresponding files in COS. In this case, **View** will display the file name in COS, which can be used for file call in a program.

Managing Clusters

Viewing the Information of a Cluster

Last updated：2023-11-07 15:51:30

A job in Stream Compute Service needs to run on a private cluster, which can be created as follows: Log in to the Stream Compute Service console, and select **Compute resources** > **Create**. For details, see Creating a Private Cluster and Overview. After a cluster is created, you can click its name in **Compute resources** to view its information and jobs running on it.

Some cluster information fields are described as follows:

Field	Description
Cluster name	A user-defined name.
Cluster ID	The unique identification of the cluster, automatically generated by the system.
Cluster status	The current status of the cluster.
Cluster description	A user-defined description that helps identify the cluster.
Computer resources (CU)	The idle/total CUs in the cluster.
Region/AZ	The region/AZ where the cluster resides.
VPC	The VPC and subnet associated with the cluster. Stream Compute Service connects a private cluster with your VPC through ENI to access resources and services in this network environment.
COS	The COS bucket associated with the cluster during its creation.
Logging	The CLS logsets and log topics associated with the cluster during its creation.
Tag	The tags attached to the cluster.
Billing mode	Monthly subscription is supported.
Flink version	The Flink version used on the cluster.
Creation	The cluster creation time.

time	
DNS	The DNS information of the cluster.
Flink UI access policy	You can set a Flink UI access policy. If no access IP allowlist is set, Flink UI is accessible to all public IPs.

Scaling Out a Cluster

Last updated：2023-11-07 15:46:05

Scale out a cluster following these steps: Log in to the Stream Compute Service console, identify the target cluster from the cluster list in **Compute resources**, and select **Adjust configuration** on the cluster details page. Adjust **Target CUs** to your desired value, confirm the fees, select **I have read and agree to Configuration Adjustment Costs of Pay-As-You-Go Clusters**, and click **Confirm**. After you confirm the order and pay successfully, the cluster scale-out starts. During the scale-out, existing jobs on the cluster are not affected, and you can also create new jobs on it. After scale-out is complete, added CUs will be available for use. For scale-out fees, see Configuration Adjustments.

Note

Considering the performance of underlying machines, you can add up to 300 CUs at a time, and a single cluster can have a maximum of 800 CUs by default. If you need to raise the limit, contact us.

Terminating a Cluster

Last updated：2023-11-07 15:48:27

You may terminate a cluster when it is no longer in service. However, jobs are resources attached to a cluster, and deleting a cluster will stop and permanently delete all jobs on it. Please proceed with caution.
Terminate a cluster following these steps: Log in to the Stream Compute Service console, identify the target cluster in the cluster list in **Compute resources**, select **More** on its details page, click **Terminate** in the drop-down list, confirm the information of the cluster to terminate and jobs on it in the pop-up window, select **I have read and agree to Terminating Clusters and Refund**, and click **Confirm**. You may receive refunds from terminating a cluster. For details, see Refund.
A terminated cluster will be **Isolated** within 7 days from the date of termination, during which the cluster is out of service (with its data retained). You can resume the cluster via renewal, and the new billing cycle after renewal will start from the time when the cluster is terminated. If the cluster is not renewed within this period, it will be released upon expiry of the period, and its data will be permanently cleared.

Scaling In a Cluster

Last updated：2023-11-07 15:47:27

Scale in a cluster following these steps: Log in to the Stream Compute Service console, identify the target cluster from the cluster list in **Compute resources**, and select **Adjust configuration** on the cluster details page. Adjust **Target CUs** to your desired value, confirm the fees, select **I have read and agree to Configuration Adjustment Costs of Pay-As-You-Go Clusters**, and click **Confirm**. After you confirm the order, the cluster scale-in starts. **During scale-in, existing jobs on the cluster may be affected and thus restarted**. For scale-in refunds, see Configuration Adjustments.

Note

The total number of CUs cannot be smaller than the number of CUs in use or than 12.

When setting **Target CUs**, you need to set it to the total number of CUs after scale-in, but not the number of CUs to remove.

Migrating a Cluster

Last updated：2023-11-07 15:48:54

When your cluster is at a lower edition, some new features (such as metric monitoring and checkpointing) may be unavailable to it. The optimal solution is to migrate your jobs to a cluster at a higher edition.
This document describes how to migrate a cluster in the Stream Compute Service console. If you have any trouble during the migration, submit a ticket to contact us.

Creating a new cluster

Log in to the Stream Compute Service console, and click **Create** in **Compute resources**. For details, see Creating a Private Cluster.

Note

When you create a new private cluster, the region selected must be the same as that of the VPC and the existing cluster to be migrated.
Custom configurations of the existing cluster (such as DNS configurations) need to be synced to the new cluster.
The new cluster shall be associated with the workspace of the existing cluster.

Initializing jobs on the existing cluster to the new cluster

The jobs on the existing cluster can be copied to the new cluster via the "batch create copy" feature detailed in Job Operations.
The steps are as follows: Go to the workspace associated with the existing cluster, click **Batch** on the top of the job list, and select **Create copy**.

Note

If the destination cluster does not support the Flink version used by the source jobs, the job copies will use the default Flink version of the destination cluster.
If the source jobs use fine-grained resources that are not supported by the destination cluster, both the JobManager spec and TaskManager spec of the job copies will be set to 1 CU.
Jobs can be copied only within the workspace of the existing cluster, so the destination cluster selected must be one associated with the workspace.

Starting/Stopping the jobs on the new/existing cluster

You need to stop jobs on the existing cluster, and start those on the new cluster. **After the jobs on the new cluster are started, check the job status for exceptions.**

Stop the jobs on the existing cluster

Start the jobs on the new cluster

Note

New jobs will not retain the state data of the original jobs. Please separately consider the data completion logic.

Terminating the existing cluster

After all jobs are successfully migrated, terminate the existing cluster as instructed in Terminating a Cluster.

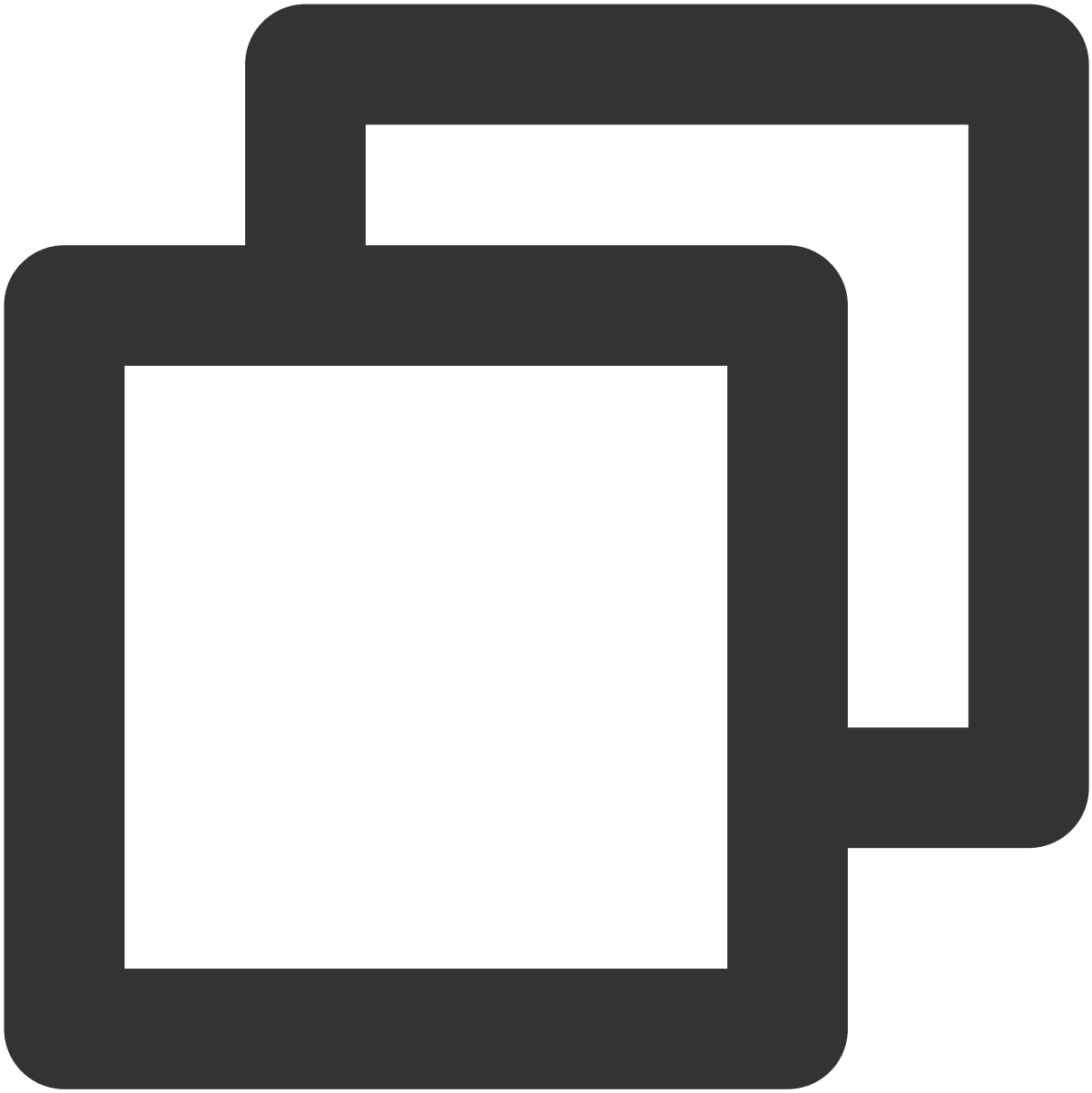
Customizing DNS Service

Last updated：2023-11-07 15:44:49

What is custom DNS service

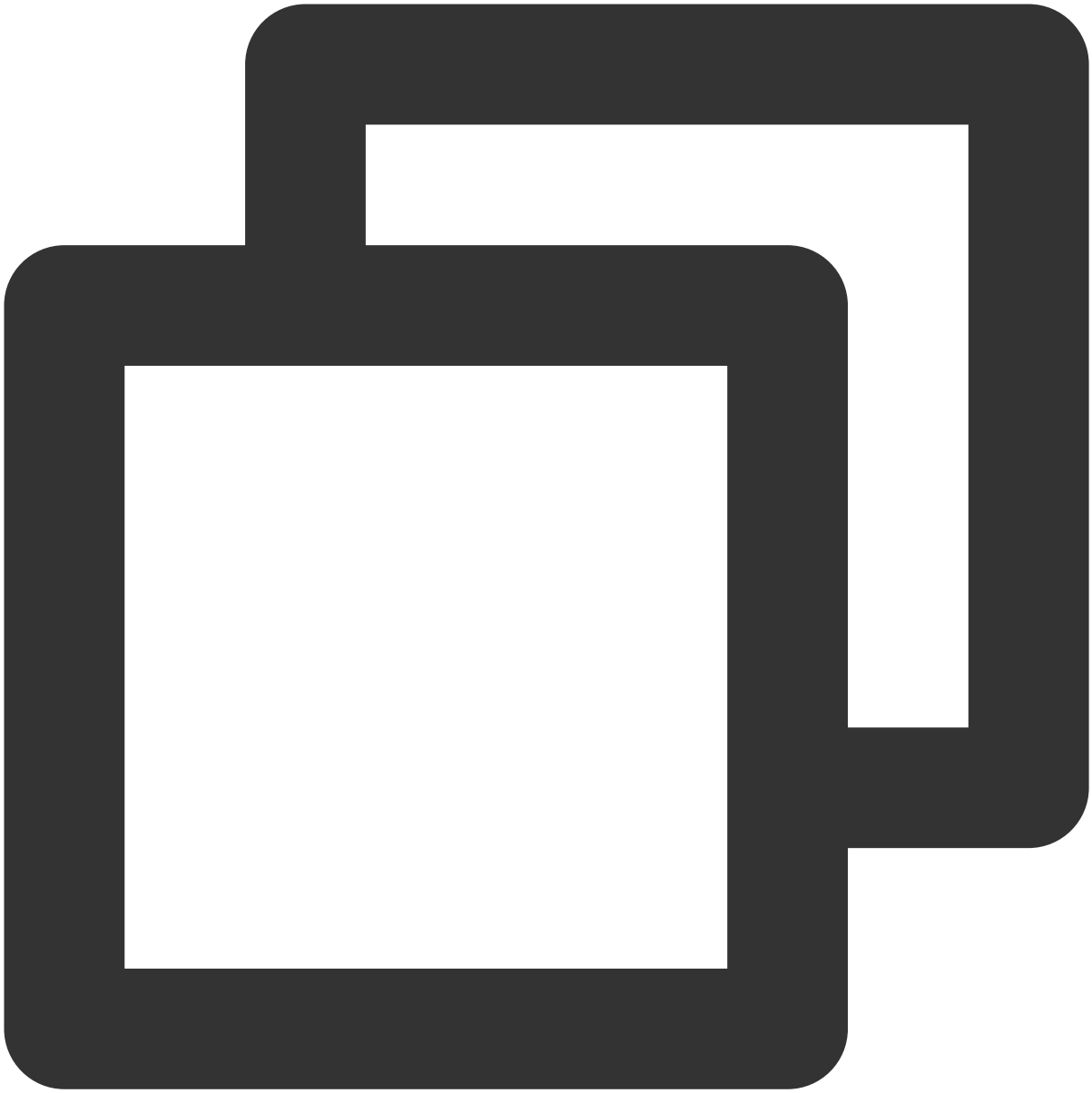
Your jobs may access external resources (such as CKafka and TencentDB for MySQL) via domains to facilitate Ops. You can customize DNS service to resolve domains. For the implementation of custom DNS service, see the Kubernetes document Customizing DNS Service. The DNS resolution process is generally completed with one of the following methods.

1. DNS host mapping. As shown below, you can access a CKafka instance (IP: `172.17.0.2`) with `kafka.example.com`.



```
172.17.0.2 kafka.example.com
172.17.0.3 mysql.example.com
```

2. DNS resolution. As shown below, assume that your DNS server IPs are `172.17.0.253` and `172.17.0.254`. Your requests in a job to access any domain in the form of `*.example.com` will be resolved using your DNS server. If you set the mapping of `172.17.0.2 kafka.example.com` in your DNS server, requests to access `kafka.example.com` will be resolved to `172.17.0.2`.



```
example.com {
 forward . 172.17.0.253 172.17.0.254
}
```

How to customize DNS service

Customize DNS service for a cluster on its details page. Note: If both host mapping and DNS resolution are configured, DNS resolution will be used.

Directions

1. Select Compute resources, click **Cluster info** of a target cluster,
2. set the host or domain in the pop-up window, and click **Confirm** to save the settings (can be modified if necessary).

Testing Network Connectivity

Last updated：2023-11-07 15:45:14

Your jobs may access external resources (such as CKafka and TencentDB for MySQL) via domains or IPs + ports to facilitate Ops. Network connectivity test allows you to check whether the network of your cluster is successfully connected to external resources.

Directions

1. Select **Compute resources** > **Cluster info** > **More**, and click **Connectivity test** in the drop-down list.
2. Test IP.
3. Test IP and port.

Managing Permissions

Overview

Last updated：2023-11-08 10:18:09

This document describes the permissions that can be granted to sub-users of Stream Compute Service. Sub-users can access the service only after authorization by the root account. For authorization steps, see Configuring Basic Permissions and Space Role Permissions.

Permission management in Stream Compute Service involves three aspects:

1. CAM policy
2. Access to other services
3. Space role permissions

CAM policy involves the basic access to Tencent Cloud resources. Space role permissions involve the fine-grained management of permissions on jobs and resources in Stream Compute Service. Access to other services involves the access to your other Tencent Cloud resources by Stream Compute Service.

CAM policy

After Signup, the generated Tencent Cloud account is the root account, which has the permission to manage all Tencent Cloud resources under it. If you need other users to help you manage the Tencent Cloud resources under your account, you can create, manage, and terminate users (groups) in CAM and use identity and policy management features to control their access to your resources.

Access to other services

The underlying system services of Stream Compute Service must be authorized to access various cloud service resources such as CKafka, COS, and CLS via your VPC. This is the most basic authorization required for the proper running of the Stream Compute Service system. When this authorization is required during the use of Stream Compute Service, the authorization page will automatically appear.

Space role permissions

Within the framework of the unified Tencent Cloud CAM, Stream Compute Service has its own predefined system for space role permissions to help coordinate between different business departments of your organization. These permissions help you isolate compute resources of different businesses and control at a finer granularity the

permissions of different members to view and operate jobs and files. A space isolates the jobs, metadata, dependencies, and other resources in it from the outside world. In the space, each sub-account is assigned a predefined role with the required permissions.

Configuring Basic Permissions

Last updated：2023-11-08 10:18:35

This document describes how a root account grants a Stream Compute Service sub-user the required permissions. If you are a sub-user, contact your root account to grant you the permissions. The specific authorization steps are as follows.

CAM policy

Stream Compute Service uses the unified Tencent Cloud CAM service to help organizations manage users' access to their resources. For details, see Cloud Access Management.

Granting a sub-user access to Stream Compute Service

By default, a root account has access to all Stream Compute Service resources, but a sub-account has no access to these resources. If you try to access Stream Compute Service with a sub-account, a CAM authentication error will occur.
In this case, the root account needs to associate the sub-account with the predefined policy `QcloudOceanusFullAccess` in the CAM console as instructed in Authorization Management. After the sub-account is associated with the policy `QcloudOceanusFullAccess`, it will have access to Stream Compute Service. For details, see CAM.

Access to other services

The underlying system services of Stream Compute Service must be authorized to access various cloud service resources such as CKafka, COS, and CLS via your VPC. This is the most basic authorization required for the proper running of the Stream Compute Service system.
When this authorization is required during the use of Stream Compute Service, the authorization page will automatically appear. However, only a root account, a sub-user with `QcloudCamRoleFullAccess`, and a sub-user with `QcloudCamSubaccountsAuthorizeRoleFullAccess` can perform this operation for themselves.
In the other case, a sub-account is granted an additional PassRole.

Granting a sub-account a PassRole

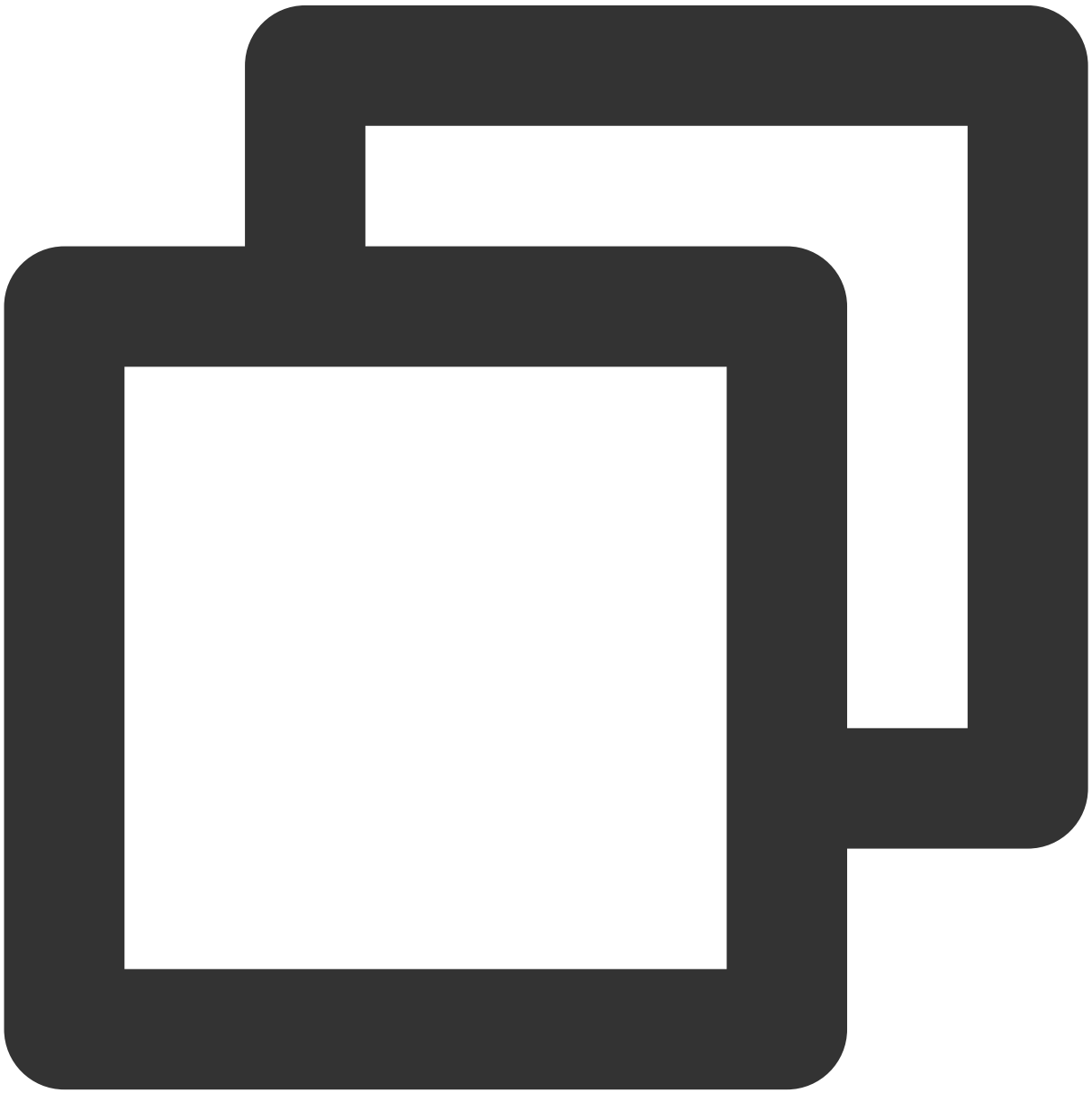
When a user logs in with a sub-account, although the above authorizations have been completed and the `Oceanus_QCSRole` role created successfully, the underlying system services of Stream Compute Service still

cannot play the `Oceanus_QCSRole` role.

In this case, the root account or a sub-account with the admin permission needs to grant the sub-account the PassRole permissions, so that PassRole can pass the Stream Compute Service role to the underlying system services. After the settings, the underlying system services can access various cloud service resources such as CKafka, COS, and CLS via your VPC.

Steps: The root account or a sub-account with the admin permission creates a policy and grants the sub-account the `cam:PassRole` permission.

Policy content



```
{
    "version": "2.0",
    "statement": [
        {
            "effect": "allow",
            "action": "cam:PassRole",
            "resource": "qcs::cam::uin/${OwnerUin}:roleName/Oceanus_QCSRole"
        }
    ]
}
```

Note

`OwnerUin` in the policy refers to the account ID of the root account.

For how to create a policy, see Creating Custom Policy.

For authorization, see Authorization Management.

Directions

1. On the Create by Policy Syntax page, select **Blank Template**,
2. enter the above policy content (replace the UIN of the root account with your UIN) on the **Edit Policy** page,
3. go back to the User List page, click **Authorize** of
4. the target user, select the policy created, and click **OK**.

Note

Till now, the sub-account can properly access Stream Compute Service and various cloud resources such as CKafka, COS, and CLS via VPC in the Stream Compute Service console. To control access to jobs and resources at a finer granularity, see Space Role Permissions.

Space Role Permissions

Last updated：2023-11-08 10:16:06

Within the framework of the unified Tencent Cloud CAM, Stream Compute Service has its own predefined system for space role permissions to help coordinate between different business departments of your organization. These permissions help you isolate compute resources of different businesses and control at a finer granularity the permissions of different members to view and operate jobs and files.

Predefined role permissions

Stream Compute Service provides four predefined space roles:
1. Super admin: Specified by the root account, a super admin has the highest level of access other than operating the root account and can be shared between different regions.
2. Space admin: Specified by the root account or a super admin account, a space admin has the permission to add or remove the members in a space.
3. Developer: Added to a space by a space admin/super admin/root account in the Members module, a developer can operate jobs in the space.
4. Guest: Added to a space by a space admin/super admin/root account in the Members module, a guest can only view resources in the space.
The detailed permissions of all predefined roles are as follows:

Permission	Super Admin	Space Admin	Developer	Guest
Create/Terminate cluster	✔	×	×	×
Modify cluster info	✔	×	×	×
Renew/Upgrade cluster	✔	×	×	×
View cluster	✔	✔	✔	✔
Add/Delete space	✔	×	×	×
Modify space attribute	✔	×	×	×
Associate/Disassociate cluster with/from space	✔	×	×	×
Add/Delete space member	✔	✔	×	×

Modify space member role	✔	✔	✖	✖
Edit super admin	✔	✖	✖	✖
Create/Delete job	✔	✔	✔	✖
Run/Stop job	✔	✔	✔	✖
Develop/Test job	✔	✔	✔	✖
Monitor alarm	✔	✔	✔	✖
View job	✔	✔	✔	✔
Create/Delete dependency	✔	✔	✔	✖
Edit dependency	✔	✔	✔	✖
View dependency	✔	✔	✔	✔
Create/Delete metadatabase	✔	✔	✔	✖
Create/Delete metadata table	✔	✔	✔	✖
View metadata	✔	✔	✔	✔
Operate directory	✔	✔	✔	✖

Granting predefined role permissions

Before granting space role permissions, you must have granted the target sub-account the access to Stream Compute Service and associated it with the required CAM policy. For details, see Granting Basic Permissions.

1. Add a super admin.

Log in to the console with the root account or a super admin account, select **Role permissions** on the left sidebar, and click **Edit** on the page to add one or more sub-accounts as super admin. A super admin has the highest level of access other than operating the root account and can be shared between different regions.

Note

A super admin account can assist the root account in cases where it is inconvenient to use the root account. You can set super admins as needed.

If you log in with an account other than the root account or a super admin account, the **Edit** button will not appear.

2. Create a space with the root account or a super admin account.

Log in the console with the root account or a super admin account, select **Workspaces** on the left sidebar, and click **Create workspace** on the page.

Note

You can create up to 30 workspaces in a region with the same APPID.

3. Associate a space with compute resources.

Log in the console with the root account or a super admin account, select **Workspaces** on the left sidebar, and click **Associate now** next to the compute resources field of the workspace created to go to the **Compute resources** module.

Select the cluster to be associated with the space. Till now, the compute resources and the space are associated with each other, and the compute resources will be available when you create a job in the space. To disassociate the space from compute resources, go to the **Compute resources** module, and click **Disassociate space**.

Note

Space and cluster association limits: A cluster can be used by up to 10 spaces, but there is no limit on the number of clusters a space can use.

4. Add a sub-account and grant a role in a space.

Log in the console with the root account or a super admin account, select **Workspaces** on the left sidebar, go to the space created, select **Members**, and click **Add member**.

Adding custom role permissions

1. On the **Role permissions** page, click **Custom role**.
2. Enter the required information and click **Save**.
3. Grant the permissions based on rules.